WWU
MÜNSTER

› Infrastructures and Practices for Reproducible Research in Geography, Geosciences, and GIScience

Daniel Nüst

2021

Geoinformatik

Infrastructures and Practices for Reproducible Research in Geography, Geosciences, and GIScience

Inaugural-Dissertation zur Erlangung des Doktorgrades der Naturwissenschaften im Fachbereich Geowissenschaften der Mathematisch-Naturwissenschaftlichen Fakultät der Westfälischen Wilhelms-Universität Münster

vorgelegt von

Daniel Nüst
aus Gütersloh

2021

https://orcid.org/0000-0002-0024-5046

Dekan:	Prof. Dr. Dr. h.c. Norbert Hölzel
Erstgutachter:	Prof. Dr. Edzer Pebesma
Zweitgutachter/-in:	Prof. Dr. Ben Marwick
Tag der mündlichen Prüfung:	____________________
Tag der Promotion:	____________________

ifgi
Institute for Geoinformatics
University of Münster

WWU
MÜNSTER

Citation for this document:

Nüst, Daniel. 2021. *Infrastructures and Practices for Reproducible Research in Geography, Geosciences, and GIScience.* Doctoral dissertation, University of Münster, Germany. https://doi.org/10.5281/zenodo.4768096

Acknowledgement

...

Abstract

Reproducibility of computational research poses enormous challenges to all branches of science. In this dissertation, technologies and practices are developed to increase reproducibility and to connect it better with the process of scholarly communication with a particular focus on geography, geosciences, and GIScience. Based on containerisation, this body of work creates a platform that connects existing academic infrastructures with a newly established executable research compendium (ERC). The ERC is shown to improve transparency, understandability, reproducibility, and reusability of research outcomes, e.g., for peer review, by capturing all parts of a computational workflow. The core part of the ERC platform is software that can automatically capture the computing environment, requiring authors only to create computational notebooks, which are virtual documents that combine text and analysis code. The work further investigates how containerisation is and can be applied independent of ERCs, including ~~with~~ a complex remote sensing workflow, for data science in general, and in use cases within the R language community. Based on these technical foundations, the work concludes that functioning practical solutions exists for making reproducibility possible through infrastructure and making it easy through user experience. Several downstream applications built on top of ERCs provide novel ways to discover and inspect the next generation of publications.

To understand why reproducible research has not been widely adopted and to contribute to the propagation of reproducible research practices, the dissertation continues to investigate the state of reproducibility in GIScience and develops and demonstrates workflows that can better integrate the execution of computational analyses into peer review procedures.

We make recommendations for how to (re)introduce reproducible research into peer reviewing and how to make practices to achieve the highest possible reproducibility normative, rewarding, and, ultimately, required in science. These recommendations are rest upon over 100 GIScience papers which were assessed as irreproducible, the experiences from over 30 successful reproductions of workflows across diverse scientific fields, and the lessons learned from implementing the ERC.

Besides continuing the development of the contributed concepts and infrastructure, the dissertation points out broader topics of future work, such as surveying practices for code execution during peer review of manuscripts, or reproduction and replication studies of the fundamental works in the considered scientific disciplines. The technical and social barriers to higher reproducibility are strongly intertwined with other transformations in academia, and, therefore, improving reproducibility meets similar challenges around culture change and sustainability. However, reproducible research is achievable already today. The transferability of cross-disciplinary lessons facilitates the establishment of reproducible research practices and, more than other transformations, the one toward greater reproducibility can draw from ~~from~~ accessible and convincing arguments both for individual researchers as well as their communities.

Zusammenfassung

Die Reproduzierbarkeit von rechnergestützter Forschung stellt alle Wissenschaftszweige vor enorme Herausforderungen. In dieser Dissertation werden Technologien und Praktiken entwickelt, um die Reproduzierbarkeit zu erhöhen und sie besser mit dem Prozess der wissenschaftlichen Kommunikation zu verbinden, mit besonderem Fokus auf Geographie, Geowissenschaften und GIScience. Basierend auf Containerisierung wird in dieser Arbeit eine Plattform geschaffen, die bestehende akademische Infrastrukturen mit einem neuartigen ausführbarem Forschungskompendium (Executable Research Compendium; ERC) verbindet. Es wird gezeigt, dass das ERC die Transparenz, Verständlichkeit, Reproduzierbarkeit und Wiederverwendbarkeit von Forschungsergebnissen, zum Beispiel für Peer-Reviews, verbessert, indem es alle Teile eines computergestützten Arbeitsablaufs erfasst. Das Kernstück der ERC-Plattform ist eine Software, welche die Rechenumgebung automatisch erfassen kann, so dass die Autoren nur noch sogenannte computational notebooks, virtuelle Notizbücher die Text und Analysecode verbinden, erstellen müssen. Die Arbeit untersucht weiter, wie Containerisierung unabhängig von ERCs angewendet wird und werden kann, unter anderem bei einer komplexen Analyse aus der Fernerkundung, für Datenwissenschaften im Allgemeinen sowie innerhalb der Anwenderschaft der Programmiersprache R. Basierend auf diesen technischen Grundlagen kommt die Arbeit zu dem Schluss, dass es funktionierende praktische Lösungen gibt, die Reproduzierbarkeit durch geeignete Infrastruktur möglich machen und die Benutzung deutlich vereinfachen. Mehrere nachgelagerte Anwendungen, die auf ERCs aufbauen, bieten neuartige Möglichkeiten, die nächste Generation von Publikationen besser suchen und inspizieren zu können.

Um zu verstehen, warum reproduzierbare Forschung nicht weit verbreitet ist, und um zur Verbreitung reproduzierbarer Forschungspraktiken beizutragen, untersucht die Dissertation weiterhin den Stand der Reproduzierbarkeit in der wissenschaftlichen Disziplin GIScience. Sie entwickelt und demonstriert Arbeitsabläufe, mit welchen die Durchführung von rechnerischen Analysen besser in Peer-Review-Verfahren integriert werden können.

Es werden Empfehlungen gegeben, wie reproduzierbare Forschung in Peer-Review-Verfahren (wieder) eingeführt werden kann und wie Praktiken um die höchstmögliche Reproduzierbarkeit zu erreichen in der Wissenschaft normativ, lohnend und letztlich verpflichtend werden können. Diese Empfehlungen stützen sich auf über 100 als irreproduzierbar befundenen Artikeln aus der GIScience, auf die Erfahrungen aus über 30 erfolgreichen Reproduktionen von computerbasierten Arbeitsabläufen in verschiedenen Wissenschaftsbereichen und auf die Erkenntnisse von der Implementierung des ERC.

Neben der Weiterentwicklung der eingebrachten Konzepte und der Infrastruktur weist die Dissertation auf weitergehende Themen zukünftiger Arbeit hin, wie zum Beispiel die Untersuchung von Prozessen für Code-Ausführung als Teil von Begutachtungen von Manuskripten, oder Reproduktions- und Replikationsstudien für grundlegende Arbeiten in den betrachteten Wissenschaftsdisziplinen. Die technischen und sozialen Barrieren für höhere Reproduzierbarkeit sind stark mit anderen Transformationsprozessen in der Wissenschaft verwoben und daher trifft die Verbesserung der Reproduzierbarkeit auf ähnliche Herausforderungen rund um Kulturwandel und Nachhaltigkeit. Reproduzierbare Forschung ist jedoch schon heute machbar. Die Übertragbarkeit von disziplinübergreifenden Erkenntnissen begünstigt die Etablierung reproduzierbarer Forschungspraktiken, und mehr als andere Transformationen kann jene zu mehr Reproduzierbarkeit aus zugänglichen und überzeugenden Argumenten sowohl für einzelne Forscher als auch für ihre Gemeinschaften schöpfen.

Contents

1 Introduction

1.1 Scope

This dissertation investigates ways to realise open reproducible research[1] in computational geospatial data science[2]. The work's core idea is to use methods from mainstream IT to capture and control computing environments in order to make the sharing, evaluating, and extending of computational workflows become a streamlined part of scientific communication and peer review processes. As these technologies are complex, one main goal is to open up the practices and methods for reproducible research by making them accessible, usable, and understandable for the broader communities of researchers in geography and geosciences. This dissertation was created as part of the project *Opening Reproducible Research* (o2r, Pebesma et al., 2020). The o2r project approaches the challenges of computational reproducibility from multiple angles, as, ultimately, the barriers to achieving higher transparency, reproducibility, and reusability are neither only technical nor simply a question of better community practices. Quite the contrary, all stakeholders in the scientific community who participate in scholarly communication, be they authors, readers, reviewers, journal editors, publishers, or universities, need to be involved to achieve one of the hardest challenges ahead for science: cultural change that values, rewards, and, eventually, requires computational reproducibility. That is why o2r as a research project as well as this dissertation work spans fields of engineering, science, and metascience. More than expected at the outset of this PhD project, we ended up having to consider reducing barriers not just related to individual researchers, motivations, and technological abilities, but also their social and cultural environments. Therefore, the research summarised here also ranges from engineering, i.e., providing the technical building blocks for more reproducibility, to metascience, i.e., understanding the current state of practices, barriers, and potential around reproducibility. Eventually, the work even approached science policy and included advocating for and actively engaging in opening reproducible research in the GIScience community and beyond. This dissertation is one of two dissertation projects conducted in the o2r project which greatly influenced and supported each other (cf. Konkol, 2019).

Brian Nosek describes the Center for Open Science's (COS) comprehensive strategy for culture and behaviour change using a pyramid with five highly interdependent layers (B. Nosek, 2019), shown in Figure 1. B. A. Nosek et al. (2021) describe the strategy in detail. This culture change pyramid categorises the publications of this cumulative dissertation and structures the Introduction and Synopsis sections. A similarly helpful concept was introduced by Lawrence Lessig's pathetic dot theory[3], where he describes four forces that regulate our lives as individuals. These forces can also help to describe the professional live of scientists, with *law* providing the policies and incentives as well as possibly threatening sanctions, with *social norms* and *markets* controlled by and providing value for the community, and *architecture* as the made or found constraining facts and infrastructure. If applied to scholarly communication and scientific progress, the important point is that all forces can be shaped by scientists, just as the layers of the culture change pyramid.

[1] Reproducible research means that an independent party can execute the original code on the same dataset and come to the same results. It is distinct from replicable, robust, and generalisable research, which provide different combinations of the same or different data or code; see The Turing Way's Table of Definitions.

[2] This new term is used with the intention to capture broadly all researchers who use computers *"to measure and describe features a features and phenomena on the Earth's surface"* as Brachman (2020) described 'geography,' and should be seen as interchangeable with lists like "geography, geosciences, and GIScience" used later in this work.

[3] https://en.wikipedia.org/wiki/Pathetic_dot_theory

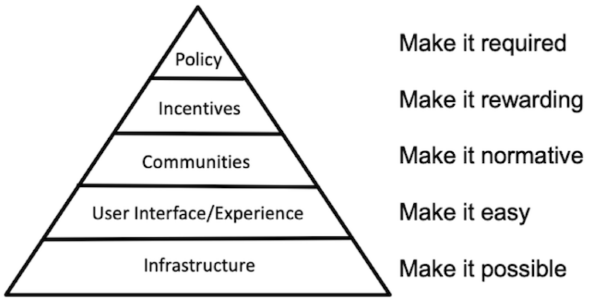


Figure 1: **Culture change pyramid**. Image by Brian Nosek (licensed under CC BY-ND 4.0), reproduced from the blog post Strategy for Culture Change.

1.2 Infrastructure & user experience

The bottom two layers of the culture change pyramid are *Infrastructure* and *User Interface/Experience*. At the beginning of the o2r project and this dissertation, the **executable research compendium** (ERC) was designed by the o2r team. The ERC concept and the *o2r reproducibility service* to create and examine ERCs provide the infrastructure and ensure a good user experience for packaging computational research. The ERC has four core parts: *data*, e.g., input data, pre-processed data, result data, *software*, e.g., scripts or special libraries, ~~but always~~ a containerised runtime environment, *documentation*, e.g., developer docs, user docs, but also the actual article, and *UI bindings* (Konkol, Kray, & Suleiman, 2019) as entry points for interaction. The ERC, thereby, aims to make reproducible research not only possible but even easy by building on the accessible literate programming paradigm (Wikipedia contributors, 2021) for the main document to capture text and workflow code. Furthermore, the ERC enables new kinds of workflow steps for in-depth examination, including manipulation and substitution of parts and reuse of results (Konkol & Kray, 2019). The only requirement for researchers is to share their work in a computational notebook.

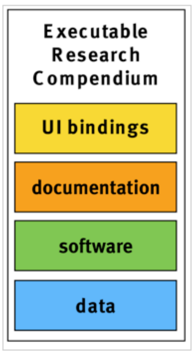


Figure 2: The parts of the **executable research compendium**: data, software, documentation, and UI bindings. Image from Nüst et al., 2017.

Nüst et al., 2017. *Opening the Publication Process with Executable Research Compendia*

The ERC concept was described by the o2r team in Chapter 3 (Nüst et al., 2017). Core contributions of this work include designing the component architecture and software components to integrate ERCs into scientific publishing systems and research infrastructures (Nüst, 2017a; Nüst & Schutzeichel, 2017), and implementing a prototype (Nüst, Kray, et al., 2018; Nüst, 2021c). The architecture design and implementation were in part based on experiences of making our own workflows reproducible and from collaborations on the reproducibility of complex geospatial data science analyses. The first full workflow made reproducible was a complex analysis in the area of remote sensing (Knoth & Nüst, 2017) . Chapter 4 presents the collaboration with C. Knoth, during which we packaged a geographic and object-based image analysis (GEOBIA) workflow based on a collection of Open Source tools in a container. The packaging of further smaller workflows from different domains was presented through several posters at the EGU General Assembly. These works explore use cases and tools on geoscientific data analysis in general (Nüst, Boettiger, & Eddelbuettel, 2018), on open environmental data analysis in citizen science (Nüst & Bartoschek, 2018), and on scalable Earth observation analytics (Appel et al., 2017). The citizen science use case in particular demonstrates the possibilities that computational notebooks provide for collaboration and transparency, not only with respect to Open Data and Open Source code, but even down to the Open Hardware of the measurement devices. What all these reproducible workflows share is the use of *containerisation* (Wikipedia contributors, 2020). In fact, the idea to use containerisation is at the core of most attempts (cf. Konkol et al., 2020) to connect reproducibility with the goals of the bottom two layers of the above pyramid (Figure 1, namely *"Make it possible"* and *"Make it easy"*.

Knoth & Nüst, 2017. *Reproducibility and Practical Adoption of GEOBIA with Open-Source Software in Docker Containers*

The main part of this dissertation offers a set of publications that help to realise the encapsulation of computational environments with containerisation. To realise the concept of ERCs, we designed and implemented a **technical specification and web service for the ERC**. These specifications enable creation and interaction with ERCs, and they are complemented with a system architecture that allows the ERC to be integrated into the existing infrastructure for scholarly communication, e.g., journal publishing platforms, software registries, and data repositories. The ERC specification, the web service's API specification, the architecture, and a reference implementation are described in Chapter 5 (Nüst, 2021c) and publicly available via the o2r project website[4]. o2r's specifications and reference implementation reflect only a few of several tools and services which were developed ~~independently and, for the most part, simultaneously~~ during the time of this dissertation. We reviewed and compared these options in Konkol et al. (2020) and list more in Nüst (2021c). Like most of the reviewed platforms, the o2r system builds on the literate programming paradigm (i.e., computational notebooks) and containerisation. One core feature of the o2r reproducibility service is the automated capturing of the computing environment. This feature relies on the containerit R package (Nüst & Hinz, 2017, 2019), which was developed as a standalone tool for independent use (see Chapter 7). containerit's unique feature is that is uses the state of the session after execution of the complete workflow to capture all the required libraries and tools. Unlike other approaches, containerit will not break if it faces an unconventional declaration of dependencies and can bring forth indirect or hidden dependencies. Other tools in the background of the ERC web service with potential for reuse are o2r-meta and geoextent for automatic extraction of (geospatial) metadata and metadata conversion.

Nüst, 2021. *A Web service for executable research compendia enables reproducible publications and transparent reviews in geospatial sciences*

Konkol, Nüst & Goulier, 2020. *Publishing computational research - a review of infrastructures for reproducible and transparent scholarly communication*

Nüst & Hinz, 2019. *containerit: Generating Dockerfiles for reproducible research with R*

Although the openness of the o2r platform avoids a problematic lock-in, it is also sensible to look into the ways researchers can use **containerisation independent of specific infrastructures** to make their research more transparent and reproducible. Therefore, two publications investigate the state of the art of the core technology for sharing reproducible workflows, i.e., containerisation, and the primary programming language, i.e., R, used in the o2r project. A practical article (Nüst, Sochat, et al., 2020) published in the *Ten Simple Rules* collection[5] describes the crafting of bespoke computing environments with Docker contain-

Nüst et al., 2020. *Ten simple rules for writing Dockerfiles for reproducible data science*

[4]ERC specification: https://o2r.info/erc-spec/; System architecture: https://o2r.info/architecture/; web service API: https://o2r.info/api/; demo server: https://o2r.uni-muenster.de/; all software and specifications are archived on Zenodo at https://doi.org/10.5281/zenodo.2203843.

[5]https://collections.plos.org/ten-simple-rules

ers (see Chapter 8). Explorations on the very broad area of "anything containers and R" led to a large collaborative article describing the *Rockerverse* in Chapter 9 (Nüst, Eddelbuettel, et al., 2020).

Nüst et al., 2020. *The Rockerverse: packages and applications for containerisation with R*

The final paper of this part on the **practical foundations for working reproducibly** is an invited paper published in the *Annals of the American Association of Geographers* as part of the *Forum on Reproducibility and Replicability in Geography*[6]. The Forum was compiled as a follow-up to the workshop on Reproducibility and Replicability in Geospatial Research at the Spatial Analysis Research Center at Arizona State University in February 2019. In the article (see Chapter 10) we present the state-of-the-art approaches for reaching a high degree of computational reproducibility and discusses the challenges for practical reproducibility specific to geography and geosciences. The practical solutions are illustrated with an overview of existing reproduction efforts and with new reproductions. Other articles from the Forum (Goodchild et al., 2020) provide diverse perspectives on the topic of reproducibility and replicability, especially on more theoretical deliberations in geography, ethical replicability in critical applications, and replication when working with data streams.

Nüst & Pebesma, 2020. *Practical reproducibility in geography and geosciences*

For the two lower layers of the culture change pyramid (Figure 1), namely *Infrastructure* and *User experience*, the following research questions were formulated and are discussed in the Synopsis (Chapter 17) based on the papers relevant to this theme (Chapters 3 to 10).

IUE1 How can packaging of computational analyses serve the needs of authors, publishers, readers, and preservationists?

IUE2 To what extent can the process of capturing the runtime, software, data, and metadata of reproducible research packages be automated in geoscientific analyses?

IUE3 How can the ERC fit into the existing infrastructure of services and platforms for research and publishing in geography and geosciences?

1.3 Communities, incentives & policy

The previous section presents an investigation of and solutions to technical challenges around reproducible research in geography, geosciences, and GIScience based on tools and infrastructure. To put these solutions into practice, not only do researchers have to adopt them, but other stakeholders, including educational institutions, scholarly societies, and publishers, must also support the transition. While educational institutions are important to provide the needed training, scholarly societies and publishers are needed to shift practices at journals and conferences, and all of these parties are crucial for creating the environment to drive the cultural change process. Thus, we must answer the question of how to increase provision and adoption of presented technical solutions to make better practice through higher reproducibility *normative*, *rewarding*, and eventually *required*. The text of the previous sentence that is set in italics represents goals of the top three layers of the culture change pyramid. These layers are titled *Communities*, *Incentives*, and *Policy*. They are very much intertwined and not strictly ordered, e.g., to shift policy, one must think about incentives, and to provide incentives, one must consider the policies that can support the incentives. Therefore, the works presented in this part of the dissertation often span all three considered layers of the culture change pyramid and are collaborative efforts that even go beyond spatial data sciences.

[6]The full list of articles is available at https://sgsup.asu.edu/forum-articles-reproducibility-and-replicability-published-annals-american-association-geographers.

First, one must understand the **state of reproducibility** to build a convincing case for cultural change. Maybe it is not that bad? Two papers of this dissertation (Chapters 11 and 12) evaluate the reproducibility of GIScience research and complement related research from the o2r project on reproducibility in geography and geoscience (Konkol, Kray, & Pfeiffer, 2019; Nüst & Pebesma, 2020). In the first (Chapter 11), a sample of 32 papers from the annual AGILE Conference series was evaluated (Nüst, Granell, et al., 2018) using a novel set of criteria to assess the level of reproducibility across core categories of data-based science (input data, preprocessing, methods/analysis, computational environment, results). Next, in Chapter 12), the same methodology was transferred to the biannual GIScience conference series, where 75 papers were assessed (Ostermann et al., 2020) . These articles also clearly communicate the options and possible future practices to the respective audiences in the community, including concrete recommendations for individuals, groups, and institutions for how to contribute to the improvement of reproducibility in GIScience.

Nüst et al., 2018. *Reproducible research and GIScience: An evaluation using AGILE conference papers*

Ostermann et al., 2020. *Reproducible research and GIScience: An evaluation using AGILE conference papers*

With these assessments, it is now possible for the GIScience *Community*, the culture change pyramid's middle layer, to take action. The **Reproducible AGILE** initiative was co-founded during this dissertation to initiate a community discourse about reproducibility, discuss incentives for and recognition of reproducibility, and eventually shift policy, practices, and norms. The initiative members organised a series of workshops, the first of which resulted in the first of two assessment papers mentioned above (Nüst, Granell, et al., 2018). The Reproducible AGILE team received financial support as an official *AGILE Initiative*[7] to create the AGILE Reproducible Paper Guidelines (Nüst, Ostermann, et al., 2021), which were introduced in 2020[8] and made mandatory in 2021. Daniel Nüst served as chair of the reproducibility committee for both events. The members of Reproducible AGILE continue to organise the reproducibility review and to share their experiences and explore novel ways to improve reproducible research adoption and practices (Granell et al., 2018; Nüst, Ostermann, et al., 2020; Nüst, 2021d), e.g., understanding and improving reproducibility of graduate theses (Granell et al., 2020).

In addition to contributions to the activities of Reproducible AGILE, the work of this dissertation also co-founded an even broader Open Science initiative. Together with neuroscientist Stephen J. Eglen, University of Cambridge, the **CODECHECK** initiative (Eglen & Nüst, 2019; Nüst & Eglen, 2021) was started to bring code execution (back) into peer review. Chapter 13 describes CODECHECK, which is a "non-tech" approach to computational reproducibility. Just like the Reproducible AGILE initiative, CODECHECK aims to introduce execution of computational workflows into the peer review process and relies on the judgement of a peer codechecker, similar to how a peer reviewer evaluates the scientific merit of an article. CODECHECK relies on a set of principles, seeks collaborations with existing journals, and is realised with an open infrastructure[9].

Nüst & Eglen, 2021. *CODECHECK: An Open Science initiative for the independent execution of computations underlying research articles during peer review to improve reproducibility*

Finally, besides recognition and credit, there can also be more practical benefits to sharing computational workflows in the form of ERCs. These benefits can also function as *incentives.* Several beneficial **downstream applications** of ERCs were explored as part of this dissertation. Chapters 14 to 16 describe using geospatial metadata for discovery of research papers (Niers & Nüst, 2020), investigate distributing automatically generated badges without stakeholder support (Nüst et al., 2019), and suggest how ERCs may change the way we "read" articles, or rather "examine" workflows (Nüst, Boettiger, & Marwick, 2018).

Niers & Nüst, 2020. *Geospatial Metadata for Discovery in Scholarly Publishing*; Nüst et al., 2019. *Guerrilla Badges for Geoscience Research Packages*; Nüst, Boettiger & Marwick, 2018. *How to read a research compendium*

[7] See the report of the AGILE Initiative https://osf.io/hupxr/; it extends the recommendations from the paper towards more modern open practices beyond reproducibility.

[8] See report of the reproducibility committee: https://osf.io/7rjpe/.

[9] Details about partnering journals, publishers, and stakeholders in scientific publishing are available on the website at https://codecheck.org.uk. The CODECHECK certificates, the register of checks, and further documents and software are deposited on Zenodo: https://zenodo.org/communities/codecheck/.

The following research questions were posed with connection to the top three layers of the culture change pyramid (Figure 1), namely *Communities*, *Incentives*, and *Policy*, and are discussed in the Synopsis (Chapter 17) based on the papers relevant to this theme (Chapters 10 to 16).

CIP1 What are domain-specific challenges and solutions for the geosciences domain in the context of reproducible publications?

CIP2 What new services and features can be built upon reproducible workflows, e.g., when packaged as an ERC?

2 List of publications

The papers listed below form the cumulative dissertation and are chapters in this document. They are grouped by the layers of the cultural change pyramid described in the Introduction and discussed in the Synopsis. Each chapter starts with a cover page mentioning authors and the contributions, original publication venue with *Source Normalised Impact per Publication*[10] (SNIP), license, link to ERC (applicable if figures are based on data), link to paper repository (where available), publication date (month/year), and publication status. A comprehensive list of publications and talks from the project Opening Reproducible Research and by Daniel Nüst can be found at https://o2r.info/publications/ and https://orcid.org/0000-0002-0024-5046, respectively. Note that in the digital version of this document, hyperlinks in the included PDFs do not work—please resort to the online version or publisher PDFs for easy access to linked resources and references. Figure 3 illustrates all publications, introduction, and synopsis in a word stem cloud.

Infrastructure & user experience

Knoth, C., & Nüst, D. (2017). Reproducibility and Practical Adoption of GEOBIA with Open-Source Software in Docker Containers. *Remote Sensing*, *9*(3), 290. https://doi.org/10.3390/rs9030290

Konkol, M., Nüst, D., & Goulier, L. (2020). Publishing computational research - a review of infrastructures for reproducible and transparent scholarly communication. *Research Integrity and Peer Review*, *5*(1), 10. https://doi.org/10.1186/s41073-020-00095-y

Nüst, D. (2021). *A web service for executable research compendia enables reproducible publications and transparent reviews in geospatial sciences.* https://zivgitlab.uni-muenster.de/d_nues01/architecture-paper/

Nüst, D., Eddelbuettel, D., Bennett, D., Cannoodt, R., Clark, D., Daróczi, G., Edmondson, M., Fay, C., Hughes, E., Kjeldgaard, L., Lopp, S., Marwick, B., Nolis, H., Nolis, J., Ooi, H., Ram, K., Ross, N., Shepherd, L., Sólymos, P., Swetnam, T. L., Turaga, N., Petegem, C. V., Williams, J., Willis, C., & Xiao, N. (2020). The Rockerverse: Packages and Applications for Containerisation with R. *The R Journal*, *12*(1). https://doi.org/10.32614/RJ-2020-007

Nüst, D., & Hinz, M. (2019). Containerit: Generating Dockerfiles for reproducible research with R. *Journal of Open Source Software*, *4*(40), 1603. https://doi.org/10.21105/joss.01603

Nüst, D., Konkol, M., Pebesma, E., Kray, C., Schutzeichel, M., Przibytzin, H., & Lorenz, J. (2017). Opening the Publication Process with Executable Research Compendia. *D-Lib Magazine*, *23*(1/2). https://doi.org/10.1045/january2017-nuest

Nüst, D., & Pebesma, E. (2020). Practical reproducibility in geography and geosciences. *Annals of the American Association of Geographers*, *0*(0), 1–11. https://doi.org/10.1080/24694452.2020.1806028

Nüst, D., Sochat, V., Marwick, B., Eglen, S., Head, T., & Hirst, T. (2020). *Ten Simple Rules for Writing Dockerfiles for Reproducible Data Science.* https://doi.org/10.31219/osf.io/fsd7t

[10]See https://www.journalindicators.com/methodology; the SNIP documents that the required number of publications for the dissertation has been published in ranked journals. It is used instead of the more common yet largely considered broken metric *Journal Impact Factor* (JIF).

Communities, incentives & policy

Niers, T., & Nüst, D. (2020). Geospatial Metadata for Discovery in Scholarly Publishing. *Septentrio Conference Series*, *4*. https://doi.org/10.7557/5.5590

Nüst, D., Boettiger, C., & Marwick, B. (2018). How to Read a Research Compendium. *arXiv:1806.09525 [Cs]*. http://arxiv.org/abs/1806.09525

Nüst, D., & Eglen, S. J. (2021). CODECHECK: An Open Science initiative for the independent execution of computations underlying research articles during peer review to improve reproducibility. *F1000Research*, *10*, 253. https://doi.org/10.12688/f1000research.51738.1

Nüst, D., Granell, C., Hofer, B., Konkol, M., Ostermann, F. O., Sileryte, R., & Cerutti, V. (2018). Reproducible research and GIScience: An evaluation using AGILE conference papers. *PeerJ*, *6*, e5072. https://doi.org/10.7717/peerj.5072

Nüst, D., Lohoff, L., Einfeldt, L., Gavish, N., Götza, M., Jaswal, S. T., Khalid, S., Meierkort, L., Mohr, M., Rendel, C., & Eek, A. van. (2019). *Guerrilla Badges for Reproducible Geospatial Data Science (AGILE 2019 Short Paper)*. https://doi.org/10.31223/osf.io/xtsqh

Ostermann, F. O., Nüst, D., Granell, C., Hofer, B., & Konkol, M. (2020). *Reproducible Research and GIScience: An evaluation using GIScience conference papers*. https://doi.org/10.31223/X5ZK5V

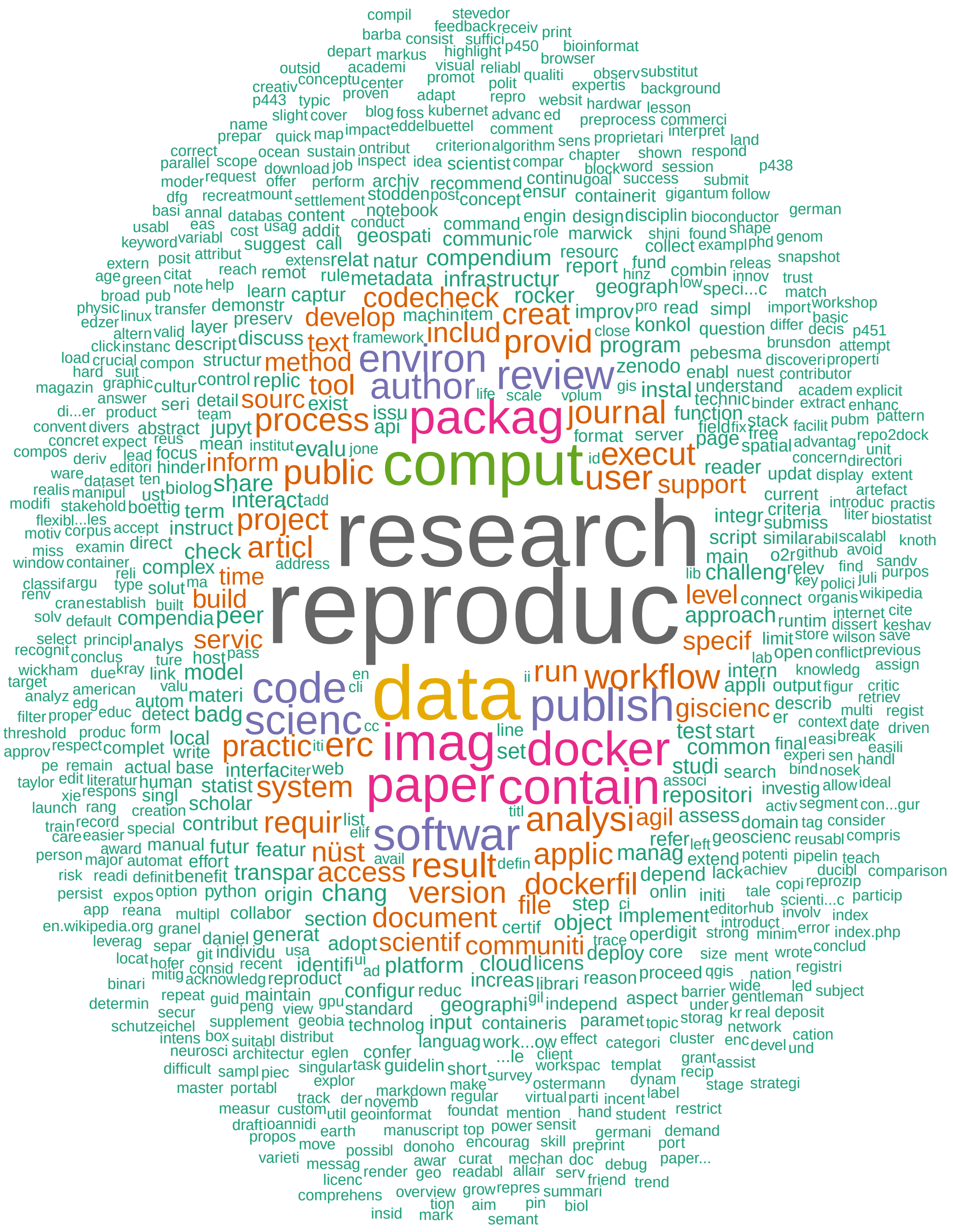


Figure 3: Word stem cloud. Contains all terms used in this document, including references, excluding select stop words (based on `tidytext::stop_words` and selected abbreviations, publisher names, code, markup, domains, etc.); figure is based on 92212 word stems, minimum occurrence is manually set to 21 times; the top word stem ”reproduc” occurs 1743 times.

3 Opening the publication process with executable research compendia

Authors & contribution Nüst D (25%), Konkol M (25%), Pebesma E, Kray C, Schutzeichel M, Przibytzin, H, Lorenz, J

Venue *D-Lib Magazine* (article peer-reviewed as part of the workshop RepScience 2016) 10.1045/january2017-nuest

Date 01/2017

Licence D-Lib Magazine Access Terms and Conditions (attribute, unabridged, for research but not commercial purposes, “AS IS” basis)

D-Lib Magazine

January/February 2017
Volume 23, Number 1/2
Table of Contents

Opening the Publication Process with Executable Research Compendia

Daniel Nüst*
Institute for Geoinformatics, Münster
daniel.nuest [at] uni-muenster.de

Markus Konkol*
Institute for Geoinformatics, Münster
m.konkol [at] uni-muenster.de

Marc Schutzeichel
University and State Library, Münster
m.schutzeichel [at] uni-muenster.de

Edzer Pebesma
Institute for Geoinformatics, Münster
edzer.pebesma [at] uni-muenster.de

Christian Kray
Institute for Geoinformatics, Münster
c.kray [at] uni-muenster.de

Holger Przibytzin
University and State Library, Münster
holger.przibytzin [at] uni-muenster.de

Jörg Lorenz
University and State Library, Münster
joerg.lorenz [at] uni-muenster.de

**Shared co-first authorship*

https://doi.org/10.1045/january2017-nuest

Abstract

A strong movement towards openness has seized science. Open data and methods, open source software, Open Access, open reviews, and open research platforms provide the legal and technical solutions to new forms of research and publishing. However, publishing reproducible research is still not common practice. Reasons include a lack of incentives and a missing standardized infrastructure for providing research material such as data sets and source code together with a scientific paper. Therefore we first study fundamentals and existing approaches. On that basis, our key contributions are the identification of core requirements of authors, readers, publishers, curators, as well as preservationists and the subsequent description of an *executable research compendium* (ERC). It is the main component of a publication process providing a new way to publish and access computational research. ERCs provide a new standardisable packaging mechanism which combines data, software, text, and a user interface description. We discuss the potential of ERCs and their challenges in the context of user requirements and the established publication processes. We conclude that ERCs provide a novel potential to find, explore, reuse, and archive computer-based research.

Keywords: Executable Research Compendium, ERC, Open Access, Containerization, Research Data, Computational Research

1 Introduction

Open Access is not only a form of publishing that ensures research papers become available to the large public free of charge, it is also connected to a trend towards Open Science, which makes research more transparent (Nosek *et al.*, 2015, and see also Charles W. Bailey, What is Open Access? and Open Access to Scientific Information, History of the Open Access Movement). To fully realise Open [computational] Science we expect everyone can reproduce findings because access is granted to papers, research data, methodology, and the computational environment. In parallel, the scientific paper "is evolving into a multi-part distributed object that can include an article, data, software, and more" (Hanson *et al.*, 2015). Access to these components is rarely given, making it challenging to archive and reproduce methods and results. Reasons include privacy concerns and copyright issues. Another key obstacle is the lack of standardized means for sharing all the parts of an 'evolved' scientific paper in an easy way (Hanson *et al.*, 2015).

The main contribution of this work is the definition of a compendium-based publication process. It facilitates Open Science and enables new ways to conduct research. Its core component is the executable research compendium (ERC), which opens and integrates the scientific process across all activities and stakeholders. The compendium includes, besides the actual paper, source code, the computational environment, the data set, and a definition of a user interface. It has internal connections which facilitate new ways of interacting and reuse. Such a compendium is a self-contained entity and can be executed almost entirely on its own, requiring only a generic virtualization environment. ERCs are intended for the huge number of research projects which live on a researcher's desktop computer. Larger scale undertakings with high computational or storage requirements, e.g. distributed infrastructures, and using external third-party services ("black boxes") are out of scope of this work although being subject to vivid research (cf. Chen *et al.*, 2016, Goecks *et al.*, 2010).

The paper is structured as follows. The next section provides a review of related work and basic concepts. In the main part, we explain how the ERC builds upon stakeholder requirements and current technologies. We describe the process of publishing ERCs and conclude by highlighting the main contributions and future work.

2 Related Work

In this chapter we introduce the term "reproducibility" and Open Access as motivators of an open publication process. Our work builds on the concept of research compendia/objects, data platforms, and methods to capture software environments. We also summarize key challenges in open reproducible research.

2.1 Definition of Reproducibility

In current research, talking about computational reproducibility can lead to confusion as a universally accepted definition of the term "reproducibility" is not yet established. While some researchers treat "reproducibility" and "replicability" as interchangeable terms (Bollen *et al.*, 2015), others distinguish them carefully (Leek and Peng, 2015; Goodman *et al.*, 2016). "Replicability" is given if researchers receive the same results by using the same methodology but a different data set (Bollen *et al.*, 2015). Research findings are "reproducible" if researchers are able to compute the same results by using the same procedure and the same data set (Bollen *et al.*, 2015). This means a researcher requires the entire set of information which constitute the basis of the results reported in the paper including data sets, source code, and configuration details (Vandewalle *et al.*, 2009). A key difference between the two terms is the data set which needs to be different from the originally used data set in the case of "replicability", and to be the same in the case of "reproducibility". Consequently, it does not mean reproducible research findings are true. They can still be subject to flaws in the study design (Leek and Peng, 2015). However, reproducible outcomes are more reliable as they allow other researchers to understand the (computational) steps described in the paper. In ***open*** reproducible research, the components required to reproduce the results are publicly available. Other scientists are thus able to reuse parts for their own research.

2.2 Open Data & Open Access

Today, most researchers find it difficult to reproduce the analysis reported in papers published 5-10 years ago, or even recover the data. For instance, Vines *et al.* (2014) reported a half-life of four years for data recovery from the original authors. Nowadays there is a strong trend towards publishing scientific papers as Open Access, meaning anyone has free access to the published material, including free access to data and software (cf. The Open Definition: "Open data and content can be freely used, modified, and shared by anyone for any purpose."). Different models exist (cf. Harnad *et al.*, 2004) and publishers increasingly offer Open Access routes for research papers. For instance, PLoS-One's policy requires authors to publish data by default (Bloom *et al.*, 2014). The Directory of Open Access Journals (DOAJ) lists over 9000 journals. Open Access is a key topic in the European Union's research programme Horizon 2020. Dissemination and reuse of scientific findings shall be improved by making them accessible at no charge through suitable e-infrastructures (European Commission, 2015) such as the European Open Science Cloud.

2.3 Research Compendia & Research Objects

Gentleman and Temple Lang (2007) use the term Research Compendium (RC) to refer to the unit of scholarly communication which includes the research paper, the code, and the data with a "dynamic document" (cf. Knuth, 1984) at the core. They present the advantages and potential uses of (executable) research compendia, which allow to completely reproduce the computational aspects of a scientific paper. (Randomness of specific simulations or machine learning methods can be handled by setting a seed. Numerical differences between runs can be mitigated by appropriate rounding. Both these aspects require careful design of the actual analysis and are beyond the scope of ERC.) The practical implementation focuses on "single language compendia", which are essentially the respective languages' standard packaging mechanism (e.g. R packages). Stodden *et al.* (2015) developed a platform for sharing, executing and validating Research Compendia.

Research Objects (RO) are "semantically rich aggregations of resources that bring together the data, methods and people involved in (scientific) investigations" (Bechhofer *et al.*, 2013). They comprise metadata standards and bundling of any kind of resource across a range of scopes, such as scientific workflows (Belhajjame *et al.*, 2015), preservation (Research Object BagIt archive), or computational jobs. These can be included or remote (i.e. linked) resources. ROs are also Linked Data resources themselves and can be managed and preserved in a tailored platform (Palma *et al.*, 2013).

Both Nature and Nature Geosciences published about the need to publish reproducible code (Nature Editorial, 2014; Easterbrook, 2014). The majority of scientific journals allow adding supplementary material to publications, and hence creating and publishing Research Compendia or Research Objects has been possible for a long time. Yet current supplemental material frequently looks very much like a directory on the researcher's personal computer.

2.4 Research Data Platforms

We see a large number of technical solutions for research repositories, including RunMyCode, the Open Science Framework, Zenodo, and figshare (see Dave Wilkinson's "rubric" for platforms.) All these platforms provide identification (some only by email; others support ORCID or GitHub identities), uploading, and persistent identifiers for citation of deposited items. All platforms have a data size limitation, some have paid upgrade plans. Some sites argue users should trust their long-term availability, others just offer the service as-is, for example, RunMyCode is provided on an "as is" and "as available" basis. All these platforms are fine for viewing and downloading, but none of them actually performs the computations needed to carry out reproducible research. This is most likely due to recomputation requiring considerably more resources than storage, and thus causing substantial costs, and security concerns. Also, the platform would need to understand a plethora of different computational setups and would have to cater for execution of compendia within these.

2.5 Capture Workflows and Runtime Environments

Researchers enjoy to conduct their research in an environment composed of software of their own choice. Consequently, the number of unique research environments approximates the number of researchers, and hence expecting one standardised computational workflow would not be feasible. But the complete computational environment of the original researcher, the runtime environment, is an important aspect of reproducibility. Different approaches to capture the workflows and runtimes can be chosen.

Regarding workflows, a commonality of all computational sciences is carrying out a number of steps (process input file(s), process data into result, e.g. number or graph). These steps are ordered and must be documented to allow understanding, execution, and reproduction. The canonical example is the classic unix utility "make" (Mecklenburg, 2004), which executes commands according to the instructions in a Makefile. The concept has been adapted for specific scenarios, e.g. remake for data analysis in R, drake for data workflow management, or Taverna (Wolstencroft *et al.*, 2013) for web-based workflows. Santana-Perez *et al.* (2016) demonstrate a semantic modelling approach to conserve scientific workflow executions based on semantic vocabularies. Thain *et al.* (2015) discuss two broad approaches to capture scientific software executions: "preserving the mess, and encouraging cleanliness". They provide an extensive picture on technical and organizational challenges in the context of in-silico experiments and present prototypical solutions (e.g. Umbrella).

Alternatively, the environment can be confined to an interpreter of a particular language, such as R (R Core Team, 2014). The environment is reproduced if the correct versions of the interpreter and all extension packages/libraries/modules are used. These dependencies quickly become complex so a manual documentation is not feasible. The R extension packages checkpoint or packrat ease the process of reproducing project dependencies, except the interpreter itself. ReproZip (Chirigati *et al.*, 2013) or Parrot (Thain *et al.*, 2015) apply tracing techniques to capture the minimal set of objects and commands needed for an analysis and use it to build a virtual machine (VM) or container image for execution.

Containerization originates from packaging applications and their dependencies for deployment in cloud infrastructures (cf. Dua *et al.*, 2014). Containers have proven to be a suitable technology in reproducible research: Howe (2012) lists improvements virtualization and cloud computing provide for reproducibility, all of which apply directly to containerization. Boettiger (2015) demonstrates its usage for computational analysis in R and derives best practices. Marwick (2015) accompanies a research article by Clarkson *et al.* (2015) with a complete research compendium containing the analysis (also available on GitHub). Hung *et al.* (2016) use containers to package graphical user interface-based research environments with multiple tools from multiple languages across operating systems in the context of bioinformatics.

The dominant platform for container is Docker (Merkel, 2014). Its images are created using a recipe called Dockerfile — a Makefile with inheritance support for creating containers. Images can be build and executed on any machine running a Docker host, and are distributed in ready to run form via online hubs, for example Docker Hub or Quay.

2.6 Challenges in Open Reproducible Research

Solutions to technical and legal challenges with regard to implementing reproducible research exist (cf. Stodden *et al.*, 2014). Other challenges, such as the question of incentives and developing reward mechanisms, are closely connected to the intricacies of academic authorship and credit. The challenges and some approaches are summarized in the Wikipedia article on academic authorship, for example the individual h-index (George A. Lozano) or Nature's author contributions statements.

Among the reasons why researchers do not publish reproducibly, Borgman (2007) mentions (1) a lack of incentives in terms of citations or promotion, (2) the effort required to clean data and codes, (3) the creation of a competitive advantage over other fellows, and (4) intellectual property issues. Other reasons include privacy or confidentiality issues (Glandon, 2012). None of these issues can be solved by technical solutions alone. Instead they require a discourse within the scientific community, or even a mindshift to "build a 'culture of reproducibility'" based on selfish reasons to publish reproducibly (Markowetz, 2015).

Although moving the relevant parts of a researcher's hard drive to public and citable archives is a step forward in the direction of Open Access to research findings, we believe this alone will not convince researchers to adopt it as standard practice. Such archived workspaces could be incomprehensible (Easterbrook, 2014). The potential reward is considered smaller than the possible reputation damage or expected follow-up work, e.g. support questions.

Each scientific community must address such worries to make publication of workspaces an effective incentive for reproduction. Encouraging statements such as Nick Barnes' column "Publish your computer code: it is good enough" (Barnes, 2010) pave the way towards venturing reproducibility of software. However, they also highlight the necessity for communication of researchers' uncertainties regarding reproducibility as a deliberate act of exposure to critics and competitors.

The research data repositories mentioned before support reproducibility, but they cannot carry out the actual reproduction, nor do they suggest conventions for doing so in an automated fashion. Although they are designed to deposit code and data supplementing a scientific paper, none of them require documenting the runtime environment, under which the reproduction material is expected to reproduce the paper, in a systematic way. This puts more burden on the original author, as each is supposed to know how to describe this in a generic way. The same burden is placed on a reader as the reproducer.

3 The Executable Research Compendium

3.1 Stakeholders and Roles in the Publication Process

The ERC brings together perspectives of different user groups involved in scientific publications. To develop a common ground for them, we first consider some key requirements of users separately. We identified these requirements in discussions including experts from university, library, and publishing (members of the project team and external partners are listed here).

Authors must be given support to create an ERC without too much additional work, ideally starting from their digital workspace "as is". Since filling out forms for metadata is often perceived as a daunting and time-consuming task, an ERC should be restricted to a minimal set of required metadata. Ideally, the needed information can be (semi-)automatically deduced via existing identifiers and catalogues, such as DataCite (Brase, 2009) for data sets and ORCID (Haak *et al.*, 2012) for researchers, thus avoiding the need for entering them multiple times. If common practices for reproducible research are followed then creating an ERC should be as simple as manually editing a configuration file and running a single command.

Readers want to be able to rerun the analysis in an ERC. On a basic level, a compendium must provide a simple "start" button to run an analysis and a traffic-light like signal to learn about the result, cf. Elsevier's Earth and Planetary Innovation Challenge winning submission "One-Click-Reproduce" by Edzer Pebesma. A green light shows a successful, a red one an unsuccessful reproduction of the results. Here a green light means the computational steps are executable and the results are equal to those submitted by the original author. A red light means this is not the case. Either way the status light is no indication on the validity of the findings. At an advanced level, readers can examine detailed information on, for example, data and code underlying figures, or errors preventing successful execution. A typical reader's question might be about assumptions and concrete tools the authors did not mention. Moreover, it would be interesting for readers to see how the results change and if the conclusions still hold true after manipulating parameters.

Reviewers must be put in the position to scrutinize a piece of research. While potential malicious intent of the submitting authors is not considered here, review processes still involve a trust component as far as scientific and scholarly best practices are concerned. Reviewers often have to rely on the analysis in a research paper to be complete, correct, and consistent with the included visualizations. In order to verify the analysis, they need tools to recreate computation-based analyses with minimal effort. As a review and publication process is well established, the ERC shall align itself with it. It becomes the item under review and should help to increase the quality of the review. Since this also increases the quality of journals, ERCs ultimately serve the needs of *editors*, *authors*, *readers*, and *publishers*.

Libraries and publishers make research articles accessible for the scientific community and the public. However, access to the source material is not always granted. One of the reasons for this is neither libraries nor researchers are currently familiar with a publication process that allows for including procedures used to produce research outcomes (tables, figures). Any new tool for this purpose must leverage their expertise and pay attention to their existing workflows.

Curators and preservationists require the reader's basic level to assess a compendium's status. A compendium run must entail an execution of the contained computation and an automatic validation of the result to ensure the integrity of the digital asset. Since they need to integrate an ERC into a curation workflow, the required metadata must follow standards for digital preservation. As they cannot rely on the execution platform to be persistent, they need access to relevant metadata in "their" formats independently from the data and software within the compendium. Subsequently, the top-level package of the compendium should follow a preservation standard.

To fulfil the needs of these different user groups, we define conventions in the remainder of this chapter. They allow to create research compendia supporting automatic execution, and to create services for executing such research compendia.

3.2 Core Parts of ERC

The core parts of an ERC are data, software, documentation, and user interface (UI) bindings (see Figure 1).

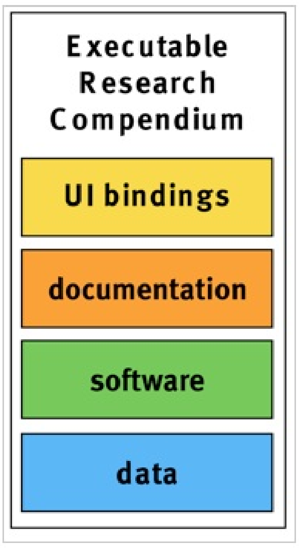


Figure 1: The core parts of an ERC are data, software, documentation, and UI bindings.

Data comprises all inputs for an analysis, ideally starting with raw measurements, for example, in form of text files, or databases.

Software comprises code created by a researcher and all underlying libraries or tools to reproduce the analysis in form of scripts/source code, a Dockerfile, and a Docker container.

Documentation comprises both instructions, such as a README file, and the actual scientific publication, e.g. in PDF format, any supplemental records, and metadata in standardized formats. The actual publication is the main output of the compendium and the core element for validation. An important metadata element are licenses for the different parts of a compendium.

UI bindings provide linkage between research components and user interface widgets. They can be used to attach UI widgets to static diagrams in order to make them interactive. Their representation can be stored as metadata within an ERC as part of the documentation. The resulting UI widgets open up the container and allow readers to drill deeper into results. UI bindings can unveil parameters which are required for a comprehensive understanding but are often buried in the code.

3.3 Creation and Reproducibility

Two approaches for creating an ERC are possible: post-hoc or on-the-fly.

Post-hoc creation is based on the regular workspace, which is a collection of files (data, code, documentation). Authors of a publication can submit a workspace to an ERC building service to generate a compendium. Such a service requires complex logic and user intervention, either by an expert on the service provider side or by the author, to detect how to start the analysis and how to validate its result. Following best practices from reproducible research, such as literate programming (Knuth, 1984) or having a default main file and execution command (comparable to a Makefile), can considerably reduce the required intervention. An ERC specification can support this with a "convention over configuration" approach (see Wikipedia, Convention over configuration, Nicholas Chen, "Convention over Configuration" and also Maven standard directory layout). Information that cannot be derived automatically must be elicited during the submission process, ideally with pre-filled forms.

Alternatively the creation and maintenance of a compendium happens on-the-fly while carrying out research. A user starts with a template compendium. Since the compendium evolves with the work in progress, it can be executed and checked regularly.

In both cases, the ERC is bound to a specific publication document. This "exit point" is also well-defined in the ERC and is exploited for result validation and to highlight changes in the results after parameter manipulation.

In order to reproduce an analysis, we make use of the containerization technology Docker. Storing both a docker image and the corresponding Dockerfile creates two levels of reproduction: (i) the ability to provide the original runtime environment, and (ii) recreating it from scratch, potentially in a more recent version. Docker mitigates some issues for the reproduction service to run the software of the original researcher, for example with a varying environment of the workspace (Linux, Windows, OS-X), but cannot solve others, for example requiring licensed software (MATLAB™, ArcGIS™) which must not be redistributed.

Existing approaches using Docker to replicate environments (see previous chapter) require readers to follow individual written instructions, and to have expertise in the used software setup. This is not apt for a one-click execution (for readers) nor for automatic content validation (for reviewers and curators). The ERC defines machine-readable conventions for computer systems to control and evaluate the embedded container, namely command-line interface instructions to run it (e.g. "docker run" as part of ERC metadata), and rules to check a successful execution based on the created workspace. The Dockerfile used to build this container defines the environment and command for analysis execution.

After an ERC's execution, the result is evaluated. A minimal evaluation relies on the exit code of the main process in the container. At an advanced level, checksums or contents of files and execution logs can be evaluated.

Once the analysis can be executed by the creation service, the information accessible at runtime, e.g. actually loaded modules and libraries as well as dataset objects, is a fast and reliable way to derive both software metadata and dataset metadata. The potential information stretches from detailed names and versions of attached libraries to the spatial and temporal extent of the data subsets. This is prone to be more correct than analysing source code, using system-wide installed software, trying to read all imaginable data formats while assuming all data files in a workspace are actually used, or manual documentation.

Docker is well suited for storing and transporting software and its dependencies, but it does not serve the needs of data repositories and archives well, that are concerned with bitstream preservation and integrity of files.

3.4 Publication Process

An ERC facilitates a scientific publication that is not only composed of a paper but also the data, the source code, and the software environment, packaged in an executable manner. The executable research compendium described in this work is a fundamental component of the proposed publication process consisting of four consecutive steps (see Figure 2). The process is aligned with peer-review based journals. Their existing practices and protocols can be applied for transfer of compendia between stages and handling the required data, such as related communications or the state within the process.

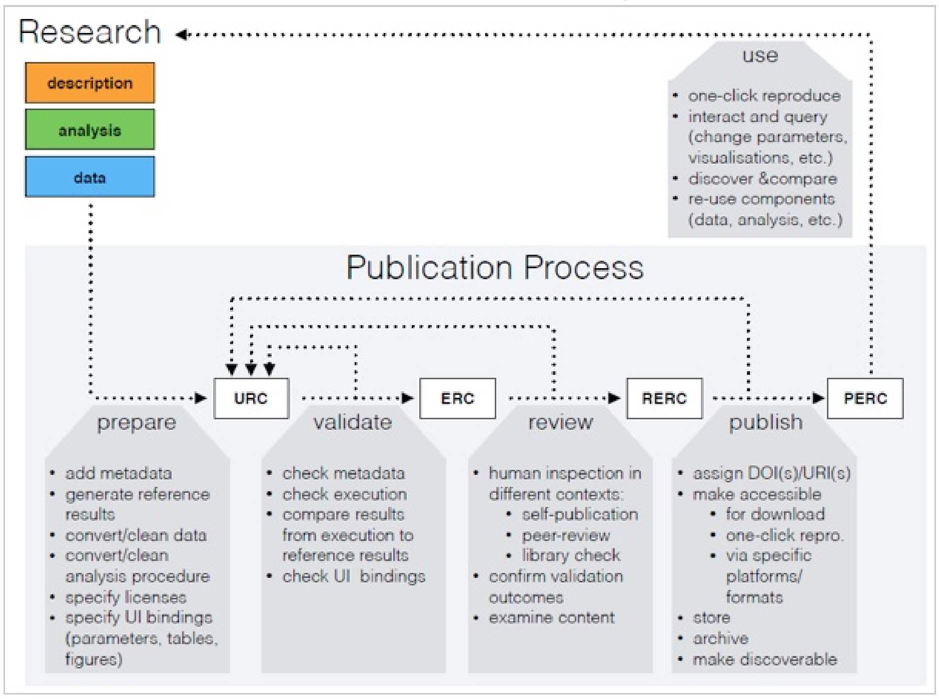


Figure 2: The ERC-based publication process: a research workspace is prepared for a URC, which is validated to become an ERC, which turns into an RERC after peer review, and eventually published as a PERC to be used, e.g. for subsequent cycles.

In the first subprocess, the author *prepares* the research material as described in the previous section resulting in an unvalidated research compendium (URC). A URC is neither executable, nor necessarily visible to the public, but it ideally contains already the entire research material including data set, source code, and the software environment. A URC forms the input to the validation subprocess which involves (i) metadata verification, (ii) the error-free execution of the reproducible parts of the paper, and (iii) a confirmation from the primary author that the automatically reproduced results are identical to the actual publication. The first two aspects should be automated as part of a submission. If the validation succeeds, it produces an executable research compendium (ERC), which is passed on to the next subprocess.

The *review* process focuses on human inspection. It is necessary to avoid publishing meaningless or questionable compendia. Human inspection can be done by library staff (e.g. to ensure adherence to non-formal criteria), maintainers of a data repository (e.g. to filter offending or illegal content), journal editors and peer reviewers (e.g. to warrant scientific quality in the context of a journal submission). If the review process has a positive outcome, it produces a reviewed executable research compendium (RERC), which constitutes the input to the final subprocess: *publish*. During this subprocess, a RERC is enriched (e.g. by assigning DOIs/URIs) and then made accessible as a published executable research compendium (PERC). Researchers can reproduce the results reported in a PERC via a single click. They can also reuse components of a PERC (e.g. its data) for their own research. In order to make PERCs discoverable and comparable, there has to be a minimum of meta information attached to them.

Finally, if the authors provide UI bindings during the publication process, readers can interact with an ERC in a much deeper way. For example, if the authors specify which variables in the code contain specific threshold values, then readers of the PERC can interactively change them — i.e. via an automatically generated UI — to explore whether the reported outcomes still hold true for different threshold values. Another benefit relates to diagrams and visualizations: readers could select from different visualizations to more easily compare it to a diagram in a second paper.

3.5 Findability and Preservation

ERCs can have layers of meta information which are divided into reproducibility metadata and discovery metadata.

Packages, libraries, and specific versions of software is crucial information for reproducibility. Therefore these must be made available for the URC's validation.

The ERC itself is conceptualized as a ready-for-ingest digital asset as defined in OAIS reference model (CCSDS, 2012). Its outer container uses the BagIt file packaging format by Kunze *et al.* (2011). It provides minimal metadata and checksums for a file based payload. The payload comprises at least (i) the aforementioned Docker image (an inner container) and Dockerfile, (ii) metadata files in different formats such as Codemeta, DataCite (Brase, 2009), (iii) the workspace of the conducted research including code and data files, and (iv) created output documents, e.g. a PDF of the original paper. Research data and code can be stored in the image to make distributed reproduction easier but making them accessible as part of the outer container is advantageous for long-term preservation. This is also the case for the metadata.

By design ERCs encapsulate any information relevant to the reproduction of its contents (cf. "Core parts of ERC"). External contextualization of an ERC can be achieved by generating additional discovery metadata and ultimately connecting the ERC as a whole to the Linked Open Data cloud (Bizer *et al.*, 2009) by using discipline-specific vocabularies such as GeoSPARQL. This is done to support findability of the ERC. Semantic references make ERC metadata interoperable and ready for discovery.

For the retrieval of software metadata, it is necessary to identify software dependencies. The metadata included in an ERC also provides structural information needed to connect the entry point for an execution with the internal structure of the underlying parts (code, data, text, UI bindings). It is therefore reasonable to start connecting the available information about the underlying parts as early in the workflow as possible and thus contextualize the files within an ERC.

Our specification also considers preservation requirements by extending bitstream preservation. The recreation of the original computational environment is vital when rerunning the code of a paper. Jon Claerbout's idea of an article being a mere advertising of the underlying scholarship has been concisely paraphrased in Buckheit *et al.* (1995, p. 5): "The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures". Consequently, when it comes to preservation, this complete environment must be included and the instructions must be executable. This refers to the internal structure and the dependencies.

In addition, ERCs can be integrated into the existing ecosystem of persistent identifiers using mechanisms to uniquely and permanently identify the involved resources and agents. Furthermore, metadata comprises explicit license information, ideally using well-known abbreviations or vocabularies, for each component of a compendium. Authors must be assisted in defining data, software, and text licenses, as this is a tedious but crucial aspect of openness — only a suitable license clearly allows and defines possible reuse.

The user interface definitions (UI bindings) are an optional element of an ERC and allow for interaction with the included analysis. On top of that, they document the analysis code by providing a guide to the main functions and their manipulable parameters. We see different approaches to generate them ranging from fully automatic to manual definition. Fully automatic UI bindings analyze the code and generate UI widgets based on the input type (text, number, etc.). UI bindings are created by the author, for example by means of adding statements in the source code or by using a software designed for this purpose.

3.6 Interaction, Exploration, and Reuse

An ERC not only enables third parties to reproduce the original research results (figures, tables), but also facilitates interaction with them. Because an ERC transparently abstracts diverse analyses, i.e. it is a "white box", it seems trivial to build a web service which downloads a research compendium from a repository, unpacks it, executes it, and returns the results created. This extends the minimalistic control functionalities described before with relevant features. Once reproducibility becomes easy to use via ERCs, scientists can benefit from the ability to drill deeper into the computational instructions. For example, they can interactively investigate whether the originally reported results change if they manipulate an underlying parameter, which are fixed in a 'classical' article. In an ERC they can be exposed as changeable variables, e.g. via dedicated components in the user interfaces such as a slider. The original author could provide information about what constitutes a reasonable range of values for this variable. In addition, being able to inspect analysis procedures more closely can help in detecting errors and in safeguarding the integrity of the publication process. For example, a reader might update an underlying library with known bugs but leave the ERC otherwise unchanged.

The recent Reinhart and Rogoff case (Reinhart and Rogoff, 2010; Herndon *et al.*, 2014) showed how damaging it can be when a paper insufficiently describes the analysis on which its conclusions are based.

4 Discussion

4.1 Approach

In this paper we propose the executable research compendium as a new form for supporting the creation and provision of research results. The four parts of an ERC open up archival of and interaction with computational research with the following improvements: (i) reviewers obtain tools for easily validating results in scientific publications submitted by researchers, (ii) results are well-grounded since the research steps and data described in an article are shared, and (iii) other researchers benefit because they obtain tools for accessing, reusing, and extending research components.

The problem of scientists each having an individual workflow is tackled by packaging not just the workspace with data and code, but also the actual publication, the runtime environment, and UI bindings. ERCs not only enable third parties to reproduce the original research and hence recreate the original research results (figures, tables), but also facilitate interaction and recombination, e.g. with other data or new methods. This recombination is complex and probably feasible only for compatible data respectively computational methods and thus restricted to particular domains.

4.2 Distinction of ERC

ERCs rely on the concept introduced as research compendium (RC) by Gentleman and Temple Lang (2007). The core difference to an RC is that an ERC is aware of its complete software environment and contains the so called "transformation software". The transformation software generates different views (e.g. PDF documents or graphics) from the RC by processing code chunks within dynamic documents, e.g. by passing them to an interpreter of the so called "definition language" (Gentleman and Temple Lang, 2007). The authors mention the possibility to include "general purpose software" as part of the "auxiliary software" into an RC in case a version must be specific or might disappear. ERCs take this one step further and embed all software required to run the compendium's code, including the interpreter of the "definition language". It effectively removes all requirements towards the host machine except a Docker runtime.

Research Objects focus in the linked data technology. As such, they are characterized by aggregation and referencing of distributed resources, including workflows and their execution. In contrast, ERCs seek to provide consistent packaging for simple reproduction ("one-click") and interlinking within the contained parts. Both ROs and ERCs are containers for research data and code needed to preserve and reproduce an analysis but they approach this goal from different directions (cf. RO bundle). ROs focus on the outer perspective: provenance, dependency, interconnections. ERCs target the inner scope: consistency, completeness, independence.

4.3 Creation and Usage

The two creation patterns support the majority of user workflows. Post-hoc is less intrusive during research, while a template can put good reproducible research practices (e.g. literate programming) into effect. It remains to be examined which approach requires less effort or finds higher uptake by researchers, and if the expected benefits of publishing an ERC instead of a classic paper outweigh the additional efforts for authors. In any case researchers have to adjust their workflows to remove all "manual" tasks in favour of replicable scripts.

The packaging format specified by the ERC assists different applications (e.g. one-click reproduce, long-term archival, research information systems) and thus allows future usage independent from the described purposes. The runtime packaging feature is based on open source software with high uptake in industry, but although an open specification process is underway (cf. the Open Container Initiative), longevity issues cannot be put aside at this point.

Storing the image and its recipe, i.e. the Dockerfile, increases the chances of reproducing work long after the original publication. With this burden on the Dockerfile, it must be evaluated what criteria it shall comply with for long-term archival, because clarity and extensiveness (e.g. explicit versions) outweigh typical concerns (e.g. image size, up-to-dateness). Nevertheless, ERCs mitigate some of Howe's (2012) challenges for virtualization, e.g. reuse and limitations to interactivity.

ERCs do not cover privacy issues, requiring to anonymize data prior to publication. This restriction holds for other publication forms as well. ERCs cover security concerns, because the Docker container provides an effective sandboxing mechanism.

Interactivity is confined by computation time. Because manipulation of one parameter requires a rerun of the whole container, an execution platform needs to be transparent and reliable in communicating this issue to the user, for example, by indicating the expected run duration.

The efforts needed to create an ERC can be minimized by using automated metadata derivation. This aspect also applies to the generation of UI bindings, which must be possible without too much effort by the author. We still have to evaluate to which extent UI bindings can be generated automatically.

4.4 Challenges

The key challenges for a publishing process based on ERC are (i) the creation of ERCs must be easy for authors, (ii) ERC-based interaction, discovery, exploration, and reuse must provide sufficient benefits for scientists to result in a broad uptake of the concept, and (iii) ERCs must handle diverse workspaces and integrate requirements from all stakeholders. The adoption of the ERC will be limited if tackling these challenges leads to a system that is too complex or not understood by users.

Some core aspects of the publication process cannot be defined to the required level of detail at this point, namely management of review state (Put review metadata and state into the container or keep it outside?) or transfer and storage (Can journal platforms handle ERC file sizes and execution?). It is inevitable to accompany the concepts in this work by a practical implementation to settle these questions.

Research compendia are designed to support science during preparation, implementation, and publication. However, the definition of ERCs alone cannot enforce correct methodology or proper reviews. Communities of practice have to develop conventions and to expand education to put compendia into effect. Currently, researchers use libraries rarely to curate their work. ERCs can connect the research and library communities as a step towards better digital curation, one of the major challenges for memory institutions of the future.

5 Conclusion

Reproducible research is a goal with extraordinary meaning for scientific publications. ERCs provide an innovation for the publication process by opening its result for reuse. Subsequently they help to implement the goals of Open Science. In this paper we provide the following key contributions:

- a compendium-based publication process
- reproduction of the computational steps, the results, and visualizations in ERCs
- packaging computational research for long-term reproducibility
- new ways of interaction with research

Reproducible research can only be realized by creating technical and communicational solutions for the difficulties outlined above. Our design focuses on the interaction of the different roles within the scientific research and publication culture. The executable research compendium reduces efforts on the technical side of reproducibility and thus fosters the community’s acceptance of openness and reproducibility and creates the basis for open collaborations between scientists.

6 Future Work

Because ERCs only work when taking care of the specifics of both a scientific domain and the software, we expect a focussed solution to deliver best results for users. We are currently in the process of developing an open, formal specification for ERCs and an open source web-platform allowing users to build, store, execute, and interact with ERCs in the context of computational geosciences in R, going beyond most current research data platforms (see Opening Reproducible Research project on GitHub).

The prototypical implementation will be subject to a series of usability evaluations considering the views of all stakeholders and roles. In particular, the usability while creating ERCs will be a crucial factor for the acceptance of ERCs as a form of publication. Although ERCs are designed with existing platforms and workflows in mind, a practical evaluation of their successful integration is needed, i.e. a demonstration of a complete publishing process from submission on a journal platform, evaluation during review, storage in repositories, publication and interaction on an online platform, and long-term archival. Inherently manual steps of a publishing workflow, e.g. copy-editing, create new challenges for systematic interpretation of the output. A user friendly interactive execution of compendia has to address open questions regarding parameter transfer and partial container execution.

Acknowledgements

This work is supported by the project Opening Reproducible Research (see also Offene Reproduzierbare Forschung) funded by the German Research Foundation (DFG) under project numbers PE 1632/10-1, KR 3930/3-1 and TR 864/6-1.

References

[1] Barnes, Nick. "Publish Your Computer Code: It Is Good Enough." *Nature News* 467, no. 7317 (October 13, 2010): 753-753. https://doi.org/10.1038/467753a

[2] Bechhofer, Sean, Iain Buchan, David De Roure, Paolo Missier, John Ainsworth, Jiten Bhagat, Philip Couch, *et al*. "Why Linked Data Is Not Enough for Scientists". *Future Generation Computer Systems, Special section: Recent advances in e-Science* 29, no. 2 (February 2013): 599-611. https://doi.org/10.1016/j.future.2011.08.004

[3] Belhajjame, Khalid, Jun Zhao, Daniel Garijo, Matthew Gamble, Kristina Hettne, Raul Palma, Eleni Mina, et al. "Using a Suite of Ontologies for Preserving Workflow-Centric Research Objects". *Web Semantics: Science, Services and Agents on the World Wide Web* 32 (May 2015): 16-42. https://doi.org/10.1016/j.websem.2015.01.003

[4] Bizer, Christian, Tom Heath, and Tim Berners-Lee. "Linked Data - The Story So Far." *International Journal on Semantic Web and Information Systems* 5, no. 3 (33 2009): 1-22. https://doi.org/10.4018/jswis.2009081901

[5] Bloom, T., Ganley, E., & Winker, M. (2014). Data access for the open access literature: PLOS's data policy. *PLoS Biol* 12(2), https://doi.org/10.1371/journal.pbio.1001797

[6] Boettiger, C. (2015). An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review* 49(1), 71-79. https://doi.org/10.1145/2723872.2723882

[7] Bollen, K., Cacioppo, J. T., Kaplan, R. M., Krosnick, J. A., & Olds, J. L. (2015). Social, behavioral, and economic sciences perspectives on robust and reliable science: Report of the Subcommittee on Replicability in Science, Advisory Committee to the National Science Foundation Directorate for Social, Behavioral, and Economic Sciences.

[8] Borgman, C. (2007). *Scholarship in the digital age: information, infrastructure and the internet*. MIT University Press Group Ltd., p. 336. ISBN: 9780262026192

[9] Brase, J. "DataCite — A Global Registration Agency for Research Data." In *Fourth International Conference on Cooperation and Promotion of Information Resources in Science and Technology*, 2009. COINFO '09, 257-61, 2009. https://doi.org/10.1109/COINFO.2009.66

[10] Buckheit, Jonathan B., and David L. Donoho. "WaveLab and Reproducible Research". In *Wavelets and Statistics*, edited by Anestis Antoniadis and Georges Oppenheim, 55-81. *Lecture Notes in Statistics* 103. Springer New York,

1995. https://doi.org/10.1007/978-1-4612-2544-7_5

[11] CCSDS (2012). Consultative Committee for Space Data Systems, Reference model for an open archival information system (OAIS), Magenta Book CCSDS 650.0-M-2, Open Archives Initiative, 2012.

[12] Chen, Xiaoli, Sünje Dallmeier-Tiessen, Anxhela Dani, Robin Dasler, Javier Delgado Fernández, Pamfilos Fokianos, Patricia Herterich, and Tibor Šimko. "CERN Analysis Preservation: A Novel Digital Library Service to Enable Reusable and Reproducible Research". In *Research and Advanced Technology for Digital Libraries*, edited by Norbert Fuhr, László Kovács, Thomas Risse, and Wolfgang Nejdl, 9819:347-56. Cham: Springer International Publishing, 2016. https://doi.org/10.1007/978-3-319-43997-6_27

[13] Chirigati, F., Shasha, D., & Freire, J. (2013). Reprozip: Using provenance to support computational reproducibility. *Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance*, Lombard, IL, USA, April 2-3, 2013. USENIX Association Berkeley, CA, USA.

[14] Clarkson, Chris, Mike Smith, Ben Marwick, Richard Fullagar, Lynley A. Wallis, Patrick Faulkner, Tiina Manne, *et al*. "The Archaeology, Chronology and Stratigraphy of Madjedbebe (Malakunanja II): A Site in Northern Australia with Early Occupation". *Journal of Human Evolution* 83 (June 2015): 46-64. https://doi.org/10.1016/j.jhevol.2015.03.014

[15] Dua, R., A. R. Raja, and D. Kakadia. "Virtualization vs Containerization to Support PaaS". In *2014 IEEE International Conference on Cloud Engineering (IC2E)*, 610-14, 2014. https://doi.org/10.1109/IC2E.2014.41

[16] Easterbrook, S. M. (2014). Open code for open science?. *Nature Geoscience* 7(11), 779-781. https://doi.org/10.1038/ngeo2283

[17] European Commission (2015). Access to and preservation of scientific information in Europe. Report on the implementation of Commission Recommendation C(2012) 4890 final. https://doi.org/10.2777/975917

[18] Gentleman, R., & Temple Lang, D. (2007). Statistical analyses and reproducible research. *Journal of Computational and Graphical Statistics* 16:2, 1-23, https://doi.org/10.1198/106186007X178663

[19] Glandon, P. (2011). "Appendix to the Report of the Editor: Report on the American Economic Review Data Availability Compliance Project. *American Economic Review* 101(3): 695-699.

[20] Goecks, Jeremy, Anton Nekrutenko, and James Taylor. "Galaxy: A Comprehensive Approach for Supporting Accessible, Reproducible, and Transparent Computational Research in the Life Sciences". *Genome Biology* 11 (2010): R86. https://doi.org/10.1186/gb-2010-11-8-r86

[21] Goodman, S. N., Fanelli, D., & Ioannidis, J. P. (2016). What does research reproducibility mean? *Science Translational Medicine* 8(341), 341ps12-341ps12. https://doi.org/10.1126/scitranslmed.aaf5027

[22] Haak, Laurel L., Martin Fenner, Laura Paglione, Ed Pentz, and Howard Ratner. "ORCID: A System to Uniquely Identify Researchers." *Learned Publishing* 25, no. 4 (October 1, 2012): 259-64. https://doi.org/10.1087/20120404

[23] Hanson, Karen L., Tim DiLauro, and Mark Donoghue. "The RMap Project: Capturing and Preserving Associations Amongst Multi-Part Distributed Publications". In *Proceedings of the 15th ACM/IEEE-CS Joint Conference on Digital Libraries*, 281-282. JCDL '15. New York, NY, USA: ACM, 2015. https://doi.org/10.1145/2756406.2756952

[24] Harnad, Stevan, *et al*. "The access/impact problem and the green and gold roads to open access." *Serials Review* 30.4 (2004): 310-314. https://doi.org/10.1080/00987913.2004.10764930

[25] Herndon, T., Ash, M., & Pollin, R. (2014). Does high public debt consistently stifle economic growth? A critique of Reinhart and Rogoff. *Cambridge Journal of Economics*, 38(2), 257-279. https://doi.org/10.1093/cje/bet075

[26] Howe, B. (2012). Virtual appliances, cloud computing, and reproducible research. *Computing in Science & Engineering*, 14(4), 36-41. https://doi.org/10.1109/MCSE.2012.62

[27] Hung, L. H., Kristiyanto, D., Lee, S. B., & Yeung, K. Y. (2016). GUIdock: Using Docker Containers with a Common Graphics User Interface to Address the Reproducibility of Research. *PloS one*, 11(4), e0152686.

[28] Knuth, Donald E. "Literate Programming." *The Computer Journal* 27, no. 2 (May 1984): 97-111. https://doi.org/10.1093/comjnl/27.2.97

[29] Kunze, J., Littman, J., Madden L., Summers, E., Boyko, A. & Vargas, B. "The bagit file packaging format (v0. 97)." Washington DC (2011).

[30] Leek, J. T., & Peng, R. D. (2015). Opinion: Reproducible research can still be wrong: Adopting a prevention approach. *Proceedings of the National Academy of Sciences*, 112(6), 1645-1646.

[31] Markowetz, Florian. "Five Selfish Reasons to Work Reproducibly." *Genome Biology* 16 (2015): 274. https://doi.org/10.1186/s13059-015-0850-7

[32] Marwick, Ben. "1989-Excavation-Report-Madjebebe." March 23, 2015. https://doi.org/10.6084/m9.figshare.1297059.v2

[33] Mecklenburg, R. (2004). *Managing Projects with GNU Make*, 3rd Edition. O'Reilly Media, ISBN: 0-596-00610-1.

[34] Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239), 2, ISSN: 1075-3583.

[35] Nature Editorial. "Code Share". *Nature* 514, no. 7524 (2014): 536-536. https://doi.org/10.1038/514536a

[36] Nosek, B. A., G. Alter, G. C. Banks, D. Borsboom, S. D. Bowman, S. J. Breckler, S. Buck, *et al*. "Promoting an Open Research Culture". *Science* 348, no. 6242 (26 June 2015): 1422-25. https://doi.org/10.1126/science.aab2374

[37] Palma, Raúl, Oscar Corcho, Piotr Hołubowicz, Sara Pérez, Kevin Page, and Cezary Mazurek. "Digital Libraries for the Preservation of Research Methods and Associated Artifacts". In *Proceedings of the 1st International Workshop on Digital Preservation of Research Methods and Artefacts*, 8-15. DPRMA '13. New York, NY, USA: ACM, 2013. https://doi.org/10.1145/2499583.2499589

[38] R Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing. Vienna, Austria, 2014.

[39] Reinhart, C. M., & Rogoff, K. S. (2010). Growth in a time of debt (digest summary). *American Economic Review*, 100(2), 573-578. https://doi.org/10.2469/dig.v40.n3.19

[40] Santana-Perez, I., Ferreira da Silva, R., Rynge, M., Deelman, E., Pérez-Hernández, M. S., Corcho, O. (2017). Reproducibility of Execution Environments in Computational Science Using Semantics and Clouds. *Future Generation Computer Systems*. https://doi.org/10.1016/j.future.2015.12.017

[41] Stodden, V., Leisch, F., & Peng, R. D. (Eds.). (2014). Implementing reproducible research. CRC Press.

[42] Stodden, V., Miguez, S. and Seiler, J. (2015). Researchcompendia. org: Cyberinfrastructure for reproducibility and collaboration in computational science. *Computing in Science & Engineering*, 17(1), pp.12-19.

[43] Thain, D., Ivie P., Meng, H. (2015). Techniques for Preserving Scientific Software Executions: Preserve the Mess or Encourage Cleanliness?. https://doi.org/10.7274/R0CZ353M

[44] Vandewalle, P., Kovacevic, J., & Vetterli, M. (2009). Reproducible research in signal processing. *IEEE Signal Processing Magazine*, 26(3), 37-47. https://doi.org/10.1109/MSP.2009.932122

[45] Vines, T. H., Albert, A. Y., Andrew, R. L., Débarre, F., Bock, D. G., Franklin, M. T., Rennison, D. J. (2014). The availability of research data declines rapidly with article age. *Current Biology* 24(1), 94-97. https://doi.org/10.1016/j.cub.2013.11.014

[46] Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D., Owen, S., Soiland-Reyes, S., Dunlop, I., Nenadic, A., Fisher, P., Bhagat, J., Belhajjame, K., Bacall, F., Hardisty, A., Hidalga, A., Vargas, M., Sufi, S., Goble, C. The Taverna Workflow Suite: Designing and Executing Workflows of Web Services on the Desktop, Web or in the Cloud. *Nucleic Acids Research* 41, no. W1 (1 July 2013): W557-61. https://doi.org/10.1093/nar/gkt328

About the Authors

Daniel Nüst is a researcher at the Institute for Geoinformatics, University of Münster. He completed his studies in Münster with a Diploma in Geoinformatics in 2011 and afterwards worked at 52°North Initiative for Geospatial Open Source Software as a consultant and software developer. Since 2016 he has worked on creating, storing and executing reproducible research packages in the DFG-project "Opening Reproducible Research".

Markus Konkol is a research associate in the DFG-project "Opening Reproducible Research" at the Institute for Geoinformatics, University of Münster. He is currently doing his Ph.D. in the DFG-funded project Opening Reproducible Research that aims at facilitating access to and interaction with research results. His main topic is about interconnecting the textual publication, data sets, source code and UI elements in order to assist scientists in exploring and creating dynamic publications.

Marc Schutzeichel is a research associate in the DFG-project "Opening Reproducible Research". He works at the University and State Library, Münster.

Edzer Pebesma is professor of Geoinformatics at University of Münster since 2007. He is developer and maintainer of several popular R packages for handling and analyzing spatial and spatiotemporal data (sp, spacetime, gstat), co-author of the book Applied Spatial Data Analysis with R, and active member of the r-sig-geo community. He is Co-Editor-in-Chief for the Journal of Statistical Software and Computers & Geosciences, and associate editor for Spatial Statistics.

Chris Kray is a professor of Geoinformatics at the Institute for Geoinformatics (ifgi) at the University of Münster. His research

interests include location-based services, smart cities and human-computer interaction (in particular: interaction with spatial information). Chris is the scientific coordinator of the ITN "GEO-C: enabling open cities" at ifgi, where he works on realising transparency, accessibility and privacy protection in the context of smart cities.

Holger Przibytzin is Department Manager of Scientific Information Systems at the University and State Library, Münster.

Jörg Lorenz is Head of the Science and Innovation department of the University and State Library, Münster.

(At the authors' request on June 14, 2017, the publication year 2017 was added to Reference No. 40, and the publcation year was corrected to 2015 for Reference No. 43.)

4 Reproducibility and practical adoption of GEOBIA with open-source software in Docker containers

Authors & contribution Christian Knoth (50%), Daniel Nüst (50%)

Date 03/2017

Venue *Remote Sensing* (SNIP 2020: 1.71) 10.3390/rs9030290

Licence Creative Commons Attribution (CC BY 4.0)

Article

Reproducibility and Practical Adoption of GEOBIA with Open-Source Software in Docker Containers

Christian Knoth * and Daniel Nüst

Institute for Geoinformatics, University of Münster, Heisenbergstraße 2, 48149 Münster, Germany; daniel.nuest@uni-muenster.de
* Correspondence: christian.knoth@uni-muenster.de; Tel.: +49-251-833-3056

Academic Editors: Norman Kerle, Markus Gerke, Sébastien Lefèvre and Prasad S. Thenkabail
Received: 30 December 2016; Accepted: 6 March 2017; Published: 18 March 2017

Abstract: Geographic Object-Based Image Analysis (GEOBIA) mostly uses proprietary software, but the interest in Free and Open-Source Software (FOSS) for GEOBIA is growing. This interest stems not only from cost savings, but also from benefits concerning reproducibility and collaboration. Technical challenges hamper practical reproducibility, especially when multiple software packages are required to conduct an analysis. In this study, we use containerization to package a GEOBIA workflow in a well-defined FOSS environment. We explore the approach using two software stacks to perform an exemplary analysis detecting destruction of buildings in bi-temporal images of a conflict area. The analysis combines feature extraction techniques with segmentation and object-based analysis to detect changes using automatically-defined local reference values and to distinguish disappeared buildings from non-target structures. The resulting workflow is published as FOSS comprising both the model and data in a ready to use Docker image and a user interface for interaction with the containerized workflow. The presented solution advances GEOBIA in the following aspects: higher transparency of methodology; easier reuse and adaption of workflows; better transferability between operating systems; complete description of the software environment; and easy application of workflows by image analysis experts and non-experts. As a result, it promotes not only the reproducibility of GEOBIA, but also its practical adoption.

Keywords: reproducibility; GEOBIA; Docker; conflict monitoring; reproducible research; object-based image analysis; QGIS; containerization

1. Introduction

1.1. Motivation

For a scientific method to gain impact in research, it must be understandable and replicable by fellow scientists. To gain impact in practice, it also needs to be easy to adopt by users from different fields. In computational sciences, such as Geographic Object-Based Image Analysis (GEOBIA), replicability requires access to code and data. Practical applicability in addition requires ease of use and the customizability of methods. In this work, we present a novel solution for the GEOBIA community to conduct research in a reproducible way by packaging code, data and even the required runtime environment in executable units called containers. We discuss the dilemma of balancing reproducibility, ease of use and the customizability of containerized methods and propose an approach to enable interaction with those methods through parameterized containers and a graphical user interface. The solutions can foster reproducibility and practical adoption of complex workflows, not least because they rely on Free and Open-Source Software (FOSS).

1.2. Reproducible Research

Openness in research is not a new topic, but there is a recent trend towards transparency and availability under the terms Open Science and Open Access (cf. [1]). All stakeholders in the research process contribute rules, incentives or guidelines to foster openness: on the funding side, the European Union requires Open Access as part of the Horizon 2020 framework programme (cf. [2]) and builds the European Open Science Cloud [3]; on the publishing side, journals such as Science [4] and Bioinformatics [5] encourage reproducibility; on the research institution side, scientists themselves argue for reproducibility with "Five selfish reasons to work reproducibly" [6], develop guidelines like the Vienna principles [7], or publish "Ten Simple Rules for Reproducible Computational Research" arguing in favour of a proper scientific workflow simply to be able to reproduce *your own* results [8]. A core notion of all of these activities is the ideal to publish data, methods, and software along with scholarly publications.

A definition of the term reproducibility is far from trivial, not least because it is often used together with other terms to describe different levels of redoing. The Vienna Principles' definition focuses on traceability [7]; others treat "reproducibility" and "replicability" as interchangeable [9] or completely different terms [10]; others qualify the term further, e.g., "computational reproducibility" [5].

For the remainder of this work, we use "reproduce", "reproducibility" or "computational reproducibility" to say a third party can re-run an analysis using code and data provided by the author of a published work, and this execution creates the same computational result (following the definition by Peng [5]). Besides technical difficulties, which is the focus of this work, computational reproducibility is likewise a question of proper practices. For example, researchers must fix seeds of random number generators [8], design transparent processing workflows [11] or utilize virtualization [12]. Nevertheless, the uniqueness of data (e.g., it can only be captured once by one sensor) or processing environments (e.g., supercomputers) can make "real" replication of results impossible, so that trust in the applied methods must be established instead [13]. In (GE)OBIA, the term "reproducible" is thus far used to describe a shift from manual analysis (based on interaction with a user interface) to script-based analysis using sets of processing steps and classification rules (cf. [14–16]).

Packaging data, code and documentation for reproducibility was previously described under the term research compendia by means of programming language-specific packaging mechanisms [9,17] or as research objects with a focus on workflows using semantic enrichment and provenance [18]. Alternative approaches to create self-contained packages for reproducibility are tracing techniques (cf. [19,20]).

1.3. FOSS for GEOBIA

To achieve trust in computational reproducibility, open-sourcing of workflows and the underlying software is crucial. While benefits of Free and Open-Source Software for business and security have been documented widely (see for example [21] and [22]), the important aspect of FOSS in science is the potential for scrutiny and collaboration. FOSS facilitates audits down to the level of the source code. It also promotes the reuse, improvement and adaption of a methodology or software. FOSS licensing models (for a quick introduction we recommend http://choosealicense.com/) allow to combine individual contributions of functional parts into a solution for a larger problem at hand. FOSS projects must allow (technically and legally) maintenance and re-purposing by third parties. This modularity lies at the roots of many open-source software projects and is propagated by the *Unix philosophy* [23]: Each programme should only provide a specific feature and excel at it.

A number of publications at GEOBIA conferences over the last few years demonstrate the growing interest in an FOSS approach to GEOBIA. For example, [24] developed a workflow for urban Land Use and Land Cover (LULC) classification using GRASS GIS and R. They also published the analysis reproducibly as an executable notebook. The work in [25] used the Orfeo ToolBox and R for automated selection of segmentation parameters and classification of urban scenes. The work in [26] applied

GRASS GIS to map impervious surfaces using aerial images and LIDAR (Light Detection and Ranging) data. Specific OBIA FOSS projects exist, as well; for example GeoDMA [27] and InterIMAGE [28] for desktop environments or InterCloud [29] for cloud computing infrastructures. Nevertheless these activities are still in the minority.

1.4. Balancing Reproducibility and Customizability for Practical Adoption

Complex workflows based on FOSS often integrate a number of independent and interdependent tools. These building blocks are developed by different communities and in differing maintenance cycles. Their use can be visible or invisible (e.g., transitive dependencies) to the analyst. This hampers reproducibility because of potential compatibility conflicts between different software packages in changing versions. Containerizing a computational method with all required software and data mitigates these problems because it eliminates the need to recreate the original runtime environment on the executing computer. In addition to the positive implications for research (see Section 1.2), we see a high potential in this approach to foster practical adoption of new GEOBIA methods. One main barrier to the application of GEOBIA approaches by practitioners besides cost factors are technical challenges in adopting complex analysis methods for their use case. Containerization can significantly simplify this process by providing non-experts with easy access to complex methods.

However, another important aspect with regard to practical adoption is the transferability of a method, i.e., the applicability to different study areas or different image types. GEOBIA facilitates the development of adaptive and transferable workflows, e.g., through the inclusion of context knowledge and human semantics [30]. There has been significant progress in the GEOBIA community concerning transferability, e.g., through automatic determination of segmentation parameters [31,32] or classification thresholds [33] and fuzzy classification [34]. However, although the workflows are increasingly robust, some adaptions (e.g., of segmentation parameters or classification thresholds) are usually necessary to tune an analysis method to a specific study area or image type [35,36]. Therefore, the possibility to customize a GEOBIA method is an important requirement for practical adoption, which can be hampered if analysis tools are containerized as black boxes without manipulation options. To reconcile reproducibility and customizability and thereby facilitate practical adoption, containerization should allow one to change model parameters or to use one's own input data in a containerized method. For widespread adoption, a Graphical User Interface (GUI) is required.

1.5. Contribution and Overview

The main contribution of this work is a fully-reproducible and open workflow for Geographic Object-Based Image Analysis (GEOBIA). We package a GEOBIA example study including the data and specific combination of software in an executable container. The container is parameterized so that the analysis can be evaluated and adapted. Users can interact with the container and manipulate an analysis through a GUI. The container is based on mainstream IT technology, and the software is a collection of general-purpose scripts, image analysis libraries and Geographic Information Systems (GIS). All components are FOSS. The implemented method detects destruction in a bitemporal image subset of a conflict area and builds on previous work [37,38]. The application of remote sensing for monitoring of human rights issues has been explored in a number of studies, and GEOBIA has proven to be a promising tool, e.g., for refugee camp monitoring [39,40] or damage assessment [41]. A comprehensive review of research and applications in this field (pixel- as well as object-based analyses) can be found in [42]. An open and transparent approach is specifically advantageous in human rights fact-finding because non-profit organizations face budget restrictions and because transparency is crucial when using complex techniques in a politically sensitive environment [43].

The following section describes the image analysis workflow, how it is implemented with FOSS and how it is made reproducible (Section 2). Section 3 explains the execution of the packaged analysis. Finally, we discuss the solution and the challenges (Section 4), and conclude with a summary and outlook in Section 5. This paper is an extended version of our contribution to the GEOBIA 2016 conference [44].

2. Materials and Methods

2.1. Data

Two images of a village in Darfur, Sudan, are our example data. They are available online as part of a blog post by the American Association for the Advancement of Science (AAAS) Geospatial Technologies Project [45]. The copyright holder DigitalGlobe granted permission to re-publish them as part of this work. The images are previews of remote sensing imagery of the village Jonjona, located at 13.686°N, 24.979°E, before (December 2004) and after (February 2007) reported attacks in the area (see Figure 1). They were downloaded from the website in *jpg* format, manually georeferenced (entering coordinates retrieved from georeferenced imagery), resampled to a spatial resolution of 0.5 metres (nearest neighbour resampling), and saved as GeoTIFF files. The spatial resolution approximates the one of current commercial very high resolution satellites.

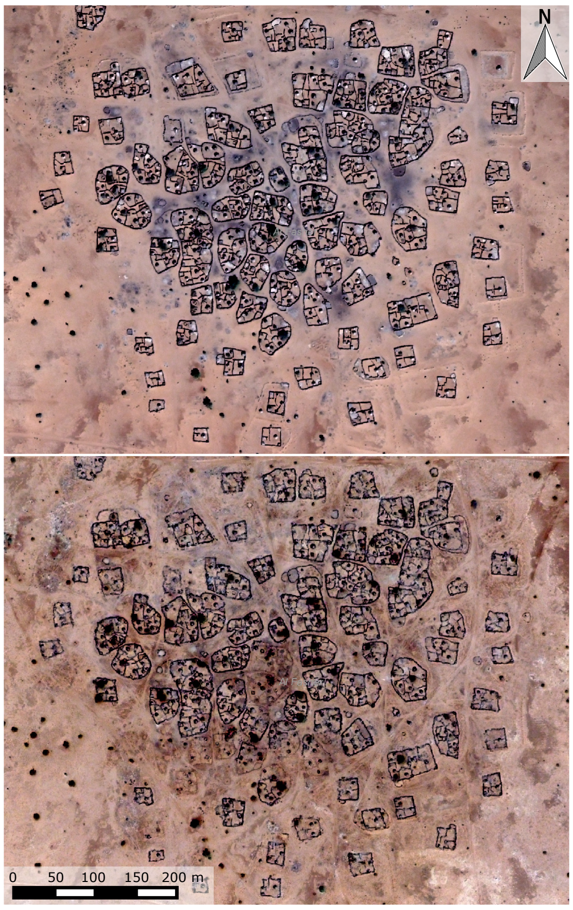


Figure 1. Example image of a village in Darfur (location: 13.686°N, 24.979°E) before (December 2004, top) and after (February 2007, bottom) a reported attack. Images © 2016 DigitalGlobe.

2.2. Example Analysis: Conflict Damage Assessment

2.2.1. Summary

The analysis model applies a rule-based approach to investigate bi-temporal images at two different scales represented by different image object levels. It demonstrates image interpretation using object-based features (e.g., shape, topological and hierarchical relations) using FOSS and is a simplified version of the method described in [38]. In the first stage, the imagery is automatically searched for areas where settlements exist in the pre-conflict image (see Section 2.2.2). In the second stage, the change analysis is conducted on a finer segmentation level (see Section 2.2.3) within detected settlement areas. In these areas, the model identifies changed dwelling structures using relative change values, as well as shape and size. As demonstrated in [38], this approach facilitates the application in complex analyses of images with varying properties (such as sensor configurations, illumination conditions), especially when spectral features are investigated. Figure 2 shows an overview of the analysis workflow.

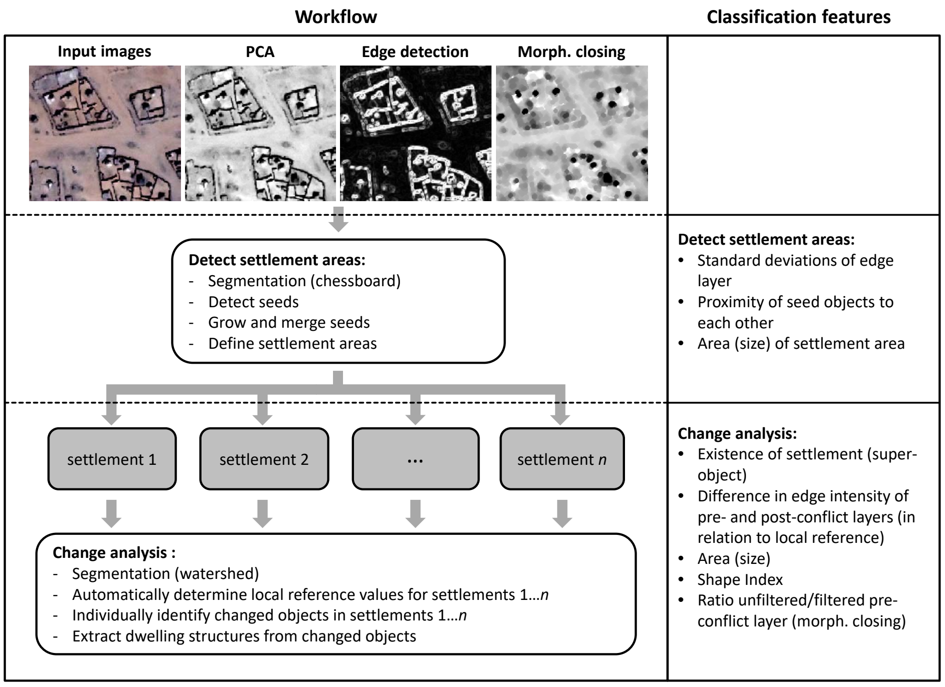

Workflow
Classification features
Input images
PCA
Edge detection
Morph. closing
Detect settlement areas:
- Segmentation (chessboard)
- Detect seeds
- Grow and merge seeds
- Define settlement areas
Detect settlement areas:
• Standard deviations of edge layer
• Proximity of seed objects to each other
• Area (size) of settlement area
settlement 1
settlement 2
...
settlement n
Change analysis :
- Segmentation (watershed)
- Automatically determine local reference values for settlements 1…n
- Individually identify changed objects in settlements 1…n
- Extract dwelling structures from changed objects
Change analysis:
• Existence of settlement (super-object)
• Difference in edge intensity of pre- and post-conflict layers (in relation to local reference)
• Area (size)
• Shape Index
• Ratio unfiltered/filtered pre-conflict layer (morph. closing)

Figure 2. Overview of the example analysis workflow. It shows the key analysis steps on the left and the applied features for the object-based change detection and classification on the right. The first row shows the input images and image processing results created in the first step of the workflow and used as input for the object-based analysis in the subsequent steps (modified after [38]).

2.2.2. Detect Settlement Areas

First, a Principal Component Analysis (PCA) is applied to each file in order to compress the highly redundant spectral information of the three RGB bands to one dimension, the first principal component. The results are then used as pre- and post-conflict layers of a bi-temporal dataset. The detection of settlement areas is performed on a chessboard segmentation level with a 25-m side length. The segments are analyzed regarding the edge intensity of the pre-conflict image since edge detection has proven to be an effective means for the identification and analysis of diverse anthropogenic structures [46–48]. In this example, the edge intensity is determined using an edge detection algorithm after [49]. Segments are analyzed regarding the standard deviation of the edge layer values within each segment. The standard deviation is a good measure to identify anthropogenic structures on this coarse image object level because it accounts for the intensity of edges of those structures in contrast to the background. In addition to the edge intensity, the proximity of segments possibly covering settlement structures (candidate segments) to each other is taken into account. This reflects the specific structure of the sparsely developed villages in Darfur. While dwelling units within these villages are not directly connected, they usually do exist in certain proximity to each other. An area is therefore considered a settlement if it is composed of candidate segments that occur in a certain proximity ($\leq$100 m) to each other. Those segments are grown and merged into a coherent settlement area. Single candidate segments without proximity to others are ignored.

The threshold for the standard deviation of the edge layer within each segment is a predefined value that can be tuned by the user (see Section 3.2). In addition, a minimum size of a settlement area can be defined to focus the analysis on larger settlements.

2.2.3. Change Analysis within Settlement Areas

The analysis model segments the pre-conflict layer and analyzes the resulting image objects regarding their change values derived from the pre- and post-conflict temporal layers. This exemplary method is an "Image-object overlay", following the categorization by [50]. The workflow can be divided into three major steps: (i) feature extraction and segmentation; (ii) change analysis; and (iii) extraction of dwelling objects from changed objects.

In the first step, image objects are created using a watershed segmentation algorithm, which is based on the identification of local extrema [51]. The segmentation is hampered by the specific structural properties of the objects of interest. The buildings are often directly attached to fences or walls, so they poorly separate from the background. Earlier studies in similar areas showed that the mathematical morphology can eliminate such interfering features [37,52]. Therefore, a morphological closing operator with a disc-shaped structural element of a 3-pixel radius precedes the segmentation to smooth out small and linear features.

In the second step, the objects are analyzed for structural differences between the pre- and post-conflict layer. The analysis focuses on the change of edges because the spectral information in the example images is limited. The change is calculated as the difference in mean edge density per object.

To extract possibly changed objects regarding this change attribute, the applied method does not use absolute values, but instead, detects these objects using a local reference value. Hierarchical relations, using settlement areas as super-objects, allow one to analyze changes of sub-objects with regard to the distribution of the change values of all other sub-objects in the same settlement (e.g., using the mean, minimum or maximum change value). In our example, we compute a "mode value". It represents the most abundant degree of change in the corresponding settlement and is used as a reference. To calculate this reference, the value range of the change feature of all objects within a settlement area is divided into equal intervals. Each object is then classified into its corresponding value range interval. The interval covering the largest area is chosen as the reference interval. The mean change feature value of the objects within the reference interval is defined as the reference value within the corresponding village. All objects are investigated by the difference between their change and the local reference value (for more details, please refer to [38]).

We apply two methods to analyze the objects based on this difference: a thresholding and a k-means cluster analysis. The thresholding defines a minimum value and determines the method's sensitivity to change. The threshold can be tuned manually, and its default value is 0.3. Disappeared objects are detected based on this threshold. The cluster analysis isolates disappeared dwellings in the cluster of highest difference. Its result is not used in the detection of disappeared objects, but provided for manual inspection (see Section 2.2.4).

In the third step, the disappeared objects are investigated regarding their extent, shape and values in the pre-conflict morphological closing layer in relation to the unfiltered layer, i.e., regarding the impact of the closing operator (see Figure A1). The step distinguishes between changed dwelling structures and other, similarly changed objects (e.g., fences). The shape of objects is computed using the shape index [53,54]. It measures how well an object approximates a circle. The more the shape differs from a circle, the higher the shape index value.

The impact of the morphological closing filter on the objects grey scale values is determined to identify features that do not fit the structural element (e.g., small and linear structures), but have similar shape attributes after applying segmentation on the filtered image. This is done by calculating the ratio of the standard deviation values (per object) of the unfiltered pre-conflict layer to those after filtering. For the shape and size attributes, fixed thresholds are applied to extract objects of interest, i.e., huts and sheds. More details on the object properties and thresholds used in the example analysis can be found in Table A1.

2.2.4. Analysis Output

The workflow produces three outputs (see Figure 3): (i) the main output: a point shapefile with centroids of dwelling objects detected as disappeared; (ii) a polygon shapefile of settlement areas (see Section 2.2.2); and (iii) a supplementary shapefile for understanding the change detection result. It contains polygons of all objects (changed and unchanged) without the thresholding of the second step. The polygon attributes include among others the change cluster (resulting from the unsupervised clustering), as well as the computed change feature for each polygon. They can be used to refine the analysis workflow, e.g., by adapting thresholds. Figure 4 shows outputs (i) disappeared dwellings (resulting from a change sensitivity of 0.33) and (ii) the settlement area (resulting from a settlement detection sensitivity of 0.3). The result well reflects the overall pattern of destruction showing hot spots in the central and southwestern parts of the village. The accuracy is of course impacted by the limited quality and information content of the preview images used as example data. Fifty nine objects were detected as destructed dwelling structures (huts or sheds); 36 of them could be confirmed by visual inspection. Most of the false positives were caused by trees (no infrared channel) and certain configurations of disappeared fences with similar object properties. In addition, we found 20 possibly destructed structures that were not detected by the algorithm.

2.3. Implementation and Packaging of QGIS-Based Workflow

2.3.1. Development in the QGIS Modeler

QGIS is a FOSS GIS [55]. Its processing framework [56] provides access to native QGIS algorithms, as well as a huge number of geoprocessing capabilities of third-party applications, such as GRASS GIS, Orfeo ToolBox, SAGA GIS or R. In addition, user-created algorithms written in Python [57] and consequently the features of any Python library can be added. Analyses can be built using a graphical modeler and are saved as model files. These models can then be run on a selected set of inputs (e.g., layers in the QGIS desktop application) and user-defined parameters.

In our example, the image processing (PCA including subsequent rescaling, morphological closing, edge detection) and segmentation steps are conducted using the algorithms of the Orfeo ToolBox (OTB). OTB is an open-source C++ library for remote sensing and provides a substantial set of image processing tools, including feature extraction, filtering, classification and segmentation algorithms [58]. The segmentation using the OTB process initially creates polygons without any object-specific properties. Therefore, we use native QGIS algorithms to compute image layer value statistics for each of these objects (e.g., mean values regarding edge intensity). Other native algorithms are used for example to perform calculations on object feature values, to extract objects by thresholds or to identify centroids of polygons for the output of results. To compute shape and size properties of objects, the SAGA GIS algorithm polygon shape indices is used [59]. Another SAGA algorithm called identity is applied to establish hierarchical relations: for each object on the level of single buildings (defined as sub-objects), this process identifies the object on the level of settlements (super-objects) in which it is contained. The IDs of these super-objects are saved as attributes of the sub-objects.

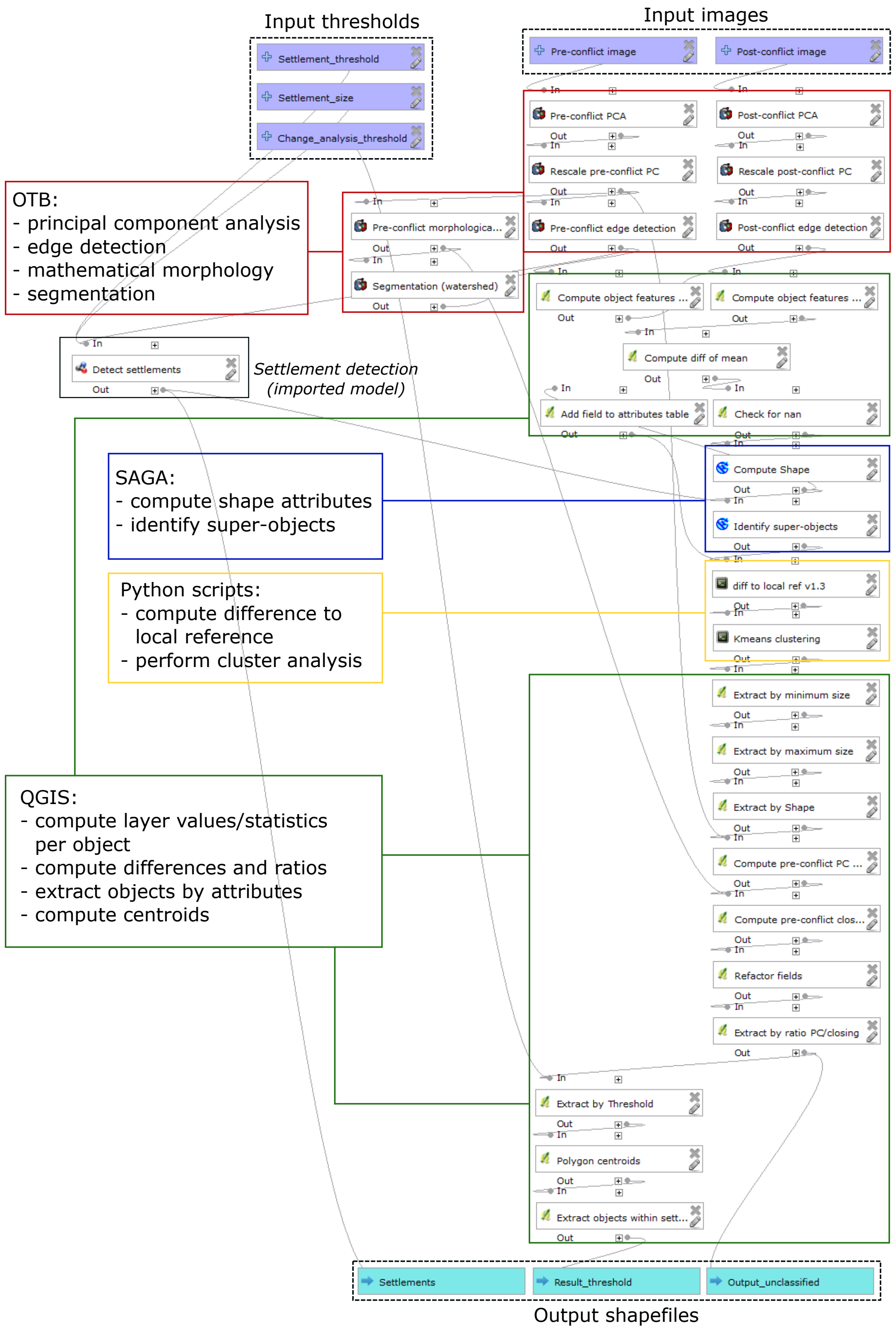


Figure 3. Screenshot of the QGIS graphical workflow modeler showing the example change analysis. Two input images and three numerical parameters are shown in purple boxes at the top. The inputs, processing steps and outputs are connected by grey arcs and roughly ordered from top to bottom. The analysis steps are based on four libraries highlighted by the colored boxes: OTB (red), QGIS (green), SAGA GIS (blue) and Python (yellow). A detailed view of the sub-model detect settlements (grey, left hand side) is in Figure A2. At the bottom, the three output shapefiles are shown in turquoise boxes.

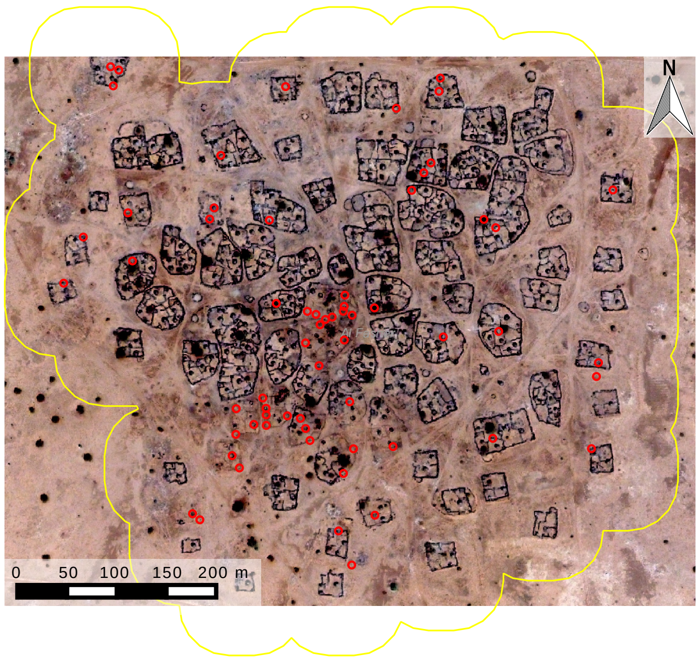


Figure 4. Post-conflict image (location: 13.686°N, 24.979°E) with two results of the example analysis. The detected settlement area is the yellow polygon. The results of the damage assessment, i.e., the disappeared dwellings, are the red circles (image © 2016 DigitalGlobe).

To calculate each sub-object's difference in change to their local reference values (see Section 2.2.3), we developed a Python script. The script uses the super-object IDs to determine sub-objects within the same settlement area and computes the local reference within each settlement individually. It then calculates the difference of each sub-object to the corresponding local reference. For the unsupervised clustering, another Python script was developed because the required algorithm is not available in the QGIS modeler. It reads attribute values of image objects from shapefiles and performs unsupervised clustering of those attributes using the k-means algorithm from the SciPy library [60].

Using the graphical modeler, all processes were combined and saved as a model, which only needs the pre- and post-conflict images as input. Optionally, the thresholds for the settlement detection and the change analysis can be defined (otherwise, the model uses default values). It returns three shapefiles of the three results explained in Section 2.2 as output. Figure 3 shows a screenshot of the modeler view of the change analysis, with annotations of analysis steps and used software packages. A detailed list of all FOSS packages and algorithms used, as well as their function within the workflow can be found in Table A2.

2.3.2. Workspace Preparation

The user workspace comprises the directories and files shown in Listing 1. They are stored in a specific directory structure, so that the model executor can find them. The contents of the workspace are:

- a subdirectory `data` with the two georeferenced data files in TIFF format
- a Python script file, `model.py`, calling the actual model using the QGIS Python API (Application Programming Interface, based on [61])
- analogous to the QGIS models and scripts directories, a `models` and a `scripts` directory containing `.model` (for a visual summary, see Figures 3 and A2, respectively) and Python files

Listing 1: Excerpt of workspace directory tree; the full workspace is available on GitHub [62] and in the reproducibility package, see Section 3.4.

```
/workspace
|-- data
| |-- COPYRIGHT
| |-- jonjona_pos_conflict_proj.tif
| `-- jonjona_pre_conflict_proj.tif
|-- model.py
|-- models
| |-- detect_settlements_on_edgelayer.model
| `-- example_analysis_linux_v3.1.model
`-- scripts
  |-- diff_to_local_ref_v1.3.py
  `-- kmeans_clustering_v2.3.py
```

2.3.3. Containerization of the Workspace and Runtime Environment

The prepared workspace is packaged in an executable container using a tool originally developed for DevOps (cf. [63]) called Docker (http://docker.io). It provides lightweight virtualization and process separation to package an application and its dependencies, for example for scalable deployment in cloud infrastructures. We use a Docker image to encapsulate the GEOBIA workflow with a well-defined software environment. The image can be executed anywhere where a Docker host environment is running, including Linux, Windows and OSX (https://docs.docker.com/engine/installation/).

For execution, a container is started based on an image. A container can be paused, stopped and restarted or be removed from the host. The image is built from a human- and machine-readable definition of the complete environment called `Dockerfile`. This "recipe" allows a scripted definition of Docker images, i.e., installation and configuration of contained software and files, and consequently, a repeatable building of a runtime environment. `Dockerfiles` can start from scratch or a base image and contain arbitrary textual metadata using labels. Image layering allows one to re-use well-vetted images, for example a base image with all typical OBIA software, across projects. The data and specific tools or configuration are added to project-specific images, which can override files and environments of base images.

While not being intended for it, Docker is a means to ensure long-term reproducibility of computational research, as demonstrated for example for R [64]. A Docker image suffices to capture the data, software and runtime environment in a well-defined manner and facilitates reproducibility. For reproduction, a user must only have the concrete project's Docker image. It can be downloaded from an image repository or loaded from a file.

Docker images of software used in our workflow have been published on Docker Hub (for example Todd Stavish's QGIS [65] and OTB [66] images; the Kartoza image for QGIS [67]), but because these execute a GUI by default and do not provide complete control over each software's version, we created our own set of images to run standalone models. Our Dockerfiles are published on GitHub [62] and the corresponding images on Docker Hub (https://hub.docker.com/r/nuest/qgis-model).

Listing 2: Excerpt from the base image Dockerfile. For brevity and illustration, environment variables and commands are shortened.

```
FROM ubuntu:16.04

RUN apt-get update \
  && apt-get install -qqy --no-install-recommends gdal-bin qgis=2.8.6+dfsg-1build1

RUN wget http://[..].sourceforge.net/[..]/saga_2.2.0.tar.gz \
  && tar -xvzf saga*.tar.gz
RUN ./configure && make make install
RUN wget https://www.orfeo-toolbox.org/[...]/OTB-5.6.1-Linux64.run -q \
  && ./OTB-5.6.1-Linux64.run

ENV PYTHONPATH=/usr/share/qgis/python:/usr/share/qgis/python/plugins
ENV QGIS_WORKSPACE=/workspace
ENV QGIS_MODELFILE=/workspace/models/*.model
ENV QGIS_MODELSCRIPT=/workspace/model.py
ENV QGIS_RESULT=/results
ENV QGIS_USER_MODELDIR=/root/.qgis2/processing/models

WORKDIR /qgis
COPY model.sh model.sh
RUN chmod 0755 model.sh

VOLUME $QGIS_WORKSPACE
VOLUME $QGIS_RESULT

ENTRYPOINT ["/bin/bash", "/qgis/model.sh"]
```

In our specific case, the base image `ubuntu/Dockerfile.xenial` (see Listing 2) installs the required software, sets environment variables and configures the container's default command. The installation commands rely on software packages from the Ubuntu repositories and source installations for SAGA and OTB (SAGA is installed from the source in a specific version not available in the repositories to solve compatibility issues with QGIS; see http://hub.qgis.org/issues/13279 for details). Environment variables provide a single point of configuration. The default command is run with a Bash shell (see [68]). The project Dockerfile `workspace/rs-jonjona/Dockerfile` (see Listing 3) extends the base image by copying the workspace data into the container and by defining an image label with the information about configurable workflow options.

Listing 3: Project Dockerfile.

```
FROM nuest/qgis-model:xenial-multimodel
COPY . /workspace
LABEL de.ifgi.qgis-model.options '[ \
  [...]
  { "id": "change_analysis_threshold", \
   "name": "change sensitivity", \
   "value": "0.3", \
   "comment": "minimum change in edge intensity for objects
        to be flagged as changed" }]'
```

Figure 5 shows the complete control flow in the container. Excerpts from the core files `model.sh` and `model.py` are shown in Listings 4 and 5 respectively (using XVFB [69] for a virtual frame buffer because the container does not have a physical display, but QGIS needs a display even if not used; mounting the hosts physical display would be possible on desktop computers, but not in cloud environments).

Listing 4: Excerpt from `model.sh`; utility code left out for brevity.

```
cp /workspace/models/*.model /root/.qgis2/processing/models
cp /workspace/scripts/*.py /root/.qgis2/processing/scripts
xvfb-run python /workspace/model.py
```

1) `docker run` starts a container and executes the entry point script `/qgis/model.sh` using a Bash shell
2) `/qgis/model.sh` ...
    a) copies model and script files
    from `/workspace/models/*` to `/root/.qgis2/processing/models`
    from `/workspace/scripts/*` to `/root/.qgis2/processing/sripts`
    b) executes `model.py` as a Python file with a virtual frame buffer
3) `/workspace/model.py` ...
    a) initiates QGIS application
    b) loads manipulation parameters and construct input and output paths
    c) runs the model `example_analysis_linux_v3.1.model` using the QGIS Python API passing configuration parameters
4) `/root/.qgis/processing/models/example_analysis_linux_v3.1.model` ...
    a) executes the model steps, using user scripts from `/root/.qgis/processing/scripts`
    b) saves the files to the result directory
5) `/results` holds the output files for user access

Figure 5. Control flow during an execution of the Docker container in a list of numbered steps. Starting from the command `docker run`, the control flow goes through two script files, one in Bash and one in Python, each in turn acting on other files and being configured using environment variables. Supplementary steps such as logging or loading libraries are omitted for brevity. At the end, the output files are available in a pre-defined directory.

Listing 5: Excerpt from `model.py`; command construction based on environment variables and utility code left out for brevity.

```
app = QgsApplication([], True)
QgsApplication.initQgis()
Processing.initialize()
import processing
processing.runalg("modeler:example_analysis_linux_v3.1", # qgis_model_name
  "/workspace/data/jonjona_pre_conflict_proj.tif",  # inputimage_pre
  "/workspace/data/jonjona_pos_conflict_proj.tif",  # inputimage_post
  0.3,                      # change_analysis_threshold
  0.3,                      # settlement_threshold
  0,                       # settlement_size
  "/results/settlements.shp",           # output_settlements
  "/results/result_threshold.shp",          # output_result_threshold
  "/results/result_unclassified.shp")         # output_result_unclassified
```

2.4. InterIMAGE-Based Analysis

InterIMAGE is another candidate for a FOSS-based OBIA workflow. It provides different segmentation algorithms including the widely-used multiresolution segmentation [70], and operators for calculation of attributes, such as shape, texture or topological characteristics [28]. A so-called batch mode feature has been available since Version 1.39. It allows the automatic execution of InterIMAGE interpretation projects, so-called semantic networks. The networks store the classes and operators to be executed.

We were able to demonstrate running the user interface of the latest available Linux release (1.27) in a Docker container by sharing a local X11 socket [71]. However, several issues hinder the implementation of the use case. Firstly, the software focuses on image interpretation, and not all required algorithms for processing the image layers (e.g., the edge detection) are available in the basic package. A combination with other tools is possible to add missing functionality (see, e.g., [72,73]), but it is unclear how to achieve that in a scripted workflow. More importantly, the latest available download for Linux is outdated (Version 1.27; see http://www.lvc.ele.puc-rio.br/projects/interimage/download). We were not successful in compiling a later version of the source code for Linux as part of this work due to a lack of documentation and community support (https://groups.google.com/forum/#!topic/interimage/924t-uZrAMs).

While Linux is currently the main operating system for both Docker containers and hosts, support for multi-platform containers exists and is developed further (see [74] for information on Windows containers). Linux containers can be executed in a native Docker for Windows application for recent Windows versions with Hyper-V technology (see https://docs.docker.com/docker-for-windows/). Therefore, Windows-based containers for InterIMAGE will be possible in the future, although the question of licensing is not answered yet.

3. Results

3.1. Running the Container: Command Line Interface

The container can be executed on any computer with Docker. The image with the analysis is published on Docker Hub. Only the first command shown in Listing 6 is required to run the container and reproduce the analysis, because Docker downloads images automatically from Docker Hub. The configuration options enable console output and name the container for later reference. The log (see [75] for a full log) comprises all installed software and their versions, the configured parameters and the output of the started processes.

Listing 6: Full reproduction commands: run the container from Docker Hub and extract the result.

```
docker run -it --name repro nuest/qgis-model:rs-jonjona
docker cp repro:/workspace/results /tmp/repro_results
```

The second command copies the output of the workflow to a directory of the host computer. Listing 7 shows the contents: a directory with a timestamp of the current execution with three shapefiles, the actual model output. The shapefiles can now be inspected or processed further. Figure 4 shows a visualization of the files `result_threshold.shp` and `settlements.shp`.

Listing 7: Result directory tree after execution, supplementary shapefile files, i.e., .dbf, .prj, .qpj, and .shx, and workspace files (see previous Listing 1) not shown.

```
|/result
|‘-- 20161212-172947
|   |-- result_threshold.shp
|   |-- result_unclassified.shp
|   |-- settlements.shp
```

The image can be used to apply the same analysis to another use case. Listing 8 shows the exchange of data (mounting a different workspace) and parameter manipulation (changing the environment variable).

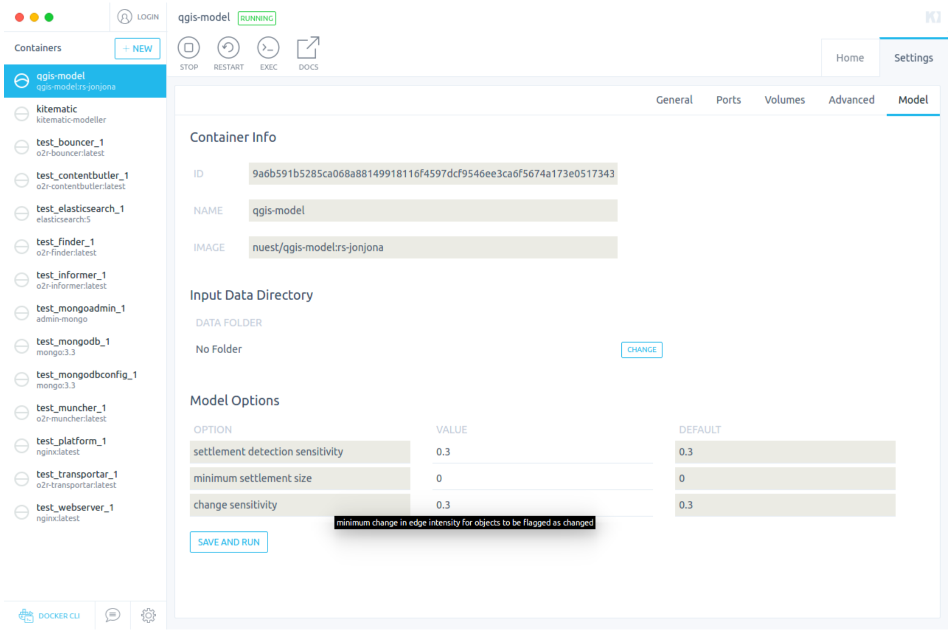
Listing 8: Analysis control and data switching examples. From top to bottom: (a) mounting another workspace; (b) mounting only input files; (c) changing model options via environment variables.

```
# (a)
docker run -it -v /my/analysis:/workspace nuest/qgis-model:rs-jonjona

# (b)
docker run -it -v mypreconflict.tif:/workspace/data/pre_conflict.tif
  -v mypostconflict.tif:/workspace/data/pos_conflict.tif nuest/qgis-model:rs-jonjona

# (c)
docker run -it -e change_analysis_threshold=0.28 nuest/qgis-model:rs-jonjona
```

3.2. Running the Container: Graphical User Interface

As mentioned in Section 1.4, the transferability of rule sets and analysis models is an important aspect in GEOBIA, but some parameters usually need to be adapted to tune an analysis method to a specific study area. In terms of practical application of GEOBIA, a flexible parameterization of analysis workflows by the users must be possible after containerization. Input data need to be easily interchangeable to enable the transfer of a reproducible analysis method to the study area at hand.

To reconcile the issues of reproducibility of and interaction with OBIA workflows, we extended Kitematic (https://kitematic.com/), a FOSS project for managing and running Docker containers with a GUI, with model control functions. We forked the project (https://github.com/nuest/kitematic/tree/model-ui) and added a simple form for controlling the options of workflows (see Figure 6), which hides the complexity of manipulation using environment variables. Instead, the user configures settings (i.e., images to compute the analysis on or thresholds used in the workflow) through fields and buttons. The container's full output log is also readily available.



qgis-model RUNNING

Containers	
qgis-model	qgis-model:rs-jonjona
kitematic	kitematic-modeller
test_bouncer_1	o2r-bouncer:latest
test_contentbutler_1	o2r-contentbutler:latest
test_elasticsearch_1	elasticsearch:5
test_finder_1	o2r-finder:latest
test_informer_1	o2r-informer:latest
test_mongoadmin_1	admin-mongo
test_mongodb_1	mongo:3.3
test_mongodbconfig_1	mongo:3.3
test_muncher_1	o2r-muncher:latest
test_platform_1	nginx:latest
test_transportar_1	o2r-transportar:latest
test_webserver_1	nginx:latest

Container Info

ID	9a6b591b5285ca068a88149918116f4597dcf9546ee3ca6f5674a173e0517343
NAME	qgis-model
IMAGE	nuest/qgis-model:rs-jonjona

Input Data Directory

DATA FOLDER

No Folder CHANGE

Model Options

OPTION	VALUE	DEFAULT
settlement detection sensitivity	0.3	0.3
minimum settlement size	0	0
change sensitivity	0.3	0.3

minimum change in edge intensity for objects to be flagged as changed

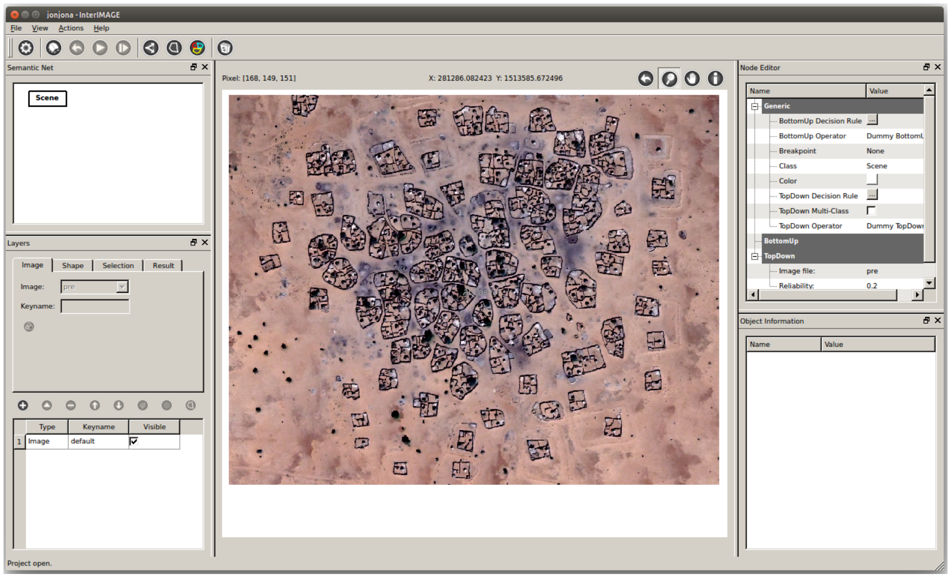
SAVE AND RUN

Figure 6. Screenshot of extended Kitematic software. The left-hand side lists locally available containers and the currently-selected one is highlighted in blue. At the top, buttons control the container state. Two nested levels of tabs show information on and allow configuration of the selected container. The tab "model" is active and was developed as part of this work. Its contents in the central area of the UI provide a form-based user interface for controlling parameterized GEOBIA workflows. The list of model options (bottom) shows the option name, current value and default value. A pop-up displays an information text for the third option as the cursor hovers over that line. A "Save and run" button at the bottom can be used to restart the analysis with the changed parameters.

Users can access the result files by mapping the volume where the container stores the results to a directory on the host. Another volume allows one to exchange the whole workspace, i.e., input data and model. Hence, the graphical user interface allows the same level or manipulation as command line options.

3.3. Running InterIMAGE inside Container

Figure 7 shows a screenshot of the InterIMAGE GUI running inside the container. As stated above, it was not possible to run a fully-automated workflow by use of a script as in Section 2.3. Instead, this example shows a different level of integration, which is to run the user interface of the software inside a container without triggering any predefined models. This provides the users with the full capabilities of that software for building their own analyses without the need to recreate the complete runtime environment.



	Type	Keyname	Visible
1	Image	default	✓

Name	Value
Generic	
BottomUp Decision Rule	...
BottomUp Operator	Dummy BottomU
Breakpoint	None
Class	Scene
Color	
TopDown Decision Rule	...
TopDown Multi-Class	☐
TopDown Operator	Dummy TopDowr
BottomUp	
TopDown	
Image file:	pre
Reliability:	0.2

Figure 7. Screenshot of InterIMAGEUI. The software was started with a shared X server using the command `xhost + && docker run -it -rm -v /tmp/.X11-unix:/tmp/.X11-unix -e DISPLAY=unix$DISPLAY -e uid=$(id -u) -e gi d=$(id -g) -v /data:/data nuest/docker-interimage:1.27 ./interimage`. The command mounts the display and configures the user within the container to be the same as the executing user. It also mounts a data directory containing the workspace, of which one input image is displayed in the central area of the UI.

3.4. Reproducibility Package

In the spirit of reproducibility and being well-aware of the ephemeral nature of online platforms such as GitHub or Docker Hub, we provide a comprehensive reproducibility package with this work. It contains software, data, Docker images and `Dockerfiles`, documentation of how it was created and instructions on how to run it. It is published at Zenodo, a repository for long-term preservation [75]. The software comprises a current version of Docker and the developed software. Both are included as installers for common operating systems and as source code.

3.5. Reproducible GEOBIA

Three observations can be made for reproducibility in the GEOBIA domain. First, the existing reproducibility spectrum [13] does not represent typical GEOBIA analyzes well because of the limited availability of open data and the dominance of a single commercial software. Since many researchers in the domain actually have access to a de facto standard software, most of them could reproduce an open workspace, although the software is not FOSS. Consequently open-sourcing of the workspace could be distinguished from open-sourcing the used software.

Second, acquisition of remote sensing data is costly, and often, there are no free suitable alternatives. Consequently, one could accept data not being open, because it is readily available for anyone having the financial resources.

Third, no specific guidelines for authors of (GE)OBIA papers exist comparable to the examples from other domains (see Section 1.2). Such guidelines can comprise the aspects documentation, scripted workflows, best practices for project structures, freeware (free as in "free beer"; see [76]) and

FOSS and open data. The former two are already common, yet the latter could have a high impact on reproducibility and practical adoption.

4. Discussion

We successfully demonstrate packaging a complete GEOBIA workflow using FOSS. The package created is transferable between machines (different host operating systems, as well as desktop and cloud platforms), and all tools are available free of charge. This is the first time environment variables and Docker image labels are used to parameterize a scientific workflow with a GUI. Our experiments show that containerization is useful not only for reproducibility by third parties, but also for the original development of a FOSS-based analysis, because of the numerous tools involved in different versions and potential conflicts between them. The customized user interface removes barriers for practitioners with limited computer science experience. However, this only concerns the use of a containerized method. For authors of new methods, the creation of a container requires knowledge in the area of Docker and the QGIS Python API.

A Docker container is not a black box, since each applied software is documented in detail in the Docker file. The presented solution also allows one to customize a method by changing input parameters and data. In our example, we only enable three parameters to be manipulated, but the approach can accommodate any number of additional variables (e.g., segmentation parameters). It is also possible to develop more complex containers allowing users to choose different algorithms. We thereby present a means to technically reconcile the conflicting priorities of customizability and reproducibility. However, the conceptual question on the desired degree of customizability, i.e., to what extent the target group of practitioners is expected to redevelop a methodology provided by an expert, remains open and strongly depends on the specific use case.

Our work also reveals challenges with regard to the overarching goal of reproducible research. The presented solution is arguably a one-off effort to containerize a specific workflow and does not require any standardization beyond the `docker run` command. Yet, only an experienced developer can trace the complete flow of information, from the Docker entry point via used scripts to the actually executed code and used parameters, to grasp the complete picture. The user scripts and analysis model are embedded in the container, and extracting them requires the container to be started. This could be a barrier for users and for the purposes of development and exploration, but a copy could be kept outside the image or be made available by reproducibility tools. Keeping a copy outside the image naturally leads to a nested packaging approach.

The selection of Docker as the container engine makes it crucial for both reproduction and archiving. This dependency on a specific product is mitigated by Docker being open-source and highly adopted in the IT industry. An open standardization effort also is underway: the Open Container Initiative (https://www.opencontainers.org/). This effort contributes to a proper long-term archiving solution because the runtime environment must be preserved, and archiving cannot rely solely on the `Dockerfile` without replicating all source repositories or download sites. Layers of images, i.e., base images and an analysis image, make it possible to easily store, share, adopt and collaborate on complex analyses because they can be shared or extended further. However, they also increase complexity.

The approach for UI-based GEOBIA within containers (see Section 3.3) using the current tools is only possible on Unix-based operating systems. Interactive interfaces for containerized workflows are possible in an OS-independent manner by developing the whole analysis within a container, which provides a user interface via HTTP and HTML to a regular web browser (cf. [77]). The advantages are complete and consistent containers and immediate visual access to results. However, we suspect that most users prefer to develop an analysis in the environment they are used to and only package a complete analysis.

Besides the technical challenges, best practices for reproducible research could provide a meaningful yet generic workspace structure and enforce general practices, such as managing scripts in a version control system [78]. Such practices could be implemented in a standardized format for container-based reproducibility packages, be supported by ready-to-use templates and even be partially automated for example with an "export to container"-button in the QGIS workflow modeler to generate a `Dockerfile`. The presented solution can accommodate such best practices. The formal specification of a container format and supporting services for semi-automatic creation are subject of current research and can mitigate the above-mentioned knowledge requirements for the authors of methods (cf. [79]).

The availability of open data remains a general issue. Especially in GEOBIA, where very high resolution imagery plays an important role in many analyses, the applied images are often not freely available. In these cases, it is not possible to publish the data along with the analysis workflow and software. On the other hand, GEOBIA is a widely-used tool in the rapidly-growing field of analyzing very high-resolution data, e.g., from unmanned aerial systems (UAS). Here, scientists become producers of their own image data. This makes an approach as presented here especially useful because the studied images can be published and at the same time become an essential requirement for full reproducibility due to their uniqueness.

The FOSS solutions applied in the containerized workflow, at the current stage, cannot compete with commercial software packages, such as eCognition, regarding functionality and data models for GEOBIA. The available functions of FOSS tools already provide a substantial set of algorithms, and the analysis is created with a user-friendly interactive modeler in a Desktop environment. However, the number of actual OBIA operations for image interpretation is limited. Our example analysis shows that many aspects of GEOBIA can already be realized, e.g., by a combination of algorithms and tailored scripts. However, more complex models, including iterative sequences of segmentation, merging and interpretation of objects (e.g., for a better extraction of relevant dwelling structures, cf. [46,80]) are still difficult to develop in FOSS. However, since FOSS tools are easily extensible, the missing functionality can be contributed as new functions or independent tools.

5. Conclusions

Docker containers and a combination of established free and open-source GIS and image analysis software enable reproducible GEOBIA. We build and distribute a container to carry all required software and data in a transparent manner. The provided user interface makes the package easy to use. This is a breakthrough for creating a transferable and executable package of a GEOBIA workflow. The presented analysis goes well beyond simple processing by successfully integrating tools into a complex multi-step analysis. Packaging GEOBIA software and workflows opens new possibilities for reviews of scientific work, collaboration between researchers and adoption by practitioners. The example analysis in conflict damage assessment presents an application field where transparency and cost are important factors, so that an open approach is advantageous. The shortcomings with respect to the reproducibility of analyses are mostly related to usability. To reach a comprehensive feature set, high user-friendliness and subsequently practical adoption, a community of GEOBIA users applying and contributing to open-source technologies is needed. Although there are commonalities across all scientific disciplines, domain-specific requirements demand: (i) targeted education; (ii) high-quality specialized FOSS; and (iii) best practices. The challenge starts with an open discourse on reproducible research and a working definition of reproducibility specifically for GEOBIA (cf. [81]), to which this work intends to be a first step.

Acknowledgments: This research has been conducted in the context of the Graduate School for Geoinformatics (http://www.uni-muenster.de/Geoinformatics/en/Studies/study_programs/PhD/). It has partly been supported by the project Opening Reproducible Research (http://o2r.info and https://www.uni-muenster.de/forschungaz/project/9520) funded by the German Research Foundation (DFG) under Project Number PE 1632/10-1. We thank Edzer Pebesma for valuable comments and support.

Author Contributions: Both authors contributed equally to the paper. Christian Knoth conceived the study and developed the analysis workflow and its FOSS-based implementation. Daniel Nüst performed the containerization of the workflow and runtime environment and implemented the user interface for the interaction with the containerized workflow.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

MDPI	Multidisciplinary Digital Publishing Institute
GIS	Geographic Information System
GEOBIA	Geographic Object-Based Image Analysis
OBIA	Object-Based Image Analysis
FOSS	Free and Open-Source Software
LULC	Land Use and Land Cover
LIDAR	Light Detection and Ranging
GUI	Graphical User Interface
AAAS	American Association for the Advancement of Science
PCA	Principal Component Analysis
OTB	Orfeo ToolBox
SAGA	System for Automated Geoscientific Analyses
API	Application Programming Interface
XVFB	X Window Virtual Frame Buffer
UAS	Unmanned Aerial Systems
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language

Appendix A

Table A1. List of properties used in the settlement detection and the subsequent detection of disappeared structures (within detected settlement areas) along with the corresponding rules and recommended thresholds.

Object Property	**Rule or Threshold**	**Analysis Step**
Standard deviation of edge layer (pre-conflict) of seed segments	≥ 0.3	Settlement detection
Proximity of seed segments to each other	≤ 100 m	Settlement detection
Number of seed segments (per settlement)	≥ 2	Settlement detection
Optionally: Size of settlement area (after merging of seeds)	≥ 0 (no default threshold set in this example)	Settlement detection
Existence of super-object of class settlement	True (super-object ID > 0)	Change analysis
Change of edge intensity	Difference to local reference value ≥ 0.33	Change analysis
Minimum size (area)	10 m^2	Change analysis
Maximum size (area)	60 m^2	Change analysis
Shape Index value	≤ 1.55	Change analysis
Impact of morphological closing (ratio of standard deviation of pre-conflict layer values per object before and after morphological closing)	≤ 5.5	Change analysis

Table A2. Summary of the QGIS-based analysis workflow showing the processing steps and the corresponding algorithms (Python: the scripts written in Python; see Section 2.3.1).

Analysis Step	**Algorithm**	**Stage of Workflow**
Extract first principal component of pre- and post-conflict image	OTB:DimensionalityReduction (pca)	Image processing
Rescale both principal components to 8bit	OTB:Rescale Image	Image processing
Edge detection on both layers	OTB:EdgeExtraction (touzi)	Image processing
Morphological closing on pre-conflict layer	OTB:GrayScaleMorphologicalOperation (closing)	Image processing
Determine extent of raster layer	QGIS:Raster layer bounds	Settlement detection
Create chessboard segmentation within extent	QGIS:Create grid	Settlement detection
Compute standard deviation of edge layer within segments	QGIS:Zonal statistics	Settlement detection
Extract settlement candidate segments according to standard deviation of edge layer	QGIS:Extract by attribute	Settlement detection
Create settlement area objects by growing and merging candidate segments that are within proximity (100 m max) to each other (ignore isolated candidates)	QGIS: Fixed distance buffer QGIS:Multipart to singleparts SAGA:Polygon shape indices QGIS:Extract by attribute QGIS:Fill holes	Settlement detection
Create IDs in attribute table and specify field name	QGIS:Add autoincremental field QGIS:Refactor fields	Settlement detection
Create objects on level of single huts	OTB:Segmentation (watershed)	Change analysis
Compute mean of edge intensity within objects (pre- and post-conflict)	QGIS:Zonal statistics	Change analysis
Calculate difference in mean edge density between pre- and post-conflict (check for NULL)	QGIS:Adv. Python Field Calculator QGIS:Extract by attribute	Change analysis
Compute shape and size properties of objects	SAGA:Polygon shape indices	Change analysis
For all sub-objects, get IDs of containing super-objects (settlements)	SAGA:Identity	Change analysis
Compute local reference (of change) within settlements and difference of sub-objects to this reference	Python:Difference to local reference v1.3	Change analysis
Compute unsupervised clustering regarding change	Python:Kmeans clustering v2.3	Change analysis
Extract objects by minimum and maximum size	QGIS:Extract by attribute	Change analysis
Extract objects by their shape index	QGIS:Extract by attribute	Change analysis
Compute statistics of pre-conflict layer per object before and after morphological closing	QGIS:Zonal statistics	Change analysis
Calculate ratio of sdev. values of pre-conflict layer before and after morphological closing	QGIS:Refactor fields	Change analysis
Extract objects by ratio value	QGIS:Extract by attribute	Change analysis
Extract objects by change value (difference in mean edge density) using pre-defined threshold	QGIS:Extract by attribute	Change analysis
Compute centroids of objects extracted by threshold and within settlements	QGIS:Polygon centroids QGIS:Extract by attribute	Change analysis

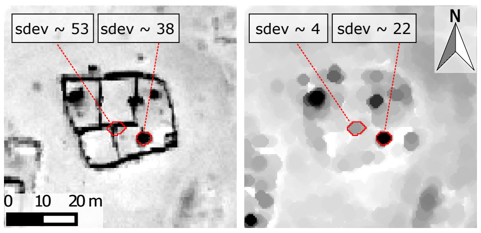
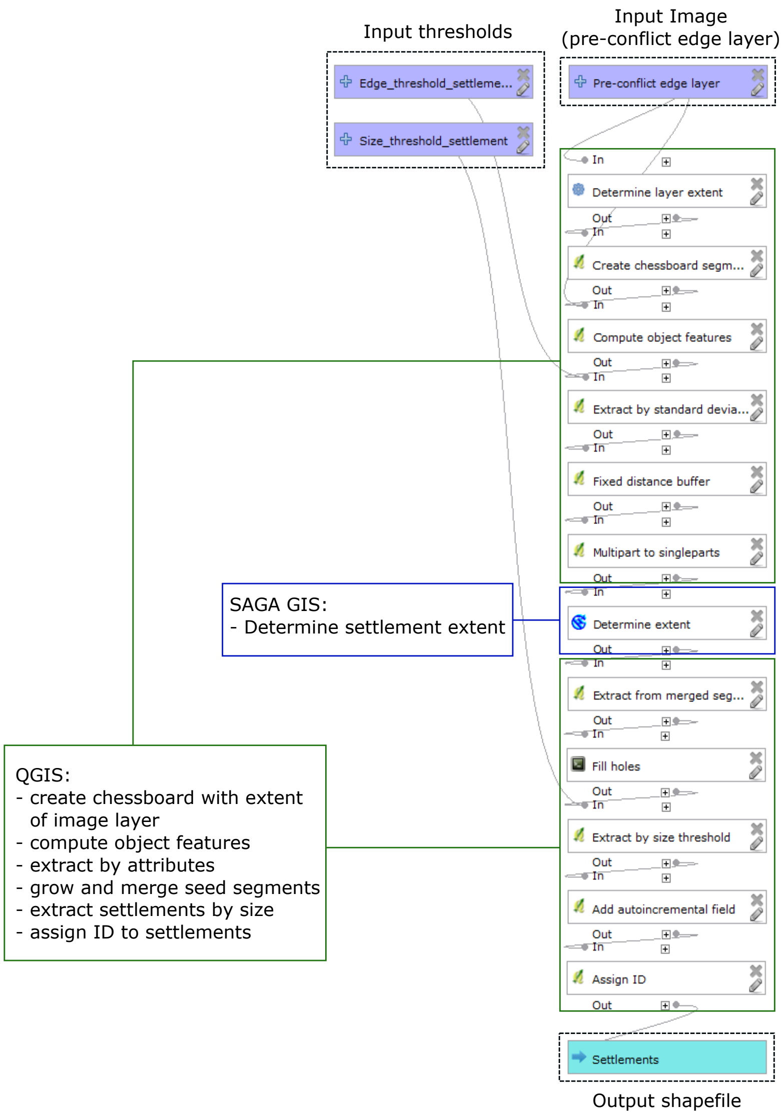


Figure A1. Subset of the pre-conflict layer before (left image) and after (right image) morphological closing. It shows the effect of the filter on a dwelling object (right object) and a fence (left object). The higher impact of the filter on small, linear structures is used as an additional feature to remove them by measuring the ratio of the standard deviation per object of the unfiltered to that of the filtered layer.



Input thresholds
Input Image (pre-conflict edge layer)
Edge_threshold_settleme...
Size_threshold_settlement
Pre-conflict edge layer
In
Determine layer extent
Out
Create chessboard segm...
Compute object features
Extract by standard devia...
Fixed distance buffer
Multipart to singleparts
SAGA GIS:
- Determine settlement extent
Determine extent
Extract from merged seg...
Fill holes
Extract by size threshold
Add autoincremental field
Assign ID
QGIS:
- create chessboard with extent of image layer
- compute object features
- extract by attributes
- grow and merge seed segments
- extract settlements by size
- assign ID to settlements
Settlements
Output shapefile

Figure A2. Screen-shot of analysis workflow in the QGIS graphical modeler with highlighting. Three inputs, two numerical thresholds and the pre-conflict edge layer are shown in purple at the top. The analysis steps are connected with grey arcs and executed from top to bottom. They are based on QGIS (green boxes) and SAGA GIS (dark blue box). This model is applied as one algorithm in the analysis workflow depicted in Figure 3. The model output is a single shape file with detected settlements, shown in turquoise at the bottom.

References

1. Bailey, C.W. What Is Open Access? Available online: http://digital-scholarship.org/cwb/WhatIsOA.htm (accessed on 19 December 2016).
2. European Commission Horizon 2020 Open Science (Open Access). Available online: https://ec.europa.eu/programmes/horizon2020/en/h2020-section/open-science-open-access (accessed on 19 December 2016).
3. The Commission High Level Expert Group on the European Open Science Cloud. *Realising the European Open Science Cloud*; European Commission: Brussels, Belgium, 2016.
4. Nosek, B.A.; Alter, G.; Banks, G.C.; Borsboom, D.; Bowman, S.D.; Breckler, S.J.; Buck, S.; Chambers, C.D.; Chin, G.; Christensen, G.; et al. Promoting an open research culture. *Science* **2015**, *348*, 1422–1425.
5. Peng, R.D. Reproducible research and Biostatistics. *Biostatistics* **2009**, *10*, 405–408.
6. Markowetz, F. Five selfish reasons to work reproducibly. *Genome Biol.* **2015**, *16*, 274.
7. Kraker, P.; Dörler, D.; Ferus, A.; Gutounig, R.; Heigl, F.; Kaier, C.; Rieck, K.; Šimukovič, E.; Vignoli, M.; Aspöck, E.; et al. The Vienna Principles: A Vision for Scholarly Communication in the 21st Century. *Mitteilungen der Vereinigung Österreichischer Bibliothekarinnen und Bibliothekare* **2016**, *3*, 436–446.
8. Sandve, G.K.; Nekrutenko, A.; Taylor, J.; Hovig, E. Ten Simple Rules for Reproducible Computational Research. *PLoS Comput. Biol.* **2013**, *9*, e1003285.
9. Gentleman, R.; Temple Lang, D. Statistical Analyses and Reproducible Research. *J. Comput. Graph. Stat.* **2007**, *16*, 1–23.
10. Goodman, S.N.; Fanelli, D.; Ioannidis, J.P.A. What does research reproducibility mean? *Sci. Transl. Med.* **2016**, *8*, 341ps12.
11. Amitrano, D.; Di Martino, G.; Iodice, A.; Riccio, D.; Ruello, G. A New Framework for SAR Multitemporal Data RGB Representation: Rationale and Products. *IEEE Trans. Geosci. Remote Sens.* **2015**, *53*, 117–133.
12. Howe, B. Virtual Appliances, Cloud Computing, and Reproducible Research. *Comput. Sci. Eng.* **2012**, *14*, 36–41.
13. Peng, R.D. Reproducible Research in Computational Science. *Science* **2011**, *334*, 1226–1227.
14. Lucieer, V.; Hill, N.A.; Barrett, N.S.; Nichol, S. Do marine substrates 'look' and 'sound' the same? Supervised classification of multibeam acoustic data using autonomous underwater vehicle images. *Estuar. Coast. Shelf Sci.* **2013**, *117*, 94–106.
15. Dupuy, S.; Barbe, E.; Balestrat, M. An Object-Based Image Analysis Method for Monitoring Land Conversion by Artificial Sprawl Use of RapidEye and IRS Data. *Remote Sens.* **2012**, *4*, 404–423.
16. Tormos, T.; Dupuy, S.; Van Looy, K.; Barbe, E.; Kosuth, P. An OBIA for fine-scale land cover spatial analysis over broad territories: Demonstration through riparian corridor and artificial sprawl studies in France. In Proceedings of the GEOBIA 2012: 4th International Conference on GEographic Object-Based Image Analysis, Rio de Janeiro, Brazil, 7–9 May 2012.
17. Stodden, V.; Miguez, S.; Seiler, J. ResearchCompendia.org: Cyberinfrastructure for Reproducibility and Collaboration in Computational Science. *Comput. Sci. Eng.* **2015**, *17*, 12–19.
18. Bechhofer, S.; Buchan, I.; De Roure, D.; Missier, P.; Ainsworth, J.; Bhagat, J.; Couch, P.; Cruickshank, D.; Delderfield, M.; Dunlop, I.; et al. Why linked data is not enough for scientists. *Future Gen. Comput. Syst.* **2013**, *29*, 599–611.
19. Chirigati, F.; Rampin, R.; Shasha, D.; Freire, J. ReproZip: Computational Reproducibility With Ease. In Proceedings of the SIGMOD 2016 International Conference on Management of Data, San Francisco, CA, USA, 26 June–1 July 2016.
20. Douglas T.; Haiyan Meng, P. Techniques for Preserving Scientific Software Executions: Preserve the Mess or Encourage Cleanliness? In Proceedings of the 12th International Conference on Digital Preservation (iPres) 2015, Chapel Hill, NC, USA, 2–6 November 2016.
21. Open Source Initiative. Open Source Case for Business:Advocacy. Available online: https://opensource.org/advocacy/case_for_business.php (accessed on 19 December 2016).
22. Wightman, T.. What's keeping you from using open source software?. Available online: https://opensource.com/business/13/12/using-open-source-software (accessed on 19 December 2016).
23. Salus, P. *A Quarter-Century of Unix*; Addison-Wesley: Boston, MA, USA, 1994; pp. 52–53.

24. Grippa, T.; Lennert, M.; Beaumont, B.; Vanhuysse, S.; Stephenne, N.; Wolff, E. An Open-Source Semi-Automated Processing Chain for Urban OBIA Classification. In Proceedings of the GEOBIA 2016: Solutions & Synergies, Enschede, The Netherlands, 14–16 September 2016.
25. Böck, S.; Immitzer, M.; Atzberger, C. Automated Segmentation Parameter Selection and Classification of Urban Scenes Using Open-Source Software. In Proceedings of the GEOBIA 2016: Solutions & Synergies, Enschede, The Netherlands, 14–16 September 2016.
26. Van De Kerchove, R.; Hanson, E.; Wolff, E. Comparing pixel-based and object-based classification methodologies for mapping impervious surfaces in Wallonia using ortho-imagery and LIDAR data. In Proceedings of the GEOBIA 2014: Advancements, Trends and Challenges, Thessaloniki, Greece, 22 May 2014.
27. Körting, T.; Fonseca, L.; Câmara, G. GeoDMA—Geographic Data Mining Analyst. *Comput. Geosci.* **2013**, *57*, 133–145.
28. Costa, G.; Feitosa, R.; Fonseca, L.; Oliveira, D.; Ferreira, R.; Castejon, E. Knowledge-based interpretation of remote sensing data with the InterImage system: Major characteristics and recent developments. In Proceedings of the GEOBIA 2010: Geographic Object-Based Image Analysis, Ghent, Belgium, 29 June–2 July 2010.
29. Antunes, R.; Happ, P.; Bias, E.; Brites, R.; Costa, G.; Feitosa, R. An Object-Based Image Interpretation Application on Cloud Computing Infrastructure. In Proceedings of the GEOBIA 2016: Solutions & Synergies, Enschede, The Netherlands, 14–16 September 2016.
30. Blaschke, T.; Hay, G.; Maggi, K.; Lang, S.; Hofmann, P.; Addink, E.; Feitosa, R.; van der Meer, F.; van der Werff, H.; van Coillie, F.; et al. Geographic Object-Based Image Analysis—Towards a new paradigm. *ISPRS J. Photogramm. Remote Sens.* **2014**, *87*, 180–191.
31. Drăguţ, L.; Tiede, D.; Levick, S. ESP: A tool to estimate scale parameter for multiresolution image segmentation of remotely sensed data. *Int. J. Geograph. Inf. Sci.* **2010**, *24*, 859–871.
32. Drăguţ, L.; Csillik, O.; Eisank, C.; Tiede, D. Automated parameterisation for multi-scale image segmentation on multiple layers. *ISPRS J. Photogramm. Remote Sens.* **2014**, *88*, 119–127.
33. Martha, T.; Kerle, N.; van Westen, C. Segment Optimization and Data-Driven Thresholding for Knowledge-Based Landslide Detection by Object-Based Image Analysis. *Int. J. Geograph. Inf. Sci.* **2011**, *49*, 4928–4943.
34. Hofmann, P. Defuzzification Strategies for Fuzzy Classifications of Remote Sensing Data. *Remote Sens.* **2016**, *8*, 467.
35. Kohli, D.; Warwadekar, P.; Kerle, N.; Sliuzas, R.; Stein, A. Transferability of Object-Oriented Image Analysis Methods for Slum Identification. *Remote Sens.* **2013**, *5*, 4209–4228.
36. Hofmann, P.; Blaschke, T.; Strobl, J. Quantifying the robustness of fuzzy rule sets in object-based image analysis. *Int. J. Remote Sens.* **2011**, *32*, 7359–7381.
37. Knoth, C.; Pebesma, E. Detecting destruction in conflict areas in Darfur. In Proceedings of the GEOBIA 2014: Advancements, Trends and Challenges, Thessaloniki, Greece, 22 May 2014.
38. Knoth, C.; Pebesma, E. Detecting dwelling destruction in Darfur through object-based change analysis of very high-resolution imagery. *Int. J. Remote Sens.* **2017**, *38*, 273–295.
39. Tiede, D.; Füreder, P.; Lang, S.; Hölbling, D.; Zeil, P. Automated Analysis of Satellite Imagery to provide Information Products for Humanitarian Relief Operations in Refugee Camps - from Scientific Development towards Operational Services. *Photogramm. Fernerkund. Geoinf.* **2013**, *2013*, 185–195.
40. Giada, S.; De Groeve, T.; Ehrlich, D.; Soille, P. Information extraction from very high resolution satellite imagery over Lukole refugee camp, Tanzania. *Int. J. Remote Sens.* **2003**, *24*, 4251–4266.
41. Al-Khudhairy, D.; Caravaggi, I.; Giada, S. Structural damage assessments from Ikonos data using change detection, Object-level Segmentation, and Classification Techniques. *Photogramm. Eng. Remote Sens.* **2005**, *71*, 825–837.
42. Witmer, F. Remote sensing of violent conflict: Eyes from above. *Int. J. Remote Sens.* **2015**, *36*, 2326–2352.
43. Wolfinbarger, S. Remote Sensing as a Tool for Human Rights Fact-Finding. In *The Transformation of Human Rights Fact-Finding*; Alston, P.; Knuckey, S., Eds.; Oxford University Press: New York, NY, USA, 2016; pp. 463–477.
44. Knoth, C.; Nüst, D. Enabling reproducible OBIA with open-source software in docker containers. In Proceedings of the GEOBIA 2016: Solutions & Synergies, Enschede, The Netherlands,

14–16 September 2016; Kerle, N., Gerke, M., Lefèvre, S., Eds.; University of Twente Faculty of Geo-Information and Earth Observation (ITC): Enschede, The Netherlands, 2016.
45. American Association for the Advancement of Science Appendix A: Darfur, Sudan and Chad Imagery Characteristics Available online: http://www.aaas.org/page/appendix-darfur-sudan-and-chad-imagery-characteristics (accessed on 19 December 2016).
46. Lang, S.T.D.; Hölbling, D.; Füreder, P.; Zeil, P. Earth observation (EO)-based ex post assessment of internally displaced person (IDP) camp evolution and population dynamics in Zam Zam, Darfur. *Int. J. Remote Sens.* **2010**, *31*, 5709–5731.
47. Wei, Y.; Zhao, Z.; Song, J. Urban building extraction from high-resolution satellite panchromatic image using clustering and edge detection. In Proceedings of the 2004 IEEE International Geoscience and Remote Sensing Symposium, Anchorage, AK, USA, 20–24 September 2004.
48. De Kok, R.; Wezyk, P. Principles of full autonomy in image interpretation. The basic architectural design for a sequential process with image objects. In *Object-Based Image Analysis. Spatial Concepts for Knowledge-Driven Remote Sensing Applications*; Blaschke, T.; Lang, S.; Hay, G.J., Eds.; Springer: Berlin, Germany, 2008; pp. 697–710.
49. Touzi, R.; Lopes, A.; Bousquet, P. A statistical and geometrical edge detector for SAR images. *IEEE Trans. Geosci. Remote Sens.* **1988**, *26*, 764–773.
50. Tewkesbury, A.P.; Comber, A.J.; Tate, N.J.; Lamb, A.; Fisher, P.F. A critical synthesis of remotely sensed optical image change detection techniques. *Remote Sens. Environ.* **2015**, *160*, 1–14.
51. OTB Development Team. The ORFEO Tool Box Software Guide. Available online: https://www.orfeo-toolbox.org//packages/OTBSoftwareGuide.pdf (accesssed on 27 June 2016).
52. Sulik, J.; Edwards, S. Feature extraction for Darfur: Geospatial applications in the documentation of human rights abuses. *Int. J. Remote Sens.* **2010**, *31*, 2521–2533.
53. Lang, S.; Blaschke, T. *Landschaftsanalyse Mit GIS*; Ulmer: Stuttgart, Germany, 2007; pp. 241–243.
54. Forman, R.; Godron, M. *Landscape Ecology*; Wiley: New York, NY, USA, 1986; pp. 106–108.
55. QGIS Development Team. QGIS Geographic Information System. Available online: http://qgis.osgeo.org (accessed on 24 June 2016).
56. Graser, A.; Oyala, V. Processing: A Python Framework for the Seamless Integration of Geoprocessing Tools in QGIS. *ISPRS Int. J. Geo-Inf.* **2015**, *4*, 2219–2245.
57. Rossum, G. Python Reference Manual. Available online: http://www.python.org/ (accessed on 19 December 2016).
58. Inglada, J.; Christophe, E. The Orfeo Toolbox remote sensing image processing software. In Proceedings of the 2009 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), Cape Town, South Africa, 12–17 July 2009.
59. Conrad, O.; Bechtel, B.; Bock, M.; Dietrich, H.; Fischer, E.; Gerlitz, L.; Wehberg, J.; Wichmann, V.; Böhner, J. System for Automated Geoscientific Analyses (SAGA) v. 2.1.4. *Geosci. Model Dev.* **2015**, *8*, 1991–2007.
60. Jones, E.; Oliphant, T.; Peterson, P. SciPy: Open Source Scientific Tools for Python. Available online: http://www.scipy.org/ (accessed on 27 June 2016).
61. PyQGIS Developer Cookbook Using PyQGIS in standalone scripts. Available online: http://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/intro.html#using-pyqgis-in-standalone-scripts (accessed on 27 June 2016).
62. Nüst, D. Docker Container for QGIS Models on GitHub. Available online: https://github.com/nuest/docker-qgis-model (accessed on 24 November 2016).
63. Loukides, M. *What is DevOps? Infrastructure as Code*; O'Reilly Media: Sebastopol, CA, USA, 2012.
64. Boettiger, C. An introduction to Docker for reproducible research, with examples from the R environment. *ACM SIGOPS Oper. Syst. Rev.* **2015**, *49*, 71–79.
65. Stavish, T. Docker for QGIS. Available online: https://hub.docker.com/r/toddstavish/qgis (accessed on 24 November 2016).
66. Stavish, T. Docker for OTB. Available online: https://hub.docker.com/r/toddstavish/orfeo_toolbox (accessed on 24 November 2016).
67. Sutton, T. Docker for QGIS Desktop. Available online: https://hub.docker.com/r/kartoza/qgis-desktop (accessed on 24 November 2016).

68. GNU-Project GNU Bash Available online: https://www.gnu.org/software/bash/ (accessed on 24 November 2016).
69. Wiggins, D.P. XVFB Documentation. Available online: https://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml (accessed on 24 November 2016).
70. Baatz, M.; Schäpe, A. Multiresolution segmentation—An optimization approach for high quality multi-scale image segmentation. In *Angewandte Geographische Informations-Verarbeitung XII*; Strobl, J.; Blaschke, T.; Griesebner, G., Eds.; Wichmann: Karlsruhe, Germany, 2000; pp. 12–23.
71. Nüst, D.; Knoth, C. Docker-Interimage: Running the Latest InterIMAGE Linux Release in A Docker Container with User Interface. Available online: http://zenodo.org/record/55083 (accessed on 6 July 2016).
72. Antunes, R.; Bias, E.; Brites, R.; Costa, G. Integration of Open-Source Tools for Object-Based Monitoring of Urban Targets. In Proceedings of the GEOBIA 2016: Solutions & Synergies, Enschede, The Netherlands, 14–16 September 2016.
73. Passo, D.; Bias, E.; Brites, R.; Costa, G.; Antunes, R. Susceptibility mapping of linear erosion processes using object-based analysis of VHR images. In Proceedings of the GEOBIA 2016: Solutions & Synergies, Enschede, The Netherlands, 14–16 September 2016.
74. Friis, M. Docker on Windows Server 2016 Technical Preview 5. Available online: https://blog.docker.com/2016/04/docker-windows-server-tp5/ (accessed on 19 December 2016)
75. Nüst, D.; Knoth, C. Data and code for: Reproducibility and Practical Adoption of GEOBIA with Open-Source Software in Docker Containers. Available online: https://doi.org/10.5281/zenodo.168370 (accessed on 19 December 2016).
76. GNU-Project What is free software? Available online: https://www.gnu.org/philosophy/free-sw.en.html (accessed on 24 November 2016).
77. Marwick, B. 1989-Excavation-Report-Madjebebe. Available online: https://doi.org/10.6084/m9.figshare.1297059.v2 (accessed on 24 November 2016).
78. Ram, K. Git can facilitate greater reproducibility and increased transparency in science. *Source Code Biol. Med.* **2013**, *8*, 7.
79. Nüst, D.; Konkol, M.; Schutzeichel, M.; Pebesma, E.; Kray, C.; Przibytzin, H.; Lorenz, J. Opening the Publication Process with Executable Research Compendia. *D-Lib Mag.* **2017**, doi:10.1045/january2017-nuest.
80. Tiede, D.; Lang, S.; Hölbling, D.; Füreder, P. Transferability of OBIA rulesets for IDP camp analysis in Darfur. In Proceedings of the GEOBIA 2010: Geographic Object-Based Image Analysis, Ghent, Belgium, 29 June–2 July 2010.
81. Baker, M. Muddled meanings hamper efforts to fix reproducibility crisis. *Nat. News* **2016**, doi:10.1038/nature.2016.20076.

5 A Web service for executable research compendia enables reproducible publications and transparent reviews in geospatial sciences

Authors & contribution Daniel Nüst

Venue Preprinted on Zenodo 10.5281/zenodo.4818120; submission planned to CiSE special track or GigaScience

Date 07/2021

Repository https://zivgitlab.uni-muenster.de/d_nues01/architecture-paper

A Web service for executable research compendia enables reproducible publications and transparent reviews in geospatial sciences

Daniel Nüst[a]

[a]Institute for Geoinformatics (ifgi), University of Münster, Germany (daniel.nuest@uni-muenster.de)

Preprint published on Zenodo at https://doi.org/10.5281/zenodo.4818120 under CC-BY-4.0 license. This version was compiled on 2021-07-08 based on git commit d00c4205 from the repository https://zivgitlab.uni-muenster.de/d_nues01/architecture-paper.

The Executable Research Compendium (ERC) is a concept for packaging data, code, text, and user interface configurations in a single package to increase transparency, reproducibility, and reusability of computational research. This article introduces the ERC reproducibility service (ERS) for a publication workflow enhanced by ERCs. The ERS connects with existing scientific infrastructures and was deployed and tested with a focus on data and visualisation methods for open geospatial sciences. We describe the architecture of a reference implementation for the reproducibility service, including the created Web API. We critically discuss both the project set-up and features of ERC and ERS, and examine them in the light of various classifications for reproducible research. The ERC and ERS are found to be a powerful tool to improve reproducibility and thereby enable better investigating and understanding of computational workflows during peer review. We derive lessons learned and challenges for future scholarly publishing of computer-based geospatial research.

reproducible research | reproducibility | open science | executable research compendium | ERC | research infrastructure | research compendia | containerisation

1. Introduction

Open Science and reproducibility are enormous challenges for research, as computers and algorithms infuse all scientific disciplines, including geography and geosciences (David *et al.*, 2016; Nüst and Pebesma, 2020), and the scientific paper falls short in communicating the actual scholarship (Brammer *et al.*, 2011; Marwick, 2015; Gil *et al.*, 2016). The relevance of openness and reproducible reusable research are undisputed, just as the problems applying them in daily work, challenges around reproducibility, and handling in digital scholarly publishing workflows are real (e.g., Davison, 2012; Freire *et al.*, 2016). Software failures have led to wrong results and retractions (Miller, 2006; Gronenschild *et al.*, 2012) and *"the lack of reported failures from geography and geosciences is not reassuring"* (Nüst and Pebesma, 2020). Reproducibility in geospatial sciences, similar to most scientific disciplines, is low (e.g., Konkol *et al.*, 2019a; Nüst and Pebesma, 2020; Yan *et al.*, 2020; Nüst *et al.*, 2018). Although progress is made on openness in geospatial sciences, reproducibility has not been systematically addressed and increasing requirements for publication has just begun (Minghini *et al.*, 2020; cf. Peng and Hicks, 2021). A continued development of infrastructure supporting reproducibility is needed (Peng and Hicks, 2021). To achieve sufficient openness and reproducibility, all aspects of science must be adopted with the common goal of a culture change in mind. Only a general broad change can support and motivate researchers shift towards good Open Science and reproducible research practices. Examples for areas where change is needed are (a) requirements of funders and journals (cf. Hardwicke *et al.*, 2018, and Stodden *et al.* (2018); Nüst *et al.*, 2018), mechanisms to award recognition to all types of research outputs (Piwowar, 2013), and (c) education and tools, so that all stakeholders have the means, i.e., resources, time, and knowledge, to create, examine, review, and publish reproducible open scientific workflows. To facilitate change on these levels, we have conceptualised and implemented an infrastructure to lower the barriers for creating, sharing, and reviewing reproducible publications. This work's main contribution is a detailed description of that infrastructure and the demonstration of it's functionality.

We present a Web service for open and reproducible publications for computational research in geography and geosciences: the ERC reproducibility service (ERS). The ERS is connected with the existing processes, services, and platforms for scholarly publications and serves the particular needs of geospatial data sciences. Examples and applications are taken from these domains as well, i.e., data-based workflows using observational data of the Earth. The ERS focuses on the third area of cultural change, education and tools, by putting the concept of the Executable Research Compendium (ERC, Nüst *et al.*, 2017) into practice as part of the scholarly publication process. The ERC at the centre of scholarly communication enables communicating, sharing, and collaborating on the actual scholarship as it includes data, software, and documentation (cf. Buckheit and Donoho, 1995; Davenport *et al.*, 2020). Previous work presented the benefits for authors and readers (Konkol *et al.*, 2019b). Here we describe the technical background and implementation of the ERS, and how it provides a missing functionality in scholarly publishing infrastructure.

In the remainder of this work, we first present related initiatives and approaches. Then we introduce a technical specification for the ERC followed by an architecture and reference implementation for a Web service for ERC creation and examination, which is connected with the existing landscape of scholarly publication infrastructures. Finally, we discuss limitations and lessons learned, and conclude with a summary and an outlook on future work.

2. Related work

Containerisation is widely adopted as a technology to capture computing environments around computational workflows in general (Boettiger, 2015), but also more specifically for academic papers (Liu and Salganik, 2019) and for research infrastructures (Konkol *et al.*, 2020). The common drivers behind using containers are the need to *capture* data, code, and the computational environment ideally in an automated fashion, *portability*, e.g., between researcher's computers and cloud infrastructures, and *ease of use*, i.e., abstracting away the complexities of managing the environment from researchers. Workflow tools can automate the process to capture experimental details required for reproducibility (Davison *et al.*, 2014; Wolstencroft *et al.*, 2013), but they are not directly connected with scholarly review and publishing procedures.

The approaches to capture environments are manifold, and once the respective package exists, portability is given. However, the approaches do vary considerable in their usability and accessibility.

Binder (Project Jupyter *et al.*, 2018) uses common configuration and dependency management files from different programming languages as part of its *Reproducible Execution Environment Specification* (REES) specification[1]. The user cannot access the created container specification or image, instead the project promises to consistently create images that remain similar enough over time. In the *Whole Tale* project a related underlying technology is used to create and share reproducible computational research (Chard *et al.*, 2019). The project provides a multi-user platform, which goes well beyond o2r's scope, and uses references to code and data, but core steps are very similar to the ERS, e.g., publishing a *tale* to repositories and interactive examination for reproduction and verification. Tales are published in a format extending DataONE Data Packages (Mecum *et al.*, 2018), which rely on BagIt for serialisation.

ReproZip (Chirigati *et al.*, 2016) is a prominent example for tools that use tracing of system calls to create ReproZip packages, which can be extracted into different environments, e.g., a container. *Umbrella* (Meng and Thain, 2015) is another tracing-based tool with a particular focus on high-performance computing. These solutions, however, are less portable and require the authors to execute them, being overall slightly less accessible than requiring just a Notebook-based workflow. The *Popper* (Jimenez *et al.*, 2017) convention therefore gives authors a lot of flexibility by allowing them to combine software from the DevOps toolbox. The convention provides generic domain-independent templates for project structure, but also require authors to be familiar with a number of complex tools. Chuah *et al.* (2020) bridge between tracing and declarative approaches and also generate `Dockerfiles` for workflows, however, using log files and for C/C++ and Python-based workflows. For the tracing, they use a command-line only tool, *Sciunit* (That *et al.*, 2017), developed by the same group. *Science Capsules* (Ghoshal *et al.*, 2021) and the *Cloud of Reproducible Records* [2] capture end-to-end workflows, but put an emphasis on collaboration and their respective scientific disciplines, and are not connected to scientific publishing. Similarly, *RENKU* is a platform for creating workflows with interlinking of artefacts, like the ERC, yet with a focus on collaboration and providing interactive environments, not with preserving a specific state. *Occam* (Oliveira *et al.*, 2018) focuses on preservation of the full source code to mitigate shortcomings of only saving executable binaries. *Boutiques* (Glatard *et al.*, 2018) is an application description frameworks for packaging CLI tools. The *REANA* platform (imko *et al.*, 2019) enables creation and manipulation of reproducible computational workflows of complex large scale analyses that go beyond the computational Notebooks at the core of the ERC. *Maneage* (Akhlaghi *et al.*, 2021) focuses on the lineage aspect of computational workflows, capturing contributions pre-publication and extensions post-publication relying on GNU Make, requiring familiarity with low-level tools. *Encapsulator* (Pasquier *et al.*, 2018) creates time capsules for reproducible code, capturing the computational environment in a virtual machine using Vagrant (Wikipedia contributors, 2021h). Encapsulator generates a Vagrant file and can be used with a command line interface.

Earlier approaches similar to the ERC and ERS include *Paper Mâché* (Brammer *et al.*, 2011), which uses virtual machines for capturing papers and defines a format quite similar to the ERC, the *Paper Mâché file* (`.pm`). This file can be inspected using an online workbench or downloaded and executed on a local computer. The main differences are the capturing of reviewer comments and ratings within the `.pm` file, the use of VMs, and *Inkling* (Castleberry *et al.*, 2013) has own file formats for documents and workflow configuration based on LaTeX and creation of the required CLI commands, which is much less accessible for non-technical users than R Markdown.

The *ActivePapers* project (Hinsen, 2015) describes a platform for publishing and archiving computer-aided research by turning scientific contents of software into so called pure computations (Hinsen, 2015). Hinsen presents extensive requirements, two prototypical implementations, and important lessons learned. Similarly to ERC and ERS, ActivePapers demonstrate feasibility of packaging reproducible research, but with a different approach without containerisation. The strong theoretical base and implementation from the ground up of that work are a counterweight to the more practical approach of the ERS, which largely adapts general purpose tools. Each approach has its own limitations. Hinsen (2015) concludes with the idea that computational models and methods should be separated from software tools for better preservation. However, this requires a deeper intrusion into the tools used by researchers daily, and thus is a more long-term change than the current scope of the ERS.

None of these related projects and ideas, just as the ERC and ERS, have found considerable uptake outside of specific groups or communities. What could help to close the adoption gap are author guidelines by journals and publishers. Several journals have established processes to execute workflows that belong to submitted manuscripts. Some of these processes rely on communication between reviewers and authors to ensure that the reproducing party can execute a workflow (Nüst and Eglen, 2021; Heroux, 2015) while others partner with commercial platforms (cf. Konkol *et al.*, 2020; **?**), and others develop their

[1] https://repo2docker.readthedocs.io/en/latest/specification.html
[2] https://www.nist.gov/programs-projects/cloud-reproducible-records

own formats for reproducible articles, most prominently eLife's ERA (Guizzardi *et al.*, 2021). Only few publishers actually recommend specific tools[3]. One of the exceptions is the journal *GigaScience*, which suggests multiple tools, including ERA and Gigantum, giving authors a lot of flexibility[4] and reducing risk of betting on the wrong approach.

3. Executable research compendium: technical specification

3.1. Design. A research compendium[5] is made up of parts, namely (i) data, e.g., collected or simulated inputs and calculated outputs, (ii) software, i.e., a fully automated or "scripted" computational workflow using, e.g., scripts, source code projects, and programming language libraries/modules, and (iii) text and graphics for consumption, e.g., instructions, a full manuscript, or figures. The term research compendium was coined by Gentleman and Lang (2007), reused by Stodden *et al.* (2015) and extended to an executable research compendium (ERC) by Nüst *et al.* (2017). The ERC extends the parts of a compendium in several respects: (i) it adds a further part allowing interaction, the UI bindings (Konkol *et al.*, 2019b), (ii) it extends the generic idea of software with a well-defined runtime environment based on containerisation, and (iii) it requires a literate programming (Knuth, 1984) document as the main document to execute the workflow. Figure 1 gives an overview of the ERC components. Based on these extensions, ERCs realise a portable and executable snapshot of a computational workflow with all documentation and presentation files and can be used as the core building block within scholarly publishing.

More practically, the ERC technical specification should support the goals of the ERC as described in Nüst *et al.* (2017) and serve as the foundation for the implementation of a Web service for creation, examination, and discovery of ERCs. The realisation is guided by several design goals. All these goals intent to be "preservation friendly", in the sense that preservation is never something that can be completed, but an ongoing activity.

1. Simplicity and convention over configuration The specification should not re-do something which already exists, e.g., in form of an open specification or tool, and not duplicate metadata unnecessarily. The risk of scattering information is mitigated by clear documentation and outweighed by the advantages of reuse. Furthermore, it must be possible to create a valid and working ERC manually and every researcher should be able to fully understand how ERCs work. Therefore, ERCs should be generally text file based, e.g., no embedded database or binaries unless needed. Supporting tools should be used to cover typical use cases with minimal required input by a creating user. We must also acknowledge that most ERCs will be created "post-hoc", so before submission or after completing a research project. While it would be beneficial to steer researcher's workflows on a highly reproducible track from the beginning, because it provides a basis for real collaboration (Whitehouse, 2019), researcher freedom, diversity in previous knowledge, and the evolutionary slow change of habits and practices make this unrealistic. The majority of cases should be covered by following regular conventions, whereas special cases should be support with configuration.

2. Nested containers We acknowledge existing standards for packaging a set of files and capturing computing environments. To be able to reuse these formats, the ERC has an outward facing packaging, where all components of the ERC are put into, but also contains a composite components which themselves package complex contents. Figure 1 shows how we distinguish these containers into the *inner or "runtime" container*, which holds the software dependencies of a particular workflow, and the *outer container*, which holds the inner container and all other text, data, and code files. Using the four layers of software stacks in scientific computing from Figure 1 in Hinsen (2018), the outer container contains project-specific code and the inner container contains domain-specific tools, scientific infrastructure, and non-scientific infrastructure. The outer container can be used for content-unaware validation and more easily adheres to established preservation practices. The nesting gives a separation that, in the long-term as computing environments are likely to evolve and likely break, maintains access to the core files for a specific workflow. This also means that data and control code is not (only) within the inner container so that one barrier to access, e.g., data in a PDF, is not replaced with another, e.g., data in a binary container image[6]. The inner container should also be created transparently based on an actionable text file. The *duality* of executable runtime container and recipe ensures transparency and a fallback option (Nüst and Hinz, 2019). The nesting also supports the idea of *"layered reproducibility"*[7] to handle different levels of dependencies in a used software stack. The outer container can contain a language-specific code package, e.g., for R or Python, enabling reusability and understandability, whereas the inner container captures the system dependencies. Users with different skill sets may interact with the layers differently, and layer usefulness may change over time.

3. Transparency, stability, and Openness All configuration and, as much as possible, also the content should be based on plain text files. Plain text files are usable by both humans and computers, ensure ERCs are acceptable by users with varying backgrounds and levels of expertise today, secure that ERCs remain understandable tomorrow, and enable that ERCs are easy to extend. If possible, "old" technologies are also preferable, as they are tested and stable, and are likely to outlive innovative formats[8]. It is therefore possible to both create and examine an ERC manually, i.e., without any supporting infrastructure or tools. All specifications and tools are published under open permissive licenses.

[3] Cf. ACM SIGMOD's reproducibility initiative recommending ReproZip, http://reproducibility.sigmod.org/.

[4] http://gigasciencejournal.com/blog/gigantum-joins-giga-reproducibility-machine-learning-toolkit/

[5] For a full list of publications on research compendia see https://research-compendium.science/.

[6] A problem pointed out by Greg Wilson on Twitter at https://twitter.com/gvwilson/status/1164240321028534274.

[7] Concept introduced by Noam Ross in an online discussion thread on rOpenSci at https://discuss.ropensci.org/t/creating-a-package-to-reproduce-an-academic-paper/1210.

[8] As argued by Wilson *et al.* (2017), in deference to the saying: *"What's oldest lasts longest."*

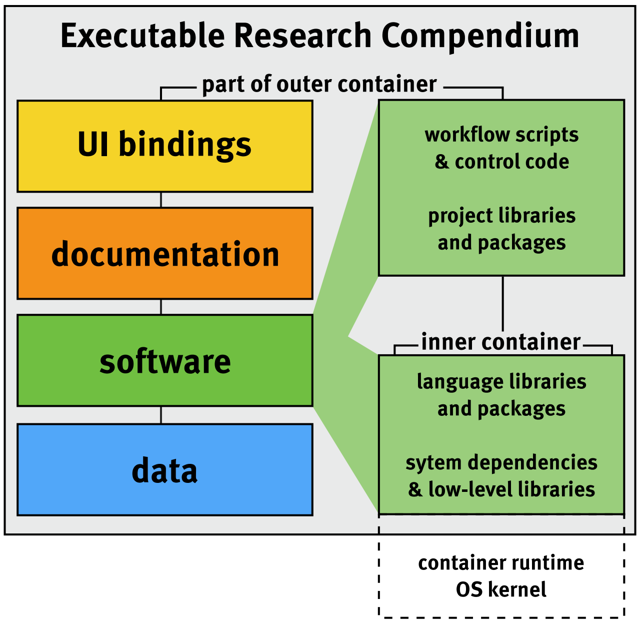

Executable Research Compendium
part of outer container
UI bindings
documentation
software
data
workflow scripts & control code
project libraries and packages
inner container
language libraries and packages
sytem dependencies & low-level libraries
container runtime
OS kernel

Fig. 1. Executable research compendium. Detailed look at software components and the inner container, with language libraries and system dependencies, and the outer container, with UIăbindings, documentation, data, workflow code and project libraries.

4. **Multiple entrypoints** Both humans and machines need to act on ERCs. Human users need a convenient and efficient way to interact with the substance of the research that is described in publications, which Marwick and Pilaar Birch (2018) describe with the useful "bottle-opener" metaphor. For machines, we need a "one-click" (Pebesma, 2013) command that can be used to execute and rudimentarily validate a full workflow. For users, we need a file that can be opened manually, or be shown to them as the default document by tools when opening an ERC. The literate programming paradigm, or computational notebooks, can support both these needs giving authors flexibility, readers accessibility, and machines a well-definedness.

The specification is accompanied by guides for users, namely for readers and preservationists, and developers, which comprises background on goals, design decisions, and the development process. The specification document uses technical language to clearly identify requirements and optional features, but is also enriched with examples and introductory texts.

3.2. The specification.

The specifications is published under a Creative Commons CC0 1.0 Universal License at https://o2r.info/erc-spec/spec/ in HTML and PDF format, is developed openly in an online repository[9], and archived in Nüst (2018). This section summarises the ERC specification—see the online specification for details.

An ERC must include a **main file** and a **display file**. The main file follows the literate programming paradigm (Knuth, 1984) and can be executed to create the display file. Both files should be named accordingly `main.extension` and `display.extension`, using correct file extensions and media type to use convention over configuration. The ERC specification encourages R Markdown (Allaire *et al.*, 2021a; Xie *et al.*, 2018) as the format for R-based analyses, and includes details for modelling metadata in the YAML front matter of R Markdown files, and for ensuring reproducibility by not using any caching features. These two files provide the entrypoints for human readers and executing tools.

Alternative names for main file and display file may be configured in the **ERC configuration file** `erc.yml`, which is the third required file in an ERC. The ERC configuration must include a globally unique identifier for the ERC, and the version of the used ERC specification. Authorship information is expected to be present in the main file and is therefore not repeated in the ERC configuration file. However, due to the lack of alternatives, the *licenses* of the core components or ERC can be explicitly modelled in the configuration file.

The final content of an ERC is the **runtime environment** which is represented by two files: an *executable runtime image*, which includes all base software and libraries to execute the packaged analysis, and a *runtime manifest*, which documents the images and contents as a self-contained complete recipe in an actionable format to create the executable runtime image. This approach uses containerisation and the runtime environment is the *inner container*. Due to Docker's standing as a de-facto standard, the ERC specification further defines the runtime environment, and how tools are expected to interact with the manifest and image, based on Docker. For example, the image file should be saved from a cache-less container built and must be tagged matching to the ERC ID. To enable controlling the workflow through tools, the default commands of the image must must render the main document and the working directory must be fixed so that files from the ERC can be connected into the runtime environment correctly, i.e., mounted into the container. The specification describes how these mounts are to be used to full executions of workflows, but also for substituting specific files between ERCs. Finally, the specification requires images to have an image tag with the ERC identifier.

The following files are an example of the payload for a minimal ERC using R Markdown and Docker:

```
main.Rmd
display.html
Dockerfile
image.tar
erc.yml
```

An example ERC configuration file is as follows:

```yaml
id: b9b0099e-9f8d-4a33-8acf-cb0c062efaec
spec_version: 1
main: main.Rmd
display: display.html
licenses:
  code: MIT
  data: "data_licenses_info.pdf"
  text: CC-BY-4.0
  metadata: CC0-1.0
```

The ERC bundles multiple parts to make a computational workflow and its documentation accessible, but is itself also a digital artefact that can be distributed, shared, and archived. Therefore the specification ends with sections on interacting with ERCs, preservation of ERCs, and

[9] https://github.com/o2r-project/erc-spec/

checking ERCs. For interactivity, the ERC configuration file can include metadata about the ERC's UI bindings (see Konkol *et al.*, 2019b). For preservation, the *outer container* of an ERC is a "**Bag**" following the BagIt specification (Kunze *et al.*, 2018). BagIt which ensures reliable storage and transfer through file checksums and ensures compatibility with established preservation workflows in form of bitstream preservation. The descriptive metadata of the bag also labels an ERC as such. A draft for a possible BagIt profile is included in the specification. This profile could make required metadata more explicit, and, for example, disallow usage of the "fetch" feature to require self-contained bags for ERCs. To reduce the risk of information loss, the specification deviates from the goal to not duplicate information and instead suggests to store metadata in all formats that specific use cases may need within the ERC. This secondary metadata are copies of the main metadata, e.g., the required fields and encoding of the data repository used for ERC storage, and increase the likelihood of at least some metadata being accessible in the unforeseeable future. One example for such secondary metadata is Zenodo record metadata in a JSON format. For checking ERCs, the specification defines a procedure, which ERC-supporting tools can implement. The check of an ERC comprises the execution of the workflow, and the comparison of the ERC's files after the execution. Most important file in the comparison set is, naturally, the display file because differences can point to meaningful deviations in a workflow's results.

4. Opening reproducible research system architecture

4.1. Structure. The architecture for a publishing workflow enhanced by ERC describes a system for opening reproducible research as part of a scholarly publication process— the *o2r architecture*. It is developed in an online repository[10], published online at https://o2r.info/architecture/ in HTML and PDF format, follows the arc42 Documentation template[11], and is archived in Nüst (2018). The arc42 template mandates a number of sections and contents, not all of which are described here— see the online architecture for details.

4.2. Goals. The o2r architecture describes the relationship of a **reproducibility service** (RS) with other services from the context of scientific collaboration, publishing, and preservation. Together these services can be combined into a new system for transparent and reproducible scholarly publications. As one part of such an system, the ERS must not replicate already existing functions but instead, inspired by the Unix philosophy (Wikipedia contributors, 2021g), do only one thing but do it well, namely provide a reliable way to create and examine packages of computational research, i.e. ERCs as reproducible publications. Existing functionalities, such as storage, authentication, or persistent identifiers must be connected with via APIs. *Creation* comprises uploading of a researcher's workspace with code, data, and documentation to the ERS, where a reproducible runtime environment is captured. This runtime environment forms the basis for *examination*, i.e. discovering, inspecting details, and manipulating workflows on an online platform. For the users, it is important that these features are provided in a guided process with excellent user experience, without too much expose of the underlying complex technology. Technology is more successful when it is easy to get things done (Bouffler, 2019). At the same time, the system must be transparent, so itself can be scrutinised and does not put the rigorousness of the actual ERCs into question.

The considered **stakeholders** in the architecture are author (scientist), who publishes an ERC as part of a scientific publication process to build a convincing argument, reviewer or editor (scientist), who examines an ERC during a review process to assess reproducibility and validity of results, reader (scientist), who views and interacts with an ERC on a journal website to understand methods and build upon results, publisher, who offer ERC-based publishing to increase quality of publications, curator or preservationist, who ensures research is complete and archivable using ERC, operator, who provides infrastructure to researchers at own university or publisher to communicate high-quality research using an ERC, and developer, who use and extend the tools around ERCs. For the remainder of this section, a focus lies on the author, reviewer, publisher, and preservationist.

4.3. Scope, context, and solution strategy. The **system scope and business context** are summarised in Figure 2 and describe the relations between infrastructures and services for communicating scientific computational workflows. The stakeholders interact with a number of platforms (leftmost column), but no directly with the ERS (second column). The publishing platforms, which authors and reviewers use, connect with the ERS through its API. Publishing platforms such as journal submission and review systems offer users to upload or create ERCs, track the submission status and access rights, e.g., for reviewers, and eventually expose published ERCs through their search results and journal websites. The ERS may retrieve files from collaborations platforms, where authors collaborate on data, code or text, if authors submit links instead of directly uploading files, and it can use registries to both harvest and publish metadata. These registries power catalogues and search portals directly and mediately via data repositories and archives, and thereby enable users to discover ERCs. The ERS offers ERC creation and examination services and uses different supporting services (third column) to authenticate users, to retrieve software artefacts, to store runtime environment images, to execute workflows, and to store ERCs. Using an existing ID provider frees the ERS from storing authentication data securely and from ensuring that users are real persons. The execution infrastructure is accessed through containerisation tools based on the HTTP protocol and thus is scalable, e.g., when deployed in a distributed cloud-based infrastructure. Software repositories repository provide software artefacts during ERC cre-

[10] https://github.com/o2r-project/architecture/
[11] https://docs.arc42.org/home/

ation, e.g., installing software libraries from a programming language's package distribution infrastructure, and can also provide standardised APIs to store to containers of the executable runtime environments. Data repository service the reproducibility service with content for ERC creation but can also store the completed ERCs. In turn the data repositories may connect to archives and digital preservation systems (rightmost column) for long-term storage. These archives employ extended data and metadata management because a different kind of access and re-use is of concern for these systems, e.g., to ensure long-term access rights, and therefore these concerns are relevant for the ERS even though it does not directly connect to archives, as the ERS should ensure a smooth transfer of created ERCs from storage to archives. The supporting services also connect with each other, for example, the execution infrastructure can access trusted data repositories to download data that for reasons of storage size are not included within an ERC. All of these systems are connected through Web protocols.

The solution strategy of the architecture describe the **architectural decisions**. First, the developed solution is set in an existing system of services, and first and foremost must integrate well with these systems, focusing on the specific missing features of building and running ERCs. These features are provided via a well-defined Web API in the ERS. Second, internally a microservice architecture is used to allow dynamic development, e.g., independent development and deployment cycles, and support the large variety of skills available in the academic development team. This architecture comes at the cost of increased application complexity when it comes to testing and deployment. The application state is shared between microservices through a database. The database's operation log is used to power notifications and events across microservices and real-time updates of the user interface based on WebSockets. Third, the ERS itself does not provide a reliable storage solution. The microservices simply share a common pointer to a local file system path which should be regarded as ephemeral. Forth, the client application manages the control flow of all user interactions and ensures the Web API operations are executed in the required order. Finally, generic functions should be developed as standalone tools with a command-line interface (CLI). The CLI allows integration into microservices and independent usage at the same time. These generic functions can be packaged in container images and executed as containers by the microservices, which ensures easy distribution through a container registry and independent updating from the microservices themselves, but also allows to run tools either next to the microservices or in an independent container cluster, thus providing scalability.

4.4. Building block view. The arc42 template defines architectural components in alternating layers of black box, where only the outside appearance and interaction options are described, and white box, where internal details are given. A white box layer then includes components described as black boxes, etc. In this work, we single out the white box view on the reproducibility service (RS) as shown in Figure 3. The ERS itself consists of a webserver to distribute incoming API calls to the microservices as a reverse proxy and to serve the static files of the user interface. The webserver also manages secure communication via HTTPS. The microservices run in containers. The use containerised tools, namely containerit (Nüst and Hinz, 2019) and o2r-meta[12], connect to a MongoDB document database for ERC metadata, users, and session information, connect to an Elasticsearch search index for full-text search and advanced queries, and access a local shared file storage which is mounted into every microservice container. The webserver as well as the databases also run as containers. The microservices are implemented in multiple programming languages, namely JavaScript (Node.js), R, and Python. Each microservice is generally responsible for one endpoint in the API or for larger sets of features, such as live notifications or exporting of ERCs.

4.5. Runtime view. The two main scenarios of ERC creation and ERC examination are described with sequence diagrams in Figure 4 and Figure 5 respectively. In the **ERC creation sequence**, the author creates an ERC from their workspace of data, code, and documentation. The author can provide these resources as a direct upload, but a more comfortable process is loading the files from a collaboration platform, e.g., from a public share created at a cloud file storage provider. After the files are available, the o2r-meta tool tries to *extract metadata* from the available files, so the user must not fill out all fields manually. In the same step, the metadata is translated into multiple file formats, *broker metadata*, and saved so that exported ERCs are more likely to include metadata understood by other services, such as archives. The ERC is now a non-public *candidate compendium*, until the the users has checked and possible *updated metadata*, which triggers a metadata validation, i.e., if all mandatory fields are provided, and possible a further brokering. Then the compendium is saved. If users want to provide access to a candidate compendium, they can also create a *public link* that gives read-only access and allows execution (option not included in diagram). Next, a user can *start jobs* for a published compendium, i.e., execute the workflow. Part of the job execution is to automatically create missing configuration files, i.e., the `erc.yml`, and the runtime environment manifest and image. This relies on the containerit tool and the execution infrastructure. As the last step of a successful job, the runtime environment image is exported into the ERC. If configuration file and runtime environment are already present, their generation is skipped. Finally, the user *starts a shipment*, i.e., a deposition of the ERC to a data repository. For this step, the ERC is packaged as a Bag. To be able to check the correct upload at the repository, the publishing of the shipment is an extra action by the user.

In the **ERC examination sequence**, the user initiates the opening of an existing ERC by providing a reference such as DOI or URL. The ERS retrieves the ERC, saves the files locally and loads the contained metadata. Then the user can start a new job for the compendium. The

[12] https://github.com/o2r-project/o2r-meta

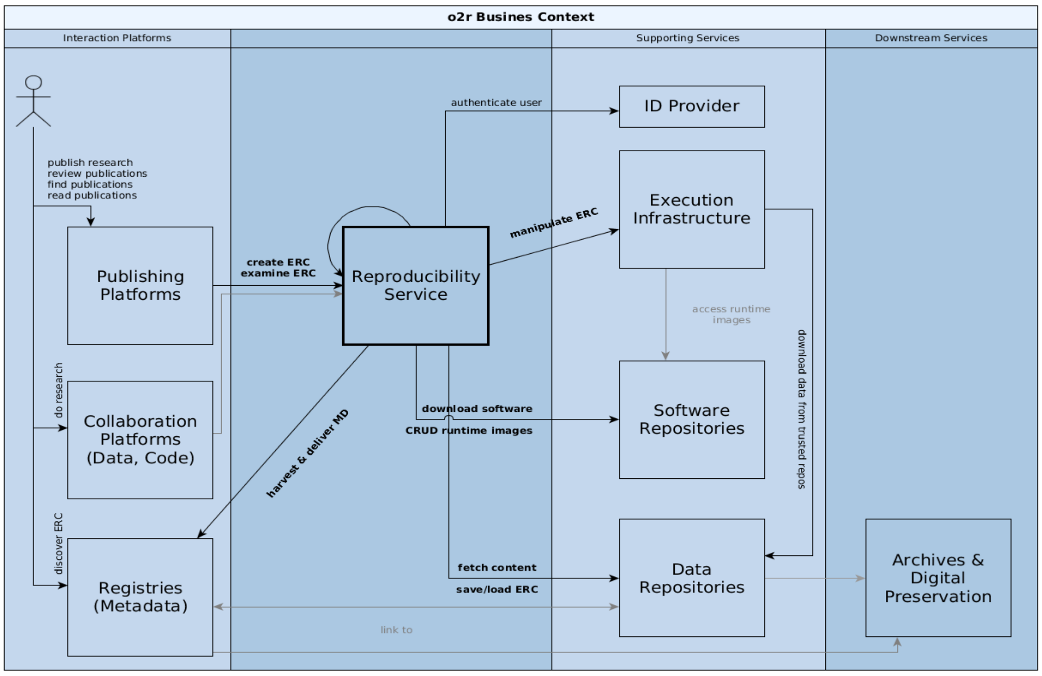


Fig. 2. Business context; full scale image online at https://o2r.info/architecture/#31-business-context.

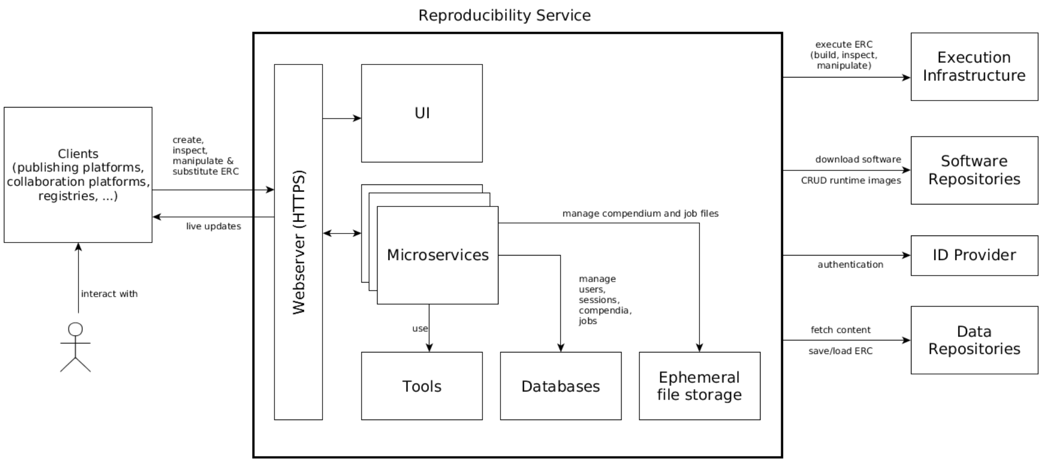


Fig. 3. White box Reproducibility Service; full scale image online at https://o2r.info/architecture/#527-whitebox-reproducibility-service.

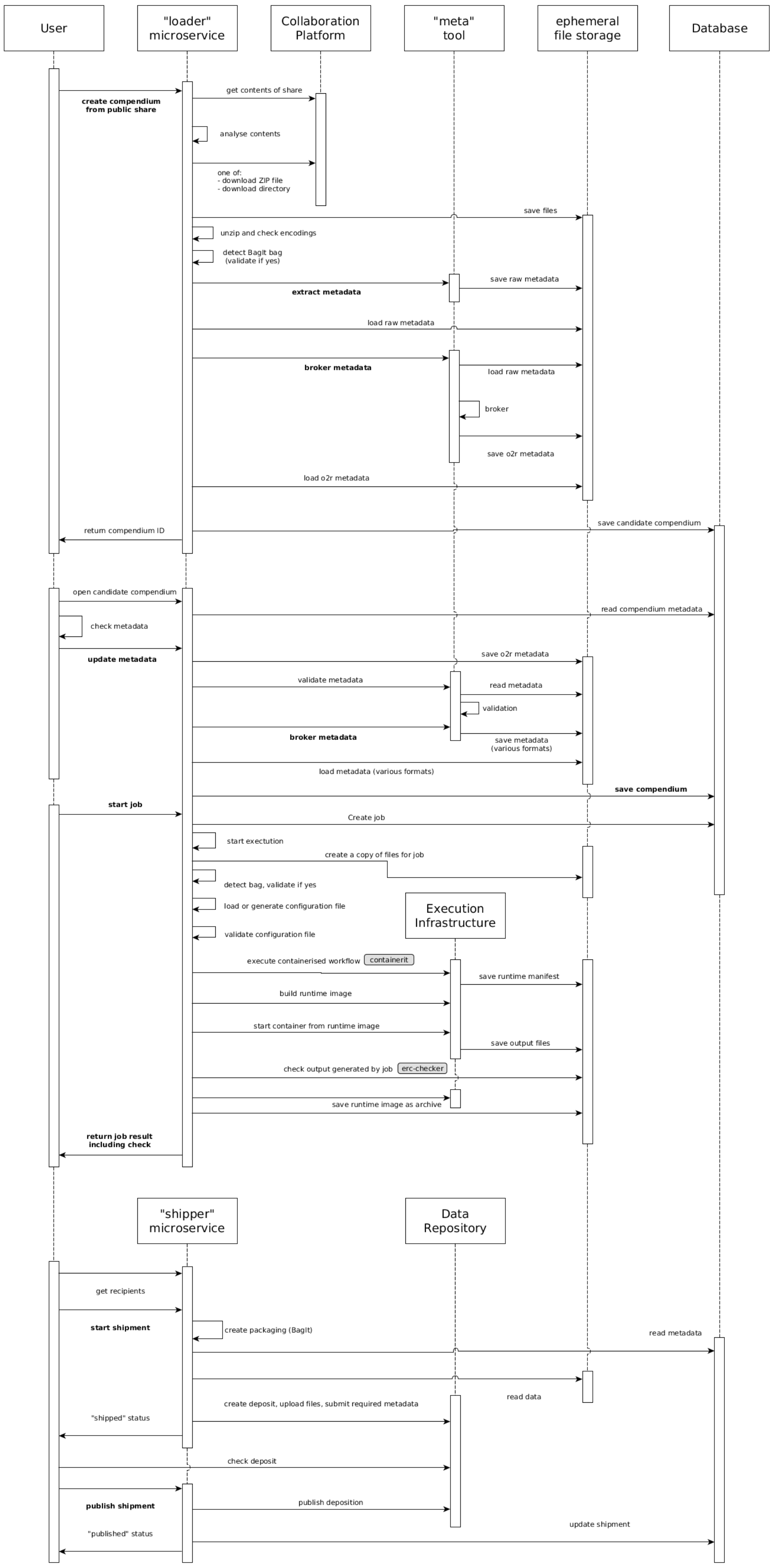


Fig. 4. Runtime view ERC Creation; full scale image online at https://o2r.info/architecture/#61-erc-creation.

user's client can use the ID to connect to the live logs as the job runs through all steps (see Section 5.1 for details about the job steps). The job starts with creating a copy of the compendium's files for the job. The copy allows to compare the original output, i.e., the display file, with the newly created one. A copy-on-write file system is advantageous for this step. Then the archived runtime image is loaded from the file in the compendium into a runtime repository. This repository may be remote and either public or private, e.g. based on the Docker Registry or a GitLab instance, or simply the local image storage. Then all files except the runtime image archive are packed so they can be send to a container runtime. The container runtime can be local, e.g., the Docker daemon, or a container orchestration infrastructure such as Kubernetes. The container run provides log updates as a stream to the microservices, which update the database, whose changes trigger updates of the user interface. When the container is finished, the microservice compares the created outputs with the ones provided in the compendium using the erc-checker[13] tool. The result is a display file with highlighted differences both in text and graphics, which is shown to the user as can be seen in Figure 8. Based on these aids, the reader, e.g., the reviewer, can quickly determine if deviations in the outputs are relevant or not, e.g., if they are only graphical artefacts or acceptable numerical variation.

5. Reproducibility service

5.1. API. The ERS exposes its functionality via a RESTful HTTP API. The API is specified using the OpenAPI model (Wikipedia contributors, 2021f). It uses WebSockets (Wikipedia contributors, 2021i) for push-based notification from server to client and encodes requests and responses in JSON. It is developed in a public repository[14], hosted online at https://o2r.info/api/, and archived in Nüst (2018). The website provides access to the machine-readable specification in YAML format[15] and an HTML rendering for reading.

The API provides several endpoints to manage compendia and their metadata, compendium execution (jobs, and links for authentication-free execution), compendium substitution, compendium shipments, and users and their authentication and access levels. For full examples of resources, e.g., for ERC metadata, please see the demo server and reference implementation (Section 5.2). The management operations use matching HTTP verbs for creating (`POST`), listing or retrieving (`GET`), updating (`PUT`), and deleting (`DELETE`) resources. Different user levels allow or prevent certain operations only for specific users, most importantly only "known" users after a manual check following the registration are allowed to create and examine ERCs. User authentication is based on the OAuth 2.0 protocol (Hardt, 2012) and operation authentication against the API uses a session cookie. The API includes a version in the URL path and provides index responses to support client-side construction endpoints and stability. The following JSON documents are the responses to the `/api/` and `/api/v1` endpoints. The latter document lists all resources of the API that must be combined in sequence, controlled by the client-side, to realise the runtime interactions described in the Section 4.5.

```
{
  "about": "https://o2r.info",
  "versions": {
    "current": "/api/v1",
    "v1": "/api/v1"
  }
}
```

```
{
  "auth": "/api/v1/auth",
  "compendia": "/api/v1/compendium",
  "jobs": "/api/v1/job",
  "users": "/api/v1/user",
  "search": "/api/v1/search",
  "shipments": "/api/v1/shipment",
  "recipients": "/api/v1/recipient",
  "substitutions": "/api/v1/substitution",
  "links": "/api/v1/link"
}
```

Complex compound resources, such as `../compendium`, also provide sub-resources to access parts or related resources more conveniently, e.g., `../compndium/abc12/jobs` to access related jobs. Query parameters on selected resources are provided to filter the results. For example, the URL (spaces added for readability) `../job? limit=10&compendium_id=abc12 &status=success&fields=user` returns at most 10 jobs which succeeded, including the `users` who started them, for the compendium with identifier `abc12`. The `../users` resource facilitates user management and `../search` the discovery of ERCs and jobs. As these are common API features, they are not described in detail here. Konkol *et al.* (2019b) describes the concepts of user interface *bindings* and how to create new ERCs through *substitution*, modelled in the `../bindings` and `../substitution` resources, in detail.

The execution of a compendium consists of a fixed sequence of **job steps**. The steps can have one of multiple statuses: `success`, `failure`, and `running`. The overall job status is a combination of the steps' statuses—if at least on step is `failure` or `running`, so is the job. Some steps can be skipped because the job for executing a complete compendium and executing a workspace to create a complete compendium share some steps that should be readily reused in implementations of the API. The job metadata captures logging messages and start/end time separately for each step. Because jobs are computationally intensive operations, users must be logged in to start a job. The job steps are (cf. Section 4.5):

1. Validate the bag (skipped if workspace)
2. Generate compendium configuration (skipped if present)
3. Validate compendium
4. Generate inner container manifest (skipped if present)
5. Prepare image payload archive (to build and run the image on remote hosts; possibly costly operation)

[13] https://o2r.info/erc-checker/
[14] https://github.com/o2r-project/api-spec/
[15] https://o2r.info/api/o2r-openapi.yml

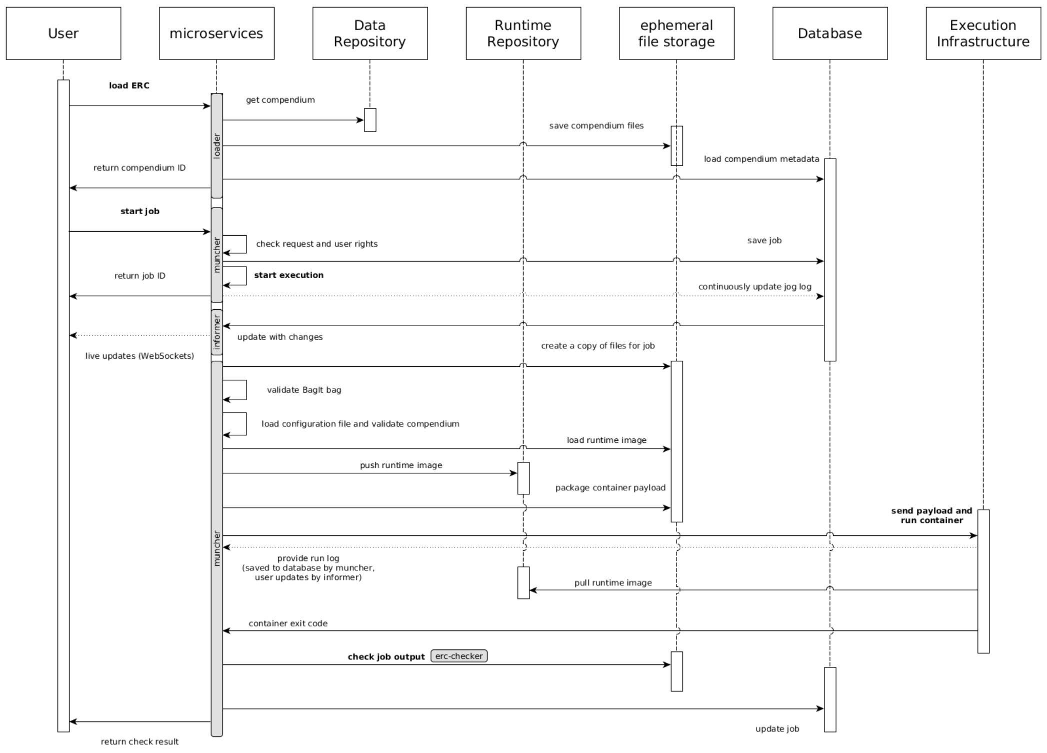
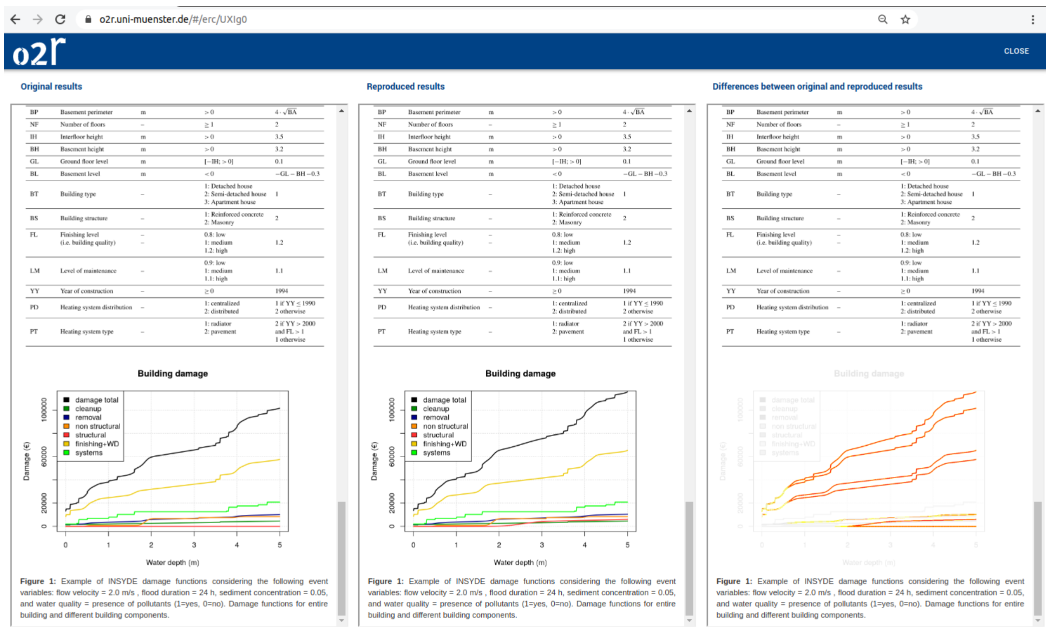


Fig. 5. Runtime view ERC examination; full scale image online at https://o2r.info/architecture/#62-erc-inspection.



Original results

BP	Basement perimeter	m	>0	$4\cdot\sqrt{BA}$
NF	Number of floors	–	≥ 1	2
IH	Interfloor height	m	>0	3.5
BH	Basement height	m	>0	3.2
GL	Ground floor level	m	$[-IH; >0]$	0.1
BL	Basement level	m	<0	$-GL - BH - 0.3$
BT	Building type	–	1: Detached house 2: Semi-detached house 3: Apartment house	1
BS	Building structure	–	1: Reinforced concrete 2: Masonry	2
FL	Finishing level (i.e. building quality)	– –	0.8: low 1: medium 1.2: high	1.2
LM	Level of maintenance	–	0.9: low 1: medium 1.1: high	1.1
YY	Year of construction	–	≥ 0	1994
PD	Heating system distribution	–	1: centralized 2: distributed	1 if YY $\leq$ 1990 2 otherwise
PT	Heating system type	–	1: radiator 2: pavement	2 if YY > 2000 and FL > 1 1 otherwise

Figure 1: Example of INSYDE damage functions considering the following event variables: flow velocity = 2.0 m/s , flood duration = 24 h, sediment concentration = 0.05, and water quality = presence of pollutants (1=yes, 0=no). Damage functions for entire building and different building components.

Reproduced results

BP	Basement perimeter	m	>0	$4\cdot\sqrt{BA}$
NF	Number of floors	–	≥ 1	2
IH	Interfloor height	m	>0	3.5
BH	Basement height	m	>0	3.2
GL	Ground floor level	m	$[-IH; >0]$	0.1
BL	Basement level	m	<0	$-GL - BH - 0.3$
BT	Building type	–	1: Detached house 2: Semi-detached house 3: Apartment house	1
BS	Building structure	–	1: Reinforced concrete 2: Masonry	2
FL	Finishing level (i.e. building quality)	– –	0.8: low 1: medium 1.2: high	1.2
LM	Level of maintenance	–	0.9: low 1: medium 1.1: high	1.1
YY	Year of construction	–	≥ 0	1994
PD	Heating system distribution	–	1: centralized 2: distributed	1 if YY $\leq$ 1990 2 otherwise
PT	Heating system type	–	1: radiator 2: pavement	2 if YY > 2000 and FL > 1 1 otherwise

Figure 1: Example of INSYDE damage functions considering the following event variables: flow velocity = 2.0 m/s , flood duration = 24 h, sediment concentration = 0.05, and water quality = presence of pollutants (1=yes, 0=no). Damage functions for entire building and different building components.

Differences between original and reproduced results

BP	Basement perimeter	m	>0	$4\cdot\sqrt{BA}$
NF	Number of floors	–	≥ 1	2
IH	Interfloor height	m	>0	3.5
BH	Basement height	m	>0	3.2
GL	Ground floor level	m	$[-IH; >0]$	0.1
BL	Basement level	m	<0	$-GL - BH - 0.3$
BT	Building type	–	1: Detached house 2: Semi-detached house 3: Apartment house	1
BS	Building structure	–	1: Reinforced concrete 2: Masonry	2
FL	Finishing level (i.e. building quality)	– –	0.8: low 1: medium 1.2: high	1.2
LM	Level of maintenance	–	0.9: low 1: medium 1.1: high	1.1
YY	Year of construction	–	≥ 0	1994
PD	Heating system distribution	–	1: centralized 2: distributed	1 if YY $\leq$ 1990 2 otherwise
PT	Heating system type	–	1: radiator 2: pavement	2 if YY > 2000 and FL > 1 1 otherwise

Figure 1: Example of INSYDE damage functions considering the following event variables: flow velocity = 2.0 m/s , flood duration = 24 h, sediment concentration = 0.05, and water quality = presence of pollutants (1=yes, 0=no). Damage functions for entire building and different building components.

Fig. 6. Screenshot of the ERS user interface showing the result of a failed job execution due to differences in a figure; left column: display file provided by the author, middle column: display file generated by the ERS, right column: display file with highlighted difference generated by erc-checker.

6. Build image and add image tag `erc:<erc identifier>`
7. Execute container
8. Check the display file of the job against the compendium's baseline
9. Save image to tarball (skipped if check failed)
10. Cleanup (implementation specific)

An editor or admin, but not users themselves, can create a **link** with the resource `../link` which provides a second identifier for a specific compendium which allows users to execute a compendium without logging in. Such link identifiers may be short lived or dynamic.

The process of exporting a compendium to a storing repository is called **shipment** (cf. Section 4.5) and is modelled in the two endpoints `../recipient`, which lists supported services, and `../shipment`, which controls the possibly costly and irreversible operation. To allow validation in the receiving service, the export is a two step process: first the new shipment is created, then the actual publishment can be triggered.

5.2. Reference implementation. The microservice architecture results in numerous projects within the o2r code organisation[16]. For easier evaluation and reproducibility, all microservices are integrated in one single code repository `reference-implementation`[17] using git submodules, which is archived in Nüst (2018). The online **demo server** is available at https://o2r.uni-muenster.de.

The following instructions require Docker (Wikipedia contributors, 2021c) (tested with version 20.x) and GNU Make (Wikipedia contributors, 2021e) (tested with version 4.1). The commands must be executed in the base directory of the `reference-implementation`. If Make is not available (e.g., on Windows OS), then the instructions of `make` targets in the `Makefile` may be executed manually on a command line. The target `reproduce` loads the images saved to tarballs and executes them in a configuration suitable for local testing and development based on `docker-compose` (Docker Inc., 2019). The demonstration project includes a small OAuth provider so that users can log in with different user levels with a single click.

```
clone https://github.com/o2r-project/reference-implementation
cd reference-implementation
make reproduce
# to use git repo configuration and not Zenodo: make release
```

5.3. Examples. A number of example ERCs have been published on the demonstration platform of the o2r project—see Section 5.2 and on GitHub[18]. These examples include over a dozen scientific articles reproduced as part of Konkol *et al.* (2019a). One ERC was part of a pilot collaboration with the Copernicus Journal ESSD, which conducts open reviews. The referee report (González Ávalos, 2020) mentions the ERC positively.

A complete minimal example is given by the ERC configuration file above (see Section 3.2) and the following four documents. The minimal example is published at https://o2r.uni-muenster.de/erc/q7Eje. The data file, `data.csv`, provides a simple statistics about cargo ships[19]:

```
"year","capacity"
"1980",11
"1985",20
"1990",26
"1995",44
"2000",64
"2005",98
"2010",169
"2014",216
"2015",228
"2016",244
```

The `Dockerfile` defines the computational environment (extra line breaks for readability).

```
FROM rocker/geospatial:3.4.4
LABEL maintainer="o2r"
# Packages skipped because in base image: [shortened]
WORKDIR /erc/
CMD ["R", "--vanilla", "-e",
  "rmarkdown::render(input = \"/erc/main.Rmd\",
    output_format = rmarkdown::html_document(),
    output_dir = \"/erc\", output_file = \"display.html\")"]
```

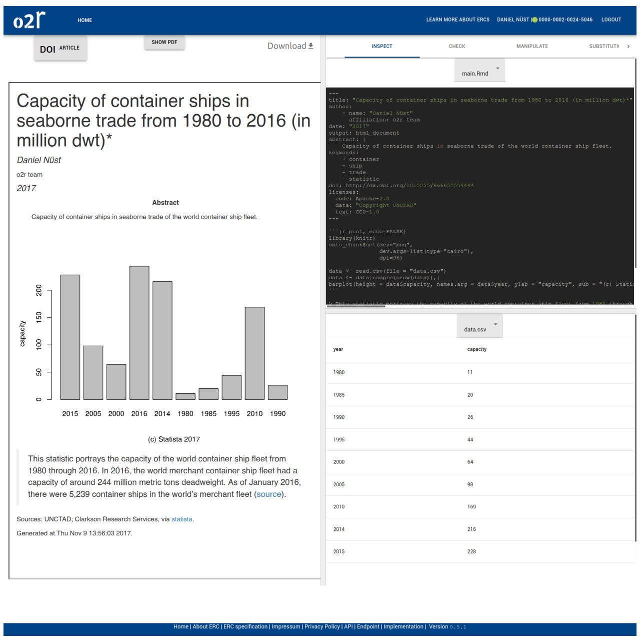
Because of the small example, it installs no software into the base image. The final line configures the command to be run when the ERC is executed. The `Dockerfile` and `erc.yml` are generated by the ERS, ensuring that the rendering command matches the way that the ERS mounts the ERC's files into the container. The other files are created by the author.

The source of the HTML display file, `display.html`, shown in the left hand side of Figure 7, is not included here. The display file can serve as the baseline for assessing whether the reproduction was successful. The R Markdown document includes metadata and a simple plot function to show the input data:

~~~
---
title: "Capacity of container ships in seaborne trade from 1980
    to 2016 (in million dwt)*"
author:
    - name: "Daniel Nüst"
      affiliation: o2r team
date: "2017"
output: html_document
abstract: |
    Capacity of container ships in seaborne trade of [shortened]
doi: http://dx.doi.org/10.5555/666655554444
---

```{r plot, echo=FALSE}
library(knitr)
opts_chunk$set(dev="png", dev.args=list(type="cairo"), dpi=96)

data <- read.csv(file = "data.csv")
data <- data[sample(nrow(data)),]
barplot(height = data$capacity, names.arg = data$year,
    ylab = "capacity", sub = "(c) Statista 2017")
```

[shortened for inclusion in paper]
~~~

Note the use of the `sample(..)` function, which randomises the order of the data to demonstrate the display of the check, shown in Figure 8. The R Markdown frontmatter could include additional information, such as a `licenses` element or `keywords`, which are used by the

[16] https://github.com/o2r-project/
[17] https://github.com/o2r-project/reference-implementation
[18] https://github.com/o2r-project/erc-examples/
[19] Statista 2017, Source: https://www.statista.com/statistics/267603/capacity-of-container-ships-in-the-global-seaborne-trade/.



Capacity of container ships in seaborne trade from 1980 to 2016 (in million dwt)*

Daniel Nüst

o2r team

2017

Abstract

Capacity of container ships in seaborne trade of the world container ship fleet.

(c) Statista 2017

This statistic portrays the capacity of the world container ship fleet from 1980 through 2016. In 2016, the world merchant container ship fleet had a capacity of around 244 million metric tons deadweight. As of January 2016, there were 5,239 container ships in the world's merchant fleet (source).

Sources: UNCTAD; Clarkson Research Services, via statista.

Generated at Thu Nov 9 13:56:03 2017.

~~~
---
title: "Capacity of container ships in seaborne trade from 1980 to 2016 (in million dwt)*"
author:
    - name: "Daniel Nüst"
      affiliation: o2r team
date: "2017"
output: html_document
abstract: |
    Capacity of container ships in seaborne trade of the world container ship fleet.
keywords:
    - container
    - ship
    - trade
    - statistic
doi: http://dx.doi.org/10.5555/666655554444
licenses:
  code: Apache-2.0
  data: "Copyright UNCTAD"
  text: CC0-1.0
---

```{r plot, echo=FALSE}
library(knitr)
opts_chunk$set(dev="png",
               dev.args=list(type="cairo"),
               dpi=96)

data <- read.csv(file = "data.csv")
data <- data[sample(nrow(data)),]
barplot(height = data$capacity, names.arg = data$year, ylab = "capacity", sub = "(c) Stati
```

> This statistic portrays the capacity of the world container ship fleet from 1980 through
~~~

year	capacity
1980	11
1985	20
1990	26
1995	44
2000	64
2005	98
2010	169
2014	216
2015	228

Home | About ERC | ERC specification | Impressum | Privacy Policy | API | Endpoint | Implementation | Version 0.5.1

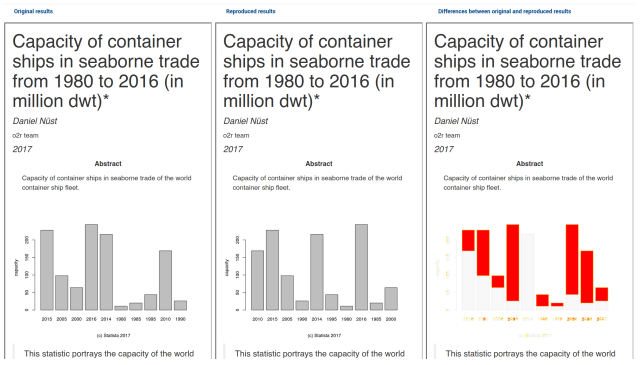
Fig. 7. Screenshot of ERC examination view in the ERS. The left hand side shows the display file rendering in HTML, the right hand side allows to inspect the source RăMarkdown document and the input data.

ERS to pre-fill the ERC creation form. The doi can link to a related publication in case the ERC is created as a supplement.

This example also demonstrates that creation of an ERC is possible by hand. None of the generated files (`Dockerfile`, `erc.yml`) are more complex than the main file authored by a researcher, even if researchers need to educate themselves to create the former file (cf. Nüst *et al.*, 2020). To create the outer package, command line tools such as `bagit`[20] can be employed. Finally, the ERC also demonstrates that manual examination is feasible. First, the outer package can be unzipped. Then, the ERC configuration file defines the main document to inspect for control code, trivial in this example but in complex workflows possibly not quite so easy, and the display file to open for reading. The commands in the `Dockerfile` can be used to recreate the computational environment manually, with the complication that the base image must be available to dig out the commands used to create it from the image layer metadata.

6. Discussion

This work presents one specific implementation of how computational reproducibility can be connected with scholarly review and publishing. Naturally, a single implementation of an API used only by one project team has severe limitations and experiences are not generalisable, not the least because of the confining context of a research project. Nevertheless, the implementation nevertheless points out many important aspects and taught valuable lessons, which can help to adopt concepts, specifications, or even software for a productive infrastructure.



Capacity of container ships in seaborne trade from 1980 to 2016 (in million dwt)*

Daniel Nüst

o2r team

2017

Abstract

Capacity of container ships in seaborne trade of the world container ship fleet.

This statistic portrays the capacity of the world

Fig. 8. Partial screenshot of ERC check result in the ERS. Three columns compare the display file provided by the author (left hand side) with the display file generated by the ERS (middle). The right hand side column adds a visual highlight to show the differences between the two plots, in this case quite exaggerated in the columns, but small differences, e.g., in the figure margins, could be easily judged by a human examiner as irrelevant.

6.1. Project set-up, maintainability, and security.

On the project set-up and maintainability, the presented web service does fulfil the need for an extensible trustworthy software by being an open, FOSS project itself, prohibiting vendor lock-in and standards lock-in, and ensuring the crucial option to examine the platform itself. The ERS focuses on one specific problem: creating and examining ERCs for aiding scholarly peer review so that code central to claims made in a submission can be evaluated (cf. Hawkins, 2019). It does not solve data curation[21] or storage, nor has it measures to evaluate quality of data, software, or the scientific merit. This reduction, albeit the internal complexity of the tools, improves the usability and extensibility.

The complexity introduced through the many microservices was good at the start as it provided flexibility, but the need for consolidation for sustainability lead to re-integration of some services since their inception. The ERS itself uses containers and is therefore readily installed in various infrastructures. Furthermore, the ERS as such is not limited to geospatial sciences at all, but the communication within the community and the testing and demonstrations with examples need to be tailored towards a specific audience to increase chances for adoption. This is partly an explanation for the numerous, seemingly redundant, tools presented in Section 2. On the long-term maintainability of the project, the individual software components have a *bus factor* (Wikipedia contributors, 2021b) of 1, at most 2, which is of course bad. A mitigation that works at least until the ERS itself breaks, could be the integration of the ERS with `repo2docker` [cf. project_jupyter_binder_2018], so that an ERC saved to a repository can be loaded into BinderHub, where the `erc.yml` triggers the Binder to be opened with the ERS and o2r user interface. Regarding *security*, containerisation offers good mechanisms for controlling unknown code with respect to used resources, and the container sandbox should be further hardened for productive systems, e.g., using AppArmor (Wikipedia contributors, 2021a). The ERS could also be extended to only ex-

[20] https://libraryofcongress.github.io/bagit-python/

[21] To get a glimpse of the curators' perspective, take a look at the first draft for a GeoJSON curation primer, a file format probably deemed "simple" be geospatial data researchers[˜14].

amine ERCs created by itself through signing ERCs. This gives the ERS control over the image build process, especially the base image and the allowed software repositories. The main security feature are the real user profiles through the user login based on ORCID. The ORCID project has measures to identify fake accounts, and users are given the rights to create ERCs manually by ERS administrators.

Finally, the ERS is fully dependent on the Docker container runtime at this point—a technology that while stable at its core and subject to standardisation itself[22], could be reduced considerably in its features to provide a stabler footing tailored to research and preservation requirements. More modern and less vendor-specific alternatives, including rootless Docker or plain OCI-based tools for building and running images, should also be explored. Beyond the sandboxing of Docker and the controlled user access, no further security measures have been explored. The more containers are used in research, the more likely it becomes that a special container and image specification, which can be maintained long term and is tested with preservation strategies (cf. Emsley and De Roure, 2018; Rechert *et al.*, 2017), will be developed, e.g., based on Singularity Image Format[23] or on OCI Image Format[24]. Besides preservation, specialised container runtimes can also provide provenance metadata, improve performance, and enable composition into pipelines (Youngdahl *et al.*, 2019; Molenaar *et al.*, 2018). These adoptions are needed to resolve the conflict between reproducibility and tools that are largely driven by requirements for scalable cloud computing, which where not designed with preservation in mind [25]. An alternative mitigation could be multiple (cf. Glatard *et al.*, 2018) container engines, which could not be realised for the ERS yet. Furthermore, the lessons learned from alternative approaches, such as ActivePapers (Hinsen, 2015), should be critically evaluated and translated into improvements for the next generation of the ERC and ERS. Finally, the many diverse approaches for sharing reproducible workflows (cf. related work) are important to explore alternatives and serve specific needs, but there certainly is also potential for standardisation and consolidation that would be beneficial for long-term maintenance of the ERS or other platforms (cf. Mecum *et al.*, 2018).

6.2. Understandability and usability. The ERC is not an abstraction that hides uncertainty. Instead it is simple enough that is should be understandable by all **researchers** using computational methods. The core concepts of computational notebooks and containerisation are becoming more widespread across researchers improving reproducibility of their works, and therefore the combination of both into the ERC is likely to be understandable and usable, too. One can examine ERCs without the reproducibility service[26], and the ideas of multiple entrypoints and nested containers are quickly explained. However, it is more realistic to require from researchers to use R Markdown than to ask them to learn metadata standards and become proficient in containerisation. The price for a stable capturing of the computational environment—executing the workflow once— is therefore acceptable. The tedious task of capturing relevant metadata is also automated as much as possible and ensures high user-friendliness for authors. The organisation of the ERC contents beyond the entrypoints are lie with the authoring researchers. The ERS makes sure the ERC is not a black box, but the author makes sure the contents are understandable. More expressive modelling of the workflow could be beneficial and related specifications do it, but the flexibility does have advantages when it comes to adoption and adaptability for different communities. Authors may choose to use, e.g., digital scientific notations [cf.] that are suitable for their work, as long the the full workflow is executed from the main document. The used template or structure can be exposed transparently in the ERC metadata via a resolvable identifier. Notably, the ERC is not a collaboration format. We expect collaborating researchers to work on the level of notebooks and workflow pipelines, which they can then wrap in an R Markdown document when submitting their study. Finally, the ERC and Web service need to be evaluated from a user perspective with a larger pilot (cf. product based approach and focus on user needs as argued in (Whitehouse, 2019)) to complement the internal reflections presented here. Some specific challenges could already be identified, such as the lack of an explicit configuration of the time zone that leads to check failures because times are off by one hour between original and reproduced display file. In that case, the environment variable `TZ=CET` in the original workflow could resolve the issue. However, only and exposure to various types of users and workflows can harden the processes enough against edge cases.

One core challenge is the proper modelling and documentation of **licenses**. This is quite complex for an aggregated artefact like the ERC, though it naturally works best with open data/methods/source software/text licenses, if it can made to work with non-open licenses or proprietary software at all, which was not considered for this work. The current specification and implementation merely scratch the surface with individual licenses for the main components, but also go further in explicitly modelling them as other reproducibility formats. This is a compromise to at least provide compatible licenses for important parts, but does not do the importance of software citation (Katz *et al.*, 2021) and giving contributors credit enough justice. The redistribution of full software stacks, however, should be less of a licensing issue as free and open source software licenses explicitly allow this, especially for unchanged software. Software and data citation remain a challenge for all aggregating reproducibility packages, yet the ERC could have the potential to derive machine-readable metadata for automating parts of workflow citation networks.

For **developers and operators** we see the usability of

[22] https://opencontainers.org/
[23] https://github.com/hpcng/sif
[24] https://github.com/opencontainers/image-spec
[25] See for example the challenges around the `tar` format used in container images: https://www.cyphar.com/blog/post/20190121-ociv2-images-i-tar
[26] https://o2r.info/erc-spec/user-guide/examination/#manual

the API as quite good, though a better distinction between loading of workspace and opening of an ERC, which currently is realised with complex payloads to the same API endpoint, could be helpful. The integration into publishing systems has not been realised, e.g., regarding user authentication. only implemented authentication provider is ORCID[27], which may not work for interested publishers. Furthermore, the procedural integration with publishing platforms is still under development with a focus on the Open Journal Systems (OJS). The performance of the ERS was investigated with a bespoke load test script which simulated parallel ERC creation and examination sessions. The sessions included a small randomness and relatively long pauses where use interaction, e.g., reading a paper or filling out a form, can be expected, and a fixed execution time of the actual process. Using the existing demo server, the wait times during tests were found to be generally acceptable, given that the user is aware of rather complex operations happening. A detailed report on the load tests is part of the ERS API documentation[28]. The custom load testing code is very well suited for evaluating ERS deployments and their scalability in different infrastructures. Finally, the sustainability of the implementation is, naturally for a research prototype, unclear. While several developers have worked on the platform, which increases trust in documentation and maintainability, the microservice-based approach also led to some fragmentation with multiple used programming languages (Node.js, Python).

6.3. Capabilities and features. Regarding the capabilities and features, the ERS can serve an important purpose for integrating workflow reproductions into peer review. The ERS allows to take snapshots at the point of submission and make these snapshots available to peer reviewers can **assist examination of manuscripts**. First, a visual comparison of the display files created by the authors and the ERS itself. Second, the UI bindings to interact with specific parts of the workflow. Third, the substitution of individual files in an ERC with files from a second ERC, or, in the future, with locally available files, enables creation new workflows and even deeper examination. The ERS can thereby assist the human, who needs to be in the loop to make the judgement call about how close something has to be to the original result to be deemed a reproduction, i.e., a margin of acceptable discordance or "zone of reproducibility" that helps to separate reproducibility from validity (ter Riet *et al.*, 2019). Hinsen (2018) distinguishes reproducibility as a software challenge, whereas a Human-Computer Interaction (HCI) perspective focuses on usage and reasoning, which is more important for verifiability In the same sense, UI bindings aide verification on the basis of a reproducible computation.

A part of the potential for assisting researchers that is largely untapped is the area of research discovery based on ERC. While the search endpoint of the API had been implemented using a powerful search index, Elasticsearch, the support was dropped because (a) it made the reproducibility service development and installation more complex, and (b) discovery through the reproducibility service is not a long-term solution as it only has short term storage of ERCs. Leveraging the connections between the ERC's parts and the exposure of main document and software stack for search and discovery should be placed at the repositories storing ERCs.

The **snapshot** is naturally a compromise between reliability—something works now for a specific purpose—and reusability—something can be extended, build upon. An *"active maintenance"* (Peer *et al.*, 2021), where workflows are constantly tested with new software releases and fixes are applied, would be more sustainable and more powerful to enable extensibility and reuse. However, the ERC as a snapshots but one that is in line with the common rhythm of term-based funding and paper publications as scientists' main means of communication (cf. Peng and Hicks, 2021).

The "closed" self-contained approach of an ERC has advantages for many workflows and can fit anything that works on a researcher's regular machine, but needs to be revisited with an increasing number of **big datasets, sensitive data, and complex computations**, e.g., in Remote Sensing or tracking data. Authors may already choose the most suitable level of detail and preprocessing needed to communicate their work effectively, and widely known and standardised steps may be skipped. Also, higher-level integrated data of more manageable size, e.g., analysis ready data [ARD; Frantz (2019)], may help to reduce ERC sizes. Going beyond steps that individual authors can take to support big data science, the idea to support an *allowlist* of trusted data repositories and computing services (cf. Nüst and Schutzeichel, 2017), which may be contacted by ERCs during creation and execution through a controlled network channel, is currently under development. For example, a journal may allow a collaborating repository or a reliable open computing infrastructure to be used by an ERC's workflow. The long-term feasibility could be improved if a journal critically picks these services and prefers open APIs, such as openEO for integrating external computing resources (Schramm *et al.*, 2021). These allowed connections must be made transparent in the ERC configuration to enable reproducibility services to be able to decide if they can create or examine a particular compendium. Recording outside communication during the initial execution and replaying for future examinations could also be a way to create a backup of external resources, similar to a performance enhancing cache. The nature of secured communication leads to these backups being black boxes, though, and they are therefore challenging for open research and preservation. Moreover, external connections may also be a solution to the following problem: The ERC and ERS do not have a build-in option to handle privacy or sensitive data, though the file-based substitution mechanism could be extended to replace synthetic public data with protected real datasets. Furthermore, the ERS could be extended with existing approaches for controlled access both for during and after peer review (cf. Nüst and Pebesma, 2020).

At first glance, the ERS seems to be severely limited by

[27] https://www.orcid.org/
[28] https://o2r.info/api/evaluation/load_test.html

the focus on **R Markdown** for the main file and HTML for the display file. Yet, R is the lingua franca of statistics and more and more used, but more importantly, R Markdown as common ground format for reproducible research is second to none when it comes to creating publication ready display documents, including citation management, templates, and both Web and print output formats (i.e., PDF), and transparency due to its plain text nature[29]. Templates for R Markdown could be provided by publishers though today are mostly community maintained (Allaire *et al.*, 2021b). However, adopting R Markdown as the core internal format may be too high of a hurdle for publishers, despite the problems that copy-editing poses for detailed reproducibility. Publisher-led approach such as ERA (Guizzardi *et al.*, 2021) that connect computational notebooks with standardised publishing formats could be easier to adopt, but lack some of the ERS's features. R Markdown also supports more programming languages than just R and, if nothing else works, a quite short and simple R Markdown notebook could be used as a wrapper for starting the actual process. Such wrapping may even be automated (cf. Glatard *et al.*, 2018) and templates can lower entry barriers. Even authors used to common word processors can participate in collaborations thanks to round-trip conversion tools with support for tracking changes using the prototypical `redoc` package (Ross, 2021).

6.4. Extent of capturing and ERCs' lifespan. The ERC captures all building blocks of a given piece of research. It clearly distinguishes between workflow specific files and the required runtime environment through its concept of nested containers. The ERC specifically attempts to capture relevant metadata for reproducibility, such as authors, the used libraries, or the UI bindings, and provides these metadata in multiple encodings. Also, not only the extent but also where parts are captured are crucial for reassuring accessibility. In the ERC, the actual workflow scripts and data are captured in the outer container because data is more long-lived than software[30] and it will be accessible even when the ERS and the inner container break. The ERS procedures allow to capture these detailed metadata with very limited user interaction, e.g., the metadata extraction capabilities for geospatial extent. Nevertheless, the interaction with the actual code session, which is used to capture the computing environment, is yet to be tapped for even better metadata. The inner container does explicitly not capture the operating system kernel. This limitation is acceptable—the kernel almost never introduces breaking changes. Furthermore, the ERC does not capture hardware, which makes sense, but it should better document the required hardware. Containers can very well be connected to accelerated computing infrastructures, such as GPUs (Haydel *et al.*, 2015), and the ERC configuration file should document this.

The limitations of the ERC's self-containedness were discussed above. With respect to the extent of capturing, the ERS could be enhanced to support the often service-based GIScience and geospatial data science by not only containing a single runtime environment for the workflow, but by including **multiple containers** for running the required APIs. These containers would have to be orchestrated, e.g., using `docker-compose` (Docker Inc., 2019), for ERC examination. Examples are scientific data storage and processing capabilities using services such as SciDB (Appel *et al.*, 2018) or OGC WPS implementations (Díaz *et al.*, 2008). While many of these services use geospatial libraries that could also be directly used in a workflow, capturing them as-is and keeping the client-side workflow code could reduce overheads for authors. How much this could be automated would depend on the openness of the used third party services, but manual ERC creation seems likely to be required. Furthermore, limitations concerning scalability might arise though data subsets for demonstration could mitigate this problem.

Regarding the ERC's lifespan, making research reproducible forever is not a wise goal. The lifespan is discussed here disregarding general ignorance of how quickly digital resources and free services may decay or disappear[31]. First, we cannot imagine today how computers will look like in 50 or 60 years. Science historians might still find a lot of valuable information in ERC, though. Second, even though some software (e.g., FORTRAN, GNU Make) has been around a long time, for the majority of research workflows it is reasonable to assume that after not being actively maintained (Peer *et al.*, 2021) for a while, a re-implementation based on the logic, which still readable within the source code, is more feasible than making a workflow executable again. At the same time, we do not expect pieces of software that are relevant and useful to simply disappear within a few years and only be preserved in ERCs. Therefore, the benchmark should be whether a snapshot of an often fragile software stack is executable for around the same time that is currently required for data to be kept available—around 10 years. We think the ERC and the ERS, both using current containerisation technology, can achieve that and an organisation (e.g., a publisher) which bases their workflow on ERCs could reasonably support a software system for at least that time frame. The longevity of ERS and ERC could be increased with a specialised container runtime that may reduce the feature set but focuses on long-term execution of containers. At this point, however, this assumption cannot be tested but should be checked in a few years. Then, "old" ERCs could be revisited to learn more about the preservation of computational workflows, e.g., how to ensure the "deep integrity" of fully containerised workflows. ERCs could be recreated regularly with current versions of the computing environment (re-capturing of the inner container) in a fully automatic way to identify both when dependencies and when the infrastructure breaks. We acknowledge a half life of computations and "exact repeatability", but the medium term executability of ERC is already a huge improvement over the current state of

[29] The second broadly used notebook format, Jupyter Notebook, is actively developing similar capabilities, e.g., using Jupytext (https://jupytext.readthedocs.io/), nbconvert (https://nbconvert.readthedocs.io/), and Jupyter Book (Executable Books Community, 2020). With these tools, the ERC concept of transparent main document and display file could be implemented.

[30] cf. http://www.activepapers.org/

[31] This XKCD comic illustrates the fragility of what we just assume will still work next year: https://xkcd.com/1909/.

declining availability already for data (Vines *et al.*, 2014).

Peng (2017) suggests to introduce limiting principles so that practical implications do not break the idea reproducibility. He discusses the audience (*Reproducible for Whom?*) and time span (*For How Long?*) and comes up with the idea of an endowment for reproducible publications with an author pays model, which would fit grant-based research because of a single payment. Peng's back of the napkin calculation for data storage of just 10GB alone easily reaches costs higher than many of today's APCs. The same considerations need to be explored for ERCs, and reasonable limits may very well be required for a widespread adoption.

6.5. ERCs in the spotlight. In this section, we critically discuss the ERC concept and the ERS implementation against a number of scales and terms for reproducibility. The classifications are ordered by year of publication and stem from all scientific disciplines.

Vandewalle *et al.* (2009) distinguish *six degrees of reproducibility*, of which an ERC could reach the highest level, 5, because an independent researcher can use the ERS as a free tool and with minimal effort. The requirement to spend "at most 15 min", however, depends on the packaged data and method, and the author's decision whether to package a reduced example or not.

Peng (2011) defines the *spectrum of reproducibility* as ranging from the irreproducible "publication only" to a gold standard of fully linked executable code and data. The ERC reaches the gold standard. When code and data are linked and executable, one may zoom into the spectrum and define a *spectrum of executability* within the highly reproducible workflows. This position on this sub-spectrum is determined through time since ERC creation, workflow complexity, and reviewers expertise—all at the same time. In practice, the executability may at a minimum start with a README file, which puts the highest burden on the reviewer. Increasingly more accessible practices would be a computational notebook, a research compendium, and finally an ERC, which puts increasing burden on the author while easing executability for the reviewer.

Gavish and Donoho (2012) describe three *"Dream Applications"* that would be possible if verifiable computational research (VCR) would be adopted. The ERC enables all three applications. It indirectly supports *Search* for research that uses a specific dataset or code and *Amalgate* for fusioning data and results, as data, code, and results are contained and could be indexed. The UI bindings and the substitution mechanism are a realisation of the *Tweak* application to interact and experiment with computational results.

Zhao *et al.* (2012) investigate decay of computational workflows over time regarding their re-execution and reproduction. They classify causes for workflow decay into four categories, all of which can be mitigated effectively by the ERC (as well as by their own tool). The ERC prohibits *volatile third-party resources*, *missing example data*, *missing execution environment*, and *insufficient descriptions about workflows*, because of the captured building blocks and self-containedness.

Stodden *et al.* (2013) devised a five-level taxonomy for computational research, classifying it as reviewable, replicable, confirmable, auditable, and open/reproducible. They also define the terms verification and validation. Research published as an ERC reaches the highest level of *open and reproducible research*, because it demands full openness for a fully available auditable workflow, and provides *verification*, because it allows to check if there are no errors in the code, and is thereby a support for *validation* by other researchers.

Thain *et al.* (2015) describe techniques for keeping software and computing environments executable and list a number of objectives for digital preservation. These techniques are presented between the two extremes of "preserving the mess" and "encouraging cleanliness". We place the ERC between those extremes. The outside packaging is quite clean for the execution of the full workflow, and UI bindings document some configurable parameters, however, the ERC is far from the details captured by workflow engines such as Umbrella (Meng and Thain, 2015) or Taverna (Wolstencroft *et al.*, 2013). The automated creation of containers for the runtime environment mitigates some of the shortcomings Thain et al. describe for virtual machines and container technology as "messy", but the ERC cannot capture distributed systems (machines, file systems). Regarding the preservation objectives, the ERC focuses on the *Identical Verification* verifying the same software and data lead to the same results. The execution within the ERS also realises a *New Environment Verification*, especially if the author does not provide a recipe for the inner container. The ERS does not allow to update the computing environment for *New Software Verification*, but the substitution can, within limits, be used for *Extension to New Data* respectively .. *New Software*.

Benureau and Rougier (2018) define *five ordered characteristics for useful code* in a scientific publication: it should be re-runnable, repeatable, reproducible, reusable, and replicable. The ERC can fulfil all these requirements through its self-contained yet transparent properties, though the author must still carefully set-up a workflow to not fall into any traps, e.g., with randomness, and enable reuse, e.g., with documentation, modularisation, or ease of configuration. The characteristics are achieved in part because different parties execute the code, the author and the ERS, and the ERS is designed for peer review processes, providing more eyes on the code and data. The ERC surely provides the details that are often missing from the manuscript itself and can thereby support replication.

Chen *et al.* (2019) define **Guiding principles towards reproducibility** for individual researchers or research groups, but the principles are transferable to a reproducibility infrastructure. The ERC clearly defines a *reproducibility goal*: package a workflow so that it can enable evaluation during peer review. However, it does not required *incorporate best practices early*, as it only requires a reproducible workflow at the time of submission, and admittedly creates a new platform instead of extending existing ones to be able to innovate. One would hope though that the expectation to submit an ERC should lead to adopting reproducibility practices early in projects. The ERC and ERS do require *structure* to make knowledge both human and machine readable and *capture content and workflows* well. The last three principles are rather cultural goals that could be pursued with the help of ERC and ERS.

Oliveira *et al.* (2020) describe an approach to evaluate software systems for reproducible software artefacts. Their *reproducibility pyramid* has 7 levels in three main categories: accessibility, executability, and interactive. The ERC and ERS enable all these levels, though only only binaries of the runtime environment are preserved in the inner container, which Oliveira et al. see as a risk, and the interactivity is focused on UI bindings and substitution, but does not provide a full development environment.

7. Conclusion & future work

The functionality of ERCs and the connection with crucial parts for scientific infrastructure has been demonstrated based on the reproducibility service and selected workflows. The user interface and service implementation can lower the barriers to share snapshots of research workflows for review and reading, and they can be integrated in a scholarly publication process. At least the artefacts of a reproducibility package can be preserved, for a time frame suitable to improve understanding and collaboration of relatively recently published results. ERCs and the designed infrastructure could also be connected to more radical changes in publishing practices. For example, piecemeal publication, review and publishing approaches like Octopus (Octopus team, 2020) and other evolutions in academic publishing (Tennant *et al.*, 2019), disruptive ways to distribute and review research such as Academic Torrents (Cohen and Lo, 2014) or overlay journals (Brown, 2010), and also novel ways of presenting, collaborating, and interacting with research outputs (Kray *et al.*, 2019). It remains to be seen if the technical and organisational innovations can benefit from each other or are better introduced successively.

However, a broad uptake was not achieved yet and the open research challenges summarised a few years ago by Freire *et al.* (2016) and Thain *et al.* (2015) are far from being answered today, though the ERS and ERC can contribute to addressing them. The slow pace of change can attributed to the many moving parts for adopting to more reproducible and transparent, such as author guidelines, researcher skills, editorial and review procedures, and publishing systems. This makes it really challenging for publishers to innovate, though their options to promote reproducibility are widely discussed (e.g., Hrynaszkiewicz, 2020; Eglen *et al.*, 2018). However, existing pilots yield promising results with a strong institutional support (Guizzardi *et al.*, 2021; Hawkins, 2019). The complexity of cultural change also slows down seemingly small but possibly impactful changes, such as establishing "reproducibility package" a first level citizen or type of research output in databases such as CrossRef or repositories such as Zenodo. This would give recognition and aide discovery of both data and software. Nevertheless, the presented software does provide a basis for further testing with stakeholders, as to improve the understanding of the remaining barriers for individuals (e.g., authors, reviewers, editors) and organisations (e.g, publishers, scholarly societies, scientific communities). The technical solutions for reproducible publications and transparent reviews can thereby help to support change in community practices and norms. These tests can go hand in hand with other solutions to supersede PDF papers, such as peer-reviewed Jupyter Notebooks[32], and with support offerings for reproducible research, e.g., by academic libraries (Sayre and Riegelman, 2019).

With respect to the further development of the technologies, the o2r project aims to realise a tight integration of the ERC with Open Journal Systems[33] (OJS). Usage in other publishing software platforms would strengthen the validation of the concepts and implementation, however, a more realistic step-by-step approach could be to use ERCs in specialised workflow review and execution processes as part of "regular" peer review (Nüst and Eglen, 2021). For a scalable infrastructure, existing tools for orchestration of ERC creation and examination sessions, such as Kubernetes (Wikipedia contributors, 2021d) or BinderHub (Project Jupyter *et al.*, 2018), could be used thanks to the fully containerised approach of both the reproducibility service implementation and the runtime environment of the ERC. While all specifications and implementations of the ERC Web service are open, the creation of reusable tools in different languages to more directly work with ERCs, e.g., validation and inspection functions in R or Python, and finding external collaborators to improve the specifications, e.g., by creating a formal schema file for the ERC configuration file format, would benefit uptake and usability for developers.

The biggest barrier remains the question of who takes responsibility to enable and finance computational reproducibility for scientific papers by providing infrastructure—a problem that is not even solved for financing possibly less dynamic and demanding infrastructures for data (Tennant *et al.*, 2019; Nature Editorial, 2017). There is a need to better integrate different research outputs and to convince funders and journals that they should request and better support openness and reproducibility as defaults (EOSC Executive Board Working Group (WG) Architecture Task Force (TF) SIRS, 2020; Porubsky *et al.*, 2021)—ERC and ERS can facilitate such goals. Is infrastructure for computational reproducibility a service offered by publishers as part of their business model, or will readers pay with every execution? If readily usable computing resources are given, it will become relevant to understand working practices on code execution during peer review (Nüst *et al.*, 2021). The increased openness and a growing number of individual practitioners as well as local to international initiatives around open science and reproducibility are promising drivers so that an open community-owned research in-

[32] https://www.earthcube.org/notebooks
[33] See blog post at https://o2r.info/2020/02/26/OJS-workshop-HD/.

frastructures will eventually strive[34]. *"Transparency can improve our practices even if no one actually looks, simply because we know that someone could look."* (Nosek *et al.*, 2012) Packaging research workflows and outputs as Executable Research Compendia can enhance existing scientific practices by eventually enabling infrastructures providing transparency, reproducibility, and reusability.

Acknowledgments. This work is supported by the project Opening Reproducible Research (o2r, o2r.info) funded by the German Research Foundation (DFG) under project numbers PE 1632/10-1, KR 3930/3-1 and TR 864/6-1 for phase 1 and PE 1632/17-1, KR 3930/8-1, and TR 864/12-1 for phase 2. We thank Edzer Pebesma for repeated feedback on this manuscript. The ideas and implementations presented in this article are based on the work of the whole o2r team: Markus Konkol, Marc Schutzeichel, Edzer Pebesma, Christian Kray, Holger Przibytzin, Jörg Lorenz, Rehan Chaudhary, Fabian Fermazin, Philipp Glahe, Juan Sebastian Garzón Alvarado, Laura Goulier, Matthias Hinz, Nick Jakuschona, Jan Koppe, Timm Kühnel, Torben Kraft, Lukas Lohoff, Tom Niers, Jan Suleiman, Yousef Qamaz. On top of that, the great online communities of Open Science and Reproducible Research have inspired and directly shaped many of the ideas in this work; we have tried to diligently record origins of earlier ideas and parallel developments.

References

Akhlaghi M, Infante-Sainz R, Roukema BF, Khellat M, Valls-Gabaud D, Baena-Gallé R (2021). "Toward Long-Term and Archivable Reproducibility." *Computing in Science Engineering*, **23**(3), 82–91. ISSN 1558-366X. doi:10.1109/MCSE.2021.3072860. Conference Name: Computing in Science Engineering.

Allaire J, Xie Y, McPherson J, Luraschi J, Ushey K, Atkins A, Wickham H, Cheng J, Chang W, Iannone R (2021a). *rmarkdown: Dynamic Documents for R.* R package version 2.8, URL https://github.com/rstudio/rmarkdown.

Allaire J, Xie Y, R Foundation, Wickham H, Journal of Statistical Software, Vaidyanathan R, Association for Computing Machinery, Boettiger C, Elsevier, Broman K, Mueller K, Quast B, Pruim R, Marwick B, Wickham C, Keyes O, Yu M, Emaasit D, Onkelinx T, Gasparini A, Desautels MA, Leutnant D, MDPI, Taylor and Francis, Öreden O, Hance D, Nüst D, Uvesten P, Campitelli E, Muschelli J, Hayes A, Kamvar ZN, Ross N, Cannoodt R, Luguern D, Kaplan DM, Kreutzer S, Wang S, Hesselberth J, Dervieux C (2021b). *rticles: Article Formats for R Markdown.* R package version 0.20, URL https://CRAN.R-project.org/package=rticles.

Appel M, Lahn F, Buytaert W, Pebesma E (2018). "Open and scalable analytics of large Earth observation datasets: From scenes to multidimensional arrays using SciDB and GDAL." *ISPRS Journal of Photogrammetry and Remote Sensing*, **138**, 47–56. ISSN 0924-2716. doi:10.1016/j.isprsjprs.2018.01.014. URL https://www.sciencedirect.com/science/article/pii/S0924271617300898.

Auer S, Haelterman N, Weissgerber T, Erlich JC, Susilaradeya D, Julkowska M, Gazda MA, Abitua A, Niraulu A, Shah A, Clyburne-Sherin A, Guiquel B, Alicea B, LaManna C, Ganguly D, Perkins EJ, Jambor H, Li IMH, Tsang J, Kamens J, Teytelman L, Paul M, Phuyal S, Schmelling N, Crisp P, Sarabipour S, Roy S, Bachle S, Tran MTK, Ford T, Steeves V, Ilangovan V, Schwessinger B, Jadavji N (2020). "Reproducibility for Everyone: A Community-Led Initiative with Global Reach in Reproducible Research Training." *Technical report*, OSF Preprints. doi:10.31219/osf.io/dxw67.

Benureau FCY, Rougier NP (2018). "Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions." *Frontiers in Neuroinformatics*, **11**. ISSN 1662-5196. doi:10.3389/fninf.2017.00069.

Boettiger C (2015). "An Introduction to Docker for Reproducible Research." *SIGOPS Oper. Syst. Rev.*, **49**(1), 71–79. ISSN 0163-5980. doi:10.1145/2723872.2723882.

Bouffler B (2019). "Keynote: Delivering on the promise of Research Computing." Gesellschaft für Informatik e.V. in TIB AV-PORTAL. https://doi.org/10.5446/42484#t=15:31,16:20 (time stamp 15:31; last accessed: 31 May 2021).

Brammer GR, Crosby RW, Matthews SJ, Williams TL (2011). "Paper Mâché: Creating Dynamic Reproducible Science." *Procedia Computer Science*, **4**, 658–667. ISSN 1877-0509. doi:10.1016/j.procs.2011.04.069.

Brown J (2010). "An introduction to overlay journals." *Report*, Repositories Support Project, UK. URL https://discovery.ucl.ac.uk/id/eprint/19081/.

Buckheit JB, Donoho DL (1995). "WaveLab and Reproducible Research." In A Antoniadis, G Oppenheim (eds.), *Wavelets and Statistics*, number 103 in Lecture Notes in Statistics, pp. 55–81. Springer New York. ISBN 978-0-387-94564-4 978-1-4612-2544-7. doi:10.1007/978-1-4612-2544-7_5.

Castleberry DG, Brandt SR, Löffler F (2013). "Inkling: An Executable Paper System for Reviewing Scientific Applications." In *2013 International Conference on Social Computing*, pp. 917–922. doi:10.1109/SocialCom.2013.142.

Chard K, Gaffney N, Jones MB, Kowalik K, Ludäscher B, McPhillips T, Nabrzyski J, Stodden V, Taylor I, Thelen T, Turk MJ, Willis C (2019). "Application of BagIt-Serialized Research Object Bundles for Packaging and Re-Execution of Computational Analyses." In *2019 15th International Conference on eScience (eScience)*, pp. 514–521. doi:10.1109/eScience.2019.00068.

Chen X, Dallmeier-Tiessen S, Dasler R, Feger S, Fokianos P, Gonzalez JB, Hirvonsalo H, Kousidis D, Lavasa A, Mele S, Rodriguez DR, imko T, Smith T, Trisovic A, Trzcinska A, Tsanaktsidis I, Zimmermann M, Cranmer K, Heinrich L, Watts G, Hildreth M, Iglesias LL, Lassila-Perini K, Neubert S (2019). "Open is not enough." *Nature Physics*, **15**(2), 113. ISSN 1745-2481. doi:10.1038/s41567-018-0342-2.

Chirigati F, Rampin R, Shasha D, Freire J (2016). "ReproZip: Computational Reproducibility With Ease." In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pp. 2085–2088. ACM, New York, NY, USA. ISBN 978-1-4503-3531-7. doi:10.1145/2882903.2899401.

Chuah J, Deeds M, Malik T, Choi Y, Goodall JL (2020). "Documenting Computing Environments for Reproducible Experiments." *Parallel Computing: Technology Trends*, pp. 756–765. doi:10.3233/APC200106. Publisher: IOS Press.

Cohen JP, Lo HZ (2014). "Academic Torrents: A Community-Maintained Distributed Repository." In *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment - XSEDE '14*, pp. 1–2. ACM Press, Atlanta, GA, USA. ISBN 978-1-4503-2893-7. doi:10.1145/2616498.2616528.

Cruz M, de Jonge H (2020). "Beyond mandates: For open science to become a norm, it must be recognised and rewarded." URL https://blogs.lse.ac.uk/impactofsocialsciences/2020/12/01/beyond-mandates-for-open-science-to-become-a-norm-it-must-be-recognised-and-rewarded/.

Davenport JH, Grant J, Jones CM (2020). "Data Without Software Are Just Numbers." *Data Science Journal*, **19**(1), 3. ISSN 1683-1470. doi:10.5334/dsj-2020-003. URL http://datascience.codata.org/articles/10.5334/dsj-2020-003/.

David CH, Gil Y, Duffy CJ, Peckham SD, Venayagamoorthy SK (2016). "An introduction to the special issue on Geoscience Papers of the Future." *Earth and Space Science*, **3**(10), 2016EA000201. ISSN 2333-5084. doi:10.1002/2016EA000201.

Davison A (2012). "Automated Capture of Experiment Context for Easier Reproducibility in Computational Research." *Computing in Science Engineering*, **14**(4), 48–56. ISSN 1521-9615. doi:10.1109/MCSE.2012.41.

Davison AP, Mattioni M, Samarkanov D, Telenczuk B (2014). "Sumatra: A Toolkit for Reproducible Research." In V Stodden, F Leisch, RD Peng (eds.), *Implementing Reproducible Research*, Chapman & Hall/CRC The R Series, p. 448. Taylor & Francis. ISBN 978-1-4665-6159-5.

Docker Inc (2019). "Overview of Docker Compose." URL https://docs.docker.com/compose/.

Díaz L, Granell C, Gould M, Olaya V (2008). "An open service network for geospatial data processing." In *Proceedings of the academic track of the*

[34]For example, Repro4Everyone (https://repro4everyone.org/, Auer *et al.*, 2020), ReproHacks (https://reprohack.github.io/reprohack-hq/), and The Turing Way (https://www.turing.ac.uk/research/research-projects/turing-way-handbook-reproducible-data-science) on educating researchers, Invest in Open Infrastructure (IOI, https://investinopen.org/) for funding community-owned open technologies and systems for research and scholarship, or novel priorities and processes in funding schemes (Cruz and de Jonge, 2020).

2008 Free and Open Source Software for Geospatial (FOSS4G) Conference, pp. 410–420. Cape Town, South Africa. ISBN 978-0-620-42117-1. URL https://www.researchgate.net/profile/Laura-Diaz-75/publication/228655946_An_open_service_network_for_geospatial_data_processing/links/0deec5195f4bdb0f86000000/An-open-service-network-for-geospatial-data-processing.pdf.

Eglen SJ, Mounce R, Gatto L, Currie AM, Nobis Y (2018). "Recent developments in scholarly publishing to improve research practices in the life sciences." *Emerging Topics in Life Sciences*, **2**(6), 775–778. ISSN 2397-8554, 2397-8562. doi:10.1042/ETLS20180172.

Emsley I, De Roure D (2018). "A Framework for the Preservation of a Docker Container | International Journal of Digital Curation." *International Journal of Digital Curation*, **12**(2). doi:10.2218/ijdc.v12i2.509.

EOSC Executive Board Working Group (WG) Architecture Task Force (TF) SIRS (2020). "Scholarly infrastructures for research software: report from the EOSC Executive Board Working Group (WG) Architecture Task Force (TF) SIRS." *Technical report*, Edited by the EOSC Executive Board. doi:10.2777/28598.

Executable Books Community (2020). "Jupyter Book." doi:10.5281/zenodo.4539666.

Frantz D (2019). "FORCELandsat + Sentinel-2 Analysis Ready Data and Beyond." *Remote Sensing*, **11**(9), 1124. doi:10.3390/rs11091124.

Freire J, Fuhr N, Rauber A (2016). "Reproducibility of Data-Oriented Experiments in e-Science (Dagstuhl Seminar 16041)." *Dagstuhl Reports*, **6**(1), 108–159. ISSN 2192-5283. doi:10.4230/DagRep.6.1.108.

Gavish M, Donoho D (2012). "Three Dream Applications of Verifiable Computational Results." *Computing in Science Engineering*, **14**(4), 26–31. ISSN 1558-366X. doi:10.1109/MCSE.2012.65. Conference Name: Computing in Science Engineering.

Gentleman R, Lang DT (2007). "Statistical Analyses and Reproducible Research." *Journal of Computational and Graphical Statistics*, **16**(1), 1–23. ISSN 1061-8600. doi:10.1198/106186007X178663.

Ghoshal D, Bianchi L, Essiari A, Beach M, Paine D, Ramakrishnan L (2021). "Science Capsule - Capturing the Data Life Cycle." *Journal of Open Source Software*, **6**(62), 2484. ISSN 2475-9066. doi:10.21105/joss.02484. URL https://joss.theoj.org/papers/10.21105/joss.02484.

Gil Y, David C, Demir I, Essawy B, Fulweiler R, Goodall J, Karlstrom L, Lee H, Mills H, Oh J, Pierce S, Pope A, Tzeng M, Villamizar S, Yu X (2016). "Toward the Geoscience Paper of the Future: Best practices for documenting and sharing research from data to software to provenance." *Earth and Space Science*, **3**(10), 2015EA000136. doi:10.1002/2015EA000136.

Glatard T, Kiar G, Aumentado-Armstrong T, Beck N, Bellec P, Bernard R, Bonnet A, Brown ST, Camarasu-Pop S, Cervenansky F, Das S, Ferreira da Silva R, Flandin G, Girard P, Gorgolewski KJ, Guttmann CRG, Hayot-Sasson V, Quirion PO, Rioux P, Rousseau MÉ, Evans AC (2018). "Boutiques: A Flexible Framework to Integrate Command-Line Applications in Computing Platforms." *GigaScience*, **7**(giy016). ISSN 2047-217X. doi:10.1093/gigascience/giy016.

González Ávalos E (2020). "Good overall quality, additional discussion on certain points would be desirable." *other*, Geosciences Marine Geology/essd-2020-22. doi:10.5194/essd-2020-22-RC1.

Gronenschild E, Habets P, Jacobs H, Mengelers R, Rozendaal N, van Os J, Marcelis M (2012). "The effects of FreeSurfer version, workstation type, and Macintosh operating system version on anatomical volume and cortical thickness measurements." *PLoS One*, **7**(6), e38234. doi:10.1371/journal.pone.0038234.

Guizzardi G, Bentley N, Maciocci G (2021). "Announcing the next phase of Executable Research Articles." Publisher: eLife Sciences Publications Limited, URL https://elifesciences.org/labs/a04d2b80/announcing-the-next-phase-of-executable-research-articles.

Hardt D (2012). "The OAuth 2.0 Authorization Framework." RFC 6749. doi:10.17487/RFC6749. URL https://rfc-editor.org/rfc/rfc6749.txt.

Hardwicke TE, Mathur MB, MacDonald K, Nilsonne G, Banks GC, Kidwell MC, Mohr AH, Clayton E, Yoon EJ, Tessler MH, Lenne RL, Altman S, Long B, Frank MC (2018). "Data Availability, Reusability, and Analytic Reproducibility: Evaluating the Impact of a Mandatory Open Data Policy at the Journal Cognition." *Royal Society Open Science*, **5**(8), 180448. ISSN 2054-5703. doi:10.1098/rsos.180448.

Hawkins E (2019). "What we have learnt testing container-platforms for peer review and publication of code : Of Schemes and Memes Blog." URL http://blogs.nature.com/ofschemesandmemes/2019/10/09/what-we-have-learnt-testing-container-platforms-for-peer-review-and-publication-of-code.

Haydel N, Madey G, Gesing S, Dakkak A, de Gonzalo SG, Taylor I, Hwu WmW (2015). "Enhancing the usability and utilization of accelerated architectures via docker." In *Proceedings of the 8th International Conference on Utility and Cloud Computing*, UCC '15, pp. 361–367. IEEE Press, Limassol, Cyprus. ISBN 978-0-7695-5697-0.

Heroux MA (2015). "Editorial: ACM TOMS Replicated Computational Results Initiative." *ACM Transactions on Mathematical Software*, **41**(3), 13:1–13:5. ISSN 0098-3500. doi:10.1145/2743015.

Hinsen K (2015). "ActivePapers: a platform for publishing and archiving computer-aided research." *F1000Research*, **3**, 289. ISSN 2046-1402. doi:10.12688/f1000research.5773.3.

Hinsen K (2018). "Verifiability in computer-aided research: the role of digital scientific notations at the human-computer interface." *PeerJ Computer Science*, **4**, e158. ISSN 2376-5992. doi:10.7717/peerj-cs.158.

Hrynaszkiewicz I (2020). "Publishers Responsibilities in Promoting Data Quality and Reproducibility." In A Bespalov, MC Michel, T Steckler (eds.), *Good Research Practice in Non-Clinical Pharmacology and Biomedicine*, Handbook of Experimental Pharmacology, pp. 319–348. Springer International Publishing, Cham. ISBN 978-3-030-33656-1. doi:10.1007/164_2019_290.

Jimenez I, Sevilla M, Watkins N, Maltzahn C, Lofstead J, Mohror K, Arpaci-Dusseau A, Arpaci-Dusseau R (2017). "The Popper Convention: Making Reproducible Systems Evaluation Practical." In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 1561–1570. doi:10.1109/IPDPSW.2017.157.

Katz DS, Chue Hong NP, Clark T, Muench A, Stall S, Bouquin D, Cannon M, Edmunds S, Faez T, Feeney P, Fenner M, Friedman M, Grenier G, Harrison M, Heber J, Leary A, MacCallum C, Murray H, Pastrana E, Perry K, Schuster D, Stockhause M, Yeston J (2021). "Recognizing the value of software: a software citation guide." *F1000Research*, **9**, 1257. ISSN 2046-1402. doi:10.12688/f1000research.26932.2.

Knuth DE (1984). "Literate Programming." *Comput. J.*, **27**(2), 97–111. ISSN 0010-4620. doi:10.1093/comjnl/27.2.97.

Konkol M, Kray C, Pfeiffer M (2019a). "Computational reproducibility in geoscientific papers: Insights from a series of studies with geoscientists and a reproduction study." *International Journal of Geographical Information Science*, **33**(2), 408–429. ISSN 1365-8816. doi:10.1080/13658816.2018.1508687.

Konkol M, Kray C, Suleiman J (2019b). "Creating Interactive Scientific Publications Using Bindings." *Proceedings of the ACM on Human-Computer Interaction*, **3**(EICS), 16:1–16:18. doi:10.1145/3331158.

Konkol M, Nüst D, Goulier L (2020). "Publishing computational research - a review of infrastructures for reproducible and transparent scholarly communication." *Research Integrity and Peer Review*, **5**(1), 10. ISSN 2058-8615. doi:10.1186/s41073-020-00095-y.

Kray C, Pebesma E, Konkol M, Nüst D (2019). "Reproducible Research in Geoinformatics: Concepts, Challenges and Benefits (Vision Paper)." volume 142 of *Leibniz International Proceedings in Informatics (LIPIcs)*, p. 8:1–8:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany. doi:10.4230/LIPIcs.COSIT.2019.8. URL http://drops.dagstuhl.de/opus/volltexte/2019/11100.

Kunze JA, Littman J, Madden L, Scancella J, Adams C (2018). "The BagIt File Packaging Format (V1.0)." RFC 8493. doi:10.17487/RFC8493. URL https://rfc-editor.org/rfc/rfc8493.txt.

Liu DM, Salganik MJ (2019). "Successes and Struggles with Computational Reproducibility: Lessons from the Fragile Families Challenge." *Socius*, **5**, 2378023119849803. ISSN 2378-0231. doi:10.1177/2378023119849803.

Marwick B (2015). "How Computers Broke Science – and What We Can Do to Fix It." http://theconversation.com/how-computers-broke-science-and-what-we-can-do-to-fix-it-49938.

Marwick B, Pilaar Birch SE (2018). "How researchers can solve the bottle-opener problem with compute capsules." URL https://www.cambridge.org/core/blog/2018/07/30/how-researchers-can-solve-the-bottle-opener-problem-with-compute-capsules/.

Mecum B, Jones MB, Vieglais D, Willis C (2018). "Preserving Reproducibility: Provenance and Executable Containers in DataONE Data Packages." In *2018 IEEE 14th International Conference on E-Science (e-Science)*, pp. 45–49. ISSN null. doi:10.1109/eScience.2018.00019.

Meng H, Thain D (2015). "Umbrella: A Portable Environment Creator for

Reproducible Computing on Clusters, Clouds, and Grids." In *Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing*, VTDC '15, pp. 23–30. ACM, New York, NY, USA. ISBN 978-1-4503-3573-7. doi:10.1145/2755979.2755982.

Miller G (2006). "A Scientist's Nightmare: Software Problem Leads to Five Retractions." *Science*, **314**(5807), 1856–1857. ISSN 0036-8075, 1095-9203. doi:10.1126/science.314.5807.1856.

Minghini M, Mobasheri A, Rautenbach V, Brovelli MA (2020). "Geospatial openness: from software to standards & data." *Open Geospatial Data, Software and Standards*, **5**(1), 1. ISSN 2363-7501. doi:10.1186/s40965-020-0074-y.

Molenaar G, Makhathini S, Girard JN, Smirnov O (2018). "KlikoThe scientific compute container format." *Astronomy and Computing*, **25**, 1–9. ISSN 2213-1337. doi:10.1016/j.ascom.2018.08.003.

Nature Editorial (2017). "Empty rhetoric over data sharing slows science." *Nature News*, **546**(7658), 327. doi:10.1038/546327a.

Nosek BA, Spies JR, Motyl M (2012). "Scientific Utopia II. Restructuring Incentives and Practices to Promote Truth Over Publishability." *Perspectives on Psychological Science*, **7**(6), 615–631. ISSN 1745-6916, 1745-6924. doi:10.1177/1745691612459058.

Nüst D, Granell C, Hofer B, Konkol M, Ostermann FO, Sileryte R, Cerutti V (2018). "Reproducible Research and GIScience: An Evaluation Using AGILE Conference Papers." *PeerJ*, **6**, e5072. ISSN 2167-8359. doi:10.7717/peerj.5072.

Nüst D, Pebesma E (2020). "Practical Reproducibility in Geography and Geosciences." *Annals of the American Association of Geographers*, **111**(5), 1–11. doi:10.1080/24694452.2020.1806028.

Nüst D (2018). "Reproducibility Service for Executable Research Compendia: Technical Specifications and Reference Implementation." *Technical report*. doi:10.5281/zenodo.2203843.

Nüst D, Eglen S (2021). "CODECHECK: an Open Science initiative for the independent execution of computations underlying research articles during peer review to improve reproducibility." *F1000Research*, **10**, 253. doi:10.12688/f1000research.51738.1.

Nüst D, Hinz M (2019). "containerit: Generating Dockerfiles for reproducible research with R." *Journal of Open Source Software*, **4**(40), 1603. doi:10.21105/joss.01603.

Nüst D, Konkol M, Pebesma E, Kray C, Schutzeichel M, Przibytzin H, Lorenz J (2017). "Opening the Publication Process with Executable Research Compendia." *D-Lib Magazine*, **23**(1/2). ISSN 1082-9873. doi:10.1045/january2017-nuest.

Nüst D, Schutzeichel M (2017). "An Architecture for Reproducible Computational Geosciences." In *Poster abstracts of AGILE 2017*. Wageningen, The Netherlands. doi:10.5281/zenodo.1478542.

Nüst D, Seibold H, Eglen S, Schulz-Vanheyden L (2021). "Code Execution in Peer Review." doi:10.17605/osf.io/x32nc.

Nüst D, Sochat V, Marwick B, Eglen SJ, Head T, Hirst T, Evans BD (2020). "Ten simple rules for writing Dockerfiles for reproducible data science." *PLOS Computational Biology*, **16**(11), e1008316. ISSN 1553-7358. doi:10.1371/journal.pcbi.1008316.

Octopus team (2020). "More about Octopus." URL https://science-octopus.org/about.

Oliveira L, Wilkinson D, Mossé D, Childers B (2018). "Supporting Long-term Reproducible Software Execution." In *Proceedings of the First International Workshop on Practical Reproducible Evaluation of Computer Systems*, P-RECS'18, pp. 1–6. Association for Computing Machinery, New York, NY, USA. ISBN 978-1-4503-5861-3. doi:10.1145/3214239.3214245.

Oliveira L, Wilkinson D, Mossé D, Childers BR (2020). "Stimulating Reproducible Software Artifacts." In *Proceedings of the 3rd International Workshop on Practical Reproducible Evaluation of Computer Systems*, P-RECS '20, pp. 3–7. Association for Computing Machinery, New York, NY, USA. ISBN 978-1-4503-7977-9. doi:10.1145/3391800.3398177.

Pasquier T, Lau MK, Han X, Fong E, Lerner BS, Boose ER, Crosas M, Ellison AM, Seltzer M (2018). "Sharing and Preserving Computational Analyses for Posterity with encapsulator." *Computing in Science Engineering*, **20**(4), 111–124. ISSN 1558-366X. doi:10.1109/MCSE.2018.042781334.

Pebesma E (2013). "Earth and Planetary Innovation Challenge (EPIC) submission "One-Click-Reproduce"." URL http://pebesma.staff.ifgi.de/epic.pdf.

Peer L, Orr LV, Coppock A (2021). "Active Maintenance: A Proposal for the Long-Term Computational Reproducibility of Scientific Results." *PS: Political Science & Politics*, pp. 1–5. ISSN 1049-0965, 1537-5935. doi:10.1017/S1049096521000366. Publisher: Cambridge University Press.

Peng R (2017). "Reproducible Research Needs Some Limiting Principles." URL https://simplystatistics.org/2017/02/01/reproducible-research-limits/.

Peng RD (2011). "Reproducible Research in Computational Science." *Science*, **334**(6060), 1226–1227. ISSN 0036-8075, 1095-9203. doi:10.1126/science.1213847.

Peng RD, Hicks SC (2021). "Reproducible Research: A Retrospective." *Annual Review of Public Health*, **42**(1), 79–93. ISSN 0163-7525. doi:10.1146/annurev-publhealth-012420-105110.

Piwowar H (2013). "Value all research products." *Nature*, **493**, 159. doi:10.1038/493159a.

Porubsky V, Smith L, Sauro HM (2021). "Publishing reproducible dynamic kinetic models." *Briefings in Bioinformatics*, **22**(3). ISSN 1477-4054. doi:10.1093/bib/bbaa152.

Project Jupyter, Bussonnier M, Forde J, Freeman J, Granger B, Head T, Holdgraf C, Kelley K, Nalvarte G, Osheroff A, Pacer M, Panda Y, Perez F, Ragan-Kelley B, Willing C (2018). "Binder 2.0 - Reproducible, Interactive, Sharable Environments for Science at Scale." *Proceedings of the 17th Python in Science Conference*, pp. 113–120. doi:10.25080/Majora-4af1f417-011.

Rechert K, Liebetraut T, Kombrink S, Wehrle D, Mocken S, Rohland M (2017). "Preserving Containers." In J Kratzke, V Heuveline (eds.), *Forschungsdaten managen*, pp. 143–151. Heidelberg. ISBN 978-3-946531-75-3. doi:10.11588/heibooks.285.377.

Ross N (2021). *redoc: Reversible Reproducible Documents*. R package version 0.1.0.9000, URL https://github.com/noamross/redoc.

Sayre F, Riegelman A (2019). "Replicable Services for Reproducible Research: A Model for Academic Libraries." *College & Research Libraries*, **80**(2), 260. doi:10.5860/crl.80.2.260.

Schramm M, Pebesma E, Milenkovi M, Foresta L, Dries J, Jacob A, Wagner W, Mohr M, Neteler M, Kadunc M, Miksa T, Kempeneers P, Verbesselt J, GöSSwein B, Navacchi C, Lippens S, Reiche J (2021). "The openEO APIHarmonising the Use of Earth Observation Cloud Services Using Virtual Data Cube Functionalities." *Remote Sensing*, **13**(6), 1125. doi:10.3390/rs13061125.

Stodden V, Bailey DH, Borwein J, LeVeque RJ, Rider B, Stein W (2013). "Setting the Default to Reproducible: Reproducibility in Computational and Experimental Mathematics." *Technical report*, The Institute for Computational and Experimental Research in Mathematics. Workshop website with full list of workshop participants: https://icerm.brown.edu/topical_workshops/tw12-5-rcem/ This report was developed collaboratively by the ICERM workshop participants, and compiled and edited by the organizers., URL https://icerm.brown.edu/topical_workshops/tw12-5-rcem/icerm_report.pdf.

Stodden V, Miguez S, Seiler J (2015). "ResearchCompendia.org: Cyberinfrastructure for Reproducibility and Collaboration in Computational Science." *Computing in Science & Engineering*, **17**(1), 12–19. ISSN 1521-9615. doi:10.1109/MCSE.2015.18.

Stodden V, Seiler J, Ma Z (2018). "An empirical analysis of journal policy effectiveness for computational reproducibility." *Proceedings of the National Academy of Sciences*, **115**(11), 2584–2589. ISSN 0027-8424, 1091-6490. doi:10.1073/pnas.1708290115.

Tennant JP, Crane H, Crick T, Davila J, Enkhbayar A, Havemann J, Kramer B, Martin R, Masuzzo P, Nobes A, Rice C, Rivera-López B, Ross-Hellauer T, Sattler S, Thacker PD, Vanholsbeeck M (2019). "Ten Hot Topics around Scholarly Publishing." *Publications*, **7**(2), 34. doi:10.3390/publications7020034.

ter Riet G, Storosum BW, Zwinderman AH (2019). "What is reproducibility?" *F1000Research*, **8**, 36. ISSN 2046-1402. doi:10.12688/f1000research.17615.1.

Thain D, Ivie P, Meng H (2015). "Techniques for Preserving Scientific Software Executions: Preserve the Mess or Encourage Cleanliness?" In *Proceedings of the 12th International Conference on Digital Preservation (iPres)*. doi:10.7274/R0CZ353M.

That DHT, Fils G, Yuan Z, Malik T (2017). "Sciunits: Reusable Research Objects." In *2017 IEEE 13th International Conference on e-Science (e-Science)*, pp. 374–383. doi:10.1109/eScience.2017.51.

Vandewalle P, Kovacevic J, Vetterli M (2009). "Reproducible research in signal processing." *IEEE Signal Processing Magazine*, **26**(3), 37–47. ISSN 1053-5888, 1558-0792. doi:10.1109/MSP.2009.932122.

Vines T, Albert AK, Andrew R, Débarre F, Bock D, Franklin M, Gilbert K, Moore JS, Renaut S, Rennison D (2014). "The Availability of Research Data Declines Rapidly with Article Age." *Current Biology*, **24**(1), 94–97. ISSN 0960-9822. doi:10.1016/j.cub.2013.11.014.

Whitehouse T (2019). "Making Reproducibility Reproducible." URL https://medium.com/gigantum/making-reproducibility-reproducible-7457d656680c.

Wikipedia contributors (2021a). "AppArmor." Page Version ID: 1027531383, URL https://en.wikipedia.org/w/index.php?title=AppArmor&oldid=1027531383.

Wikipedia contributors (2021b). "Bus factor." Page Version ID: 1024613010, URL https://en.wikipedia.org/w/index.php?title=Bus_factor&oldid=1024613010.

Wikipedia contributors (2021c). "Docker (software)." Page Version ID: 1019840030, URL https://en.wikipedia.org/w/index.php?title=Docker_(software)&oldid=1019840030.

Wikipedia contributors (2021d). "Kubernetes." Page Version ID: 1024839217, URL https://en.wikipedia.org/w/index.php?title=Kubernetes&oldid=1024839217.

Wikipedia contributors (2021e). "Make (software)." Page Version ID: 1016565702, URL https://en.wikipedia.org/w/index.php?title=Make_(software)&oldid=1016565702.

Wikipedia contributors (2021f). "OpenAPI Specification." Page Version ID: 1023136282, URL https://en.wikipedia.org/w/index.php?title=OpenAPI_Specification&oldid=1023136282.

Wikipedia contributors (2021g). "Unix philosophy." Page Version ID: 1022001416, URL https://en.wikipedia.org/w/index.php?title=Unix_philosophy&oldid=1022001416.

Wikipedia contributors (2021h). "Vagrant (software)." Page Version ID: 1014463164, URL https://en.wikipedia.org/w/index.php?title=Vagrant_(software)&oldid=1014463164.

Wikipedia contributors (2021i). "WebSocket." Page Version ID: 1028455012, URL https://en.wikipedia.org/w/index.php?title=WebSocket&oldid=1028455012.

Wilson G, Bryan J, Cranston K, Kitzes J, Nederbragt L, Teal TK (2017). "Good enough practices in scientific computing." *PLOS Computational Biology*, **13**(6), e1005510. ISSN 1553-7358. doi:10.1371/journal.pcbi.1005510.

Wolstencroft K, Haines R, Fellows D, Williams A, Withers D, Owen S, Soiland-Reyes S, Dunlop I, Nenadic A, Fisher P, Bhagat J, Belhajjame K, Bacall F, Hardisty A, Hidalga ANdl, Vargas MPB, Sufi S, Goble C (2013). "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud." *Nucleic Acids Research*, **41**(W1), W557–W561. ISSN 0305-1048, 1362-4962. doi:10.1093/nar/gkt328.

Xie Y, Allaire J, Grolemund G (2018). *R Markdown: The Definitive Guide*. Chapman and Hall/CRC, Boca Raton, Florida. ISBN 9781138359338, URL https://bookdown.org/yihui/rmarkdown.

Yan A, Huang C, Lee JS, Palmer CL (2020). "Cross-disciplinary data practices in earth system science: Aligning services with reuse and reproducibility priorities." *Proceedings of the Association for Information Science and Technology*, **57**(1), e218. ISSN 2373-9231. doi:https://doi.org/10.1002/pra2.218.

Youngdahl A, Ton-That DH, Malik T (2019). "SciInc: A Container Runtime for Incremental Recomputation." In *2019 15th International Conference on eScience (eScience)*, pp. 291–300. IEEE, San Diego, CA, USA. ISBN 978-1-72812-451-3. doi:10.1109/eScience.2019.00040.

Zhao J, Gomez-Perez JM, Belhajjame K, Klyne G, Garcia-Cuesta E, Garrido A, Hettne K, Roos M, De Roure D, Goble C (2012). "Why workflows break Understanding and combating decay in Taverna workflows." In *2012 IEEE 8th International Conference on E-Science*, pp. 1–9. doi:10.1109/eScience.2012.6404482.

imko T, Heinrich L, Hirvonsalo H, Kousidis D, Rodríguez D (2019). "REANA: A System for Reusable Research Data Analyses." *EPJ Web of Conferences*, **214**, 06034. ISSN 2100-014X. doi:10.1051/epjconf/201921406034.

6 Publishing computational research - a review of infrastructures for reproducible and transparent scholarly communication

Authors & contribution Markus Konkol, Daniel Nüst (15%), Laura Goulier

Venue *Research Integrity and Peer Review* 10.1186/s41073-020-00095-y

Date 10/2020

Licence Creative Commons Attribution 4.0 International (CC BY 4.0)

Repository https://github.com/o2r-project/reviewpaper

ERC https://o2r.uni-muenster.de/erc/dkYhi

Research Integrity and Peer Review

REVIEW Open Access

Publishing computational research - a review of infrastructures for reproducible and transparent scholarly communication

Markus Konkol[*], Daniel Nüst and Laura Goulier

Abstract

Background: The trend toward open science increases the pressure on authors to provide access to the source code and data they used to compute the results reported in their scientific papers. Since sharing materials reproducibly is challenging, several projects have developed solutions to support the release of executable analyses alongside articles.

Methods: We reviewed 11 applications that can assist researchers in adhering to reproducibility principles. The applications were found through a literature search and interactions with the reproducible research community. An application was included in our analysis if it **(i)** was actively maintained at the time the data for this paper was collected, **(ii)** supports the publication of executable code and data, **(iii)** is connected to the scholarly publication process. By investigating the software documentation and published articles, we compared the applications across 19 criteria, such as deployment options and features that support authors in creating and readers in studying executable papers.

Results: From the 11 applications, eight allow publishers to self-host the system for free, whereas three provide paid services. Authors can submit an executable analysis using Jupyter Notebooks or R Markdown documents (10 applications support these formats). All approaches provide features to assist readers in studying the materials, e.g., one-click reproducible results or tools for manipulating the analysis parameters. Six applications allow for modifying materials after publication.

Conclusions: The applications support authors to publish reproducible research predominantly with literate programming. Concerning readers, most applications provide user interfaces to inspect and manipulate the computational analysis. The next step is to investigate the gaps identified in this review, such as the costs publishers have to expect when hosting an application, the consideration of sensitive data, and impacts on the review process.

Keywords: Open reproducible research, Open science, Computational statistics, Scholarly communication

Background

In many scientific fields, the results of scientific articles can be based on computations, e.g., a statistical analysis implemented in R. For this type of research, publishing the used source code and data to adhere to "open reproducible research" (ORR) principles (i.e., public access to the code and data underlying the reported results [1]) seems simple. Nevertheless, several studies have concluded that papers rarely include or link to these materials [2, 3]. Reasons for that are manifold:

First, due to *technical challenges*, e.g., capturing the analyst's original computational environment, even

* Correspondence: m.konkol@uni-muenster.de
Institute for Geoinformatics, University of Münster, Münster, Germany

having accessible materials does not guarantee that results can be reproduced [4, 5]. Second, many authors hesitate to share their work because publishing erroneous papers can *damage an author's reputation* [6] as well as *trust* in science [7]. These perspectives, however, overlook the fact that engaging in open practices offers some career advantages [8, 9] and can help in identifying and correcting mistakes [10, 11].

As a result of authors not including their source code and underlying data, several further problems arise. For example, reviewers *cannot verify the results*, because without the code, they are required to understand the analysis just by reading the text [12]. Hence, finding errors in results is difficult and often impossible [6], raising the question of whether the traditional research article is suitable for conveying a complex computational analysis [13]. Additionally, other researchers working in similar areas *cannot continue building on existing work* but have to collect data and implement the analysis from scratch [14]. All these issues are also to society's disadvantage, as the public cannot benefit fully from publicly funded research [15].

Funding agencies, e.g., Horizon 2020, are increasingly requiring data and software management plans as part of grant proposals. Accordingly, more journal editors are starting to make sure that author guidelines include a section on code and data availability [16, 17], and reviewers are now considering reproducibility in their decision processes [10]. Moreover, concepts and tools to package code, the computing environment, data, and the text of a research workflow (a so-called "Research Compendium" [18]) are becoming more advanced and applied. This form of publishing research allows reviewers to verify the reported results and readers to reuse the materials [19].

Nevertheless, neither the cultural and systematic developments [20] for ORR nor the existence of technologies for packaging research reproducibly can alone solve the plethora of reproducibility issues. Authors often do not know how to fulfill the requirements of funding bodies and journals, such as the Transparency and Openness Promotion (TOP) guidelines [21], or they do not have the programming skills. It is important to consider that the range of researchers' programming expertise varies from trained research software engineers to self-taught beginners. For these reasons, more and more applications have been created to support the publication of executable computational research for transparent and reproducible research. This paper aims at reviewing these applications in order to help researchers find the application that best suits their individual needs.

Methods

Study design

In this review study, we surveyed and compared 11 applications that assist authors in publishing reproducible research. The goal of the review was to obtain an overview of the benefits and limitations of these applications considering the challenges outlined in the previous section. We contrasted the solutions to create a set of criteria that addresses the needs of the stakeholders involved in the scholarly publication process, i.e. publishers, editors, authors, readers/reviewers, and librarians [22].

Sample

We identified the applications during a literature search as well as through discussions at conferences[1] and workshops.[2] We included an application in our analysis if it (i) was actively maintained at the time the data for this paper was collected (5th–13th Dec 2019[3]), (ii) supports publishing executable code and data that can be inspected and reused, and (iii) is explicitly connected to the publication process. Hence, we did not consider technologies (e.g., containerization) that alone cannot support the publication process of code because further infrastructure is needed, systems that only provide access to materials (e.g., Zenodo), or workflow systems (e.g., Taverna [23]). Based on the sample criteria, we selected the following eleven applications for the review, presented in alphabetical order (see Table 1).

Variables

In a next step, we reviewed literature to identify a set of comparison criteria (highlighted in bold in the following) relevant for the stakeholders mentioned above. According to Hrynaszkiewicz [17], publishers refrain from hosting data, raising the question of whether the applications allow (1) "free self-hosting" by the publishers. Since self-hosting might require changes to the software, we also checked whether the applications are released under an (2) "open license". Next, a proxy for assessing the stage and the reliability of an application is to check whether it is already (3) "in use". To provide an initial estimate of the application's longevity, we looked up whether the applications are (4) "grant-based", since such funds are usually temporary. Also, because using literate programming tools is a frequently mentioned recommendation for creating executable documents [35], we checked whether the applications support (5) "R Markdown" and (6) "Jupyter Notebooks". However, since researchers might have individual requirements [22], we also

[1]One conference we attended: EGU General Assembly 2019 (last access of this and the following URLs: 22nd May 2020); it also had a session on open science.

[2]Workshop: eLife innovation sprint 2019, which brought together people interested in open science.

[3]A reviewer directed us to the application Authorea, which we missed in our first analysis, and the lack of pricing information. We thus collected data to address these aspects on 22nd May 2020

Table 1 Overview of applications we included in the analysis

Application	Description
Authorea	In Authorea, authors can create executable papers collaboratively. They can attach code and data to figures to make them reproducible. Authors can also directly submit to a journal and, at the same time, publish a preprint.
Binder	Binder creates a containerized executable environment based on a repository (e.g., on GitHub/Lab, Zenodo) including a Jupyter Notebook [24]. Readers can launch the analysis and inspect the workflow in a browser.
Code Ocean	Code Ocean creates "capsules" containing code, data, and the computational environment. While reading, users can execute and inspect the analysis in a separate window below the article or on Code Ocean's website [25].
eLife Reproducible Document Stack (RDS)	RDS originates from the life sciences. Authors can publish executable documents based on Stencila (https://stenci.la/), an open-source editor for articles. The executable document, which contains the whole narrative and executable code snippets, is not only a supplement but the actual scientific article.
Galaxy	Galaxy [26] provides features tailored to use cases in the life sciences. It is a web app for developing comput. Analyses without programming expertise. Scientists can upload and analyze data using Jupyter Notebooks [27].
Gigantum	Gigantum packages code, data, the computational environment, and the work history into a Git repository. Gigantum is composed of a client app for creating as well as executing analyses locally and a cloud-based infrastructure for sharing computations and collaborating with peers.
Manuscripts	Manuscripts is an online tool for writing executable documents collaboratively based on the concept of literate programming, but featuring a "What you see is what you get" user interface. The runtime environment of the author is, however, not considered.
o2r	o2r [22] originates from the geosciences and addresses publishers who want to extend their infrastructure via a reproducibility service during the process of paper submission [28]. Authors can create interactive figures, allowing readers to change model parameters using a slider [29].
REANA	REANA [4, 30] originates from particle physics and provides a specification for capturing data, code, and the comput. Environment. Based on this structure and manually created configuration files, REANA provides command line interface (CLI) commands to run large analyses on a remote REANA cloud.
ReproZip	ReproZip [31, 32] provides a set of CLI commands for encapsulating data, code, and the computational environment. Users can execute the resulting bundle on a server provided by ReproZip [33] or locally on different systems.
Whole Tale	With Whole Tale [34], authors can create so-called "Tales" that combine narrative, data, code, and the computational environment. Readers can inspect the materials and execute the analysis in the original environment.

investigated whether the applications are (7) "extensible", meaning, for example, whether users can add a new submission format. A further relevant piece of information for authors is whether they need to (8) "upload" their materials to the application and whether (9) "copyright" is addressed explicitly in the documentation. Copyright is a main concern of authors in the context of ORR [36] and needs to be considered when it comes to reusing research materials [37]. We also checked whether (10) "sensitive data" can be shared. Based on the benefits of ORR summarized by Konkol et al. [36], we checked whether the applications provide tools to enable the (11) "discovery" of articles, (12) "inspection" of the materials, (13) "execution" of the analyses (one-click reproducible), (14) "manipulation" of parameter values, (15) "substitution" of datasets, and (16) "download" of materials. Finally, papers describing open science guidelines [21] or assessing reproducibility of published papers [3] often refer to the importance of making materials permanently available, for example, for future use and education [38]. Hence, we investigated whether it is possible to (17) "modify or delete materials after publication" and whether these materials can be (18) "shared via a DOI" or (19) "shared via a URL".

Data collection

Based on the comparison criteria, two authors collected information iteratively by investigating the project websites, applications, GitHub/Lab repositories, scientific articles, and blog posts. In the first iteration, one author of this paper gathered information on the applications one by one and took screenshots. In the second iteration, another author independently checked the information collected in the first iteration. Conflicts concerning the data were resolved through discussion amongst all authors. In order to give the scientific community, particularly the developers of the considered applications, the opportunity to comment on the analysis, we published a preprint [39] of the paper three months before submission. All collected data is available in the supplement (see Availability of data and materials). Thus, it is possible to continue the work in this paper as a "rolling review". Since some sources (e.g., documentation) were not scientific articles, we also attached links to and screenshots of the original information.

Results

Table 2 summarizes aspects relevant for publishers, editors, authors, readers, and librarians.

Table 2 Overview of which application supports the corresponding criteria. (N/D = no data)

	Authorea	Binder	Code Ocean	eLife RDS	Galaxy	Gigantum	Manuscripts	o2r	REANA	Repro Zip	Whole Tale
Free self-hosting	–	+	–	+*	+	–	+	+	+*	+	+
Open license	–	+	–	+	+	+/–	+	+	+	+	+
In use	in use [40]	in use [2]	in use [41]	in use [42]	in use [43]	–	–	–	in use [44]	in use [31]	–
Grant-based	–	+	–	+	+	–	N/D	+	+	+	+
R Markdown	–	+	+	+	–	+	–	+	–	–	+
Jupyter Notebooks	+	+	+	+	+	+	–	–	+	+	+
Extensible	–	+	+	+	+	–	–	–	+	+	+
Upload	+	+	+	–	+	–	+	+	–	–	+
Copyright	+	N/D	+	N/D	+	+	N/D	+	N/D	N/D	+
Sensitive data	–	–	–	–	–	–	–	–	–	–	–
Discovery	+	–	+	+	+	–	–	+	–	–	+
Inspection	+	+	+	+	+	+	+	+	–	–	+
Execution	+	+	+	+	+	+	+	+	+	+	+
Manipulation	+	+	+	+	+	+	+	+	+	+	+
Substitution	–	–	–	–	–	–	–	+	–	+	–
Download	+	+	+	+	+	+	+	+	–	+	+
Modify/Delete after publishing	–	+	–	–	+	+	+	–	+	+	–
Shared via DOI	+	–	+	+	–	–	–	–	–	–	+
Shared via URL	+	+	+	+	+	+	+	+	–	+	–

From the eleven applications, eight allow self-hosting for free. eLife RDS and REANA (in Table 2 marked by *) require deployment, since no free online version exists. Eight applications are released under an open license, and Gigantum has only published the client tool under an open license. The open source applications that have an online version running can be used by researchers for free. The three commercial providers, namely Authorea, Code Ocean, and Gigantum, provide the service in exchange for payment but they also offer free subscriptions with limited features and resources (i.e. storage and computation time).

In total, seven applications are already in use, as shown by the example papers with reproducible workflows. Seven applications currently receive funding from public or private organizations.[4]

Ten applications support literate programming, e.g., R Markdown or Jupyter Notebooks; the Manuscripts application supports Markdown but also code execution via embedded Jupyter Notebooks. Seven applications are extensible and can be configured to support further programming languages. Except for Code Ocean, which also supports MATLAB and Stata, all applications only support non-proprietary programming languages.

Seven applications require authors to create their projects online, whereas eLife's RDS (based on Stencila), REANA, and ReproZip allow local usage. Researchers can also work locally with Gigantum, but they then need to synchronize with the online service to access all features.

Regarding copyright, we could not find explicit information on assigning copyright for research materials in five applications. Whole Tale and Gigantum only allow open licenses, whereas Code Ocean, Galaxy, and o2r encourage them. We could not find information on sensitive data in any of the applications.

From the eleven applications, six provide a keyword-based search for papers, of which o2r provides a spatio-temporal search combined with thematic properties, such as libraries used in the code. Five applications embed a programming environment (e.g., JupyterLab, RStudio) for inspecting code and data, whereas four provide their own user interface (UI).

All applications provide support for executing the analysis. REANA projects are executed via the CLI in a remote REANA cloud, and this also applies to ReproZip, which in addition to a remote cloud also provides a

[4]Further details on funding are available in the supplement.

ReproServer for executing code online. Gigantum's local client allows users to run code in the browser. The remaining applications allow users to execute the analysis in a browser on a remote server.

Each application allows users to manipulate the code and rerun it based on a new parameter. Most commonly, users can directly manipulate the code in the browser (8 applications provide this option). In REANA and ReproZip, users can pass new parameter values via the CLI, whereas the o2r platform enables authors to configure UI widgets that allow reviewers/readers to manipulate parameter values interactively, e.g., by using a slider to change a model parameter within a defined range. Features for substituting the input datasets used in an analysis are provided by o2r and ReproZip.

While ten applications provide a feature for downloading materials, REANA projects need to be stored on a third-party service to be downloadable.

Overall, six applications allow users to modify/delete materials after publication. In Binder, REANA, and ReproZip, modifying/deleting content is possible if the research materials are stored on GitHub/Lab, but not when they are stored on Zenodo. Authorea, Code Ocean, eLife RDS, and Whole Tale assign DOIs to published content, ensuring long-term availability and making it impossible to edit these after publication. In eLife's RDS, the article is composed of text and code; thus, deleting it is equivalent to withdrawing a paper.

Discussion

Several developers have created applications for publishing computational research. One might think the applications, since they all strive for the same overall goal, resemble each other. However, we showed in this paper that the applications address different issues and needs, which increases the chances that stakeholders will find an application that best suits their individual requirements. The review can be used by the various stakeholders in different ways: Publishers who want to comply with reproducibility principles may use it to decide for a certain application, editors and program committees may use it when planning to include code review in their review process [45], applicants designing data and software management plans may use it when writing their funding proposals, and authors who are searching for tools to disseminate their work in a convincing, sustainable, and transparent manner may also find it valuable. In addition to these stakeholders, we also considered librarians, who are tasked with aspects related to the preservation and long-term accessibility of research materials. Given the variety of the stakeholders and their considerations, it is difficult to determine the best application or objectively provide a ranking. Identifying the ideal application strongly depends on the needs and goals of the stakeholders.

Hosting of the applications

Publishers need to decide whether they want to host an infrastructure by themselves or engage a provider. Applications exist for both approaches, though the majority of them can be self-hosted. Some of the self-hosting solutions are also available as online versions, but it should be considered that these have limited resources regarding storage and processing power. Moreover, it is difficult to estimate when the projects will expire. If the applications are grant-based, they might receive follow-up funding, but this depends on the projects' success and whether they aim at carrying out research (which might come with many changes to the software) or developing a scalable and sustainable platform. As public information on all funding levels and grant durations are too uncertain and incomplete to be included in the analysis, we refrained from drawing concrete conclusions in terms of longevity and how likely they will exist in the next years.

All self-hosting solutions have an open license, allowing operators to host their own service as well as modify the software according to their individual needs and styles. A further advantage of self-hosting is the mitigation of risks regarding vendor lock-in. However, hosting one's own service means that publishers also have to provide the required technological resources and personnel. It remains unclear what kinds of costs publishers will have to expect when hosting a platform and incorporating it into their publishing infrastructure. The final costs strongly depend on the number of views, execution attempts, workflow sizes, and ease of integration into technical systems. These parameters differ between use cases and could be used as measures for future research, e.g., on stress tests. Therefore, the metrics of existing publications might provide the first ways to calculate the required resources. While the Binder instance MyBinder.org published an initial estimate regarding costs,[5] further data from other services would help to calculate costs more specifically. Moreover, it would be interesting to see usage statistics showing how often the services are used, for example, by authors, readers, and reviewers, albeit this transparency is only realistic for non-profit projects. Nevertheless, since reproducibility studies are rarely successful [5], using these services seems to be uncommon.

A further criterion we investigated was whether the applications are in use. While applications in use offer initial evidence that they work, it might take more effort

[5]MyBinder costs: https://mybinder.org/v2/gh/jupyterhub/binder-billing/master?urlpath=lab/tree/analyze_data.ipynb

to adjust them to fit a publisher's infrastructure. In contrast, beta applications can adjust their features without worrying about running instances and backwards compatibility, but the deployment of such applications might reveal new issues.

Creating executable analyses

Regarding submission formats, there is a trend toward literate programming approaches. Most applications either support Jupyter Notebooks or R Markdown, which both have proven to support reproducibility [46]. However, some journals and publishers rely on different formats, e.g., LaTeX. Transformations to other document types are often cumbersome and adapting author requirements can be a lengthy process. Hence, it might be easier to have reproducible documents as a supplement, potentially for a transition period, until researchers have adjusted their workflows and executable documents are widely accepted. Nevertheless, eLife's RDS has already shown that combining executable code with narrative in a scientific article is possible today and comes with advantages related to communicating scientific results. For example, readers can, while studying the text, also manipulate the analysis. A limitation in this context is related to the peer-review process. All applications require an account for creating reproducible results, and since the name of the creator is usually visible, a double-blind review is not guaranteed. However, access to code and data is particularly important for reviewers who need to recommend acceptance or rejection of a submission. One solution might be to create anonymous versions of the materials, as is possible with Open Science Framework,[6] or to adopt an open peer-review process.

A further critical issue is that not all applications address copyright in their documentation. Those that do either require or encourage open licenses, which is a recommendation mentioned frequently in papers discussing reproducibility guidelines [21, 37]. Hence, the platforms should inform users about licenses, e.g., by referring to existing advising resources (e.g., https://choosealicense.com/). Licensing is important to enable reusability and, thus, is ideally assigned to code, data, and text separately, as is done, for example, by Authorea. Computational reproducibility is challenging also because of sensitive data. None of the applications address this issue, but platforms allowing self-hosting can be combined with existing solutions, such as involving trustworthy authorities [47] and cloud-based data enclaves [48].

[6]Anonymized links: https://help.osf.io/hc/en-us/articles/360019930333-Create-a-View-only-Link-for-a-Project

Studying reproducible research

Being able to reproduce the computational results in a paper is a clear benefit, but open reproducible research comes with a number of further incentives [20]. Concerning the discovery of papers, most search tools provided by the applications do not take full advantage of the information contained in code and data files, e.g., spatiotemporal properties. Instead, they either only provide a keyword-based search or no search at all. For inspecting materials, most solutions either provide their own UI or integrate a development environment, e.g., JupyterLab. In both cases, users can directly access, manipulate, and reuse the code. However, readers (including experienced programmers) might still find it challenging to understand complex code scripts. Moreover, identifying specific parameters buried in the code and finding out how to change these can be a daunting task. The concept of nano-publications [49] or bindings [29] might help to solve these issues. A further need in this context is a UI for comparing original and manipulated figures, since differences in the figure after changing parameters might be difficult to spot. Most applications do not provide any support for substituting research components, e.g., by other input datasets, which might be due to the plethora of complex interoperability issues with respect to data formats or column names in tabular data. Only ReproZip [32] and o2r [36] provide basic means to substitute input datasets, yet they require users to ensure compatibility.

Researchers who are writing or studying computational research articles might be programming beginners or experts. Hence, while the learning curve may be either shallow or steep, it is present in any case. Although the applications are well documented, programming novices in particular might need to invest effort at the beginning of use. For example, they would need to learn how to write R Markdown documents and create configuration files manually. Some of the creation steps might be automated, but this usually comes at the cost of flexibility. The learning curve not only exists for authors but also for consumers, particularly reviewers who need to verify the results and those who want to build upon the materials. Nevertheless, such an effort only needs to be invested once and will eventually result in more convincing and transparent research.

Sharing computational research

The state of the research materials is an issue when it comes to publication. While some applications fix the state of the research materials by assigning a DOI and archiving a snapshot, others allow changing and deleting them. This is a disadvantage with respect to reproducibility since verifiability and accessibility are lost. In addition, if self-hosting is not possible, the computational analysis of an article will be executable only as long as the project and its infrastructure exist; this dependence is a crucial aspect with respect to archiving. However, this issue can be mitigated if researchers "go the extra mile" and also publish their materials in long-term repositories in addition to an executable version using one of the applications.

A further dependence is the technology underlying the infrastructure. For example, without the Docker container runtime, the captured computing environment might not work even though it remains human readable [50]. This is also true for source code scripts, which are plain text files and, thus, can be opened using any editor, even if they cannot be compiled and executed. These examples demonstrate the importance of using open and text-based file formats instead of proprietary and binary file formats in science.

Limitations

This work is subject to a number of limitations. The scope of this review is narrow and does not cover all applications that are connected with computational research (e.g., workflow systems, such as Taverna [23]). Also, we have no access to publishers' actual systems, preventing us from being able to evaluate the usability of APIs and documentation and how easy they can be incorporated into existing infrastructures. In addition, this review is a snapshot of the highly dynamic area of publishing infrastructures. Hence, the information might become outdated quickly, e.g., an application might extend the set of functionalities or be discontinued. Still, reviewing the current state of the landscape to reflect on available options might be helpful for researchers. Furthermore, the properties we investigated in this survey do not cover all possible aspects and discipline-specific needs, but, nevertheless, stakeholders requiring more information can use the overview as a starting point for further research. Also, we collected and interpreted the data ourselves and did not contact the application developers, which might have increased the accuracy of the data. Finally, our evaluation only considered documented features. However, programmers with sufficient expertise can build upon the open source applications and implement missing features.

Conclusions

In this review, we compared eleven applications in order to identify their benefits and limitations for assisting researchers to publish and study open reproducible research. Our findings show that publishers have the choice between using provided services or self-hosting solutions, but more data is needed to estimate the costs for publishers to maintain their own infrastructure. The review revealed a trend towards literate programming approaches as well as tools for reviewers and readers, e.g., for inspecting an analysis or manipulating the assumptions underlying the analysis. We found that being able to change the materials after publication might result in conflicts between the version referred to in an article and the available version, which might have been changed since the article was first published. In addition to investigating these issues, the next step is to examine how using an application affects a reviewer's decision and how much additional effort is needed to study the materials.

Abbreviations

CLI : Command line interface; DOI : Digital Object Identifier; o2r : Opening reproducible research (project); ORR : Open reproducible researchers; REANA : Reproducible Analysis (project); RDS : Reproducibility Document Stack; UI : User Interface

Acknowledgements

We thank Timothy Errington and Mario Malicki for their reviews, Vicky Steeves for a helpful discussion on the article preprint [39], and Celeste Brennecka for proofreading.

Authors' contributions

Markus Konkol wrote the paper, collected the data, and conceptualized the analysis. Daniel Nüst wrote the paper. Laura Goullier collected data and wrote the paper. All authors discussed the results and approved the final manuscript.

Funding

This work is supported by the project Opening Reproducible Research 2 (https://www.uni-muenster.de/forschungaz/project/12343) funded by the German Research Foundation (DFG) under project numbers KR 3930/8–1; TR 864/12–1; PE 1632/17–1. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Availability of data and materials

The data is openly available on Zenodo: https://doi.org/10.5281/zenodo.3562269. The repository includes a list of all applications we looked at and, for excluded applications, the reasons for exclusion.

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Competing interests

The authors of this paper are members of the o2r project that was also discussed in this paper (http://o2r.info/).

Received: 5 March 2020 Accepted: 24 June 2020
Published online: 14 July 2020

References

1. Stodden V, McNutt M, Bailey DH, Deelman E, Gil Y, Hanson B, et al. Enhancing reproducibility for computational methods. Science. 2016; 354(6317):1240–1. https://doi.org/10.1126/science.aah6168.
2. Stagge JH, Rosenberg DE, Abdallah AM, Akbar H, Attallah NA, James R. Assessing data availability and research reproducibility in hydrology and water resources. Sci Data. 2019;6(1). https://doi.org/10.1038/sdata.2019.30.
3. Nüst D, Granell C, Hofer B, Konkol M, Ostermann FO, Sileryte R, Cerutti V. Reproducible research and GIScience: an evaluation using AGILE conference papers. PeerJ. 2018;6:e5072. https://doi.org/10.7287/peerj.preprints.26561.
4. Chen X, Dallmeier-Tiessen S, Dasler R, Feger S, Fokianos P, Benito Gonzalez J, Hirvonsalo H, et al. Open is not enough. Nat Phys. 2018;15(2):113–9. https://doi.org/10.1038/s41567-018-0342-2.
5. Konkol M, Kray C, Pfeiffer M. Computational reproducibility in geoscientific papers: insights from a series of studies with geoscientists and a

reproduction study. Int J Geogr Inf Sci. 2018;33(2):408–29. https://doi.org/10.1080/13658816.2018.1508687.
6. Herndon T, Ash M, Pollin R. Does high public debt consistently stifle economic growth? A critique of Reinhart and Rogoff. Camb J Econ. 2013; 38(2):257–79. https://doi.org/10.1093/cje/bet075.
7. National Academies of Sciences, Engineering, Medicine & others. Reproducibility and Replicability in science. Washington, DC.: National Academies Press; 2019. https://doi.org/10.17226/25303.
8. Markowetz F. Five selfish reasons to work reproducibly. Genome Biol. 2015; 16(1). https://doi.org/10.1186/s13059-015-0850-7.
9. McKiernan EC, Bourne PE, Brown CT, Buck S, Kenall A, Lin J, Yarkoni T. Author response: how open science helps researchers succeed. 2016 https://doi.org/10.7554/elife.16800.008.
10. Stark PB. Before reproducibility must come preproducibility. Nature. 2018; 557(7707):613. https://doi.org/10.1038/d41586-018-05256-0.
11. Vazire S. A toast to the error detectors. Nature. 2020.
12. Bailey DH, Borwein JM, Stodden V. Facilitating reproducibility in scientific computing: principles and practice. Reproducibility. 2016:205–31. https://doi.org/10.1002/9781118865064.ch9.
13. Donoho DL. An invitation to reproducible computational research. Biostatistics. 2010;11(3):385–8. https://doi.org/10.1093/biostatistics/kxq028.
14. Powers SM, Hampton SE. Open science, reproducibility, and transparency in ecology. Ecol Appl. 2018;29(1). https://doi.org/10.1002/eap.1822.
15. Piwowar H. Sharing detailed research data is associated with increased citation rate. Nat Preced. 2007. https://doi.org/10.1038/npre.2007.361.1.
16. Nüst D, Ostermann FO. Sileryte R, Hofer B, Granell C, Teperek M, Graser A, Broman KW, Hettne KM. (2019). AGILE reproducible paper guidelines. https://doi.org/10.17605/OSF.IO/CB7Z8.
17. Hrynaszkiewicz I. Publishers' responsibilities in promoting data quality and reproducibility. Handb Exp Pharmacol. 2019. https://doi.org/10.1007/164_2019_290.
18. Gentleman R, Temple Lang D. Statistical analyses and reproducible research. J Comput Graph Stat. 2007;16(1):1–23. https://doi.org/10.1198/106186007x178663.
19. Barba LA. Terminologies for reproducible research. arXiv preprint arXiv:1802.03311; 2018.
20. Munafò MR, Nosek BA, Bishop D, Button KS, Chambers CD, Sert NP, Simonsohn U, Wagenmakers E-J, Ware JJ, Ioannidis JPA. A manifesto for reproducible science. Nat Hum Behav. 2017;1(1). https://doi.org/10.1038/s41562-016-0021.
21. Nosek BA, Alter G, Banks GC, Borsboom D, Bowman SD, Breckler SJ, et al. Promoting an open research culture. Science. 2015;348(6242): 1422–5.
22. Nüst D, Konkol M, Pebesma E, Kray C, Schutzeichel M, Przibytzin H, Lorenz J. Opening the publication process with executable research compendia. D-Lib Magazine. 2017;23(1/2). https://doi.org/10.1045/january2017-nuest.
23. Wolstencroft K, Haines R, Fellows D, Williams A, Withers D, Owen S, et al. The Taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. Nucleic Acids Res. 2013; 41(W1):W557–61. https://doi.org/10.1093/nar/gkt328.
24. Jupyter P, Bussonnier M, Forde J, Freeman J, Granger B, Head T, Willing C. Binder 2.0 - reproducible, interactive, sharable environments for science at scale. Proceedings of the 17th Python in Science Conference. 2018. https://doi.org/10.25080/majora-4af1f417-011.
25. Clyburne-Sherin A, Fei X, Green SA. Computational reproducibility via Containers in Social Psychology. Meta-Psychology 3. 2019. https://doi.org/10.15626/MP.2018.892.
26. Goecks J, Nekrutenko A, Taylor J, Galaxy Team T. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol. 2010;11(8):R86. https://doi.org/10.1186/gb-2010-11-8-r86.
27. Grüning BA, Rasche E, Rebolledo-Jaramillo B, Eberhard C, Houwaart T, Chilton J, et al. Jupyter and Galaxy: easing entry barriers into complex data analyses for biomedical researchers. PLoS Comput Biol. 2017;13(5):e1005425. https://doi.org/10.1371/journal.pcbi.1005425.
28. Nüst D. Reproducibility Service for Executable Research Compendia: technical specifications and reference implementation (version 1.0.0). *Zenodo*.2018. https://doi.org/10.5281/zenodo.2203844.
29. Konkol M, Kray C, Suleiman J. Creating interactive scientific publications using bindings. *Proceedings of the ACM on Human-Computer Interaction*,2019:1–18. https://doi.org/10.1145/3331158.
30. Šimko T, Heinrich L, Hirvonsalo H, Kousidis D, Rodríguez D. REANA: a system for reusable research data analyses. EPJ Web Conf. 2019;214:06034. https://doi.org/10.1051/epjconf/201921406034.
31. Steeves V, Rampin R, Chirigati F. Using ReproZip for reproducibility and library services. *IASSIST Quarterly*. 2017;42(1):14. https://doi.org/10.29173/iq18.
32. Chirigati F, Doraiswamy H, Damoulas T, Freire J. Data polygamy. Proceedings of the 2016 International Conference on Management of Data - SIGMOD *'16*. 2016. https://doi.org/10.1145/2882903.2915245.
33. Rampin R, Chirigati F, Steeves V, Freire J. ReproServer: making reproducibility easier and less intensive. arXiv Preprint arXiv:1808.01406; 2018.
34. Brinckman A, Chard K, Gaffney N, Hategan M, Jones MB, Kowalik K, Stodden V, Turner K, et al. Computing environments for reproducibility: Capturing the "Whole Tale". Futur Gener Comput Syst. 2019;94:854–67. https://doi.org/10.1016/j.future.2017.12.029.
35. Peng RD, Dominici F, Zeger SL. Reproducible epidemiologic research. Am J Epidemiol. 2006;163(9):783–9.
36. Konkol M, Kray C. In-depth examination of spatiotemporal figures in open reproducible research. Cartogr Geogr Inf Sci. 2018;46(5):412–27. https://doi.org/10.1080/15230406.2018.1512421.
37. Stodden V. The legal framework for reproducible scientific research: licensing and copyright. Comput Sci Eng. 2009;11(1):35–40. https://doi.org/10.1109/mcse.2009.19.
38. Sayre F, Riegelman A. Replicable Services for Reproducible Research: a model for academic libraries. Coll Res Libraries. 2019;80(2):260. https://doi.org/10.5860/crl.80.2.260.
39. Konkol M, Nüst D, Goulier L. Publishing computational research - a review of infrastructures for reproducible and transparent scholarly communication. arXiv preprint arXivarXiv:2001.00484; 2020.
40. Hanwell MD, Harris C, Genova A, et al. Open chemistry, JupyterLab, REST, and quantum chemistry. *Authorea*. 2020. https://doi.org/10.22541/au.158687268.81852407.
41. Chitre M. Editorial on writing reproducible and interactive papers. IEEE J Ocean Eng. 2018;43(3):560–2. https://doi.org/10.1109/joe.2018.2848058.
42. Lewis LM, Edwards MC, Meyers ZR, Talbot Jr, CC, Hao H, Blum D. Replication Study: Transcriptional Amplification in Tumor Cells with Elevated c-Myc Cancer Biol 7. 2018. https://doi.org/10.7554/eLife.30274.
43. Ide N, Suderman K, Verhagen M, Pustejovsky J. The language application grid web service exchange vocabulary. Lect Notes Comput Sci. 2016:18–32. https://doi.org/10.1007/978-3-319-31468-6_2.
44. Prelipcean D. Physics examples for reproducible analysis. *CERN*. 2019. https://cds.cern.ch/record/2690231.
45. Eglen S, Nüst D. CODECHECK: an open-science initiative to facilitate sharing of computer programs and results presented in scientific publications. Septentrio Conf Series. 2019;1. https://doi.org/10.7557/5.4910.
46. Grüning B, Chilton J, Köster J, Dale R, Soranzo N, van den Beek M, et al. Practical computational reproducibility in the life sciences. Cell Syst. 2018; 6(6):631–5. https://doi.org/10.1016/j.cels.2018.03.014.
47. Pérignon C, Gadouche K, Hurlin C, Silberman R, Debonnel E. Certify reproducibility with confidential data. Science. 2019;365(6449). https://doi.org/10.1126/science.aaw2825.
48. Foster I. Research infrastructure for the safe analysis of sensitive data. Ann Am Acad Political Soc Sci. 2017;675(1):102–20. https://doi.org/10.1177/0002716217742610.
49. Kuhn T, Chichester C, Krauthammer M, Queralt-Rosinach N, Verborgh R, Giannakopoulos G, et al. Decentralized provenance-aware publishing with nanopublications. PeerJ Comp Sci. 2016;2:e78. https://doi.org/10.7717/peerj-cs.78.
50. Boettiger C. An introduction to Docker for reproducible research. ACM SIGOPS Operating Syst Rev. 2015;49(1):71–9. https://doi.org/10.1145/2723872.2723882.

Publisher's Note

7 containerit: generating Dockerfiles for reproducible research with R

Authors & contribution Daniel Nüst (75%), Matthias Hinz, Edzer Pebesma

Venue *JOSS* 10.21105/joss.01603

Date 08/2019

Licence Creative Commons Attribution 4.0 International (CC BY 4.0)

Repository https://github.com/o2r-project/containerit/

containerit: Generating Dockerfiles for reproducible research with R

Daniel Nüst[1] **and Matthias Hinz**[2]

1 Institute for Geoinformatics, University of Münster, Germany **2** Professorship for Geoinformatics and Geodesy, Faculty of Agricultural and Environmental Sciences, University of Rostock, Germany

DOI: 10.21105/joss.01603

Software

- Review
- Repository
- Archive

Submitted: 25 July 2019
Published: 21 August 2019

Statement of Need

Linux containers have become a promising tool to increase transparency, portability, and reproducibility of research in several domains and use cases: data science (Boettiger, 2015), software engineering research (Cito & Gall, 2016), multi-step bioinformatics pipelines (Kim, Ali, Lijeron, Afgan, & Krampis, 2017), standardised environments for exchangeable software (Belmann et al., 2015), computational archaeology (Marwick, 2017), packaging algorithms (Hosny, Vera-Licona, Laubenbacher, & Favre, 2016), or geographic object-based image analysis (Knoth & Nüst, 2017). Running an analysis in a container increases reliability of a workflow, as it can execute packaged code independently of the author's computer and its available configurations and dependencies. However, capturing a computational environment in containers can be complex, making container use difficult for domain scientists with limited programming experience. `containerit` opens up the advantages of containerisation to a much larger user base by assisting researchers, who are unfamiliar with Linux, command lines or containerisation, in packaging workflows based on R (R Core Team, 2018) in container images by using only user-friendly R commands.

Recently containerisation took off as a technology for packaging applications and their dependencies for fast, scalable, and secure sandboxed deployments in cloud-based infrastructures (cf. Osnat, 2018). The most widely used containerisation software is Docker with the following core building blocks (cf. Docker: Get Started): The *image* is built from the instructions in a recipe called `Dockerfile`. The image is executed as a *container* using a *container runtime*. An image can be moved between systems as a file (image tarball) or based on an *image registry*. A `Dockerfile` may use the image created by another `Dockerfile` as the starting point, a so-called *base image*. While containers can be manually altered, the common practice is to conduct all configurations with the scripts and instructions originating in the `Dockerfile`.

An important advantage of containers over virtual machines is that their duality between recipe and image provides and additional layer of transparency and safeguarding. The `Dockerfile` and image can be published alongside a scientific paper to support peer review and, to some extent, preserve the original results (Nüst et al., 2017). Even if an image cannot be executed or a `Dockerfile` can no longer be built, the instructions in the `Dockerfile` are human-readable, and files in the image can be extracted to recreate an environment that closely resembles the original. Further useful features are (a) portability, thanks to a single runtime dependency, which allows readers to explore an author's virtual laboratory, including complex dependencies or custom-made code, either on their machines or in cloud-based infrastructures (e.g., by using Binder, see Project Jupyter et al., 2018), and (b) transparency, because an image's filesystem can be easily inspected. This way, containers can enable verification of reproducibility and auditing without requiring reviewers to manually download, install, and re-run analyses (Beaulieu-Jones & Greene, 2017).

Container preservation is an active field of research (Emsley & De Roure, 2018; Rechert et al., 2017). It is reasonable to assume that key stakeholders interested in workflow preservation, such as universities or scientific publishers, should be able to operate container runtimes on a time scale comparable to data storage requirements by funding agencies, e.g., 10 years in case of the German DFG or British EPSRC. To enable and leverage the stakeholders' infrastructure, container creation must become easier and more widespread.

Summary

The package `containerit` automates the generation of `Dockerfiles` for workflows in R, based on images by the Rocker project (Boettiger & Eddelbuettel, 2017). The core feature of `containerit` is that it transforms the local session information into a set of instructions which can be serialised as a `Dockerfile`, as shown in the code snippet below:

```
> suppressPackageStartupMessages(library("containerit"))
> my_dockerfile <- containerit::dockerfile(from = utils::sessionInfo())
> print(my_dockerfile)
FROM rocker/r-ver:3.5.2
LABEL maintainer="daniel"
RUN export DEBIAN_FRONTEND=noninteractive; apt-get -y update \
  && apt-get install -y git-core \
    libcurl4-openssl-dev \
    libssl-dev \
    pandoc \
    pandoc-citeproc
RUN ["install2.r", "curl", "digest", "evaluate", "formatR", \
  "futile.logger", "futile.options", "htmltools", "jsonlite", \
  "knitr", "lambda.r", "magrittr", "Rcpp", "rjson", \
  "rmarkdown", "rsconnect", "semver", "stevedore", "stringi", \
  "stringr", "xfun", "yaml"]
WORKDIR /payload/
CMD ["R"]
```

The created `Dockerfile` has installation instructions for the loaded packages and their system dependencies. It uses the `r-ver` stack of Rocker images, matching the R version to the environment encountered locally by `containerit`. These images use MRAN snapshots to control installed R package versions in a reproducible way. The system dependencies required by these packages are identified using the `sysreqs` package (Csardi, 2019) and the corresponding database and API.

`dockerfile(..)` is the package's main user function and accepts session information objects, session information saved in a file, a set of R commands, an R script file, a `DESCRIPTION` file, or an R Markdown document (Allaire et al., 2018). Static program analysis using the package `automagic` (Brokamp, 2017) is used to increase the chances that the capturing environment has all required packages available, such as when creating Dockerfiles for R Markdown documents as a service (Nüst, 2018). To capture the workflow environment, `containerit` executes the whole workflow in a new R session using the package `callr` (Csárdi & Chang, 2018), because static program analysis can be broken by using helper functions, such as `xfun::pkg_attach()` (Xie, 2018), by unintended side effects, or by seemingly clever or user-friendly yet customised ways of loeading packages (cf. first lines in R script file `tgis_a_1579333_sm7524.r` in https://doi.org/10.6084/m9.figshare.7757069.v1). Further parameters for the function comprise, for example, image metadata, base image, versioned installations, and filtering of R packages already installed in the base image.

The package `containerit`'s main contribution is that it allows for automated capturing of runtime environments as `Dockerfiles` based on literate programming workflows (Gentleman & Lang, 2007) to support reproducible research. Together with `stevedore` (FitzJohn, 2019), `containerit` enables a completely R-based creation and manipulation of Docker containers. Using `containerit` only minimally affects researchers' workflows because it can be applied after completing a workflow, while at the same time the captured snapshots can enhance the scholarly publication process (in particular review, interaction, and preservation) and may form a basis for more reusable and transparent publications. In the future, `containerit` may support alternative container software such as Singularity (Kurtzer, Sochat, & Bauer, 2017), enable parametrisation of container executions and pipelines as demonstrated by Kliko (Molenaar, Makhathini, Girard, & Smirnov, 2018), or support proper accreditation of software (Jones et al., 2017; D. S. Katz & Chue Hong, 2018).

Related Work

`renv` is an R package for managing reproducible environments for R providing isolation, portability, and pinned versions of R packages, but it does not handle system dependencies. The Experiment Factory similarly focuses on ease of use for creating `Dockerfiles` for behavioural experiments, yet it uses a CLI-based interaction and generates extra shell scripts to be included in the images. ReproZip (Chirigati, Rampin, Shasha, & Freire, 2016) packages files identified by tracing in a self-contained bundle, which can be unpacked to a Docker container/`Dockerfile`. In the R domain, the package `dockerfiler` (Fay, 2018) provides an object-oriented API for manual Dockerfile creation, and `liftr` (Xiao, 2018) creates a `Dockerfile` based on fields added to the metadata header of an R Markdown document. `automagic` (Brokamp, 2017), Whales, `dockter`, and `repo2docker` use static program analysis to create environment descriptions from common project configuration files for multiple programming languages. Namely, `automagic` analyses R code and can store dependencies in a bespoke YAML format. Whales and `dockter` provide different formats, including `Dockerfile`. Finally, `repo2docker` primarily creates containers for interactive notebooks to run as a Binder (Project Jupyter et al., 2018) but does not actively expose a `Dockerfile`. None of them apply the strict code execution approach as `containerit` does.

Acknowledgements

This work is supported by the project Opening Reproducible Research (Offene Reproduzierbare Forschung) funded by the German Research Foundation (DFG) under project numbers `PE 1632/10-1` and `1632/17-1`.

References

Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., et al. (2018). *rmarkdown: Dynamic documents for R*. Retrieved from https://rmarkdown.rstudio.com

Beaulieu-Jones, B. K., & Greene, C. S. (2017). Reproducibility of computational workflows is automated using continuous analysis. *Nature Biotechnology*, *advance online publication*. doi:10.1038/nbt.3780

Belmann, P., Dröge, J., Bremges, A., McHardy, A. C., Sczyrba, A., & Barton, M. D. (2015). Bioboxes: Standardised containers for interchangeable bioinformatics software. *GigaScience*, *4*(1), 47. doi:10.1186/s13742-015-0087-0

Boettiger, C. (2015). An introduction to Docker for reproducible research, with examples from the R environment. *ACM SIGOPS Operating Systems Review*, *49*(1), 71–79. doi:10.1145/2723872.2723882

Boettiger, C., & Eddelbuettel, D. (2017). An Introduction to Rocker: Docker Containers for R. *The R Journal*, *9*(2), 527–536. doi:10.32614/RJ-2017-065

Brokamp, C. (2017). *Automagic: Automagically document and install packages necessary to run R code*. Retrieved from https://CRAN.R-project.org/package=automagic

Chirigati, F., Rampin, R., Shasha, D., & Freire, J. (2016). ReproZip: Computational reproducibility with ease. In *Proceedings of the 2016 international conference on management of data*, SIGMOD '16 (pp. 2085–2088). San Francisco, California, USA: ACM. doi:10.1145/2882903.2899401

Cito, J., & Gall, H. C. (2016). Using Docker Containers to Improve Reproducibility in Software Engineering Research. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16 (pp. 906–907). ACM. doi:10.1145/2889160.2891057

Csardi, G. (2019). *Sysreqs: Install systemrequirements of packages*. Retrieved from https://github.com/r-hub/sysreqs

Csárdi, G., & Chang, W. (2018). *Callr: Call R from R*. Retrieved from https://CRAN.R-project.org/package=callr

Emsley, I., & De Roure, D. (2018). A Framework for the Preservation of a Docker Container International Journal of Digital Curation. *International Journal of Digital Curation*, *12*(2). doi:10.2218/ijdc.v12i2.509

Fay, C. (2018). *Dockerfiler: Easy Dockerfile creation from R*. Retrieved from https://CRAN.R-project.org/package=dockerfiler

FitzJohn, R. (2019). *Stevedore: Docker client*. Retrieved from https://CRAN.R-project.org/package=stevedore

Gentleman, R., & Lang, D. T. (2007). Statistical Analyses and Reproducible Research. *Journal of Computational and Graphical Statistics*, *16*(1), 1–23. doi:10.1198/106186007X178663

Hosny, A., Vera-Licona, P., Laubenbacher, R., & Favre, T. (2016). AlgoRun: A Docker-based packaging system for platform-agnostic implemented algorithms. *Bioinformatics*, *32*(15), 2396–2398. doi:10.1093/bioinformatics/btw120

Jones, M. B., Boettiger, C., Mayes, A. C., Arfon Smith, Slaughter, P., Niemeyer, K., Gil, Y., et al. (2017). CodeMeta: An exchange schema for software metadata. KNB Data Repository. doi:10.5063/schema/codemeta-2.0

Katz, D. S., & Chue Hong, N. P. (2018). Software Citation in Theory and Practice. In J. H. Davenport, M. Kauers, G. Labahn, & J. Urban (Eds.), *Mathematical Software – ICMS 2018*, Lecture Notes in Computer Science (pp. 289–296). Springer International Publishing. doi:10.1007/978-3-319-96418-8_34

Kim, B., Ali, T. A., Lijeron, C., Afgan, E., & Krampis, K. (2017). Bio-Docklets: Virtualization Containers for Single-Step Execution of NGS Pipelines. *bioRxiv*, 116962. doi:10.1101/116962

Knoth, C., & Nüst, D. (2017). Reproducibility and Practical Adoption of GEOBIA with Open-Source Software in Docker Containers. *Remote Sensing*, *9*(3), 290. doi:10.3390/rs9030290

Kurtzer, G. M., Sochat, V., & Bauer, M. W. (2017). Singularity: Scientific containers for mobility of compute. *PLOS ONE*, *12*(5), e0177459. doi:10.1371/journal.pone.0177459

Marwick, B. (2017). Computational Reproducibility in Archaeological Research: Basic Principles and a Case Study of Their Implementation. *Journal of Archaeological Method and Theory*, *24*(2), 424–450. doi:10.1007/s10816-015-9272-9

Molenaar, G., Makhathini, S., Girard, J. N., & Smirnov, O. (2018). Kliko—The scientific compute container format. *Astronomy and Computing*, *25*, 1–9. doi:10.1016/j.ascom.2018.08.003

Nüst, D. (2018, December). Reproducibility Service for Executable Research Compendia: Technical Specifications and Reference Implementation. Zenodo. doi:10.5281/zenodo.2203844

Nüst, D., Konkol, M., Pebesma, E., Kray, C., Schutzeichel, M., Przibytzin, H., & Lorenz, J. (2017). Opening the Publication Process with Executable Research Compendia. *D-Lib Magazine*, *23*(1/2). doi:10.1045/january2017-nuest

Osnat, R. (2018, March). A Brief History of Containers: From the 1970s to 2017. Retrieved from https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016

Project Jupyter et al. (2018). Binder 2.0 - reproducible, interactive, sharable environments for science at scale. In *Proceedings of the 17th python in science conference*. doi:10.25080/Majora-4af1f417-011

R Core Team. (2018). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://www.R-project.org/

Rechert, K., Liebetraut, T., Kombrink, S., Wehrle, D., Mocken, S., & Rohland, M. (2017). Preserving Containers. In J. Kratzke & V. Heuveline (Eds.), *Forschungsdaten managen* (pp. 143–151). doi:10.11588/heibooks.285.377

Xiao, N. (2018). *Liftr: Containerize R Markdown documents for continuous reproducibility*. Retrieved from https://CRAN.R-project.org/package=liftr

Xie, Y. (2018). *Xfun: Miscellaneous functions by 'Yihui Xie'*. Retrieved from https://CRAN.R-project.org/package=xfun

8 Ten simple rules for writing Dockerfiles for reproducible data science

Authors & contribution Daniel Nüst (40%), Vanessa Sochat, Ben Marwick, Stephen J. Eglen, Tim Head, Tony Hirst, Benjamin D. Evans

Venue *PLoS Computational Biology* (SNIP 2020: 1.71) 10.1371/journal.pcbi.1008316

Date 11/2020

Repository https://github.com/nuest/ten-simple-rules-dockerfiles/

EDITORIAL

Ten simple rules for writing Dockerfiles for reproducible data science

Daniel Nüst[1]*, Vanessa Sochat[2], Ben Marwick[3], Stephen J. Eglen[4], Tim Head[5], Tony Hirst[6], Benjamin D. Evans[7]

1 Institute for Geoinformatics, University of Münster, Münster, Germany, **2** Stanford Research Computing Center, Stanford University, Stanford, California, United States of America, **3** Department of Anthropology, University of Washington, Seattle, Washington, United States of America, **4** Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Cambridge, Cambridgeshire, Great Britain, **5** Wild Tree Tech, Zurich, Switzerland, **6** Department of Computing and Communications, The Open University, Great Britain, **7** School of Psychological Science, University of Bristol, Bristol, Great Britain

* daniel.nuest@uni-muenster.de

Citation: Nüst D, Sochat V, Marwick B, Eglen SJ, Head T, Hirst T, et al. (2020) Ten simple rules for writing Dockerfiles for reproducible data science. PLoS Comput Biol 16(11): e1008316. https://doi.org/10.1371/journal.pcbi.1008316

Editor: Scott Markel, Dassault Systemes BIOVIA, UNITED STATES

Published: November 10, 2020

Funding: The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript. DN is supported by the project Opening Reproducible Research II (https://o2r.info/; https://www.uni-muenster.de/forschungaz/project/12343) funded by the German Research Foundation (DFG) under project number PE 1632/17-1. DN and SJE are supported by a Mozilla mini science grant. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

Abstract

Computational science has been greatly improved by the use of containers for packaging software and data dependencies. In a scholarly context, the main drivers for using these containers are transparency and support of reproducibility; in turn, a workflow's reproducibility can be greatly affected by the choices that are made with respect to building containers. In many cases, the build process for the container's image is created from instructions provided in a `Dockerfile` format. In support of this approach, we present a set of rules to help researchers write understandable `Dockerfiles` for typical data science workflows. By following the rules in this article, researchers can create containers suitable for sharing with fellow scientists, for including in scholarly communication such as education or scientific papers, and for effective and sustainable personal workflows.

Author summary

Computers and algorithms are ubiquitous in research. Therefore, defining the computing environment, i.e., the body of all software used directly or indirectly by a researcher, is important, because it allows other researchers to recreate the environment to understand, inspect, and reproduce an analysis. A helpful abstraction for capturing the computing environment is a container, whereby a container is created from a set of instructions in a recipe. For the most common containerisation software, Docker, this recipe is called a Dockerfile. We believe that in a scientific context, researchers should follow specific practices for writing a Dockerfile. These practices might be somewhat different from the practices of generic software developers in that researchers often need to focus on transparency and understandability rather than performance considerations. The rules presented here are intended to help researchers, especially newcomers to containerisation, leverage containers for open and effective scholarly communication and collaboration while avoiding the pitfalls that are especially irksome in a research lifecycle. The recommendations cover a deliberate approach to Dockerfile creation, formatting and style, documentation, and habits for using containers.

Introduction

Computing infrastructure has advanced to the point where not only can we share data underlying research articles, but we can also share the code that processes these data. The sharing of code files is enabled by collaboration platforms such as GitHub or GitLab and is becoming an increasingly common practice. The sharing of the computing environment is enabled by containerisation, which allows for documenting and sharing entire workflows in a comprehensive way. Importantly, this sharing of computational assets is paramount for increasing the reproducibility of computational research. While papers based on the traditional journal article format can share extensive details about the research, computational research is often far too complicated to be effectively disseminated in this format [1]. Approaches such as containerisation are needed to support computational research, or when analysing or visualising data, because a paper's actual contribution to knowledge includes the full computing environment that produced a result [2].

Containerisation helps provide instructions for packaging the building blocks of computer-based research (i.e., code, data, documentation, and the computing environment). Specifically, containers are built from plain text files that represent a human- and machine-readable recipe for creating the computing environment and interacting with data. By providing this recipe, authors of scientific articles greatly improve their work's level of documentation, transparency, and reusability. This is an important part of common practice for scientific computing [3,4]. An overall goal of these practices is to ensure that both the author and others are able to reproduce and extend an analysis workflow. The containers built from these recipes are portable encapsulated snapshots of a specific computing environment that are both more lightweight and transparent than virtual machines. Such containers have been demonstrated for capturing scientific notebooks [5] and reproducible workflows [6].

While several tutorials exist on how to use containers for reproducible research ([7–11] and Gruening and colleagues [12] give very helpful recommendations for packaging reusable software in a container), there is no detailed manual for how to write the actual instructions to create the containers for computational research besides generic best practice guides [13,14]. Here, we introduce a set of recommendations for producing container configurations in the context of data science workflows using the popular `Dockerfile` format, summarised in Fig 1.

Prerequisites and scope

To start with, we assume the existence of a scripted scientific workflow, i.e., you can, at least at a certain point in time, execute the full process with a fixed set of commands, for example, `make prepare_data` followed by `Rscript analysis.R` or only `python3 my-workflow.py`. To maximise reach, we assume that containers, which you eventually share with others, can only run open-source software; tools like Mathematica and Matlab are out of scope for this example. A workflow that does not support scripted execution is also out of scope for reproducible research, as it does not fit well with containerisation. Furthermore, workflows interacting with many petabytes of data and executed in high-performance computing (HPC) infrastructures are out of scope. Using such HPC job managers or cloud infrastructures would require a collection of "Ten Simple Rules" articles in their own right. For the HPC use case, we encourage the reader to look at Singularity [15]. For this article, we focus on workflows that typically run on single machine, e.g., a researcher's own laptop computer or a virtual

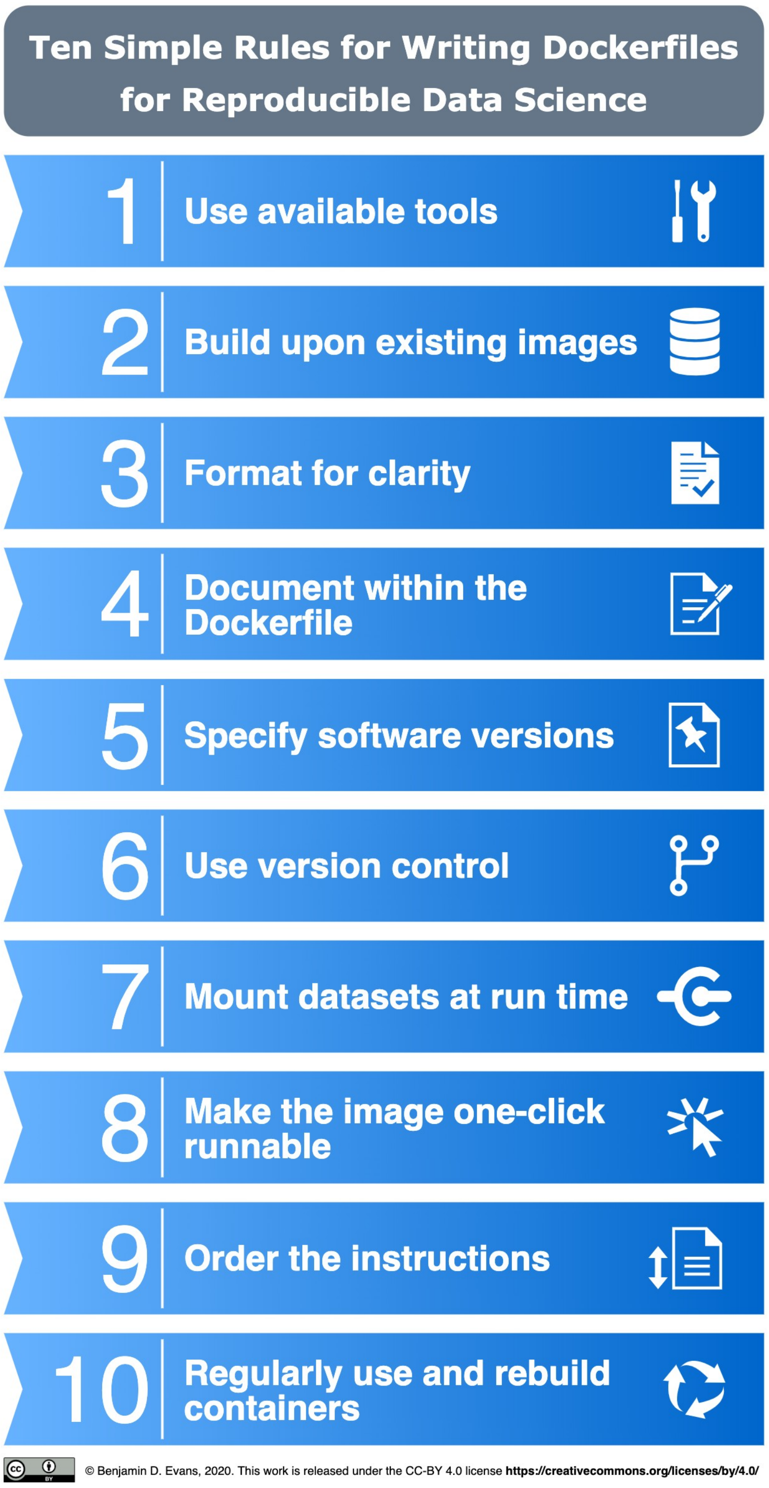

Ten Simple Rules for Writing Dockerfiles for Reproducible Data Science
1 Use available tools
2 Build upon existing images
3 Format for clarity
4 Document within the Dockerfile
5 Specify software versions
6 Use version control
7 Mount datasets at run time
8 Make the image one-click runnable
9 Order the instructions
10 Regularly use and rebuild containers
© Benjamin D. Evans, 2020. This work is released under the CC-BY 4.0 license https://creativecommons.org/licenses/by/4.0/

Fig 1. Summary of the 10 simple rules for writing `Dockerfile`s for reproducible data science.

https://doi.org/10.1371/journal.pcbi.1008316.g001 server. The reader might scope the data requirement to under a terabyte, and compute requirement to a machine with 16 cores running over the weekend.

Although it is outside the scope of this article, we point readers to `docker-compose` [16] in the case where one might need container orchestration for multiple applications, e.g., web servers, databases, and worker containers. A `docker-compose.yml` configuration file allows for defining mounts, environment variables, and exposed ports and helps users stick to "one purpose per container", which often means one process running in the container, and to combine existing stable building blocks instead of bespoke massive containers for specific purposes.

Because "the number of unique research environments approximates the number of researchers" [17], sticking to conventions helps every researcher to understand, modify, and eventually write container recipes suitable for their needs. Even if they are not sure how the underlying technology actually works, researchers should leverage containerisation following good practices. The practices that are to be discussed in this article are strongly related to software engineering in general and research software engineering in particular, which is concerned with quality, training, and recognition of software in science [18]. We encourage you to reach out to your local or national community of research software engineers (see list of organisations) if you have questions on software development in research that go beyond the rules of this work.

While many different container technologies exist, this article focuses on Docker [19]. Docker is a highly suitable tool for reproducible research (e.g., [20]), and our observations indicate it is the most widely used container technology in academic data science. The goal of this article is to guide you as you write a `Dockerfile`, the file format used to create Docker container images. The rules will help you ensure that the `Dockerfile` allows for interactive development as well as for reaching the higher goals of reproducibility and preservation of knowledge. Such practices do not generally appear in generic containerisation tutorials, and they are rarely found in the `Dockerfile`s published as part of software projects that are often used as templates by novices. The differences between a helpful, stable `Dockerfile` and one that is misleading, prone to failure, and full of potential obstacles are not obvious, especially for researchers who do not have extensive software development experience or formal training. By committing to this article's rules, one can ensure that their workflows are reproducible and reusable, that computing environments are understandable by others, and that researchers have the opportunity to collaborate effectively. Applying these rules should not be triggered by the publication of a finished project but should instead be weaved into day-to-day habits (cf. thoughts on openness as an afterthought by [21] and on computational reproducibility by [2]).

Docker and Dockerfiles

Docker [19] is a container technology that has been widely adopted and is supported on many platforms, and it has become highly useful for research. Containers are distinct from virtual machines or hypervisors, as they do not emulate hardware or operating system kernels and hence do not require the same system resources. Several solutions for facilitating reproducible research are built on top of containers [17,22–25], but these solutions intentionally hide most of the complexity from the researcher.

To create Docker containers for specific workflows, we write text files that follow a particular format called `Dockerfile` [26]. A `Dockerfile` is a machine- **and** human-readable recipe for building **images**. Here, images are executable files that include the application, e.g., the programming language interpreter needed to run a workflow, and the system libraries required by an application to run. Thus, a `Dockerfile` consists of a sequence of instructions to copy files and install software. Each instruction adds a layer to the image, which can be

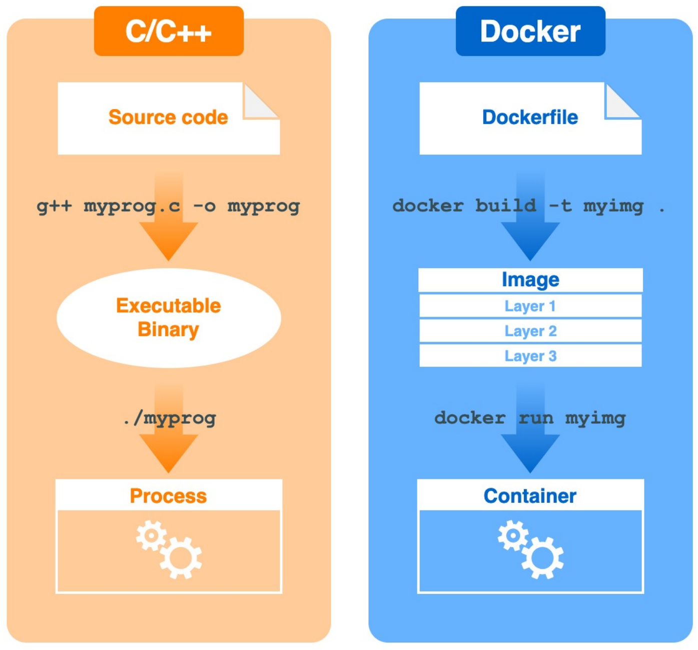


Fig 2. The workflow to create Docker containers by analogy. Containers begin with a `Dockerfile`, a recipe for building the computational environment (analogous to source code in a compiled programming language). This is used to build an image with the `docker build` command, analogous to compiling the source code into an executable (binary) file. Finally, the image is used to launch one or more containers with the `docker run` command (analogous to running an instance of the compiled binary as a process).

https://doi.org/10.1371/journal.pcbi.1008316.g002

cached across image builds for minimising build and download times. Once an image is built or downloaded, it is then launched as a running instance known as a **container**. The images have a main executable exposed as an “entrypoint” that is started when they are run as stateful containers. Further, containers can be modified, stopped, restarted, and purged.

A visual analogy for building and running a container is provided in Fig 2. Akin to compiling source code for a programming language, creating a container also starts with a plain text file (`Dockerfile`), which provides instructions for building an image. Similar to using a compiled binary file to launch a program, the image is then run to create a container instance. See Listing 1 for a full `Dockerfile`, which we will refer to throughout this article.

While Docker was the original technology to support the `Dockerfile` format, other container technologies now offer support for it, including podman/buildah supported by RedHat, kaniko, img, and buildkit. The container software Singularity [15], which is optimised for scientific computing and the security needs of HPC environments, uses its own format, called the Singularity recipe, but it can also import and run Docker images. The rules here are, to some extent, transferable to Singularity recipes.

While some may argue against publishing reproducibly, e.g., due to a lack of time and incentives, a reluctance to share (cf. [28]), and the substantial technical challenges involved in maintaining software and documentation, it should become increasingly straightforward for the average researcher to provide computational environment support for their publication in the form of a `Dockerfile`, a pre-built Docker image, or another type of container. If a researcher can find and create containers or write a `Dockerfile` to address their most common use cases, then, arguably, sharing it would not make for extra work after this initial setup (cf. README.md of [29]). In fact, the Dockerfile itself represents powerful documentation to show from where data and code were derived, i.e., downloaded or installed, and, consequently, where a third party might obtain the data again.

Listing 1. Dockerfile full example. The Dockerfile and all other files are published in the full-demo example, see Section Examples; the image docker.io/nuest/datascidockerfiles:1.0.0 is a ready-to-use build of this example.

```
FROM docker.io/rocker/verse:3.6.2

### INSTALL BASE SOFTWARE ###################################################
# Install Java, needed for package rJava
RUN apt-get update && \
  apt-get install -y default-jdk && \
  rm -rf /var/lib/apt/lists/*

### INSTALL WORKFLOW TOOLS ##################################################
# Install system dependencies for R packages
RUN apt-get update && \
  apt-get install -y \
    # needed for RNetCDF, found via https://sysreqs.r-hub.io/pkg/RNetCDF
    libnetcdf-dev libudunits2-dev \
    # needed for git2r:
    libgit2-dev
# Install R packages, based on https://github.com/rocker-org/geospatial/blob/master/Dockerfile
RUN install2.r --error \
    RColorBrewer \
    RNetCDF \
    git2r \
    rJava

WORKDIR /tmp

# Install Python tools and their system dependencies
RUN apt-get update && \
  apt-get install -y python-pip && \
  rm -rf /var/lib/apt/lists/*
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt

# Download superduper image converter
RUN wget https://downloads.apache.org/pdfbox/2.0.19/pdfbox-app-2.0.19.jar

### ADD MY OWN SCRIPTS #######################################################
# Add workflow scripts
WORKDIR /work
COPY myscript.sh myscript.sh
COPY analysis.py analysis.py
COPY plots.R plots.R

# Configure workflow
```

```
 ENV DATA_SIZE 42

 # Uncomment the following lines to execute preprocessing tasks during build
 #RUN python analysis.py
 #RUN Rscript plots.R

 ### WORKFLOW CONTAINER FEATURE ############################################
 # CMD from base image used for development, uncomment the following lines to
 # have a "run workflow only" image
 # CMD["./myscript.sh"]

 ### Usage instructions ####################################################
 # Build the images with
 # > docker build --tag datascidockerfiles:1.0.0.
 # Run the image interactively with RStudio, open it on http://localhost/
 # > docker run -it -p 80:8787 -e PASSWORD = ten --volume $(pwd)/input:/input
datascidockerfiles:1.0.0
 # Run the workflow:
 # > docker run -it --name gwf datascidockerfiles:1.0.0 /work/myscript.sh
 # Extract the data:
 # > docker cp gwf:/output/ ./outputData
 # Extract the figures:
 # > docker cp gwf:/work/figures/ ./figures
```

Rule 1: Use available tools

Rule 1 could informally be described as "Don't bother to write a Dockerfile!". Writing a `Dockerfile` from scratch can be difficult, and even experts sometimes take shortcuts. A good initial strategy is to look at tools that can help generate a `Dockerfile` for you. The developers of such tools have likely thought about and implemented good practices, and they may even have incorporated newer practices when reapplied at a later point in time. Therefore, the most important rule is to apply a multistep process to creating a `Dockerfile` for your specific use case.

First, you want to determine whether there is an existing image that you can use; if so, you want to be able to use it and add the instructions for doing so to your workflow documentation. As an example, you might be doing some kind of interactive development. For interactive development environments such as notebooks and development servers or databases, you can readily find images that come installed with all the software that you need. You can look for information about images in (a) the documentation of the software you intend to use; (b) the Docker image registry Docker Hub; or (c) the source code projects of the software being used, as many developers today rely on containers for development, testing, and teaching.

Second, if there is no suitable preexisting image for your needs, you might next look to well-maintained tools to help with `Dockerfile` generation. These tools can add required software packages to an existing image without you having to manually write a `Dockerfile` at all. "Well-maintained" not only refers to the tool's own stability and usability but also indicates that suitable base images are used, typically from the official Docker library [30], to ensure that the container has the most recent security fixes for the operating system in question. See the next section "Tools for container generation" for details.

Third, if these tools do not meet your needs, you may want to write your own `Dockerfile`. In this case, follow the remaining rules.

Tools for container generation

`repo2docker` [25] is a tool maintained by Project Jupyter that can help to transform a source code or data repository, e.g., GitHub, GitLab, or Zenodo, into a container. The tool relies on common configuration files for defining software dependencies and versions, and it supports a few more special files; see the supported configuration files. As an example, we might install `jupyter-repo2docker` and then run it against a repository with a `requirements.txt` file, an indication of being a Python workflow with dependencies on the Python Package Index (PyPI), using the following command:

```
jupyter-repo2docker https://github.com/norvig/pytudes
```

The resulting container image installs the dependencies listed in the requirements file, and it provides an entrypoint to run a notebook server to interact with any existing workflows in the repository. Since `repo2docker` is used within MyBinder.org, if you make sure your workflow is "Binder-ready", you and others can also obtain an online workspace with a single click. However, one precaution to consider is that the default command above will create a home for the current user, meaning that the container itself would not be ideal to share; instead, any researcher interested in interacting with the code inside should run `repo2-docker` themselves and create their own container. Because `repo2docker` is deterministic, the environments are the same (see Rule 5 for ensuring the same software versions).

Additional tools to assist with writing `Dockerfile`s include `containerit` [31] and `dockta` [32]. `containerit` automates the generation of a stand-alone Dockerfile for workflows in R. This utility can provide a starting point for users unfamiliar with writing a `Dockerfile`, or it can, together with other R packages, provide a full image creation and execution process without having to leave an R session. `dockta` supports multiple programming languages and configurations files, just as `repo2docker` does, but it attempts to create a readable `Dockerfile` compatible with plain Docker and to improve user experience by cleverly adjusting instructions to reduce build time. While perhaps more useful for fine-tuning, linters can also be helpful when writing Dockerfiles, by catching errors or non-recommended formulations (see Rule 10).

Tools for templating

It is likely that over time you will work on projects and develop images that are similar in nature to each other. To avoid constantly repeating yourself, you should consider adopting a standard workflow that will give you a quick start for a new project. As an example, cookie cutter templates [33] or community templates (e.g., [34]) can provide the required structure and files (e.g., for documentation, continuous integration (CI), and licenses), for getting started. If you decide to build your own cookie cutter template, consider collaborating with your community during development of the standard to ensure it will be useful to others.

Part of your project template should be a protocol for publishing the `Dockerfile` and even exporting the image to a suitable location, e.g., a container registry or data repository, taking into consideration how your workflow can receive a DOI for citation. A template is preferable to your own set of base images because of the maintenance efforts the base images require. Therefore, instead of building your own independent solution, consider contributing to existing suites of images (see Rule 2) and improving these for your needs.

For any tool that you use, be sure to look at documentation for usage and configuration options, and look for options to add metadata (e.g., labels; see Rule 4).

Rule 2: Build upon existing images

Many pre-built community and developer contributed Docker images are publically available for anyone to pull, run and extend, without having to replicate the image construction process. However, a good understanding of how **base images** and **image tags** work is crucial, as the image and tag that you choose has important implications for your derived images and containers. It is good practice to use base images that are maintained by the Docker library, so called "official images" [35], which benefit from a review for best practices and vulnerability scanning [13]. You can identify these images by the missing user portion of the image name, which comes before the `/`, e.g., `r-base` or `python`. However, these images only provide basic programming languages or very widely used software, so you will likely use images maintained by organisations or fellow researchers.

While some organisations can be trusted to update images with security fixes (see list below), for most individual accounts that provide ready-to-use images, it is likely that these will not be updated regularly. Further, it's even possible that an image or a `Dockerfile` could disappear, or an image could be published with malicious intent (though we have not heard of any such case in academia). Therefore, for security, transparency, and reproducibility, you should only use images where you have access to the `Dockerfile`. In case a repository goes away, we suggest that you save a copy of the `Dockerfile` within your project (see Rule 7).

The following list is a selection of communities that produce widely used, regularly updated images, including ready-to-use images with preinstalled collections of software configured to work out of the box. Do take advantage of such images, especially for complex software environments, e.g., machine learning tool stacks, or a specific BLAS library.

- Rocker for R and RStudio images [20]
- Bioconductor Docker images for bioinformatics with R
- NeuroDebian images for neuroscience [36]
- Jupyter Docker Stacks for Notebook-based computing
- Taverna Server for running Taverna workflows

For example, here is how we would use a base image `verse`, which provides the popular Tidyverse suite of packages [37], with `R version 3.5.2` from the `rocker` organisation on Docker Hub (`docker.io`, which is the default and can be omitted).

```
FROM docker.io/rocker/verse:3.6.2
```

Use version-specific tags

Images have **tags** associated with them, and these tags have specific meanings, e.g., a semantic version indicator such as `3.7` or `dev`, or variants like `slim` that attempt to reduce image size. Tags are defined at the time of image build and appear in the image name after the `:` when you use an image, e.g., `python:3.7`. By convention a missing tag is assumed to be the word `latest`, which gives you the latest updates but is also a moving target for your computing environment that can break your workflow. Note that a version tag means that the tagged software is frozen, but it does not mean that the image will not change, as backwards compatible fixes (cf. semantic versioning, [38]), e.g., version `1.2.3` that fixes a security problem in version `1.2.2` or updates to an underlying system library, would be published to the parent tag `1.2`.

For data science workflows, you should always rely on version-specific image tags, both for base images that you use and for images that you build yourself and then run (see usage instructions in Listing 1 for an example of the `--tag` parameter of `docker build`). When keeping different versions (tags) available, it is good practice to publish an image in an image registry. For details, we refer you to the documentation on automated builds, see Docker Hub Builds or GitLab's Container Registry as well as CI services such as GitHub actions, or CircleCI that can help you get started. Do not `docker push` a locally built image, because that counteracts the considerations outlined above. If a pre-built image is provided in a public image registry, do not forget to direct the user to it in your documentation, e.g., in the `README` file or in an article.

Rule 3: Format for clarity

First, it is good practice to think of the `Dockerfile` as a human- **and** machine-readable file. This means that you should use indentation, new lines, and comments to make your `Dockerfile` well documented and readable. Specifically, carefully indent commands and their arguments to make clear what belongs together, especially when connecting multiple commands in a `RUN` instruction with `&&`. Use `\` at the end of a line to break a single command into multiple lines. This will ensure that no single line gets too long to comfortably read. Content spread across more and shorter lines also improves readability of changes in version control systems. Further, use long versions of parameters for readability (e.g`.,` `--input` instead of `-i`). When you need to change a directory, use `WORKDIR`, because it not only creates the directory if it does not exist but also persists the change across multiple `RUN` instructions.

Second, clarity of the steps within a Dockerfile is most important, and if it requires verbosity or adds to the final image size, that is an acceptable trade-off. For example, if your container uses a script to run a complex install routine, instead of removing it from the container upon completion (a practice commonly seen in production `Dockerfile`s aiming at small image sizes, cf. [12]), you should keep the script in the container for a future user to inspect; the script size is negligible compared to the image size. One common pattern you will encounter is a single and very lengthy `RUN` instruction chaining multiple commands to install software and clean up afterwards. For example (a) the instruction updates the database of available packages, installs a piece of software from a package repository, and purges the cache of the package manager; or (b) the instruction downloads a software's source archive, unpacks it, builds and installs the software, and then removes the downloaded archive and all temporary files. Although this pattern creates instructions that may be hard to read, it is very common and can even increase clarity within the image file system because installation and build artifacts are gone. In general, if your container is mostly software dependencies, you should not need to worry about image size because (a) your data is likely to have much larger storage requirements; and (b) transparency and inspectability outweigh storage concerns in data science. If you really need to reduce the size, you may look into using multiple containers (cf. [12]) or multistage builds [39].

Depending on the programming language used, your project may already contain files to manage dependencies, and you may use a package manager to control this aspect of the computing environment. This is a very good practice and helpful, though you should consider the externalisation of content to outside of the `Dockerfile` (see Rule 7). Often, a single long `Dockerfile` with sections and helpful comments can be more understandable than a collection of separate files.

Generally, aim to design the `RUN` instructions so that each performs one scoped action, e.g., download, compile, and install one tool. This makes the lines of your `Dockerfile` a well-documented recipe for the user as well as a machine. Each instruction will result in a new layer, and reasonably grouped changes increase readability of the `Dockerfile` and facilitate inspection of the image, e.g., with tools like dive [40]. Convoluted `RUN` instructions can be acceptable to reduce the number of layers, but careful layout and consistent formatting should be applied.

Although you will find `Dockerfiles` that use build-time variables to dynamically change parameters at build time, such a customisation option reduces clarity for data science workflows.

Rule 4: Document within the Dockerfile

Explain in comments

As you are writing the `Dockerfile`, be mindful of how other people (including future you!) will read it and why. Are your choices and commands being executed clearly, or are further comments warranted? To assist others in making sense of your `Dockerfile`, you can add comments that include links to online forums, code repository issues, or version control commit messages to give context for your specific decisions. For example, this `Dockerfile` by Kaggle does a good job of explaining the reasoning behind the contained instructions. If you copy instructions from another `Dockerfile`, acknowledge the source in a comment. Also, it can be helpful to include comments about commands that did not work so you do not repeat past mistakes. Further, if you find that you need to remember an undocumented step, that is an indication this step should be documented in the `Dockerfile`. All instructions can be grouped starting with a short comment, which also makes it easier to spot changes if your `Dockerfile` is managed in some version control system (see Rule 6). Listing 2 shows a selection of typical kinds of comments that are useful to include in a `Dockerfile`.

Listing 2. Partial **`Dockerfile`** with examples for helpful comments.

```
...
# apt-get install specific version, use 'apt-cache madison <pkg>'
# to see available versions
RUN apt-get install python3-pandas = 0.23.3+dfsg-4ubuntu1
# install required R packages; before log the used repository
# for better provenance in the build log
RUN R -e 'getOption("repos")' && \
  install2.r \
    fortunes \
    here
# this library must be installed from source to get version newer
# than in apt sources
RUN git clone http://url.of/repo && \
  cd repo && \
  make build &&
  make install
```

Add metadata as labels

Docker automatically captures useful information in the image metadata, such as the version of Docker used for building the image. The `LABEL` instruction can add **custom metadata** to images. You can view all labels and other image metadata with `docker inspect` command. Listing 3 shows the most relevant ones for data science workflows. Labels serve as structured metadata that can be leveraged by services, e.g., https://microbadger.com/labels. For example, software versions of containerised applications (cf. [12]), licenses, and maintainer contact information are commonly seen, and they are very useful if a `Dockerfile` is discovered out of context. Regarding licensing information, this should include the license of your own code and could point to a `LICENSE` file within the image (cf. [12]). While you can add arbitrarily complex information with labels, for data science scenarios the user-facing documentation is much more important. Relevant metadata that might be utilised with future tools include global identifiers such as ORCID identifiers, DOIs of the research compendium (cf. https://research-compendium.science), e.g., reserved on Zenodo, or a funding agency's grant number. You can use the `ARG` instruction to pass variables at build time, for example, to pass values into labels, such as the current date or version control revision. However, a script or `Makefile` might be required so that you do not forget that you set the argument, or how you set it (see Rule 10).

The Open Container Initiative (OCI) Image Format Specification provides some common label keys (see the "Annotations" section in [41]) to help standardise field names across container tools, as shown below. Some keys hold specific content, e.g., `org.opencontainers.image.documentation` is a URL as character string pointing to documentation on the image, and `org.opencontainers.image.licenses` is the SPDX license identifier. You may also commonly find labels in the deprecated `org.label-schema-specification` format, e.g., `org.label-schema.description`. However, we encourage the use of the OCI schema in all new and unlabelled projects.

Listing 3. Partial `Dockerfile` with commonly used labels; note the line breaks within the values (using the \ character), which were added to limit line length, are not preceded by a space character, as this space would appear in the value, whereas the line breaks between keys and values are separated by white space for readability.

```
...
LABEL maintainer = "D. Nüst <daniel.nuest@uni-muenster.de>" \
  org.opencontainers.image.authors = "Nüst (daniel.nuest@uni-muenster.de), \
Sochat, Marwick, Eglen, Head, Hirst, and Evans" \
  org.opencontainers.image.url = "\
https://github.com/nuest/ten-simple-rules-dockerfiles"
  org.opencontainers.image.documentation = "https://nuest.github.io/\
ten-simple-rules-dockerfiles/ten-simple-rules-dockerfiles.pdf"
  org.opencontainers.image.version = "1.0.0"
LABEL org.opencontainers.image.vendor = "Ten Simple Institute, Uni of Rules"
  org.opencontainers.image.description = "Reproducible workflow image"
  org.opencontainers.image.licenses = "Apache-2.0"
LABEL edu.science.data.group.project = "Find out something (Grant #123456)"
  edu.science.data.group.name = "Data Science Lab" \
  author.orcid = "0000-0002-1825-0097"
```

Define versions, parameters, and paths once

The `ENV` instruction in a `Dockerfile` allows for defining environment variables. These variables persist inside the container and can be useful, for example, for (a) setting software versions or paths and reusing them across multiple instructions to avoid mistakes; (b) specifying metadata intended to be discovered by installed libraries or software; or (c) adding binaries to the path (`PATH`) or library path (`LD_LIBRARY_PATH`). You should be careful to distinguish these environment variables from those that might vary and be required at runtime. Listing 4 shows some examples. For runtime environment variables, either to set a new variable or override one set in the container, you can use the `--env` parameter of `docker run` (see Listings 4 and 6).

Listing 4. Partial `Dockerfile` showing usage of environment variables with the `ENV` instruction.

```
...
# Define number of cores used by PowerfulAlgorithm
ENV POWER_ALG_CORES 2

# Install UsefulSoft tool in specific version from source
ENV USEFULSOFT_VERSION = 1.0.0 \
  USEFULSOFT_INSTALLDIR = /workspace/bin

RUN wget http://usesoft.url/useful_software/$USEFULSOFT_VERSION/useful-
$USEFULSOFT_VERSION.zip && \
  unzip useful-$USEFULSOFT_VERSION.zip -d useful-src && \
  cd useful-src && \
  bash install.sh --target $USEFULSOFT_INSTALLDIR && \
  cd .. && \
  rm -r useful-src useful-$USEFULSOFT_VERSION.zip

# Put UsefulSoft tool on the path for subsequent instructions
ENV PATH $PATH:$USEFULSOFT_INSTALLDIR

### Usage instructions ###
# [...]
# Run the image (defining the number of cores used):
# > docker run --it --env POWER_ALG_CORES 32 my_workflow
```

Include usage instructions

It is often helpful to provide usage instructions, i.e., how to `docker build` and `docker run` the image, **within** the `Dockerfile`, either at the top or bottom where the reader is likely to find them. Such documentation is especially relevant if bind mounts, specific names, or ports are important for using the container; see, for example, the final lines of Listing 1. These instructions are not limited to `docker <command>` but include the usage of bespoke scripts, a `Makefile`, or `docker-compose` (see Rule 8 and Rule 10). Following a common coding aphorism, we might say "A Dockerfile you wrote three months ago may just as well have been written by someone else." Thus, usage instructions help others, because they quickly get them running your workflow and interacting with the container in the intended way without reading all of the instructions (a "tl;dr"-kind of usage). Usage instructions also provide a de facto way of testing that your container works in a way that others can try out. The `Dockerfile` alongside your documentation strategy is a demonstration of your careful work habits and good intentions for transparency and computational reproducibility.

Rule 5: Specify software versions

The reproducibility of your `Dockerfile` heavily depends on how well you define the versions of software to be installed in the image. The more specifically you can define them the better, because using the desired version leads to reproducible builds. The practice of specifying versions of software is called **version pinning** (e.g., on `apt`: https://blog.backslasher.net/my-pinning-guidelines.html). For stable workflows in a scientific context, it is generally advised to freeze the computing environment explicitly and not rely on the "current" or "latest" software, which is a moving target.

System libraries

System library versions can largely come from the base image tag that you choose to use, e.g., `ubuntu:18.04`, because the operating system's software repositories are very unlikely to introduce breaking changes but will predominantly fix errors with newer versions. However, you can also install specific versions of system packages with the respective package manager. For example, you might want to demonstrate a bug, prevent a bug in an updated version, or pin a working version if you suspect an update could lead to a problem. Generally, system libraries are more stable than software modules supporting analysis scripts, but in some cases, they can be highly relevant to your workflow. Installing from source is a useful way to install very specific versions, but it comes at the cost of longer build time and more complex instructions. Here are some examples of terminal commands that will list the currently installed versions of software on your system:

- Debian/Ubuntu: `dpkg --list`
- Alpine: `apk -vv info|sort`
- CentOS: `yum list installed` or `rpm -qa`

When you install several system libraries, it is good practice to add comments about why the dependencies are needed (see Listing 1). This way, if a piece of software is removed from the container, it will be easier to remove the system dependencies that are no longer needed, thereby reducing maintenance overhead: you will not unnecessarily fix problems with a library that is no longer needed or include long-running installations. A test provided via a `HEALTHCHECK` [42] can further ensure proper functioning of your container.

Version control

Software can often be installed directly from a version controlled repository (e.g., GitHub, GitLab, or Mercurial). It's recommended to check out a specific version, tag, or commit to ensure pinning a version for the repository. For example, here is how to clone a specific release tag (`v3.6.1`) of the Singularity container software:

```
RUN git clone -b v3.6.1 https://github.com/hpcng/singularity
```

In the case that you want to clone and check out a specific commit, you can use the `checkout` command.

```
RUN git clone https://github.com/hpcng/singularity && \
  cd singularity && \
  git checkout 8a92cf127a49118cab61579bb36b3d51ba5c6434 && \
  # install steps go here \
```

Extension packages and programming language modules

If you need to install packages or dependencies for a specific language, package managers are a good option. Package managers generally provide reliable mirrors or endpoints to download software; many packages are tested before release, and, most importantly, they provide access to specific versions. Most package managers have a command line interface that can be used from `RUN` commands in your `Dockerfile`, along with various flavours of "freeze" commands that can output a text file listing all software packages and versions (cf. https://markwoodbridge.com/2017/03/05/jupyter-reproducible-science.html cited by [5]). The biggest risk with using package managers with respect to a `Dockerfile` is outsourcing configuration. As an example, here are configuration files supported by commonly used languages in scientific programming:

- Python: `requirements.txt` (pip tool, [43]) and `environment.yml` (Conda, [44])
- R: `DESCRIPTION` file format [45] and r ("little R", [46])
- JavaScript: `package.json` of `npm` [47]
- Julia: `Project.toml` and `Manifest.toml` [48].

In some cases (e.g., Conda) the package manager is also able to make decisions about what versions to install, which is likely to lead to a non-reproducible build. For this reason, it is necessary to pin the dependency versions. In the case of having few packages, it may be simplest to write the install steps and versions directly into the `Dockerfile` (also for clarity, see Rule 3):

```
RUN pip install \
geopy = = 1.20.0 \
uszipcode = = 0.2.2
```

Alternatively, versions may be specified in a separate dependency file (e.g., `requirements.txt` or `environment.yml`) and `COPY`ied to the image for installation:

```
COPY requirements.txt.
RUN pip install -r requirements.txt
```

This modularisation may reduce readability, but provides more flexibility in facilitating different ways of building a reproducible environment, provided the dependency file is under version control in the same repository (see Rule 6). You can also use package managers to install software from source code `COPY`ied into the image (see Rule 7). Finally, you can use many package managers to install software from source obtained from external code management repositories, e.g., installing a tool from a specific version tag or commit hash. Be aware of the risk that such installations may later fail, especially when the external repositories are out of your control. However, these concerns can be mitigated by running the installation command with the full URL (including the specific version tag or commit hash), which is helpful in troubleshooting if problems arise. The version pinning capabilities of these file formats and package managers are described in their respective documentation.

As a final note on software installation, you should be aware of the `USER` instruction in a `Dockerfile` and how your base image might change the user for particular instructions, restricting which commands can be run within the container. It is common to use images with the default user `root`, which is required for installing system dependencies. However, you may encounter base images running as a non-root user (e.g., in the Jupyter and Rocker image stacks) in order to avoid permission problems when mounting files into the container, especially for "output" files (see Rule 7). We recommend ensuring that the image works without specifying any users, and, if your image deviates from that, we suggest you document it precisely.

Rule 6: Use version control

As plain text files, `Dockerfile`s are well suited for use with version control systems. Including a `Dockerfile` alongside your code and data is an effective way to consistently build your software, to show visitors to the repository how it is built and used, to solicit feedback and collaborate with your peers, and to increase the impact and sustainability of your work (cf. [49]).

Most importantly, you should publish **all** files `COPY`ied into the image, e.g., test data or files for software installation from source (see Rule 7), in the same public repository as the `Dockerfile`, e.g., in a research compendium. If you prefer to edit your scripts more interactively in a running container (e.g., using Jupyter), then it may be more convenient to bind mount their directory from the host at run time, provided all changes are committed before sharing.

Online collaboration platforms (e.g., GitHub, GitLab) also make it easy to use CI services to test building and executing your image in an independent environment. CI increases stability and trust, and it allows for images to be published automatically. Automation strategies exist to build and test images for multiple platforms and software versions, even with CI. Such approaches are often used when developing popular software packages for a broad user base operating across a wide range of target platforms and environments, and they can be leveraged if you expect your workflow to fall into this category. Furthermore, the commit messages in your version-controlled repository preserve a record of all changes to the `Dockerfile`, and you can use the same versions in tags for both the container's image and the git repository.

Rule 7: Mount datasets at run time

The role of containers is to provide the computing environment, not to encapsulate (potentially very large) datasets. It is better to insert large data files from the local machine into the container at runtime, and use the image primarily for the software and dependencies. This insertion is achieved by using **bind mounts**. Mounting these files is preferable to using the `ADD`/`COPY` instructions in the `Dockerfile`, because files persist when the container instance or image is removed from your system, and the files are more accessible when the workspace is published. If you want to add local files to the container (and do not need `ADD`'s extra features), we recommend `COPY` because it is simpler and explicit. Volumes are useful for persisting changes across runs of a container and offer faster file I/O compared to other mounting methods (particularly useful with databases for example). However, they are less suitable for reproducibility, since these changes exist within the image (making them less in line with treating containers as ephemeral; see Rule 10) and are not so easy to access or place under version control. Unless specific features are needed, bind mounts are preferable to storage volumes since the contents are directly accessible from both the container and the host. The files can also be more easily included in the same repository.

Storing **data files** outside of the container allows handling of very large or sensitive datasets, e.g., proprietary data or private information. Do not include such data in an image! To avoid publishing sensitive data by accident, you can add the data directory to the `.dockerignore` file, which excludes files and directories from the build context, i.e., the set of files considered by `docker build`. Ignoring data files also speeds up the build in cases where there are very large files or many small files. As an exception, you should include dummy or small test datasets in the image to ensure that a container is functional without the actual dataset, e.g., for automated tests, instructions in the user manual, or peer review (see also "functional testing logic" in [12]). For all these cases, you should provide clear instructions in the `README` file on how to use the actual (or dummy) data, and how to obtain and mount it if it is kept outside of the image. When publishing your workspace, e.g., on Zenodo, having datasets outside of the container also makes them more accessible to others, for example, for reuse or analysis.

A mount can also be used to access **output data** from a container; this can be an extra mount or the same `data` directory. Alternatively, you can use the `docker cp` command to access files from a running or stopped container, but this requires a specific handling, e.g., naming the container when starting it or using multiple shells, which requires very detailed instructions for users.

You can use the `-v/--volume` or preferably `--mount` flags to `docker run` to configure bind mounts of directories or files [50], including options, as shown in the following examples. If the target path exists within the image, the bind mount will replace it for the started container. (Note, `$HOME` is an environment variable in UNIX systems representing the path to the current user's home directory, e.g., `/home/moby`, and `$(pwd)` returns the current path).

```
  # mount directory
  docker run --mount type = bind,source = $HOME/project,target = /project
mycontainer

  # mount directory as read-only
  docker run --mount type = bind,src = $HOME/project,dst = /workspace,readonly
mycontainer

  # mount multple directories, one with write access relative to current path
(Linux)
  docker run --mount type = bind,src = $(pwd)/article-x-supplement/data,dst =
/input-data,readonly \-mount type = bind,src = $(pwd)/outputs,dst = /output-
data mycontainer
```

How your container expects external resources to be mounted into it should be included in the example commands (see Rule 4). In these commands, you can also make sure to avoid issues with file permissions by using Docker's `--user` option. For example, by default, writing a new file from inside the container will be owned by user `root` on your host, because that is the default user within the container.

Rule 8: Make the image one-click runnable

Containers are very well suited for day-to-day development tasks (see also Rule 10), because they support common interactive environments for data science and software development. But they are also useful for a "headless" execution of full workflows. For example, [51] demonstrates a container for running an agent-based model with video files as outputs, and this article's R Markdown source, which included cells with analysis code, rendered into a PDF in a container. A workflow that does not support headless execution may even be seen as irreproducible.

These 2 usages can be configured by the `Dockerfile`'s author and exposed to the user based on the `Dockerfile`'s `ENTRYPOINT` and `CMD` instructions. An image's main purpose is reflected by the default process and configuration, though the `ENTRYPOINT` and `CMD` can also be changed at runtime. It is considered good practice to have a combination of default entrypoint and command that meets reasonable user expectations. For example, a container known to be a workflow should execute the entrypoint to the workflow and perhaps use `--help` as the command to print out usage. The container entrypoint should **not** execute the workflow, as the user is likely to run the container for basic inspection, and starting an analysis as a surprise that might write files is undesired. As the maintainer of the workflow, you should write clear instructions for how to properly interact with the container, both for yourself and others. A possible weakness with using containers is that they can only provide one default entrypoint and command. However, tools, e.g., The Scientific Filesystem [52], have been developed to expose multiple entrypoints, environments, help messages, labels, and even install sequences. With plain Docker, you can override the defaults as part of the `docker run` command or in an extra `Dockerfile` using the primary image as a base, as shown in Listing 5. In any case, you should document different variants very well and potentially capture build and run commands in a `Makefile` [27]. If you use a `Makefile`, then keep it in the same repository (see Rule 7), and include instructions for its usage (see Rule 4). To support more complex configuration options, it is helpful to expose settings via a configuration file, which can be bind mounted from the host [51], via environment variables (see Rule 4 and [53]), or via wrappers using Docker, such as Kliko [54].

Listing 5. Workflow `Dockerfile` and derived "runner image" `Dockerfile` with file name `Dockerfile.runner`.

```
#----- File: Dockerfile ------------------------
# base image (interactive)
FROM jupyter/datascience-notebook:python-3.7.6

# Usage instructions:
# docker build --tag workflow:1.0.
# docker run workflow:1.0

#----- File: Dockerfile.runner ----------------
# interactive image
FROM workflow:1.0

ENTRYPOINT ["python"]
CMD ["/workspace/run-all.sh"]

# Usage instructions:
# docker build --tag workflow-runner:1.0 --file Dockerfile.runner.
# docker run -e ITERATIONS = 10 -e ALGORITHM = advanced \
   --volume /tmp/results:/workspace/output_data workflow-runner:1.0
```

Interactive graphical interfaces, such as RStudio, Jupyter, or Visual Studio Code, can run in a container to be used across operating systems and both locally and remotely via a regular web browser. The HTML-based user interface is exposed over HTTP. Use the `EXPOSE` instruction to document the ports of interest for both humans and tools, because they need to be bound to the host to be accessible to the user using the `docker run` option `-p/--publish <host port>:<container port>`. The container should also print to the screen of the used ports along with any login credentials needed. For example, this is done in the last few lines of the output of running a Jupyter Notebook server locally (lines abbreviated):

```
docker run -p 8888:8888 jupyter/datascience-notebook:7a0c7325e470
[...]
[I 15:44:31.323 NotebookApp] The Jupyter Notebook is running at:
[I 15:44:31.323 NotebookApp] http://9027563c6465:8888/?token=6a92d [..]
[I 15:44:31.323 NotebookApp] or http://127.0.0.1:8888/?token=6a92 [..]
[I 15:44:31.323 NotebookApp] Use Control-C to stop this server and [..]
```

A person who is unfamiliar with Docker but wants to use your image may rely on graphical tools like ContainDS, Portainer, or the Docker Desktop Dashboard for assistance in managing containers on their machine without using the Docker CLI. Such tools will often detect exposed ports and declared volumes so as to make the user aware of them.

Interactive usage of a command-line interface is quite straightforward to access from containers, if users are familiar with this style of user interface. Running the container will provide a shell where a tool can be used and where help or error messages can assist the user. For example, complex workflows in any programming language can, with suitable pre-configuration, be triggered by running a specific script file. If your workflow can be executed via a command line client, you may use that to validate correct functionality of an image in automated builds, e.g., by using a small toy example and checking the output by checking successful responses from HTTP endpoints provided by the container, such as with an HTTP response code of `200`, or by using a browser automation tool such as Selenium [55].

The following example runs a simple R command counting the lines in this article's source file. The file path is passed as an environment variable.

Listing 6. Passing a parameter via environment variable; working code in example "pass-parameter-env", see Examples.

```
docker run \
  --env CONFIG_PARAM = "/data/ten-simple-rules-dockerfiles.Rmd" \
  --volume $(pwd):/data \
  jupyter/datascience-notebook:7a0c7325e470 \
  R --quiet -e "l = length(readLines(Sys.getenv('CONFIG_PARAM'))); \
    print(paste('Number of lines: ', l))"

> l = length(readLines(Sys.getenv('CONFIG_PARAM')));
> print(paste('Number of lines: ', l))
[1] "Number of lines: 568"
```

If there is only a regular desktop application, the host's window manager can be connected to the container. Although this raises notable security issues, they can be addressed by using the "X11 forwarding" natively supported by Singularity [56], which can execute Docker containers, or by leveraging supporting tools such as `x11docker` [57]. Other alternatives include bridge containers [58] and exposing a regular desktop via the browser (e.g., for Jupyter Hub [59]). This variety of approaches renders seemingly more convenient uncontainerised environments unnecessary. Just using one's local machine is only slightly more comfortable but much less reproducible and portable.

Rule 9: Order the instructions

You will regularly build an image during development of your workflow. You can take advantage of **build caching** to avoid execution of time-consuming instructions, e.g., install from a remote resource or copying a file that gets cached. Therefore, you should keep instructions in order of least likely to change to most likely to change. Docker will execute the instructions in the order that they appear in the `Dockerfile`; when one instruction is completed, the result is cached, and the build moves to the next one. If you change something in the `Dockerfile` and rebuild the image, each instruction is inspected in turn. If it has not changed, the cached layer is used and the build progresses. Conversely, if the line has changed, that build step is executed afresh, and then every subsequent instruction will have to be executed in case the changed line influences a later instruction. You should regularly rebuild the image using the `--no-cache` option to learn about broken instructions as soon as possible (cf. Rule 10 as an aside, `docker image prune --all` is a good way to remove unused images, as these tend to accrue silently in your system and take up significant disk space). Such a rebuild is also a good occasion to revisit the order of instructions, e.g., if you appended an instruction at the end to save time while iteratively developing the `Dockerfile`, and the formatting. You can add a version tag to the image before the rebuild to make sure to keep a working environment at hand. A recommended ordering based on these considerations is as follows, and you can use comments to visually separate these sections in your file (cf. Listing 1):

1. System libraries
2. Language-specific libraries or modules
    a. from repositories (i.e., binaries)
    b. from source (e.g., GitHub)
3. Installation of your own software and scripts (if not mounted)
4. Copying data and configuration files (if not mounted)
5. Labels
6. Entrypoint and default command.

Rule 10: Regularly use and rebuild containers

Using containers for research workflows requires not only technical understanding but also an awareness of risks that can be managed effectively by following a number of **good habits**, discussed in this section. While there is no firm rule, if you use a container daily, it is good practice to rebuild that container every 1 or 2 weeks; this helps identify breaking changes early and prevents multiple issues compounding on each other. At the time of publication of research results, it is good practice to save a copy of the image in a public data repository so that readers of the publication can access the resources that produced the published results.

First, it is a good habit to use your container every time you work on a project and not just as a final step during publication. If the container is the only platform you use, you can be highly confident that you have properly documented the computing environment [60]. You should prioritise this usage over others, e.g., noninteractive execution of a full workflow, because it gives you personally the highest value and does not limit your use or others' use of your data and code at all (see Rule 8).

Second, for reproducibility, we can treat containers as transient and disposable, and even intentionally rebuild an image at regular intervals. Ideally, containers that we built years ago should rebuild seamlessly, but this is not necessarily the case, especially with rapidly changing technology relevant to machine learning and data science. Habitually deleting a container and performing a cache-less rebuild of the image (a) increases security due to updating underlying software; (b) helps to reveal issues requiring manual intervention, e.g., changes to code or configuration that are not documented in the `Dockerfile` but perhaps should be; and (c) allows you to more incrementally debug issues. This habit can be supported by using continuous deployment or CI strategies.

In case you need a setup or configuration for the first 2 habits, it is good practice to provide a Makefile alongside your Dockerfile, which can capture the specific commands. Furthermore, when you rebuild the image, you can take a fresh look at the Dockerfile and improve it over time, because it will be hard to apply all rules at once. Various linting tools, either on the command line [61] or as a web service [62], are available and can be integrated into your workflow.

Third, you can export the image to file and deposit it in a public data repository, where it not only becomes citable but also provides a snapshot of the actual environment you used at a specific point in time. You should include instructions for how to import and run the workflow based on the image archive, and add your own image tags using semantic versioning (see Rule 2) for clarity. Depositing the image next to other project files, i.e., data, code, and the used `Dockerfile`, in a public repository makes them likely to be preserved, but it is highly unlikely that over time you will be able to recreate it precisely from the accompanying `Dockerfile`. Publishing the image and the contained metadata therein (e.g., the Docker version used) may even allow future science historians to emulate the Docker environment. Sharing the actual image via a registry and a version-controlled `Dockerfile` together allows you to freely experiment and continue developing your workflow and keep the image up to date, e.g., updating versions of pinned dependencies (see Rule 5) and regular image building (see above).

Finally, for a sanity check and to foster even higher trust in the stability and documentation of your project, you can ask a colleague or community member to be your code copilot (see https://twitter.com/Code_Copilot) to interact with your workflow container on a machine of their own. You can do this shortly before submitting your reproducible workflow for peer review, so you are well positioned for the future of scholarly communication and open science, where these may be standard practices required for publication [21,63–65].

Examples

To demonstrate the 10 rules, we maintain a collection of annotated example `Dockerfiles` in the `examples` directory of this article's GitHub repository. The `Dockerfiles` were mostly discovered in public repositories and updated to adhere better to the rules; see https://github.com/nuest/ten-simple-rules-dockerfiles/tree/master/examples, archived at https://doi.org/10.5281/zenodo.3878582.

Conclusions

In this article we have provided guidance for using `Dockerfiles` to create containers for use and communication in smaller-scale data science research. Reproducibility in research is an endeavour of incremental improvement and best efforts, not about achieving the perfect solution; such a solution may be not achievable for many researchers with limited resources, and its definition may change over time. Even if imperfect, the effort to create and document scientific workflows provides incredibly useful and valuable transparency for a project. We encourage researchers to follow these steps taken by their peers to use `Dockerfiles` to practice reproducible research, and we encourage them to change the way they communicate towards "preproducibility" [66], which values openness, transparency, and honesty to find fascinating problems and advance science. So, we ask researchers, with their best efforts and with their current knowledge, to strive to write readable `Dockerfiles` for functional containers that are realistic about what might break and what is unlikely to break. In a similar vein, we accept that researchers will freely break these rules if another approach makes more sense for their use case. Also, we ask that researchers not overwhelm themselves by trying to follow all the rules right away, but that they set up an iterative process to increase their computing environment's manageability over time. Most importantly, we ask researchers to share and exchange their `Dockerfiles` freely and to collaborate in their communities to spread the knowledge about containers as a tool for research and scholarly collaboration and communication.

Acknowledgments

We thank Dav Clark who provided feedback on the preprint [67].

References

1. Marwick B. How computers broke science—and what we can do to fix it [Internet]. The Conversation. 2015. https://theconversation.com/how-computers-broke-science-and-what-we-can-do-to-fix-it-49938
2. Donoho DL. An invitation to reproducible computational research. Biostatistics. 2010; 11:385–388. https://doi.org/10.1093/biostatistics/kxq028 PMID: 20538873
3. Wilson G, Aruliah DA, Brown CT, Hong NPC, Davis M, Guy RT, et al. Best practices for scientific computing. PLoS Biol. 2014; 12:e1001745. https://doi.org/10.1371/journal.pbio.1001745 PMID: 24415924
4. Wilson G, Bryan J, Cranston K, Kitzes J, Nederbragt L, Teal TK. Good enough practices in scientific computing. PLoS Comput Biol. 2017; 13:e1005510. https://doi.org/10.1371/journal.pcbi.1005510 PMID: 28640806
5. Rule A, Birmingham A, Zuniga C, Altintas I, Huang S-C, Knight R, et al. Ten simple rules for writing and sharing computational analyses in Jupyter Notebooks. PLoS Comput Biol 2019; 15:e1007007. https://doi.org/10.1371/journal.pcbi.1007007 PMID: 31344036
6. Sandve GK, Nekrutenko A, Taylor J, Hovig E. Ten simple rules for reproducible computational research. PLoS Comput Biol. 2013; 9:e1003285. https://doi.org/10.1371/journal.pcbi.1003285 PMID: 24204232
7. Nüst D. Author Carpentry: Docker for reproducible research [Internet]. 2017. https://nuest.github.io/docker-reproducible-research/
8. Chapman P. Reproducible data science environments with Docker Phil Chapman's Blog [Internet]. 2018. https://chapmandu2.github.io/post/2018/05/26/reproducible-data-science-environments-with-docker/
9. rOpenSci Labs. R Docker tutorial [Internet]. 2015. https://ropenscilabs.github.io/r-docker-tutorial/
10. Udemy, Zhbanko V. Docker Containers for Data Science and Reproducible Research [Internet]. Udemy. 2019. https://www.udemy.com/course/docker-containers-data-science-reproducible-research/
11. Psomopoulos FE. Lesson "Docker and Reproducibility" in Workshop "Reproducible analysis and Research Transparency" [Internet]. Reproducible analysis and Research Transparency. 2017. https://reproducible-analysis-workshop.readthedocs.io/en/latest/8.Intro-Docker.html
12. Gruening B, Sallou O, Moreno P, Leprevost F da Veiga, Ménager H, Søndergaard D, et al. Recommendations for the packaging and containerizing of bioinformatics software. F1000Research. 2019; 7:742. https://doi.org/10.12688/f1000research.15140.2 PMID: 31543945
13. Docker Inc. Best practices for writing Dockerfiles [Internet]. Docker Documentation. 2020. https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
14. Vass T. Intro Guide to Dockerfile Best Practices [Internet]. Docker Blog. 2019. https://www.docker.com/blog/intro-guide-to-dockerfile-best-practices/
15. Kurtzer GM, Sochat V, Bauer MW. Singularity: Scientific containers for mobility of compute. PLoS ONE. 2017; 12:e0177459. https://doi.org/10.1371/journal.pone.0177459 PMID: 28494014
16. Docker Inc. Overview of Docker Compose [Internet]. Docker Documentation. 2019. https://docs.docker.com/compose/
17. Nüst D, Konkol M, Pebesma E, Kray C, Schutzeichel M, Przibytzin H, et al. Opening the Publication Process with Executable Research Compendia. D-Lib Magazine 2017; 23. https://doi.org/10.1045/january2017-nuest
18. Cohen J, Katz DS, Barker M, Chue Hong NP, Haines R, Jay C. The Four Pillars of Research Software Engineering. IEEE Softw. 2020. https://doi.org/10.1109/MS.2020.2973362
19. Wikipedia contributors. Docker (software) [Internet]. Wikipedia. 2019. https://en.wikipedia.org/w/index.php?title=Docker_(software)&oldid=928441083
20. Boettiger C, Eddelbuettel D. An Introduction to Rocker: Docker Containers for R. The R Journal. 2017; 9:527–536. https://doi.org/10.32614/RJ-2017-065
21. Chen X, Dallmeier-Tiessen S, Dasler R, Feger S, Fokianos P, Gonzalez JB, et al. Open is not enough. Nat Phys. 2019; 15:113. https://doi.org/10.1038/s41567-018-0342-2
22. Brinckman A, Chard K, Gaffney N, Hategan M, Jones MB, Kowalik K, et al. Computing environments for reproducibility: Capturing the "Whole Tale". Futur Gener Comput Syst. 2018. https://doi.org/10.1016/j.future.2017.12.029
23. Code Ocean [Internet]. 2019. https://codeocean.com/
24. Šimko T, Heinrich L, Hirvonsalo H, Kousidis D, Rodríguez D, REANA: A System for Reusable Research Data Analyses. EPJ Web Conf. 2019; 214:06034. https://doi.org/10.1051/epjconf/201921406034
25. Project Jupyter, Bussonnier M, Forde J, Freeman J, Granger B, Head T, et al. Binder 2.0—Reproducible, interactive, sharable environments for science at scale. Proceedings of the 17th Python in Science Conference. 2018;113–120. 10.25080/Majora-4af1f417-011

26. Docker Inc. Dockerfile reference [Internet]. Docker Documentation. 2019. https://docs.docker.com/engine/reference/builder/

27. Wikipedia contributors. Make (software) [Internet]. Wikipedia. 2019. https://en.wikipedia.org/w/index.php?title=Make_(software)&oldid=929976465

28. Boettiger C. An Introduction to Docker for Reproducible Research. SIGOPS Oper Syst Rev. 2015; 49:71–79. https://doi.org/10.1145/2723872.2723882

29. Ben Marwick. 1989-excavation-report-Madjebebe. 2015. 10.6084/m9.figshare.1297059

30. Docker Inc. Official Images on Docker Hub [Internet]. Docker Documentation 2019. https://docs.docker.com/docker-hub/official_images/

31. Nüst D, Hinz M. Containerit: Generating Dockerfiles for reproducible research with R. J Open Source Softw. 2019; 4:1603. https://doi.org/10.21105/joss.01603

32. Stencila. Stencila/dockta [Internet]. Stencila. 2019. https://github.com/stencila/dockta

33. Cookiecutter contributors. Cookiecutter/cookiecutter [Internet]. cookiecutter. 2019. https://github.com/cookiecutter/cookiecutter

34. Marwick B. Benmarwick/rrtools [Internet]. 2019. https://github.com/benmarwick/rrtools

35. Docker Inc. Official Images on Docker Hub [Internet]. Docker Documentation 2020. https://docs.docker.com/docker-hub/official_images/

36. Halchenko YO, Hanke M. Open is Not Enough. Let's Take the Next Step: An Integrated, Community-Driven Computing Platform for Neuroscience. Front Neuroinform. 2012; 6. https://doi.org/10.3389/fninf.2012.00022 PMID: 23055966

37. Wickham H, Averick M, Bryan J, Chang W, McGowan L, François R, et al. Welcome to the tidyverse. J Open Source Softw. 2019; 4:1686. https://doi.org/10.21105/joss.01686

38. Preston-Werner T. Semantic Versioning 2.0.0 [Internet]. Semantic Versioning. 2013. https://semver.org/

39. Docker Inc. Use multi-stage builds [Internet]. Docker Documentation 2020. https://docs.docker.com/develop/develop-images/multistage-build/

40. Goodman A. Wagoodman/dive [Internet]. 2019. https://github.com/wagoodman/dive

41. Opencontainers. Opencontainers/image-spec v1.0.1—Annotations [Internet]. GitHub. 2017. https://github.com/opencontainers/image-spec/blob/v1.0.1/annotations.md

42. Docker Inc. Dockerfile reference, healthcheck [Internet]. Docker Documentation. 2020. https://docs.docker.com/engine/reference/builder/#healthcheck

43. The Python Software Foundation. Requirements Files—pip User Guide [Internet]. 2019. https://pip.pypa.io/en/stable/user_guide/#requirements-files

44. Continuum Analytics. Managing environments—conda documentation [Internet]. 2017. https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html

45. R Core Team. The DESCRIPTION file in "writing r extensions" [Internet]. 1999. https://cran.r-project.org/doc/manuals/r-release/R-exts.html#The-DESCRIPTION-file

46. Eddelbuettel D, Horner J. Littler: R at the command-line via 'r' [Internet]. 2019. https://CRAN.R-project.org/package=littler

47. npm. Creating a package.json file npm Documentation [Internet]. 2019. https://docs.npmjs.com/creating-a-package-json-file

48. The Julia Language Contributors. 10. Project.Toml and Manifest.Toml Pkg.Jl [Internet]. 2019. https://julialang.github.io/Pkg.jl/v1/toml-files/

49. Emsley I, De Roure D. A Framework for the Preservation of a Docker Container International Journal of Digital Curation. Int J Digit Curation. 2018; 12. https://doi.org/10.2218/ijdc.v12i2.509

50. Docker Inc. Use bind mounts [Internet]. Docker Documentation. 2019. https://docs.docker.com/storage/bind-mounts/

51. Verstegen JA. JudithVerstegen/PLUC_Mozambique: First release of PLUC for Mozambique [Internet]. Zenodo. 2019. https://doi.org/10.5281/zenodo.3519987

52. Sochat V. The Scientific Filesystem. GigaScience 2018; 7. https://doi.org/10.1093/gigascience/giy023 PMID: 29718213

53. Knoth C, Nüst D. Reproducibility and Practical Adoption of GEOBIA with Open-Source Software in Docker Containers. Remote Sens. 2017; 9:290. https://doi.org/10.3390/rs9030290

54. Molenaar G, Makhathini S, Girard JN, Smirnov O. Kliko—The scientific compute container format. Astronomy Comput. 2018; 25:1–9. https://doi.org/10.1016/j.ascom.2018.08.003

55. Selenium contributors. SeleniumHQ/selenium [Internet]. Selenium. 2019. https://github.com/SeleniumHQ/selenium

56. Singularity. Frequently Asked Questions Singularity [Internet]. 2019. http://singularity.lbl.gov/archive/docs/v2-2/faq#can-i-run-x11-apps-through-singularity

57. Viereck M. X11docker: Run GUI applications in Docker containers. J Open Source Softw. 2019; 4:1349. https://doi.org/10.21105/joss.01349

58. Yaremenko E. JAremko/docker-x11-bridge [Internet]. 2019. https://github.com/JAremko/docker-x11-bridge

59. Panda Y. Yuvipanda/jupyter-desktop-server [Internet]. 2019. https://github.com/yuvipanda/jupyter-desktop-server

60. Marwick B. README of 1989-excavation-report-Madjebebe. 2015. 10.6084/m9.figshare.1297059

61. A rule-based linter for dockerfiles [Internet]. 2020. https://github.com/projectatomic/dockerfile_lint

62. Dockerfile linter [Internet]. 2020. https://hadolint.github.io/hadolint/

63. Eglen S, Nüst D. CODECHECK: An open-science initiative to facilitate sharing of computer programs and results presented in scientific publications. Septentrio Conference Series 2019. 10.7557/5.4910

64. Schönbrodt F. Training students for the Open Science future. Nat Hum Behav. 2019; 3:1031–1031. https://doi.org/10.1038/s41562-019-0726-z PMID: 31602034

65. Eglen SJ, Mounce R, Gatto L, Currie AM, Nobis Y. Recent developments in scholarly publishing to improve research practices in the life sciences. Emerg Top Life Sci. 2018; 2:775–778. https://doi.org/10.1042/ETLS20180172

66. Stark PB. Before reproducibility must come preproducibility. Nature. 2018. https://doi.org/10.1038/d41586-018-05256-0 PMID: 29795524

67. Nüst D, Sochat V, Marwick B, Eglen S, Head T, Hirst T. Ten Simple Rules for Writing Dockerfiles for Reproducible Data Science [Internet]. Open Science Framework. 2020 Apr. https://doi.org/10.31219/osf.io/fsd7t

9 The Rockerverse: Packages and applications for containerisation with R

Authors & contribution Daniel Nüst (30%), Dirk Eddelbuettel, Dom Bennett, Robrecht Cannoodt, Dav Clark, Gergely Daróczi, Mark Edmondson, Colin Fay, Ellis Hughes, Lars Kjeldgaard, Sean Lopp, Ben Marwick, Heather Nolis, Jacqueline Nolis, Hong Ooi, Karthik Ram, Noam Ross, Lori Shepherd, Péter Sólymos, Tyson Lee Swetnam, Nitesh Turaga, Charlotte Van Petegem, Jason Williams, Craig Willis, Nan Xiao

Venue *The R Journal* (SNIP 2020: 3.63) 10.32614/RJ-2020-007

Date 09/2020

Repository https://github.com/nuest/rockerverse-paper/

The Rockerverse: Packages and Applications for Containerisation with R

by Daniel Nüst, Dirk Eddelbuettel, Dom Bennett, Robrecht Cannoodt, Dav Clark, Gergely Daróczi, Mark Edmondson, Colin Fay, Ellis Hughes, Lars Kjeldgaard, Sean Lopp, Ben Marwick, Heather Nolis, Jacqueline Nolis, Hong Ooi, Karthik Ram, Noam Ross, Lori Shepherd, Péter Sólymos, Tyson Lee Swetnam, Nitesh Turaga, Charlotte Van Petegem, Jason Williams, Craig Willis, Nan Xiao

Abstract The Rocker Project provides widely used Docker images for R across different application scenarios. This article surveys downstream projects that build upon the Rocker Project images and presents the current state of R packages for managing Docker images and controlling containers. These use cases cover diverse topics such as package development, reproducible research, collaborative work, cloud-based data processing, and production deployment of services. The variety of applications demonstrates the power of the Rocker Project specifically and containerisation in general. Across the diverse ways to use containers, we identified common themes: reproducible environments, scalability and efficiency, and portability across clouds. We conclude that the current growth and diversification of use cases is likely to continue its positive impact, but see the need for consolidating the Rockerverse ecosystem of packages, developing common practices for applications, and exploring alternative containerisation software.

Introduction

The R community continues to grow. This can be seen in the number of new packages on CRAN, which is still on growing exponentially (Hornik et al., 2019), but also in the numbers of conferences, open educational resources, meetups, unconferences, and companies that are adopting R, as exemplified by the useR! conference series[1], the global growth of the R and R-Ladies user groups[2], or the foundation and impact of the R Consortium[3]. These trends cement the role of R as the *lingua franca* of statistics, data visualisation, and computational research. The last few years, coinciding with the rise of R, have also seen the rise of Docker as a general tool for distributing and deploying of server applications—in fact, Docker can be called the *lingua franca* of describing computing environments and packaging software. Combining both these topics, the *Rocker Project* (`https://www.rocker-project.org/`) provides Docker images with R (see the next section for more details). The considerable uptake and continued evolution of the Rocker Project has led to numerous projects that extend or build upon Rocker images, ranging from reproducible[4] research to production deployments. As such, this article presents what we may call the *Rockerverse* of projects across all development stages: early demonstrations, working prototypes, and mature products. We also introduce related activities that connect the R language and environment with other containerisation solutions. Our main contribution is a coherent picture of the current status of using containers in, with, and for R.

The article continues with a brief introduction of containerisation basics and the Rocker Project, followed by use cases and applications, starting with the R packages specifically for interacting with Docker, next the second-level packages that use containers indirectly or only for specific features, and finally some complex use cases that leverage containers. We conclude by reflecting on the landscape of packages and applications and point out future directions of development.

Containerisation and Rocker

Docker, an application and service provided by the eponymous company, has, in just a few short years, risen to prominence for developing, testing, deploying and distributing computer software (cf. Datadog, 2018; Muñoz, 2019). While related approaches exist, such as LXC[5] or Singularity (Kurtzer et al., 2017), Docker has become synonymous with "containerisation"—the method of taking software artefacts and bundling them in such a way that use becomes standardized and portable across operating systems. In doing so, Docker had recognised and validated the importance of one very

[1]https://www.r-project.org/conferences/

[2]https://www.r-consortium.org/blog/2019/09/09/r-community-explorer-r-user-groups, https://www.r-consortium.org/blog/2019/08/12/r-community-explorer

[3]https://www.r-consortium.org/news/announcements, https://www.r-consortium.org/blog/2019/11/14/data-driven-tracking-and-discovery-of-r-consortium-activities

[4]"Reproducible" in the sense of the *Claerbout/Donoho/Peng* terminology (Barba, 2018).

[5]https://en.wikipedia.org/wiki/LXC

important thread that had been emerging, namely virtualisation. By allowing (one or possibly) multiple applications or services to run concurrently on one host machine without any fear of interference between them, Docker provides an important scalability opportunity. Beyond this though, Docker has improved this compartmentalisation by accessing the host system—generally Linux—through a much thinner and smaller shim than a full operating system emulation or virtualisation. This containerisation, also called operating-system-level virtualisation (Wikipedia contributors, 2020b), makes efficient use of operating system resources (Felter et al., 2015) and allows another order of magnitude in terms of scalability of deployment (cf. Datadog, 2018), because virtualisation may emulate a whole operating system, a container typically runs only one process. The single process together with sharing the host's kernel results in a reduced footprint and faster start times. While Docker makes use of Linux kernel features, it has become important enough that some required aspects of running Docker have been added to other operating systems so that those systems can more efficiently support Docker (Microsoft, 2019b). The success of Docker has even paved the way for industry collaboration and standardisation (OCI, 2019).

The key accomplishment of Docker as an "application" is to make a "bundled" aggregation of software, the so-called "image", available to any system equipped to run Docker, without requiring much else from the host besides the actual Docker application installation. This is a rather attractive proposition, and Docker's very easy to operate user interface has led to widespread adoption and use of Docker in a variety of domains, e.g., cloud computing infrastructure (e.g., Bernstein, 2014), data science (e.g., Boettiger, 2015), and edge computing (e.g., Alam et al., 2018). It has also proven to be a natural match for "cloud deployment" which runs, or at least appears to run, "seamlessly" without much explicit reference to the underlying machine, architecture or operating system: Containers are portable and can be deployed with very little dependencies on the host system—only the container runtime is required. These Docker images are normally built from plain text documents called `Dockerfiles`; a `Dockerfile` has a specific set of instructions to create and document a well-defined environment, i.e., install specific software and expose specific ports.

For statistical computing and analysis centred around R, the **Rocker Project** has provided a variety of Docker containers since it began in 2014 (Boettiger and Eddelbuettel, 2017). The Rocker Project provides several lines of containers spanning from building blocks with `R-release` or `R-devel`, via containers with RStudio Server and Shiny Server, to domain-specific containers such as `rocker/geospatial` (Boettiger et al., 2019). These containers form *image stacks*, building on top of each other for easier maintainability (i.e., smaller `Dockerfiles`), better composability, and to reduce build time. Also of note is a series of "versioned" containers which match the R release they contain with the *then-current* set of packages via the MRAN Snapshot views of CRAN (Microsoft, 2019a). The Rocker Project's impact and importance was acknowledged by the Chan Zuckerberg Initiative's *Essential Open Source Software for Science*, which provides funding for the project's sustainable maintenance, community growth, and targeting new hardware platforms including GPUs (Chan Zuckerberg Initiative et al., 2019).

Docker is not the only containerisation software. **Singularity** stems from the domain of high-performance computing (Kurtzer et al., 2017) and can also run Docker images. Rocker images work out of the box if the main process is R, e.g., in `rocker/r-base`, but Singularity does not succeed in running images where there is an init script, e.g., in containers that by default run RStudio Server. In the latter case, a `Singularity` file, a recipe akin to a `Dockerfile`, needs to be used to make necessary adjustments. To date, no comparable image stack to the Rocker Project's images exists on Singularity Hub. A further tool for running containers is **podman**, which can also build `Dockerfiles` and run Docker images. Proof of concepts exists for using podman to build and run Rocker containers[6], but the prevalence of Docker, especially in the broader user community beyond experts or niche systems and the vast amount of blog posts and courses for Docker currently cap specific development efforts for both Singularity and podman in the R community. This might quickly change if the usability and spread of Singularity or podman increase, or if security features such as rootless/unprivileged containers, which both these tools support out of the box, become more sought after.

Interfaces for Docker in R

Users interact with the Docker daemon typically through the Docker Command Line Interface (Docker CLI). However, moving back and forth between an R console and the command line can create friction in workflows and reduce reproducibility because of manual steps. A number of first-order R packages provide an interface to the Docker CLI, allowing for the interaction with the Docker CLI from an R console. Table 1 gives an overview of packages with client functionality, each of which provides functions for interacting with the Docker daemon. The packages focus on different aspects and support different stages of a container's life cycle. As such, the choice of which package is most useful depends

[6]See https://github.com/nuest/rodman and https://github.com/rocker-org/rocker-versioned/issues/187

Functionality	AzureContainers	babelwhale	dockermachine	dockyard	googleCloudRunner	harbor	stevedore
Generate a Dockerfile				✓			
Build an image	✓			✓	✓		
Execute a container locally or remotely	✓	✓	✓	✓	✓	✓	✓
Deploy or manage instances in the cloud	✓		✓		✓	✓	✓
Interact with an instance (e.g., file transfer)		✓	✓				✓
Manage storage of images						✓	✓
Supports Docker and Singularity		✓					
Direct access to Docker API instead of using the CLI							✓
Installing Docker software			✓				

Table 1: R packages with Docker client functionality.

on the use case at hand as well as on the user's level of expertise.

harbor (`https://github.com/wch/harbor`) is no longer actively maintained, but it should be honourably mentioned as the first R package for managing Docker images and containers. It uses the **sys** package (Ooms, 2019) to run system commands against the Docker CLI, both locally and through an SSH connection, and it has convenience functions, e.g., for listing and removing containers/images and for accessing logs. The outputs of container executions are converted to appropriate R types. The Docker CLI's basic functionality, although it evolves quickly and with little concern for avoiding breaking changes, has remained unchanged in core functions, meaning that a core function such as `harbor::docker_run(image = "hello-world")` still works despite its stopped development.

stevedore is currently the most powerful Docker client in R (FitzJohn, 2020). It interfaces with the Docker daemon over the Docker HTTP API[7] via a Unix socket on Linux or MacOS, over a named pipe on Windows, or over an HTTP/TCP connection. The package is the only one not using system calls to the `docker` CLI tool for managing images and containers. The package thereby enables connections to remote Docker instances without direct configuration of the local Docker daemon. Furthermore using the API gives access to information in a structured way, is system independent, and is likely more reliable than parsing command line output. **stevedore**'s own interface is automatically generated based on the OpenAPI specification of the Docker daemon, but it is still similar to the Docker CLI. The interface is similar to R6 objects, in that an object of class `"stevedore_object"` has a number of functions attached to it that can be called, and multiple specific versions of the Docker API can be supported thanks to the automatic generation[8].

AzureContainers is an interface to a number of container-related services in Microsoft's Azure Cloud (Ooi, 2019). While it is mainly intended for working with Azure, as a convenience feature it includes lightweight, cross-platform shells to Docker and Kubernetes (tools `kubectl` and `helm`). These can be used to create and manage arbitrary Docker images and containers, as well as Kubernetes clusters on any platform or cloud service.

googleCloudRunner is an interface with Google Cloud Platform container-related services, with tools to make it easier for R users to interact with them for common use cases (Edmondson, 2020). It includes deployment functions for creating R APIs using the Docker-based Cloud Run service. Users can create long running batch jobs calling any Docker image including Rocker via Cloud Build and schedule services using Cloud Scheduler.

babelwhale provides a unified interface to interact with Docker and Singularity containers (Cannoodt and Saelens, 2019). Users can, for example, execute a command inside a container, mount a volume, or copy a file with the same R commands for both container runtimes.

[7]https://docs.docker.com/engine/api/latest/

[8]See https://github.com/richfitz/stevedore/blob/master/development.md.

dockyard (`https://github.com/thebioengineer/dockyard`) has the goal of lowering the barrier to creating `Dockerfiles`, building Docker images, and deploying Docker containers. The package follows the increasingly used piping paradigm of the Tidyverse-style (Wickham et al., 2019) of programming for chaining R functions representing the instructions in a `Dockerfile`. An existing `Dockerfile` can be used as a template. **dockyard** also includes wrappers for common steps, such as installing an R package or copying files, and provides built-in functions for building an image and running a container, which make Docker more approachable within a single R-based user interface.

dockermachine (`https://github.com/cboettig/dockermachine`) is an R package to provide a convenient interface to Docker Machine from R. The CLI tool `docker-machine` allows users to create and manage a virtual host on local computers, local data centres, or at cloud providers. A local Docker installation can be configured to transparently forward all commands issued on the local Docker CLI to a selected (remote) virtual host. Docker Machine was especially crucial for local use in the early days of Docker, when no native support was available for Mac or Windows computers, but it remains relevant for provisioning on remote systems. The package has not received any updates for two years, but it is functional with a current version of `docker-machine` (`0.16.2`). It potentially lowers the barriers for R users to run containers on various hosts if they perceive that using the Docker Machine CLI directly as a barrier and it enables scripted workflows with remote processing.

Use cases and applications

Image stacks for communities of practice

Bioconductor (`https://bioconductor.org/`) is an open-source, open development project for the analysis and comprehension of genomic data (Gentleman et al., 2004). As of October 30th 2019, the project consists of 1823 R software packages, as well as packages containing annotation or experiment data. *Bioconductor* has a semi-annual release cycle, where each release is associated with a particular version of R, and Docker images are provided for current and past versions of *Bioconductor* for convenience and reproducibility. All images, which are described on the *Bioconductor* web site (see `https://bioconductor.org/help/docker/`), are created with `Dockerfiles` maintained on GitHub and distributed through Docker Hub[9]. *Bioconductor*'s "base" Docker images are built on top of the `rocker/rstudio` image. *Bioconductor* installs packages based on the R version in combination with the Bioconductor version and, therefore, uses Bioconductor version tagging `devel` and `RELEASE_X_Y`, e.g., `RELEASE_3_10`. Past and current combinations of R and *Bioconductor* will therefore be accessible via specific image tags.

The *Bioconductor* `Dockerfile` selects the desired R version from Rocker images, adds required system dependencies, and uses the **BiocManager** package for installing appropriate versions of *Bioconductor* packages (Morgan, 2019). A strength of this approach is that the responsibility for complex software configuration and customization is shifted from the user to the experienced *Bioconductor* core team. However, a recent audit of the *Bioconductor* image stack `Dockerfile` led to the deprecation of several community-maintained images, because the numerous specific images became too hard to understand, complex to maintain, and cumbersome to customise. As part of the simplification, a recent innovation is the `bioconductor_docker:devel` image, which emulates the *Bioconductor* environment for nightly builds as closely as possible. This image contains the environment variables and the system dependencies needed to install and check almost all *Bioconductor* software packages (1813 out of 1823). It saves users and package developers from creating this environment themselves. Furthermore, the image is configured so that `.libPaths()` has '`/usr/local/lib/R/host-site-library`' as the first location. Users mounting a location on the host file system to this location can persistently manage installed packages across Docker containers or image updates. Many R users pursue flexible workflows tailored to particular analysis needs rather than standardized workflows. The new `bioconductor_docker` image is well suited for this preference, while `bioconductor_docker:devel` provides developers with a test environment close to *Bioconductor*'s build system.

Data science is a widely discussed topic in all academic disciplines (e.g., Donoho, 2017). These discussions have shed light on the tools and craftspersonship behind the analysis of data with computational methods. The practice of data science often involves combining tools and software stacks and requires a cross-cutting skillset. This complexity and an inherent concern for openness and reproducibility in the data science community has led to Docker being used widely. The remainder of this section presents example Docker images and image stacks featuring R intended for data science.

- The *Jupyter Docker Stacks* project is a set of ready-to-run Docker images containing Jupyter applications and interactive computing tools (Jupyter, 2018). The `jupyter/r-notebook` image

[9]See https://github.com/Bioconductor/bioconductor_docker and https://hub.docker.com/u/bioconductor respectively.

includes R and "popular packages", and naturally also the IRKernel (`https://irkernel.github.io/`), an R kernel for Jupyter, so that Jupyter Notebooks can contain R code cells. R is also included in the catchall `jupyter/datascience-notebook` image[10]. For example, these images allow users to quickly start a Jupyter Notebook server locally or build their own specialised images on top of stable toolsets. R is installed using the Conda package manager[11], which can manage environments for various programming languages, pinning both the R version and the versions of R packages[12].

- *Kaggle* provides the `gcr.io/kaggle-images/rstats` image (previously `kaggle/rstats`) and corresponding `Dockerfile` for usage in their Machine Learning competitions and easy access to the associated datasets. It includes machine learning libraries such as Tensorflow and Keras (see also image `rocker/ml` in Section Common or public work environments), and it also configures the **reticulate** package (Ushey et al., 2019). The image uses a base image with *all packages from CRAN*, `gcr.io/kaggle-images/rcran`, which requires a Google Cloud Build because Docker Hub would time out[13]. The final extracted image size is over 25 GB, which calls into question whether having everything available is actually convenient.
- The *Radiant* project provides several images, e.g., `vnijs/rsm-msba-spark`, for their browser-based business analytics interface based on **Shiny** (Chang et al., 2019), and for use in education as part of an MSc course[14]. As data science often applies a multitude of tools, this image favours inclusion over selection and features Python, Postgres, JupyterLab and Visual Studio Code besides R and RStudio, bringing the image size up to 9 GB.
- *Gigantum* (`http://gigantum.com/`) is a platform for open and decentralized data science with a focus on using automation and user-friendly tools for easy sharing of reproducible computational workflows. Gigantum builds on the *Gigantum Client* (running either locally or on a remote server) for development and execution of data-focused *Projects*, which can be stored and shared via the *Gigantum Hub* or via a zipfile export. The Client is a user-friendly interface to a backend using Docker containers to package, build, and run Gigantum projects. It is configured to use a default set of Docker base images (`https://github.com/gigantum/base-images`), and users are able to define and configure their own custom images. The available images include two with R based on Ubuntu Linux and these have the `c2d4u` CRAN PPA pre-configured for installation of binary R packages[15]. The R images vary in the included authoring environment, i.e., Jupyter in `r-tidyverse` or both Jupyter & RStudio in `rstudio-server`. The independent image stack can be traced back to the Gigantum environment and its features. The R images are based on Gigantum's `python3-minimal` image, originally to keep the existing front-end configuration, but also to provide consistent Python-to-R interoperability. The `Dockerfiles` also use build args to specify bases, for example for different versions of NVIDIA CUDA for GPU processing[16], so that appropriate GPU drivers can be enabled automatically when supported. Furthermore, Gigantum's focus lies on environment management via GUI and ensuring a smooth user interaction, e.g., with reliable and easy conflict detection and resolution. For this reason, project repositories store authoritative package information in a separate file per package, allowing Git to directly detect conflicts and changes. A Dockerfile is generated from this description that inherits from the specified base image, and additional custom Docker instructions may be appended by users, though Gigantum's default base images do not currently include the `littler` tool, which is used by Rocker to install packages within `Dockerfiles`. Because of these specifics, instructions from `rocker/r-ubuntu` could *not* be readily re-used in this image stack (see Section Conclusions). Both approaches enable the `apt` package manager (Wikipedia contributors, 2020a) as an installation method, and this is exposed via the GUI-based environment management[17]. The image build and publication process is scripted with Python and JSON template configuration files, unlike Rocker images which rely on plain `Dockerfiles`. A further reason in the creation of an independent image stack were project constraints requiring a Rocker-incompatible licensing of the `Dockerfiles`, i.e., the MIT License.

[10]https://jupyter-docker-stacks.readthedocs.io/en/latest/using/selecting.html

[11]https://conda.io/

[12]See `jupyter/datascience-notebook`'s `Dockerfile` at https://github.com/jupyter/docker-stacks/blob/master/datascience-notebook/Dockerfile#L47.

[13]Originally, a stacked collection of over 20 images with automated builds on Docker Hub was used, see https://web.archive.org/web/20190606043353/http://blog.kaggle.com/2016/02/05/how-to-get-started-with-data-science-in-containers/ and https://hub.docker.com/r/kaggle/rcran/dockerfile

[14]'Dockerfile' available on GitHub: https://github.com/radiant-rstats/docker.

[15]https://docs.gigantum.com/docs/using-r

[16]See https://github.com/gigantum/base-images/blob/master/_templates/python3-minimal-template/Dockerfile for the `Dockerfile` of `python3-minimal`.

[17]See https://docs.gigantum.com/docs/environment-management

Capture and create environments

Community-maintained images provide a solid basis so users can meet their own individual requirements. Several second-order R packages attempt to streamline the process of creating Docker images and using containers for specific tasks, such as running tests or rendering reproducible reports. While authoring and managing an environment with Docker by hand is possible and feasible for experts[18], the following examples show that when environments become too cumbersome to create manually, automation is a powerful tool. In particular, the practice of *version pinning*, with system package managers for different operating systems and with packages **remotes** and **versions** or by using MRAN for R, can greatly increase the reproducibility of built images and are common approaches.

dockerfiler is an R package designed for building `Dockerfiles` straight from R (Fay, 2019). A scripted creation of a `Dockerfile` enables iteration and automation, for example for packaging applications for deployment (see Deployment and continuous delivery). Developers can retrieve system requirements and package dependencies to write a `Dockerfile`, for example, by leveraging the tools available in R to parse a `DESCRIPTION` file.

containerit (`https://github.com/o2r-project/containerit/`) attempts to take this one step further and includes these tools to automatically create a `Dockerfile` that can execute a given workflow (Nüst and Hinz, 2019). **containerit** accepts an R object of classes `"sessionInfo"` or `"session_info"` as input and provides helper functions to derive these from workflows, e.g., an R script or R Markdown document, by analysing the session state at the end of the workflow. It relies on the **sysreqs** (`https://github.com/r-hub/sysreqs/`) package and it's mapping of package system dependencies to platform-specific installation package names[19]. **containerit** uses **stevedore** to streamline the user interaction and improve the created `Dockerfiles`, e.g., by running a container for the desired base image to extract the already available R packages.

dockr is a similar package focusing on the generation of Docker images for R packages, in which the package itself and all of the R dependencies, including local non-CRAN packages, are available (Kjeldgaard, 2019a,b). **dockr** facilitates the organisation of code in the R package structure and the resulting Docker image mirrors the package versions of the current R session. Users can manually add statements for non-R dependencies to the `Dockerfile`.

liftr (Xiao, 2019) aims to solve the problem of persistent reproducible reporting in statistical computing based on the R Markdown format (Xie et al., 2018). The irreproducibility of authoring environments can become an issue for collaborative documents and large-scale platforms for processing documents. **liftr** makes the dynamic R Markdown document the main and sole workflow control file and the only file that needs to be shared between collaborators for consistent environments, e.g. demonstrated in the DockFlow project (`https://dockflow.org`). It introduces new fields to the document header, allowing users to manually declare the versioned dependencies required for rendering the document. The package then generates a `Dockerfile` from this metadata and provides a utility function to render the document inside a Docker container, i.e., `render_docker("foo.Rmd")`. An RStudio addin even allows compilation of documents with the single push of a button.

System dependencies are the domain of Docker, but for a full description of the computing environment, one must also manage the R version and the R packages. R versions are available via the versioned Rocker image stack (Boettiger and Eddelbuettel, 2017). r-online leverages these images and provides an app for helping users to detect breaking changes between different R versions and for historic exploration of R. With a standalone NodeJS app or r-online, the user can compare a piece of code run in two separate versions of R. Internally, r-online opens one or two Docker instances with the given version of R based on Rocker images, executes a given piece of code, and returns the result to the user. Regarding R package management, this can be achieved with MRAN, or with packages such as **checkpoint** (Ooi et al., 2020) and **renv** (Ushey, 2020), which can naturally be applied within images and containers. For example, **renv** helps users to manage the state of the R library in a reproducible way, further providing isolation and portability. While **renv** does not cover system dependencies, the **renv**-based environment can be transferred into a container either by restoring the environment based on the main configuration file `renv.lock` or by storing the **renv**-cache on the host and not in the container (Ushey, 2019). With both the system dependencies and R packages consciously managed in a Docker image, users can start using containers as the *only* environment for their workflows, which allows them to work independently of physical computers[20] and to assert a specific degree of confidence in the stability of a developed software (cf. `README.Rmd` in Marwick, 2017).

[18]See, e.g., this tutorial by RStudio on how to manage environments and package versions and to ensure deterministic image builds with Docker: https://environments.rstudio.com/docker.

[19]See https://sysreqs.r-hub.io/.

[20]Allowing them to be digital "nomads", cf. J. Bryan's https://github.com/jennybc/docker-why.

Development, debugging, and testing

Containers can also serve as playgrounds and provide specific or ad hoc environments for the purposes of developing R packages. These environments may have specific versions of R, of R extension packages, and of system libraries used by R extension packages, and all of the above in a specific combination.

First, such containers can greatly facilitate **fixing bugs and code evaluation**, because developers and users can readily start a container to investigate a bug report or try out a piece of software (cf. Ooms, 2017). The container can later be discarded and does not affect their regular system. Using the Rocker images with RStudio, these disposable environments lack no development comfort (cf. Section Packaging research reproducibly). Ooms (2017) describes how `docker exec` can be used to get a root shell in a container for customisation during software evaluation without writing a `Dockerfile`. Eddelbuettel and Koenker (2019) describes an example of how a Docker container was used to debug an issue with a package only occurring with a particular version of Fortran, and using tools which are not readily available on all platforms (e.g., not on macOS).

Second, the strong integration of **system libraries in core packages** in the R-spatial community makes containers essential for stable and proactive development of common classes for geospatial data modelling and analysis. For example, GDAL (GDAL/OGR contributors, 2019) is a crucial library in the geospatial domain. GDAL is a system dependency allowing R packages such as **sf**, which provides the core data model for geospatial vector data, or **rgdal**, to accommodate users to be able to read and write hundreds of different spatial raster and vector formats (Pebesma, 2018; Bivand et al., 2019). **sf** and **rgdal** have hundreds of indirect reverse imports and dependencies and, therefore, the maintainers spend a lot of effort trying not to break them. Purpose-built Docker images are used to prepare for upcoming releases of system libraries, individual bug reports, and for the lowest supported versions of system libraries[21].

Third, special-purpose images exist for identifying problems beyond the mere R code, such as **debugging R memory problems**. These images significantly reduce the barriers to following complex steps for fixing memory allocation bugs (cf. Section 4.3 in R Core Team, 1999). These problems are hard to debug and critical, because when they do occur they lead to fatal crashes. `rocker/r-devel-san` and `rocker/r-devel-ubsan-clang` are Docker images that have a particularly configured version of R to trace such problems with gcc and clang compilers, respectively (cf. **sanitizers** for examples, Eddelbuettel, 2014). `wch/r-debug` is a purpose-built Docker image with *multiple* instrumented builds of R, each with a different diagnostic utility activated.

Fourth, containers are useful for **testing** R code during development. To submit a package to CRAN, an R package must work with the development version of R, which must be compiled locally; this can be a challenge for some users. The **R-hub** project provides *"a collection of services to help R package development"*, with the package builder as the most prominent one (R-hub project, 2019). R-hub makes it easy to ensure that no errors occur, but fixing errors still often warrants a local setup, e.g., using the image `rocker/r-devel`, as is testing packages with native code, which can make the process more complex (cf. Eckert, 2018). The R-hub Docker images can also be used to debug problems locally using various combinations of Linux platforms, R versions, and compilers[22]. The images go beyond the configurations, or *flavours*, used by CRAN for checking packages[23], e.g., with CentOS-based images, but they lack a container for checking on Windows or OS X. The images greatly support package developers to provide support on operating systems with which they are not familiar. The package **dockertest** (`https://github.com/traitecoevo/dockertest/`) is a proof of concept for automatically generating `Dockerfiles` and building images specifically to run tests[24]. These images are accompanied by a special launch script so the tested source code is not stored in the image; instead, the currently checked in version from a local Git repository is cloned into the container at runtime. This approach separates the test environment, test code, and current working copy of the code. Another use case where a container can help to standardise tests across operating systems is detailed the vignettes of the package **RSelenium** (Harrison, 2019). The package recommends Docker for running the Selenium Server application needed to execute test suites on browser-based user interfaces and webpages, but it requires users to manually manage the containers.

Fifth, Docker images can be used **on continuous integration (CI) platforms** to streamline the testing of packages. Ye (2019) describes how they speed up the process of testing by running tasks on Travis CI within a container using `docker exec`, e.g., the package check or rendering of documentation. Cardozo (2018) also saved time with Travis CI by re-using the testing image as the basis for an image

[21]Cf. https://github.com/r-spatial/sf/tree/master/inst/docker, https://github.com/Nowosad/rspatial_proj6, and https://github.com/r-spatial/sf/issues/1231

[22]See https://r-hub.github.io/rhub/articles/local-debugging.html and https://blog.r-hub.io/2019/04/25/r-devel-linux-x86-64-debian-clang/

[23]https://cran.r-project.org/web/checks/check_flavors.html

[24]**dockertest** is not actively maintained, but mentioned still because of its interesting approach.

intended for publication on Docker Hub. `r-ci` is, in turn, used with GitLab CI, which itself is built on top of Docker images: the user specifies a base Docker image and control code, and the whole set of tests is run inside a container. The `r-ci` image stack combines `rocker` versioning and a series of tools specifically designed for testing in a fixed environment with a customized list of preinstalled packages. Especially for long-running tests or complex system dependencies, these approaches to separate installation of build dependencies with code testing streamline the development process. Containers can also simplify the integration of R software into larger, multi-language CI pipelines. Furthermore, with each change, even this manuscript is rendered into a PDF and deployed to a GitHub-hosted website (see `.travis.yml` and `Dockerfile` in the manuscript repository), not because of concern about time, but to control the environment used on a CI server. This gives, on the one hand, easy access after every update of the R Markdown source code and, on the other hand, a second controlled environment to make sure that the article renders successfully and correctly.

Processing

The portability of containerised environments becomes particularly useful for improving expensive processing of data or shipping complex processing pipelines. First, it is possible to **offload complex processing to a server** or clouds and also to execute processes in parallel to speed up or to serve many users. **batchtools** provides a parallel implementation of the `Map` function for various schedulers (Lang et al., 2017). For example, the package can schedule jobs with Docker Swarm. **googleComputeEngineR** has the function `gce_vm_cluster()` to create clusters of 2 or more virtual machines, running multi-CPU architectures (Edmondson, 2019). Instead of running a local R script with the local CPU and RAM restrictions, the same code can be processed on all CPU threads of the cluster of machines in the cloud, all running a Docker container with the same R environments. **googleComputeEngineR** integrates with the R parallelisation package **future** (Bengtsson, 2020a) to enable this with only a few lines of R code[25]. Google Cloud Run is a CaaS (Containers as a Service) platform. Users can launch containers using any Docker image without worrying about underlying infrastructure in a so-called serverless configuration. The service takes care of network ingress, scaling machines up and down, authentication, and authorisation—all features which are non-trivial for a developer to build and maintain on their own. This can be used to scale up R code to millions of instances if need be with little or no changes to existing code, as demonstrated by the proof of concept `cloudRunR`[26], which uses Cloud Run to create a scalable R-based API using **plumber** (Trestle Technology, LLC, 2018). Google Cloud Build and the Google Container Registry are a continuous integration service and an image registry, respectively, that offload building of images to the cloud, while serving the needs of commercial environments such as private Docker images or image stacks. As Google Cloud Build itself can run any container, the package **googleCloudRunner** demonstrates how R can be used as the control language for one-time or batch processing jobs and scheduling of jobs[27]. **drake** is a workflow manager for data science projects (Landau, 2018). It features implicit parallel computing and automated detection of the parts of the work that actually needs to be re-executed. **drake** has been demonstrated to run inside containers for high reproducibility[28]. Furthermore, **drake** workflows have been shown to use **future** package's function `makeClusterPSOCK()` for sending parts of the workflow to a Docker image for execution[29] (see package's function documentation; Bengtsson, 2020b). In the latter case, the container control code must be written by the user, and the **future** package ensures that the host and worker can connect for communicating over socket connections. **RStudio Server Pro** includes a functionality called Launcher (since version 1.2, released in 2019). It gives users the ability to spawn R sessions and background/batch jobs in a scalable way on external clusters, e.g., Kubernetes based on Docker images or Slurm clusters, and optionally, with Singularity containers. A benefit of the proprietary Launcher software is the ability for R and Python users to leverage containerisation's advantages in RStudio without writing specific deployment scripts or learning about Docker or managing clusters at all.

Second, containers are perfectly suited for **packaging and executing software pipelines** and required data. Containers allow for building complex processing pipelines that are independent of the host programming language. Due to its original use case (see Introduction), Docker has no standard mechanisms for chaining containers together; it lacks definitions and protocols for how to use environment variables, volume mounts, and/or ports that could enable the transfer of input (parameters and data) and output (results) to and from containers. Some packages, e.g., **containerit**,

[25]https://cloudyr.github.io/googleComputeEngineR/articles/massive-parallel.html

[26]https://github.com/MarkEdmondson1234/cloudRunR

[27]https://code.markedmondson.me/googleCloudRunner/articles/cloudbuild.html

[28]See for example https://github.com/joelnitta/pleurosoriopsis or https://gitlab.com/ecohealthalliance/drake-gitlab-docker-example, the latter even running in a continuous integration platform (cf. Development, debugging, and testing.

[29]https://docs.ropensci.org/drake/index.html?q=docker#with-docker

provide Docker images that can be used very similar to a CLI, but this usage is cumbersome[30]. **outsider** (`https://docs.ropensci.org/outsider/`) tackles the problem of integrating external programs into an R workflow without the need for users to directly interact with containers (Bennett et al., 2020). Installation and usage of external programs can be difficult, convoluted and even impossible if the platform is incompatible. Therefore, **outsider** uses the platform-independent Docker images to encapsulate processes in *outsider modules*. Each outsider module has a `Dockerfile` and an R package with functions for interacting with the encapsulated tool. Using only R functions, an end-user can install a module with the **outsider** package and then call module code to seamlessly integrate a tool into their own R-based workflow. The **outsider** package and module manage the containers and handle the transmission of arguments and the transfer of files to and from a container. These functionalities also allow a user to launch module code on a remote machine via SSH, expanding the potential computational scale. Outsider modules can be hosted code-sharing services, e.g., on GitHub, and **outsider** contains discovery functions for them.

Deployment and continuous delivery

The cloud is the natural environment for containers, and, therefore, containers are the go-to mechanism for deploying R server applications. More and more continuous integration (CI) and continuous delivery (CD) services also use containers, opening up new options for use. The controlled nature of containers, i.e., the possibility to abstract internal software environment from a minimal dependency outside of the container is crucial, for example to match test or build environments with production environments or transfer runnable entities to as-a-service infrastructures.

First, different packages use containers for the **deployment of R and Shiny apps**. **Shiny** is a popular package for creating interactive online dashboards with R, and it enables users with very diverse backgrounds to create stable and user-friendly web applications (Chang et al., 2019). *ShinyProxy* (`https://www.shinyproxy.io/`) is an open-source tool to deploy Shiny apps in an enterprise context, where it features single sign-on, but it can also be used in scientific use cases (e.g., Savini et al., 2019; Glouzon et al., 2017). ShinyProxy uses Docker containers to isolate user sessions and to achieve scalability for multi-user scenarios with multiple apps. ShinyProxy itself is written in Java to accommodate corporate requirements and may itself run in a container for stability and availability. The tool is built on ContainerProxy (`https://www.containerproxy.io/`), which provides similar features for executing long-running R jobs or interactive R sessions. The started containers can run on a regular Docker host but also in clusters. Continuous integration and deployment (CI/CD) for Shiny applications using Shinyproxy can be achieved, e.g., via GitLab pipelines or with a combination of GitHub and Docker Hub. A pipeline can include building and checking R packages and Shiny apps. After the code has passed the checks, Docker images are built and pushed to the container registry. The pipeline finishes with triggering a webhook on the server, where the deployment script is executed. The script can update configurations or pull the new Docker images. There is a ShinyProxy 1-Click App in the DigitalOcean marketplace that is set up with these webhooks. The documentation explains how to set up HTTPS with ShinyProxy and webhooks.

Another example is the package **golem**, which makes heavy use of **dockerfiler** when it comes to creating the `Dockerfile` for building and deploying production-grade Shiny applications (Guyader et al., 2019). **googleComputeEngineR** enables quick deployments of key R services, such as RStudio and Shiny, onto cloud virtual machines (VMs) with Google Cloud Compute Engine (Edmondson, 2019). The package utilises `Dockerfiles` to move the labour of setting up those services from the user to a premade Docker image, which is configured and run in the cloud VM. For example, by specifying the template `template="rstudio"` in functions `gce_vm_template()` and `gce_vm()` an up-to-date RStudio Server image is launched for development work, whereas specifying `template="rstudio-gpu"` will launch an RStudio Server image with a GPU attached, etc.

Second, containers can be used to create **platform installation packages** in a DevOps setting. The OpenCPU system provides an HTTP API for data analysis based on R. Ooms (2017) describes how various platform-specific installation files for OpenCPU are created using Docker Hub. The automated builds install the software stack from the source code on different operating systems; afterwards a script file downloads the images and extracts the OpenCPU binaries.

Third, containers can greatly facilitate the **deployment to existing infrastructures**. *Kubernetes* (`https://kubernetes.io/`) is a container-orchestration system for managing container-based application deployment and scaling. A *cluster* of containers, orchestrated as a single deployment, e.g., with Kubernetes, can mitigate limitations on request volumes or a container occupied with a computationally intensive task. A cluster features load-balancing, autoscaling of containers across numerous servers (in the cloud or on premise), and restarting failed ones. Many organisations already use a Kubernetes cluster for other applications, or a managed cluster can be acquired from service providers.

[30]https://o2r.info/containerit/articles/container.html

Docker containers are used within Kubernetes clusters to hold native code, for which Kubernetes creates a framework around network connections and scaling of resources up and down. Kubernetes can thereby host R applications, big parallel tasks, or scheduled batch jobs in a scalable way, and the deployment can even be triggered by changes to code repositories (i.e., CD, see Edmondson, 2018). The package **googleKubernetesR** (`https://github.com/RhysJackson/googleKubernetesR`) is a proof of concept for wrapping the Google Kubernetes Engine API, Google's hosted Kubernetes solution, in an easy-to-use R package. The package **analogsea** provides a way to programmatically create and destroy cloud VMs on the Digital Ocean platform (Chamberlain et al., 2019). It also includes R wrapper functions to install Docker in such a VM, manage images, and control containers straight from R functions. These functions are translated to Docker CLI commands and transferred transparently to the respective remote machine using SSH. **AzureContainers** is an umbrella package that provides interfaces for three commercial services of Microsoft's Azure Cloud, namely Container Instances for running individual containers, Container Registry for private image distribution, and Kubernetes Service for orchestrated deployments. While a package like **plumber** provides the infrastructure for turning an R workflow into a web service, for production purposes it is usually necessary to take into account scalability, reliability and ease of management. AzureContainers provides an R-based interface to these features and, thereby, simplifies complex infrastructure management to a number of R function calls, given an Azure account with sufficient credit[31]. Heroku is another cloud platform as a service provider, and it supports container-based applications. `heroku-docker-r` (`https://github.com/virtualstaticvoid/heroku-docker-r`) is an independent project providing a template for deploying R applications based on Heroku's image stack, including multiple examples for interfacing R with other programming languages. Yet the approach requires manual management of the computing environment.

Independent integrations of R for different cloud providers lead to repeated efforts and code fragmentation. To mitigate these problems and to avoid vendor lock-in motivated the OpenFaaS project. OpenFaas facilitates the deployment of functions and microservices to Kubernetes or Docker Swarm. It is language-agnostic and provides auto-scaling, metrics, and an API gateway. Reduced boilerplate code is achieved via templates. Templates for R[32] are provided based on Rocker's Debian and R-hub's r-minimal Alpine images. The templates use multi-stage Docker builds to combine R base images with the OpenFaaS 'watchdog', a tiny Golang web server. The watchdog marshals an HTTP request and invokes the actual application. The R session uses **plumber** or similar packages for the API endpoint with packages and data preloaded, thus minimizing response times.

The prevalence of Docker in industry naturally leads to the use of R in containers, as companies already manage platforms in Docker containers. These products often entail a large amount of open-source software in combination with proprietary layers adding the relevant commercialisation features. One such example is RStudio's data science platform RStudio Team. It allows teams of data scientists and their respective IT/DevOps groups to develop and deploy code in R and Python around the RStudio Open-Source Server inside of Docker images, without requiring users to learn new tools or directly interact with containers. The best practices for running RStudio with Docker containers as well as Docker images for RStudio's commercial products are publicly available.

Using R to power enterprise software in production environments

R has been historically viewed as a tool for analysis and scientific research, but not for creating software that corporations can rely on for production services. However, thanks to advancements in R running as a web service, along with the ability to deploy R in Docker containers, modern enterprises are now capable of having real-time machine learning powered by R. A number of packages and projects have enabled R to respond to client requests over TCP/IP and local socket servers, such as **Rserve** (Urbanek, 2019), **svSocket** (Grosjean, 2019), rApache and more recently **plumber** (`https://www.rplumber.io/`) and **RestRserve** (`http://restrserve.org`), which even processes incoming requests in parallel with forked processes using **Rserve**. The latter two also provide documentation for deployment with Docker or ready-to-use images with automated builds[33]. These software allow other (remote) processes and programming languages to interact with R and to expose R-based function in a service architecture with HTTP APIs. APIs based on these packages can be deployed with scalability and high availability using containers. This pattern of deploying code matches those used by software engineering services created in more established languages in the enterprise domain, such as Java or Python, and R can be used alongside those languages as a first-class member of a software engineering technical stack.

[31]See *"Deploying a prediction service with Plumber"* vignette for details: https://cran.r-project.org/web/packages/AzureContainers/vignettes/vig01_plumber_deploy.html.

[32]See OpenFaaS R templates at https://github.com/analythium/openfaas-rstats-templates.

[33]See https://www.rplumber.io/docs/hosting.html#docker, https://hub.docker.com/r/trestletech/plumber/ and https://hub.docker.com/r/rexyai/restrserve/.

CARD.com implemented a web application for the optimisation of the acquisition flow and the real-time analysis of debit card transactions. The software used **Rserve** and rApache and was deployed in Docker containers. The R session behind **Rserve** acted as a read-only in-memory database, which was extremely fast and scalable, for the many concurrent rApache processes responding to the live-scoring requests of various divisions of the company. Similarly deodorised R scripts were responsible for the ETL processes and even the client-facing email, text message and push notification alerts sent in real-time based on card transactions. The related Docker images were made available at `https://github.com/cardcorp/card-rocker`. The images extended `rocker/r-base` and additionally entailed an SSH client and a workaround for being able to mount SSH keys from the host, Pandoc, the Amazon Web Services (AWS) SDK, and Java, which is required by the AWS SDK. The AWS SDK allowed for running R consumers reading from real-time data processing streams of AWS Kinesis [34]. The applications were deployed on Amazon Elastic Container Service (ECS). The main takeaways from using R in Docker were not only that pinning the R package versions via MRAN is important, but also that moving away from Debian testing to a distribution with long-term support can be necessary. For the use case at hand, this switch allowed for more control over upstream updates and for minimising the risk of breaking the automated builds of the Docker images and production jobs.

The AI @ T-Mobile team created a set of machine learning models for natural language processing to help customer care agents manage text-based messages from customers (T-Mobile et al., 2018). For example, one model identifies whether a message is from a customer (see **Shiny**-based demo further described by Nolis and Werdell, 2019), and others tell which customers are likely to make a repeat purchase. If a data scientist creates a such a model and exposes it through a **plumber** API, then someone else on the marketing team can write software that sends different emails depending on that real-time prediction. The models are convolutional neural networks that use the **keras** package (Allaire and Chollet, 2019) and run in a Rocker container. The corresponding `Dockerfiles` are published on GitHub. Since the models power tools for agents and customers, they need to have extremely high uptime and reliability. The AI @ T-Mobile team found that the models performed well, and today these models power real-time services that are called over a million times a day.

Common or public work environments

The fact that Docker images are portable and well defined make them useful when more than one person needs access to the same computing environment. This is even more useful when some of the users do not have the expertise to create such an environment themselves, and when these environments can be run in public or using shared infrastructure. For example, *RCloud* (`https://rcloud.social`) is a cloud-based platform for data analysis, visualisation and collaboration using R. It provides a `rocker/drd` base image for easy evaluation of the platform[35].

The **Binder** project, maintained by the team behind Jupyter, makes it possible for users to **create and share computing environments** with others (Jupyter et al., 2018). A *BinderHub* allows anyone with access to a web browser and an internet connection to launch a temporary instance of these custom environments and execute any workflows contained within. From a reproducibility standpoint, Binder makes it exceedingly easy to compile a paper, visualize data, and run small examples from papers or tutorials without the need for any local installation. To set up Binder for a project, a user typically starts at an instance of a BinderHub and passes the location of a repository with a workspace, e.g., a hosted Git repository, or a data repository like Zenodo. Binder's core internal tool is `repo2docker`. It deterministically builds a Docker image by parsing the contents of a repository, e.g., project dependency configurations or simple configuration files[36]. In the most powerful case, `repo2docker` builds a given `Dockerfile`. While this approach works well for most run-of-the-mill Python projects, it is not so seamless for R projects. This is partly because `repo2docker` does not support arbitrary base images due to the complex auto-generation of the `Dockerfile` instructions.

Two approaches make using Binder easier for R users. First, **holepunch** (`https://github.com/karthik/holepunch`) is an R package that was designed to make sharing work environments accessible to novice R users based on Binder. For any R projects that use the Tidyverse suite (Wickham et al., 2019), the time and resources required to build all dependencies from source can often time out before completion, making it frustrating for the average R user. **holepunch** removes some of these limitations by leveraging Rocker images that contain the Tidyverse along with special Jupyter dependencies, and only installs additional packages from CRAN and Bioconductor that are not already part of these images. It short circuits the configuration file parsing in `repo2docker` and starts with the Binder/Tidyverse base images, which eliminates a large part of the build time and, in most cases, results in a Binder instance launching within a minute. **holepunch** also creates a `DESCRIPTION` file

[34] See useR!2017 talk "Stream processing with R in AWS".
[35] https://github.com/att/rcloud/tree/master/docker
[36] See supported file types at https://repo2docker.readthedocs.io/en/latest/config_files.html. For R, the

for essential metadata and dependency specification, and thereby turns any project into a research compendium (see Packaging research reproducibly). The `Dockerfile` included with the project can also be used to launch an RStudio Server instance locally, i.e., independent of Binder, which is especially useful when more or special computational resources can be provided there. The local image usage reduces the number of separately managed environments and, thereby, reduces work and increases portability and reproducibility.

Second, the **Whole Tale** project (`https://wholetale.org`) combines the strengths of the Rocker Project's curated Docker images with `repo2docker`. Whole Tale is a National Science Foundation (NSF) funded project developing a scalable, open-source, multi-user platform for reproducible research (Brinckman et al., 2019; Chard et al., 2019b). A central goal of the platform is to enable researchers to easily create and publish executable research objects[37] associated with published research (Chard et al., 2019a). Using Whole Tale, researchers can create and publish Rocker-based reproducible research objects to a growing number of repositories including DataONE member nodes, Zenodo and soon Dataverse. Additionally, Whole Tale supports automatic data citation and is working on capabilities for image preservation and provenance capture to improve the transparency of published computational research artefacts (Mecum et al., 2018; McPhillips et al., 2019). For R users, Whole Tale extends the Jupyter Project's `repo2docker` tool to simplify the customisation of R-based environments for researchers with limited experience with either Docker or Git. Multiple options have been discussed to allow users to change the Ubuntu LTS (long-term support, currently *Bionic Beaver*) base image, `buildpack-deps:bionic`, used in `repo2docker`. Whole Tale implemented a custom `RockerBuildPack`[38]. The build pack combines a `rocker/geospatial` image with `repo2docker`'s composability[39]. This works because both Rocker images and the `repo2docker` base image use distributions with APT (Wikipedia contributors, 2020a) so that the instructions created by the latter work because of the compatible shell and package manager.

In **high-performance computing**, one use for containers is to run workflows on shared local hardware where teams manage their own high-performance servers. This can follow one of several design patterns: Users may deploy containers to hardware as a work environment for a specific project, containers may provide per-user persistent environments, or a single container can act as a common multi-user environment for a server. In all cases, though, the containerised approach provides several advantages: First, users may use the same image and thus work environment on desktop and laptop computers. The first to patterns provide modularity, while the last approach is most similar to a simple shared server. Second, software updates can be achieved by updating and redeploying the container rather than by tracking local installs on each server. Third, the containerised environment can be quickly deployed to other hardware, cloud or local, if more resources are necessary or in case of server destruction or failure. In any of these cases, users need a method to interact with the containers, be it an IDE exposed over an HTTP port or command-line access via tools such as SSH. A suitable method must be added to the container recipes. The Rocker Project provides containers pre-installed with the RStudio IDE. In cases where users store nontrivial amounts of data for their projects, the data needs to persist beyond the life of the container. This may be in shared disks, attached network volumes, or in separate storage where it is uploaded between sessions. In the case of shared disks or network-attached volumes, care must be taken to match user permissions, and of course backups are still necessary.

CyVerse is an open-source, NSF-funded cyberinfrastructure platform for the life sciences providing easy access to computing and storage resources (Merchant et al., 2016). CyVerse has a browser-based 'data science workbench' called the Discovery Environment (DE). The DE uses a combination of HTCondor and Kubernetes for orchestrating container-based analysis and integrates with external HPC, i.e., NSF-XSEDE, through TAPIS (TACC-API's). CyVerse hosts a multi-petabyte Data Store based on iRODS with shared access by its users. The DE runs Docker containers on demand, with users able to integrate bespoke containers from DockerHub or other registries (Devisetty et al., 2016). Rocker image integration in the DE is designed to provide researchers with scalable, compute-intensive, R analysis capabilities for large and complex datasets (e.g., genomics/multi-omics, GWAS, phenotypic data, geospatial data, etc.). These capabilities give users flexibility similar to Binder, but allow containers to be run on larger computational resources (RAM, CPU, Disk, GPU), and for longer periods of time (days to weeks). The Rocker Project's RStudio and Shiny are integrated into the DE by deriving new images from Rocker images[40]. These new images include a reverse proxy using `nginx` to handle

[37]In Whole Tale a *tale* is a research object that contains metadata, data (by copy or reference), code, narrative, documentation, provenance, and information about the computational environment to support computational reproducibility.

[38]See https://github.com/whole-tale/repo2docker_wholetale.

[39]Composability refers to the ability to combine multiple package managers and their configuration files, such as R, 'pip', and 'conda'; see Section Common or public work environments for details.

[40]See https://github.com/cyverse-vice/ for `Dockerfiles` and configuration scripts; images are auto-built on DockerHub at https://hub.docker.com/u/cyversevice.

communication with CyVerse's authentication system (RStudio Support, 2020); CyVerse also allows owners to invite other registered users to securely access the same instance. The CyVerse Rocker images further include tools for connecting to its Data Store, such as the CLI utility `icommands` for iRODS. CyVerse accounts are free (with some limitations for non-US users), and the CyVerse Learning Center provides community members with information about the platform, including training and education opportunities.

Using **GPUs** (graphical processing units) as specialised hardware from containerised common work environments is also possible and useful (Haydel et al., 2015). GPUs are increasingly popular for compute-intensive machine learning (ML) tasks, e.g., deep artificial neural networks (Schmidhuber, 2015). Although in this case containers are not completely portable between hardware environments, but the software stack for ML with GPUs is so complex to set up that a ready-to-use container is helpful. Containers running GPU software require drivers and libraries specific to GPU models and versions, and containers require a specialized runtime to connect to the underlying GPU hardware. For NVIDIA GPUs, the NVIDIA Container Toolkit includes a specialized runtime plugin for Docker and a set of base images with appropriate drivers and libraries. The Rocker Project has a repository with (beta) images based on these that include GPU-enabled versions of machine-learning R packages, e.g., `rocker/ml` and `rocker/tensorflow-gpu`.

Teaching

Two use cases demonstrate the practical usefulness and advantages of containerisation in the context of teaching. On the one hand a special case of shared computing environments (see Section 2.4.7), and on the other hand leveraging sandboxing and controlled environments for auto-grading.

Prepared environments for teaching are especially helpful for (a) introductory courses, where students often struggle with the first step of installation and configuration (Çetinkaya Rundel and Rundel, 2018), and (b) courses that require access to a relatively complex setup of software tools, e.g., database systems. Çetinkaya Rundel and Rundel (2018) describe how a Docker-based deployment of RStudio (i) avoided problems with troubleshooting individual students' computers and greatly increased engagement through very quickly showing tangible outcomes, e.g., a visualisation, and (ii) reduced demand on teaching and IT staff. Each student received access to a personal RStudio instance running in a container after authentication with the university login, which gives the benefits of sandboxing and the possibility of limiting resources. Çetinkaya Rundel and Rundel (2018) found that for the courses at hand, actual usage of the UI is intermittent so a single cloud-based VM with four cores and 28 GB RAM sufficed for over 100 containers. An example for mitigating *complex setups* is teaching databases. R is very useful tool for interfacing with databases, because almost every open-source and proprietary database system has an R package that allows users to connect and interact with it. This flexibility is even broadened by **DBI** (R Special Interest Group on Databases (R-SIG-DB) et al., 2019), which allows for creating a common API for interfacing these databases, or the **dbplyr** package (Wickham and Ruiz, 2019), which runs **dplyr** (Wickham et al., 2020) code straight against the database as queries. But learning and teaching these tools comes with the cost of deploying or having access to an environment with the software and drivers installed. For people teaching R, it can become a barrier if they need to install local versions of database drivers or connect to remote instances which might or might not be made available by IT services. Giving access to a sandbox for the most common environments for teaching databases is the idea behind `r-db`, a Docker image that contains everything needed to connect to a database from R. Notably, with `r-db`, users do not have to install complex drivers or configure their machine in a specific way. The `rocker/tidyverse` base image ensures that users can also readily use packages for analysis, display, and reporting.

The idea of a common environment and partitioning allows for using **containers in teaching for secure execution and automated testing** of submissions by students. First, Dodona is a web platform developed at Ghent University that is used to teach students basic programming skills, and it uses Docker containers to test submissions by students. This means that both the code testing the students' submissions and the submission itself are executed in a predictable environment, avoiding compatibility issues between the wide variety of configurations used by students. The containerisation is also used to shield the Dodona servers from bad or even malicious code: memory, time and I/O limits are used to make sure students cannot overload the system. The web application managing the containers communicates with them by sending configuration information as a JSON document over standard input. Every Dodona Docker image shares a `main.sh` file that passes through this information to the actual testing framework, while setting up some error handling. The testing process in the Docker containers sends back the test results by writing a JSON document to its standard output channel. In June 2019, R support was added to Dodona using an image derived from the `rocker/r-base` image that sets up the `runner` user and `main.sh` file expected by Dodona[41]. It also

[41]https://github.com/dodona-edu/docker-images/blob/master/dodona-r.dockerfile

installs the packages required for the testing framework and the exercises so that this does not have to happen every time a student's submission is evaluated. The actual testing of R exercises is done using a custom framework loosely based on **testthat** (Wickham, 2011). During the development of the testing framework, it was found that the **testthat** framework did not provide enough information to its reporter system to send back all the fields required by Dodona to render its feedback. Right now, multiple statistics courses are developing exercises to automate the feedback for their lab classes.

Second, PrairieLearn is another example of a Docker-based teaching and testing platform. PrairieLearn is being developed at the University of Illinois at Urbana-Champaign (Zilles et al., 2018) and has been in extensive use across several faculties along with initial use on some other campuses. It uses Docker containers as key components, both internally for its operations (programmed mainly in Python as well as in Javascript), as well as for two reference containers providing, respectively, Python and R auto-graders. A key design decision made by PrairieLearn permits *external* grading containers to be supplied and accessed via a well-defined interface of invoking, essentially, a single script, `run.sh`. This script relies on a well-defined file layout containing JSON-based configurations, support files, exam questions, supplementary data, and student submissions. It returns per-question evaluations as JSON result files, which PrarieLearn evaluates, aggregates and records in a database. The Data Science Programming Methods course (Eddelbuettel, 2019) uses this via the custom `rocker-pl` container (Barbehenn and Eddelbuettel, 2019).[42] The `rocker-pl` image extends `rocker/r-base` with the **plr** R package (Eddelbuettel and Barbehenn, 2019b) for integration into PrarieLearn testing and question evaluation, along with the actual R packages used in instruction and testing for the course in question. As automated grading of submitted student answers is close to the well-understood problem of unit testing, the **tinytest** package (van der Loo, 2019) is used for both its core features for testing as well as clean extensibility. The package **ttdo** (Eddelbuettel and Barbehenn, 2019a) utilizes the extensibility of **tinytest** to display context-sensitive colourized differences between incorrect answers and reference answers using the **diffobj** package (Gaslam, 2019). Additionally, **ttdo** addresses the issue of insufficient information collection that Dodona faced by allowing for the collection of arbitrary, test specific attributes for additional logging and feedback. The setup, described in more detail by Eddelbuettel and Barbehenn (2020), is an excellent illustration of both the versatility and flexibility offered by Docker-based approaches in teaching and testing.

Packaging research reproducibly

Containers provide a high degree of isolation that is often desirable when attempting to capture a specific computational environment so that others can reproduce and extend a research result. Many computationally intensive research projects depend on specific versions of original and third-party software packages in diverse languages, joined together to form a pipeline through which data flows. New releases of even just a single piece of software in this pipeline can break the entire workflow, making it difficult to find the error and difficult for others to reuse existing pipelines. These breakages can make the original the results irreproducible and, and the chance of a substantial disruption like this is high in a multi-year research project where key pieces of third-party software may have several major updates over the duration of the project. The classical "paper" article is insufficient to adequately communicate the knowledge behind such research projects (cf. Donoho, 2010; Marwick, 2015).

Gentleman and Lang (2007) coined the term **Research Compendium** for a dynamic document together with supporting data and code. They used the R package system (R Core Team, 1999) for the functional prototype all the way to structuring, validating, and distributing research compendia. This concept has been taken up and extended[43], not in the least by applying containerisation and other methods for managing computing environments—see Section Capture and create environments. Containers give the researcher an isolated environment to assemble these research pipelines with specific versions of software to minimize problems with breaking changes and make workflows easier to share (cf. Boettiger, 2015; Marwick et al., 2018). Research workflows in containers are safe from contamination from other activities that occur on the researcher's computer, for example the installation of the newest version of packages for teaching demonstrations or specific versions for evaluation of others' works. Given the users in this scenario, i.e., often academics with limited formal software development training, templates and assistance with containers around research compendia is essential. In many fields, we see that a typical unit of research for a container is a research report or journal article, where the container holds the compendium, or self-contained set of data (or connections to data elsewhere) and code files needed to fully reproduce the article (Marwick et al., 2018). The package **rrtools** (`https://github.com/benmarwick/rrtools`) provides a template and convenience functions to apply good practices for research compendia, including a starter `Dockerfile`. Images of compendium containers can be hosted on services such as Docker Hub for convenient sharing

[42]The reference R container was unavailable at the time, and also relies on a heavier CentOS-based build so that a lighter alternative was established.

[43]See full literature list at https://research-compendium.science/.

among collaborators and others. Similarly, packages such as **containerit** and **dockerfiler** can be used to manage the `Dockerfile` to be archived with a compendium on a data repository (e.g. Zenodo, Dataverse, Figshare, OSF). A typical compendium's `Dockerfile` will pull a rocker image fixed to a specific version of R, and install R packages from the MRAN repository to ensure the package versions are tied to a specific date, rather than the most recent version. A more extreme case is the *dynverse* project (Saelens et al.), which packages over 50 computational methods with different environments (R, Python, C++, etc.) in Docker images, which can be executed from R. *dynverse* uses a CI platform (see Development, debugging, and testing) to build Rocker-derived images, test them, and, if the tests succeed, publish them on Docker Hub.

Future researchers can download the compendium from the repository and run the included `Dockerfile` to build a new image that recreates the computational environment used to produce the original research results. If building the image fails, the human-readable instructions in a `Dockerfile` are the starting point for rebuilding the environment. When combined with CI (see Development, debugging, and testing), a research compendium set-up can enable *continuous analysis* with easier verification of reproducibility and audits trails (Beaulieu-Jones and Greene, 2017).

Further safeguarding practices are currently under development or not part of common practice yet, such as preserving images (Emsley and De Roure, 2018), storing both images and `Dockerfiles` (cf. Nüst et al., 2017), or pinning system libraries beyond the tagged base images, which may be seen as stable or dynamic depending on the applied time scale (see discussion on `debian:testing` base image in Boettiger and Eddelbuettel, 2017). A recommendation of the recent National Academies' report on *Reproducibility and Replicability in Science* is that journals *"consider ways to ensure computational reproducibility for publications that make claims based on computations"* (Committee on Reproducibility and Replicability in Science, 2019). In fields such as political science and economics, journals are increasingly adopting policies that require authors to publish the code and data required to reproduce computational findings reported in published manuscripts, subject to independent verification (Jacoby et al., 2017; Vilhuber, 2019; Alvarez et al., 2018; Christian et al., 2018; Eubank, 2016; King, 1995). Problems with the computational environment, installation and availability of software dependencies are common. R is gaining popularity in these communities, such as for creating a research compendium. In a sample of 105 replication packages published by the *American Journal of Political Science* (AJPS), over 65% use R. The NSF-funded Whole Tale project, which was mentioned above, uses the Rocker Project community images with the goal of improving the reproducibility of published research artefacts and simplifying the publication and verification process for both authors and reviewers by reducing errors and time spent specifying the environment.

Conclusions

This article is a snapshot of the R corner in a universe of applications built with a many-faced piece of software, Docker. `Dockerfiles` and Docker images are the go-to methods for collaboration between roles in an organisation, such as developers and IT operators, and between participants in the communication of knowledge, such as researchers or students. Docker has become synonymous with applying the concept of containerisation to solve challenges of reproducible environments, e.g., in research and in development & production, and of scalable deployments because it can easily move processing between machines, e.g., locally, a cloud provider's VM, another cloud provider's Container-as-a-Service. Reproducible environments, scalability & efficiency, and portability across infrastructures are the common themes behind R packages, use cases, and applications in this work.

The projects presented above show the growing number of users, developers, and real-world applications in the community and the resulting innovations. But the applications also point to the challenges of keeping up with a continuously evolving landscape. Some use cases have considerable overlap, which can be expected as a common language and understanding of good practices is still taking shape. Also, the ease with which one can create complex software systems with Docker to serve one's specific needs, such as an independent Docker image stack, leads to parallel developments. This ease-of-DIY in combination with the difficulty of reusing parts from or composing multiple `Dockerfiles` is a further reason for **fragmentation**. Instructions can be outsourced into distributable scripts and then copied into the image during build, but that makes `Dockerfiles` harder to read. Scripts added to a `Dockerfile` also add a layer of complexity and increase the risk of incomplete recipes. Despite the different image stacks presented here, the pervasiveness of Rocker images can be traced back to its maintainers and the user community valuing collaboration and shared starting points over impulses to create individual solutions. Aside from that, fragmentation may not be a bad sign but may instead be a reflection of a growing market that is able to sustain multiple related efforts. With the maturing of core building blocks, such as the Rocker suite of images, more working systems will be built, but they may simply work behind the curtains. Docker alone, as a flexible core technology, is not a feasible level of collaboration and abstraction. Instead, the use cases and applications observed in this work provide a more useful division.

Nonetheless, at least on the level of R packages some **consolidation** seems in order, e.g., to reduce the number of packages creating `Dockerfiles` from R code or controlling the Docker daemon with R code. It remains to be seen which approach to control Docker, via the Docker API as **stevedore** or via system calls as **dockyard**/**docker**/**dockr**, is more sustainable, or whether the question will be answered by the endurance of maintainers and sufficient funding. Similarly, capturing environments and their serialisation in form of a `Dockerfile` currently is happening at different levels of abstraction, and re-use of functionality seems reasonable, e.g., **liftr** could generate the environment with **containerit**, which in turn may use **dockerfiler** for low-level R objects representing a `Dockerfile` and its instructions. In this consolidation of R packages, the Rocker Project could play the role of a coordinating entity. Nonetheless, for the moment, it seems that the Rocker Project will focus on maintaining and extending its image stacks, e.g., images for GPU-based computing and artificial intelligence. Even with coding being more and more accepted as a required and achievable skill, an easier access, for example by exposing containerisation benefits via simple user interfaces in the users' IDE, could be an important next step, since currently containerisation happens more in the background for UI-based development (e.g., a `rocker/rstudio` image in the cloud). Furthermore, the maturing of the Rockerverse packages for managing containers may lead to them being adopted in situations where manual coding is currently required, e.g. in the case of **RSelenium** or **drake** (see Sections Development, debugging, and testing and Processing respectively). In some cases, e.g., for **analogsea**, the interaction with the Docker daemon may remain too specific to re-use first-order packages to control Docker.

New features which make complex workflows accessible and reproducible and the variety in packages connected with containerisation, even when they have overlapping features, are a signal and support for a growing user base. This growth is possibly the most important goal for the foreseeable future in the *Rockerverse*, and, just like the Rocker images have matured over years of use and millions of runs, the new ideas and prototypes will have to prove themselves. It should be noted that the dominant position is that Docker is a blessing and a curse for these goals. It might be wise to start experimenting with non-Docker containerisation tools now, e.g., R packages interfacing with other container engines, such as podman/buildah, or an R package for creating `Singularity` files. Such efforts might help to avoid lock-in and to design sustainable workflows based on concepts of *containerisation*, not on their implementation in Docker. If adoption of containerisation and R continue to grow, the missing pieces for a success predominantly lie in (a) coordination and documentation of activities to reduce repeated work in favour of open collaboration, (b) the sharing of lessons learned from use cases to build common knowledge and language, and (c) a sustainable continuation and funding for development, community support, and education. A first concrete effort to work towards these missing pieces should be sustaining the structure and captured status quo from this work in the form of a *CRAN Task View on containerisation*.

Author contributions

The ordering of authors following DN and DE is alphabetical. DN conceived the article idea, initialised the formation of the writing team, wrote sections not mentioned below, and revised all sections. DE wrote the introduction and the section about containerisation and the Rocker Project, and reviewed all sections. DB wrote the section on **outsider**. GD contributed the CARD.com use case. RC contributed to the section on interfaces for Docker in R (*dynverse* and `dynwrap`). DC contributed content on Gigantum. ME contributed to the section on processing and deployment to cloud services. CF wrote paragraphs about **r-online**, **dockerfiler**, **r-ci** and **r-db**. EH contributed content on **dockyard**. LK contributed content on **dockr**. SL contributed content on RStudio's usage of Docker. BM wrote the section on research compendia and made the project Binder-ready. HN & JN co-wrote the section on the T-Mobile use case. KR wrote the section about **holepunch**. NR wrote paragraphs about shared work environments and GPUs. LS & NT wrote the section on Bioconductor. PS wrote the paragraphs about CI/CD pipelines with Shinyproxy 1-Click app and OpenFaaS templates. TS & JW wrote the section on CyVerse. CvP wrote the section on the usage of Docker containers in Dodona. CW wrote the sections on Whole Tale and contributed content about publication reproducibility audits. NX contributed content on **liftr**. All authors approved the final version. This articles was collaboratively written at https://github.com/nuest/rockerverse-paper/. The contributors page and discussion issues provide details on the respective contributions.

Acknowledgements

DN is supported by the project Opening Reproducible Research (o2r) funded by the German Research Foundation (DFG) under project number PE 1632/17-1. The funders had no role in data collection and analysis, decision to publish, or preparation of the manuscript. KR was supported in part by a grant from The Leona M. and Harry B. Helmsley Charitable Trust, award number 2016PG-BRI004. LS and NT are supported by US NIH / NHGRI awards U41HG00405 and U24HG010263. CW is supported by the Whole Tale project (`https://wholetale.org`) funded by the US National Science Foundation (NSF) under award OAC-1541450. NR is supported in part by the Chan-Zuckerberg Initiative Essential Open Source Software for Science program. We would like to thank Celeste R. Brennecka from the Scientific Editing Service of the University of Münster for her editorial support.

Bibliography

M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen. Orchestration of Microservices for IoT Using Docker and Edge Computing. *IEEE Communications Magazine*, 56(9):118–123, Sept. 2018. ISSN 0163-6804, 1558-1896. doi: 10.1109/MCOM.2018.1701233. [p438]

J. Allaire and F. Chollet. *keras: R Interface to 'Keras'*, 2019. URL `https://CRAN.R-project.org/package=keras`. R package version 2.2.5.0. [p447]

R. M. Alvarez, E. M. Key, and L. Núñez. Research replication: Practical considerations. *PS: Political Science & Politics*, 51(2):422–426, Apr 2018. ISSN 1049-0965, 1537-5935. doi: 10.1017/S1049096517002566. [p451]

L. A. Barba. Terminologies for Reproducible Research. *arXiv:1802.03311 [cs]*, Feb. 2018. URL `http://arxiv.org/abs/1802.03311`. arXiv: 1802.03311. [p437]

A. Barbehenn and D. Eddelbuettel. *rocker-pl: Docker image for grading R in PrairieLearn*, 2019. URL `https://github.com/stat430dspm/rocker-pl`. Docker container to support STAT 430 'Data Science Programming Methods', Department of Statistics, University of Illinois at Urbana-Champaign. [p450]

B. K. Beaulieu-Jones and C. S. Greene. Reproducibility of computational workflows is automated using continuous analysis. *Nature Biotechnology*, advance online publication, Mar. 2017. ISSN 1087-0156. doi: 10.1038/nbt.3780. [p451]

H. Bengtsson. *future: Unified Parallel and Distributed Processing in R for Everyone*, 2020a. URL `https://CRAN.R-project.org/package=future`. R package version 1.16.0. [p444]

H. Bengtsson. *future: Unified Parallel and Distributed Processing in R for Everyone*, 2020b. URL `https://CRAN.R-project.org/package=future`. R package version 1.16.0. [p444]

D. Bennett, H. Hettling, D. Silvestro, R. Vos, and A. Antonelli. outsider: Install and run programs, outside of r, inside of r (under review). *Journal of Open Source Software*, 5(45):2038, 2020. doi: 10.21105/joss.02038. [p445]

D. Bernstein. Containers and cloud: From LXC to docker to kubernetes. *IEEE Cloud Computing*, 1(3): 81–84, Sept. 2014. doi: 10.1109/mcc.2014.51. [p438]

R. Bivand, T. Keitt, and B. Rowlingson. *rgdal: Bindings for the 'Geospatial' Data Abstraction Library*, 2019. URL `https://CRAN.R-project.org/package=rgdal`. R package version 1.4-8. [p443]

C. Boettiger. An introduction to Docker for reproducible research, with examples from the R environment. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, Jan. 2015. ISSN 01635980. doi: 10.1145/2723872.2723882. [p438, 450]

C. Boettiger and D. Eddelbuettel. An Introduction to Rocker: Docker Containers for R. *The R Journal*, 9 (2):527–536, 2017. doi: 10.32614/RJ-2017-065. [p438, 442, 451]

C. Boettiger, R. Lovelace, M. Howe, and J. Lamb. rocker-org/geospatial, Dec. 2019. [p438]

A. Brinckman, K. Chard, N. Gaffney, M. Hategan, M. B. Jones, K. Kowalik, S. Kulasekaran, B. Ludäscher, B. D. Mecum, J. Nabrzyski, et al. Computing environments for reproducibility: Capturing the "Whole Tale". *Future Generation Computer Systems*, 94:854–867, 2019. doi: 10.1016/j.future.2017.12.029. [p448]

R. Cannoodt and W. Saelens. *babelwhale: Talking to 'Docker' and 'Singularity' Containers*, 2019. URL https://CRAN.R-project.org/package=babelwhale. R package version 1.0.1. [p439]

L. Cardozo. Faster Docker builds in Travis CI for R packages, 2018. URL https://lecardozo.github.io/2018/03/01/automated-docker-build.html. [p443]

S. Chamberlain, H. Wickham, and W. Chang. *analogsea: Interface to 'Digital Ocean'*, 2019. URL https://CRAN.R-project.org/package=analogsea. R package version 0.7.2. [p446]

Chan Zuckerberg Initiative, C. Boettiger, N. Ross, and D. Eddelbuettel. Maintaining Rocker: Sustainability for Containerized Reproducible Analyses, 2019. URL https://chanzuckerberg.com/eoss/proposals/maintaining-rocker-sustainability-for-containerized-reproducible-analyses/. [p438]

W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *shiny: Web Application Framework for R*, 2019. URL https://CRAN.R-project.org/package=shiny. R package version 1.4.0. [p441, 445]

K. Chard, N. Gaffney, M. B. Jones, K. Kowalik, B. Ludäscher, T. McPhillips, J. Nabrzyski, V. Stodden, I. Taylor, T. Thelen, M. J. Turk, and C. Willis. Application of BagIt-Serialized Research Object Bundles for Packaging and Re-execution of Computational Analyses. 2019a. doi: 10.5281/zenodo.3381754. To appear in 2019 IEEE 15th International Conference on e-Science (e-Science). [p448]

K. Chard, N. Gaffney, M. B. Jones, K. Kowalik, B. Ludäscher, J. Nabrzyski, V. Stodden, I. Taylor, M. J. Turk, and C. Willis. Implementing computational reproducibility in the whole tale environment. In *Proceedings of the 2nd International Workshop on Practical Reproducible Evaluation of Computer Systems*, P-RECS '19, pages 17–22, 2019b. doi: 10.1145/3322790.3330594. [p448]

T.-M. Christian, W. G. Jacoby, S. Lafferty-Hess, and T. Carsey. Operationalizing the replication standard. *International Journal of Digital Curation*, 13(1), 2018. doi: 10.2218/ijdc.v13i1.555. [p451]

Committee on Reproducibility and Replicability in Science. *Reproducibility and Replicability in Science*. National Academies Press, 2019. ISBN 978-0-309-48616-3. doi: 10.17226/25303. [p451]

Datadog. 8 surprising facts about real Docker adoption, June 2018. URL https://www.datadoghq.com/docker-adoption/. [p437, 438]

U. K. Devisetty, K. Kennedy, P. Sarando, N. Merchant, and E. Lyons. Bringing your tools to CyVerse Discovery Environment using Docker. *F1000Research*, 5:1442, Dec. 2016. ISSN 2046-1402. doi: 10.12688/f1000research.8935.3. [p448]

D. Donoho. 50 Years of Data Science. *Journal of Computational and Graphical Statistics*, 26(4):745–766, Oct. 2017. ISSN 1061-8600. doi: 10.1080/10618600.2017.1384734. [p440]

D. L. Donoho. An invitation to reproducible computational research. *Biostatistics*, 11(3):385–388, July 2010. ISSN 1465-4644. doi: 10.1093/biostatistics/kxq028. [p450]

A. Eckert. Building and testing R packages with latest R-Devel, Feb. 2018. URL https://alexandereckert.com/post/testing-r-packages-with-latest-r-devel/. [p443]

D. Eddelbuettel. *sanitizers: C/C++ source code to trigger Address and Undefined Behaviour Sanitizers*, 2014. URL https://CRAN.R-project.org/package=sanitizers. R package version 0.1.0. [p443]

D. Eddelbuettel. *STAT430: Data Science Programming Methods*, 2019. URL https://stat430.com. Fourth and fifth year topics course, Department of Statistics, University of Illinois at Urbana-Champaign. [p450]

D. Eddelbuettel and A. Barbehenn. *ttdo: Extend 'tinytest' with 'diffobj'*, 2019a. URL https://CRAN.R-project.org/package=ttdo. R package version 0.0.4. [p450]

D. Eddelbuettel and A. Barbehenn. *plr: Utility Functions for 'PrairieLearn' and R*, 2019b. URL https://github.com/stat430dspm/plr. R package supporting Docker for STAT 430 'Data Science Programming Methods', Department of Statistics, University of Illinois at Urbana-Champaign. [p450]

D. Eddelbuettel and A. Barbehenn. An R Autograder for PrairieLearn, 2020. URL http://arxiv.org/abs/2003.06500. [p450]

D. Eddelbuettel and R. Koenker. Debugging with Docker and Rocker – A Concrete Example helping on macOS, Aug. 2019. URL http://dirk.eddelbuettel.com/blog/2019/08/05/. [p443]

M. Edmondson. R on Kubernetes - serverless Shiny, R APIs and scheduled scripts, May 2018. URL https://code.markedmondson.me/r-on-kubernetes-serverless-shiny-r-apis-and-scheduled-scripts/. [p446]

M. Edmondson. *googleComputeEngineR: R Interface with Google Compute Engine*, 2019. URL https://CRAN.R-project.org/package=googleComputeEngineR. R package version 0.3.0. [p444, 445]

M. Edmondson. *googleCloudRunner: R Scripts in the Google Cloud via Cloud Run, Cloud Build and Cloud Scheduler*, 2020. URL https://CRAN.R-project.org/package=googleCloudRunner. R package version 0.1.1. [p439]

I. Emsley and D. De Roure. A Framework for the Preservation of a Docker Container | International Journal of Digital Curation. *International Journal of Digital Curation*, 12(2), Apr. 2018. doi: 10.2218/ijdc.v12i2.509. [p451]

N. Eubank. Lessons from a decade of replications at the Quarterly Journal of Political Science. *PS: Political Science & Politics*, 49(2):273–276, Apr 2016. ISSN 1049-0965, 1537-5935. doi: 10.1017/S1049096516000196. [p451]

C. Fay. *dockerfiler: Easy Dockerfile Creation from R*, 2019. URL https://CRAN.R-project.org/package=dockerfiler. R package version 0.1.3. [p442]

W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and Linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 171–172, Mar. 2015. doi: 10.1109/ISPASS.2015.7095802. [p438]

R. FitzJohn. *stevedore: Docker Client*, 2020. URL https://CRAN.R-project.org/package=stevedore. R package version 0.9.3. [p439]

B. Gaslam. *diffobj: Diffs for R Objects*, 2019. URL https://CRAN.R-project.org/package=diffobj. R package version 0.2.3. [p450]

GDAL/OGR contributors. *GDAL/OGR Geospatial Data Abstraction software Library*. Open Source Geospatial Foundation, 2019. URL https://gdal.org. [p443]

R. Gentleman and D. T. Lang. Statistical Analyses and Reproducible Research. *Journal of Computational and Graphical Statistics*, 16(1):1–23, Mar. 2007. ISSN 1061-8600. doi: 10.1198/106186007X178663. [p450]

R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. Yang, and J. Zhang. Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology*, 5(10):R80, Sept. 2004. ISSN 1474-760X. doi: 10.1186/gb-2004-5-10-r80. [p440]

J.-P. S. Glouzon, J.-P. Perreault, and S. Wang. Structurexplor: a platform for the exploration of structural features of RNA secondary structures. *Bioinformatics*, 33(19):3117–3120, Oct. 2017. ISSN 1367-4803. doi: 10.1093/bioinformatics/btx323. [p445]

P. Grosjean. *SciViews-R: A GUI API for R*. UMONS, MONS, Belgium, 2019. URL http://www.sciviews.org/SciViews-R. [p446]

V. Guyader, C. Fay, S. Rochette, and C. Girard. *golem: A Framework for Robust Shiny Applications*, 2019. URL https://CRAN.R-project.org/package=golem. R package version 0.1. [p445]

J. Harrison. *RSelenium: R Bindings for 'Selenium WebDriver'*, 2019. URL https://CRAN.R-project.org/package=RSelenium. R package version 1.7.5. [p443]

N. Haydel, G. Madey, S. Gesing, A. Dakkak, S. G. de Gonzalo, I. Taylor, and W.-m. W. Hwu. Enhancing the Usability and Utilization of Accelerated Architectures via Docker. In *Proceedings of the 8th International Conference on Utility and Cloud Computing*, UCC '15, pages 361–367. IEEE Press, 2015. ISBN 978-0-7695-5697-0. URL http://dl.acm.org/citation.cfm?id=3233397.3233456. [p449]

K. Hornik, U. Ligges, and A. Zeileis. Changes on cran. *The R Journal*, 11(1):438–441, June 2019. URL http://journal.r-project.org/archive/2019-1/cran.pdf. [p437]

W. G. Jacoby, S. Lafferty-Hess, and T.-M. Christian. Should journals be responsible for reproducibility? *Inside Higher Ed*, Jul 2017. URL https://www.insidehighered.com/blogs/rethinking-research/should-journals-be-responsible-reproducibility. [p451]

P. Jupyter. Jupyter Docker Stacks — docker-stacks latest documentation, 2018. URL https://jupyter-docker-stacks.readthedocs.io/en/latest/. [p440]

P. Jupyter, M. Bussonnier, J. Forde, J. Freeman, B. Granger, T. Head, C. Holdgraf, K. Kelley, G. Nalvarte, A. Osheroff, M. Pacer, Y. Panda, F. Perez, B. Ragan-Kelley, and C. Willing. Binder 2.0 - Reproducible, interactive, sharable environments for science at scale. *Proceedings of the 17th Python in Science Conference*, pages 113–120, 2018. doi: 10.25080/Majora-4af1f417-011. [p447]

G. King. Replication, replication. *PS: Political Science & Politics*, 28(3):444–452, Sep 1995. doi: 10.2307/420301. [p451]

L. Kjeldgaard. *dockr: Creation of Lightweight Docker Images for Your Packages*, 2019a. URL https://CRAN.R-project.org/package=dockr. R package version 0.8.6. [p442]

L. Kjeldgaard. 'dockr': easy containerization for R - pRopaganda by smaakagen, Dec. 2019b. URL http://smaakage85.netlify.com/2019/12/21/dockr-easy-containerization-for-r/. [p442]

G. M. Kurtzer, V. Sochat, and M. W. Bauer. Singularity: Scientific containers for mobility of compute. *PLOS ONE*, 12(5):e0177459, May 2017. ISSN 1932-6203. doi: 10.1371/journal.pone.0177459. [p437, 438]

W. M. Landau. The drake r package: a pipeline toolkit for reproducibility and high-performance computing. *Journal of Open Source Software*, 3(21), 2018. [p444]

M. Lang, B. Bischl, and D. Surmann. batchtools: Tools for r to work on batch systems. *The Journal of Open Source Software*, 2(10):135, 2 2017. ISSN 2475-9066. doi: 10.21105/joss.00135. [p444]

B. Marwick. How computers broke science – and what we can do to fix it, Nov. 2015. URL http://theconversation.com/how-computers-broke-science-and-what-we-can-do-to-fix-it-49938. [p450]

B. Marwick. Research compendium for the 1989 excavations at Madjedbebe rockshelter, NT, Australia, July 2017. [p442]

B. Marwick, C. Boettiger, and L. Mullen. Packaging Data Analytical Work Reproducibly Using R (and Friends). *The American Statistician*, 72(1):80–88, Jan. 2018. ISSN 0003-1305. doi: 10.1080/00031305.2017.1375986. [p450]

T. McPhillips, C. Willis, M. Gryk, S. Nunez-Corrales, and B. Ludäscher. Reproducibility by Other Means: Transparent Research Objects. 2019. doi: 10.5281/zenodo.3382423. To appear in 2019 IEEE 15th International Conference on e-Science (e-Science). [p448]

B. Mecum, M. B. Jones, D. Vieglais, and C. Willis. Preserving reproducibility: Provenance and executable containers in dataone data packages. In *2018 IEEE 14th International Conference on e-Science (e-Science)*, pages 45–49. IEEE, 2018. doi: 10.1109/eScience.2018.00019. [p448]

N. Merchant, E. Lyons, S. Goff, M. Vaughn, D. Ware, D. Micklos, and P. Antin. The iPlant Collaborative: Cyberinfrastructure for Enabling Data to Discovery for the Life Sciences. *PLOS Biology*, 14(1): e1002342, Jan. 2016. ISSN 1545-7885. doi: 10.1371/journal.pbio.1002342. [p448]

Microsoft. CRAN Time Machine - MRAN, 2019a. URL https://mran.microsoft.com/timemachine. [p438]

Microsoft. Linux Containers on Windows, Sept. 2019b. URL https://docs.microsoft.com/en-us/virtualization/windowscontainers/deploy-containers/linux-containers. [p438]

M. Morgan. *BiocManager: Access the Bioconductor Project Package Repository*, 2019. URL https://CRAN.R-project.org/package=BiocManager. R package version 1.30.10. [p440]

S. Muñoz. The history of Docker's climb in the container management market, June 2019. URL https://searchservervirtualization.techtarget.com/feature/The-history-of-Dockers-climb-in-the-container-management-market. [p437]

J. Nolis and J. Werdell. Small data, big value, Dec. 2019. URL https://medium.com/tmobile-tech/small-data-big-value-f783ceca4fdb. [p447]

D. Nüst and M. Hinz. containerit: Generating Dockerfiles for reproducible research with R. *Journal of Open Source Software*, 4(40):1603, Aug. 2019. ISSN 2475-9066. doi: 10.21105/joss.01603. URL https://joss.theoj.org/papers/10.21105/joss.01603. [p442]

D. Nüst, M. Konkol, E. Pebesma, C. Kray, M. Schutzeichel, H. Przibytzin, and J. Lorenz. Opening the Publication Process with Executable Research Compendia. *D-Lib Magazine*, 23(1/2), Jan. 2017. ISSN 1082-9873. doi: 10.1045/january2017-nuest. [p451]

OCI. Open Containers Initiative - About, 2019. URL `https://www.opencontainers.org/about`. [p438]

H. Ooi. *AzureContainers: Interface to 'Container Instances', 'Docker Registry' and 'Kubernetes' in 'Azure'*, 2019. URL `https://CRAN.R-project.org/package=AzureContainers`. R package version 1.2.0. [p439]

H. Ooi, A. de Vries, and Microsoft. *checkpoint: Install Packages from Snapshots on the Checkpoint Server for Reproducibility*, 2020. URL `https://CRAN.R-project.org/package=checkpoint`. R package version 0.4.9. [p442]

J. Ooms. OpenCPU - Why Use Docker with R? A DevOps Perspective, Oct. 2017. URL `https://www.opencpu.org/posts/opencpu-with-docker/`. [p443, 445]

J. Ooms. *sys: Powerful and Reliable Tools for Running System Commands in R*, 2019. URL `https://CRAN.R-project.org/package=sys`. R package version 3.3. [p439]

E. Pebesma. Simple Features for R: Standardized Support for Spatial Vector Data. *The R Journal*, 10(1): 439–446, 2018. doi: 10.32614/RJ-2018-009. [p443]

R Core Team. 1999. URL `https://cran.r-project.org/doc/manuals/r-devel/R-exts.html`. [p443, 450]

R-hub project. R-hub Docs, 2019. URL `https://docs.r-hub.io/`. [p443]

R Special Interest Group on Databases (R-SIG-DB), H. Wickham, and K. Müller. *DBI: R Database Interface*, 2019. URL `https://CRAN.R-project.org/package=DBI`. R package version 1.1.0. [p449]

RStudio Support. Running rstudio server with a proxy, Jan. 2020. URL `https://support.rstudio.com/hc/en-us/articles/200552326-Running-RStudio-Server-with-a-Proxy`. [p449]

W. Saelens, R. Cannoodt, H. Todorov, and Y. Saeys. A comparison of single-cell trajectory inference methods. 37. ISSN 15461696. doi: 10.1038/s41587-019-0071-9. [p451]

L. Savini, L. Candeloro, S. Perticara, and A. Conte. EpiExploreR: A Shiny Web Application for the Analysis of Animal Disease Data. *Microorganisms*, 7(12):680, Dec. 2019. doi: 10.3390/microorganisms7120680. [p445]

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, Jan. 2015. ISSN 0893-6080. doi: 10.1016/j.neunet.2014.09.003. [p449]

T-Mobile, J. Nolis, and H. Nolis. Enterprise Web Services with Neural Networks Using R and TensorFlow, Nov. 2018. URL `https://opensource.t-mobile.com/blog/posts/r-tensorflow-api/`. [p447]

Trestle Technology, LLC. *plumber: An API Generator for R*, 2018. URL `https://CRAN.R-project.org/package=plumber`. R package version 0.4.6. [p444]

S. Urbanek. *Rserve: Binary R server*, 2019. URL `https://CRAN.R-project.org/package=Rserve`. R package version 1.7-3.1. [p446]

K. Ushey. Using renv with Docker, 2019. URL `https://rstudio.github.io/renv/articles/docker.html`. [p442]

K. Ushey. *renv: Project Environments*, 2020. URL `https://CRAN.R-project.org/package=renv`. R package version 0.9.3. [p442]

K. Ushey, J. Allaire, and Y. Tang. *reticulate: Interface to 'Python'*, 2019. URL `https://CRAN.R-project.org/package=reticulate`. R package version 1.14. [p441]

M. van der Loo. *tinytest: Lightweight and Feature Complete Unit Testing Framework*, 2019. URL `https://CRAN.R-project.org/package=tinytest`. R package version 1.1.0. [p450]

L. Vilhuber. Report by the AEA Data Editor. *AEA Papers and Proceedings*, 109:718–729, May 2019. ISSN 2574-0768. doi: 10.1257/pandp.109.718. [p451]

H. Wickham. testthat: Get started with testing. *The R Journal*, 3:5–10, 2011. URL `https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf`. [p450]

H. Wickham and E. Ruiz. *dbplyr: A 'dplyr' Back End for Databases*, 2019. URL https://CRAN.R-project.org/package=dbplyr. R package version 1.4.2. [p449]

H. Wickham, M. Averick, J. Bryan, W. Chang, L. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. Pedersen, E. Miller, S. Bache, K. Müller, J. Ooms, D. Robinson, D. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, and H. Yutani. Welcome to the Tidyverse. *Journal of Open Source Software*, 4(43):1686, Nov. 2019. ISSN 2475-9066. doi: 10.21105/joss.01686. [p440, 447]

H. Wickham, R. François, L. Henry, and K. Müller. *dplyr: A Grammar of Data Manipulation*, 2020. URL https://CRAN.R-project.org/package=dplyr. R package version 0.8.4. [p449]

Wikipedia contributors. APT (software), Feb. 2020a. URL https://en.wikipedia.org/w/index.php?title=APT_(software)&oldid=939802209. Page Version ID: 939802209. [p441, 448]

Wikipedia contributors. OS-level virtualization, Jan. 2020b. URL https://en.wikipedia.org/w/index.php?title=OS-level_virtualization&oldid=935110975. Page Version ID: 935110975. [p438]

N. Xiao. *liftr: Containerize R Markdown Documents for Continuous Reproducibility*, 2019. URL https://CRAN.R-project.org/package=liftr. R package version 0.9.2. [p442]

Y. Xie, J. J. Allaire, and G. Grolemund. *R Markdown: The Definitive Guide*. Chapman and Hall/CRC, 2018. [p442]

H. Ye. Docker Setup for R package Development, 2019. URL https://haoye.us/post/2019-10-10-docker-for-r-package-development/. [p443]

C. Zilles, M. West, D. Mussulman, and T. Bretl. Making testing less trying: Lessons learned from operating a computer-based testing facility. In *Proceedings of the 2018 Frontiers in Education Conference (FIE 2018)*, 2018. URL http://lagrange.mechse.illinois.edu/pubs/ZiWeMuBr2018/ZiWeMuBr2018.pdf. [p450]

M. Çetinkaya Rundel and C. Rundel. Infrastructure and Tools for Teaching Computing Throughout the Statistical Curriculum. *The American Statistician*, 72(1):58–65, Jan. 2018. ISSN 0003-1305. doi: 10.1080/00031305.2017.1397549. [p449]

Daniel Nüst
University of Münster
Institute for Geoinformatics
Heisenbergstr. 2
48149 Münster, Germany
0000-0002-0024-5046
daniel.nuest@uni-muenster.de

Dirk Eddelbuettel
University of Illinois at Urbana-Champaign
Department of Statistics
Illini Hall, 725 S Wright St
Champaign, IL 61820, USA
0000-0001-6419-907X
dirk@eddelbuettel.com

Dom Bennett
Gothenburg Global Biodiversity Centre, Sweden
Carl Skottsbergs gata 22B
413 19 Göteborg, Sweden
0000-0003-2722-1359
dominic.john.bennett@gmail.com

Robrecht Cannoodt
Ghent University
Data Mining and Modelling for Biomedicine group
VIB Center for Inflammation Research
Technologiepark 71
9052 Ghent, Belgium

0000-0003-3641-729X
robrecht@cannoodt.dev

Dav Clark
Gigantum, Inc.
1140 3rd Street NE
Washington, D.C. 20002, USA
0000-0002-3982-4416
dav@gigantum.com

Gergely Daróczi

0000-0003-3149-8537
daroczig@rapporter.net

Mark Edmondson
IIH Nordic A/S, Google Developer Expert for Google Cloud Platform
Artillerivej 86
2300 København S, Denmark
0000-0002-8434-3881
mark@markedmondson.me

Colin Fay
ThinkR
5O rue Arthur Rimbaud
93300 Aubervilliers, France
0000-0001-7343-1846
contact@colinfay.me

Ellis Hughes
Fred Hutchinson Cancer Research Center
Vaccine and Infectious Disease
1100 Fairview Ave. N., P.O. Box 19024
Seattle, WA 98109-1024, USA
ehhughes@fredhutch.org

Lars Kjeldgaard
Danish Tax Authorities
Oestbanegade 123
2100, Koebenhavn Oe
lars_kjeldgaard@hotmail.com

Sean Lopp
RStudio, Inc
250 Northern Ave
Boston, MA 02210, USA
sean@rstudio.com

Ben Marwick
University of Washington
Department of Anthropology
Denny Hall 230, Spokane Ln
Seattle, WA 98105, USA
0000-0001-7879-4531
bmarwick@uw.edu

Heather Nolis
T-Mobile
12920 Se 38th St.
Bellevue, WA, 98006, USA
heather.wensler1@t-mobile.com

Jacqueline Nolis
Nolis, LLC
Seattle, WA, USA
0000-0001-9354-6501
`jacqueline@nolisllc.com`

Hong Ooi
Microsoft
Level 5, 4 Freshwater Place
Southbank, VIC 3006, Australia
`hongooi@microsoft.com`

Karthik Ram
Berkeley Institute for Data Science
University of California
Berkeley, CA 94720, USA
0000-0002-0233-1757
`karthik.ram@berkeley.edu`

Noam Ross
EcoHealth Alliance
460 W 34th St., Ste. 1701
New York, NY 10001, USA
0000-0002-2136-0000
`ross@ecohealthalliance.org`

Lori Shepherd
Roswell Park Comprehensive Cancer Center
Elm & Carlton Streets
Buffalo, NY, 14263, USA
0000-0002-5910-4010
`lori.shepherd@roswellpark.org`

Péter Sólymos
Analythium Solutions
#258 150 Chippewa Road
Sherwood Park, AB, T8A 6A2, Canada
0000-0001-7337-1740
`peter@analythium.io`

Tyson Lee Swetnam
University of Arizona
1657 E Helen St.
Tucson, AZ, 85721, USA
0000-0002-6639-7181
`tswetnam@arizona.edu`

Nitesh Turaga
Roswell Park Comprehensive Cancer Center
Elm & Carlton Streets
Buffalo, NY, 14263, USA
0000-0002-0224-9817
`nitesh.turaga@roswellpark.org`

Charlotte Van Petegem
Ghent University
Department WE02
Krijgslaan 281, S9
9000 Gent, Belgium
0000-0003-0779-4897
`charlotte.vanpetegem@ugent.be`

Jason Williams
Cold Spring Harbor Laboratory
1 Bungtown Rd.
Cold Spring Harbor, NY, 11724, USA
0000-0003-3049-2010
`williams@cshl.edu`

Craig Willis
University of Illinois at Urbana-Champaign
501 E. Daniel St.
Champaign, IL 61820, USA
0000-0002-6148-7196
`willis8@illinois.edu`

Nan Xiao
Seven Bridges Genomics
529 Main St, Suite 6610
Charlestown, MA 02129, USA
0000-0002-0250-5673
`me@nanx.me`

10 Practical reproducibility in geography and geosciences

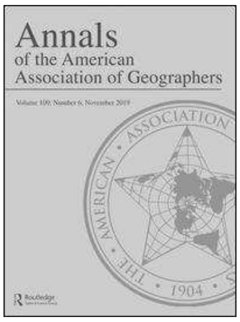
Authors & contribution Daniel Nüst (90%), Edzer Pebesma

Venue *Annals of the American Association of Geographers* (SNIP 2020: 1.93)

10.1080/24694452.2020.1806028

Date 10/2020

License Publisher copyright; AAM PDF.



Annals of the American Association of Geographers

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/raag21

Practical Reproducibility in Geography and Geosciences

Daniel Nüst & Edzer Pebesma

To cite this article: Daniel Nüst & Edzer Pebesma (2020): Practical Reproducibility in Geography and Geosciences, Annals of the American Association of Geographers, DOI: 10.1080/24694452.2020.1806028

To link to this article: https://doi.org/10.1080/24694452.2020.1806028

Published online: 13 Oct 2020.

Practical Reproducibility in Geography and Geosciences

Daniel Nüst and Edzer Pebesma

Institute for Geoinformatics, University of Münster

Reproducible research is often perceived as a technological challenge, but it is rooted in the challenge to improve scholarly communication in an age of digitization. When computers become involved and researchers want to allow other scientists to inspect, understand, evaluate, and build on their work, they need to create a research compendium that includes the code, data, computing environment, and script-based workflows used. Here, we present the state of the art for approaches to reach this degree of computational reproducibility, addressing literate programming and containerization while paying attention to working with geospatial data (digital maps, geographic information systems). We argue that all researchers working with computers should understand these technologies to control their computing environment, and we present the benefits of reproducible workflows in practice. Example research compendia illustrate the presented concepts and are the basis for challenges specific to geography and geosciences. Based on existing surveys and best practices from different scientific domains, we conclude that researchers today can overcome many barriers and achieve a very high degree of reproducibility. If the geography and geosciences communities adopt reproducibility and the underlying technologies in practice and in policies, they can transform the way researchers conduct and communicate their work toward increased transparency, understandability, openness, trust, productivity, and innovation. *Key Words: computational reproducibility, reproducible research, scholarly communication.*

Reproducible research is often perceived as primarily a technological challenge, but it is really rooted in the challenge to adjust scholarly communication to today's level of digitization and diversity of scientific outputs. Common academic challenges, such as broken metrics and pressure to publish articles over other products (see, e.g., Piwowar 2013; Nosek et al. 2015), have a negative impact on reproducibility. The state of reproducibility in geosciences and GIScience was investigated by Konkol, Kray, and Pfeiffer (2019) and Nüst, Boettiger, and Marwick (2018), respectively, and both studies show that it needs to improve. Other fields support this result; for example, Brunsdon (2016) on quantitative geography, Wainwright (2020) on an informal search in critical geography, Sui and Kedron (2020) on the conceptual challenges of reproducibility and replication in geography, and Sui and Shaw (2018) on the lack of knowledge about the state of reproducibility in human dynamics.

In this article, we present the current state of the art for practical reproducibility of research and connect it to geography and geosciences. The challenges around reproducible research manifest in the general lack of knowledge on how to work reproducibly and the small fraction of published reproducible articles. Interestingly, this is the case even though the individual and overall benefits of reproducibility (Vandewalle, Kovacevic, and Vetterli 2009; Donoho 2010; Markowetz 2015; Marwick 2015; Kray et al. 2019) and the innovative potential of working reproducibly, which include, for example, "unhelpful … non-reproducibility" (Sui and Kedron 2020), better collaboration (Singleton, Spielman, and Brunsdon 2016), and new pathways (Waters 2020), are increasingly known and common concerns are debunked (Barnes 2010). Editorial requirements and author guidelines are an effective means to encourage reproducibility, but they are not widespread enough or are still too lax (cf. Nosek et al. 2015; Singleton, Spielman, and Brunsdon 2016; Stodden, Seiler, and Ma 2018), so further incentives are needed for a change of habits and culture (Munafó et al. 2017; Nüst, Boettiger, and Marwick 2018). Because many solutions for practical reproducibility are not discipline specific, we include literature from other domains to corroborate the small body of work in "geo" fields, but we stick to examples and

Initial submission, November 2019; revised submission, March 2020; final acceptance, April 2020
Published by Taylor & Francis, LLC.

highlight particular concerns for these communities of practice. For a much more extensive and comprehensive overview of the topic, we refer the reader to the recent consensus study report *Reproducibility and Replicability in Science* (National Academies of Sciences, Engineering, and Medicine 2019).

We follow the Claerbout/Donoho/Peng terminology (Barba 2018) and distinguish reproduction from replication[1] and reproducibility to mean *computational reproducibility* (National Academies of Sciences, Engineering, and Medicine 2019). Replicability is "the ultimate standard" (Peng 2011), because it requires independent confirmation and potentially yields new findings. Yet replication poses fewer technological challenges: Hypotheses, results, and conclusions are communicated with text and are addressed by some form of peer review. A suitable methodology for independent repetition can be developed from the text. Replication demands, however, that a particular study can be replicated; that is, that data sets used can be re-collected or computations can be repeated (Peng 2011). In studies describing particular areas and time periods of the Earth, this might not be possible; for instance, satellite images or interviews can only be taken once, at a particular moment in time, by a particular instrument or person, respectively. Furthermore, large-scale computations could be prohibitively expensive to replicate (Šimko et al. 2019), specialized hardware can be singular (Santana-Perez and Pérez-Hernández 2015), and real-time data streams would have to be openly recorded constantly (cf. Brunsdon 2016).

When studies are impossible to replicate for conceptual or practical reasons, reproducibility is the only way we can ensure that a scientific claim can be evaluated, and it becomes a minimal standard (Peng 2011; Sandve et al. 2013). Open data alone do not sufficiently guarantee reproducibility despite great advancements driven by the FAIR principles and research data management (see Wilkinson et al. 2016; Higman, Bangert, and Jones 2019), but workflows and processes must be open, too (Chen et al. 2019). The dynamic nature of the development processes makes it particularly important that concerns around computational reproducibility—that is, all aspects of computers involved in research—are comprehensively considered from the start. Otherwise, science falls short of communicating results effectively, as stated in Claerbout and Karrenbach's claim: "An article about a computational result is advertising, not scholarship. The actual knowledge is the full software environment, code and data, that produced the result" (adapted from Donoho [2010], who paraphrased Claerbout and Karrenbach 1992).

So, science today is too complicated for brief articles to fully communicate knowledge (Donoho 2010; Marwick 2015), and "[…] paradoxically, these in-silico experiments are often more difficult to reproduce than traditional laboratory techniques" (Howe 2012, 36). Peng (2011) introduced a *spectrum of reproducibility*, which is useful to inclusively acknowledge limitations and identify the current state of individual pieces of work and practices. Peng (2011) further argued that researchers should not wait for a comprehensive solution and concluded, "developing a culture of reproducibility … will require time and sustained effort" (1227). As part of this effort, we present the following tools and discuss challenges for reaching a high degree of computational reproducibility, fully communicating knowledge, and making in silico experiments reproducible when using and presenting geospatial data.

Reproducible Workflows in Geography and Geosciences

Creating Reproducible Workflows

Scientists must realize how fragile the typical research workflows are today. We have grown accustomed to the experience that a computer-based analysis we conduct today still works tomorrow; yet, although this is often the case, when there are differences, they can be very hard to explain, despite their dramatic effect (as documented, e.g., in Gronenschild et al. 2012; Bhandari Neupane et al. 2019). The lack of reported failures from geography and geosciences is not reassuring, and measures to improve reproducibility have been suggested. For example, Gil et al. (2016) presented the *Geoscience Paper of the Future* based on a thorough analysis of developments and challenges, and they give useful and concrete steps for modern digital open scholarship; Singleton, Spielman, and Brunsdon (2016) described a framework for reproducible publications based on Open GIS, open data, and workflow models for an Open Geographic Information science (Open GISc) going beyond text-centric publications. Building on these ideas, we present a practical approach for reproducible workflows and extend previous work with a deliberate management of the computing environment.

A *computing environment* is the totality of hardware and software components involved in a particular workflow. The description of the computing environment must be understandable by both machines and humans: by machines so that snapshots can be taken, the environment can be moved, or infrastructure can provision required capabilities (e.g., using ontologies; Santana-Perez and Pérez-Hernández 2015); by humans so that failures can be investigated and fixed. This documentation can be crafted manually, generated with the assistance of tools (e.g., Jupyter Project et al. 2018; Nüst and Hinz 2019), or recorded as provenance; for example, using scientific workflow management systems (see National Academies of Sciences, Engineering, and Medicine [2019], for details, which are beyond the scope of this work). A well-defined computing environment increases trust in the stability of results and the chances that third parties can also execute an analysis. Hirst (2019) coined three components of a computing environment: *physical*, *logical*, and *cultural*. The Examples and Conclusions sections cover the cultural component.

The physical component is the hardware; for example, the researcher's laptop, a university's high-performance computing facility, a Global Positioning System device, sensors, or instruments. Such devices might be preserved physically and investigated if problems arise but at very high costs (e.g., regular testing and replacement of parts). These costs are probably too high for regular research, and, at this stage of reproducibility, physical components are too rarely the source of critical issues. Thus, this component must be documented in detail (e.g., product names, IDs, manufacturing batches) and, where self-built, have open construction plans. It is worth noting here that quite often software has a much longer life span than hardware, and outdated hardware can often, although much later, be emulated by software.

To capture the logical component, common software development methods, such as using a language's package manager and repository[2] and practicing version pinning in the respective configuration files, allow freezing the logical component in a specific state. Virtualization (Howe 2012) and containerization[3] (Boettiger 2015) provide adequate solutions to capture full software stacks; that is, both programs' researchers are aware of obvious and unobvious dependencies (Perkel 2019). Containers can be created from a recipe file, which provides an additional layer of transparency and safeguarding (Nüst and Hinz 2019) independent of the specific container implementation (Santana-Perez and Pérez-Hernández 2015), or even automatically in a deterministic way (Jupyter Project et al. 2018). Container preservation is actively researched (Rechert et al. 2017; Emsley and De Roure 2018). Such configuration files and recipes can be managed using a version control system for retracing errors and auditing (Ram 2013). The application of Docker,[4] Singularity,[5] or supportive automating tools (Jupyter Project et al. 2018) is a core skill for geoscientists and geographers analyzing or visualizing data with computers.

The goal of describing the computing environment is to allow others to re-create, scrutinize, or extend it. This becomes more difficult when (1) the logical component is directly linked with the physical component; for example, bespoke optimized software for a particular computing, infrastructure, such as high-performance computing; or (2) critical parts of the computations involve proprietary software.[6]

A *script-based workflow* means that a user can execute a full analysis, starting from raw data up to visualizations for publication, without any manual intervention. Ideally the main control file is a digital notebook following the literate programming paradigm (Knuth 1984), thereby integrating text, documentation, visualization, mapping (Giraud and Lambert 2017), and publication[7] in a coherent way. Jupyter (Kluyver et al. 2016; Rule et al. 2019) and R Markdown (Xie 2015) are the two most commonly used notebooks for practical reproducibility. Both support various programming languages, hybrid workflows, and operating systems. All of the workflow's parts can be openly published in the form of a *research compendium* (Gentleman and Temple Lang 2007), originally using a language's packaging mechanism and later extended and demonstrated as a powerful tool for scholarly communication.[8] A self-contained structured research compendium is "preproducible" (Stark 2018), connects the actual article with supplemental material (off-loading details; cf. Greenbaum et al. 2017), and becomes an executable research compendium (Nüst et al. 2017) if it includes both container and notebook. All parts of an (executable) research compendium must be adequately licensed to allow use and extension (cf.

Stodden 2009) and use open formats (Marwick 2015).

To summarize, authors, editors, reviewers, and publishers can achieve the highest reproducibility when they (1) familiarize themselves with common guidance for reproducible research (e.g., Sandve et al. 2013; The Turing Way Community et al. 2019), (2) consciously control computing environments, (3) use script-based workflows with notebooks, and (4) adhere to community practices for research compendia. These steps can bring researchers close to the "gold standard" end of Peng's (2011) reproducibility spectrum.

Using Reproducible Workflows

Based on a research compendium, reviewers, students, collaborators, and even the original authors years later can interact with a piece of research in a manner far beyond a classic "paper" article. Using a common format for a research compendium eases communication between authors and readers (Marwick, Boettiger, and Mullen 2018; Nüst, Boettiger, and Marwick 2018), and special infrastructures can be built to discover and interact with them (Perkel 2019). Research compendia can even underpin intelligent systems (cf. Santana-Perez and Pérez-Hernández 2015; Gil et al. 2016). There is not one special infrastructure emerging yet, nor should there be only one, because different approaches cater to different needs and communities and different actors—for example, publishers (Brunsdon 2016; Harris et al. 2017)—may provide it. For example, Code Ocean (Clyburne-Sherin, Fei, and Green 2019) is a commercial platform for researchers to conduct their work online based on Jupyter. It partners with publishers[9] to give reviewers and readers access to research compendia with a full development environment. Konkol and Kray (2019) described an enhanced examination workflow for scientific papers based on executable research compendia and used it to provide tailored interactive figures (Konkol, Kray, and Suleiman 2019). The Whole Tale (Brinckman et al. 2019) and BinderHub (Jupyter Project et al. 2018) projects build open platforms for reproducible research operated by research organizations. Such platforms are the most effective way today to leverage containerization for openly publishing practical, reproducible workflows and improving scholarly communication, without requiring additional expertise beyond creating research compendia.

Examples

The following examples illustrate the challenges, solutions, and prevailing shortcomings. They extend earlier collections of cases in geography (Brunsdon 2016), spatial data collection and analysis in ecology (Lewis, Wal, and Fifield 2018), spatial statistics[10] (Pebesma, Bivand, and Ribeiro 2015), and geosciences (Konkol, Kray, and Pfeiffer 2019).[11] A comprehensive reproducibility study in geography and geosciences is needed to substantiate these observations.

Spielman and Singleton (2015) studied neighborhoods with data from the American Community Survey and provided data and methods openly.[12] We applaud their efforts, which allowed us to partially reproduce the workflow,[13] such as setting a seed to avoid problems with nondeterministic results. This project, however, demonstrates typical shortcomings and issues for reproducibility (see also Konkol, Kray, and Pfeiffer 2019), such as lacking licenses, binary formats for data, and a data repository requiring login and acceptance of terms of use.

Marwick (2017) reported on a case study about the analysis of data from an archaeological excavation, with inherently geospatial data. In detail and suitable for a nontechnical audience, Marwick described all considerations and concrete actions for data archiving, scripting, publishing, and containerization of the computing environment.

Knoth and Nüst (2017) containerized a complex geographic object-based image analysis workflow using open source tools in a discipline where one proprietary software is ubiquitous. The work demonstrates how a combination of free tools can re-create a proprietary analysis workflow, and it shows how containerization can make it reusable by exposing configuration parameters and making the data set exchangeable.

Shannon and Walker (2018) described two case studies in housing and urban diversity for public-facing geographic research. The case studies entail *shiny*-based applications (Chang et al. 2020) with interactive plots and maps for nonexpert users to improve community engagement, which we could easily inspect and reproduce. The authors nicely use openness for transparency and provide synthetic data to handle data privacy, but the published code is sparsely documented and lacks licensing information, which hampers reuse and extension.

Verstegen (2019) published code and data for a land use change model based on PCRaster and a Python script (cf. Verstegen et al. 2012). The repository includes a container for ease of use and transparently communicates (despite lacking a notebook document) which parts of the workflow reproduce which figure and what changes were made to the code after the original article publication.

Challenges for Practical Reproducibility in Geography and Geosciences

Geography and geosciences are diverse disciplines, and their community members have equally diverse backgrounds, many of which do not include a familiarity with computational methods or software development. This diversity leads to challenges in adopting practical reproducibility in education and publishing. The focus on practical solutions in this work can inform these adaptations, which must be accompanied by changes of habits by individuals and at different organizational levels, such as research labs (cf. Nüst, Boettiger, and Marwick 2018). The large body of experience from other domains and best practices (Sandve et al. 2013; Stodden and Miguez 2014; Boettiger 2015; Eglen et al. 2017; Greenbaum et al. 2017; Marwick 2017; Eglen et al. 2018; Marwick, Boettiger, and Mullen 2018; Nüst et al. 2019; Pérignon et al. 2019; Rule et al. 2019; Schönbrodt 2019; Šimko et al. 2019) does not limit self-improvement and further training, but the amount of information might seem overwhelming. In a similar way, ongoing disruptions and innovations in scholarly publishing (cf. Gil et al. 2016; Singleton, Spielman, and Brunsdon 2016; Eglen et al. 2018; Tennant et al. 2019) pose challenges for geographers and geoscientists in their roles as authors, reviewers, and editors, especially for early career researchers and due to a complex mixture of community, commercial, and political interests.

Giraud and Lambert (2017) described the multiplicity of tools in the cartographic process as an impediment for reproducibility. They transferred Peng's spectrum into a spectrum of map reproducibility and set the equivalent of a research compendium (linked and executable code and data) at the highest level. They argued that cartography is often considered a design process and an art, but this should not be at the cost of reproducibility, for example, due to manual tweaking of visual appearances. Konkol, Kray, and Pfeiffer (2019) even found that the differences in the created maps were an effective way to assess reproducibility. Similar to the aforementioned spectra, Wilson et al. (2020) present a five-star classification for sharing geospatial research, addressing challenges in geographic information systems (GIS) software and algorithms.

Geospatial data and processing are often realized via spatial data infrastructures, such as the data, processing, and map interfaces by the Open Geospatial Consortium or OpenStreetMap. Online services pose a challenge for reproducibility, because they could change over time or disappear. A service-oriented approach, however, also promises improvement through standardization, less duplication of effort, and easier translation into different tools for cross-validation (Wilson et al. 2020). Still, the code to access geoservices and the requests sent as well as the retrieved responses must all be stored (cf. real-time data; Brunsdon 2016) to build a research compendium. When analyzing large data sets, processing is increasingly shifted to remote infrastructures closer to the data, which requires open availability not only of the application programming interface but also of the implementations (Hinz et al. 2013; Pebesma et al. 2017). "Free" platforms, such as Google Earth Engine, provide complex script-based processing to a broad audience, but the analyses are not reproducible because the computational environment cannot be captured or inspected in full (Sidhu, Pebesma, and Câmara 2018). When creating research compendia, compromises can be made as to the amount of detail they include to reduce storage size; for example, include only relevant data after preprocessing or allow referencing data in trusted data repositories or spatial data infrastructures (Nüst and Schutzeichel 2017).

Qualitative GIS was judged as nonreproducible by Preston and Wilson (2014), partially due to their mixed-methods approach. In our view, however, such an approach does not free researchers to work as reproducible as possible. Data collection and creation of visualizations can be reproducible and should be, because maps are commonly used for interaction with study participants during data collection and for communicating results. Muenchow, Schäfer, and Krüger (2019) reviewed the body of work in qualitative GIS research and identified reproducibility as having promising potential for the field.

Prevailing GIS software is based on a graphical user interface, proprietary, or both. To fix these limitations, either these tools must be updated to provide an executable workflow (i.e., recording a trace of the user interactions; cf. Brunsdon 2016) or researchers need to switch to open tools to achieve a unified toolchain (Giraud and Lambert 2017) and to avoid the risk of a digital divide but rather enable faster collaborative development (cf. Muenchow, Schäfer, and Krüger 2019). Proprietary software might in some cases be user friendly for conducting research, and Open Source alternatives require a higher computer literacy (Muenchow, Schäfer, and Krüger 2019), but such closed tools are ultimately unsuitable for science: No access to source code prohibits examination and extension of methods and can increase the potential of errors (Singleton, Spielman, and Brunsdon 2016) and restrictive non-open license agreements prohibit reproduction by others without access or even by authors at a future point in time (Eaton 2012; Lees 2012; Singleton, Spielman, and Brunsdon 2016). Most important, open software stacks much better with core tools for practical reproducibility. The pace of digitization and the trend toward openness (cf. Nosek et al. 2015) put pressure on scientists at all career stages to switch to open tools[14] and require future geoscientists and geographers to be trained as "Pi-shaped researchers" with a deep knowledge both in their domain as well as in reproducibility and computing (Marwick 2017).

Limitations of sensitive data are commonly mentioned impediments to practical reproducibility, but various solutions exist. O'Loughlin et al. (2015) discussed the balance of disclosure and source protection in the field of political geography, and they mentioned redaction as a means to check research using quantitative data and statistical data rigorously. These limitations can also be seen as a need in establishing processes and providing infrastructure for controlled access to research compendia. Pérignon et al. (2019) and Foster (2018) described the tensions between reproducibility and data privacy, and they presented a public research infrastructure for confidential government data in France and cloud-based data enclaves. Shannon and Walker (2018) suggested an analysis infrastructure that restricts access to raw data and only provides derived results. In the context of geocomputation, Brunsdon argued the advantages of "'domains of reproducibility'—that is, groups of people who are permitted to access this information adopting reproducible practices amongst themselves—so that internal scrutiny, and updating of analyses becomes easier" (Harris et al. 2017, 608).

An approach to reduce the limitations induced by big, proprietary, export-controlled, or sensitive data is providing a synthetic data set (e.g., Shannon and Walker 2018). A data set of more manageable size reduces storage space as well as workflow execution time. Made-up data prevent deanonymization and can be tailored to illustrate the method. A copy of the original data within the research compendium ensures consistency and accessibility, but synthetic data, anonymized data, or data subsets allow third parties to evaluate, understand, validate, and build on methods.

Reproducibility of computational methods is further constrained by time. The fact that all presented platforms and tools are open themselves facilitates archival and maintenance, yet the reproduction of workflows in more than ten years is an open challenge beyond geography and geosciences. The Ten Years Challenge by the journal *ReScience*[15] is an example for learning more about problems and solutions for long-term reproducibility. Because we cannot foresee what future computers will look like, a research compendium that can be reproduced today, for example, as part of a peer review (Eglen and Nüst 2019) ensures that everything needed is there and ensures a starting point for future generations of geographers, geoscientists, and science historians.

Conclusions

In this article we describe practical solutions that facilitate computational reproducibility in scholarly communication. Wilson and Burrough (1999) stated on a new geography, "It is also clear that improved understanding of landscapes comes … from the study of large quantities of data in a reproducible data-handling environment that extends from the field to the laboratory and the computer" (743). They further argued for the adoption of new methods and that "geographers will need to be comfortable in new sneakers that incorporate the [new methods]" (Wilson and Burrough 1999, 743). As the new method, we suggest replacing traditional text-centric research papers as the final product of research with executable research compendia: digital artifacts that encapsulate the data, the script-based workflow and its computing environment, and the article based on a notebook.

The emerging infrastructure for research compendia greatly reduces the needed software engineering skills, yet a lack of academic recognition for openness and reproducibility and a lack of hard, minimal requirements posed by editorial boards of scientific journals still keep scientists from adopting methods supporting practical reproducibility. Chen et al. (2019) rightly argued that new research practices must be tailored to the needs of scientific disciplines. In geography and geosciences, this discourse has just started (Pebesma, Nüst, and Bivand 2012a, 2012b; Gil et al. 2016; Nüst, Boettiger, and Marwick 2018; Kedron et al. 2019, and the articles of this Forum). These scientific communities must decide which degree of reproducibility is "good enough," but we believe that in most cases "very, very close to the original" is feasible and practical. Irrespective of whether the "reproducibility crisis" does or does not exist (cf. Fanelli 2018), the benefits of working reproducibly are by now clear. Technical, systemic, and cultural barriers are conquerable. The advantages of reproducibility for scientific progress lie in strengthened trust in results through transparency, higher productivity through openness, and more innovation through collaboration and exploration of new pathways. The scientific community should embrace the disruptions in scholarly publishing and reap the benefits and advantages by setting up new platforms and standards for scholarly communication (e.g., Munafò et al. 2017; Kray et al. 2019). The maxim of the new technology for practical reproducibility should be open source software implementing an open and self-correcting public infrastructure controlled by scientists (cf. Buck 2015; Santana-Perez and Pérez-Hernández 2015; Munafò et al. 2017).

Acknowledgments

We thank Celeste R. Brennecka from the Scientific Editing Service, University of Münster, for her editorial review. Two anonymous reviewers provided valuable comments on an earlier version of this article. We thank the organizers of the workshop on Reproducibility and Replicability in Geospatial Research at Arizona State University and of the subsequent Forum for the opportunities to contribute.

Funding

This work is supported by the project Opening Reproducible Research II (https://www.uni-muenster.de/forschungaz/project/12343) funded by the German Research Foundation (DFG) under project number PE 1632/17-1.

ORCID

Daniel Nüst http://orcid.org/0000-0002-0024-5046
Edzer Pebesma http://orcid.org/0000-0001-8049-7069

Notes

1. *Reproduction* means that the authors' materials are available for third parties to re-create identical results, whereas replication means different data and methods lead to the same findings. From a computational standpoint, *identical* is more complicated than it sounds; for example, floating point computations might result in small yet insignificant numerical differences, or image-rendering algorithms might introduce nondeterministic artifacts.
2. For example, CRAN (https://cran.r-project.org) and renv (https://cran.r-project.org/package = renv) for R, or PyPI (https://pypi.org/) and conda (https://conda.io) for Python, which even has tooling for separating full installations in virtual environments; for example, virtualenv (https://virtualenv.pypa.io).
3. For the simplicity of the argument, we use *recipe* instead of Dockerfile and *containers* as a catch-all term, whereas the experienced reader might expect a distinction between *container* and *image*.
4. Docker is the most common containerization solution today (see https://en.wikipedia.org/wiki/Docker_(software)). It is open source, and relevant parts are standardized (see https://www.opencontainers.org/).
5. Singularity is mostly used in scientific contexts and high-performance computing, see Kurtzer, Sochat, and Bauer (2017).
6. Proprietary software cannot be avoided in some areas, such as the system BIOS or device drivers.
7. The notebook might render directly into submission-ready manuscripts with R Markdown and the *rticles* package by Allaire et al. (2020), which supports a variety of journals, including the publisher of the *Annals*, Taylor & Francis, and other publishers close to the disciplines such as AGU or Copernicus Publications (EGU).
8. See https://research-compendium.science/ for a minimal definition, extensive literature, and examples.

The R (R Core Team 2019) community is at the forefront of enabling reproducibility both in the available tools and in the mindset of the user community (e.g., Pebesma, Nüst, and Bivand 2012b; Marwick 2015).
9. For example, Sage (Estop 2019), De Gruyter (Code Ocean 2018), or Nature ("Easing the Burden of Code Review" 2018).
10. All articles in this special issue on software for spatial statistics in the *Journal of Statistical Software* are in principle reproducible, but these articles by software developers are probably not representative of the whole community using the software.
11. The largest study to date, it reproduced thirty-one research articles. See the full list at https://osf.io/sfqjg/.
12. Code on GitHub: https://github.com/geoss/acs_demographic_clusters; data on openICPSR: http://doi.org/10.3886E41329V1.
13. A summary of the issues, changes, suggestions, and subsequent communication with the authors is available at https://github.com/geoss/acs_demographic_clusters/issues/2.
14. The Carpentries (https://carpentries.org/) is an excellent resource to learn data science skills outside of topical studies.
15. See https://rescience.github.io/ten-years/.

References

Allaire, J. J., Y. Xie, R Foundation, H. Wickham, *Journal of Statistical Software*, R. Vaidyanathan, Association for Computing Machinery, et al. 2020. *rticles: Article formats for R markdown*. CRAN. R package version 0.14.1. Accessed August 31, 2020. https://github.com/rstudio/rticles.

Barba, L. A. 2018. Terminologies for reproducible research. *arXiv*:1802.03311. http://arxiv.org/abs/1802.03311.

Barnes, N. 2010. Publish your computer code: It is good enough. *Nature News* 467 (7317):753. doi: 10.1038/467753a.

Bhandari Neupane, J., R. P. Neupane, Y. Luo, W. Y. Yoshida, R. Sun, and P. G. Williams. 2019. Characterization of leptazolines A–D, polar oxazolines from the cyanobacterium *Leptolyngbya* sp., reveals a glitch with the "Willoughby–Hoye" scripts for calculating NMR chemical shifts. *Organic Letters* 21 (20):8449–53. doi: 10.1021/acs.orglett.9b03216.

Boettiger, C. 2015. An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review* 49 (1):71–79. doi: 10.1145/2723872.2723882.

Brinckman, A., K. Chard, N. Gaffney, M. Hategan, M. B. Jones, K. Kowalik, S. Kulasekaran, B. Ludascher, B. D. Mecum, J. Nabrzyski, et al. 2019. Computing environments for reproducibility: Capturing the "Whole Tale." *Future Generation Computer Systems* 94:854–67. doi: 10.1016/j.future.2017.12.029.

Brunsdon, C. 2016. Quantitative methods I: Reproducible research and quantitative geography. *Progress in Human Geography* 40 (5):687–96. doi: 10.1177/0309132515599625.

Buck, S. 2015. Solving reproducibility. *Science* 348 (6242):1403. doi: 10.1126/science.aac8041.

Chang, W., J. Cheng, J. J. Allaire, Y. Xie, and J. McPherson. 2020. *shiny: Web application framework for R*. CRAN. R package version 1.4.0.2. Accessed August 31, 2020. https://CRAN.R-project.org/package=shiny.

Chen, X., S. Dallmeier-Tiessen, R. Dasler, S. Feger, P. Fokianos, J. B. Gonzalez, H. Hirvonsalo, D. Kousidis, A. Lavasa, S. Mele, et al. 2019. Open is not enough. *Nature Physics* 15 (2):113–19. doi: 10.1038/s41567-018-0342-2.

Claerbout, J., and M. Karrenbach. 1992. Electronic documents give reproducible research a new meaning. In *SEG Technical Program Expanded Abstracts 1992*, 601–4. Tulsa, OK: Society of Exploration Geophysicists. doi: 10.1190/1.1822162.

Clyburne-Sherin, A., X. Fei, and S. A. Green. 2019. Computational reproducibility via containers in psychology. *Meta-Psychology* 3. doi: 10.15626/MP.2018.892.

Code Ocean. 2018. De Gruyter partners with Code Ocean to improve research reproducibility. Accessed April 24, 2020. https://codeocean.com/press-release/de-gruyter-partners-with-code-ocean-to-improve-research-reproducibility.

Donoho, D. L. 2010. An invitation to reproducible computational research. *Biostatistics* 11 (3):385–88. doi: 10.1093/biostatistics/kxq028.

Easing the burden of code review [editorial]. 2018. *Nature Methods* 15 (9):641. doi: 10.1038/s41592-018-0137-5.

Eaton, J. W. 2012. GNU Octave and reproducible research. *Journal of Process Control* 22 (8):1433–38. doi: 10.1016/j.jprocont.2012.04.006.

Eglen, S. J., B. Marwick, Y. O. Halchenko, M. Hanke, S. Sufi, P. Gleeson, R. A. Silver, A. P. Davison, L. Lanyon, M. Abrams, et al. 2017. Toward standard practices for sharing computer code and programs in neuroscience. *Nature Neuroscience* 20 (6):770–73. doi: 10.1038/nn.4550.

Eglen, S. J., R. Mounce, L. Gatto, A. M. Currie, and Y. Nobis. 2018. Recent developments in scholarly publishing to improve research practices in the life sciences. *Emerging Topics in Life Sciences* 2 (6):775–78. doi: 10.1042/ETLS20180172.

Eglen, S. J., and D. Nüst. 2019. CODECHECK: An open-science initiative to facilitate sharing of computer programs and results presented in scientific publications. In *The 14th Munin Conference on Scholarly Publishing 2019, Septentrio Conference Series*. University Library, UiT The Arctic University of Norway. https://doi.org/10.7557/5.4910.

Emsley, I., and D. De Roure. 2018. A framework for the preservation of a Docker container. *International Journal of Digital Curation* 12 (2):125–35. doi: 10.2218/ijdc.v12i2.509.

Estop, H. 2019. SAGE trials Code Ocean to improve research reproducibility. Accessed April 24, 2020. https://journalsblog.sagepub.com/blog/sage-trials-code-ocean-to-improve-research-reproducibility.

Fanelli, D. 2018. Opinion: Is science really facing a reproducibility crisis, and do we need it to? *Proceedings of the National Academy of Sciences* 115 (11):2628–31. doi: 10.1073/pnas.1708272114.

Foster, I. 2018. Research infrastructure for the safe analysis of sensitive data. *The Annals of the American Academy of Political and Social Science* 675 (1):102–20. doi: 10.1177/0002716217742610.

Gentleman, R., and D. Temple Lang. 2007. Statistical analyses and reproducible research. *Journal of Computational and Graphical Statistics* 16 (1):1–23. doi: 10.1198/106186007X178663.

Gil, Y., C. H. David, I. Demir, B. T. Essawy, R. W. Fulweiler, J. L. Goodall, L. Karlstrom, H. Lee, H. J. Mills, J.-H. Oh, et al. 2016. Toward the geoscience paper of the future: Best practices for documenting and sharing research from data to software to provenance. *Earth and Space Science* 3 (10):388–415. doi: 10.1002/2015EA000136.

Giraud, T., and N. Lambert. 2017. Reproducible cartography. In *Advances in cartography and GIScience*, ed. M. P. Peterson, 173–83. Cham, Switzerland: Springer. doi:10.1007/978-3-319-57336-6_13.

Greenbaum, D., J. Rozowsky, V. Stodden, and M. Gerstein. 2017. Structuring supplemental materials in support of reproducibility. *Genome Biology* 18 (1):64. doi: 10.1186/s13059-017-1205-3.

Gronenschild, E. H. B. M., P. Habets, H. I. L. Jacobs, R. Mengelers, N. Rozendaal, J. van Os, and M. Marcelis. 2012. The effects of FreeSurfer version, workstation type, and Macintosh operating system version on anatomical volume and cortical thickness measurements. *PLoS ONE* 7 (6):e38234. doi: 10.1371/journal.pone.0038234.

Harris, R., D. O'Sullivan, M. Gahegan, M. Charlton, L. Comber, P. Longley, C. Brunsdon, N. Malleson, A. Heppenstall, A. Singleton, et al. 2017. More bark than bytes? Reflections on 21+ years of geocomputation. *Environment and Planning B: Urban Analytics and City Science* 44 (4):598–617. doi: 10.1177/2399808317710132.

Higman, R., D. Bangert, and S. Jones. 2019. Three camps, one destination: The intersections of research data management, FAIR and open. *Insights* 32 (1):1–9. doi:10.1629/uksg.468.

Hinz, M., D. Nüst, B. Proß, and E. Pebesma. 2013. Spatial statistics on the geospatial Web. In *The 16th AGILE International Conference on Geographic Information Science, Short papers*. *AGILE*, ed. D. Vandenbroucke, B. Bucher, and J. Crompvoets, 1–7. Leuven, Belgium: AGILE. https://doi.org/10.31223/osf.io/j8x2e.

Hirst, T. 2019. "Fragment—Some rambling thoughts on computing environments in education. Accessed April 24, 2020. https://blog.ouseful.info/2019/03/20/fragment-some-rambling-thoughts-on-computing-environments-in-education/.

Howe, B. 2012. Virtual appliances, cloud computing, and reproducible research. *Computing in Science & Engineering* 14 (4):36–41. doi: 10.1109/MCSE.2012.62.

Jupyter Project, M. Bussonnier, J. Forde, J. Freeman, B. Granger, T. Head, C. Holdgraf, K. Kelley, G. Nalvarte, A. Osheroff, et al. 2018. Binder 2.0—Reproducible, interactive, sharable environments for science at scale. *Proceedings of the 17th Python in Science Conference*, 113–20. https://doi.org/10.25080/Majora-4af1f417-011.

Kedron, P., A. E. Frazier, A. B. Trgovac, T. Nelson, and A. S. Fotheringham. 2019. Reproducibility and replicability in geographical analysis. *Geographical Analysis*. Advance online publication. doi: 10.1111/gean.12221.

Kluyver, T., B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonier, J. Frederic, K. Kelley, et al. 2016. Jupyter Notebooks—A publishing format for reproducible computational workflows. In *Proceedings of the 20th International Conference on Electronic Publishing*, ed. F. Loizides and B. Schmidt, 87–90. Amsterdam, The Netherlands. https://doi.org/10.3233/978-1-61499-649-1-87.

Knoth, C., and D. Nüst. 2017. Reproducibility and practical adoption of GEOBIA with open-source software in Docker containers. *Remote Sensing* 9 (3):290. doi: 10.3390/rs9030290.

Knuth, D. E. 1984. Literate programming. *The Computer Journal* 27 (2):97–111. doi: 10.1093/comjnl/27.2.97.

Konkol, M., and C. Kray. 2019. In-depth examination of spatiotemporal figures in open reproducible research. *Cartography and Geographic Information Science* 46 (5):412–27. doi:10.1080/15230406.2018.1512421.

Konkol, M., C. Kray, and M. Pfeiffer. 2019. Computational reproducibility in geoscientific papers: Insights from a series of studies with geoscientists and a reproduction study. *International Journal of Geographical Information Science* 33 (2):408–29. doi: 10.1080/13658816.2018.1508687.

Konkol, M., C. Kray, and J. Suleiman. 2019. Creating interactive scientific publications using bindings. *Proceedings of the ACM on Human-Computer Interaction* 3:1–16. doi: 10.1145/3331158.

Kray, C., E. Pebesma, M. Konkol, and D. Nüst. 2019. Reproducible research in geoinformatics: Concepts, challenges and benefits (Vision paper). In *14th International Conference on Spatial Information Theory (COSIT 2019)*, ed. S. Timpf, C. Schlieder, M. Kattenbeck, B. Ludwig, and K. Stewart, 1–8. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.COSIT.2019.8.

Kurtzer, G. M., V. Sochat, and M. W. Bauer. 2017. Singularity: Scientific containers for mobility of compute. *PLoS ONE* 12 (5):e0177459. doi: 10.1371/journal.pone.0177459.

Lees, J. M. 2012. Open and free: Software and scientific reproducibility. *Seismological Research Letters* 83 (5):751–52. doi: 10.1785/0220120091.

Lewis, K. P., E. Vander Wal, and D. A. Fifield. 2018. Wildlife biology, big data, and reproducible research. *Wildlife Society Bulletin* 42 (1):172–79. doi: 10.1002/wsb.847.

Markowetz, F. 2015. Five selfish reasons to work reproducibly. *Genome Biology* 16:274. doi: 10.1186/s13059-015-0850-7.

Marwick, B. 2015. How computers broke science—and what we can do to fix it. Accessed April 24, 2020.

https://theconversation.com/how-computers-broke-science-and-what-we-can-do-to-fix-it-49938.

Marwick, B. 2017. Computational reproducibility in archaeological research: Basic principles and a case study of their implementation. *Journal of Archaeological Method and Theory* 24 (2):424–50. doi: 10.1007/s10816-015-9272-9.

Marwick, B., C. Boettiger, and L. Mullen. 2018. Packaging data analytical work reproducibly using R (and friends). *The American Statistician* 72 (1):80–88. doi: 10.1080/00031305.2017.1375986.

Muenchow, J., S. Schäfer, and E. Krüger. 2019. Reviewing qualitative GIS research—Toward a wider usage of open-source GIS and reproducible research practices. *Geography Compass* 13 (6). doi: 10.1111/gec3.12441.

Munafò, M. R., B. A. Nosek, D. V. M. Bishop, K. S. Button, C. D. Chambers, N. Percie Du Sert, U. Simonsohn, E-J. Wagenmakers, J. J. Ware, and J. P. A. Ioannidis, 2017. A manifesto for reproducible science. *Nature Human Behaviour* 1:0021. doi: 10.1038/s41562-016-0021.

National Academies of Sciences, Engineering, and Medicine. 2019. *Reproducibility and replicability in science*. Washington, DC: National Academies Press. doi: 10.17226/25303.

Nosek, B. A., G. Alter, G. C. Banks, D. Borsboom, S. D. Bowman, S. J. Breckler, S. Buck, C. D. Chambers, G. Chin, G. Christensen, et al. 2015. Scientific standards: Promoting an open research culture. *Science* 348 (6242):1422–25. doi: 10.1126/science.aab2374.

Nüst, D., C. Boettiger, and B. Marwick. 2018. How to read a research compendium. *arXiv*:1806.09525. http://arxiv.org/abs/1806.09525.

Nüst, D., C. Granell, B. Hofer, M. Konkol, F. O. Ostermann, R. Sileryte, and V. Cerutti. 2018. Reproducible research and GIScience: An evaluation using AGILE conference papers. *PeerJ* 6:e5072. doi: 10.7717/peerj.5072.

Nüst, D., and M. Hinz. 2019. containerit: Generating Dockerfiles for reproducible research with R. *Journal of Open Source Software* 4 (40):1603. doi: 10.21105/joss.01603.

Nüst, D., M. Konkol, E. Pebesma, C. Kray, M. Schutzeichel, H. Przibytzin, and J. Lorenz. 2017. Opening the publication process with executable research compendia. *D-Lib Magazine* 23 (1–2). doi: 10.1045/january2017-nuest.

Nüst, D., F. Ostermann, R. Sileryte, B. Hofer, C. Granell, M. Teperek, A. Graser, K. Broman, and K. Hettne. 2019. *AGILE reproducible paper guidelines*. OSF. doi: 10.17605/OSF.IO/CB7Z8.

Nüst, D., and M. Schutzeichel. 2017. An architecture for reproducible computational geosciences. Poster presented at AGILE 2017, Wageningen, The Netherlands, June. doi:10.5281/zenodo.1478542.

O'Loughlin, J., P. Raento, J. P. Sharp, J. D. Sidaway, and P. E. Steinberg. 2015. Data ethics: Pluralism, replication, conflicts of interest, and standards in political geography. *Political Geography* 44:A1–A3. doi:10.1016/j.polgeo.2014.11.001.

Pebesma, E., R. Bivand, and P. J. Ribeiro. 2015. Software for spatial statistics. *Journal of Statistical Software* 63 (1):1–8. doi:10.18637/jss.v063.i01.

Pebesma, E., D. Nüst, and R. Bivand. 2012a. R for reproducible geographical research. Paper presented at the AAG Annual Meeting 2012, New York, February 24. Accessed August 31, 2020. http://pebesma.staff.ifgi.de/r_repr.pdf.

Pebesma, E., D. Nüst, and R. Bivand. 2012b. The R software environment in reproducible geoscientific research. *Eos: Transactions of the American Geophysical Union* 93 (16):163. doi: 10.1029/2012EO160003.

Pebesma, E., W. Wagner, M. Schramm, A. V. Beringe, C. Paulik, M. Neteler, and J. Reiche. 2017. OpenEO—A common, open source interface between Earth observation data infrastructures and front-end applications. Zenodo. https://doi.org/10.5281/zenodo.1065474.

Peng, R. D. 2011. Reproducible research in computational science. *Science* 334 (6060):1226–27. doi: 10.1126/science.1213847.

Pérignon, C., K. Gadouche, C. Hurlin, R. Silberman, and E. Debonnel. 2019. Certify reproducibility with confidential data. *Science* 365 (6449):127–28. doi: 10.1126/science.aaw2825.

Perkel, J. M. 2019. Make code accessible with these cloud services. *Nature* 575 (7781):247–48. doi: 10.1038/d41586-019-03366-x.

Piwowar, H. 2013. Altmetrics: Value all research products. *Nature* 493 (7431):159. doi: 10.1038/493159a.

Preston, B., and M. W. Wilson. 2014. Practicing GIS as mixed method: Affordances and limitations in an urban gardening study. *Annals of the Association of American Geographers* 104 (3):510–29. doi: 10.1080/00045608.2014.892325.

R Core Team. 2019. *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Ram, K. 2013. Git can facilitate greater reproducibility and increased transparency in science. *Source Code for Biology and Medicine* 8 (1):7. doi: 10.1186/1751-0473-8-7.

Rechert, K., T. Liebetraut, S. Kombrink, D. Wehrle, S. Mocken, and M. Rohland. 2017. Preserving containers. In *Forschungsdaten managen*, ed. J. Kratzke and V. Heuveline, 143–51. Heidelberg, Germany: heiBOOKS. http://books.ub.uni-heidelberg.de/heibooks/catalog/book/285.

Rule, A., A. Birmingham, C. Zuniga, I. Altintas, S.-C. Huang, R. Knight, N. Moshiri, M. H. Nguyen, S. B. Rosenthal, F. Perez, et al. 2019. Ten simple rules for writing and sharing computational analyses in Jupyter Notebooks. *PLoS Computational Biology* 15 (7):e1007007. doi: 10.1371/journal.pcbi.1007007.

Sandve, G. K., A. Nekrutenko, J. Taylor, and E. Hovig. 2013. Ten simple rules for reproducible computational research. *PLoS Computational Biology* 9 (10):e1003285. doi: 10.1371/journal.pcbi.1003285.

Santana-Perez, I., and M. S. Pérez-Hernández. 2015. Towards reproducibility in scientific workflows: An infrastructure-based approach. *Scientific Programming* 2015:1–11. doi: 10.1155/2015/243180.

Schönbrodt, F. 2019. Training students for the open science future. *Nature Human Behaviour* 3 (10):1031. doi: 10.1038/s41562-019-0726-z.

Shannon, J., and K. Walker. 2018. Opening GIScience: A process-based approach. *International Journal of Geographical Information Science* 32 (10):1911–26. doi: 10.1080/13658816.2018.1464167.

Sidhu, N., E. Pebesma, and G. Câmara. 2018. Using Google Earth Engine to detect land cover change: Singapore as a use case. *European Journal of Remote Sensing* 51 (1):486–500. doi:10.1080/22797254.2018.1451782.

Šimko, T., L. Heinrich, H. Hirvonsalo, D. Kousidis, and D. Rodríguez. 2019. REANA: A system for reusable research data analyses. *EPJ Web of Conferences* 214. doi: 10.1051/epjconf/201921406034.

Singleton, A. D., S. Spielman, and C. Brunsdon. 2016. Establishing a framework for open geographic information science. *International Journal of Geographical Information Science* 30 (8):1507–21. doi: 10.1080/13658816.2015.1137579.

Spielman, S. E., and A. Singleton. 2015. Studying neighborhoods using uncertain data from the American Community Survey: A contextual approach. *Annals of the Association of American Geographers* 105 (5):1003–25. doi: 10.1080/00045608.2015.1052335.

Stark, P. B. 2018. Before reproducibility must come preproducibility. *Nature* 557:613. doi: 10.1038/d41586-018-05256-0.

Stodden, V. 2009. The legal framework for reproducible scientific research: Licensing and copyright. *Computing in Science & Engineering* 11 (1):35–40. doi: 10.1109/MCSE.2009.19.

Stodden, V., and S. Miguez. 2014. Best practices for computational science: Software infrastructure and environments for reproducible and extensible research. *Journal of Open Research Software* 2 (1):e21. doi: 10.5334/jors.ay.

Stodden, V., J. Seiler, and Z. Ma. 2018. An empirical analysis of journal policy effectiveness for computational reproducibility. *Proceedings of the National Academy of Sciences of the United States of America* 115 (11):2584–89. doi: 10.1073/pnas.1708290115.

Sui, D., and P. Kedron. 2020. Reproducibility and replicability in the context of the contested identities of geography. *Annals of the American Association of Geographers*. doi: 10.1080/24694452.2020.1806024.

Sui, D., and S.-L. Shaw. 2018. Outlook and next steps: From human dynamics to smart and connected communities. In *Human dynamics research in smart and connected communities*, ed. S.-L. Shaw and D. Sui, 235–45. Cham, Switzerland: Springer International. doi: 10.1007/978-3-319-73247-3_13.

Tennant, J. P., H. Crane, T. Crick, J. Davila, A. Enkhbayar, J. Havemann, B. Kramer, R. Martin, P. Masuzzo, A. Nobes, et al. 2019. Ten hot topics around scholarly publishing. *Publications* 7 (2):34. doi: 10.3390/publications7020034.

The Turing Way Community, B. Arnold, L. Bowler, S. Gibson, P. Herterich, R. Higman, A. Krystalli, A. Morley, M. O'Reilly, and K. Whitaker. 2019. The Turing Way: A handbook for reproducible data science. Zenodo. https://doi.org/10.5281/zenodo.3233986.

Vandewalle, P., J. Kovacevic, and M. Vetterli. 2009. Reproducible research in signal processing. *IEEE Signal Processing Magazine* 26 (3):37–47. doi: 10.1109/MSP.2009.932122.

Verstegen, J. A. 2019. JudithVerstegen/PLUC_Mozambique: First release of PLUC for Mozambique (Version v1.0.0). *Zenodo*. https://doi.org/10.5281/zenodo.3519987.

Verstegen, J. A., D. Karssenberg, F. van der Hilst, and A. Faaij. 2012. Spatio-temporal uncertainty in spatial decision support systems: A case study of changing land availability for bioenergy crops in Mozambique. *Computers, Environment and Urban Systems* 36 (1):30–42. doi: 10.1016/j.compenvurbsys.2011.08.003.

Wainwright, J. 2020. Is critical human geography research replicable? *Annals of the American Association of Geographers*. doi: 10.1080/24694452.2020.1806025.

Waters, N. 2020. Motivations and methods for replication in geography: Working with data "streams." *Annals of the American Association of Geographers*. doi: 10.1080/24694452.2020.1806027.

Wilkinson, M. D., M. Dumontier, I. J. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J-W. Boiten, L. Bonino da Silva Santos, P. E. Bourne, et al. 2016. The FAIR guiding principles for scientific data management and stewardship. *Scientific Data* 3:160018. doi: 10.1038/sdata.2016.18.

Wilson, J. P., and P. A. Burrough. 1999. Dynamic modeling, geostatistics, and fuzzy classification: New sneakers for a new geography? *Annals of the Association of American Geographers* 89 (4):736–46. doi: 10.1111/0004-5608.00173.

Wilson, J. P., K. Butler, S. Gao, W. Li, and D. J. Wright. 2020. The replicability and reproducibility of the GIS software and algorithms used in environmental applications. *Annals of the American Association of Geographers*. doi: 10.1080/24694452.2020.1806026.

Xie, Y. 2015. *Dynamic documents with R and knitr*. 2nd ed. Boca Raton, FL: CRC.

DANIEL NÜST is Researcher at the Institute for Geoinformatics, University of Münster, 48149 Münster, Germany. E-mail: daniel.nuest@uni-muenster.de. He develops tools for creation and execution of research compendia in geography and geosciences in the project Opening Reproducible Research (o2r, https://o2r.info).

EDZER PEBESMA is Professor of Geoinformatics at University of Münster, 48149 Münster, Germany. E-mail: edzer.pebesma@uni-muenster.de. He is developer and maintainer of several popular R packages for handling and analyzing spatial and spatiotemporal data (sp, spacetime, gstat, sf).

11 Reproducible research and GIScience: An evaluation using AGILE conference papers

Authors & contribution Daniel Nüst (40%), Carlos Granell, Barbara Hofer, Markus Konkol, Frank O. Ostermann, Rusne Sileryte, Valentina Cerutti

Venue *PeerJ* (SNIP 2020: 1.04) 10.7717/peerj.5072

Date 07/2018

Licence Creative Commons Attribution (CC BY 4.0)

Repository https://github.com/nuest/reproducible-research-and-giscience

ERC https://o2r.uni-muenster.de/erc/F9Gxs

Reproducible research and GIScience: an evaluation using AGILE conference papers

Daniel Nüst[1], Carlos Granell[2], Barbara Hofer[3], Markus Konkol[1], Frank O. Ostermann[4], Rusne Sileryte[5] and Valentina Cerutti[4]

[1] Institute for Geoinformatics, University of Münster, Münster, Germany
[2] Institute of New Imaging Technologies, Universitat Jaume I de Castellón, Castellón, Spain
[3] Interfaculty Department of Geoinformatics - Z_GIS, University of Salzburg, Salzburg, Austria
[4] Faculty of Geo-Information Science and Earth Observation (ITC), University of Twente, Enschede, The Netherlands
[5] Faculty of Architecture and the Built Environment, Delft University of Technology, Delft, The Netherlands

ABSTRACT

The demand for reproducible research is on the rise in disciplines concerned with data analysis and computational methods. Therefore, we reviewed current recommendations for reproducible research and translated them into criteria for assessing the reproducibility of articles in the field of geographic information science (GIScience). Using this criteria, we assessed a sample of GIScience studies from the Association of Geographic Information Laboratories in Europe (AGILE) conference series, and we collected feedback about the assessment from the study authors. Results from the author feedback indicate that although authors support the concept of performing reproducible research, the incentives for doing this in practice are too small. Therefore, we propose concrete actions for individual researchers and the GIScience conference series to improve transparency and reproducibility. For example, to support researchers in producing reproducible work, the GIScience conference series could offer awards and paper badges, provide author guidelines for computational research, and publish articles in Open Access formats.

Submitted 13 March 2018
Accepted 4 June 2018
Published 13 July 2018

Corresponding author
Daniel Nüst,
daniel.nuest@uni-muenster.de

Academic editor
Nicolas Rougier

Additional Information and Declarations can be found on page 19

DOI 10.7717/peerj.5072

Subjects Science Policy, Computational Science, Data Science, Spatial and Geographic Information Science
Keywords GIScience, Open science, Reproducible research, Data science, AGILE, Reproducible conference publications, Open access

INTRODUCTION

A "reproducibility crisis" has been observed and discussed in several scientific disciplines such as economics (*Ioannidis, Stanley & Doucouliagos, 2017*), medical chemistry (*Baker, 2017*), neuroscience (*Button et al., 2013*), and for scientific studies in general, across various disciplines (*Ioannidis, 2005*). The lack of reproducibility in scientific studies stems from researchers facing challenges in understanding and re-creating others' results, a situation that is common in data-driven and algorithm-based research. However, even though algorithms are becoming more relevant in GIScience, a reproducibility crisis has not yet been observed in this field. In GIScience, failures to reproduce are not yet a topic of broad and common interest, but this field should be working to prevent a crisis instead of reacting to one. Given this motivation, we aim to adapt the observations and challenges of reproducible research from other disciplines to the GIScience community, and then use these adapted criteria to assess the reproducibility of research produced by members of this field and presented at a conference for the Association of Geographic Information Laboratories in Europe (AGILE), which has organised annual conferences on GIScience topics since 1998 (https://agile-online.org/index.php/conference/past-agile-conferences; all links last accessed Nov 23 2017). The conference series's broad topical scope and its notoriety in the GIScience community make it a reasonable starting point to investigate the level of reproducibility in GIScience research. This publication continues a collaboration started at the AGILE 2017 pre-conference workshop "Reproducible Geosciences Discussion Forum" (http://o2r.info/reproducible-agile/2017/).

In this work, we first review papers from other disciplines, which provide recommendations on how to make research more transparent and reproducible. This literature study provides the general criteria we used to systematically evaluate a sample of 32 AGILE conference papers from the last eight years. From this evaluation and the lessons learned by others, we formulate recommendations for the AGILE community, ranging from individual researchers' practises to practises to be carried out by conference organisations. Because of its international reach, broad range of topics, and long-sustained community, we argue that AGILE is in a unique position to take a leading role to promote reproducibility in GIScience. The following research questions (RQs) structure the remainder of this article:

RQ 1 *What are general criteria for reproducible research?*
RQ 2 *What are key criteria for reproducible research in GIScience?*
RQ 3 *How do AGILE conference papers meet these reproducibility criteria?*
RQ 4 *What strategies could improve reproducibility in AGILE contributions and GIScience in general?*

'Related work' provides references targeting RQ 1, which are detailed further in 'Assessment of Reproducibility' to address RQ 2. The results of applying the criteria ('Results') answer RQ 3, and the discussion ('Discussion') responds to RQ 4.

RELATED WORK

Reproducible research is a frequently discussed topic in editorials and opinion articles in high-impact journals (cf. 'Recommendations and suggestions in literature'). Extensive studies on the state of reproducibility have been conducted in some domains, e.g., in computer systems research (*Collberg & Proebsting, 2016*, see also project website http://reproducibility.cs.arizona.edu/) or bioinformatics (*Hothorn & Leisch, 2011*). For the field of geoscience research, some discussion of reproducibility has happened sporadically for quantitative geography (*Brunsdon, 2016*), cartography (*Giraud & Lambert, 2017*) and volunteered geographic information (VGI) (*Ostermann & Granell, 2017*), but no comprehensive study of reproducibility in the GIScience domain has been conducted.

Even though recent studies highlight an increased awareness of and willingness for open research, they also draw attention to persistent issues and perceived risks associated with data sharing and publication, such as the lack of rewards and the risk of losing recognition in a competitive academic environment (*Tenopir et al., 2011*; *Ioannidis, 2014*). Beyond individual concerns, there are systematic impediments. Some studies have mentioned that reproducible research is not in the individual researcher's domain but instead is a multi-actor endeavour, which requires a collective mind shift within the scientific community (*Stodden et al., 2016*; *McNutt, 2014*; *Ioannidis, 2014*). Funding agencies, research institutions, publishers, journals, and conferences are all responsible for promoting reproducible research practises. Existing examples (journals welcoming reproducible papers: *Information Systems* (https://www.elsevier.com/journals/information-systems/0306-4379), *Vadose Zone Journal* (https://dl.sciencesocieties.org/publications/vzj/articles/14/10/vzj2015.06.0088), *GigaScience* (https://academic.oup.com/gigascience/pages/instructions_to_authors), *JASA* (http://www.sph.umn.edu/news/wolfson-named-reproducibility-editor-asa-statistics-journal/)) are remarkable, yet in general they are scarce and testimonial.

Another hindrance to reproducible research is that, given the distinct nature and variety of research practises, the term reproducibility has been used with varying meanings and may stand for repeatability, robustness, reliability or generalisability of scientific results (*Editorial, 2016*). There has been some confusion about contradictory meanings in the literature (see for example Mark Liberman's "Replicability vs. reproducibility" (http://languagelog.ldc.upenn.edu/nll/?p=21956)). Wikipedia's definition (https://en.wikipedia.org/wiki/Reproducibility) is widely used to distinguish both terms:

> Reproducibility is the ability to get the same research results using the raw data and computer programs provided by the researchers. A related concept is replicability, meaning the ability to independently achieve similar conclusions when differences in sampling, research procedures and data analysis methods may exist.

Leek & Peng (2015) similarly define reproducibility as the ability to compute exactly the same results of a study based on original input data and details of the analysis workflow. They refer to replicability as obtaining similar conclusions about a research question derived from an independent study or experiment. A *Nature* *Editorial (2016)* defines reproducibility as achieved when "another scientist using the same methods gets similar results and can draw the same conclusions". *Stodden et al.* (*2016*, p. 1240) base their reproducibility enhancement principles on "the ability to rerun the same computational steps on the same data the original authors used". While most statements in the literature show that researchers have a common understanding of what these two concepts mean, the interpretation and application of these concepts by the scientific communities is still inconsistent and leads to different methods and conventions for disseminating scientific work. In the field of GIScience, *Ostermann & Granell* (*2017*, p. 226) argue that "a reproduction is always an exact copy or duplicate, with exactly the same features and scale, while a replication resembles the original but allows for variations in scale for example". Hence, reproducibility is exact whereas replicability means confirming the original conclusions, though not necessarily with the same input data, methods, or results.

MATERIALS & METHODS

The paper corpus

We consider the AGILE conference series publications to be a representative sample of GIScience research because of the conferences' broad topical scope. Since 2007, the AGILE conference has had a full paper track (cf. *Pundt & Toppen, 2017*) and a short paper track with blind peer review. The latter is published for free on the AGILE website. Legal issues (full paper copyrights lie with the publisher Springer, see https://agile-online.org/index.php/conference/springer-series) and practical considerations (assessment of reproducibility is a manual time-consuming process; old publications introduce bias because of software unavailability) led us to choose to apply our evaluation only to nominees for the "best full and short paper" awards for 2010, and 2012 to 2017 (no records for a best paper award could be found for 2011). Typically, there are three full paper and two short paper candidates per year (https://agile-online.org/index.php/conference/proceedings). Exceptions are 2013 with only two full papers and 2010 without any short papers. The corpus contains 32 documents: 20 full papers (7.9% of 253 full papers since 2007) and 12 short papers[1].

An exploratory text analysis of the paper corpus investigated the occurrence of keywords related to reproducibility, data, and software. The code is published as an executable document[2] (cf. *Nüst, 2018*). Most frequent terms mentioned are illustrated by Fig. 1. Table 1 shows keyword occurrence per paper and in the entire corpus (bottom row "Total"). Keyword identification uses word stems, e.g., `reproduc` includes "reproducible", "reproduce", and "reproduction" (see *Nüst (2018)* for details). While this matches common and established (technical) terms, it might not capture all phrases an author could use to describe reproducibility-related aspects of the work. Putting these corner cases aside, the numbers are clear enough to draw the following conclusions. Few papers mention reproducibility, some mention code and software, and many mention processes, algorithms, and data. This points to data and analysis being generally discussed in the publications, while being able to recreate the data and analyses is not deliberated.

[1] Full number of short papers cannot be derived automatically, (cf. *Nüst, 2018*).

[2] Using R Markdown, see http://rmarkdown.rstudio.com/.

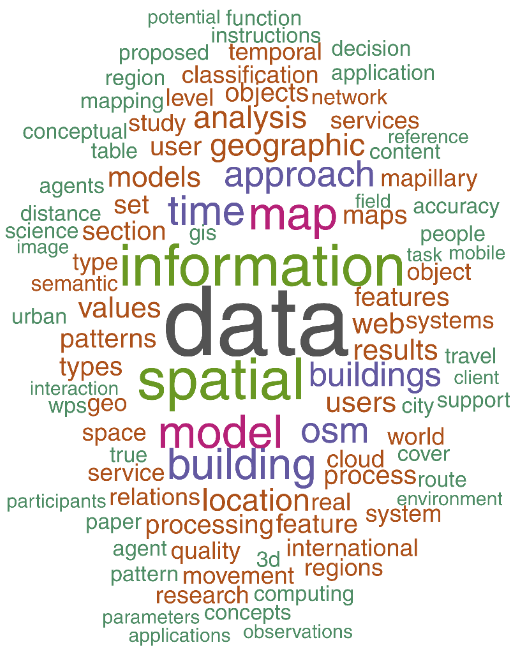
Assessment of reproducibility

Recommendations and suggestions in literature

Scientists from various disciplines suggest guidelines for open and reproducible research considering the specific characteristics of their field, e.g., *Sandve et al. (2013)* for life sciences, *McNutt (2014)* for field sciences, and *Gil et al. (2016)* for the geoscience paper of the future. Our goal was to first identify common recommendations that are applicable across research fields, including GIScience.

Suggested guidelines found in the reproducibility-related papers we investigated were categorised according to four aspects: *data* concerns all inputs; *methods* cover everything on the analysis of data, e.g., algorithms, parameters, and source code; *results* include (intermediate) data and parameters as well as outcomes such as statistics, maps, figures, or new datasets; and *structure* considers the organisation and integration of the other aspects. While some of the publications focus on specific aspects such as data (*Gewin, 2016*), code (*Stodden & Miguez, 2014*), workflow semantics (*Scheider, Ostermann & Adams, 2017*), and

A



B

place	word	n	# papers
1	data	1058	31
2	information	589	32
3	spatial	577	30
4	map	411	25
5	model	411	25
6	building	381	24
7	time	378	30
8	approach	297	32
9	osm	292	8
10	buildings	266	15
11	geographic	249	28
12	location	239	26
13	analysis	229	28
14	users	225	19
15	results	207	30
16	web	206	21
17	models	202	20
18	values	202	23
19	patterns	196	16
20	maps	189	20

Figure 1 **Two illustrations of the test corpus papers: word cloud, scaled and coloured by number of occurrence of words with at least 100 occurrences (96 unique words) (A); top words sorted by overall occurrence and number of papers including the word at least once (B).**
Full-size DOI: 10.7717/peerj.5072/fig-1

results (Sandve et al., 2013), others provide an all-embracing set of research instructions (Stodden et al., 2016; Nosek et al., 2015; Gil et al., 2016).

Data. A recurring aspect we encountered is making data accessible for other researchers (cf. Reichman, Jones & Schildhauer, 2011), ideally as archived assets having a Digital Object Identifier (DOI) and supplemented by structured metadata (Gewin, 2016). Stodden et al. (2016) consider legal aspects, such as sharing data publicly under an open license to clarify reusability. Further recommendations refer to modifying scientific practises, such as citation standards to ensure proper acknowledgement (Nosek et al., 2015), fostering data transparency (McNutt, 2014), and using open data formats to mitigate potentially disappearing proprietary software (Gewin, 2016). According to Reichman, Jones & Schildhauer (2011), journals and funders should include data sharing in their guidelines.

Methods. A key requirement (Sandve et al., 2013) concerning methods is sharing used or developed software, where software should be published using persistent links (Stodden et al., 2016; Gil et al., 2016) and descriptive metadata (Reichman, Jones & Schildhauer, 2011). Similar to data, important concerns for software are open licensing (Barba, 2016) and proper credits (Stodden et al., 2016). Researchers can accomplish software transparency by using version control systems (cf. Sandve et al., 2013), and transparency mandates

Table 1 **Reproducibility-related keywords in the corpus, ordered by sum of matches per paper.** For full references of the corpus papers see Supplemental Material.

Citation	Reproduc.	Replic.	Repeatab.	Code	Software	Algorithm(s)	(pre)process.	Data	Result(s)	All
Foerster et al. (2012)	0	0	0	2	3	11	140	129	41	326
Wiemann & Bernard (2014)	0	0	0	0	0	0	20	98	3	123
Mazimpaka & Timpf (2015)	0	0	0	3	0	4	4	97	10	118
Steuer et al. (2015)	0	0	0	0	0	25	12	64	17	118
Schäffer et al. (2010)	0	0	0	0	10	1	26	65	6	108
Rosser et al. (2016)	0	0	0	0	2	1	42	51	6	105
Gröchening et al. (2014)	0	0	0	0	0	3	2	69	27	101
Almer et al. (2016)	0	0	0	1	1	1	22	53	22	100
Magalhães et al. (2012)	0	0	0	2	1	20	52	9	1	85
Juhász & Hochmair (2016)	0	0	0	0	1	1	2	55	11	70
Wiemann (2016)	0	0	0	0	3	0	8	55	1	69
Fan et al. (2014)	0	0	0	0	0	3	8	44	12	67
Merki & Laube (2012)	0	0	0	0	0	9	6	40	6	62
Zhu et al. (2017)	2	2	0	2	0	10	7	32	6	61
Kuhn & Ballatore (2015)	0	0	1	2	14	1	5	26	8	58
Soleymani et al. (2014)	1	0	0	0	0	0	4	39	9	56
Fogliaroni & Hobel (2015)	0	0	0	0	0	3	14	30	5	52
Osaragi & Hoshino (2012)	0	0	0	0	0	0	5	36	7	48
Stein & Schlieder (2013)	0	0	0	0	0	0	3	42	3	48
Körner et al. (2010)	0	0	0	0	0	6	5	30	4	45
Knoth et al. (2017)	0	0	0	3	2	1	6	25	7	44
Raubal & Winter (2010)	0	0	0	1	1	1	18	0	13	34
Konkol et al. (2017)	1	0	0	3	1	1	2	4	19	31
Kiefer et al. (2012)	1	0	0	0	2	1	9	10	8	31
Haumann et al. (2017)	0	0	0	0	0	6	8	10	2	26
Josselin et al. (2016)	0	0	0	0	2	1	9	5	8	25
Heinz & Schlieder (2015)	1	0	0	2	1	3	2	14	2	25
Osaragi & Tsuda (2013)	0	0	0	1	1	0	3	16	2	23
Baglatzi & Kuhn (2013)	1	0	0	0	0	0	6	12	3	22
Scheider et al. (2014)	0	0	0	0	1	0	0	13	4	19
Brinkhoff (2017)	0	0	0	0	1	9	2	3	2	17
Schwering et al. (2013)	0	0	0	0	0	4	2	3	5	14
Total	7	2	1	22	47	126	454	1,179	280	2,131

using open source instead of proprietary software (*Steiniger & Hay, 2009*). Since full computational reproducibility can depend on exact software versions (*Gronenschild et al., 2012*), the computational environment needs to be reported (cf. *Stodden et al., 2016*; *Gil et al., 2016*). Further software-specific recommendations are workflow tracking (*Stodden & Miguez, 2014*) and keeping a record of analysis parameters (*Gil et al., 2016*). *Sandve et al. (2013)* suggest avoiding manual data manipulation steps and instead using scripts to increase transparency in data preprocessing.

Results. Sandve et al. (2013) focus on results-related guidelines such as storing intermediate results and noting seeds if computations include randomness. Journals should conduct a reproducibility check prior to publication (Stodden et al., 2016) or funding should be explicitly granted for making research results repeatable (Collberg & Proebsting, 2016). Finally, Barba (2016) describes the contents and benefits of a "reproducibility package" to preserve results.

Structure. While the papers discussed above focus on specific aspects of reproducibility, an overarching structure for all facets of research can provide important context. But none of the suggestions for packaging workflows are widely established, for example Gentleman & Lang (2007) use programming language packaging mechanisms, Bechhofer et al. (2013) Linked Data, or Nüst et al. (2017) nested containers.

Section summary. Most recommendations and suggestions to foster open and reproducible research address data and methods. Particularly, methods cover a broad range of aspects including recommendations on data preprocessing, the actual analysis, and the computational environment. Results receive less attention, possibly because they are strongly connected with other aspects. While most of the recommendations address authors, only few target journals and research institutions.

Definition and criteria

This paper focuses on reproducibility in the context of conference publications and adopts the described consensus (see 'Related work') for the following definition.

> A reproducible paper ensures a reviewer or reader can recreate the computational workflow of a study or experiment, including the prerequisite knowledge and the computational environment. The former implies the scientific argument to be understandable and sound. The latter requires a detailed description of used software and data, and both being openly available.

We build on the recommendations from 'Recommendations and suggestions in literature' and differentiate data, methods, and results as separate dimensions of reproducibility. We conceptualised each reproducibility dimension as a criterion, and for each criterion, we developed levels of attained reproducibility. In order to increase reproducibility of this study and improve inter-rater agreement, we created a corresponding rubric that explains the requirements. Together, the three criteria and their levels address specifics of GIScience research and allow for a fine-grained assessment of reproducibility.

However, early during the evaluation process, it became clear that the assessed corpus papers showed great variation in data, methods, and type of results. For example, data used during the reported studies varies from spatial data to qualitative results from surveys. Methods are particularly diverse, ranging from the application of spatial analysis operations to statistical approaches or simulations. Results include maps, formulas, models or diagrams. Therefore, we decided to split the methods criterion into three sub-criteria addressing the distinct phases and respective software tools: data preprocessing, analysis methods and workflows, and computational environment. Following this change, we re-evaluated already assessed corpus papers.

Figure 2 shows the reproducibility criteria for each of the categories *Data*, *Methods*, and *Results*, and their levels. The levels are either *not applicable* (`NA`) or range from *no* (value of `0`) to *full* (`3`) reproducibility. The level `0` unavailable means that there is insufficient documentation in the paper, or the information is only available upon request (since this cannot be guaranteed and we could not check availability for all studies). The level `1` documented means that there is sufficient information to recreate at least parts of the study, but no concrete data or code or models. Level `2` available means that required data and code is provided. Finally, level `3` available, open and permanent, adds the requirement of unrestricted and sustainable access, e.g., through permanent links to open repositories containing data, relevant methods and workflows (such as software versions, hardware specifications, scripts), and all results (including intermediary ones or those not discussed in detail in the study). The *Methods* criteria do not include the "permanent" aspect, because there is no suitable single identifier to define the complex properties of code, libraries and system environment, although a DOI may be used to collectively identify all these items as source or binary files. Licensing is important for reproducibility, because only a clear license, which ideally is well-known and established, allows use of copyrighted content. So in this sense "open" means "open enough for reproduction", but in practice the used licenses should be fully open and allow modification and sharing beyond mere access and use[3].

[3]Cf. The Open Definition, https://opendefinition.org/: "Open data and content can be freely used, modified, and shared by anyone for any purpose".

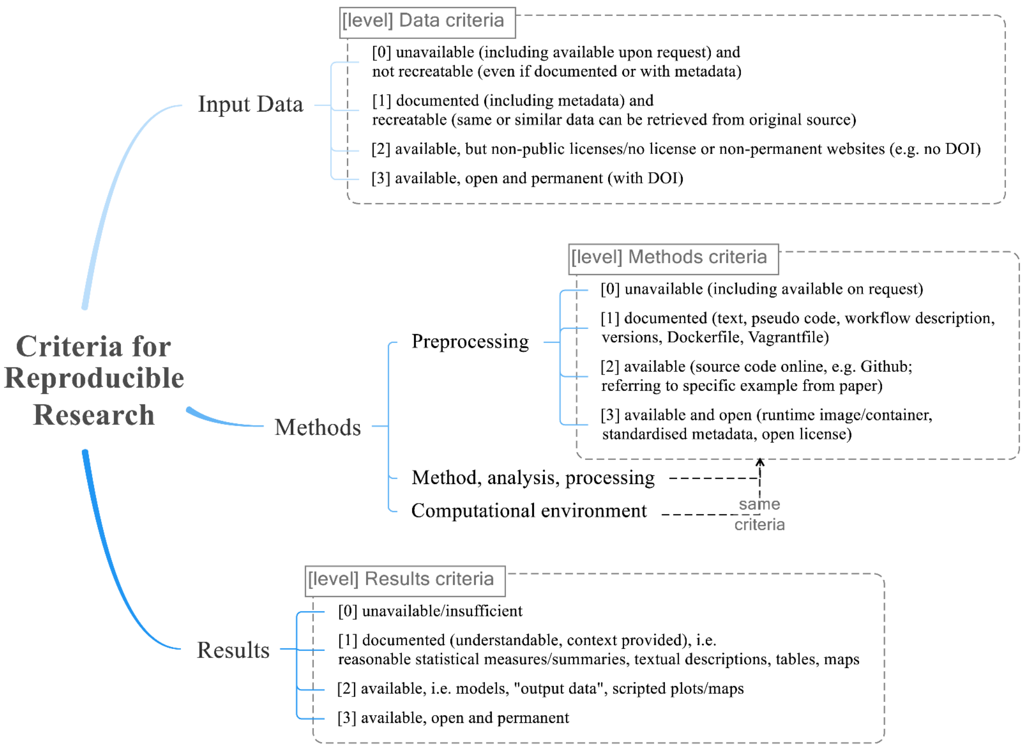
The intermediate levels (`1` and `2`) allow a differentiated evaluation. For example for data at level `1`, data is not accessible but documented sufficiently, so others can recreate it; at level `2` data is available yet in a non-persistent way or with a restrictive license. The requirements are cumulative, meaning that higher levels of reproducibility include lower levels' requirements. The reproducibility rubric was developed in iterative discussions between all raters, using the examined literature on reproducibility as point of reference.

By design, our criteria cannot be applied to conceptual research publications, namely those without data or code. Their evaluation is covered by an editorial peer review process (see for example *Ferreira et al. (2016)* for history and future of peer review), and assessing the merit of an argument is beyond the scope of this work.

Author feedback on assessment of reproducibility (survey)

To better understand the reasons behind the scores and to give the authors an opportunity to respond after the reproducibility of their research was assessed, we designed a survey using Google Forms (https://www.google.com/forms/about/) (see Table 2, cf. *Baker (2016a)* for a large scale survey on the topic). The full survey, as it was shown to the participants, is included in the Supplemental Material.

Along with the survey, authors were provided with the results of our evaluation of their specific papers, and they were asked to express their agreement or disagreement with the results. The four main questions of the survey were designed to find out whether authors considered reproducibility important in the first place, and if so, what prevented them



Criteria for Reproducible Research
Input Data
[level] Data criteria
[0] unavailable (including available upon request) and not recreatable (even if documented or with metadata)
[1] documented (including metadata) and recreatable (same or similar data can be retrieved from original source)
[2] available, but non-public licenses/no license or non-permanent websites (e.g. no DOI)
[3] available, open and permanent (with DOI)
Methods
Preprocessing
[level] Methods criteria
[0] unavailable (including available on request)
[1] documented (text, pseudo code, workflow description, versions, Dockerfile, Vagrantfile)
[2] available (source code online, e.g. Github; referring to specific example from paper)
[3] available and open (runtime image/container, standardised metadata, open license)
Method, analysis, processing
Computational environment
same criteria
Results
[level] Results criteria
[0] unavailable/insufficient
[1] documented (understandable, context provided), i.e. reasonable statistical measures/summaries, textual descriptions, tables, maps
[2] available, i.e. models, "output data", scripted plots/maps
[3] available, open and permanent

Figure 2 The final reproducible research criteria used for the evaluation. The categories *Data*, *Methods* (sub-categories: preprocessing, method/analysis/processing, and computational environment), and *Results* each have four levels ranging from 0 = not reproducible to 3 = fully reproducible.
Full-size DOI: 10.7717/peerj.5072/fig-2

from fully achieving it. Finally, the authors were asked to provide their own opinion and suggestions for the AGILE community to encourage publishing fully reproducible papers.

RESULTS

Assessment of reproducibility

To address RQ 3, we reviewed the papers in the corpus with the introduced criteria. Our objective in publishing the full evaluation results is not to criticise or rank individual papers, but to identify the current overall state of reproducibility in GIScience research in a reproducible manner. The scientific merit of all papers was already proven by their nomination for the best paper award.

The procedure was as follows: First, we determined a maximum number of papers for a single evaluator to reach two evaluators per paper. Second, we grouped evaluators according to their affiliation or research group. Evaluators then chose to review papers without a conflict of interest on a first come first served basis until two goals were achieved: the evaluator reached her maximum number of reviews and two evaluators from different research groups reviewed the paper. For assigning a level of reproducibility, the general guideline was to apply the lower of two possible levels in cases of doubt, such as partial

Table 2 **Survey questions (except for paper identification questions; for full questionnaire see Supplemental Material).**

Question	Possible answers
1. Have you considered the reproducibility of research published in your nominated paper?	• Yes, it is important to me that my research is fully reproducible • Yes, I have somewhat considered reproducibility • No, I was not concerned with it • Other (please add)
2. Do you agree with our rating of your publication? Please comment.	*Open answer*
3a. Please rate how strongly the following circumstances have hindered you from providing all data, methods and results used/developed during your research?	• The need to invest more time into the publication • Lack of knowledge how to include data/methods/results into the publication • Lack of tools that would help to attach data/methods/results to the publication • Lack of motivation or incentive • Legal restrictions Available ratings: • Not at all • Slightly hindered • Moderately hindered • Strongly hindered • Main reason
3b. Please add here if there were any other hindering circumstances	*Open answer*
4. What would you suggest to AGILE community to encourage publishing fully reproducible papers?	*Open answer*

fulfilment of a criterion or disagreement between the evaluators. All reviewers discussed disagreements and open questions after an initial round of evaluation comprising one to three reviews per researcher, and after completing all reviews. Because the assessment focuses on algorithmic and data-driven research papers, five fully conceptual papers were not assessed, while 15 partly conceptual ones were included. Notably, the data preprocessing criterion did not apply to 14 research papers. Table 3 shows the assessment's results.

Figure 3 shows the distribution of reproducibility levels for each criterion. None of the papers reach the highest level of reproducibility in any category. Only five papers reach level 2 in the data criterion, which is still the highest number of that level across all categories. Especially problematic is the high number of papers (19) with level `0` for data, meaning that the specific data is not only unavailable but it is not re-createable from the information in the paper. Data preprocessing applies to 18 publications, and the levels are low. Only one publication has level `2`. Concerning the methods and results criteria, 19 out of 32 papers have level `1` in both, meaning an understandable documentation is provided in the text.

Figure 4 shows that average reproducibility levels are low and do not change significantly over time, with the mean over all categories being below level `1` for all years. The categories are ordinal variables, meaning they have an implicit order but an unknown "distance" between them. They can be compared (`3` is higher than `2`), but absolute differences in numbers must not be interpreted. Moving one level up from `0` to `1` is not the same as from `2` to `3`. Averaging on ordinal variables must be conducted with care: `Mode` and `median`

Table 3 **Reproducibility levels for paper corpus; '-' is category not available.** For full references of the corpus papers see Supplemental Material.

Author	Short paper	Input data	Preprocessing	Method/analysis/processing	Computational environment	Results
Zhu et al. (2017)		0	1	1	1	1
Knoth et al. (2017)		0	–	0	1	1
Konkol et al. (2017)		2	2	1	1	1
Haumann et al. (2017)	X	0	1	1	0	1
Brinkhoff (2017)	X	0	–	1	0	0
Almer et al. (2016)		0	–	1	1	1
Wiemann (2016)		2	–	1	1	1
Juhász & Hochmair (2016)		0	1	1	0	0
Josselin et al. (2016)	X	1	–	0	0	1
Rosser et al. (2016)	X	0	–	1	0	0
Kuhn & Ballatore (2015)		–	–	–	–	–
Mazimpaka & Timpf (2015)		2	1	1	1	1
Steuer et al. (2015)		2	0	1	1	1
Fogliaroni & Hobel (2015)	X	–	–	–	–	–
Heinz & Schlieder (2015)	X	0	0	1	1	1
Scheider et al. (2014)		1	1	2	1	1
Gröchening et al. (2014)		2	0	1	0	1
Fan et al. (2014)		0	1	1	0	1
Soleymani et al. (2014)	X	0	0	1	0	0
Wiemann & Bernard (2014)	X	0	0	1	0	0
Osaragi & Tsuda (2013)		0	1	1	0	1
Baglatzi & Kuhn (2013)		–	–	–	–	–
Li et al. (2013)	X	0	0	1	–	1
Stein & Schlieder (2013)	X	0	–	1	0	1
Osaragi & Hoshino (2012)		0	0	1	0	1
Magalhães et al. (2012)		0	0	1	0	0
Foerster et al. (2012)		1	–	1	1	1
Merki & Laube (2012)	X	0	–	1	1	1
Kiefer et al. (2012)	X	0	1	1	0	1
Raubal & Winter (2010)		–	–	–	–	–
Schäffer et al. (2010)		0	0	1	1	1
Körner et al. (2010)		–	–	–	–	–

are mostly seen as acceptable averaging functions for ordinal data, while the `mean` is seen inapplicable by some.

We decided not to use median or mode, because they hide all differences between the categories. The `mean` should not be applied for a single paper, whereby all categories in a single paper are averaged, because different evaluation rules would be combined into a meaningless number. Being aware of these limitations and the small dataset size, we opted to apply the mean and a statistical summary to categories to compare values between the different categories, and to compare the two large groups within the paper corpus (full and short papers).

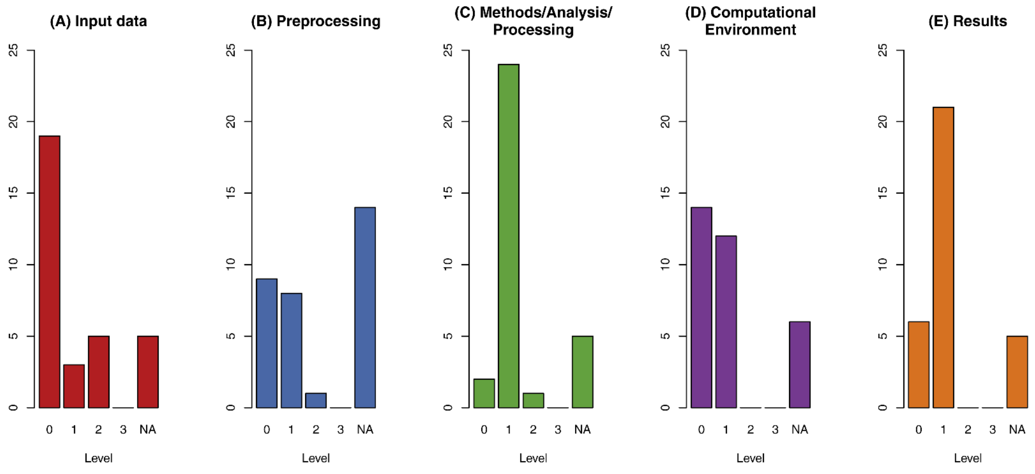


Figure 3 **Results of reproducibility assessment across all categories for the assessment of reproducibility: *Data* (A), *Methods* with sub-categories preprocessing (B), method/analysis/processing (C) and computational environment (D), and *Results* (E).** The level of reproducibility ranges from `0` (not reproducible) to `3` (fully reproducible); `NA`s include 5 conceptual papers (all categories are `NA`).

Full-size DOI: 10.7717/peerj.5072/fig-3

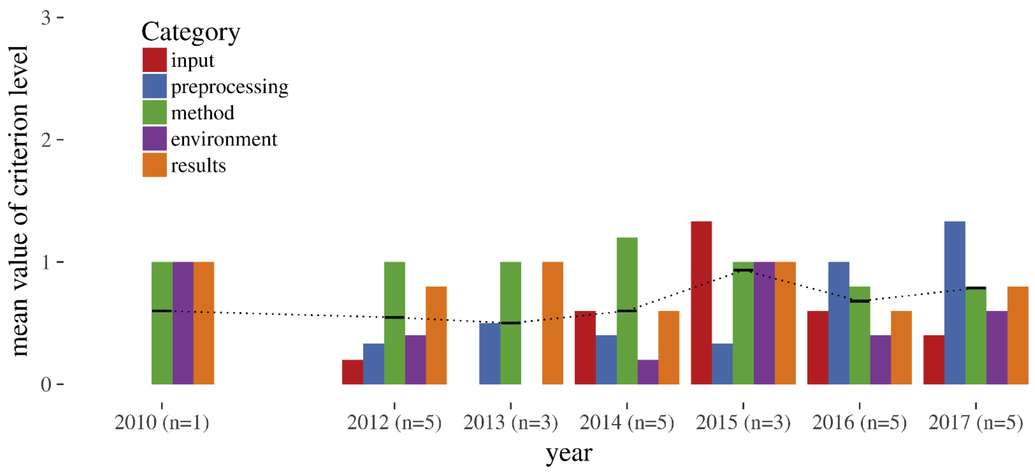


Figure 4 **Mean reproducibility levels per category over time; black dotted line connects the mean per year over all categories (in 2010 only one of three papers could be assessed, reaching level 1 for methods).**

Full-size DOI: 10.7717/peerj.5072/fig-4

Tables 4 and 5 contain summary statistics per criterion and means for full and short papers. For each criterion, full papers reach higher levels of reproducibility than short papers (see Table 5).

Author feedback (survey)

The full survey responses are included in this paper's repository (*Nüst, 2018*). The survey was sent to authors via e-mail and was open from 23 October to 24 November 2017. In case of obsolete e-mail addresses, we searched for updated ones and resent the form. Out

Table 4 Statistics of reproducibility levels per criterion.

	Input data	Preproc.	Method/analysis/proc.	Comp. env.	Results
Min.	0.00	0.00	0.00	0.00	0.00
Median	0.00	0.50	1.00	0.00	1.00
Mean	0.48	0.56	0.96	0.46	0.78
Max.	2.00	2.00	2.00	1.00	1.00
NA's	5.00	14.00	5.00	6.00	5.00

Table 5 Mean levels per criterion for full and short papers.

	Input data	preproc.	Method/analysis/proc.	Comp. env.	Results
Full papers	0.75	0.67	1.00	0.62	0.88
Short papers	0.09	0.33	0.91	0.20	0.64

of a total of 82 authors, 22 filled in the survey, resulting in responses for 17 papers, because six participants did not give consent to use their answers, two authors participated twice for different papers, and some papers had more than one individual response.

Authors were asked to comment on whether they agreed or disagreed with our evaluations of their specific paper. Seven responses fully agreed with the evaluation, five agreed partly, two expressed disagreement, and one did not answer the question. Most disagreements addressed the definition of criteria. Multiple authors argued that such requirements should not be applicable for short papers, and that specific data is not always necessary for reproducibility. Others disagreed about treating "availability upon request" as "unavailable". One argued that OpenStreetMap data is by default "open and permanent", but for our criteria citing OpenStreetMap lacked direct links to specific versioned subsets of data.

The answers suggest that authors are generally aware of the need for reproducibility and in principle know how to improve it in their work. However, many do not consider it a priority, saying that they did not incorporate reproducibility because of a lack of motivation (eight respondents) or the required extra effort required, which they say is disproportionately large in comparison to the added value.

According to the survey results, reproducibility was important to more than half of the respondents (see Fig. 5). Only two respondents claimed they were not at all concerned about it (both short papers). Further comments revealed that some authors consider short papers as introductions of new concepts and generally too short for reproducibility concerns. The paper corpus supports this opinion because short papers reach overall lower reproducibility levels.

In contrast, we argue that transparency should not depend on the publication type but is a feature of the entire scientific process. Especially at early stages, the potential for external review and collaboration can be beneficial for authors. Further, putting supplementary materials in online repositories addresses the problem of word count limits (for full and short papers), which many authors struggle with.

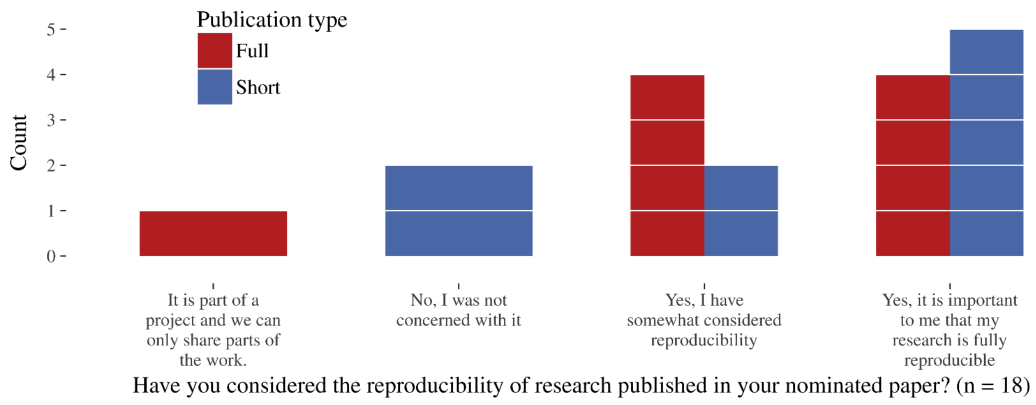


Figure 5 **Author survey results on the importance of reproducibility.**
Full-size DOI: 10.7717/peerj.5072/fig-5

To identify barriers to reproducibility, the authors were asked to rate how strongly five predefined barriers (Table 2) impacted their work's reproducibility. They could also add their own reasons, for which they mentioned paper length restrictions and the need for additional financial resources. Table 6 shows that the most frequently mentioned reasons were legal restrictions and lack of time, where only one respondent indicated that these factors played no role. Although lack of knowledge on how to include data, methods and results was not considered by many as a barrier, several respondents noted a lack of supporting tools as a main impediment for reproducibility.

Respondents also shared their ideas for how AGILE could encourage reproducibility in its publications. Four suggested Open Access publishing and asked for solutions to deal with sensitive data. A few suggested encouraging and promoting collaboration across research institutes and countries to mitigate ephemeral storage and organisations. Some respondents proposed that an award could be given for reproducible papers, reproducibility could be required for the best paper nomination, or conference fees could be waived for reproducible papers. In summary, almost all authors agreed on the importance of the topic and its relevance for AGILE.

DISCUSSION

A critical review of this paper's reproducibility

We acknowledge this paper has its own shortcomings with respect to reproducibility. The data, code, and a description of the runtime environment are transparently published on GitHub (https://github.com/nuest/reproducible-research-and-giscience) and deposited in an open repository under the DOI https://doi.org/10.5281/zenodo.1227260 (*Nüst, 2018*). The repository comprises an anonymised table with the survey results and a literate programming document, which transparently combines data preprocessing, analysis, and visualisations. The runtime environment description is based on Docker (https://en.wikipedia.org/wiki/Docker_(software)) and allows readers to easily open an

Table 6 **Hindering circumstances for reproducibility for each survey response ($n = 17$) sorted by barrier type for the category with most "Main reason" occurences; each line is one response and background colour corresponds to cell text.**

Legal restrictions	Lack of time	Lack of tools	Lack of knowledge	Lack of incentive
Main reason	Strongly hindered	Not at all	Not at all	Strongly hindered
Main reason	Not at all	Not at all	Not at all	Moderately hindered
Main reason	Slightly hindered	Strongly hindered	Moderately hindered	Strongly hindered
Main reason	Not at all	Slightly hindered	Not at all	Not at all
Strongly hindered	Strongly hindered	Strongly hindered	Moderately hindered	Strongly hindered
Moderately hindered	Main reason	Not at all	Not at all	Not at all
Slightly hindered	Moderately hindered	Slightly hindered	Slightly hindered	Moderately hindered
Slightly hindered	Not at all	Main reason	Strongly hindered	Not at all
Not at all	Moderately hindered	Not at all	Moderately hindered	Not at all
Not at all	Strongly hindered	Strongly hindered	Strongly hindered	Slightly hindered
Not at all	Moderately hindered	Not at all	Not at all	Not at all
Not at all	Slightly hindered	Main reason	Not at all	Strongly hindered
Not at all	Main reason	Not at all	Not at all	Not at all
Not at all	Main reason	Not at all	Not at all	Not at all
Not at all	Moderately hindered	Moderately hindered	Not at all	Strongly hindered
Not at all	Not at all	Not at all	Not at all	Not at all
Not at all	Slightly hindered	Not at all	Slightly hindered	Not at all

interactive analysis environment in their browser based on Binder (http://mybinder.org/, (cf. *Holdgraf, 2017*). The working link to launch the binder is https://mybinder.org/v2/gh/ nuest/reproducible-research-and-giscience/6 and the file `README.md` provides instructions on the usage. The *input data* (i.e., the paper corpus) for the text analysis cannot be re-published due to copyright restrictions. Our sample is biased (although probably positively), as we only considered award nominees. Access to all papers would have allowed a random sample from the population. Regarding the *method*, the created criteria and how they were assigned by humans cannot honour all details and variety of individual research contributions and is inherently subjective. We tried to mitigate this by applying a "four eyes" principle, and transparently sharing internal comments and discussion on the matter in the code repository. Using our own classification, we critically assign ourselves level `0` for data and level `3` for methods and results.

Improving day-to-day research in GIScience

Our evaluation clearly identifies issues of reproducibility in GIScience. Many of the evaluated papers use data and computer-based analysis. All papers were nominated for the best paper award within a double-blind peer review and thus represent the upper end of the quality spectrum at an established conference. Yet, overall reproducibility is low and no positive trend is perceivable. It seems that current practises in scientific publications lack full access to data and code. Instead, only methods and results are documented in writing.

In order to significantly improve the reproducibility of research, there must be changes in educational curricula, lab processes, universities, journal publishing, and funding agencies

(*Reproducible Research, 2010*; *McKiernan, 2017*) as well as reward mechanisms that go beyond paper citations (cf. term "altmetrics" in *Priem et al., 2010*). This is a major and long-term endeavour. Here, we focus on recommendations and suggestions for individual researchers and a specific organisation: AGILE. A snowball effect may lead to a change in practises in the GIScience community. The remainder of this paper addresses RQ 4 by formulating suggestions to researchers and the AGILE conference organisers.

Suggestions to authors

Regarding habits and workflows, the Carpentries (the union (http://www.datacarpentry.org/blog/merger/) of Data Carpentry (*Teal et al., 2015*) and Software Carpentry (*Wilson, 2006*)) offer lessons on tools to support research, such as programming and data management, across disciplines. Further resources are available from programming language and software communities, research domains, and online universities. Often these resources are available for free because the software is Free and Open Source Software (FOSS) and driven by a mixed community of users and developers. Ultimately, proprietary software is a deal-breaker for reproducibility (cf. *Ince, Hatton & Graham-Cumming, 2012*; *Baker, 2016b*). OSGeo-Live (https://live.osgeo.org/) provides a simple environment to test open alternatives from the geospatial domain, and several websites offer help in finding FOSS comparable to commercial products (e.g., https://opensource.com/alternatives or https://alternativeto.net). But, authors can do more than just use open software to improve reproducibility. It is not only about the software. They can engage in simple tasks such as "naming things" sensibly (https://speakerdeck.com/jennybc/how-to-name-files by Jennifer Bryan), they can be realistic by not striving for perfection but following "good enough practices in scientific computing" (*Wilson et al., 2017*), they can explore "selfish reasons to work reproducibly" (*Markowetz, 2015*), and they can follow FAIR[4] guidelines with "structuring supplemental material" (*Greenbaum et al., 2017*).

[4]Force11.org. Guiding principles for findable, accessible, interoperable and re-usable data publishing: version B1.0. https://www.force11.org/node/6062

Recommendations to conferences in GIScience and organisations like AGILE

What can conferences and scientific associations do to encourage reproducibility? A crucial step in improving reproducibility of GIScience research is acknowledging the important role organisations like AGILE can play in adopting reproducible research practises, which can be built upon a large body of guidelines, documentation and software. In the remainder of this section we propose concrete actions for organisations, using AGILE as the leading example.

AGILE could show that it recognizes and supports reproducibility by offering an **award for reproducible papers**. This is already done by other communities, e.g., the ACM SIGMOD 2017 Most Reproducible Paper Award (http://db-reproducibility.seas.harvard.edu/ and https://sigmod.org/2017-reproducibility-award/). At AGILE, when reviewers suggest submissions to be nominated for best (short) papers, , they could also have these papers briefly checked for reproducibility. This check could be performed by a new Scientific Reproducibility Committee led by a Reproducibility Chair, working alongside the existing committees and their chairs. Committee membership would be publicly recognised. The "most reproducible paper" could be prominently presented in the conference's closing session.

Kidwell et al. (2016) demonstrate that open data **badges** have had a positive effect on actual publishing of data in the journal *Psychological Science*, which uses badges and corresponding criteria from the Center for Open Science (https://osf.io/tvyxz/wiki/home/) (COS). Further examples are the "kite marks" used by the journal *Biostatistics* (Peng, 2011), the common standards and terms for artifacts used by the Association for Computing Machinery's (ACM) (https://www.acm.org/publications/policies/artifact-review-badging), and the Graphics Replicability Stamp Initiative (GRSI) (http://www.replicabilitystamp.org/). While AGILE could invent its own badges, re-using existing approaches has practical advantages (no need to design new badges), organisational advantages (no need to reinvent criteria), and marketing advantages (higher memorability). Author guidelines would include instructions on how to receive badges for a submission. The evaluation of badge criteria would be integrated in the review and could inform the reproducible paper award.

Author guidelines are the essential place to set the foundation for a reproducible conference (cf. SIGMOD 2018 CFP, https://sigmod2018.org/calls_papers_sigmod_research.shtml). Independently of advertising awards and badges, author guidelines should include clear guidelines on when, how, and where to publish supplemental material (data, code). *Author guidelines for computational research* must make authors aware to highlight reproducibility-related information for reviewers and readers. These guidelines should contain practical advice, such as code and data licenses[5], and instructions on how to work reproducibly, such as by providing a space for sharing tools and data, which is the most popular suggestion from the survey (seven respondents).

While the established peer-review process works well for conceptual papers, a special **track or submission type**[6] could accommodate submissions focussing on reproducibility without an original scientific contribution and an adapted process (e.g., public peer review). Such publications can include different authors, e.g., technical staff, or even reviewers as practised by Elsevier's *Information Systems* journal. Publications in a special track can also mitigate limitations on research paper lengths. Unfortunately, they can also convey the counterproductive message of reproducibility being cumbersome and uncommon.

Submissions through this special track as well as the regular conference proceedings should be published as **Open Access** (see https://open-access.net/DE-EN/information-on-open-access/open-access-strategies/) content in the future. It might even be possible to re-publish short papers and abstracts of previous conferences after solving juridical concerns (e.g., if author consent is required). To do this, AGILE could utilise existing repositories or operate its own, where using third party repositories[7] for supplements would reduce the burden on the AGILE organisation. Choosing **one repository** allows for collecting all AGILE submissions under one tag or community (cf. http://help.osf.io/m/sharing/l/524053-tags and https://zenodo.org/communities/). An AGILE-specific repository would allow more control, but would require more work and might have lower visibility, since the large repositories are well indexed by search engines. Both approaches would support a double-blind review by providing anonymous view-only copies of supplemental material (see http://help.osf.io/m/links_forks/l/524049-create-a-view-only-link-for-a-project).

[5]E.g., OSI compliant for code and Open Definition compliant for data, see http://licenses.opendefinition.org/.

[6]See IEEE's CiSE magazine's Reproducible Research Track https://www.computer.org/cise/2017/07/26/reproducible-research-track-call-for-papers/, and Elsevier journal Information Systems' section for invited reproducibility papers, https://www.elsevier.com/journals/information-systems/0306-4379/guide-for-authors.

[7]Beside the incumbents Figshare (https://figshare.com/), Open Science Framework (OSF) (https://osf.io/, community-driven) and Zenodo (https://zenodo.org/, potentially preferable given AGILE's origin because it is funded by EU), a large number of Open Access repositories exists, see http://roar.eprints.org/ and http://opendoar.org/, including platforms by publishers, e.g., Springer (https://www.springer.com/gp/open-access), or independent organisations, e.g., LIPIcs proceedings (https://www.dagstuhl.de/en/publications/lipics)

We see AGILE, carried by its member labs and mission (https://agile-online.org/index.php/about-agile), as being in a unique position among GIScience conferences to establish a common understanding and practise of reproducible research. Firstly, member labs can influence education, especially at the graduate level, and ideally collaborate on **open educational material**. Completing a Ph.D. in an AGILE member lab and participating in AGILE conferences should qualify early career scientists to publish and review reproducible scholarly works. Secondly, the conference can take a leading role in setting up new norms for conference review and publication but at the same time cooperate with other conferences (e.g., ACM SIGMOD). At first AGILE would encourage but eventually demand the highest level of reproducibility for all submissions. This process certainly will take several years to complete.

CONCLUSIONS

What skills related to reproducibility are desirable for authors at GIScience conferences in 2028? Predicting 10 years ahead might not be scientific, but it allows for formulating a vision to conclude this work. We assume that in 10 years, hardly any paper will not utilise digital methods, such as software for analysis, interactive visualisations, or open data. Ever more academics will face competitive selection processes, where quality of research will be measured by its transparency and novelty. To achieve novelty in a setting where all research is saved, findable and potentially interpreted by artificial intelligence (*Jones, 2016*), a new contribution must be traceable. Thus, the trend towards Open Science will be reinforced until it is standard practise to use and publish open source code and open data as well as to incorporate alternative metrics beyond citations. As of now, AGILE is not ready for such research. It has identifiers (DOIs) only for full publications and lacks open licenses for posters and (short) papers. Statements on preprints (publication before submission) and postprints ("green" Open Access, see https://open-access.net/DE-EN/information-on-open-access/open-access-strategies/) are missing.

Researchers, conference organisers, and programme committees will have to leave their comfort zone and change the way they work. Also, in order to overcome old habits, they will have to immediately see the benefits of the new ways (*Wilson et al., 2017*). The evidence for benefits of Open Science are strong (*McKiernan et al., 2016*), but to succeed, the community must embrace the idea of a reproducible conference. We acknowledge that fully reproducible GIScience papers are no small step for both authors and reviewers, but making them the standard would certainly be a giant leap for GIScience conferences. We are convinced a conference like AGILE can provide the required critical mass and openness, and we hope the experiences and information provided in this work represent a sound starting point.

ACKNOWLEDGEMENTS

We thank all authors who participated in the survey and the reviewers for their detailed comments and valuable suggestions for the manuscript. We would like to thank Celeste R.

Brennecka, from the Science Writing Support Service of the University of Münster, for her editorial support.

ADDITIONAL INFORMATION AND DECLARATIONS

Funding

Carlos Granell is funded by the Ramón y Cajal Programme of the Spanish government (grant number RYC-2014-16913). Daniel Nüst and Markus Konkol are supported by the project Opening Reproducible Research (https://www.uni-muenster.de/forschungaz/project/9520) funded by the German Research Foundation (DFG) under project numbers PE 1632/10-1 respectively KR 3930/3-1. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Grant Disclosures

The following grant information was disclosed by the authors:
Ramón y Cajal Programme of the Spanish government: RYC-2014-16913.
German Research Foundation (DFG): PE 1632/10-1, KR 3930/3-1.

Competing Interests

Barbara Hofer is a member of the AGILE council
(https://agile-online.org/index.php/community/council).

Author Contributions

- Daniel Nüst, Barbara Hofer and Rusne Sileryte analyzed the data, contributed reagents/materials/analysis tools, prepared figures and/or tables, authored or reviewed drafts of the paper, approved the final draft.
- Carlos Granell analyzed the data, prepared figures and/or tables, authored or reviewed drafts of the paper, approved the final draft.
- Markus Konkol and Frank O. Ostermann analyzed the data, authored or reviewed drafts of the paper, approved the final draft.
- Valentina Cerutti analyzed the data, approved the final draft.

Data Availability

The following information was supplied regarding data availability:

GitHub: https://github.com/nuest/reproducible-research-and-giscience/
Zenodo: https://doi.org/10.5281/zenodo.1227260.

Supplemental Information

Supplemental information for this article can be found online at http://dx.doi.org/10.7717/peerj.5072#supplemental-information.

REFERENCES

Baker M. 2016a. 1,500 scientists lift the lid on reproducibility. *Nature News* **533(7604)**:452–454 DOI 10.1038/533452a.

Baker M. 2016b. Why scientists must share their research code. *Nature News* DOI 10.1038/nature.2016.20504.

Baker M. 2017. Reproducibility: check your chemistry. *Nature* **548(7668)**:485–488 DOI 10.1038/548485a.

Barba LA. 2016. The hard road to reproducibility. *Science* **354(6308)**:142–142 DOI 10.1126/science.354.6308.142.

Bechhofer S, Buchan I, De Roure D, Missier P, Ainsworth J, Bhagat J, Couch P, Cruickshank D, Delderfield M, Dunlop I, Gamble M, Michaelides D, Owen S, Newman D, Sufi S, Goble C. 2013. Why linked data is not enough for scientists. *Future Generation Computer Systems* **29(2)**:599–611 DOI 10.1016/j.future.2011.08.004.

Brunsdon C. 2016. Quantitative methods I: reproducible research and quantitative geography. *Progress in Human Geography* **40(5)**:687–696 DOI 10.1177/0309132515599625.

Button KS, Ioannidis JPA, Mokrysz C, Nosek BA, Flint J, Robinson ESJ, Munafò MR. 2013. Power failure: why small sample size undermines the reliability of neuroscience. *Nature Reviews Neuroscience* **14(5)**:365–376 DOI 10.1038/nrn3475.

Collberg C, Proebsting TA. 2016. Repeatability in computer systems research. *Communications of the ACM* **59(3)**:62–69 DOI 10.1145/2812803.

Editorial. 2016. Reality check on reproducibility. *Nature* **533(7604)**:437 DOI 10.1038/533437a.

Ferreira C, Bastille-Rousseau G, Bennett AM, Ellington EH, Terwissen C, Austin C, Borlestean A, Boudreau MR, Chan K, Forsythe A, Hossie TJ, Landolt K, Longhi J, Otis J-A, Peers MJL, Rae J, Seguin J, Watt C, Wehtje M, Murray DL. 2016. The evolution of peer review as a basis for scientific publication: directional selection towards a robust discipline? *Biological Reviews* **91(3)**:597–610 DOI 10.1111/brv.12185.

Gentleman R, Lang DT. 2007. Statistical analyses and reproducible research. *Journal of Computational and Graphical Statistics* **16(1)**:1–23 DOI 10.1198/106186007X178663.

Gewin V. 2016. Data sharing: An open mind on open data. *Nature* **529(7584)**:117–119 DOI 10.1038/nj7584-117a.

Gil Y, David CH, Demir I, Essawy BT, Fulweiler RW, Goodall JL, Karlstrom L, Lee H, Mills HJ, Oh J-H, Pierce SA, Pope A, Tzeng MW, Villamizar SR, Yu X. 2016. Toward the geoscience paper of the future: best practices for documenting and sharing research from data to software to provenance. *Earth and Space Science* **3(10)**:388–415 DOI 10.1002/2015EA000136.

Giraud T, Lambert N. 2017. Reproducible cartography. In: *Advances in cartography and GIScience. Lecture notes in geoinformation and cartography*. Springer, Cham, 173–183.

Greenbaum D, Rozowsky J, Stodden V, Gerstein M. 2017. Structuring supplemental materials in support of reproducibility. *Genome Biology* **18**:Article 64 DOI 10.1186/s13059-017-1205-3.

Gronenschild EHBM, Habets P, Jacobs HIL, Mengelers R, Rozendaal N, Os JV, Marcelis M. 2012. The effects of freesurfer version, workstation type, and macintosh operating system version on anatomical volume and cortical thickness measurements. *PLOS ONE* **7(6)**:e38234 DOI 10.1371/journal.pone.0038234.

Holdgraf C. 2017. Binder 2.0, a Tech Guide. *Jupyter Blog. Available at https://blog.jupyter.org/binder-2-0-a-tech-guide-2017-fd40515a3a84* (accessed on 24 April 2018).

Hothorn T, Leisch F. 2011. Case studies in reproducibility. *Briefings in Bioinformatics* **12(3)**:288–300 DOI 10.1093/bib/bbq084.

Ince DC, Hatton L, Graham-Cumming J. 2012. The case for open computer programs. *Nature* **482**:485–488 DOI 10.1038/nature10836.

Ioannidis JPA. 2005. Why most published research findings are false. *PLOS Medicine* **2(8)**:e124 DOI 10.1371/journal.pmed.0020124.

Ioannidis JPA. 2014. How to make more published research true. *PLOS Medicine* **11(10)**:e1001747 DOI 10.1371/journal.pmed.1001747.

Ioannidis JPA, Stanley TD, Doucouliagos H. 2017. The power of bias in economics research. *The Economic Journal* **127(605)**:F236–F265 DOI 10.1111/ecoj.12461.

Jones N. 2016. AI science search engines expand their reach. *Nature News* DOI 10.1038/nature.2016.20964.

Kidwell MC, Lazarević LB, Baranski E, Hardwicke TE, Piechowski S, Falkenberg L-S, Kennett C, Slowik A, Sonnleitner C, Hess-Holden C, Errington TM, Fiedler S, Nosek BA. 2016. Badges to acknowledge open practices: a simple, low-cost, effective method for increasing transparency. *PLOS Biology* **14(5)**:e1002456 DOI 10.1371/journal.pbio.1002456.

Leek JT, Peng RD. 2015. Opinion: reproducible research can still be wrong: adopting a prevention approach: Fig. 1. *Proceedings of the National Academy of Sciences* **112(6)**:1645–1646 DOI 10.1073/pnas.1421412111.

Markowetz F. 2015. Five selfish reasons to work reproducibly. *Genome Biology* **16**:Article 274 DOI 10.1186/s13059-015-0850-7.

McKiernan EC. 2017. Imagining the "open" university: sharing scholarship to improve research and education. *PLOS Biology* **15(10)**:e1002614 DOI 10.1371/journal.pbio.1002614.

McKiernan EC, Bourne PE, Brown CT, Buck S, Kenall A, Lin J, McDougall D, Nosek BA, Ram K, Soderberg CK, Spies JR, Thaney K, Updegrove A, Woo KH, Yarkoni T. 2016. Point of view: how open science helps researchers succeed. *eLife* **5**:e16800 DOI 10.7554/eLife.16800.

McNutt M. 2014. Journals unite for reproducibility. *Science* **346(6210)**:679–679 DOI 10.1126/science.aaa1724.

Nosek BA, Alter G, Banks GC, Borsboom D, Bowman SD, Breckler SJ, Buck S, Chambers CD, Chin G, Christensen G, Contestabile M, Dafoe A, Eich E, Freese J, Glennerster R, Goroff D, Green DP, Hesse B, Humphreys M, Ishiyama J, Karlan D, Kraut A, Lupia A, Mabry P, Madon T, Malhotra N, Mayo-Wilson E, McNutt M, Miguel E, Paluck EL, Simonsohn U, Soderberg C, Spellman BA, Turitto J, Vanden-Bos G, Vazire S, Wagenmakers EJ, Wilson R, Yarkoni T. 2015. Promoting an open research culture. *Science* **348(6242)**:1422–1425 DOI 10.1126/science.aab2374.

Nüst D. 2018. Reproducibility Package for "Reproducible research and GIScience: an evaluation using AGILE conference papers". *Available at https://doi.org/10.5281/zenodo.1227260*.

Nüst D, Konkol M, Pebesma E, Kray C, Schutzeichel M, Przibytzin H, Lorenz J. 2017. Opening the publication process with executable research compendia. *D-Lib Magazine* **23(1/2)** DOI 10.1045/january2017-nuest.

Ostermann FO, Granell C. 2017. Advancing science with VGI: reproducibility and replicability of recent studies using VGI. *Transactions in GIS* **21(2)**:224–237 DOI 10.1111/tgis.12195.

Peng RD. 2011. Reproducible research in computational science. *Science* **334(6060)**: 1226–1227 DOI 10.1126/science.1213847.

Priem J, Taraborelli D, Groth P, Neylon C. 2010. altmetrics: a manifesto—altmetrics.org. *Available at http://altmetrics.org/manifesto/* (accessed on 21 November 2017).

Pundt H, Toppen F. 2017. 20 years of AGILE. In: Bregt A, Sarjakoski T, van Lammeren R, Rip F, eds. *Societal geo-innovation*. Cham: Springer International Publishing 351–367 DOI 10.1007/978-3-319-56759-4_20.

Reichman OJ, Jones MB, Schildhauer MP. 2011. Challenges and opportunities of open data in ecology. *Science* **331(6018)**:703–705 DOI 10.1126/science.1197962.

Reproducible Research. 2010. *Computing in Science Engineering* **12(5)**:8–13 DOI 10.1109/MCSE.2010.113.

Sandve GK, Nekrutenko A, Taylor J, Hovig E. 2013. Ten simple rules for reproducible computational research. *PLOS Computational Biology* **9(10)**:e1003285 DOI 10.1371/journal.pcbi.1003285.

Scheider S, Ostermann FO, Adams B. 2017. Why good data analysts need to be critical synthesists. Determining the role of semantics in data analysis. *Future Generation Computer Systems* **72(Supplement C)**:11–22 DOI 10.1016/j.future.2017.02.046.

Steiniger S, Hay GJ. 2009. Free and open source geographic information tools for landscape ecology. *Ecological Informatics* **4(4)**:183–195 DOI 10.1016/j.ecoinf.2009.07.004.

Stodden V, McNutt M, Bailey DH, Deelman E, Gil Y, Hanson B, Heroux MA, Ioannidis JPA, Taufer M. 2016. Enhancing reproducibility for computational methods. *Science* **354(6317)**:1240–1241 DOI 10.1126/science.aah6168.

Stodden V, Miguez S. 2014. Best practices for computational science: software infrastructure and environments for reproducible and extensible research. *Journal of Open Research Software* **2(1)**:1–6 DOI 10.5334/jors.ay.

Teal TK, Cranston KA, Lapp H, White E, Wilson G, Ram K, Pawlik A. 2015. Data carpentry: workshops to increase data literacy for researchers. *International Journal of Digital Curation* **10(1)**:135–143 DOI 10.2218/ijdc.v10i1.351.

Tenopir C, Allard S, Douglass K, Aydinoglu AU, Wu L, Read E, Manoff M, Frame M. 2011. Data sharing by scientists: practices and perceptions. *PLOS ONE* **6(6)**:e21101 DOI 10.1371/journal.pone.0021101.

Wilson G. 2006. Software carpentry: getting scientists to write better code by making them more productive. *Computing in Science Engineering* **8(6)**:66–69 DOI 10.1109/MCSE.2006.122.

Wilson G, Bryan J, Cranston K, Kitzes J, Nederbragt L, Teal TK. 2017. Good enough practices in scientific computing. *PLOS Computational Biology* **13(6)**:e1005510 DOI 10.1371/journal.pcbi.1005510.

12 Reproducible research and GIScience: An evaluation using GIScience conference papers

Authors & contribution Frank O. Ostermann, Daniel Nüst (25%), Carlos Granell, Barbara Hofer, Markus Konkol

Venue GIScience Conference 2021 (in press); EarthArXiv preprint 10.31223/X5ZK5V; included text is latest version from repository

Date 03/2021

Repository https://github.com/nuest/reproducible-research-at-giscience

ERC https://o2r.uni-muenster.de/erc/zOcwh

Reproducible Research and GIScience: an evaluation using GIScience conference papers

Frank O. Ostermann [1]
Faculty of Geo-Information Science and Earth Observation (ITC), University of Twente, Enschede, The Netherlands
f.o.ostermann@utwente.nl

Daniel Nüst
Institute for Geoinformatics, University of Münster, Münster, Germany
daniel.nuest@uni-muenster.de

Carlos Granell
Institute of New Imaging Technologies, Universitat Jaume I de Castellón, Castellón, Spain
carlos.granell@uji.es

Barbara Hofer
Christian Doppler Laboratory GEOHUM and Department of Geoinformatics - Z_GIS, University of Salzburg, Salzburg, Austria
barbara.hofer@sbg.ac.at

Markus Konkol
Faculty of Geo-Information Science and Earth Observation (ITC), University of Twente, Enschede, The Netherlands
m.konkol@utwente.nl

Abstract

GIScience conference authors and researchers face the same computational reproducibility challenges as authors and researchers from other disciplines who use computers to analyse data. Here, to assess the reproducibility of GIScience research, we apply a rubric for assessing the reproducibility of 75 conference papers published at the GIScience conference series in the years 2012-2018. Since the rubric and process were previously applied to the publications of the AGILE conference series, this paper itself is an attempt to replicate that analysis, however going beyond the previous work by evaluating and discussing proposed measures to improve reproducibility in the specific context of the GIScience conference series. The results of the GIScience paper assessment are in line with previous findings: although descriptions of workflows and the inclusion of the data and software suffice to explain the presented work, in most published papers they do not allow a third party to reproduce the results and findings with a reasonable effort. We summarise and adapt previous recommendations for improving this situation and propose the GIScience community to start a broad discussion on the reusability, quality, and openness of its research. Further, we critically reflect on the process of assessing paper reproducibility, and provide suggestions for improving future assessments. The code and data for this article are published at `https://doi.org/10.5281/zenodo.4032875`.

2012 ACM Subject Classification Information systems~Geographic information systems

Keywords and phrases reproducible research, open science, reproducibility, GIScience

Supplement Material The input data for this work are the full texts of GIScience conference proceedings from the years 2012 to 2018 [35, 7, 20, 34]. The paper assessment results and source code of figures are published at https://github.com/nuest/reproducible-research-at-giscience and archived on Zenodo [27]. The used computing environment is containerised with Docker pinning the R version to 3.6.3 and R packages to the MRAN snapshot of July 5th 2019.

[1] Corresponding author

Funding *Daniel Nüst* : Project *o2r*, German Research Foundation, grant number PE 1632/17-1.
Carlos Granell : Ramón y Cajal Programme of the Spanish government, grant number RYC201416913.
Markus Konkol : Project *o2r*, German Research Foundation, grant numbers KR 3930/8-1 and TR 864/12-1.

Acknowledgements Author contributions (see CRediT): *all authors* contributed to conceptualisation, investigation (number of assessed papers in brackets), and writing – original draft; FO (33): writing - review & editing, software; DN (33): software, writing - review & editing, visualisation; CG (30): writing - review & editing, software; BH (21): writing - review & editing; MK (30). We thank Celeste R. Brennecka from the Scientific Editing Service of the University of Münster for her editorial support and the anonymous reviewers for their constructive feedback.

1 Introduction

The past two decades have seen the imperative of Open Science gain momentum across scientific disciplines. The adoption of Open Science practices is partially prompted by the increasing costs of using proprietary software and subscribing to scientific journals, but more importantly because of the increased transparency and availability of data, methods, and results, which enable reproducibility [22]. This advantage is especially relevant for the computational and natural sciences, where sharing data and code is a prerequisite for reuse and collaboration. A large proportion of GIScience research today uses software to analyse data on computers, meaning that many articles published in the context of the GIScience conference series[2] fall into the categories of data science or computational research. Thereby, these articles face challenges of transparency and reproducibility in the sense of the Claerbout/Donoho/Peng terminology [2], where *reproduction* means a recreation of the same results using the same input data and methods, usually with the actual code created by the original authors. The related concept of replication, i.e., the confirmation of insights gained from a scientific study using the same method with new data, is of crucial importance to scientific progress, yet it is also frequently challenging to realise for interested readers of a published study. So far, despite the GIScience conference series' rigorous review process, reproducibility and replicability have not been a core concern in the contributions. With reproducibility now being a recognised topic in the call for papers, it is time to take stock and identify possible action. In previous work [26], we assessed the reproducibility of a selection of full and short papers from the AGILE conference series[3], a community conference organised by member labs of the Association of Geographic Information Laboratories in Europe (AGILE). Using systematic analysis based on a rubric for reproducible research, we found that the majority of AGILE papers neither provided sufficient information for a reviewer to evaluate the code and data and attempt a reproduction, nor enough material for readers to reuse or extend data or code from the analytical workflows. This is corroborated by research in related disciplines such as quantitative geography [3], qualitative GIS [21], geoscience [16], and e-Science [10]. The problems identified in these related research areas are transferable to the scientific discipline of GIScience, which operates at the intersections of aforementioned fields [11]. In any case, observations on the lack of reproducibility in all scientific fields contrast with the clear advantages and benefits of open and reproducible research both for individuals and for academia as a whole (cf. for example [6, 19, 17, 5]). As a consequence, we have initiated a process to support authors in increasing reproducibility

[2] https://www.giscience.org/
[3] https://agile-online.org/conference

for AGILE publications; as a main outcome, this initiative has produced author guidelines as well as strategies for the AGILE conference series[4].

The AGILE conference is related to GIScience conference in terms of scientific domain and contributing authors, but is different in organisational aspects. Two open questions are thus whether the GIScience conference series faces the same issues, and whether similar strategies could be applied successfully. To begin this investigation, we conducted a simple text analysis of GIScience conference proceedings[5] to evaluate the relevance of computational methods in the conference papers. The analysis searched for several word stems related to reproducibility: Generic words indicating a quantitative analysis, e.g., "data", "software", or "process"; specific platforms, e.g., "GitHub"; and concrete terms, e.g., words starting with "reproduc" or "replic". Table 1 shows the results of the search for each year analysed. The take-away message from the text analysis is that algorithms, processing, and data play an essential role in GIScience publications, but few papers mentioned code repositories or reproduction materials. Therefore, an in-depth assessment of the reproducibility of these publications was deemed necessary.

The main contribution of this work addresses two objectives: First, it aims to investigate the state of reproducibility in the GIScience conference community. This investigation broadens our knowledge base about reproducibility in the GIScience discipline and informs us about the situation in the GIScience conference series specifically (details in section 4). Second, it aims to apply the assessment procedure used for AGILE conference papers (presented in section 3) to the papers of the GIScience conference, so that the broader suitability of this procedure is evaluated using a different dataset, and thereby providing evidence of its replicability. Such a transfer validates the developed methodology. We discuss these findings and present our conclusions in the final two sections (5 and 6). Together, these objectives yield important findings for the discussion of reproducibility within the GIScience conference community and the GIScience discipline at large. We believe that GIScience as a discipline would greatly benefit from more studies that reproduce and replicate other studies, similar to other disciplines that are recognising the value of replication for innovating theory [23], and argue that such a replication study is not lacking innovation but is a prerequisite for innovating community practice. Only then can a fruitful dialogue take place on whether and how to improve reproducibility for the GIScience conference series, and whether the recent steps taken at AGILE[6] could be an inspiration for GIScience conferences as well.

2 Related work

This work builds and expands on earlier work [26], which already provides an overview of reproducible research in general, including definitions, challenges, and shortcomings. In the following, we focus therefore on recently published works and briefly introduce related

[4] See the initiative website at `https://reproducible-agile.github.io/`, the author guidelines at `https://doi.org/10.17605/OSF.IO/CB7Z8` [24] and the main OSF project with all materials `https://osf.io/phmce/` [25].

[5] The full text analysis and the results is available in this paper's repository in the following files: `giscience-historic-text-analysis.Rmd` contains the analysis code; the result data are two tables with counts for occurrences of words respectively word stems per year in `results/text_analysis_topwordstems.csv` and `results/text_analysis_keywordstems.csv`; a word-cloud per year is in file `results/text_analysis_wordstemclouds.png`.

[6] See the initiative website at `https://reproducible-agile.github.io/`, the author guidelines at `https://doi.org/10.17605/OSF.IO/CB7Z8` [24] and the main OSF project with all materials `https://osf.io/phmce/` [25].

Table 1 Reproducibility-related word stems in the corpus per year of proceedings

year	words	reproduc..	replic..	repeatab..	code	software	algorithm(s)	(pre)process..	data.*	result(s)	repository/ies	github/lab
2002	23782	6	2	0	11	61	191	150	897	129	62	0
2004	26728	4	1	0	34	50	138	258	849	263	4	0
2006	32758	6	0	0	12	32	335	250	856	164	0	0
2008	27356	3	6	1	3	11	331	146	854	218	17	0
2010	23004	3	1	0	8	16	164	276	650	162	0	0
2012	28860	2	0	0	101	27	238	190	1048	311	3	0
2014	29534	3	4	1	12	18	255	159	1070	228	3	0
2016	24838	2	0	0	23	21	333	150	1007	202	4	1
2018	23318	3	10	0	15	15	201	160	891	294	6	6
Total	240178	32	24	2	219	251	2186	1739	8122	1971	99	7

Note:
The very high value for 'code' in 2012 is due to a single paper about land use, for which different "land use codes" are defined, discussed and used.

meta-studies.

Few groups have attempted practical reproduction of computational works related to GIScience. Konkol et al. [16] conducted an in-depth examination of the computational reproducibility of 41 geoscience papers with a focus on differences between the recreated figures. The set of papers was, similar to our work, drawn from a fixed group of two outlets (journals), but it was further limited to recent papers providing code in the R language. The main issues raised by Konkol et al. [16] are similar to those identified in a recent report on the reproducibility review during the AGILE conference 2020[7], where the reproducibility committee summarised the process and documented relevant obstacles to reproducibility of accepted papers.

Within the geospatial domain, Kedron et al. [13] provide a recent review of opportunities and challenges for reproducibility and replicability. They transfer solutions from other domains but also discuss and conceptualise the specific nature of a reproducibility and replicability framework when working with geospatial data, e.g., handling context, uncertainty of spatial processes, or how to accommodate the inherent natural variability of geospatial systems. In a similar manner, Brunsdon and Comber [4] investigate reproducibility within spatial data science, with special attention to big spatial data. They support the need for open tools, knowledge about code, and reproducibility editors at domain journals and conferences, but they also introduce the perspective that spatial analysis is no longer conducted only by GI/geo-scientists or geographers and connect reproducibility with critical spatial understanding. The more conceptual work in those articles is complemented by the assessment of reproducibility conducted in this paper.

Two recent studies from distant disciplines, wildlife science [1] and hydrology [32], also relate to our work in this paper. Both studies investigate a random set of articles from selected journals and use a stepwise process of questions to determine the availability of materials and eventually reproduce workflows if possible. Archmiller et al. [1] use a final ranking of 1 to 5 to specify the degree to which a study's conclusions were eventually reproduced. Similar to our classification scheme, their ranking models fit the general notion of a *"reproducibility spectrum"* [30].

3 Reproducibility assessment method

3.1 Criteria

The assessment criteria used for the current study were originally defined in previous work, so we provide only a short introduction here and refer to Nüst et al. [26] for details. The three assessment criteria are *Input Data*, *Methods*, and *Results*. *Input Data* comprises all

[7] `https://osf.io/7rjpe/`

datasets that the computational analysis uses. *Methods* encompasses the entire computational analysis that generates the results. Since *Methods* is difficult to evaluate as a whole, we split this criterion into three subcriteria: *Preprocessing* includes the steps to prepare the *Input Data* before the main analysis; *Methods, Analysis, Processing* is the main analysis; *Computational Environment* addresses the description of hard- and software. Finally, the criterion *Results* refers to the output of analysis, e.g., figures, tables, and numbers.

For each of these (sub)criteria, we assigned one of four levels unless the criterion was not applicable (*NA*). *Unavailable* (level `0`) means that it was not possible to access the paper's data, methods, or results, and that it was impossible to recreate them based on the description in the paper. *Documented* (level `1`) indicates that the paper still did not provide direct access to datasets, methods, or results, but that there was sufficient description or metadata to potentially recreate them closely enough for an evaluation; yet, often a recreation was unlikely due to the huge amount of effort needed. For example, with regard to the methods criteria, *Documented* means that pseudo code or a textual workflow description was provided. *Available* (level `2`) was assigned if the paper provided direct access to the materials (e.g., through a link to a personal or institutional website), but not in the form of an open and permanent identifier, such as a digital object identifier (DOI). The indication of a DOI does not apply to the methods criteria, as it is not yet common practice to make a permanent reference to code, libraries, and system environments with a single identifier. The gold standard, *Available and Open* (level 3), requires open and permanent access to the materials (e.g., through public online repositories) and open licenses to allow use and extension.

Note that levels are ordinal numbers that can be compared (`3` is higher than `2`), but absolute differences between numbers must not be interpreted as equals: Moving one level up from `0` to `1` is not the same as from level 1 to level `2`. While reaching level `1` is fairly straightforward, moving to level `2` means one must create a fully reproducible paper.

3.2 Process

The overall approach to assessing the reproducibility of GIScience papers followed the previous assessment of AGILE papers [26], and was conducted by the same persons. Contrary to the AGILE investigation, all full papers in the GIScience conference series (from the 2012 to 2018 editions) were assessed. This is partly because no obvious subset exists, such as the nominees for best papers as in the case of the AGILE conference series, but also because we aimed to work with a larger dataset for potentially more informative results. Each GIScience conference paper was randomly assigned to two assessors who evaluated it qualitatively according to the reproducibility criteria. The assessors were free in the way they approached the assigned evaluations, depending on the structure of the paper and the assessor's familiarity with the topic. An evaluation could range from browsing the paper to identify relevant statements in case of high familiarity to a thorough reading of the full text. The identification of relevant content could be supported to some extent by a PDF reader with multiple highlights, using keywords like e.g., "data, software, code, download, contribution, script, workflow". The results of the individual assessments were joined in a collaborative Google Spreadsheet. This spreadsheet also had a comments column for assessors to record relevant sources and decisions. In case of disagreement between assessors, arguments for and against a certain reproducibility level were discussed in the entire group of five assessors until a consensus was reached. Only then were the assessments merged into a single value. A snapshot of both the unmerged and merged values was stored as a CSV file in the collaboration repository for transparency and provenance[8]. Two independent assessors per paper increased the objectivity of the final assessment. Disagreements and conducting the assessment one year at a time, going backwards from the most recent year, were found helpful in aligning the interpretation of criteria and, in rare cases, led to an adjustment of similar cases in other papers.

The discussion about the correct assignment of levels led to a reflection on how to apply the rubric for special situations. For the *Input Data* criterion, some papers had input data "available" at the time of writing/publication that was not available anymore at the time of evaluation, due to broken links, changes in the URL structure of a website, or projects and/or personal websites that were down or moved. In such cases, we gave the authors the benefit of the doubt and assumed the data were accessible some time after the publication of the conference proceedings. We did not give those papers an arbitrary score and discussed internally the best level per case; yet, such papers never earned a `3`, which would require permanent resolving of the link. Related to this criterion, simulation data, like the specification or configuration of agents in an agent-based system, was not treated as input data (resulting in *NA* if no other data was used), but as parameters of the main analysis, i.e., as part of the *Methods, Analysis, Processing.*

Preprocessing covers preparatory work for the actual analysis involving various tasks such as data selection, cleaning, aggregation, and integration. However, the dividing line between data preprocessing and processing (i.e., the main analysis) proved to be often vague, and occasionally assessors disagreed whether the preprocessing criterion should be assigned *NA*, *Unavailable*, or *Documented* (`0` or `1`, respectively). Therefore, we decided eventually to apply the *Preprocessing* criterion only in cases where papers specifically mentioned a preprocessing task independent of the actual analysis or method, e.g., when clearly stated in a separate sub-section of the paper.

Lastly, human subject tests and surveys were also a special case. Human-related research activities were rated as `1` in the methods/analysis/processing criterion if sufficiently documented; nonetheless, a sufficient documentation in these cases did not mean that original sources were available or could be exactly recreated.

3.3 Paper corpus

In total, 87 papers from the GIScience conferences in 2012, 2014, 2016, and 2018 were assessed. A table in the reproducibility package shows the full results of the assessment and the included raw data provides details on assigned assessors, authors, etc. [27]. 12 papers (14%) across all years were identified as conceptual papers[9] and were not included in the corpus. The number of conceptual papers in GIScience conferences was low over the analysed years (2012: 4; 2014: 5; 2016: 3), and none in 2018. This might suggest an increasingly predominant and ubiquitous role of analytical datasets and computational workflows in the generation of the final published results in the field.

[8] The assessment results are in the file `results/paper_assessment.csv`. As an example, commit `464e630` and `2e8b1be` are the pre-merge and post-merge commit after completing the assessment of the papers from 2014. The pre-merge commit contains the assessments including the assessors' initials, e.g. "CG: 1, MK: 1".

[9] See [26] for a definition of "conceptual".

Table 2 Statistics of reproducibility levels per criterion (rounded to one decimal place)

	input data	**preproc.**	**method/analysis/proc.**	**comp. env.**	**results**
Min.	0.0	0.0	0	0.0	0.0
Median	1.0	1.0	1	0.0	1.0
Mean	0.7	0.8	1	0.3	1.1
Max.	2.0	2.0	2	1.0	2.0
NA's	1.0	24.0	0	0.0	0.0

4 Reproducibility of GIScience conference papers

Table 2 shows aggregated values for the assessed reproducibility levels. If we look at the median values of the five criteria (Table 2), a typical GIScience paper scores `1 1 1 0 1`. This score translates in practical terms into a paper that is sufficiently documented to claim that reproduction could be attempted within a reasonable time frame after publication. While such a level of reproducibility is typically accepted by journals and conferences today, it does not guarantee that a reproduction would be possible and practical. A reproduction of such a paper would require considerable effort, namely technical skills, communication with authors, and time not only to both gather, recreate, and/or analyse all the necessary resources (data, code, etc.) but also to recreate the specific computational environment of the paper. Especially the latter is very unlikely, as the computational environment is generally not specified at all, as demonstrated by the median value of `0` (*Unavailable*) for this sub-criterion.

Figure 1 shows the distribution of the reproducibility levels for each criterion. None of the papers reached the highest reproducibility level of `3` (*Available and Open*) on any criterion. Only 12 papers reached level `2` (*Available*) in the *Input Data* criterion. Similar to previous results [26], the number of papers with level `0` for *Input Data* was especially high (33, corresponding to 44%), which is a significant barrier to reproduction since input data is not only unavailable but also cannot be recreated from the information provided in the paper.

Preprocessing applied to only 51 publications. For 24 papers, the *Preprocessing* criterion was not applicable (*NA*). This large number is a result of our decision to assess *Preprocessing* only if papers explicitly stated or described a preprocessing step in their analysis, which few did. This does not mean the assessment ignored missing information on preprocessing step, only that such missing information would then reduce the level of the *Methods* criterion instead. Obviously, if data preprocessing is required but it is either not indicated in the paper or is not provided as an additional (computational) step or resource, the ability to reproduce the paper will be limited. The achieved levels for *Preprocessing* remained low: 37 papers reach level `1` (*Documented*), about half of the papers with level `1` in the *Methods* criterion. For the other half, it was not clear whether data preprocessing tasks existed at all, or whether these tasks were part of the main analysis.

Methods and *Results* criteria show a similar distribution (see Figure 1). Indeed, 65 publications had level `1` in both criteria, which represents 87% of the papers assessed. In this sense, most of the assessed papers fall below the minimum standard for reproduction in the methods and results criteria. All papers except one reached level `1` for the *Results* criterion, which shows that the peer review worked as expected for almost all articles. In other words, authors are concerned with making the results understandable to the reviewers, which is not always the case for the other criteria. More generally, this aspect raises the question of whether peer review should stop in the absence of minimal evidence of the input data,

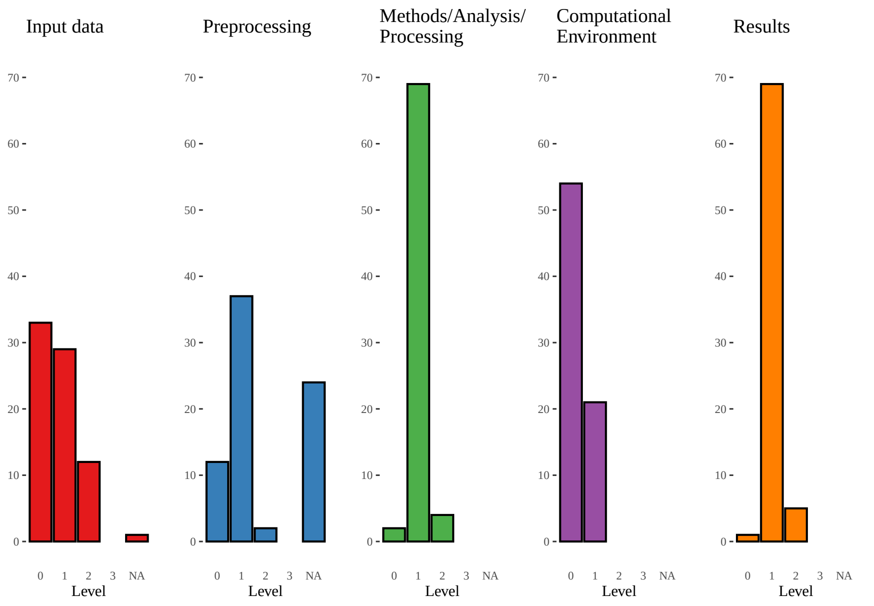


Figure 1 Barplots of reproducibility assessment results; levels range from 0 (leftmost bar) to 'not applicable' (rightmost bar).

analysis, and computational environment used in a paper.

Finally, papers scored worse on the *Computational Environment* criterion. Overall, 54 publications (72%) remained at level 0, which means that no information was provided in the paper about the computing environment, tools, or libraries used in the reported analysis. The *Computational Environment* criterion and the *Input Data* criterion accounted for a significant number of 0 values, which clearly signals an impediment to reproduction. It also shows a rather low recognition of data and software as academic outputs, because both data and software should be properly cited to give credit to their creators [18, 12].

Figure 2 shows an alluvial diagram of all scores, i.e., combinations of criteria values of those 49 papers without any *NA* criterion. Most of the excluded papers have *NA* for *Preprocessing*, therefore this criterion is not included in the figure. The diagram confirms overall patterns seen before. The vast majority of papers have level 1 in *Methods/Analysis/Processing* and *Results*. *Input data* is most diverse, with a surprisingly large number of papers with level 0 but also the largest fraction of papers reaching level 2. Many papers show low levels in *Computational Environment*.

The diagram illustrates how groups of papers with similar properties 'flow' through the different criteria Three major groups, which represent 34 of the papers (69%) included in the figure, become visible as broad bands. Two groups with 10 papers each start with level 0 for *Input Data* and 1 for *Methods/Analysis/Processing* and reach a 1 for *Results*, while they are divided equally between level 0 and 1 for *Computational Environment*. These two groups seem to indicate that the authors and reviewers alike follow the established pattern that results outweigh concerns for transparency and reproducibility, since computational papers with *Unavailable* input data are irreproducible The third and largest group matches the overall mean values for the typical GIScience paper with level 1 for all criteria except for *Computational Environment*.

The diagram also shows additional interesting patterns for a few papers. The papers with the lowest level of 0 in *Results*, i.e., according to the assessors the results are doc-

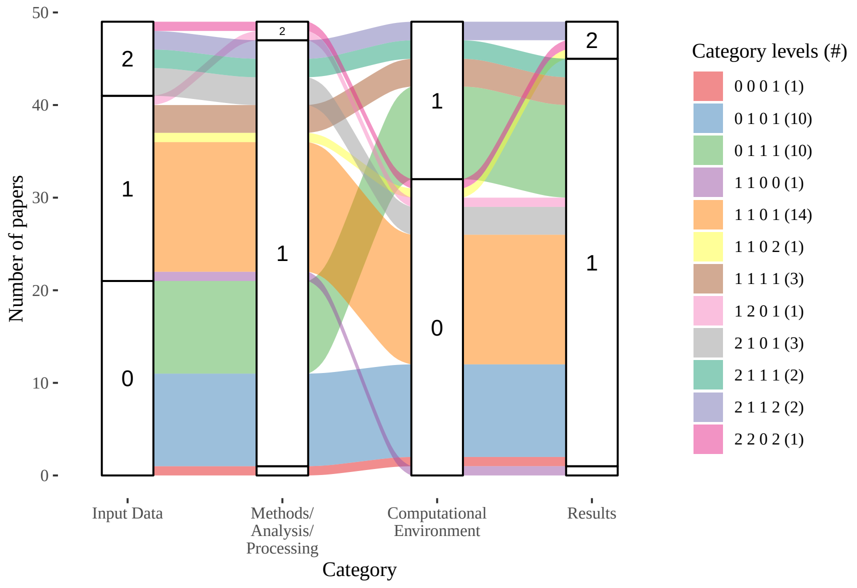


Figure 2 Alluvial diagram of common groups of papers throughout 4 of 5 categories including only papers without any "not applicable" *(Level* `NA`*)* value; category *Preprocessing* was dropped because difficulty to clearly assess it lead to many "not applicable" values.

umented insufficiently and thus difficult or impossible to fully understand, actually have better values in previous criteria. Only few papers that start with level `2` in *Input Data* can keep this level for *Methods/Analysis/Processing*, and even those who do later drop to level `0` in *Computational Environment*. Only one paper each shows the following surprising paths: Starting with level `1` for *Input Data* , then moving up to level `2` in *Methods*, before reaching level `2` in *Results* despite having only values of `1` or `0` in other criteria. In summary, not a single paper can reach the required levels for an immediate reproduction by ensuring that all required pieces are *Available* (level `2`), not even considering the further challenges for reproductions, such as incomplete documentation [28]. An investigation of yearly scores to track developments over time does not show any trend, i.e., there is little change in reproducibility over the study period[10]. The overall low values for *Computational Environment* are one signal that confirms the growing concerns for reproducibility and reusability of computational research are not misplaced.

5 Discussion

5.1 State of reproducibility in the GIScience conference series

Our first research objective was to assess the state of reproducibility in the GIScience conference series. A recurrent issue found in the analysis was the inability to access input data based on the information provided in the paper. Most of the links and pointers to datasets reported at the time of publication were either broken (e.g., non-existing resource, HTTP 404 error, invalid URL syntax) or not available anymore (URL works but redirects to a

[10] See the additional analysis and plots published at `https://nuest.github.io/reproducible-research-at-giscience/giscience-reproducibility-assessment.html` or in the paper's reproducibility package [27].

different generic page; specific resource from the paper no longer exists). In these cases, a level `2` in the *Input Data* criterion was deserved at the time of publication; however, when evaluating the level of reproducibility some time later, as was done in this work, level `2` is no longer suitable for those papers. From a reproducibility point of view, the input data was therefore not accessible, although contacting the authors could still be attempted. However, according to the meaning of the criterion and in practical terms, this is equivalent to including the statement "available upon request" in the paper and thereby level `0`. An important part of reproducibility is that access to material should not degrade over time, which is best achieved by depositing data in repositories, including sensitive data (using the appropriate mechanisms), and properly citing it. In this assessment of reproducibility, we decided to give the authors the benefit of the doubt and awarded a value of `2` for *Input Data* even if we could not conclusively determine, e.g., by using the Internet Archive's Wayback Machine[11], whether the original website ever existed.

Regarding the common situation of a paper with *Documented* (level `1`) for all criteria, our interpretation is that this is indeed a regular paper that is up to current scientific standards. Does this imply that a paper with *Unavailable* (level `0`) in any criterion should not have been accepted? We believe that this requires differentiation between past and future papers. The criteria used in this paper were not included in the previous call for papers or in the reviewer guidelines, and therefore received less attention from authors or reviewers. Thus, we have analysed work in a historical context when there were few concrete incentives to push these aspects, beyond the general concerns for good scientific practice. Nowadays, with awareness about reproducibility being raised through initiatives, projects, and publications about it, we would expect that reproducibility levels increase, and argue that papers with *Unavailable* in one more criteria should not be accepted anymore without a clear and explicit justification (e.g., sensitive data on human subjects). This does not imply that it is always necessary to achieve the gold standard of *Available and Open*. The overall objective should be to make a paper as reproducible as possible before publication. We argue that, for most currently published works at the GIScience conference, *Available* would have been achievable and feasible with reasonable efforts.

However, such a change in standards for paper acceptance would also mean that researchers, editors, and publishers might have to reevaluate their focus on publishing novel and purportedly groundbreaking results in science, and give as much weight to publishing the full process and collection of parts that would allow readers to try to fully understand the research. Clearly, *Unavailable* for *Input Data* is the most problematic, because without sufficient knowledge about the characteristics of the input data, all attempts at reproducing results are bound to fail, even when the textual documentation of the data would potentially allow for an time-intensive recreation of the computational workflow.

5.2 Transferability of method

Concerning our second research objective, we can state that the overall process and the application of the reproducibility rubric was successfully replicated with a different data set. This is not entirely surprising given that AGILE and GIScience conference series share similarities in target audience, review process, and publication of proceedings (more on that in the following section). More importantly, the process faced similar challenges as we recalled from its earlier application. This is crucial information, because the successful

[11] `https://web.archive.org/`.

replication of the process, including its challenges, enables us and others to ground any changes in solid evidence. In particular the *Preprocessing* criterion caused many discussions among the reproducibility reviewers during the assessment. It is often not clear or a matter of interpretation if a particular processing step belongs to a minor basic transformation of input data, if it is already part of the main analysis, and when it is a truly distinct step in the process. The borders are vague and hence scores should be interpreted with caution. Likewise, the *Computational environment* is also difficult to distinguish from analysis, and technology and practices for the effective management of the computing environment have reached mature states relatively recently. Future reproducibility assessments of papers could provide a more precise definition for *pre*-processing, e.g., only use it if the authors use the term, or might consider to drop the category, and benefit from rules to deal with the specific issues of older workflows, similar as discussed for input data above. Furthermore, it is important to remember that the levels of reproducibility are not equidistant in the sense that a level of `2` would be twice as good as a level of `1`, or that the effort needed is twice as high. A level of `1` should be the standard for current and future peer-reviewed papers. Reaching level `2` requires several additional steps, while reaching the gold standard of `3` is again a comparatively small step from level `2` in terms of effort - the main difference is to use public repositories with a DOI - yet with a high positive impact on permanent accessibility.

Although the replication was successful, the process was again labour-intensive, making it problematic to scale it up to assess multiple years of several popular journals, for example. Further, despite our best efforts for transparency and the four-eyes principle in the assessment, the process is inherently subjective. A different group of investigators might score papers differently. While natural language processing techniques have made great progress in the past decades, an automated assessment of a paper's reproducibility still seems out-of-reach. Including important information as machine-readable metadata could allow to come closer to automation.

5.3 Comparison of conferences

Given that we followed the same process as in [26] and demonstrated the transferability of the method, comparing the two conference series seems appropriate. It is important to remember that we do not attempt such a comparison with the objective of declaring a "winner". The published work and contributing community of the two conferences are similar enough for a comparison, yet their organisation (setup, process, geographic focus) differ too much for a simplistic ranking. However, a comparison is required to sensibly discuss whether the guidelines developed for AGILE might also be promising for GIScience: Are they transferable? If not, what adaptations seem necessary?

Concerning the contributing and participating academic communities, Egenhofer et al. [8] and Kemp et al. [14] both include both conferences series as outlets for GIScience research. Further, Keßler et al. [15] investigate the bibliographies of four GIScience conference series, including GIScience and AGILE for the year 2012, and identify 15 authors who have published in both conference series. We conducted a cursory investigation of the body of authors for full papers, revealing significant overlap[12]: Out of 571 unique AGILE and 405 unique GIScience full paper authors, 86 published in both conferences, and this includes all 15 authors mentioned by Keßler et al. [15]. Therefore, the strong relation between the AGILE

[12] The data and code for the brief exploration into the authorship across the conferences considered in this work can be found in the directory `author_analysis` of this paper's reproducibility package [27].

Table 3 Mean values per criterion for both conferences (rounded to two decimal places)

Criterion	AGILE full papers	GIScience papers
input data	0.67	0.72
method/analysis/processing	1.00	1.03
computational environment	0.62	0.28
results	0.88	1.05

and GIScience conference series confirms our approach to apply the same methodology to GIScience that has been developed for AGILE conference publications, and it might lead to similar implications for improving reproducibility.

Nevertheless, before discussing any strategies to improve reproducibility, it is important to identify and consider the differences between the two conference series. GIScience is a biannual conference series whereas AGILE is annual, and they feature different pre-publication review processes and review management systems: In AGILE both authors and reviewers are anonymous, while in GIScience only the reviewers are. Furthermore, the AGILE conference series has the AGILE association[13] as an institutional supporter, which means a more stable organisational and financial framework for activities spanning more than one or between conferences. However, like GIScience, local conference organisers for AGILE have the main financial burden and experiences are informally handed over between organising committees. Geographic focus is also different: GIScience has a global target audience, and the individual conferences are likely to be different in their contributor communities because of the moving conference location, which often means lowered accessibility for authors from other parts of the world. AGILE, by comparison, has a European focus and accessibility is more homogeneous, although the conference location moves every year,. This likely translates into a less fluctuating and less geographically diverse audience at AGILE. Clearly, these observations will need a reassessment in several years to evaluate the impact of both conferences going full online in 2020/21 because of the travel and activity restrictions due to the COVID-19 pandemic.

Concerning the paper corpora, the publication years considered here (2012-2018) are similar to the assessment of AGILE papers (2010-2017), which makes the results comparable in the sense of what methods and tools would have been available for authors. Furthermore, we note that both conferences have a similar ratio of conceptual papers which were not assessed for reproducibility: In the AGILE corpus we identified 5 of 32 conceptual papers (15.6%), in the GIScience corpus there were 12 of 87 (13.8%). This indicates that both conferences have similar share of papers that used, at least in part, computational methods. On the content of the papers, our overall impression was that a larger share of GIScience papers included theoretical, conceptual, or methodological aspects, while AGILE papers seemed to feature more empirical and/or applied geoinformation science research.

Regarding the results of the reproducibility assessments as summarised in Table 3, the nature of the data and sample size does not support statistical analyses on significant differences. Nevertheless, looking at the *Input Data* criterion, GIScience has a slightly higher mean value compared to AGILE full papers (0.72 as opposed to 0.67) and a median of 1. These values indicate that the GIScience contributions had a slightly better, but by no means optimal, availability of input data. The pattern of reproducibility of the papers' workflows (category *Method, Analysis, Processing*) was very similar for the two conference

[13] `https://agile-online.org/`.

series: The majority of papers achieved a level of `1`, resulting in a mean of 1.03 for GIScience and 1 for AGILE full papers. The *Computational Environment* category shows the largest difference (although at overall low levels): AGILE scored better with a mean of 0.62 vs. 0.28 for GIScience. The *Results* category scores were again slightly higher for GIScience, with a mean of 1.05 vs. a mean of 0.88 for AGILE. Several papers in AGILE received a level of `0` here, indicating that crucial information is missing to connect analysis outputs and presented results. We refrain from comparing the *Preprocessing* category for the reasons stated earlier.

This comparison lets us draw two main conclusions: First, we conclude that both the target audience and the content of the two conference series are similar enough to be afflicted with similar shortcomings in terms of reproducibility, and thus, they both likely respond to similar solutions. Second, we conclude that the AGILE conference series seems structurally better positioned to support changing culture, because of a more stable audience and institutional support. The introduction of the AGILE reproducibility guidelines was achieved within a short time frame and with financial support in the form of an "AGILE initiative", including travel funding for an in-person workshop. For GIScience, the task of changing the review process to foster better reproducibility falls squarely on the shoulders of the changing program committees. However, the initial results of AGILE's new guidelines show that even small changes can lead to a significantly improved outcome.

6 Conclusions and outlook

In this work we investigated the reproducibility of several years of GIScience conference publications. The paper corpus is large enough for a representative sample and comparable to that used for the AGILE assessment study due to largely overlapping time window. However, this study does not intend to make judgements on AGILE vs. GIScience conference quality, nor to question the papers' scientific soundness or relevance, since they were accepted for publication at a reputable conference. Instead, we investigated the papers along a single desirable quality dimension, reproducibility, which implies requirements on openness and transparency.

Using a similarly high bar for reproducibility as in the earlier assessment study, the results show room for improvement, as none of the presented articles were readily reproducible. The majority of articles provided some information, but not to the degree required to facilitate transparent and reusable research based on data and software. Overall, this is very similar to the outcomes of our earlier study on AGILE papers. As part of the AGILE assessment, we described concrete recommendations for individuals and organisations to improve paper reproducibility [26]. We have argued that AGILE and GIScience share a sufficiently common domain/discipline characteristics, audience, and author community, such that for both communities the strategies to improve the situation should be similar. Therefore, the previously identified recommendations are transferable to the GIScience conference series, with the most important recommendations being (1) promoting outstanding reproducible work, e.g., with awards or badges, (2) recognizing researchers' efforts to achieve reproducibility, e.g., with a special track for reproducible papers, implementing a reproducibility review, open educational resources, and helpful author guidelines including data and software citation requirements and a specific data/software repository, and (3) making an institutional commitment to a policy shift that goes beyond mere accessibility [33]. These changes require a clear roadmap with a target year, e.g., 2024, when GIScience starts to only accept computationally reproducible submissions and to check reproducibility before papers are accepted.

The concluding statement of Archmiller et al. [1] is directly transferable to GIScience: The challenges are not insurmountable, and increased reproducibility will ensure scientific integrity. The AGILE reproducible paper guidelines [24] and the associated reproducibility review processes as well as other community code review systems such as CODECHECK [9] are open and "ready to use". They can also be adopted for GIScience conferences, e.g., to suit the peer review process goals and scheduling. Kedron et al. [13] stressed the need for a comprehensive balanced approach to technical, conceptual, and practical issues. They further pointed out that simple availability does not automatically lead to adoption. Therefore, a broad discourse around these recommendations, tools, and concepts would be beneficial for all members of the community, whether their work is more towards conceptual, computational, or applied GIScience. A survey for authors, as conducted for AGILE [26], could help identify special requirements and specific circumstances, beyond the findings presented here and in related work.

Future work may replicate the reproducibility assessment at other major events and outlets for GIScience research, such as GeoComputation or COSIT conferences and domain journals (cf. [8] for an extensive list), but we would not expect significantly differing results. Practical reproductions of papers, and even more so replications of fundamental works, are promising projects to convincingly underpin a call for a culture change [29]. A successful *reproducibility turn* would not mean that every reproducible paper would be fully reproduced, nor would this be necessary. But at least for influential, e.g., highly cited papers, a validation of their applicability and transferability to other study areas should be possible—reproducibility is a prerequisite for that. For example, Egenhofer et al. [8] provide for a list of the most frequently cited articles as potential candidates. Such a project would ideally be supported with proper funding. There is currently growing activity in the GIScience discipline to address reproducibility and replicability of geospatial research. The GIScience conference community has the opportunity to play a leading and shaping role in this process, thereby ensuring its continuing attractiveness for authors to submit their work, and in consequence its high relevance for the wider GIScience discipline. A timely adoption of the technological and procedural solutions may allow GIScience researchers, together with the entirety of academia, to level up and approach the challenges of the *"second phase of reproducible research"* by tackling long-term funding for maintenance of code and data and building supporting infrastructure for reproducible research [31].

References

1 Althea A. Archmiller, Andrew D. Johnson, Jane Nolan, Margaret Edwards, Lisa H. Elliott, Jake M. Ferguson, Fabiola Iannarilli, Juliana Vélez, Kelsey Vitense, Douglas H. Johnson, and John Fieberg. Computational Reproducibility in The Wildlife Society's Flagship Journals. *The Journal of Wildlife Management*, 84(5):1012–1017, 2020. doi:10.1002/jwmg.21855.

2 Lorena A. Barba. Terminologies for Reproducible Research. *arXiv:1802.03311 [cs]*, February 2018. arXiv: 1802.03311. URL: https://arxiv.org/abs/1802.03311.

3 Chris Brunsdon. Quantitative methods I: Reproducible research and quantitative geography. *Progress in Human Geography*, 40(5):687–696, October 2016. doi:10.1177/0309132515599625.

4 Chris Brunsdon and Alexis Comber. Opening practice: supporting reproducibility and critical spatial data science. *Journal of Geographical Systems*, August 2020. doi:10.1007/s10109-020-00334-2.

5 Giovanni Colavizza, Iain Hrynaszkiewicz, Isla Staden, Kirstie Whitaker, and Barbara McGillivray. The citation advantage of linking publications to research data. *PLOS ONE*, 15(4):e0230416, April 2020. doi:10.1371/journal.pone.0230416.

6 David L. Donoho. An invitation to reproducible computational research. *Biostatistics*, 11(3):385–388, July 2010. doi:10.1093/biostatistics/kxq028.

7 Matt Duckham, Edzer Pebesma, Kathleen Stewart, and Andrew U. Frank, editors. *Geographic Information Science*. Springer International Publishing, 2014. doi:10.1007/978-3-319-11593-1.

8 M. Egenhofer, K. Clarke, S. Gao, Teriitutea Quesnot, W. Franklin, M. Yuan, and David Coleman. Contributions of GIScience over the past twenty years. In Harlan Onsrud and Werner Kuhn, editors, *Advancing Geographic InformationScience: The Past and Next Twenty Years*. GSDI Association Press, Needham, MA, 2016. URL: http://www.gsdiassociation.org/images/publications/AdvancingGIScience.pdf.

9 Stephen Eglen and Daniel Nüst. CODECHECK: An open-science initiative to facilitate sharing of computer programs and results presented in scientific publications. *Septentrio Conference Series*, (1), September 2019. doi:10.7557/5.4910.

10 Juliana Freire, Norbert Fuhr, and Andreas Rauber. Reproducibility of Data-Oriented Experiments in e-Science (Dagstuhl Seminar 16041). *Dagstuhl Reports*, 6(1):108–159, 2016. URL: http://drops.dagstuhl.de/opus/volltexte/2016/5817, doi:10.4230/DagRep.6.1.108.

11 Michael F. Goodchild. Geographical information science. *International journal of geographical information systems*, 6(1):31–45, January 1992. doi:10.1080/02693799208901893.

12 Daniel S. Katz, Neil P. Chue Hong, Tim Clark, August Muench, Shelley Stall, Daina Bouquin, Matthew Cannon, Scott Edmunds, Telli Faez, Patricia Feeney, Martin Fenner, Michael Friedman, Gerry Grenier, Melissa Harrison, Joerg Heber, Adam Leary, Catriona MacCallum, Hollydawn Murray, Erika Pastrana, Katherine Perry, Douglas Schuster, Martina Stockhause, and Jake Yeston. Recognizing the value of software: a software citation guide. *F1000Research*, 9:1257, January 2021. URL: https://f1000research.com/articles/9-1257/v2, doi:10.12688/f1000research.26932.2.

13 Peter Kedron, Wenwen Li, Stewart Fotheringham, and Michael Goodchild. Reproducibility and replicability: opportunities and challenges for geospatial research. *International Journal of Geographical Information Science*, 0(0):1–19, August 2020. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/13658816.2020.1802032. doi:10.1080/13658816.2020.1802032.

14 Karen Kemp, Werner Kuhn, and Christoph Brox. Results of a survey to rate GIScience publication outlets. Technical report, AGILE Initiative - GIScience Publication Rating, 2013. URL: https://agile-online.org/conference_paper/images/initiatives/results_of_a_survey_to_rate_giscience_publications.pdf.

15 Carsten Keßler, Krzysztof Janowicz, and Tomi Kauppinen. spatial@linkedscience – Exploring the Research Field of GIScience with Linked Data. In Ningchuan Xiao, Mei-Po Kwan, Michael F. Goodchild, and Shashi Shekhar, editors, *Geographic Information Science*, Lecture Notes in Computer Science, pages 102–115, Berlin, Heidelberg, 2012. Springer. doi:10.1007/978-3-642-33024-7_8.

16 Markus Konkol, Christian Kray, and Max Pfeiffer. Computational reproducibility in geoscientific papers: Insights from a series of studies with geoscientists and a reproduction study. *International Journal of Geographical Information Science*, 33(2):408–429, February 2019. doi:10.1080/13658816.2018.1508687.

17 Christian Kray, Edzer Pebesma, Markus Konkol, and Daniel Nüst. Reproducible Research in Geoinformatics: Concepts, Challenges and Benefits (Vision Paper). In Sabine Timpf, Christoph Schlieder, Markus Kattenbeck, Bernd Ludwig, and Kathleen Stewart, editors, *COSIT 2019*, volume 142 of *LIPIcs*, pages 8:1–8:13. Schloss Dagstuhl Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.COSIT.2019.8.

18 Lawrence, Bryan, Jones, Catherine, Matthews, Brian, Pepler, Sam, and Callaghan, Sarah. Citation and Peer Review of Data: Moving Towards Formal Data Publication. *International Journal of Digital Curation*, 6(2), 2011.

19 Florian Markowetz. Five selfish reasons to work reproducibly. *Genome Biology*, 16:274, December 2015. doi:10.1186/s13059-015-0850-7.

20 Jennifer A. Miller, David O'Sullivan, and Nancy Wiegand, editors. *Geographic Information Science*. Springer International Publishing, 2016. doi:10.1007/978-3-319-45738-3.

21 Jannes Muenchow, Susann Schäfer, and Eric Krüger. Reviewing qualitative GIS research-Toward a wider usage of open-source GIS and reproducible research practices. *Geography Compass*, 13(6):e12441, 2019. doi:10.1111/gec3.12441.

22 Marcus R. Munafò, Brian A. Nosek, Dorothy V. M. Bishop, Katherine S. Button, Christopher D. Chambers, Nathalie Percie du Sert, Uri Simonsohn, Eric-Jan Wagenmakers, Jennifer J. Ware, and John P. A. Ioannidis. A manifesto for reproducible science. *Nature Human Behaviour*, 1:0021, January 2017. doi:10.1038/s41562-016-0021.

23 Brian A. Nosek and Timothy M. Errington. What is replication? *PLOS Biology*, 18(3):e3000691, March 2020. doi:10.1371/journal.pbio.3000691.

24 Daniel Nüst, Frank Ostermann, Rusne Sileryte, Barbara Hofer, Carlos Granell, Marta Teperek, Anita Graser, Karl Broman, and Kristina Hettne. AGILE Reproducible Paper Guidelines. 2019. doi:10.17605/OSF.IO/CB7Z8.

25 Daniel Nüst, Frank Ostermann, Rusne Sileryte, Barbara Hofer, Carlos Granell, Marta Teperek, Anita Graser, Karl Broman, and Kristina Hettne. Reproducible Publications at AGILE Conferences. 2019. URL: https://reproducible-agile.github.io/, doi:10.17605/OSF.IO/PHMCE.

26 Daniel Nüst, Carlos Granell, Barbara Hofer, Markus Konkol, Frank O. Ostermann, Rusne Sileryte, and Valentina Cerutti. Reproducible research and GIScience: an evaluation using AGILE conference papers. *PeerJ*, 6:e5072, July 2018. doi:10.7717/peerj.5072.

27 Daniel Nüst, Frank Ostermann, Carlos Granell, and Barbara Hofer. Reproducibility package for "Reproducible Research and GIScience: an evaluation using GIScience conference papers", September 2020. doi:10.5281/zenodo.4032875.

28 Daniel Nüst, Frank Ostermann, Carlos Granell, and Alexander Kmoch. Improving reproducibility of geospatial conference papers lessons learned from a first implementation of reproducibility reviews. *Septentrio Conference Series*, (4), September 2020. URL: https://septentrio.uit.no/index.php/SCS/article/view/5601, doi:10.7557/5.5601.

29 Frank O. Ostermann. Linking Geosocial Sensing with the Socio-Demographic Fabric of Smart Cities. *ISPRS International Journal of Geo-Information*, 10(2):52, January 2021. doi:10.3390/ijgi10020052.

30 Roger D. Peng. Reproducible Research in Computational Science. *Science*, 334(6060):1226–1227, December 2011. doi:10.1126/science.1213847.

31 Roger D. Peng and Stephanie C. Hicks. Reproducible Research: A Retrospective. *arXiv:2007.12210 [stat]*, July 2020. arXiv: 2007.12210. URL: http://arxiv.org/abs/2007.12210.

32 James H. Stagge, David E. Rosenberg, Adel M. Abdallah, Hadia Akbar, Nour A. Attallah, and Ryan James. Assessing data availability and research reproducibility in hydrology and water resources. *Scientific Data*, 6(1):190030, February 2019. Number: 1 Publisher: Nature Publishing Group. doi:10.1038/sdata.2019.30.

33 Victoria Stodden, Jennifer Seiler, and Zhaokun Ma. An empirical analysis of journal policy effectiveness for computational reproducibility. *Proceedings of the National Academy of Sciences*, 115(11):2584–2589, March 2018. doi:10.1073/pnas.1708290115.

34 S. Winter, A. Griffin, and M. Sester, editors. *Proceedings 10th International Conference on Geographic Information Science (GIScience 2018)*, volume 114. LIPIcs, 2018. URL: http://www.dagstuhl.de/dagpub/978-3-95977-083-5.

35 Ningchuan Xiao, Mei-Po Kwan, Michael F. Goodchild, and Shashi Shekhar, editors. *Geographic Information Science*. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-33024-7.

13 CODECHECK: An Open Science initiative for the independent execution of computations underlying research articles during peer review to improve reproducibility

Authors & contribution Daniel Nüst (50%), Stephen J. Eglen

Venue *F1000Research*, status "[version 1; peer review: 1 approved, 1 approved with reservations]" (revision submitted) 10.12688/f1000research.51738.1 (SNIP 2020: 0.92)

Date 03/2020

Licence Creative Commons Attribution (CC BY 4.0)

Repository https://github.com/codecheckers/paper

Check for updates

METHOD ARTICLE

CODECHECK: an Open Science initiative for the independent execution of computations underlying research articles during peer review to improve reproducibility [version 1; peer review: 1 approved, 1 approved with reservations]

Daniel Nüst[1], Stephen J. Eglen[2]

[1]Institute for Geoinformatics, University of Münster, Münster, Germany
[2]Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Cambridge, UK

v1 **First published:** 30 Mar 2021, **10**:253
https://doi.org/10.12688/f1000research.51738.1

Latest published: 30 Mar 2021, **10**:253
https://doi.org/10.12688/f1000research.51738.1

Abstract
The traditional scientific paper falls short of effectively communicating computational research. To help improve this situation, we propose a system by which the computational workflows underlying research articles are checked. The CODECHECK system uses open infrastructure and tools and can be integrated into review and publication processes in multiple ways. We describe these integrations along multiple dimensions (importance, who, openness, when). In collaboration with academic publishers and conferences, we demonstrate CODECHECK with 25 reproductions of diverse scientific publications. These CODECHECKs show that asking for reproducible workflows during a collaborative review can effectively improve executability. While CODECHECK has clear limitations, it may represent a building block in Open Science and publishing ecosystems for improving the reproducibility, appreciation, and, potentially, the quality of non-textual research artefacts. The CODECHECK website can be accessed here: https://codecheck.org.uk/.

Keywords
reproducible research, Open Science, peer review, reproducibility, code sharing, data sharing, quality control, scholarly publishing

This article is included in the Science Policy Research gateway.

Open Peer Review

Reviewer Status ? ✓

	Invited Reviewers	
	1	**2**
version 1 30 Mar 2021	? report	✓ report

1. **Nicolas P. Rougier**, Inria Bordeaux Sud-Ouest, Talence, France

2. **Sarah Gibson**, The Alan Turing Institute, London, UK

Any reports and responses or comments on the article can be found at the end of the article.

Corresponding authors: Daniel Nüst (daniel.nuest@uni-muenster.de), Stephen J. Eglen (sje30@cam.ac.uk)

Author roles: Nüst D: Conceptualization, Data Curation, Formal Analysis, Funding Acquisition, Investigation, Methodology, Project Administration, Resources, Software, Writing – Original Draft Preparation, Writing – Review & Editing; **Eglen SJ**: Conceptualization, Data Curation, Formal Analysis, Funding Acquisition, Investigation, Methodology, Project Administration, Resources, Software, Writing – Original Draft Preparation, Writing – Review & Editing

Competing interests: SJE is on the editorial board at the journal Scientific Data until March 2021. DN is reproducibility chair at the Association of Geographic Information Laboratories in Europe's (AGILE) annual conference.

Grant information: This work was financially supported by the UK Software Sustainability Institute and a Mozilla Science mini grant. DN is supported by grant PE1632/17-1 from the German Research Foundation (DFG).
The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

How to cite this article: Nüst D and Eglen SJ. **CODECHECK: an Open Science initiative for the independent execution of computations underlying research articles during peer review to improve reproducibility [version 1; peer review: 1 approved, 1 approved with reservations]** F1000Research 2021, **10**:253 https://doi.org/10.12688/f1000research.51738.1

First published: 30 Mar 2021, **10**:253 https://doi.org/10.12688/f1000research.51738.1

Abbreviations

ACM: Association for Computing Machinery; ECRs: Early Career Researchers; RCR: Replicated Computational Results; TOMS: Transactions on Mathematical Software.

Introduction

Many areas of scientific research use computations to simulate or analyse their data. These complex computations are difficult to explain coherently in a paper[1]. To complement the traditional route of sharing research by writing papers, there is a growing demand to share the underlying artefacts, notably code and datasets, so that others can inspect, reproduce or expand that work (see Figure 1). Early proponents of this initiative were Buckheit and Donoho[2,3], who noted: *"An article about computational science in a scientific publication is not the scholarship itself, it is merely* ***advertising*** *of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures."*

If researchers start sharing more artefacts, how might these artefacts be examined to ensure that they do what they claim? For example, although scientific journals now require a data sharing statement that outlines what data the authors have (or will) share, journals implement this differently. On one hand, journals have been created to accept "data papers" (e.g., *Scientific Data*, *Earth System Science Data*, *Geoscience Data Journal*, *Biodiversity Data Journal*, *Journal of Open Psychology Data*, *Open Data Journal for Agricultural Research*, *Journal of Open Health Data*); these journals have established rigorous procedures by which data are validated according to standards in each field. On the other hand, many journals still allow authors to state "Data available upon reasonable request". Authors, while possibly well intentioned at the time of writing the article, often cannot provide data when requested as data disappears over time[4].

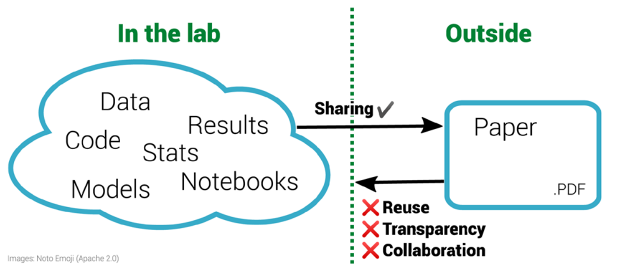


Figure 1. The inverse problem in reproducible research. The left half of the diagram shows a diverse range of materials used within a laboratory. These materials are often then condensed for sharing with the outside world via the research paper, a static PDF document. Working backwards from the PDF to the underlying materials is impossible. This prohibits reuse and is not only non-transparent for a specific paper but is also ineffective for science as a whole. By sharing the materials on the left, others outside the lab can enhance this work.

Given that data are not routinely shared, what hope might there be for sharing computer programs? Both data and software are required to validate a computational analysis; data can be seen as inert whereas code requires an environment to be run in. This makes software harder to share. Our experience is that researchers offer several reasons for why code is not shared, e.g., "there is no documentation", "I cannot maintain it", or "I do not want to give away my code to competitors". Our view is that sharing code, wherever possible, is good for the community and the individual[5,6]. Having code and data openly available, and archived, provides a valuable resource for others to learn from, even if the code is broken or lacks documentation. However, with a little effort, we believe that if an independent person can re-run the programs, this is worth documenting and that this reduces the barrier to evaluating non-text research materials. Just as data journals' validations of data and all journals' peer review provides a "baseline reassurance", i.e., that a paper has been checked by someone with an understanding of the topic[7], the same baseline could be provided for the workflow underlying a paper. With this in mind, we have developed a set of principles and an example workflow that provides a pragmatic way of checking that a paper's code works, i.e., it is reproducible following the Claerbout/Donoho/Peng terminology[8].

Here we offer a thorough description of a process and its variations to integrate a much-needed evaluation of computational reproducibility into peer review, and we demonstrate its feasibility by means of 25 reproductions across scientific disciplines. We call this system CODECHECK.

What is a CODECHECK?

Workflow and people

CODECHECK is best demonstrated by way of our example workflow, and later we expand on the underlying principles. The workflow involves three groups of people: (1) the **author** of a paper providing the code to be checked, (2) the **publisher** of a journal interested in publishing the author's paper, and (3) the **codechecker**, who checks that the author's code works. The six-step workflow we have refined is shown in Figure 2. In this article, we also refer to a **peer-reviewer** who is independent of this process, and performs the traditional academic review of the content of an article.

Step 1: The author submits their manuscript along with the code and data to the publisher. The code and data need not be openly available at this point. However, in many cases the code and data may be published on a code hosting platform, such as GitHub or GitLab. Ideally, the author is expecting the CODECHECK and prepares for it, e.g., by asking a colleague to attempt a reproduction, and providing a set of instructions on how to re-run the workflow.

Step 2: The publisher finds a codechecker to check the code. This is analogous to the publisher finding one or more peer-reviewers to evaluate the paper, except we suggest that the codechecker and the author talk directly to each other.

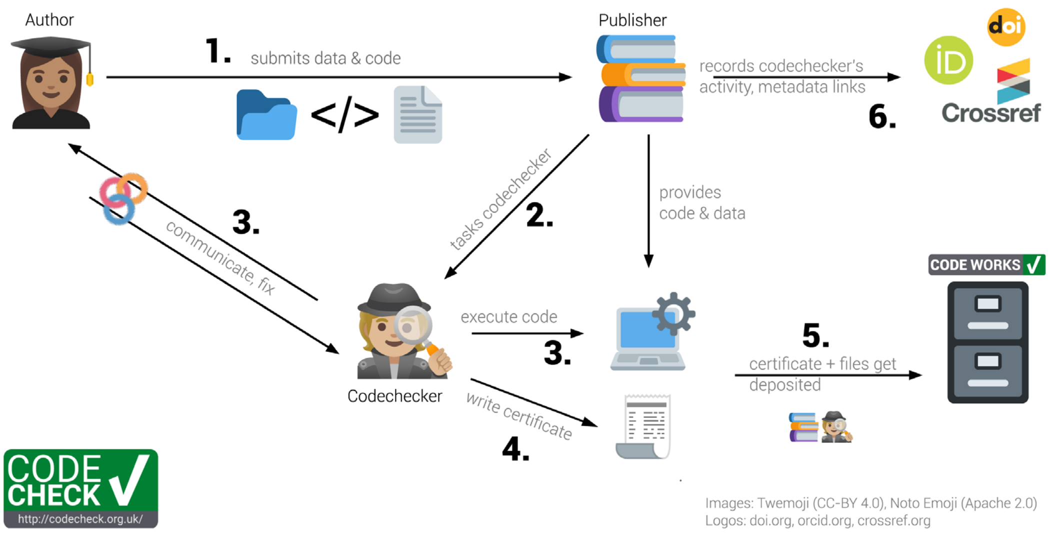

Author
Publisher
1. submits data & code
records codechecker's activity, metadata links
6.
Crossref
tasks codechecker
2.
provides code & data
3.
communicate, fix
CODE WORKS ✓
execute code
3.
5.
certificate + files get deposited
Codechecker
write certificate
4.
CODE CHECK http://codecheck.org.uk/
Images: Twemoji (CC-BY 4.0), Noto Emoji (Apache 2.0)
Logos: doi.org, orcid.org, crossref.org

Figure 2. The CODECHECK example process implementation. Codecheckers act as detectives: They investigate and record, but do not fix issues. Numbers in bold refer to steps outlined in the text.

Step 3: The codechecker runs the code, based on instructions provided by the author. They check if some or all of the results from the paper can be reproduced. If there are any problems running the code, the codechecker asks the author for help, updates, or further documentation. The burden to provide reproducible material lies with the author. The codechecker then tries to run the code again. This process iterates until either the codechecker is successful, or the codechecker concludes the workflow is not reproducible. As part of this process, the codechecker could work entirely locally, relying on their own computing resources, or in the cloud, e.g., using the open MyBinder infrastructure[9] or alternatives, some of which are more tailored to scientific publications while others offer commercial options for, e.g., publishers (cf. 10). A cloud-based infrastructure allows for the codechecker and author to collaboratively improve the code and enforces a complete definition of the computing environment; but, unless secure infrastructure is provided, e.g., by the publisher, this requires the code and data to be published openly online. Note that the task of the codechecker is to check only the "mechanics" of the workflow. In the context of mathematics, Stodden *et al.*[11] distinguish between *verification* and *validation*; following their definition, a CODECHECK ensures verification of computational results, i.e., checking that code generates the output it claims to create, but not a validation, i.e., checking that the code implements the right algorithm to solve the specific research problem. Nevertheless, simply attempting to reproduce an output may highlight a submission's shortcomings in meeting a journal's requirements (cf. 12) and may effectively increase transparency, thereby improving practices (cf. 13) even if the check does not go into every detail.

Step 4: The codechecker writes a certificate stating how the code was run and includes a copy of outputs (figures or tables) that were independently generated. The certificate may include recommendations on how to improve the material. The free text in the certificate can describe exactly what was checked, because each workflow is unique. Since no specific tool or platform is required, such that no authors are excluded, it is futile for the codechecker to use automation or fixed checklists.

Step 5: The certificate and auxiliary files created during the check, e.g., a specification of a computing environment, data subsets or helper scripts, and the original code and data get deposited in an open archive unless restrictions (data size, license or sensitivity) apply. Currently, codecheckers deposit the material on Zenodo themselves, but a publisher may complete this step after integrating CODECHECK into its review process. A badge or other visual aid may be added to the deposit and the paper and link to the certificate. Although a badge simplifies the CODECHECK into a binary value and risks introducing confusion regarding the extent of the check, a badge provides recognition value and acknowledges the completed CODECHECK. The badge and the actual check are incentives for undertaking the effort needed to provide a reproducible workflow.

Step 6: The publisher can, depending on the timing, provide the certificate to peer-reviewers or editors or publish it and link between certificate, paper, and any repositories. Currently, the codechecker creates these connections on Zenodo. They appear as links with a relationship type on the Zenodo landing page for a certificate, e.g., the "related identifiers" and "alternate identifiers" of certificate `2020-025`[14]. The publisher also credits the codechecker's work by depositing the activity in scholarly profiles, such as ORCID (see peer review contributions in ORCID records). The publisher also ensures proper publication metadata, e.g., links from the certificate repository to the published paper or the original code repository.

Variations

Dimensions of CODECHECK workflows. Our workflow is just one of many possibilities of a CODECHECK workflow. Here we consider several dimensions in a space of possible CODECHECK workflows (Figure 3). These aspects touch on timing, responsibilities, and transparency.

When to do a CODECHECK and with what importance? The time at which a CODECHECK is done and its ascribed importance are closely connected, so we describe the dimensions *When* and *Importance* together. The earlier a CODECHECK happens in the publishing process, the more it can affect editorial decisions: Is a paper published, sent back for revisions, or rejected? Even earlier checks, i.e., a CODECHECK of a preprint, may help to improve the workflow itself, even before a publisher is involved. As such, codechecking papers could be part of a preprint server's policy or initiated by interested authors.

Publishers could introduce a CODECHECK as a **strict pre-requisite**. As this can reduce the workload of reviewers, such a check should occur early in the review process. Yet, the later in the review process the check happens, the easier is it to allow bidirectional communication between the author and code-checker, e.g., because the author might already be notified of the paper's acceptance and may be more willing to share materials online closer to the paper's publication date. A **pre-review** CODECHECK means editors would send a submission for peer review only if it passes the check, or include the certificate in the submission package provided to peer-reviewers. Peer-reviewers may then judge the relevance of the computations for the results of the work.

A CODECHECK may also be conducted in **parallel** to the academic peer review. This puts less burden on the turnaround time for the CODECHECK, yet it only makes the outcomes available during the final consideration by the handling editor. The check could also be assigned after suggestion by a reviewer, which would remove the need for submissions to undergo a pre-review screening. However, soliciting such a "specialist review" is much less desirable than having a regular CODECHECK, thus avoiding the situation in which some submissions get special treatment. In both cases, the editor's decision could be based both on CODECHECK and peer-review reports.

A **post-acceptance** CODECHECK would have the smallest impact on editorial decisions and may simply provide **extra merit** on top of the submission's acceptance. This is the least impactful solution in which all material is still evaluated and the results of the check are properly acknowledged, because the check can be completed before publication of the paper. The GIScience checks (see below) falls into this category: by displaying a badge on the volume and article landing pages, the AGILE conference highlights articles whose reproducibility was confirmed. Similarly, in collaborations with journals, some GIScience articles were checked whilst authors worked on revisions.

A CODECHECK may also be conducted **post-publication**, though this requires an update to the article and article metadata to reference the check so that readers can find the CODECHECK. In general, publishers hesitate to make such revisions to published articles. We do not prefer this option as it has the least impact on current publishing practices and downplays the importance of reproducible workflows for ensuring good scientific practice.

Enhancing existing processes with CODECHECKs allows communities to gradually transition towards more open practices. When integrating a CODECHECK into existing review and publication processes, the *turnaround time* is crucial. Depending on when and who conducts the check, it might be done quickly or it might delay publication. We found that a CODECHECK generally takes 2–5 hours, with some outliers on the higher end. This time includes writing and publishing the certificate but excludes actual computation time, some of which took days. These efforts are comparable to the time needed to peer review a submission, which aligns with the efforts some volunteer codecheckers are willing to make. Currently, there is considerable

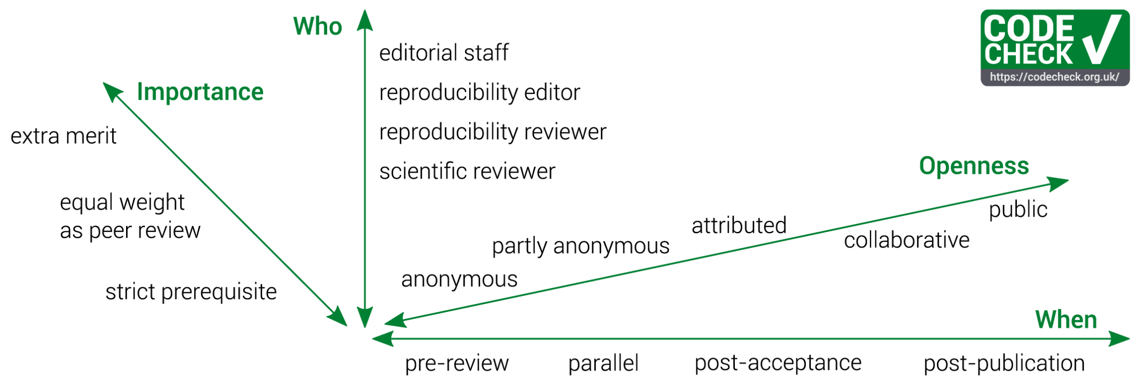


Figure 3. The dimensions of implementing a CODECHECK process.

amount of communicating about the process, especially regarding who publishes which document when, so that proper cross-referencing between paper and certificate is ensured via persistent identifiers. When integrated into a peer review platform, this handling of documents should become much more streamlined.

Openness, or "Who knows who?" Anonymity is broadly discussed, especially in the push towards open peer review as part of the Open Science movement (cf. 15). Without taking a strong stance on this topic, our motivation behind CODECHECK for higher transparency and reproducibility does indeed favour a more open review process. However, anonymity can protect individuals[16], e.g., junior scientists. The negative effects of a signed review may be reduced if a CODECHECK is not relevant for a journal's decision to accept or reject, but that is, of course, not desirable when the goal is higher transparency and reproducibility. Instead, CODECHECK is a technical process that should generally find fixable problems; it is not aimed at giving an opinion or identifying a faulty approach. If passing a CODECHECK becomes mandatory, full transparency may need revisiting as the relations between authors and codecheckers would fall under the same social and community challenges as open peer review (cf. 17).

The technical nature of the check and the challenge of providing sufficient documentation is why we see great benefits in bidirectional communication between author and codechecker. Instead of trying to fix problems or guess the next step, the codechecker can ask the author to rework the documentation or update code. Instead of struggling to provide perfect instructions and as a result possibly not sharing any code or data, the author can make a best effort to document sufficiently. Authors and readers can profit from a codecheckers' experience and approach, as during the check they may create useful and instructive files, e.g., a machine-readable computing environment specification. While communication between author and codechecker may be anonymised via the publisher, it most likely only helps to protect the identity of the codechecker, because code is hard to anonymise. Therefore, the most effective and desirable situation for the stakeholders is to hold a open and collaborative CODECHECK. The contributions by the codechecker may even be integrated into the code of the workflow and be acknowledged as code commits. This way, proper credit can be given within the research software development community.

Who does the CODECHECK? Just as with peer-reviewers, a potential codechecker should have the right skills and availability to do the work. Ideally, the codechecker has a matching code *and* domain expertise to the paper, although a well-documented workflow should be executable by any computationally-competent person. Naturally, the more prerequisite knowledge the codechecker has, the quicker they can understand the goals and mechanics of an analysis. From our experiences, the priority should be given to matching technical expertise first, as lacking knowledge in setting up a computing environment with a particular language or tool is much more of a problem than assessing the outcome, e.g., comparing created figures with the original, without an in-depth understanding of the domain. The depth of the check will mostly be driven by the time required and expertise of the checker, though in general, we expect a CODECHECK to consider reproducibility of the results above performance of the code.

Codecheckers could be drawn from a regular pool of **peer-reviewers**, or from a special group of **reproducibility reviewers** via specific roles such as **reproducibility editors**, or **editorial staff** with a publisher. One codechecker is sufficient to verify the workflow since it is mostly a factual process. Code usually harbours systematic and repeatable mistakes and is thereby more reliable and auditable than processes controlled by humans[18], e.g., in a laboratory. If however publication of the paper depends on the CODECHECK, a second opinion may be required.

We also see a great opportunity to involve *early-career researchers* (ECRs) as codecheckers. ECRs arguably have a high interest in learning about new tools and technologies, to build up their own expertise. CODECHECK offers a way for ECRs to gain insights into new research and highlight the importance of reproduction. *ReScience X*, a journal devoted to reproduction and replication experiments[19], shares an interest in this combination. ECRs are also often familiar with new technologies, thus also making them likely to author CODECHECK-ready manuscripts. A supporting data point for ECRs as early adopters is that they are responsible for 77% of 141 registered reports that were submitted[20]. As ECRs are introduced to peer review as codecheckers, they may transition into the role of peer-reviewer over time. Overall, we see several opportunities and benefits to setting up a new process for codechecking with a clear commitment to openness and transparency, independent of the current peer review process (see *Openness* dimension).

The codechecker could be a member of **editorial staff**; this is the most controlled but also resource-intensive option. Such a resource commitment would show that publishers are investing in reproducibility, yet this commitment may be hard for small publishers. These codecheckers could be fully integrated into internal workflows. Credit for doing the codecheck is also achieved, as it is part of their duties. By contrast, it is useful for researchers to be publicly credited for their reviewing activity. A regular review may be listed in public databases (e.g., ORCID, see Step 6 above, or commercial offerings such as Publons, and ReviewerCredits); a codechecker could be similarly listed. The codechecker community has over 20 volunteers who signed up in the last year, see https://github.com/codecheckers/codecheckers/. Their motivations, mentioned in the registration information, include: supporting reproducible research and Open Science, improve coding skills, gaining experience in helping scientists with their code, encouraging a sharing culture, and learning from other people's mistakes; many are also motivated simply by curiosity. We see benefits to an open shared list of codecheckers across journals rather than a private in-house group, as this may allow for better matches regarding expertise and workload sharing. This community can establish CODECHECK as a viable option for independent no-cost Open Access journals.

Core principles

The workflow and variations outlined describe our current views on how code could be checked. They are not immutable, but we believe the following core principles underpin our CODECHECK process:

1. Codecheckers record but don't investigate or fix.
The codechecker follows the author's instructions to run the code. If instructions are unclear, or if code does not run, the codechecker tells the author. We believe that the job of the codechecker is not to fix these problems but simply to report them to the author and await a fix. The level of documentation required for third parties to reproduce a workflow is hard to get right, and too often this uncertainty leads researchers to give up and not document it at all. The conversation with a codechecker fixes this problem.

2. Communication between humans is key.
Some code may work without any interaction, e.g. [21], but often there are hidden dependencies that need adjusting for a particular system. Allowing the codechecker to communicate directly and openly with the author make this process as constructive as possible; routing this conversation (possibly anonymously) through a publisher would introduce delays and inhibit community building.

3. Credit is given to codecheckers.
The value of performing a CODECHECK is comparable to that of a peer review, and it may require a similar amount of time. Therefore, the codechecker's activity should be recorded, ideally in the published paper. The public record can be realised by publishing the certificate in a citable form (i.e., with a DOI), by listing codecheckers on the journal's website or, ideally, by publishing the checks alongside peer review activities in public databases.

4. Workflows must be auditable.
The codechecker should have sufficient material to validate the workflow outputs submitted by the authors. Stark[22] calls this "preproducibility" and the ICERM report[11] defines the level "Auditable Research" similarly. Communities can establish their own good practices or adapt generic concepts and practical tools, such as publishing all building blocks of science in a research compendium (cf. https://research-compendium.science/) or "repro-pack"[23]. A completed check means that code could be executed at least once using the provided instructions, and, therefore, all code and data was given and could be investigated more deeply or extended in the future. Ideally, this is a "one click" step, but achieving this requires particular skills and a sufficient level of documentation for third parties. Furthermore, automation may lead to people gaming the system or reliance on technology, which can often hide important details. All such aspects can reduce the understandability of the material, so we estimate our approach to codechecking, done without automation and with open human communication, to be a simple way to ensure long-term transparency and usefulness. We acknowledge that others have argued in favour of bitwise reproducibility because, in the long run, it can be automated (e.g., https://twitter.com/khinsen/status/1242842759733665799), but until then we need CODECHECK's approach.

5. Open by default and transitional by disposition.
Unless there are strong reasons to the contrary (e.g., sensitive data on human subjects), all code and data, both from author and codechecker, will be made freely available when the certificate is published. Openness is not required for the paper itself, to accommodate journals in their transition to Open Access models. The code and data publication should follow community good practices. Ultimately we may find that CODECHECK activities are subsumed within peer review.

Implementation

Register

To date we have created 25 certificates (Table 1) falling into three broad themes: (1) classic and current papers from computational neuroscience, (2) COVID-19 modelling preprints, and (3) GIScience.

The first theme was an initial set of papers used to explore the concept of CODECHECK. The idea was to take well-known articles from a domain of interest (Neuroscience). Our first CODECHECK (certificate number 2020-001) was performed before publication on an article for the journal *GigaScience*, which visualized the outputs from a family of supervised classification algorithms.

The second theme was a response to the COVID-19 pandemic, selecting papers that predicted outcomes. The checks were solicited through community interaction or by our initiative rather than requested from journals. Some certificates were since acknowledged in the accepted papers[24,25]. In particular, we codechecked the well-known Imperial college model of UK lockdown procedures from March 2020, demonstrating that the model results were reproducible[26,27].

The third theme represents co-author DN's service as a Reproducibility Reviewer at the AGILE conference series, where the *Reproducible AGILE* Initiative[28] independently established a process for reproducing workflows at the AGILE conference series[29]. While using slightly different terms and infrastructure ("reproducibility reports" are published on the Open Science Framework instead of certificates on Zenodo) AGILE reproducibility reviews adhere to CODECHECK principles. A few checks were also completed as part of peer reviews for GIScience journals.

Annotated certificate and check metadata

After running the workflow, the codechecker writes a certificate stating which outputs from the original article, i.e., numbers, figures or tables, could be reproduced. This certificate is made openly available so that everyone can see which elements were reproduced and what limitations or issues were found. The certificate links to code and data used by the codechecker, allowing others to build on the work. The format of the

Table 1. Register of completed certificates as of December 2020. An interactive version is available at http://codecheck.org.uk/register.

Certificate	Research area	Description
2020-001[30]	Machine learning	Code for benchmarking ML classification tool checked post acceptance of manuscript and before its publication in *Gigascience*[31].
2020-002[32]	Neuroscience	Code written for this project checked by second project member as demonstration using paper from 1997 showing unsupervised learning from natural images[33].
2020-003[34]	Neuroscience	Code written for this project checked by second project member as demonstration using classic paper on models of associative memory[35].
2020-004[36]	Neuroscience	Code written for this project checked by second project member as demonstration using classic paper on cart-pole balancing problem[37].
2020-005[38]	Neuroscience	Check of independent reimplementation of spike-timing-dependent plasticity (STDP) model[39] conducted as demonstration for this paper.
2020-006[40]	Neuroscience	Check of independent reimplementation of a generalized linear integrate-and-fire neural model[41] conducted as demonstration for this paper
2020-007[42]	Neuroscience	Check of independent reimplementation of analysing spike patterns of neurons[43] conducted as demonstration for this paper.
2020-008[44]	COVID-19	Code for modelling of interventions on COVID-19 cases in the UK checked at preprint stage[45] and later published[24].
2020-009[46]	COVID-19	Code for analysis of effectiveness of measures to reduce transmission of SARS-CoV-2 checked as preprint[47] and later published[25].
2020-010[27]	COVID-19	Code for analysis of non-pharmaceutical interventions (Report 9) checked as a preprint[48].
2020-011[49]	COVID-19	Code for modelling of COVID-19 spread across Europe was provided by authors and checked while paper was in press[50].
2020-012[51]	COVID-19	Code for modelling of COVID-19 spread across the USA was checked as preprint[52] and later published[53].
2020-013[21]	Neuroscience	Code for analysis of rest-activity patterns in people without con-mediated vision was checked as a preprint[54] after direct contact with the authors.
2020-014[55]	Neuroscience	Code for analysis of perturbation patterns of neural activity was checked after publication as part of publisher collaboration[56].
2020-015[57]	Neuroscience	Code for a neural network model for human focal seizures was checked after publication as part of publisher collaboration[58]
2020-016[59]	GIScience	Code for models demonstrating the Modifiable Aral Unit Problem (MAUP) in spatial data science[60] was checked during peer review.
2020-017[61]	GIScience	Code for spatial data handling, analysis, and visualisation using a variety of R packages[62] was checked after peer review before publication.
2020-018[63]	GIScience	AGILE conference reproducibility report using a demonstration data subset with cellular automaton for modeling dynamic phenomena[64].
2020-019[65]	GIScience	AGILE conference reproducibility report with subsampled dataset for reachability analysis of suburban transportation using shared cars[66].
2020-020[67]	GIScience	AGILE conference reproducibility report using a container for checking in-database windows operators for processing spatio-temporal data[68].
2020-021[69]	GIScience	AGILE conference reproducibility report checking code for comparing supervised machine learning models for spatial nominal entity recognition[70].
2020-022[71]	GIScience	AGILE conference reproducibility report checking code for visualising text analysis on intents and concepts from geo-analytic questions[72].
2020-023[73]	GIScience	AGILE conference reproducibility report on analysis of spatial footprints of geotagged extreme weather events from social media[74].
2020-024[75]	Neuroscience	Code for multi-agent system for concept drift detection in electromyography[76] was checked during peer review.
2020-025[14]	GIScience	Adaptation and application of Local Indicators for Categorical Data (LICD) to archaeological data[77] was checked after peer review before publication.

certificates evolved during the project, as we learnt to automate different aspects of the certification. The metadata is stored in a machine-readable structured file in YAML, the CODECHECK configuration file `codecheck.yml`. The technical specification of the CODECHECK configuration file is published at https://codecheck.org.uk/spec/config/latest/. The configuration file enables current and future automation of workflows and meta-analyses.

Figure 4 shows pages 1–4 (of 10) of an example certificate to check predictions of COVID-19 spread across the USA[51,52]. Figure 4A shows the certificate number and its DOI, which points to the certificate and any supplemental files on Zenodo. The CODECHECK logo is added for recognition and to denote successful reproduction. Figure 4B provides the key metadata extracted from `codecheck.yml`; it names the paper that was checked (title, DOI), the authors, the codechecker, when the check was performed, and where code/data are available. Figure 4C shows a textual summary of how the CODECHECK was performed and key findings. Figure 4D (page 2 of the certificate) shows the outputs that were generated based on the MANIFEST of output files in the CODECHECK. It shows the file name (Output), the description stating to which figure/table each file should be compared in the original paper (Comment), and the file size. Page 3 of the certificate, Figure 4E gives detailed notes from the codechecker, here documenting what steps were needed to run the code and that the code took about 17 hours to complete. Finally, page 4 of the certificate shows the first output generated by the CODECHECK Figure 4F. In this case, the figure matched figure 4 of 52. The remaining pages of the certificate show other outputs and the computing environment in which the certificate itself was created (not shown here).

Tools and resources

We use freely available infrastructure, GitHub and Zenodo, to run our system. The `codecheckers` **GitHub** organisation at https://github.com/codecheckers contains projects for managing the project website, the codecheckers community and its discussions, code repositories, and the main register of CODECHECKs. Both the project website https://codecheck.org.uk/ and the register at https://codecheck.org.uk/register are hosted as GitHub pages. The register database is a single table in CSV format that connects the certificate identifier with the repository associated with a CODECHECK. Each of these repositories, which currently can be hosted on GitHub or Open Science Framework, contains the CODECHECK metadata file `codecheck.yml`. The register further contains a column for the type of check, e.g., community, journal, or conference, and the respective GitHub issue where communications and assignments around a specific check are organised. No information is duplicated between the register and the metadata files. The continuous integration infrastructure of GitHub, GitHub Actions, is used to automate generation of the register. **Zenodo** is our preferred open repository for storing certificates. It mints DOIs for deposits and ensures long-term availability of all digital artefacts related to the project. The CODECHECK community on Zenodo is available at https://zenodo.org/communities/codecheck/. It holds certificates, the regularly archived register[78], and other material related to CODECHECK.

A **custom R package**, `codecheck`, automates repetitive tasks around authoring certificates and managing the register. The package is published at https://github.com/codecheckers/codecheck under MIT license[79]. It includes scripts to deposit certificates and related files to Zenodo using the R package `zen4R`[80] and for the register update process outlined above. Codecheckers can ignore this package, and use their own tools for creating and depositing the certificate. This flexibility accommodates different skill sets and unforeseen technical advances or challenges.

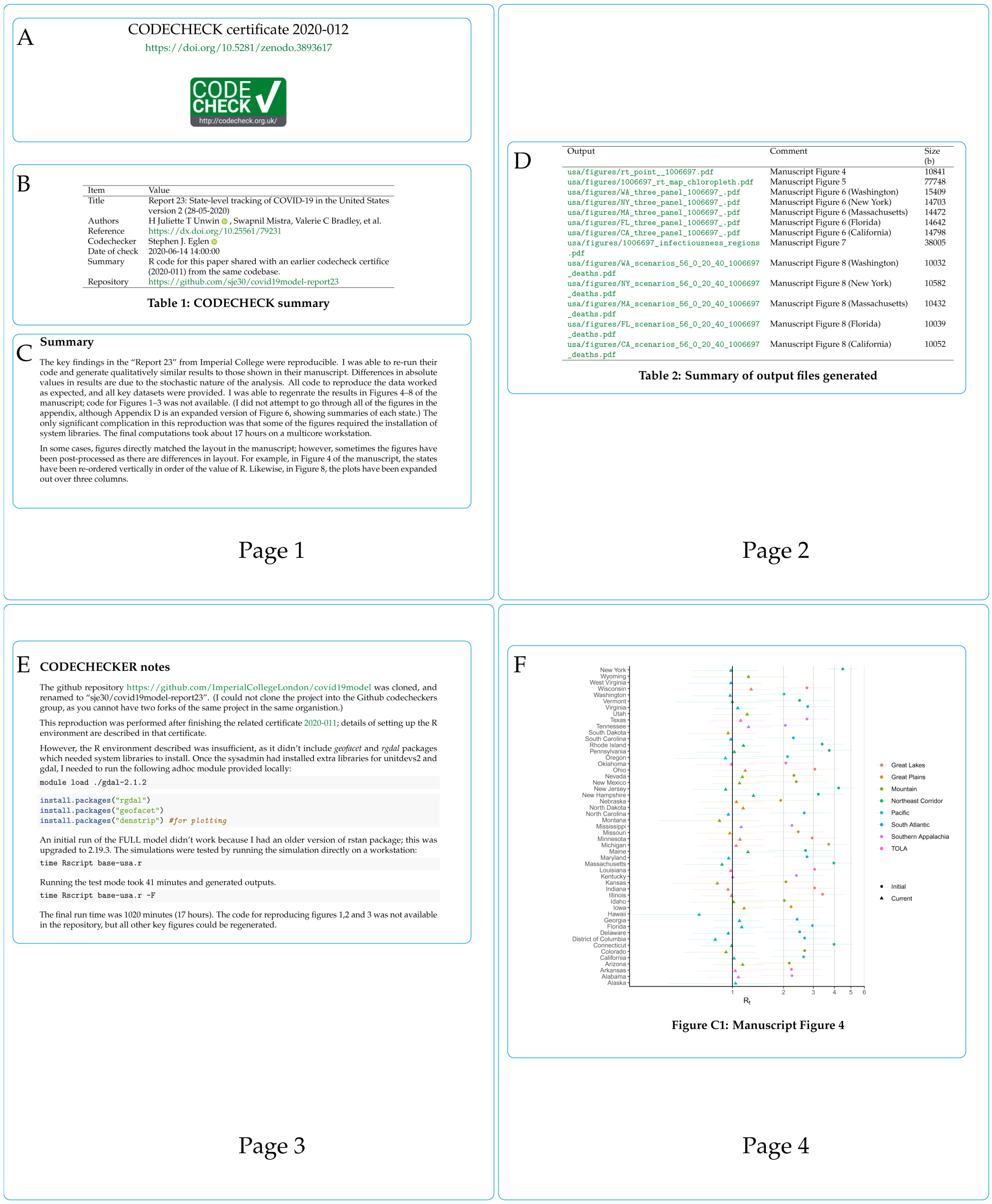
These tools and resources demonstrate that a CODECHECK process can be managed on freely available platforms. Automation of some aspects may improve turnaround time. Our main resource requirements are the **humans** needed for managing the project and processes and the codecheckers. All contributions currently rely on (partly grant-based) public funding and volunteering.

Related work

The journal *ACM Transactions on Mathematical Software (TOMS)* recently established a “Replicated Computational Results” (RCR) review process[81], where “replicable” is the same as our use of “reproducible”. Fifteen RCR Reports have been published so far (search on https://search.crossref.org/ with the term “`Replicated Computations Results (RCR) Report`”on 2020-12-10). and the process is being extended extended to the ACM journal *Transactions on Modeling and Computer Simulation*. The TOMS RCR follows CODECHECK principles 1–4, although our work was independently developed of theirs. The TOMS editorial[81] shares similar concerns about selection of reviewers, as we discussed above. Unlike existing CODECHECK certificates, the RCR reports undergo editorial review. Publication of the RCR report recognises the efforts of the reproducing person, while the potential for this motive to be a conflict of interest is acknowledged. TOMS also recognises reviewer activity in a partnership with Publons (see https://authors.acm.org/author-services/publons).

This activity in the ACM journals can be seen as one possible workflow within a CODECHECK system, and clearly shares much in spirit. CODECHECK, however, specifically aims to give codecheckers recognition as reviewers. In our view, the reviewer role removes the possible conflict of interest while keeping the public acknowledgement. Specific to the field of mathematics, the RCR is also expected to apply a review of the software itself if the system it runs on cannot be evaluated by an independent party. The TOMS RCR creators concur with the importance of communication, expect collaboration between author and RCR reviewers, share the considerations around reviewer selection, and also put trust in reviewer judgement over numerical bit-wise perfection. A key difference is that for TOMS RCR, authors opt-in with an *RCR Review Request* and the RCR reports are published in the TOMS journal next to the actual papers.

Several journals provide special article types for reproductions of published papers. *Information Systems* has an invitation only Reproducibility Section for articles describing the reproducibility efforts of published articles, which are co-authored by the



A

CODECHECK certificate 2020-012

https://doi.org/10.5281/zenodo.3893617

B

Item	Value
Title	Report 23: State-level tracking of COVID-19 in the United States version 2 (28-05-2020)
Authors	H Juliette T Unwin , Swapnil Mistra, Valerie C Bradley, et al.
Reference	https://dx.doi.org/10.25561/79231
Codechecker	Stephen J. Eglen
Date of check	2020-06-14 14:00:00
Summary	R code for this paper shared with an earlier codecheck certifice (2020-011) from the same codebase.
Repository	https://github.com/sje30/covid19model-report23

Table 1: CODECHECK summary

C

Summary

The key findings in the "Report 23" from Imperial College were reproducible. I was able to re-run their code and generate qualitatively similar results to those shown in their manuscript. Differences in absolute values in results are due to the stochastic nature of the analysis. All code to reproduce the data worked as expected, and all key datasets were provided. I was able to regenrate the results in Figures 4–8 of the manuscript; code for Figures 1–3 was not available. (I did not attempt to go through all of the figures in the appendix, although Appendix D is an expanded version of Figure 6, showing summaries of each state.) The only significant complication in this reproduction was that some of the figures required the installation of system libraries. The final computations took about 17 hours on a multicore workstation.

In some cases, figures directly matched the layout in the manuscript; however, sometimes the figures have been post-processed as there are differences in layout. For example, in Figure 4 of the manuscript, the states have been re-ordered vertically in order of the value of R. Likewise, in Figure 8, the plots have been expanded out over three columns.

Page 1

D

Output	Comment	Size (b)
`usa/figures/rt_point__1006697.pdf`	Manuscript Figure 4	10841
`usa/figures/1006697_rt_map_chloropleth.pdf`	Manuscript Figure 5	77748
`usa/figures/WA_three_panel_1006697_.pdf`	Manuscript Figure 6 (Washington)	15409
`usa/figures/NY_three_panel_1006697_.pdf`	Manuscript Figure 6 (New York)	14703
`usa/figures/MA_three_panel_1006697_.pdf`	Manuscript Figure 6 (Massachusetts)	14472
`usa/figures/FL_three_panel_1006697_.pdf`	Manuscript Figure 6 (Florida)	14642
`usa/figures/CA_three_panel_1006697_.pdf`	Manuscript Figure 6 (California)	14798
`usa/figures/1006697_infectiousness_regions.pdf`	Manuscript Figure 7	38005
`usa/figures/WA_scenarios_56_0_20_40_1006697_deaths.pdf`	Manuscript Figure 8 (Washington)	10032
`usa/figures/NY_scenarios_56_0_20_40_1006697_deaths.pdf`	Manuscript Figure 8 (New York)	10582
`usa/figures/MA_scenarios_56_0_20_40_1006697_deaths.pdf`	Manuscript Figure 8 (Massachusetts)	10432
`usa/figures/FL_scenarios_56_0_20_40_1006697_deaths.pdf`	Manuscript Figure 8 (Florida)	10039
`usa/figures/CA_scenarios_56_0_20_40_1006697_deaths.pdf`	Manuscript Figure 8 (California)	10052

Table 2: Summary of output files generated

Page 2

E

CODECHECKER notes

The github repository https://github.com/ImperialCollegeLondon/covid19model was cloned, and renamed to "sje30/covid19model-report23". (I could not clone the project into the Github codecheckers group, as you cannot have two forks of the same project in the same organistion.)

This reproduction was performed after finishing the related certificate 2020-011; details of setting up the R environment are described in that certificate.

However, the R environment described was insufficient, as it didn't include *geofacet* and *rgdal* packages which needed system libraries to install. Once the sysadmin had installed extra libraries for unitdevs2 and gdal, I needed to run the following adhoc module provided locally:

```
module load ./gdal-2.1.2
```

```
install.packages("rgdal")
install.packages("geofacet")
install.packages("denstrip") #for plotting
```

An initial run of the FULL model didn't work because I had an older version of rstan package; this was upgraded to 2.19.3. The simulations were tested by running the simulation directly on a workstation:

```
time Rscript base-usa.r
```

Running the test mode took 41 minutes and generated outputs.

```
time Rscript base-usa.r -F
```

The final run time was 1020 minutes (17 hours). The code for reproducing figures 1,2 and 3 was not available in the repository, but all other key figures could be regenerated.

Page 3

F

Figure C1: Manuscript Figure 4

Page 4

Figure 4. **Annotated certificate 2020−012[51] (first four pages only).**

original authors and the reproducibility reviewer(s) (see https://www.elsevier.com/journals/information-systems/0306-4379/guide-for-authors).

Nature Machine Intelligence recently introduced a new type of article, the reusability report[82]. Inspired by the detailed and nuanced submissions to a reproducibility challenge, the reusability report focuses on the exploration of robustness and generalizability of the original paper's claims[82]. This answers the specific community's challenges around computational reproducibility and also values these kinds of contributions as independent publications, which goes beyond the goals of CODECHECK. The journal *Cortex* has a special article type *Verification Reports*, which are actually about replication of results and are very well designed/reasoned[83]. In a similar vein, the CODECHECK certificates could also be published as a special article type within journals.

Going beyond individual articles, the journal *ReScience C* publishes only replications, also requiring open code and replication by a third party. ReScience also relies on free infrastructure (GitHub and Zenodo).

For research with high stakes, where reproduction would be too weak and post-publication replication possibly too late because of policy impact, Benjamin-Chung *et al.*[84] propose *internal replication*. A workflow that has undergone internal replication would likely be of high quality and relatively easy to check. Similarly, internal CODECHECKs may be used, with the same limitations such as group think[84], to ensure reproducibility before submission. Such internal checks are professionalised in local reproduction services, such as CISER R-squared or YARD, or in communities such as Oxford's code review network.

Gavis and Donoho[85] propose a new discipline and infrastructure for reproducible computational research. Their specific packaging format, provenance record, and cryptographic *Verifiable Result Identifier* would indeed provide excellent reproducibility. However, the system is also complex and since its creation in 2011 we are not aware of any publisher using it; also, the system is not open source. In comparison, CODECHECK is less powerful but also much more flexible and less dependent on specific tools or infrastructure. If data and code are deposited properly, i.e., very unlikely to disappear, then the certificate's DOI is practically close to the cryptographic identifier.

Another platform for publishing results of reproductions is *SciGen.Report*. It is a community-run independent platform to foster communication on reproducibility. People can report on fully, partially, or failed reproductions of articles after publication.

CODECHECK is uniquely designed to be adopted across journals or events and to build a community of codecheckers. CODECHECK shares its interdisciplinary nature with other community initiatives concerned with reproducibility awareness, education, and support, such as ReproHack, Code Copilot, or Papers with Code. The latter recently announced a collaboration with the preprint server *arXiv* on providing data and code supplements for machine learning manuscripts and runs a reproducibility challenge. Likewise, different disciplines and journals provide reproducibility checklists, e.g., science and engineering[86] or GIScience[87], which naturally share some aspects while addressing particularities as well as addressing researchers from different fields. Regarding the education and guidance for authors, we see CODECHECK's role as referencing and linking educational efforts and helpful material, not as creating and maintaining such content.

Limitations

Isn't CODECHECK what peer review should be doing already? On the surface, yes, but peer reviewers are overburdened enough and asking them to do more work around peer review is not likely to succeed. When an editor (Tsuyoshi Miyakawa) requested raw data from n=41 authors before reviewing, 21 authors withdrew their manuscripts; 19 of the 20 remaining articles were rejected after peer review[88]. Such basic checks require effort from editors, yet they only rely on the availability of data files and the content of the paper. These availability checks can be enhanced by having more complex CODECHECKs request the code and then execute it. This might fall within idealistic expectations of peer review, but is rare. Establishing a CODECHECK process acknowledges that peer reviewing practices have been unable to adapt to the challenges of computational papers. The concept of a CODECHECK, just as the concepts of reproducible research and Open Science, may be transitional by nature. If the activities described here as being part of a CODECHECK are integrated into the publication process the initiative will have succeeded.

Should CODECHECK requirements be more demanding? CODECHECK by design does not require authors to provide (and sustain) an eternally functional workflow nor suggests a specific software stack or practical approach. Creating something that anyone can reproduce has been called a fool's errand and we tend to agree. However, the package of data, code, and documentation collaboratively created by authors and codecheckers is a snapshot of a working analysis that greatly increases the likelihood of a successful reproduction and the possibility that a workflow can be extended by third parties in the future, if they have access to suitable resources and matching skill set. Concrete implementations of CODECHECK workflows, especially for specific disciplines, may reify much more helpful guidelines for authors on how to create reproducibility packages. Our author-friendly "low bar" should not stay low forever, but cultural change takes time and the encouragement and guidance that CODECHECK, as part of the widely accepted peer review concept, can provide may eventually allow the bar to be raised much higher, e.g., with executable research compendia[89], "Whole Tales"[90], or continuous analysis[91]. However, considering that missing artefacts and lack of documentation have repeatedly been identified as key barriers to reproducibility (e.g., [29,92]), we would not underestimate the power of a simple check. For example, ModelDB curation policies require that only one figure need be manually reproduced[93], but that has not limited the usefulness nor success of the platform.

A codechecker does not fulfil the same role as a *statistical reviewer*, as it is applied by some journals in the biomedical domain (cf. [94,95]). The statistical reviewer evaluates the appropriateness of statistical methods[95] and can support topical reviewers if, e.g., complex methods or sophisticated variants of statistical tests are applied[94]. The codechecker may go equally deep into the review, but only if they have the expertise and time. We can imagine a tiered workflow where a codechecker could, just as a conventional reviewer could, recommend a detailed code review (see next paragraph) to the editor if they come upon certain issues while examining the work.

A codechecker does not conduct a *code review*. Code reviews are valuable to improve reproducibility and reusability, and their proponents even believe they can improve the research[96]. Code reviews, however, have quite different structural challenges and require even more resources. That said, a well-reviewed codebase is likely to be easier to codecheck, and the awareness of high-quality code raised through CODECHECK may lead to more support for code reviewing. Initiatives and journals that conduct software reviews independent of a specific publication or workflow include ROpenSci, PyOpenSci, and JOSS. Furthermore, the codechecker's task list is intentionally not overloaded with related issues such as ensuring proper citation of data and software or depositing material in suitable repositories. Nevertheless, codecheckers are free to highlight these issues.

How are failures during checks handled? We do not yet have a process for denoting if a reproduction fails, as our case-studies were all successful. In the case that a journal adopts CODECHECK for all submissions, the question remains as what to do if a check fails, after exhausting efforts between author and codechecker to reproduce the workflow. A negative comment in a CODECHECK certificate or a failed check does not necessarily mean the paper or research is bad (cf. discussion on negative comments in 17). We doubt that publicly reporting failures (i.e., the code would not run) will increase overall reproducibility, and may prohibit authors from sharing their work, which is always more desirable than nothing shared. Therefore, we recommend sharing interim reproduction efforts only with the authors, even if that means that volunteer efforts may go unnoticed if no certificate is published. Rosenthal *et al.*[97] discuss such incentives for different actors around the implementation of reproducibility. We see CODECHECK as one way for organisations to invest in reproducibility by creating incentives until reproducible computations become the norm.

Who will pay for the compute time? For papers that take significant compute time (days, not minutes), it is unclear who will pay for it. One must carefully consider the sustainability of rerunning computations and the environmental impact large calculations, such as training machine learning models, have. A pragmatic workaround is to request that authors provide a "toy" example, or small dataset that can be quickly analysed to demonstrate that the workflow runs correctly.

What about my proprietary software and sensitive data? Given the prevalence of proprietary software, e.g MATLAB, we pragmatically decided that we should accept code as long as we could find a machine with suitable licences to run it. However, this prohibits us from using open infrastructure for reproducibility (cf. 10,98) and requires the codechecker to have access to that particular software. Non-open software also considerably hampers reuse, especially by researchers from the global south. Therefore, allowing proprietary software is a compromise that should be reconsidered.

Solutions for proprietary and sensitive data exist. Authors can provide synthetic data (cf. 99), some data can effectively be redacted[100], and publishers or independent entities can provide infrastructure for sharing data and workflows confidentially[101] or with access to derived results but not raw data[99], i.e., data enclaves[102], or domains of reproducibility[103].

Can't someone cheat? Yes. We simply check that the code runs, not that is correct or sound science. This "mechanical" test is indeed a low bar. By having code and data openly deposited, third parties can later examine the code, and we hope that knowing the code will be open ensures that authors will not cheat. It also allows researchers, potentially with new methods, to look for errors. This is more effective than engaging in an arms race on building methods to detect malicious intent now with closed datasets and code. This is analogous to storing blood samples of sport champions today to possibly detect doping in the future with more sensitive methods (cf. 104). Another comparison that helped us define the scope of a CODECHECK is that we think of the codechecker as forensic photographer, capturing details so that an investigator may later scrutinise them.

Who's got time for more peer review? Agree; codechecking takes time that could otherwise be used for traditional peer review. However, a CODECHECK is different from peer review. First, the **technical nature** of a CODECHECK sets clear expectations and thereby time budget compared to conventional peer review. For example, authors are told what to provide and the codechecker can be told when to stop. Codecheckers can always directly ask the author when clarification is required, thereby increasing **efficiency**. Second, the specific skill set of a codechecker allows for **different groups** to participate in the review process. ECRs might be attracted to learn more about recent methods, peer review, and reproducibility practices. Research Software Engineers who might not regularly be involved in writing or reviewing papers might be interested in increasing their connection with scholarly practices. An extra codechecker may simplify the matchmaking an editor does when identifying suitable reviewers for a submission, as technical and topical expertise can be provided by different people. Third, recall that CODECHECKs should always be publicly available, unlike peer review reports. With code and workflows, the codechecker's feedback may directly impact and improve the author's work. The public certificates and contributions provide **peer recognition** for the codechecker. Fourth, we found that focusing on workflow mechanics and interacting with the author makes reproductions **educational**. It also is a different role and, as such, could be a welcome option for researchers to give back their time to the community.

While such benefits are also part of idealistic peer review, they are mostly hidden behind paraphrased anonymous acknowledgement.

Do workflows need to be codechecked multiple times? If a paper is checked at the start of peer review, it might need re-checking if the paper is modified during peer review. This is inevitable, and happened to us[51]. This is desirable though, if interactions between author, reviewer, and codechecker led to improvements. Checking the manuscript the second time is likely to be much less work than the first time.

What does it mean for a figure to be reproducible? Automatically detecting if a codechecker's results are "the same" as an author's is more challenging than it might appear. That is why we do not require results to be *identical* for a CODECHECK to pass but simply that the code runs and generates output files that the author claims. Stochastic simulations mean that often we will get different results, and even the same versions of libraries can generate outputs that differ by operating system[105]. While reproducibility practices can mitigate some of these problems, e.g., by using a seed, the flexibility of the human judgement is still needed, rather than bitwise reproducibility. The codechecker is free to comment on visible differences in outputs in their report.

Shouldn't the next step be more revolutionary? CODECHECK's approach is to acknowledge short-comings around computational reproducibility and to iteratively improve the current system. It remains to be proven whether this approach is welcomed broadly and if involving publishing stakeholders helps to further the cause. We have discussed more stringent rules at length, e.g. only considering fully free and open source software, diamond Open Access journals, but we eventually decided against them on the level of the principles. For the CODECHECK community process, documented at https://codecheck.org.uk/guide/community-process, and the volunteer codechecker community, these requirements can be reconsidered.

We have deliberated requiring modern technologies to support reproducibility (cf. 10), focusing instead on the human interface and the judgement of experienced researchers and developers as a more sustainable and flexible approach. All types of research can adopt CODECHECK due to its flexible design. CODECHECK could include automated scoring (e.g., 106), yet automation and metrics bear new risks. The focus of the CODECHECK principles on code execution allows journals and publishers to innovate on financial models and peer review practices at their own pace.

Conclusions and future work

CODECHECK works — we have created a considerable number of certificates to demonstrate it. The creation of certificates and interactions with authors and editors shaped the principles and the workflow and also confirmed the approach taken. This result corroborates findings from similar evaluations of reproducible computational research in journals and conferences. CODECHECKs increase transparency of the checked papers and can contribute to building trust in research findings. The set of shared principles and common name, through recognition value, will allow researchers to judge the level of scrutiny that results have faced. CODECHECK requires direct acknowledgement of the codechecker's contributions, not indirectly via citations of reproductions or informal credit.

CODECHECK however harbours the same limitations as peer review in general and is closely connected to larger disruptions and challenges in scholarly communication[7,107,108], including the tensions between commercial publishing and reviewers' often free labour, and a global pandemic that has jumbled up academic publishing and exposed a broader general audience to preprints[109]. Establishing CODECHECK workflows must be seen as interconnected with much larger issues in research, such as broken metrics or malpractice triggered by publication pressure[110,111]. We certainly do not want the binary attribute of "code works" to become a factor in bibliometric approaches for performance assessments. While developed for the current "paper"-centric publication process, the CODECHECK principles would also work well with novel publication paradigms, e.g., peer-reviewed computational notebooks[112], iterative and granular communication of research outputs, articles with live-code[113] such as eLife's ERA, decentralized infrastructure and public reviewer reputation systems[114], and completely new visions for scholarly communication and peer review, such as described by Amy J. Ko in *A modern vision for peer review*. An explicit segmentation of research steps could even make the focus of a CODECHECK easier by only checking the "analysis" sub-publication. The discovery of CODECHECKs could be increased by depositing certificates into public databases of reproductions, such as *SciGen.Report*. Public researcher profiles, such as ORCID, may consider different types of reviewer activity to capture how independent code execution contributes to science. Notably, the discussed limitations are largely self-imposed for easier acceptance and evolutionary integration, as to not break the current system and increase demands gradually without leaving practitioners behind.

A CODECHECK system, even if temporarily adopted as a sustainable transition towards more open publication and review practices, can contribute to increased trust in research outputs. Introducing CODECHECK should be informed by lessons learned from (introducing) open peer review[15]. Our conversations with publishers and editors indicate a willingness to adopt open practices like these, but that it is hard to innovate with legacy infrastructure and established practices.

More reproducible practices initiated by CODECHECKs could lead communities to reach a state where authors provide sufficient material and reviewers have acquired sufficient skills that peer reviewers will generally conduct a CODECHECK-level of checking; only in especially sophisticated cases will a specialised codechecker be needed. The main challenge for us remains getting journals to embrace the idea behind CODECHECK and to realise processes that conform to the principles, whether or not they use CODECHECK by name. We would be keen to use the flexibility of the principles and cooperate with journals to learn more about the advantages and yet unclear specific challenges – e.g do CODECHECKs really work better with open peer review? To facilitate the adoption, the CODECHECK badge is, intentionally, not branded beyond the checkmark and green colour and simply states "code works".

Future CODECHECK versions may be accompanied by studies to ensure codechecking does not fall into the same traps as peer review did[16] and to ensure positive change within the review system. This *cultural change*, however, is needed for the valuation of the efforts that go into proper evaluation of papers. Journals can help us to answer open questions in our system: What are crucial decisions or pain points? Can authors retract code/data once a CODECHECK has started? What variants of CODECHECKs will be most common? How will open CODECHECKs influence or codevelop with the scope and anonymity of conventional review over time?

The question of training codecheckers is also relevant. We expect a mentoring scheme within the CODECHECK community (experienced codecheckers will provide on-the-job training or serve as fall-back advisors). Codecheckers may also be found by collaborating with reproducible research initiatives such as ReproHack, ReproducibiliTea,[115], and Repro4Everyone,[116]. The initial reaction of researchers to these ideas shows that scholarly peer review should continue on the path towards facilitating sharing and execution of computational workflows.

Data availability

Zenodo: codecheckers/register: CODECHECK Register Deposit January 2021 http://doi.org/10.5281/zenodo.4486559[78].

This project contains the following underlying data:

- `register.csv`. List of all CODECHECK certificates with references to repositories and reports.

Data are available under the terms of the Creative Commons Attribution Share Alike license (CC-BY-SA 4.0 International).

Software availability

Codecheckers GitHub organisation: https://github.com/codecheckers

CODECHECK community on Zenodo: https://zenodo.org/communities/codecheck

`codecheck` R package: https://github.com/codecheckers/codecheck

Archived R package as at time of publication: http://doi.org/10.5281/zenodo.4522507[79]

License: MIT

Acknowledgements

We are grateful to the following individuals for discussions regarding the work presented here: Andy Collings, Melissa Harrison, Giuliano Maciocci, Naomi Penfold, Emmy Tsang (eLife), Rebecca Kirk (PLOS Computational Biology), Scott Edmunds (GigaScience), and Andrew Hufton (Scientific Data). Iain Davies and Yuhao (Sebastian) Wang developed code and example certificates. We thank Antonio Páez (Journal of Geographical Systems) for enabling CODECHECKs, Carlos Granell and Frank Ostermann for contributing certificates as reproducibility reviewers at the AGILE conference, and all authors of auditable workflows for their participation. We thank Celeste R. Brennecka from the Scientific Editing Service, University of Münster, for her editorial review.

References

1. Marwick B: **How computers broke science – and what we can do to fix it**. 2015.
Reference Source
2. Buckheit JB, Donoho DL: **WaveLab and Reproducible Research.** In Anestis Antoniadis and Georges Oppenheim, editors, *Wavelets and Statistics.* number 103 in Lecture Notes in Statistics, Springer New York, 1995; 55–81.
Publisher Full Text
3. Claerbout JF, Karrenbach M: **Electronic documents give reproducible research a new meaning.** In *SEG Technical Program Expanded Abstracts 1992.* SEG Technical Program Expanded Abstracts, Society of Exploration Geophysicists, 1992; 601–604.
Publisher Full Text
4. Vines TH, Albert AYK, Andrew RL, *et al.*: **The availability of research data declines rapidly with article age.** *Curr Biol.* 2014; **24**(1): 94–97.
PubMed Abstract | Publisher Full Text
5. Barnes N: **Publish your computer code: it is good enough.** *Nature.* 2010; **467**(7317): 753.
PubMed Abstract | Publisher Full Text
6. Markowetz F: **Five selfish reasons to work reproducibly.** *Genome Biol.* 2015; **16**: 274.
PubMed Abstract | Publisher Full Text | Free Full Text
7. Fyfe A: **Mission or money?** *Septentrio Conference Series.* Keynote at 14th Munin Conference on Scholarly Publishing 2019. 2019.
Publisher Full Text
8. Barba LA: **Terminologies for Reproducible Research.** *arXiv: 1802.03311 [cs].* 2018.
Reference Source
9. Jupyter P, Bussonnier M, Forde J, *et al.*: **Binder 2.0 - Reproducible, interactive, sharable environments for science at scale.** *Proceedings of the 17th Python in Science Conference.* 2018; 113–120.
Publisher Full Text
10. Konkol M, Nüst D, Goulier L: **Publishing computational research - a review of infrastructures for reproducible and transparent scholarly communication.** *Res Integr Peer Rev.* 2020; **5**(1): 10.
PubMed Abstract | Publisher Full Text | Free Full Text
11. Stodden V, Bailey DH, Borwein J, *et al.*: **Setting the Default to Reproducible: Reproducibility in Computational and Experimental Mathematics.** Technical report, The Institute for Computational and Experimental Research in Mathematics, 2013.
Reference Source
12. Christian TM, Gooch A, Vision T, *et al.*: **Journal data policies: Exploring how the understanding of editors and authors corresponds to the policies themselves.** *PLoS One.* 2020; **15**(3): e0230281.
PubMed Abstract | Publisher Full Text | Free Full Text
13. Nosek BA, Spies JR, Motyl M: **Scientific Utopia: II. Restructuring Incentives and Practices to Promote Truth Over Publishability.** *Perspect Psychol Sci.* 2012; **7**(6): 615–631.
PubMed Abstract | Publisher Full Text
14. Nüst D: **CODECHECK certificate 2020-025.** *Zenodo.* 2020.
Publisher Full Text
15. Ross-Hellauer T, Görögh E: **Guidelines for open peer review implementation.** *Res Integr Peer Rev.* 2019; **4**(1): 4.
PubMed Abstract | Publisher Full Text | Free Full Text
16. Tennant JP, Ross-Hellauer T: **The limitations to our understanding of peer**

review. *Res Integr Peer Rev.* 2020; **5**(1): 6.
PubMed Abstract | Publisher Full Text | Free Full Text

17. Quintana D, Heathers J: **Everything Hertz 123: Authenticated anonymity (with Michael Eisen).** *Open Science Framework.* 2020.
Publisher Full Text

18. Bouffler B: **Keynote at deRSE 2019: Delivering on the promise of Research Computing.** *TIB AV- PORTAL.* Video recording published in TIB AV-PORTAL. 2019.
Publisher Full Text

19. Roesch EB, Rougier N: **New journal for reproduction and replication results.** *Nature.* 2020; **581**(7806): 30.
PubMed Abstract | Publisher Full Text

20. Chambers C: **Registered Reports.** Publisher: *OSF.* 2019.
Reference Source

21. Davies I: **CODECHECK certificate 2020-013.** *Zenodo.* 2020.
Publisher Full Text

22. Stark PB: **Before reproducibility must come preproducibility.** *Nature.* 2018; **557**(7707): 613.
PubMed Abstract | Publisher Full Text

23. Barba LA: **Praxis of Reproducible Computational Science**. 2018.
Publisher Full Text

24. Davies NG, Kucharski AJ, Eggo RM, *et al.*: **Effects of non-pharmaceutical interventions on COVID-19 cases, deaths, and demand for hospital services in the UK: a modelling study.** *Lancet Public Health.* 2020; **5**(7): e375–e385.
PubMed Abstract | Publisher Full Text | Free Full Text

25. Kucharski AJ, Klepac P, Conlan AJK, *et al.*: **Effectiveness of isolation, testing, contact tracing, and physical distancing on reducing transmission of SARS-CoV-2 in different settings: a mathematical modelling study.** *Lancet Infect Dis.* 2020; **20**(10): 1151–1160.
PubMed Abstract | Publisher Full Text | Free Full Text

26. Chawla DS: **Critiqued coronavirus simulation gets thumbs up from code-checking efforts.** *Nature.* 2020; **582**(7812): 323–324.
PubMed Abstract | Publisher Full Text

27. Eglen SJ: **CODECHECK certificate 2020-010.** *Zenodo.* 2020.
Publisher Full Text

28. Nüst D, Ostermann F, Hofer B, *et al.*: **Reproducible Publications at AGILE Conferences**. 2019.
Reference Source

29. Nüst D, Ostermann FO, Granell C, *et al.*: **Improving reproducibility of geospatial conference papers – lessons learned from a first implementation of reproducibility reviews.** *Septentrio Conference Series.* 2020.
Publisher Full Text

30. Eglen SJ: **CODECHECK certificate 2020-001.** *Zenodo.* 2020.
Publisher Full Text

31. Piccolo SR, Lee TJ, Suh E, *et al.*: **ShinyLearner: A containerized benchmarking tool for machine-learning classification of tabular data.** *Gigascience.* 2020; **9**(4): giaa026.
PubMed Abstract | Publisher Full Text | Free Full Text

32. Eglen SJ, Nüst D: **CODECHECK certificate 2020-002.** *Zenodo.* 2020.
Publisher Full Text

33. Hancock PJB, Baddeley RJ, Smith LS: **The principal components of natural images.** *Network: Computation in Neural Systems.* 1992; **3**(1): 61–70.
Publisher Full Text

34. Daniel N: **CODECHECK certificate 2020-003.** *Zenodo.* 2020.
Publisher Full Text

35. Hopfield JJ: **Neural networks and physical systems with emergent collective computational abilities.** *Proc Natl Acad Sci U S A.* 1982; **79**(8): 2554.
PubMed Abstract | Publisher Full Text | Free Full Text

36. Nüst D: **CODECHECK certificate 2020-004.** *Zenodo.* 2020.
Publisher Full Text

37. Barto AG, Sutton RS, Anderson CW: **Neuronlike adaptive elements that can solve difficult learning control problems.** *IEEE Trans Syst Man Cybern.* 1983; **SMC-13**(5): 834–846.
Publisher Full Text

38. Eglen SJ: **CODECHECK certificate 2020-005.** *Zenodo.* 2020.
Publisher Full Text

39. Larisch R: **[Re] Connectivity reflects coding a model of voltage-based STDP with homeostasis.** *ReScience C.* 2019; **5**(3).
Publisher Full Text

40. Eglen SJ: **CODECHECK certificate 2020-006.** *Zenodo.* 2020.
Publisher Full Text

41. Detorakis G: **[Re] A Generalized Linear Integrate-And-Fire Neural Model Produces Diverse Spiking Behaviors.** *ReScience C.* 2017; **3**(1): #7.
Publisher Full Text

42. Eglen SJ: **CODECHECK certificate 2020-007.** *Zenodo.* 2020.

43. Hathway P, Goodman DFM: **[Re] Spike Timing Dependent Plasticity Finds The Start Of Repeating Patterns In Continuous Spike Trains.** *ReScience C.* 2018; **4**(1): #6.
Publisher Full Text

44. Eglen SJ: **CODECHECK certificate 2020-008.** *Zenodo.* 2020.
Publisher Full Text

45. Davies NG, Kucharski AJ, Eggo RM, *et al.*: **Effects of non-pharmaceutical interventions on COVID-9 cases, deaths, and demand for hospital services in the UK: a modelling study.** 2020.
Reference Source

46. Eglen SJ: **CODECHECK certificate 2020-009.** *Zenodo.* 2020.
Publisher Full Text

47. Kucharski AJ, Klepac P, Conlan AJK, *et al.*: **Effectiveness of isolation, testing, contact tracing and physical distancing on reducing transmission of sars-cov-2 in different settings: a mathematical modelling study.** 2020.
Reference Source

48. Ferguson N, Laydon D, Nedjati Gilani G, *et al.*: **Report 9: Impact of non-pharmaceutical interventions (NPIs) to reduce COVID19 mortality and healthcare demand.** 2020.
Publisher Full Text

49. Eglen SJ: **CODECHECK certificate 2020-011.** *Zenodo.* 2020.
Publisher Full Text

50. Flaxman S, Mishra S, Gandy A, *et al.*: **Estimating the effects of non-pharmaceutical interventions on COVID-19 in Europe.** *Nature.* 2020; **584**(7820): 257–261.
PubMed Abstract | Publisher Full Text

51. Eglen SJ: **CODECHECK certificate 2020-012.** *Zenodo.* 2020.
Publisher Full Text

52. Unwin H, Mishra S, Bradley VC, *et al.*: **Report 23: State-level tracking of COVID-19 in the United States.** 2020.
Publisher Full Text

53. Unwin HJT, Mishra S, Bradley VC, *et al.*: **State-level tracking of COVID-19 in the United States.** *Nat Commun.* 2020; **11**(1): 6189.
PubMed Abstract | Publisher Full Text | Free Full Text

54. Spitschan M, Garbazza C, Kohl S, *et al.*: **Rest-activity cycles and melatonin phase angle of circadian entrainment in people without cone-mediated vision.** *bioRxiv.* 2020.
Publisher Full Text

55. Davies I: **CODECHECK certificate 2020-014.** 2020.
Publisher Full Text

56. Sadeh S, Clopath C: **Patterned perturbation of inhibition can reveal the dynamical structure of neural processing.** *elife.* 2020; **9**: e52757.
PubMed Abstract | Publisher Full Text | Free Full Text

57. Davies I: **CODECHECK certificate 2020-015.** See file LICENSE for license of the contained code. 2020.
Publisher Full Text

58. Liou J, Smith EH, Bateman LM, *et al.*: **A model for focal seizure onset, propagation, evolution, and progression.** *eLife.* 2020; **9**: e50927.
PubMed Abstract | Publisher Full Text | Free Full Text

59. Nüst D: **CODECHECK certificate 2020-016.** 2020.
Publisher Full Text

60. Brunsdon C, Comber A: **Opening practice: supporting reproducibility and critical spatial data science.** *J Geogr Syst.* 2020.
Publisher Full Text

61. Nüst D: **CODECHECK certificate 2020-017**. 2020.
http://www.doi.org/10.5281/zenodo.4003848

62. Bivand RS: **Progress in the r ecosystem for representing and handling spatialdata.** *J Geogr Syst.* 2020.
Publisher Full Text

63. Nüst D: **Reproducibility review of: Integrating cellular automata and discrete global grid systems: a case study into wildfire modelling**. 2020.
Publisher Full Text

64. Hojati M, Robertson C: **Integrating cellular automata and discrete global grid systems: a case study into wildfire modelling.** *AGILE: GIScience Series.* 2020; **1**: 1–23.
Publisher Full Text

65. Nüst D, Granell C: **Reproducibility review of: What to do in the meantime: A service coverage analysis for parked autonomous vehicles**. 2020.
Publisher Full Text

66. Illium S, Friese PA, Müller R, *et al.*: **What to do in the meantime: A service coverage analysis for parked autonomous vehicles.** *AGILE: GIScience Series.* 2020; **1**: 1–15.
Publisher Full Text

67. Nüst D, Ostermann F: **Reproducibility review of: Window operators for processing spatio- temporal data streams on unmanned vehicles**. 2020.
Publisher Full Text

68. Werner T, Brinkhoff T: **Window operators for processing spatio-temporal data streams on unmanned vehicles.** *AGILE: GIScience Series.* 2020; **1**: 1–23.
Publisher Full Text

69. Ostermann F, Nüst D: **Reproducibility review of: Comparing supervised learning algorithms for spatial nominal entity recognition**. 2020.
Publisher Full Text

70. Medad A, Gaio M, Moncla L, *et al.*: **Comparing super- vised learning algorithms for spatial nominal entity recognition.** *AGILE: GIScience Series.* 2020; **1**: 1-18.
Publisher Full Text

71. Nüst D: **Reproducibility review of: Extracting interrogative intents and**

concepts from geo-analytic questions. 2020.
Publisher Full Text

72. Xu H, Hamzei E, Nyamsuren E, *et al.*: **Extracting interrogative intents and concepts from geo-analytic questions.** *AGILE: GIScience Series.* 2020; **1**: 1–21.
Publisher Full Text

73. Ostermann F, Nüst D: **Reproducibility review of: Tracking hurricane dorian in gdelt and twitter**. 2020.
Publisher Full Text

74. Owuor I, Hochmair HH, Cvetojevic S: **Tracking hurricane dorian in GDELT and twitter.** *AGILE: GIScience Series.* 2020; **1**: 1–18.
Publisher Full Text

75. Eglen SJ: **CODECHECK certificate 2020-024.** *Zenodo.* 2020.
Publisher Full Text

76. Vieira DM, Fernandes C, Lucena C, *et al.*: **Driftage: a multi-agent system for concept drift detection and an application on electromyography.** *Gigascience.* (under review), 2020.
Reference Source

77. Carrer F, Kossowski TM, Wilk J, *et al.*: **The application of Local Indicators for Categorical Data (LICD) to explore spatial dependence in archae- ological spaces.** *J Archaeol Sci.* 2021; **126**: 105306.
Publisher Full Text

78. Nüst D, Eglen S, Davies L: **codecheckers/register: CODECHECK Register Deposit January 2021.** (Version 2021-01) [Data set]. *Zenodo.* 2021.
http://www.doi.org/10.5281/zenodo.4486559

79. Eglen S, Nüst D: **codecheckers/codecheck: codecheck R package version 0.1.0**. 2021.
Reference Source

80. Blondel E: **zen4R: Interface to 'Zenodo' REST API. R package version 0.4-2**. 2020.
Reference Source

81. Heroux MA: **Editorial: ACM TOMS Replicated Computational Results Initiative.** *ACM Trans Math Softw.* 2015; **41**(3): 13: 1–13: 5.
Publisher Full Text

82. **Research, reuse, repeat.** *Nat Mach Intell.* 2020; **2**(12): 729–729.
Publisher Full Text

83. Chambers CD: **Verification Reports: A new article type at Cortex.** *Cortex.* 2020; **129**: A1–A3.
Publisher Full Text

84. Benjamin-Chung J, Colford JM, Mertens A, *et al.*: **Internal replication of computational workflows in scientific research.** *Gates Open Res.* 2020; **4**: 17.
PubMed Abstract | Publisher Full Text | Free Full Text

85. Gavish M, Donoho D: **A Universal Identifier for Computational Results.** *Procedia Comput Sci.* 2011; **4**: 637–647.
Publisher Full Text

86. Rosenberg DE, Filion Y, Teasley R, *et al.*: **The Next Frontier: Making Research More Reproducible.** *J Water Resour Plann Manage.* 2020; **146**(6): 01820002.
Publisher Full Text

87. Nüst D, Ostermann F, Sileryte R, *et al.*: **AGILE Reproducible Paper Guidelines**. *OSF.* 2019.
Publisher Full Text

88. Miyakawa T: **No raw data, no science: another possible source of the reproducibility crisis.** *Mol Brain.* 2020; **13**(1): 24.
Publisher Full Text

89. Nst D, Konkol M, Pebesma E, *et al.*: **Opening the Publication Process with Executable Research Compendia.** *D-Lib Magazine.* 2017; **23**(1/2).
Publisher Full Text

90. Brinckman A, Chard K, Gaffney N, *et al.*: **Computing environments for reproducibility: Capturing the "Whole Tale".** *Future Gener Comput Syst.* 2018.
Publisher Full Text

91. Beaulieu-Jones BK, Greene CS: **Reproducibility of computational workflows is automated using continuous analysis.** *Nat Biotechnol.* 2017; **35**(4): 342–346. ISSN 1546-1696.
PubMed Abstract | Publisher Full Text | Free Full Text

92. Stagge JH, Rosenberg DE, Abdallah AM, *et al.*: **Assessing data availability and research reproducibility in hydrology and water resources.** *Sci Data.* 2019; **6**(1): 190030. ISSN 2052-4463.
PubMed Abstract | Publisher Full Text | Free Full Text

93. McDougal RA, Bulanova AS, Lytton WW: **Reproducibility in Computational Neuroscience Models and Simulations.** *IEEE Trans Biomed Eng.* 2016; **63**(10): 2021–2035. ISSN 0018-9294.
PubMed Abstract | Publisher Full Text | Free Full Text

94. Petrovečki M: **The role of statistical reviewer in biomedical scientific journal.** *Biochemia Medica.* 2009; **19**(3): 223–230.
Publisher Full Text

95. Greenwood DC, Freeman JV: **How to spot a statistical problem: advice for a non-statistical reviewer.** *BMC Med.* 2015; **13**(1): 270. ISSN 1741-7015.
PubMed Abstract | Publisher Full Text | Free Full Text

96. Petre M, Wilson G: **Code Review For and By Scientists.** *arXiv: 1407.5648 [cs].* arXiv: 1407.5648. 2014.
Reference Source

97. Rosenthal P, Mayer R, Page K, *et al.*: **Incentives and barriers to reproducibility: Investments and returns.** In Juliana Freire, Norbert Fuhr, and Andreas Rauber, editors, *Reproducibility of data-oriented experiments in e-Science*. of Dagstuhl reports. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2016; **6**: 148–151. ISSN: 2192-5283.

98. Perkel JM: **Make code accessible with these cloud services.** *Nature.* 2019; **575**(7781): 247–248.
PubMed Abstract | Publisher Full Text

99. Shannon J, Walker K: **Opening GIScience: A process-based approach.** *Int J Geogr Inf Sci.* 2018; **32**(10): 1911–1926. ISSN 1365-8816.
Publisher Full Text

100. O'Loughlin J, Raento P, Sharp JP, *et al.*: **Data ethics: Pluralism, replication, conflicts of interest, and standards in *Political Geography*.** *Political Geography.* 2015; **44**: A1–A3. ISSN 0962-6298.
Publisher Full Text

101. Pérignon C, Gadouche K, Hurlin C, *et al.*: **Certify reproducibility with confidential data.** *Science.* 2019; **365**(6449): 127–128. ISSN 0036-8075, 1095-9203.
PubMed Abstract

102. Foster I: **Research Infrastructure for the Safe Analysis of Sensitive Data.** *Ann Am Acad Pol Soc Sci.* 2018; **675**(1): 102–120. ISSN 0002-7162.
Publisher Full Text

103. Harris R, O'Sullivan D, Gahegan M, *et al.*: **More bark than bytes? Reflections on 21+ years of geocomputation.** *Environ Plan B Urban Anal City Sci.* 2017; **44**(4): 598–617. ISSN 2399-8083.
Publisher Full Text

104. Quintana D, Heathers J: **Everything Hertz 97: Slow science.** *Open Science Framework.* 2019.
Publisher Full Text

105. Gronenschild EHBM, Habets P, Jacobs HIL, *et al.*: **The effects of FreeSurfer version, workstation type, and macintosh operating system version on anatomical volume and cortical thickness measurements.** *PLoS One.* 2012; **7**(6): e38234. ISSN 1932-6203.
PubMed Abstract | Publisher Full Text | Free Full Text

106. Menke J, Roelandse M, Ozyurt B, *et al.*: **The Rigor and Transparency Index Quality Metric for Assessing Biological and Medical Science Methods.** *iScience.* 2020; **23**(11): 101698. ISSN 2589-0042.
PubMed Abstract | Publisher Full Text | Free Full Text

107. Eglen SJ, Mounce R, Gatto L, *et al.*: **Recent developments in scholarly publishing to improve research practices in the life sciences.** *Emerg Top Life Sci.* 2018; **2**(6): 775–778. ISSN 2397-8554, 2397-8562.
PubMed Abstract | Publisher Full Text | Free Full Text

108. Tennant JP, Crane H, Crick T, *et al.*: **Ten Hot Topics around Scholarly Publishing.** *Publications.* 2019; **7**(2): 34.
Publisher Full Text

109. Munafo M: **What you need to know about how coronavirus is changing science**. 2020.
Reference Source

110. Piwowar H: **Altmetrics: Value all research products.** *Nature.* 2013; **493**(7431): 159. ISSN 1476-4687.
PubMed Abstract | Publisher Full Text

111. Nosek BA, Alter G, Banks GC, *et al.*: **SCIENTIFIC STANDARDS. Promoting an open research culture.** *Science.* 2015; **348**(6242): 1422–1425. ISSN 0036-8075, 1095-9203.
PubMed Abstract | Publisher Full Text | Free Full Text

112. EarthCube: **New EarthCube Peer-Reviewed Jupyter Notebooks. Now Available.** 2020.
Reference Source

113. Perkel JM: **Pioneering 'live-code' article allows scientists to play with each other's results.** *Nature.* 2019; **567**(7746): 17–18.
PubMed Abstract | Publisher Full Text

114. Tenorio-Fornés A, Jacynycz V, Llop-Vila D, *et al.*: **Towards a Decentralized Process for Scientific Publication and Peer Review using Blockchain and IPFS.** 2019. ISBN 978-0-9981331-2-6. Accepted: 2019-01-03T00:29:12Z.
Publisher Full Text

115. Fitzgibbon L, Brady D, Haffey A, *et al.*: **Brewing up a storm: developing Open Research culture through ReproducibiliTea.** *Report.* Central Archive at the University of Reading. 2020.
Publisher Full Text

116. Auer S, Haelterman N, Weissgerber T, *et al.*: **Reproducibility for everyone: a community-led initiative with global reach in reproducible research training.** *OSF Preprints.* 2020.
Publisher Full Text

Open Peer Review

Current Peer Review Status: ? ✓

Version 1

Reviewer Report 21 April 2021

https://doi.org/10.5256/f1000research.54932.r82472

✓ **Sarah Gibson**
The Alan Turing Institute, London, UK

Paper Summary
This paper outlines a set of principles and a community of practice for verifying computational analyses can be run and research artefacts reproduced as part of, or in addition to, traditional peer review processes. The ongoing scientific reproducibility crisis and current lack of many (or any) standards for checking computational research in the publishing industry makes this an important, new framework to share with the community.
The authors demonstrate a deep and thoughtful knowledge of the cultural barriers surrounding such technological checks for peer review, such as time, expertise, and bitwise comparative reproducibility. They acknowledge that the specific incarnation of the CODECHECK practice outlined in this paper is limited to provide a low barrier for entry in order to encourage adoption, but do detail the scope in which such a workflow could be adapted and built upon to raise that bar and perform more stringent checks. Specifically, the principles are not technology-based to allow for flexibility in the complexity and domain of computational research to be checked. I particularly appreciated the authors' recommendation/suggestion that CODECHECKs become a platform for engaging Early Career Researchers in the peer review process.
Alongside CODECHECK's own workflows (which are openly published on GitHub and Zenodo), the paper outlines many similar and related initiatives that fall within the CODECHECK framework providing a wealth of examples for the community to draw inspiration from when designing and applying their own CODECHECK workflows.

Is the rationale for developing a new method clearly explained?
The authors show a deep knowledge of the pitfalls of traditional peer review of static research artefacts and clearly identify and outline the rationale for a peer review-like system capable of assessing computation-based research.

Is the description of the method technically sound?
I'm going to answer a slightly different question of "Is the description of the method *culturally* sound?" This is because the authors have intentionally not provided a technological methodology for completing a CODECHECK so as to avoid vendor lock-in (e.g. cloud platform providers) and to

provide flexibility for applying the methodology to a range of computational research domains. Instead, the focus of the methodology is on building a community of practice around having code mechanically checked by someone with comparable technical expertise from outside the project. The authors demonstrate a considerate knowledge of the burden of verifying computational reproducibility on both authors and peer reviewers and aim, not to increase this burden, but to provide an entry point into a world where checking research code can be run and produces the artefacts as they are presented in the paper is normalised. I think their recommended approach focussing on communication between codecheckers and authors, codecheckers will check and not fix, and codecheckers being an additional role to the traditional peer reviewer will aid early adoption of this framework.

Are sufficient details provided to allow replication of the method development and its use by others?
The concept of CODECHECK is intentionally presented as a set of principles and example workflows, as opposed to fixed, step-by-step actions, to allow for flexibility across computational complexity and research domains. The principles, example workflow, and potential variations under this framework are explained in depth and examples of workflows that fall under the CODECHECK framework from other publishers and/or conferences are provided, alongside CODECHECK's own community. From this wealth of detail, I believe that others would be able to replicate, adapt and apply a CODECHECK-like workflow in their journal or community.

Are the conclusions about the method and its performance adequately supported by the findings presented in this article?
It is encouraging to see that the community feedback from authors and publishers shaped the workflow and principles that uphold CODECHECK and a number of certificates have already been issued under this framework. This shows that the workflow of a CODECHECK as outlined in the paper is achievable in partnership with current peer review operations. However, I would like to see the impact of the CODECHECK certificates issued. Is there any community feedback on the transparency and reusability of research published with CODECHECK certificates? This is perhaps too big of an ask this early in the initiative as research reuse and citations are independent factors of the publication and peer review of this specific paper - but I'd still be interested in any insights the authors have to offer on this topic.

Is the rationale for developing the new method (or application) clearly explained?
Yes

Is the description of the method technically sound?
Yes

Are sufficient details provided to allow replication of the method development and its use by others?
Yes

If any results are presented, are all the source data underlying the results available to ensure full reproducibility?
No source data required

Are the conclusions about the method and its performance adequately supported by the findings presented in the article?
Partly

Competing Interests: I am a champion of reproducible research and an operator of mybinder.org which was explicitly mentioned in the paper.

Reviewer Expertise: As a Research Software Engineer, I don't have a specific area of research any more. I have skills and expertise in software best practices, computational reproducibility and cloud computing infrastructure, which I have gained through the open source communities Project Binder (running mybinder.org) and The Turing Way (a pedagogical resource which includes a volume on reproducibility) alongside working on a range of projects within the Alan Turing Institute.

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Reviewer Report 19 April 2021

https://doi.org/10.5256/f1000research.54932.r82470

?

Nicolas P. Rougier
Inria Bordeaux Sud-Ouest, Talence, France

In this article, authors propose to implement a procedure to check for the code accompanying a submission to a journal. To do so, they describe a pipeline made of 6 steps that ultimately lead to the delivery of a code check certificate meaning that someone external to the author's lab has managed to re-run the code. At this point, no checking that the results are correct is necessary. The authors already
issued several codecheck certificates in different disciplines. I find the idea really nice and certainly necessary but I've a few questions (even though some of them are already addressed in the "limitations" section). Given the structure of the paper, I'll just list my questions here:

- How does CODECHECK compare to ACM Artifact reviews badges? (https://www.acm.org/publications/policies/artifact-review-and-badging-current)

- What would be the incentive for someone to code check the code? Being aware of the increasing difficulty in finding reviewers, I don't think it would be easy to recruit people to perform a task that can rapidly become very technical and time consuming.

- How do you handle the case when specific hardware is necessary (e.g. NVidia GPU)? Is it documented somewhere such that code-checkers might first verify if they have the necessary hardware to run the code?

- How do you establish a check has failed? For example, what happens if a code-checker gets a segfault (for some unknown reason) and the author is unable to help. Is it deemed failed?
- Who will pay for the computing resources needed to run heavy simulations and/or to acquire necessary software such as e.g. Matlab? When a simulation consumes a lot of resources, it might wise to give the checker access to computing resources. This could be paid for by the journal.
- I did not see in the report example a description of the environment necessary to run the software. How did you solve the "dependency hell"? Since the code might break at some point in the future because of incompatibility in some libraries or environments, it would be necessary to have a mechanism describing the running environment such that it can be re-run later.
- What do you recommend if the reviews are both excellents but the code check failed? Does this mean the paper is blocked until code check passes or rejected or else?
- The code check proposal is close to some extents to the Journal of Open Science Software where each reviewer is assigned a list of things to check during the review. Do authors consider this pipeline when establishing their own pipeline?
- To what extent the codecheck certificate can be updated automatically via some kind of "manual continuous-integration"? I mean that when reading a paper online, would it be possible to click a button to test if the code still runs considering the latest versions of libraries? (for example, the certificate has been issued for Python 2 but I want to know if this is usable with Python 3).
- When you look at journals advertising open data policies, it is unfortunately not rare to find articles in these same journals without the actual data. Do you have some suggestion for educating editors to actually enforce the code check a journal adopt it?

Some suggestions:

- The badge that is delivered would need some time information since the check is valid at one point in time (with a given software stack) and does not guarantee future runs.
- For specialized journals, you could consider to offer a common generic environment where a code could be first tested. It this fails, then you would need only to slightly modify the environment to add missing dependencies. For example, in neuroscience, a Neuro Debian would probably suit the needs of a large number of models.
- - As editor-in-chief of ReScience C, I would like to inform authors that the journal now accepts "reproduction report". The idea it to try to re-run the code accompanying a published article and to report if it succeeded or failed. Our own procedure to check for reproduction is not standardized and we'll certainly benefit from the code check initiative.

Overall, it's nice to have a clean description of a pipeline to check for code even though some questions need to be addressed. Also, I'm not too confident that journals will adopt it immediately and I'm afraid such initiative will take time to be generalized. But we have to start somewhere.

Is the rationale for developing the new method (or application) clearly explained?
Yes

Is the description of the method technically sound?
Partly

Are sufficient details provided to allow replication of the method development and its use by others?
Yes

If any results are presented, are all the source data underlying the results available to ensure full reproducibility?
Yes

Are the conclusions about the method and its performance adequately supported by the findings presented in the article?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Computational Neuroscience, Open Science

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Comments on this article

Version 1

Reader Comment 01 Apr 2021
Cassio Amorim, CJS Inc., SciGen.Report, Kyoto, Japan

Very informative pre-print. I have 3 points to raise that the authors may or may not find useful, 2 suggestions and 1 comment, which the authors may adopt or ignore as they see fit.

1. Regarding Fig. 1, I think the left side would be better if at least vaguely structured. I believe we all acknowledge that science is messy, but finding structures and patterns in this mess is research. So, instead of a cloud with keywords, I would take some kind of blocks connected somehow, and the arrow with "sharing" leaving the whole set. Let me try to text sketch the whole image I have below, as a rough structure. I do not understand what "Stats" indicates, though, so I'm skipping it.

Also, I'm avoiding arrows for I assume directions may vary on each case, e.g., data derives from code/model (ab initio) or code derives from data (analysis)?

```
/ Data /---/ Code, models, nb / |
          |                          |---------Sharing-------->
          |                          |
    / Results /                      |
```

2. I appreciate the impact of the conclusion "CODECHECK works" and would even finish with a period for impact myself, but I'm not sure the trailing explanation sustains it. It is one thing when Richard Dawkins says "[Science] works. Planes fly, cars drive, computers compute." It does not hit me the same with "CODECHECK works. We made certificates." I'd expect concrete consequences there (and I believe there are). However, it is not to say there is any problem in the conclusion itself. I just think something more on the lines of "CODECHECK works. From AI to pandemic modeling, we verify meaningful codes and certify their reproducibility (amidst the gambling chaos we live in)." In other words, spelling out the impact of "we have created a considerable number of certificates" (what kind? what for?) would make it better in my opinion. The word-crafting art there, of course, relies on the authors' taste.

3. Just a (personal) comment about the mention of bitwise reproducibility in the "auditable research" section. I personally have a hard time understanding the concept. Considering float point arithmetics implementation (e.g., https://docs.nvidia.com/cuda/floating-point/index.html), one would need the same code, data *and* hardware+software. Such demand is so punctual that I fail to see how it is even feasible at scale. Certainly, it makes the strictest definition of reproducibility, just like an ideal gas is the "strictest" gas, but as I do not expect even Hellilum to behave as point-like particles always, I wouldn't expect such a degree of reproducibility from every research (notably not from HPC). But again, just my view on the matter, the authors may or may not want to add a few words to the auditable research session for that, whichever the case being comprehensible.

Competing Interests: I have discussed possible collaboration with Daniel Nust before, yet unrealized on the date of this comment submission.

14 Geospatial metadata for discovery in scholarly publishing

Authors Tom Niers and Daniel Nüst (20%)

Venue Extended abstract at *The 15th Munin Conference on Scholarly Publishing 2020* 10.7557/5.5590

Date 09/2020

Geospatial Metadata for Discovery in Scholarly Publishing

Tom Niers and Daniel Nüst

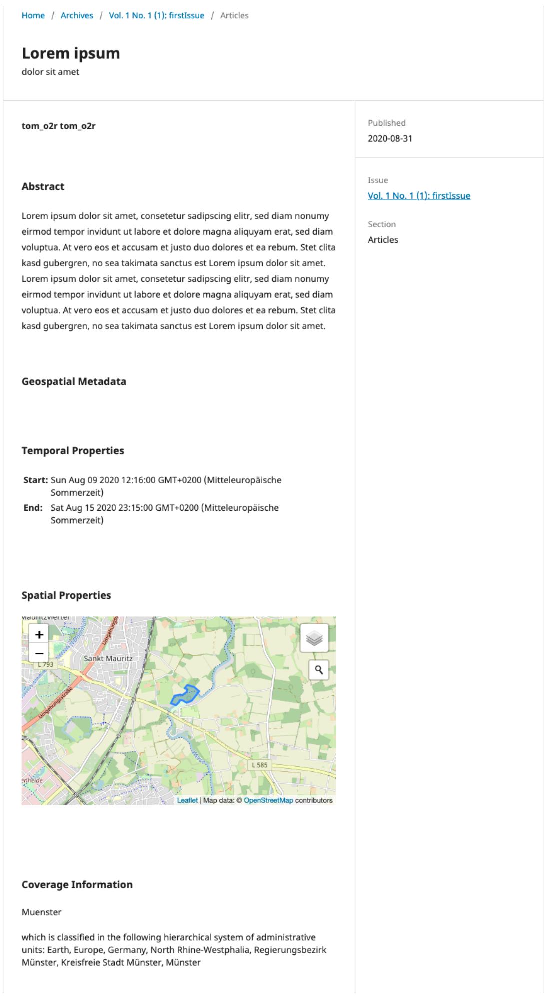
Institute for Geoinformatics, University of Münster
tom.niers@wwu.de, daniel.nuest@wwu.de

15th Munin Conference on Scholarly Publishing,
18–19 November 2020

10.7557/5.5590

Almost every scientific article that refers to existing regions of the earth, contains "[...] a narrative description of the study area" (Karl, 2019). At the same time, Shapiro and Báldi (2012) found that more than a quarter of articles omit maps and coordinates and only use vague descriptions albeit the relevance of location for said articles' content. Geospatial metadata can help to detect biases in research coverage (Karl et al., 2013; Young & Lutters, 2017), to filter search results for scientific articles (Howell et al., 2019; Karl, 2019; Karl et al., 2013), and to enhance the understanding of relations within a study area (Margulies, Magliocca, Schmill, & Ellis, 2016). In more than half of the scientific articles that refer to locations, coordinates are used to determine the location (Karl et al., 2013; Shapiro & Báldi, 2012). However, geospatial information on scientific articles is not yet exploited in scholarly publishing platforms. Coordinates can be included in articles in different formats (Karl, 2019; Kmoch, Uuemaa, Klug, & Cameron, 2018) and therefore are prone to errors such as improper formatting, incompleteness, and ambiguity (Karl, 2019; Margulies et al., 2016), so that demand for standardization increases (Karl, 2019; Karl et al., 2013; Kmoch et al., 2018; Margulies et al., 2016; Young & Lutters, 2017). In this work, we report on a novel approach to integrate well-defined geospatial metadata in a scholar publishing platform so it can enhance discovery of scientific articles.

geoOJS offers a novel way for authors to provide spatial properties of research works when submitting an article to a journal based on the open source software Open Journal Systems (OJS, https://pkp.sfu.ca/ojs/). Previous work applied text recognition (Kmoch et al., 2018) or pattern matching algorithms (Karl, 2019) to derive geospatial metadata from papers, but such fully automated workflows are not without shortcomings. Instead, we decided to streamline the user interaction to create geospatial metadata so that user's understanding of the most suitable connections with location(s) or area(s) is unambiguously recorded. Authors can either search for a location and accept the suggested bounding box or manually create one or more suitable geometric shape(s) on a map. If authors enter geometries, a gazetteer is used to suggest a matching administrative unit's name to the author. This allows *geoOJS* to store geospatial data in two forms: as text, using the above administrative unit or standardised geographical norm data, and as geospatial coordinates in GeoJSON format. Thereby the coordinates are stored accurately, while at the same time a textual description is accessible and flexible for non-map-related usage. In addition to displaying geospatial information on maps, it is also added to the HTML source code of articles' landing pages in a semantically meaningful way, e.g., using Schema.org vocabulary. This facilitates indexing by search engines and can improve accessibility by supporting screen readers better than a regular map. To evaluate these goals, we implement a prototype for *geoOJS* and demonstrate the metadata input by the authors, the storage as precise coordinates and standardized texts, and the integration of location information in article views (see Figure 1).


Home / Archives / Vol. 1 No. 1 (1): firstIssue / Articles

Lorem ipsum

dolor sit amet

tom_o2r tom_o2r

Published
2020-08-31

Issue
Vol. 1 No. 1 (1): firstIssue

Section
Articles

Abstract

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Geospatial Metadata

Temporal Properties

Start: Sun Aug 09 2020 12:16:00 GMT+0200 (Mitteleuropäische Sommerzeit)
End: Sat Aug 15 2020 23:15:00 GMT+0200 (Mitteleuropäische Sommerzeit)

Spatial Properties

Coverage Information

Muenster

which is classified in the following hierarchical system of administrative units: Earth, Europe, Germany, North Rhine-Westphalia, Regierungsbezirk Münster, Kreisfreie Stadt Münster, Münster

Figure 1: Screenshot of *geoOJS*: geospatial properties in the OJS article view

Future work includes research into usability and usefulness of geospatial metadata for discovery of articles, a search engine across OJS instances, geospatial and temporal filters in article search, and validation of geospatial metadata as part of the review process. We plan to release *geoOJS* in the OJS plugin gallery so that the increasing number of independent and Open Access journals may benefit from it.

References

Howell, R. G., Petersen, S. L., Balzotti, C. S., Rogers, P. C., Jackson, M. W., & Hedrich, A. E. (2019). Using webgis to develop a spatial bibliography for organizing, mapping, and disseminating research information: a case study of quaking aspen. *Rangelands*, *41*(6), 244–247. doi: 10.1016/j.rala.2019.10.001

Karl, J. W. (2019). Mining location information from life-and earth-sciences studies to facilitate knowledge discovery. *Journal of Librarianship and Information Science*, *51*(4), 1007–1021. doi: 10.1177/0961000618759413

Karl, J. W., Herrick, J. E., Unnasch, R. S., Gillan, J. K., Ellis, E. C., Lutters, W. G., & Martin, L. J. (2013). Discovering ecologically relevant knowledge from published studies through geosemantic searching. *BioScience*, *63*(8), 674–682. doi: 10.1525/bio.2013.63.8.10

Kmoch, A., Uuemaa, E., Klug, H., & Cameron, S. G. (2018). Enhancing location-related hydrogeological knowledge. *ISPRS International Journal of Geo-Information*, *7*(4), 132. doi: 10.3390/ijgi7040132

Margulies, J. D., Magliocca, N. R., Schmill, M. D., & Ellis, E. C. (2016). Ambiguous geographies: connecting case study knowledge with global change science. *Annals of the American Association of Geographers*, *106*(3), 572–596. doi: 10.1080/24694452.2016.1142857

Shapiro, J. T., & Báldi, A. (2012). Lost locations and the (ir) repeatability of ecological studies. *Frontiers in Ecology and the Environment*, *10*(5), 235–236. doi: 10.1890/12.WB.015

Young, A. L., & Lutters, W. G. (2017). Infrastructuring for cross-disciplinary synthetic science: Meta-study research in land system science. *Computer Supported Cooperative Work (CSCW)*, *26*(1-2), 165–203. doi: 10.1007/s10606-017-9267-z

15 Guerrilla badges for geoscience research packages

Authors & contribution Daniel Nüst (30%), Lukas Lohoff, Lasse Einfeldt, Nimrod Gavish, Marlena Götza, Shahzeib Tariq Jaswal, Salman Khalid, Laura Meierkort, Matthias Mohr, Clara Rendel and Antonia van Eek

Venue *AGILE Conference 2019 Short Paper* 10.31223/osf.io/xtsqh

Date 04/2019

Guerrilla Badges for Reproducible Geospatial Data Science

Daniel Nüst, Lukas Lohoff, Lasse Einfeldt, Nimrod Gavish,
Marlena Götza, Shahzeib Tariq Jaswal, Salman Khalid, Laura Meierkort,
Matthias Mohr, Clara Rendel and Antonia van Eek

Institute for Geoinformatics (ifgi)
University of Münster
Münster, Germany
daniel.nuest@uni-muenster.de

Abstract

The building blocks of research are developing at an unprecedented pace. Data collection, analysis, interpretation, presentation, review, and publication take place completely on computers. The final product often is still a static document with only limited links to the underlying digital material, making transparency and reproducibility a challenge. In this work we apply the mechanism of badges to provide prominent connections to underlying analyses environments and important (meta-)data to readers of scholarly publications in geospatial data science. An API specification and implementation for a badge server provide extended and regular badges. The badges leverage recognition value for executability, licensing, spatial extent, and peer-review metadata – base information which either is or should be made available. We show a client-side integration method across many third-party platforms that allows evaluation of badges in realistic scenarios. The server and client enable an independent spreading of badges. The open source publication of all used software enables reproducibility and extensibility. The badges show potential to enhance interaction with scholarly works and should be evaluated with academic users in the future.

Keywords: badges, open science, data science, publication infrastructures, scholarly publication, research compendium

1 Introduction

The building blocks of research are constantly developing, though arguably at an unprecedented pace in the current age of digitisation. Data collection, analysis, interpretation, presentation, review, and publication take place completely on computers. However, the main outcome still is often a static document (e.g. an HTML or PDF file) resembling the traditional form of dissemination – the research paper. Thus Buckheit & Donoho (1995) postulated: *"An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship."* The typical research paper provides only limited links to the underlying building blocks of the actual scholarship, such as input datasets, software/hardware environment, workflow code, or output data. Therefore reproducibility and reusability, both cornerstones of science, have been identified as important challenges in geospatial data science (Nüst et al., 2018; Konkol et al., 2018). Efforts to improve the publication of and access to data and software, e.g. establishing citation principles (Wilkinson et al., 2016; Katz & Chue Hong, 2018), exist and they work (Piwowar & Vision, 2013). Practicing Open Science (Ibanez, 2014) and the advantages of openness, transparency and reproducibility, e.g. efficiency and collaboration effects, are clear (cf. Markowetz, 2015). Research compendia, a term first used by Gentleman & Temple Lang (2007) and since then taken up and extended[1], are but one concept to package all buildings blocks of a piece of research. Nevertheless these practices are not common yet.

1 See https://research-compendium.science for a full list of recommendations and guidelines.

In this work we use the concept of badges to expose, not only advertise, the building blocks of scholarship. Badges are an established artefact in the software development community to visually highlight important pieces of information, exploiting a high recognition value. A user can quickly grasp the current version of a piece of software of interest, whether its test suite completes successfully or fails, or whether a tool is available in an established public repository for easy installation. Gaining these badges, and keeping them "green" in the case of tests, works as a motivator for developers. In science, the core medium to disseminate work between users, i.e. scientists, is the research paper. Badges for relevant building blocks behind research papers could benefit both users of this medium. Readers could quickly assess multiple or single publication items. Authors are encouraged to share more complete information (cf. Grahe, 2013) at the prospect of a larger readership and reuse. Relevance is specific to each reader, e.g. a paper may use data from the same area of interest or may contain transferable methods. For reproducible geospatial data science, we see the following questions as crucial for readers to decide if a work is interesting for them, e.g. in a literature study, and badges could help to answer them: *Is all code, data, and documentation openly available? Is a software environment documented so the results can be reproduced? What is the area of interest or data location?*

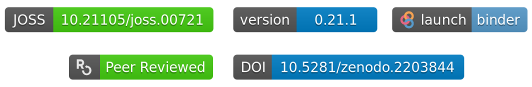
In the remainder of this work we first give a detailed background on badges in science. Then we present and discuss the first prototype of an API, server, and client implementation for creating and spreading badges on scholarly communication platforms.

2 Related Work

Digital badges to show a specific accomplishment are popular accessories to “earn” in Free and Open Source Software (FOSS) development. They are awarded by platforms providing a service, or by third-party websites based on metadata available via APIs of said platforms. Developers must only follow common practices to provide the required information, e.g. structured metadata in a project description. They then include the badges in their documentation to show relevant bits of information to their users. Most badges are generated with current data each time they are loaded. They show a tuple of property name and value, and may use colour to visually distinguish property values. Badges can include icons, e.g. a logo, and are provided in different formats, e.g. vector (as SVG – Scalable Vector Graphics) or raster (PNG) graphics. *Shields.io* (https://shields.io/) is a popular badge service. It provides badges for example for software repositories (e.g. software version, number of downloads), license, popularity (download count, ratings), or build systems (e.g. status of automated tests). Shields.io also renders own information by providing text and styling information within a URL. Other badge services cover specific use cases, e.g. *MicroBadger* (http://microbadger.com/) provides images for container images published on Docker Hub.

Due to the high recognition value, badges have been picked up by platforms and groups in a scientific context, including several journals. These mostly show static content. The data repository *Zenodo* (http://zenodo.org/) and the journal *JOSS* (http://joss.theoj.org/) provide badges with the Digital Object Identifiers (DOIs) of records. The *ROpenSci* initiative uses them for different stages of its software review process (https://badges.ropensci.org/). The *Binder* project uses badges to advertise the availability of an interactive notebook for a project repository. Examples of these badges are shown in Figure 1.

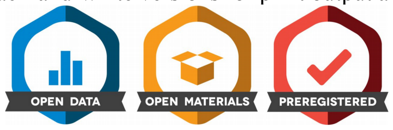
Figure 1: Badges from (clockwise beginning at top left) JOSS, MicroBadger, Binder, Zenodo, and ROpenSci Review.



The Center for Open Science (COS, https://cos.io/) designed badges for acknowledging open practices in scientific articles (see https://osf.io/tvyxz). COS offers guidelines for incorporation into peer reviews and adding badges to documents. The badges are *Open Data*, *Open Materials*, and *Preregistration* of studies (see Figure 2) and are adopted by over a dozen of journals to date[2]. A study by Kidwell et al. (2016) reports a positive effect from the introduction of open data badges in the journal *Psychological Science*: After the journal started awarding badges for open data, more articles stating open data availability actually published data (cf. Baker, 2016). The COS badges are effective in promoting data publishing and show availability and transparency, but not geospatial aspects or reproducibility.

2 https://osf.io/tvyxz/wiki/5.%20Adoptions%20and%20Endorsements/

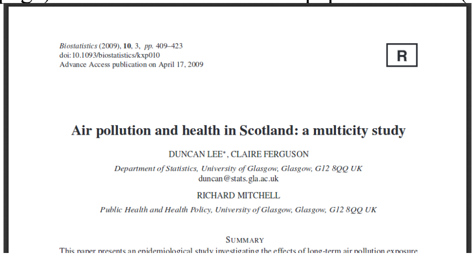
Figure 2: Full COS badges (from left to right: open data – blue, open materials – yellow, preregistered – red) in colour, reduced black-and-white versions for print output also exist.



Source: https://osf.io/tvyxz/wiki/home/.

Peng (2009, 2011) and Rowhani-Farid & Barnett (2018) report on the usage of badges in the journal *Biostatistics*: a set of “kite marks” led to a moderate increase in data sharing. Marks *D* and *C* are awarded if data respectively code is provided, and *R* if results were successfully reproduced during the review process (implying D and C). Figure 3 shows the usage of *R* on an article’s title page.

Figure 3: *Biostatistics* kite mark *R* rendering (top right part of the page) in the PDF version of the paper Lee et al. (2009).



Biostatistics (2009), **10**, 3, *pp.* 409–423
doi:10.1093/biostatistics/kxp010
Advance Access publication on April 17, 2009

R

Air pollution and health in Scotland: a multicity study

DUNCAN LEE*, CLAIRE FERGUSON
Department of Statistics, University of Glasgow, Glasgow, G12 8QQ UK
duncan@stats.gla.ac.uk

RICHARD MITCHELL
Public Health and Health Policy, University of Glasgow, Glasgow, G12 8QQ UK

SUMMARY

This paper presents an epidemiological study investigating the effects of long-term air pollution exposure

Source: https://academic.oup.com/biostatistics/article-pdf/10/3/409/17736633/kxp010.pdf.

The Association for Computing Machinery (ACM, https://www.acm.org/) provides a common terminology and standards for artefact review processes for its conferences and journals[3]. Three badges with several levels (see Figure 4) are awarded using specific criteria, e.g. the *Evaluated* badge means artefacts were made available to reviewers and it has levels *Functional* or *Reusable*. The ACM badges provide excellent information on reproducibility for human readers, but not on geospatial information and not across platforms.

The Graphics Replicability Stamp Initiative[4] (GRSI) organises a community-driven additional evaluation process for computer graphics research. Its results are the basis for different badges for a number of journals and conferences, e.g. ACM’s badges for *ACM Transaction on Graphics*.

Figure 4: ACM badges, from left to right: *Artifacts Evaluated – Functional/Reusable* (pink/red), *Artifacts Available* (green, no evaluation), *Results Replicated* (light blue, artefacts provided by author), and *Results Reproduced* (dark blue, without author-supplied artefacts).

acm
acm
acm
acm
acm

Source: https://www.acm.org/publications/policies/artifact-review-badging.

3 ACM policies: Artifact Review Badging, see https://www.acm.org/publications/policies/artifact-review-badging; Example article (badges rendered on landing page and PDF): https://doi.org/10.1145/3197517.3201397
4 http://www.replicabilitystamp.org/

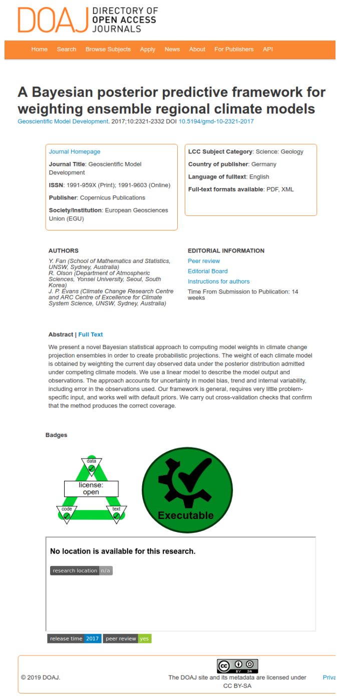
These examples show the potential, diversity, and challenges in describing and awarding badges. This work explores novel badge types on an article's reproducibility and spatial area of interest, and an independent distribution mechanism, to contribute to the landscape of badges in scholarly publishing.

3 Geospatial Data Science Badges

3.1 Sources for Badge Information

The basis for useful badges are reliable data sources. For publications in geospatial data science, there are no established platforms or metadata protocols beyond regular article metadata, i.e. for code or data licenses or spatio-temporal extents of used datasets. Publication date, peer-review type (e.g. blind or double-blind), and license (i.e. if article is Open Access) are provided via online library reference APIs, namely Crossref (http://crossref.org/) or DOAJ (https://doaj.org/). Search terms are article title or DOI. The other properties are accessible via the prototypical o2r reproducibility service (Nüst, 2018). It provides access to the metadata of Executable Research Compendia (ERC) via the DOI of the related article. ERC contain all data and code used in a particular workflow and their creation process includes automatic extraction and user validation of metadata, including a spatio-temporal bounding box, for increased transparency and reproducibility. Both sources are used for geospatial research badges, but due to the prototypical state of the o2r API following examples rely on mock-up data.

Figure 5: Extended badges on an article page on DOAJ.org, integrated below the Abstract section.



DOAJ DIRECTORY OF OPEN ACCESS JOURNALS

Home Search Browse Subjects Apply News About For Publishers API

A Bayesian posterior predictive framework for weighting ensemble regional climate models
Geoscientific Model Development. 2017;10:2321-2332 DOI 10.5194/gmd-10-2321-2017

Journal Homepage
Journal Title: Geoscientific Model Development
ISSN: 1991-959X (Print); 1991-9603 (Online)
Publisher: Copernicus Publications
Society/Institution: European Geosciences Union (EGU)

LCC Subject Category: Science: Geology
Country of publisher: Germany
Language of fulltext: English
Full-text formats available: PDF, XML

AUTHORS
Y. Fan (School of Mathematics and Statistics, UNSW, Sydney, Australia)
R. Olson (Department of Atmospheric Sciences, Yonsei University, Seoul, South Korea)
J. P. Evans (Climate Change Research Centre and ARC Centre of Excellence for Climate System Science, UNSW, Sydney, Australia)

EDITORIAL INFORMATION
Peer review
Editorial Board
Instructions for authors
Time From Submission to Publication: 14 weeks

Abstract | **Full Text**
We present a novel Bayesian statistical approach to computing model weights in climate change projection ensembles in order to create probabilistic projections. The weight of each climate model is obtained by weighting the current day observed data under the posterior distribution admitted under competing climate models. We use a linear model to describe the model output and observations. The approach accounts for uncertainty in model bias, trend and internal variability, including error in the observations used. Our framework is general, requires very little problem-specific input, and works well with default priors. We carry out cross-validation checks that confirm that the method produces the correct coverage.

Badges

No location is available for this research.
research location n/a

release time 2017 peer review yes

© 2019 DOAJ. The DOAJ site and its metadata are licensed under CC BY-SA Priva

3.2 Badge types, badge design, and an API

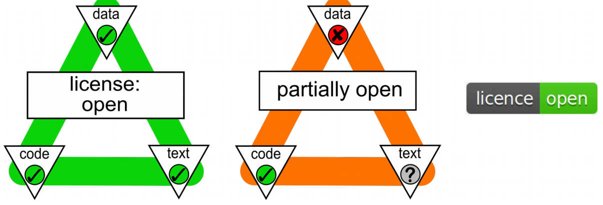
A RESTful[5] Application Programmer Interface (API) defines routes (i.e. URLs) to access badges of four types to answer the discovery questions in geospatial data science (see above):

- **executable**: code and runtime reproducibility
- **licence**: licensing information or all building blocks
- **spatial**: publication's geospatial area of interest
- **peerreview**: type of peer review

For these types we designed badges at two levels of detail, regular and extended. The extended badges contain a higher level of detail, while regular badges aggregate information to be suitable for search result listings, where they allow a visual comparison of hits, see Figure 5. Extended badges are more open in their design, while regular badges follow common badge styling.

The extended license badge has three categories (code, data and text), which are aggregated to simpler text ("open", "partially open", "mostly open", "closed") in the regular badge (see Figure 6).

Figure 6: Extended badges reporting different values for data, text and code licenses (left, middle) and regular badge (right, not to scale, based on Shields.io).



The extended spatial badge shows the bounding box as an interactive map, whereas the regular spatial badge only shows a suitable area name.

Badges for reproducibility, peer review status and license are colour coded to provide visual aids. They indicate for example (un)successful reproduction, a single-blind peer review process, or different levels of open licenses.

The API provides regular badges with HTTP GET requests, i.e. URLs following the pattern `/api/1.0/badge/:type/:doi`, where `1.0` is the API version, `:type` is the badge type, and `:doi` is the publication's DOI. Extended badges are returned when `/extended` is appended to the URL.

3.3 Implementation

Badger implements the **server-side API**. It queries public APIs to elicit metadata and provide the aforementioned badges types. It uses two badge generation methods: (a) internally created SVG-based badges, and (b) redirects to Shields.io, where the information is encoded in the Shields.io-URL, which is generated on the fly. All badges can be requested at specific size and pre-rendered as a PNG image for compatibility. The process for generating the executability badge for a paper "Global Air Quality and Pollution" from *Science* identified by the DOI 10.1126/science.1092666 is as follows.

5 https://en.wikipedia.org/wiki/Representational_state_transfer

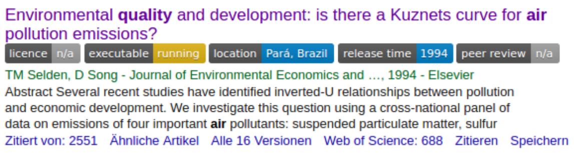
1. A client calls the URL `https://badger/api/1.0/badge/executable/10.1126%2Fscience.1092666`.
2. *Badger* queries the o2r API for ERC connected to the given DOI. If it exists, it queries for the latest reproduction result status (a "job" in the API).
3. Depending on the result (success, running, or failure) specified in the job metadata, *Badger* generates a Shields.io-URL with green, yellow or red colour and matching property value.
4. The client displays a badge: executable yes.

If an extended badge is requested, *Badger* generates an SVG graphic or an embeddable HTML snippet. For the spatial badge it converts coordinates into textual information, i.e. country and if available district or place name, using the Geonames API[6], see Figure 7. When *Badger* does not find data for a certain DOI, it returns a grey "not available" - badge, see outermost badges "license" and "peer review" in Figure 7. Such a null result, e.g. "no spatial data included", can be equally helpful during discovery.

Badges are most successful when they are widely used and consequently quickly recognised by users. Though a desirable and more sustainable approach, it is unrealistic that (competing) publishers agree on a common badge system and design. Therefore we took an unusual approach to augment existing platforms for discovery of papers using a Chrome browser extension[7], similar to Unpaywall browser extension[8]. The *Extender* implements **client-side badge integration**. It inserts badges into search results or article pages using client-side browser scripting, also known as userscripts, on several websites including DOAJ (https://doaj.org/), Google Scholar (https://scholar.google.de/), PLOS (https://www.plos.org/), Microsoft Academic (https://academic.microsoft.com/), and ScienceDirect (http://www.sciencedirect.com/),

For each article displayed on these websites, either in a search result listing or dedicated article pages, *Extender* parses the DOI from the page's HTML code, requests badges from *Badger*, and inserts them into the page. The parsing and insertion is tailored to each supported website. Figure 7 shows an exemplary result. When the DOI is not directly provided, *Extender* queries the Crossref API with the paper title and uses the returned DOI if the result is unambiguous. *Extender* also inserts controls for filtering search results using badge values and for selecting displayed badges as shown in Figure 8.

Figure 7: Regular badges integrated into Google Scholar search result listing between title and authors (partial screenshot).


Environmental **quality** and development: is there a Kuznets curve for **air** pollution emissions?
licence n/a | executable running | location Pará, Brazil | release time 1994 | peer review n/a
TM Selden, D Song - Journal of Environmental Economics and ..., 1994 - Elsevier
Abstract Several recent studies have identified inverted-U relationships between pollution and economic development. We investigate this question using a cross-national panel of data on emissions of four important **air** pollutants: suspended particulate matter, sulfur
Zitiert von: 2551 Ähnliche Artikel Alle 16 Versionen Web of Science: 688 Zitieren Speichern

6 http://www.geonames.org/
7 https://en.wikipedia.org/wiki/Google_Chrome#Extensions
8 https://unpaywall.org/products/extension

4 Discussion

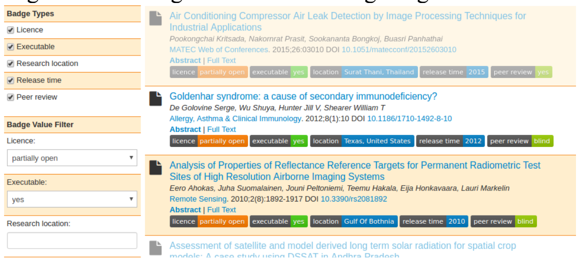
A badge server for scholarly publications has the potential to improve discovery workflows for scientists by aggregating information, including underlying spatial data, with a high recognition value. It can enable identification of related work and reusability – an important aim of reproducibility. It also demonstrates badges as a mean to communicate more complex information compared to existing approaches.

An independently operated *Badger* and client-side integration with *Extender* may be favourable to a complex process of establishing a single set of badges across all involved stakeholders. This "guerrilla" approach allows to bring a new concept onto researcher's computers beyond a specific research project's own software or websites in a secure and reliable manner. It can also facilitate long-term studies, because users are exposed during their regular work and not only in a lab setting. However, the realised userscript integration into websites is less stable than an actual integration in platform APIs would be, because any UI change or code change may break the userscript. The Open Source nature of *Badger* and *Extender* allows research domains to adopt criteria to their needs. As a further effect, they may foster improved research practices regarding publication of data and code, and reproducibility.

The current API design lacks a transparent process (akin to ACM or COS badges though theirs are manual) to award the reproducibility and geospatial badges. The provenance of badges (i.e. who awarded it, to what, using which criteria) would be crucial in a scholarly setting to establish trust. It could be made accessible with interactive badges, e.g. clicking on a badge opens a pop-up with background information, but also for other services if the information behind the badges is exposed in a structured form via the API, supplementary to the mere images. The current approach could be extended in these directions leveraging SVG's features for interaction, and content-type negotiation for alternative representations. The novelty of ERC and the o2r reproducibility service is an issue, because three badge types rely solely on their existence. Only a wider uptake of ERCs or ERC-like metadata in other platforms, e.g. geospatial properties in publication metadata, can mitigate this.

We see automatically generated and independently spread badges as a promising supplement to the inspection-based badges by COS or ACM and as a way to expose still underused properties of publications' geospatiality and reproducibility. The biggest risks are fragmentation and establishing the trustworthiness of sources for badge information, both due to the distributed approach for defining, creating, and inserting badges.

Figure 8: Filtering search results using badge values.


Badge Types
☑ Licence
☑ Executable
☑ Research location
☑ Release time
☑ Peer review

Badge Value Filter
Licence: partially open
Executable: yes
Research location:

Air Conditioning Compressor Air Leak Detection by Image Processing Techniques for Industrial Applications
Pookongchai Kritsada, Nakornrat Prasit, Sookananta Bongkoj, Buasri Panhathai
MATEC Web of Conferences. 2015;26:03010 DOI 10.1051/matecconf/20152603010
Abstract | Full Text
licence partially open | executable yes | location Surat Thani, Thailand | release time 2015 | peer review yes

Goldenhar syndrome: a cause of secondary immunodeficiency?
De Golovine Serge, Wu Shuya, Hunter Jill V, Shearer William T
Allergy, Asthma & Clinical Immunology. 2012;8(1):10 DOI 10.1186/1710-1492-8-10
Abstract | Full Text
licence partially open | executable yes | location Texas, United States | release time 2012 | peer review blind

Analysis of Properties of Reflectance Reference Targets for Permanent Radiometric Test Sites of High Resolution Airborne Imaging Systems
Eero Ahokas, Juha Suomalainen, Jouni Peltoniemi, Teemu Hakala, Eija Honkavaara, Lauri Markelin
Remote Sensing. 2010;2(8):1892-1917 DOI 10.3390/rs2081892
Abstract | Full Text
licence partially open | executable yes | location Gulf Of Bothnia | release time 2010 | peer review blind

Assessment of satellite and model derived long term solar radiation for spatial crop models: A case study using DSSAT in Andhra Pradesh

5 Future Work

While the prototypes show the technical feasibility, the most important next step is a user study to evaluate the design and content of the novel badges concerning the goal of improving user experience during discovery, and to learn more about the motivation, requirements, and preferences of involved user groups. The study should investigate potential effects on willingness to publish research compendia and elaborate on trust. It could potentially draw parallels to mechanisms behind other badges, e.g. organic food labels. Such a study can inform the further development of badges, e.g. interactive features, visual design, and regarding transparency (see Discussion). Technical measures can be taken to improve the experience, such as client- or server-side caching, and *Extender* can be reimplemented as a WebExtension[9] to make it available for other browsers such as Firefox. After solving technical and usability-related challenges, a real adoption by the scientific community requires an involvement of more stakeholders and individual early adopters, e.g. funding agencies and a leading journal or conference. Together these institutions can initiate a lighthouse project and a public discourse about the content and scope of badges, so that iterative improvements can make the badges more useful, even for larger user groups beyond the geospatial community.

Software and Data Availability

The implementations of *Badger* and *Extender* including Docker image tarballs, a docker-compose configuration, test data, and instructions for local evaluation (see file README.md) are published on Zenodo (Lohoff & Nüst, 2018). The source code projects are on GitHub at https://github.com/o2r-project/o2r-badger respectively https://github.com/o2r-project/o2r-extender.

Author Contributions

D. N. conceived the idea, supervised the development, and wrote the paper. L. L. developed software and drafted the paper. L. E., N. G., M. G., S. T. J. S. K., L. M., M. M., C. R. and A. v. E. designed badges and developed software. All authors approved the final version of the paper.

Acknowledgements

D. N. and L. L. were supported by the project Opening Reproducible Research (https://o2r.info) funded by the German Research Foundation (DFG) under project number PE 1632/10-1. We thank the members of "Scientists for Reproducible Research" for the input in the forum thread "Any journals using badges"[10], and the reviewers for their time and constructive comments.

9 https://wiki.mozilla.org/Add-ons/Terminology
10 https://groups.google.com/forum/#!topic/reproducible-research/AP0k_xi69AA/discussion

References

Baker, M. (2016) Digital badges motivate scientists to share data. *Nature News*. doi:10.1038/nature.2016.19907.

Buckheit, J. B. and Donoho, D. L. (1995) WaveLab and Reproducible Research. In: Antoniadis, A., Oppenheim, G. (eds.) *Wavelets and Statistics*, 55–81. Lecture Notes in Statistics 103. Springer New York. doi:10.1007/978-1-4612-2544-7_5.

Gentleman, R. and Temple Lang, D. (2007) Statistical Analyses and Reproducible Research. *Journal of Computational and Graphical Statistics,* 16, no. 1, 1–23. doi:10.1198/106186007X178663.

Grahe, J. E. (2014) Announcing Open Science Badges and Reaching for the Sky. *The Journal of Social Psychology*, 154, no. 1, 1–3. doi:10.1080/00224545.2014.853582.

Ibanez, L., Schroeder, W. J., and Hanwell, M. D. (2014) Practicing Open Science. In: Stodden, V., Leisch, F., Peng, R. D. (eds.) *Implementing Reproducible Research*, 241–280. Chapman & Hall/CRC The R Series. CRC Press, 2014. https://osf.io/emvbz/.

Katz, D. S., and Chue Hong, N. P. (2018) Software Citation in Theory and Practice. In: Davenport, J. H., Kauers, M., Labahn, G., Urban, J. (eds.) *Mathematical Software – ICMS 2018*, 289–96. Lecture Notes in Computer Science. Springer International Publishing. doi:10.1007/978-3-319-96418-8_34.

Kidwell, M. C., et al. (2016) Badges to Acknowledge Open Practices: A Simple, Low-Cost, Effective Method for Increasing Transparency. *PLOS Biology* 14(5):e1002456. doi:10.1371/journal.pbio.1002456.

Lee, D., Ferguson, C and Mitchell, R. (2009) Air pollution and health in Scotland: a multicity study. *Biostatistics*, Volume 10, Issue 3, Pages 409–423, doi:10.1093/biostatistics/kxp010.
Markowetz, F. (2015) Five Selfish Reasons to Work Reproducibly. *Genome Biology* 16:274. doi:10.1186/s13059-015-0850-7.

Lohoff, L., and Nüst, D. (2018) Reproducibility package for: Badges for Geoscience Containers. *Zenodo*. doi:https://doi.org/10.5281/zenodo.1199271.

Nüst, D. (2018) Reproducibility Service for Executable Research Compendia: Technical Specifications and Reference Implementation. *Zenodo*. doi:10.5281/zenodo.2203843.

Konkol, M., Kray, C. and Pfeiffer, M (2018) Computational Reproducibility in Geoscientific Papers: Insights from a Series of Studies with Geoscientists and a Reproduction Study. *International Journal of Geographical Information Science*, 1–22. doi:10.1080/13658816.2018.1508687.

Nüst, D., Konkol, M., Pebesma, E., Kray, C., Schutzeichel, M., Przibytzin, H. and Lorenz, J. (2017) Opening the Publication Process with Executable Research Compendia. *D-Lib Magazine*. doi:10.1045/january2017-nuest.

Nüst, D., Granell, C., Hofer, B., Konkol, M., Ostermann, F. O., Sileryte, R. and Cerutti, V. (2018) Reproducible Research and GIScience: An Evaluation Using AGILE Conference Papers. *PeerJ,* 6, e5072. doi:10.7717/peerj.5072.

Peng, R. D., 2009. Reproducible research and Biostatistics. *Biostatistics*, Volume 10, Issue 3, Pages 405–408. doi:10.1093/biostatistics/kxp014.

Peng, R. D. (2011) Reproducible Research in Computational Science. *Science*. 334 (6060): 1226–27. doi:10.1126/science.1213847.

Piwowar, H. A., and Vision, T. J. (2013) Data Reuse and the Open Data Citation Advantage. *PeerJ*, 1:e175. doi:10.7717/peerj.175.

Wilkinson, M. D., et al. (2016) The FAIR Guiding Principles for Scientific Data Management and Stewardship. *Scientific Data*, 3:160018. doi:10.1038/sdata.2016.18.

16 How to read a research compendium

Authors & contribution Daniel Nüst (80%), Carl Boettiger, Ben Marwick

Venue *arXiv.org* arXiv:1806.09525 [cs.GL]

Date 06/2018

Repository https://github.com/research-compendium/how-to-read-a-research-compendium

How to Read a Research Compendium

Daniel Nüst, Institute for Geoinformatics, University of Münster, Münster, Germany (daniel.nuest@uni-muenster.de)
Carl Boettiger, Department of Environmental Science, Policy and Management, University of California, Berkeley, Berkeley, California, United States (cboettig@gmail.com)
Ben Marwick, Department of Anthropology, University of Washington, Seattle, Washington, United States (bmarwick@uw.edu)

Abstract

Researchers spend a great deal of time reading research papers. Keshav (2012) provides a three-pass method to researchers to improve their reading skills. This article extends Keshav's method for reading a research compendium. Research compendia are an increasingly used form of publication, which packages not only the research paper's text and figures, but also all data and software for better reproducibility. We introduce the existing conventions for research compendia and suggest how to utilise their shared properties in a structured reading process. Unlike the original, this article is not build upon a long history but intends to provide guidance at the outset of an emerging practice.

1. Introduction

1.1 Motivation

Research compendia are an increasingly used form of publication and scholarly communication. They comprise not only the research paper's text and figures, but also all data and software used to conduct the computational workflow and create all outputs. They provide a lot of added value by revealing more of the research process to readers, but, if not done well, they can increase the difficulty of understanding the research. To help readers better understand how to read a research compendium, we extends Keshav's three-pass method targeted at improving skills for reading a research paper (Keshav 2007) with additional steps relevant to a research compendium's content.

Unlike the first version of the original (Keshav 2007), we cannot draw from a long history of experience, because until recently research compendia have been relatively rare. Our intention here is to provide guidance at the outset of an emerging practice to both readers and authors of research compendia to help them understand each others' perspectives and needs and improve their communication. Authors can use this guide to improve their research compendium's structure and content by better anticipating their readers' needs. They should not be held back by unwarranted concerns, like providing support (Barnes 2010). Readers can avoid the trap of falling too deep into technological challenges by an iterative approach to reading and using that gives attention to the scientific issues. Ultimately research compendia can enhance and deepen the reading experience, if done right. Keshav's following introduction applies directly to research compendia:

> *Researchers must read papers for several reasons: to review them for a conference or a class, to keep current in their field, or for a literature survey of a new field. A typical researcher will likely spend hundreds of hours every year reading papers.*
>
> *Learning to efficiently read a paper is a critical but rarely taught skill. Beginning graduate students, therefore, must learn on their own using trial and error. Students waste much effort in the process and are frequently driven to frustration.*
>
> *For many years I have used a simple 'three-pass' approach to prevent me from drowning in the details of a paper before getting a bird's-eye-view. It allows me to estimate the amount of time required to review a set of papers. Moreover, I can adjust the depth of paper evaluation depending on my needs and how much time I have. This paper describes the approach and its use in doing a literature survey.* (Keshav 2016)

The additions made in this work to accommodate for the content in a research compendium are quite extensive. This stems from the complexity that an interactive compendium has compared to a classic static "paper", because a research compendium goes well beyond the "mere advertising of the scholarship" (Claerbout 1994). We see the breadth of additions as a sign of potential, namely for unprecedented transparency, openness, and collaboration.

1.2 Structure

In the remainder of this paper, the excellent original work is taken over completely. It is set in *italic font* based on the most recent online version: Keshav (2016). The term "paper" was not replaced with "research compendium" for better readability.

First we briefly introduce research compendia and existing conventions. We further list relevant resources for authors related to research compendia. Then, matching the original paper's section numbering, Sections 2 extends the "Three-pass Approach" to include research compendium features in the reading process. Section 3 extends "Doing a Literature Survey" with aspects relevant reviewing many research compendia.

1.3 Research compendia

The term *research compendium* was coined by Gentleman and Lang (2007) who "introduce[d] the concept of a compendium as both a container for the different elements that make up the document and its computations (i.e. text, code, data,. . .), and as a means for distributing, managing and updating the collection." According to Marwick, Boettiger, and Mullen (2018) it provides "a standard and easily recognisable way for organising the digital materials of a research project to enable other researchers to inspect, reproduce, and extend the research". This standard may differ between scientific domains, yet the intentions and benefits are the same. Research compendia are practised Open Science culture and as such improve transparency (Nosek et al. 2015), "make more published research true" (Ioannidis 2014), and enable enhanced review and publication workflows (Nüst et al. 2017). They answer readers' needs to understand complex analyses through inspection and manipulation (Konkol and Kray 2018) and enable other researchers to reproduce and extend the research (Marwick, Boettiger, and Mullen 2018). Research compendia improve citations since code and data are openly available (Vandevalle 2012). Ultimately, their goal is to improve reproducibility (see Barba (2018) for definitions of terms) in the light of claims of a "reproducibility crisis" in several fields. Infrastructures to support the creation, scientific publication, inspection, and collaboration based on research compendia are an active field of research, but none of which have been widely deployed yet (Nüst et al. (2017); Brinckman et al. (2018); Stodden, Miguez, and Seiler (2015); Kluyver et al. (2016); Green and Clyburne-Sherin (2018)).

As this article is focused on providing hands-on guidance on using, and to some extend also creating, research compendia, we refer the reader to the references for more specific details. For the remainder of this work, we assume a minimal view of a research compendium suitable for *readers* who examine a research compendium directly. A research compendium has three integral parts: text, code, and data. Text can be instructions, software documentation, or a full manuscript with figures. Code can be scripts, software packages, specifications of dependencies and computational environments, or even virtual machines. Data can be just about anything, but probably comprises plain text or binary files that are used as input to the workflow, and produced as output from executing the workflow.

For *authors*, there is a wealth of generic recommendations guiding researchers in creating open research (software), for example Sandve et al. (2013), Taschuk and Wilson (2017), Prlić and Procter (2012), Stodden and Miguez (2014), and Wilson et al. (2017). When a research compendium is published, one can assume the authors have the intention to help the reader understanding the work and accepts there are "no excuses" to not publishing your code (Barnes 2010). Authors may attempt to reach the ideals of having one "main" file that can be executed with "one-click" (Pebesma 2013), of enabling re-use with proper licensing (Stodden 2009), and of interweaving code and text following the literate programming paradigm (Knuth 1984).

The following conventions are specifically for research compendia:

- Marwick, Boettiger, and Mullen (2018) and ROpenSci community's `rrrpkg` (https://github.com/ropensci/rrrpkg) discuss the standards and tooling of the R programming language and software engineering tools for a variety of disciplines with real-world examples, including several templates
- Jimenez et al. (2017) apply software engineering best pratices from the Open Source software domain to research (see also http://falsifiable.us/).
- Konkol, Kray, and Pfeiffer (2018) derive recommendations for authors from issues encountered reproducing research compendia in geosciences
- Gentleman and Lang (2007) recommend using programming languages' packaging mechanisms for research compendia, more specifically R and

Python packages

- Chirigati et al. (2016) describe the tool `ReproZip` (https://reprozip.org) to support capture and reproduction of a research compendium

2. The three-pass approach

> *The key idea is that you should read the paper in up to three passes, instead of starting at the beginning and plowing your way to the end. Each pass accomplishes specific goals and builds upon the previous pass: The first pass gives you a general idea about the paper. The second pass lets you grasp the paper's content, but not its details. The third pass helps you understand the paper in depth.* (Keshav 2016)

2.1 The first pass

> *The first pass is a quick scan to get a bird's-eye view of the paper. You can also decide whether you need to do any more passes. This pass should take about five to ten minutes and consists of the following steps:* (Keshav 2016)

1. *Carefully read the title, abstract, and introduction*
2. *Read the section and sub-section headings, but ignore everything else*
3. *Glance at the mathematical content (if any) to determine the underlying theoretical foundations*
4. *Read the conclusions*
5. *Glance over the references, mentally ticking off the ones you've already read*
6. Glance over the text looking for (a) URLs and `formatted` **names** referencing software and data products or repositories not yet mentioned in the sections read so far, mentally ticking off the ones you've heard about or used, and (b) tables or figures describing computational environments, deployments, or execution statistics

At the end of the first pass, you should be able to answer the seven *Cs:*

1. *Category: What type of paper is this? A measurement paper? An analysis of an existing system? A description of a research prototype?*
2. *Context: Which other papers is it related to? Which theoretical bases were used to analyze the problem?*
3. *Correctness: Do the assumptions appear to be valid?*
4. *Contributions: What are the paper's main contributions?*
5. *Clarity: Is the paper well written?*
6. Construction: What are the building blocks of the analysis workflow and how accessible are they (data set(s), programming language(s), tools, algorithms, scripts)? Under what licenses are code and data published?
7. Complexity: What is the scale of the analysis (e.g. HPC, required OS/cores/memory, typical execution time, data size) and the software (number of dependencies and is installation possible with dependency management tools)?

> *Using this information, you may choose not to read further (and not print it out, thus saving trees). This could be because the paper doesn't interest you, or you don't know enough about the area to understand the paper, or that the authors make invalid assumptions.* (Keshav 2016)

You may also choose not to pursue the parts of the research compendium further, i.e. not running the workflow or looking at data or code, thus saving resources. Reasons to not read further that relate specifically to code and data may be that you don't have the expertise or access to resources to re-use the data and code.

> *The first pass is adequate for papers that aren't in your research area, but may someday prove relevant.* (Keshav 2016)

This first pass suits research compendia comprising potentially re-usable components, like workflows or algorithms using data sets or generic software that are directly transferable to your field of research. After the first pass, you should be able to judge if the software is useful, if it works.

> *Incidentally, when you write a paper, you can expect most reviewers (and readers) to make only one pass over it. Take care to choose coherent section and sub-section titles and to write concise and comprehensive abstracts. If a reviewer cannot understand the gist after one pass, the paper will likely be rejected; if a reader cannot understand the highlights of the paper after five minutes, the paper will likely never be*

> *read. For these reasons, a 'graphical abstract' that summarizes a paper with a single well-chosen figure is an excellent idea and can be increasingly found in scientific journals.* (Keshav 2016)

When you write a paper, take care to add instructions on how a reader can reproduce your work and provide all required parts, i.e. publish a research compendium. The instructions should start with a "blank" system and be specific, i.e. ready for copy & paste, including expected or experienced execution times and resources. Such instructions give readers a good idea about what is needed to recreate your environment and execute the analysis If your work requires specialised or bespoke hardware (HPC, specific GPUs), consider creating an exemplary, reduced analysis that runs in regular environments.

Also ensure your code and data are properly deposited, citable and licensed. If you don't do this, these core parts of your work will likely never be properly evaluated or re-used. See the section "Research Compendia", above, for recommendations and further reading on how to make your reviewers' and readers' lives easier.

2.2 The second pass

> *In the second pass, read the paper with greater care, but ignore details such as proofs. It helps to jot down the key points, or to make comments in the margins, as you read. Dominik Grusemann from Uni Augsburg suggests that you "note down terms you didn't understand, or questions you may want to ask the author." If you are acting as a paper referee, these comments will help you when you are writing your review, and to back up your review during the program committee meeting.* (Keshav 2016)

1. *Look carefully at the figures, diagrams and other illustrations in the paper. Pay special attention to graphs. Are the axes properly labelled? Are results shown with error bars, so that conclusions are statistically significant? Common mistakes like these will separate rushed, shoddy work from the truly excellent.* (Keshav 2016)
2. *Remember to mark relevant unread references for further reading (this is a good way to learn more about the background of the paper).* (Keshav 2016)
3. Skim over data and source code files without opening them. Are they reasonably named (Bryan 2015)? Do they follow a well-defined structure (e.g. a Python package or a research compendium convention)? Is there a README file and/or structured documentation for functionalities?
4. Visit the online source code repository, if available. Is it established and well maintained, or orphaned? Is there only one author or are there contributors? How responsive are they to issues? Does the repository have signs of public recognition (i.e. GitHub "stars" and "forks")? Are there regular releases, using semantic versioning?
5. Follow the instructions to install the required software and execute the research compendium's workflow with the provided parameters and input or sample data. Note down errors or warnings but do not try to fix any but trivial or known problems (e.g. fixing a path or installing an undocumented dependency).
6. Compare the outputs with the expected ones reported in the paper. Also check for differences in output figures: Do labels, legends etc. match those in the paper?

Points 3 and 4 above hint at how to estimate the quality of a software, but we recommend to be realistic as to what to expect and be careful not to judge too fast. The software project you evaluate might be done by a single researcher who is not a professional programmer, working under a lot of pressure to write code for a single use case. In these situations one might find low levels of code documentation, but further documentation might be quickly provided by the authors once you as an external user show interest. Also, no recent changes or releases at a source code repository can also mean the software is stable and simply works with no problems!

> *The second pass should take up to an hour for an experienced reader.* (Keshav 2016)

This does not include the computation time of workflows in a research compendium. Use this time to complete first passes for one or several other compendia. If the software used is familiar, you may attempt to reduce the computation time by sub-setting data or simplifying the workflow. As an author, consider adding a reduced example to your research compendium for easier access by readers.

> *After this pass, you should be able to grasp the content of the paper.* (Keshav 2016)

You should have re-executed the provided workflow or understand why you could not. You should be able to complete the second pass even if you are unfamiliar with the actual language the software is written in or if you are not a developer yourself. However we do recommend not to dive too deep, i.e. not going beyond the provided instructions for the research compendium's workflow. At this stage, it is the author's responsibility to guide you through their work.

Still, you may also face unsolvable problems, like access to specific infrastructure. But if you encounter issues or have questions, you should communicate these to the author, for example in the software's public code repository, if available. It is important to do this respectfully, and give the authors a chance to fix bugs or respond to issues (Kahneman 2014). Also let the authors know if your reproduction was successful, especially if you used a different operating system or software version than reported.

At this point you should be able to judge whether the software works and if it is sustainable. Based on this evaluation you can decide to re-use parts of the analysis, i.e. software, data, or method, for your own work.

> *You should be able to summarize the main thrust of the paper, with supporting evidence, to someone else. This level of detail is appropriate for a paper in which you are interested, but does not lie in your research speciality. Sometimes you won't understand a paper even at the end of the second pass. This may be because the subject matter is new to you, with unfamiliar terminology and acronyms. Or the authors may use a proof or experimental technique that you don't understand, so that the bulk of the paper is incomprehensible. The paper may be poorly written with unsubstantiated assertions and numerous forward references.* (Keshav 2016)

The research compendium may have incomplete documentation, rely on unavailable software (e.g. proprietary) or data (e.g. sensitive), or require infrastructure not available to you (e.g. high-performance computing, HPC). It may use a programming language or programming paradigms unfamiliar to you.

> *Or it could just be that it's late at night and you're tired. You can now choose to: (a) set the paper aside, hoping you don't need to understand the material to be successful in your career, (b) return to the paper later, perhaps after reading background material or (c) persevere and go on to the third pass.* (Keshav 2016)

2.3 The third pass

> *To fully understand a paper, particularly if you are a reviewer, requires a third pass. The key to the third pass is to attempt to virtually re-implement the paper: that is, making the same assumptions as the authors, re-create the work. By comparing this re-creation with the actual paper, you can easily identify not only a paper's innovations, but also its hidden failings and assumptions. This pass requires great attention to detail.* (Keshav 2016)

If a best practice or established convention for structuring data and code was followed, familiarise yourself with it now.

> *You should identify and challenge every assumption in every statement. Moreover, you should think about how you yourself would present a particular idea. This comparison of the actual with the virtual lends a sharp insight into the proof and presentation techniques in the paper and you can very likely add this to your repertoire of tools.* (Keshav 2016)

Take a close look at data, metadata, source code including the embedded code comments, and further documentation. You now leave the realm of the mere software user to the developer's perspective. This can be a time consuming very close study of the materials. If data is not publicly available, e.g. because it contains information about human subjects, decide if you have a reasonable request to contact the original authors and ask for data access. Work though the examples and analysis scripts included in the research compendium. Play close attention not only to code, but also to code comments as they should include helpful information. A good entry point for your code read may be a "main" script (if provided by the author), makefile, or literate programming document (e.g. an R Markdown file or Jupyter Notebook). If neither of these are available, then start with the code creating the figures for the article (e.g. look for "`plot`" statements in the code) and trace your way back through the code until you reach a statement where the input data is read. Your impression of the code can help to inform your impression of the article's quality.

If you did not succeed before but the work is relevant for you, spend more time on getting the analysis to run on your computer. Do not hesitate to contact the authors of the paper or authors of the software for help, but follow common error reporting guidelines (e.g. Stack Overflow (2018) or Tatham (n.d.)). For authors it is a great experience to be contacted by an interested and respectful reader!

With regard to the analysis, you may re-implement core parts or the full workflow with a different software. For example, using a tool you know but which was not used in the research compendium. Does your code lead to the same results, or does it give different ones? Can the differences be explained or are they not significant? Note that such a replication is of very high value for science and you should share your findings with the research compendium's authors and also with the scientific community. Depending on the efforts you put in, write a blog post or even publish a replication research compendium for one or more evaluated research compendia.

If a full replication is not feasible, explore the assumptions you challenge with data and code. Play around with input parameters to get a feel for the changing results. Create exploratory plots for the data as if you would want to analyse it from scratch, without the knowledge of the existing workflow. With your understanding of the code you can extend the method to a new problem or apply it to a different dataset. This deep evaluation of code and data increases your understanding of the authors' reasoning and decisions, and may lead to new questions.

To make sure you can trace your own hands-on changes with the original code and configuration. We recommend initiating a local git repository when starting this pass. You can create branches for specific explorations and easily reset to the original functional state.

> *During this pass, you should also jot down ideas for future work. This pass can take many hours for beginners and more than an hour or two even for an experienced reader. At the end of this pass, you should be able to reconstruct the entire structure of the paper from memory, as well as be able to identify its strong and weak points. In particular, you should be able to pinpoint implicit assumptions, missing citations to relevant work, and potential issues with experimental or analytical techniques.* (Keshav 2016)

You should be able to come up with useful extensions of the used software stack and be able to judge the transferability and reusability of the analysis' building blocks. You should most certainly have improved your programming skills by reading and evaluating other people's code or even trying to extend or improve it.

3. Doing a literature survey

> *Paper reading skills are put to the test in doing a literature survey. This will require you to read tens of papers, perhaps in an unfamiliar field. What papers should you read? Here is how you can use the three-pass approach to help. First, use an academic search engine such as Google Scholar or CiteSeer and some well-chosen keywords to find three to five recent highly-cited papers in the area.* (Keshav 2016)

No search capability comparable to scientific articles exists for research compendia, though you can of course use generic and academic search engines. More and more journals encourage reproducible research and software and data publication, so that extending your search regular search with keywords such as "reproduction", "reproducible", "open data/software/code" may improve your results.

In addition, you can search online platforms where research compendia have been published and tagged as a research compendium (`research-compendium`):

- GitHub label: https://github.com/topics/research-compendium
- Zenodo community: https://zenodo.org/communities/research-compendium

There is no journal specifically for research compendia yet, but the following ones feature reproducibility, computational studies, or openness in a prominent way and can be a starting point for finding research compendia, if they fit your topic:

- *ReScience*: https://rescience.github.io/
- *Information Systems* has a reproducibility editor and special track for invited reproducibility papers: https://www.journals.elsevier.com/information-systems/

A lateral approach takes advantage of the parts of a research compendium. If you work with a specific software (tool, extension package, library) or data, find out the recommended way to cite it (and follow it yourself). Most scientific software provides this information in their FAQ or might have a built-in function to generate a citation. Scientific data is often accompanied by a "data paper" or published in repositories with citeable identifiers. Then search for recent publications which cite the referenced software or data.

> *Do one pass on each paper to get a sense of the work, then read their related work sections. You will find a thumbnail summary of the recent work, and perhaps, if you are lucky, a pointer to a recent survey paper. If you can find such a survey, you are done. Read the survey, congratulating yourself on your good luck. Otherwise, in the second step, find shared citations and repeated author names in the bibliography. These are the key papers and researchers in that area.*

You can also find shared software or data and use them as a seed for a next iteration.

> *Download the key papers and set them aside. Then go to the websites of the key researchers and see where they've published recently. That will help you identify the top conferences in that field because the best researchers usually publish in the top conferences.*

Also check where they publish their code and data. It will give you an idea where this community interacts online and can even lead you to research compendia under development.

> *The third step is to go to the website for these top conferences and look through their recent proceedings. A quick scan will usually identify recent high-quality related work. These papers, along with the ones you set aside earlier, constitute the first version of your survey. Make two passes through these papers. If they all cite a key paper that you did not find earlier, obtain and read it, iterating as necessary.* (Keshav 2016)

If a majority cites or uses a key software, technology, or dataset, then evaluate it and include it in the next iteration.

4. Related work

> *If you are reading a paper to do a review, you should also read Timothy Roscoe's paper on "Writing reviews for systems conferences" (Roscoe 2007). If you're planning to write a technical paper, you should refer both to Henning Schulzrinne's comprehensive web site (Schulzrinne n.d.) and George Whitesides's excellent overview of the process (Whitesides 2004). Finally, Simon Peyton Jones has a website that covers the entire spectrum of research skills (Peyton Jones n.d.). Iain H. McLean of Psychology, Inc. has put together a downloadable 'review matrix' that simplifies paper reviewing using the three-pass approach for papers in experimental psychology (McLean 2012), which can probably be used, with minor modifications, for papers in other areas.* (Keshav 2016)

We are working on an extended version of this matrix to provide space for notes about software, data, results of the reproduction, and application of the methods. See the corresponding repository issue for details and provide your feedback: https://github.com/nuest/how-to-read-a-research-compendium/issues/2

If you are reviewing a research compendium, a more detailed checklist is given in the "rOpenSci Analysis Best Pratice Guidelines" (rOpenSci 2017), which are partially even automated for R-based research compendia (DeCicco et al. 2018), and the Journal of Open Research Software's guidelines for reviewing research software (JORS Editorial Team 2018).

5. Acknowledgements

> *The first version of this document was drafted by my students: Hossein Falaki, Earl Oliver, and Sumair Ur Rahman. My thanks to them. I also benefited from Christophe Diot's perceptive comments and Nicole Keshav's eagle-eyed copy-editing. I would like to make this a living document, updating it as I receive comments. Please take a moment to email me any comments or suggestions for improvement. Thanks to encouraging feedback from many correspondents over the years.* (Keshav 2016)

In the spirit of the original paper, we would like to make this a living document and invite readers to provide comments or suggestions for improvement via email, as part of this preprint, or on the GitHub repository: https://github.com/nuest/how-to-read-a-research-compendium. The repository also includes open questions and is where the paper's authors openly discuss.

References

Barba, Lorena A. 2018. "Terminologies for Reproducible Research." *arXiv:1802.03311 [Cs]*, February. http://arxiv.org/abs/1802.03311.

Barnes, Nick. 2010. "Publish Your Computer Code: It Is Good Enough." *Nature News* 467 (7317):753–53. https://doi.org/10.1038/467753a.

Brinckman, Adam, Kyle Chard, Niall Gaffney, Mihael Hategan, Matthew B. Jones, Kacper Kowalik, Sivakumar Kulasekaran, et al. 2018. "Computing Environments for Reproducibility: Capturing the 'Whole Tale'." *Future Generation Computer Systems*, February. https://doi.org/10.1016/j.future.2017.12.029.

Bryan, Jenny. 2015. "Naming Things." *Speaker Deck*. https://speakerdeck.com/jennybc/how-to-name-files.

Chirigati, Fernando, Rémi Rampin, Dennis Shasha, and Juliana Freire. 2016. "ReproZip: Computational Reproducibility with Ease." In *Proceedings of the 2016 International Conference on Management of Data*, 2085–8. SIGMOD '16. New York, NY, USA: ACM. https://doi.org/10.1145/2882903.2899401.

Claerbout, Jon. 1994. "Seventeen Years of Super Computing." http://sepwww.stanford.edu/sep/jon/nrc.html.

DeCicco, Laura, Noam Ross, Alice Daish, Molly Lewis, Nistara Randhawa, Carl Boettiger, Nils Gehlenborg, Jennifer Thompson, and Nicholas Tierney. 2018. "Checkers: Automated Checking of Best Practices for Research Compendia." rOpenSci Labs. https://github.com/ropenscilabs/checkers.

Gentleman, Robert, and Duncan Temple Lang. 2007. "Statistical Analyses and Reproducible Research." *Journal of Computational and Graphical Statistics* 16 (1):1–23. https://doi.org/10.1198/106186007X178663.

Green, Seth Ariel, and April Clyburne-Sherin. 2018. "Computational Reproducibility via Containers in Social Psychology." *PsyArXiv*, February. https://doi.org/10.17605/OSF.IO/MF82T.

Ioannidis, John P. A. 2014. "How to Make More Published Research True." *PLOS Medicine* 11 (10):e1001747. https://doi.org/10.1371/journal.pmed.1001747.

Jimenez, I., M. Sevilla, N. Watkins, C. Maltzahn, J. Lofstead, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. 2017. "The Popper Convention: Making Reproducible Systems Evaluation Practical." In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 1561–70. https://doi.org/10.1109/IPDPSW.2017.157.

JORS Editorial Team. 2018. "Journal of Open Research Software - Editorial Policies, Peer Review Process." http://openresearchsoftware.metajnl.com/about/editorialpolicies/.

Kahneman, Daniel. 2014. "A New Etiquette for Replication." *Soc. Psychol.* 45 (4):310.

Keshav, S. 2007. "How to Read a Paper." *SIGCOMM Comput. Commun. Rev.* 37 (3):83–84. https://doi.org/10.1145/1273445.1273458.

———. 2016. "How to Read a Paper." Manuscript. Waterloo, ON, Canada. http://blizzard.cs.uwaterloo.ca/keshav/home/Papers/data/07/paper-reading.pdf.

Kluyver, Thomas, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonier, Jonathan Frederic, Kyle Kelley, et al. 2016. "Jupyter Notebooks - a Publishing Format for Reproducible Computational Workflows." *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, 87–90. https://doi.org/10.3233/978-1-61499-649-1-87.

Knuth, Donald E. 1984. "Literate Programming." *Comput. J.* 27 (2):97–111. https://doi.org/10.1093/comjnl/27.2.97.

Konkol, Markus, and Christian Kray. 2018. "In-Depth Examination of Spatio-Temporal Figures in Open Reproducible Research." *EarthArXiv*, April. https://doi.org/10.17605/OSF.IO/Q53M8.

Konkol, Markus, Christian Kray, and Max Pfeiffer. 2018. "The State of Reproducibility in the Computational Geosciences." https://doi.org/10.17605/osf.io/kzu8e.

Marwick, Ben, Carl Boettiger, and Lincoln Mullen.

2018. “Packaging Data Analytical Work Reproducibly Using R (and Friends).” *The American Statistician* 72 (1):80–88. https://doi.org/10.1080/00031305.2017.1375986.

McLean, Iain H. 2012. *Literature Review Matrix*. http://archive.org/details/LiteratureReviewMatrix.

Nosek, B. A., G. Alter, G. C. Banks, D. Borsboom, S. D. Bowman, S. J. Breckler, S. Buck, et al. 2015. “Promoting an Open Research Culture.” *Science* 348 (6242):1422–5. https://doi.org/10.1126/science.aab2374.

Nüst, Daniel, Markus Konkol, Edzer Pebesma, Christian Kray, Marc Schutzeichel, Holger Przibytzin, and Jörg Lorenz. 2017. “Opening the Publication Process with Executable Research Compendia.” *D-Lib Magazine* 23 (1/2). https://doi.org/10.1045/january2017-nuest.

Pebesma, Edzer. 2013. “Earth and Planetary Innovation Challenge (EPIC) Submission "One-Click-Reproduce".” http://pebesma.staff.ifgi.de/epic.pdf.

Peyton Jones, Simon. n.d. “Simon Peyton Jones at Microsoft Research.” *Simon Peyton Jones at Microsoft Research*. Accessed May 25, 2018. https://www.microsoft.com/en-us/research/people/simonpj/.

Prlić, Andreas, and James B. Procter. 2012. “Ten Simple Rules for the Open Development of Scientific Software.” *PLOS Comput Biol* 8 (12):e1002802. https://doi.org/10.1371/journal.pcbi.1002802.

rOpenSci. 2017. “rOpenSci Analysis Guide (Unconf 2017).” *Google Docs*. https://docs.google.com/document/d/1OYcWJUk-MiM2C1TIHB1Rn6rXoF5fHwRX-7_C12Blx8g/edit?usp=embed_facebook.

Roscoe, Timothy. 2007. “Writing Reviews for Systems Conferences,” March, 6. https://people.inf.ethz.ch/troscoe/pubs/review-writing.pdf.

Sandve, Geir Kjetil, Anton Nekrutenko, James Taylor, and Eivind Hovig. 2013. “Ten Simple Rules for Reproducible Computational Research.” *PLoS Comput Biol* 9 (10):e1003285. https://doi.org/10.1371/journal.pcbi.1003285.

Schulzrinne, Henning. n.d. “Writing Systems and Networking Articles.” Accessed May 25, 2018. https://www.cs.columbia.edu/~hgs/etc/writing-style.html.

Stack Overflow. 2018. “How to Create a Minimal, Complete, and Verifiable Example.” *Stack Overflow*. https://stackoverflow.com/help/mcve.

Stodden, Victoria. 2009. “The Legal Framework for Reproducible Scientific Research: Licensing and Copyright.” *Computing in Science & Engineering* 11 (1):35–40. https://doi.org/10.1109/MCSE.2009.19.

Stodden, Victoria, and Sheila Miguez. 2014. “Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research.” *Journal of Open Research Software* 2 (1). https://doi.org/10.5334/jors.ay.

Stodden, Victoria, Sheila Miguez, and Jennifer Seiler. 2015. “ResearchCompendia.Org: Cyberinfrastructure for Reproducibility and Collaboration in Computational Science.” *Computing in Science & Engineering* 17 (1):12–19. https://doi.org/10.1109/MCSE.2015.18.

Taschuk, Morgan, and Greg Wilson. 2017. “Ten Simple Rules for Making Research Software More Robust.” *PLOS Computational Biology* 13 (4):e1005412. https://doi.org/10.1371/journal.pcbi.1005412.

Tatham, Simon. n.d. “How to Report Bugs Effectively.” Accessed June 11, 2018. https://www.chiark.greenend.org.uk/~sgtatham/bugs.html.

Vandevalle, Patrick. 2012. “Code Sharing Is Associated with Research Impact in Image Processing.” *Computing in Science & Engineering* Reproducible Research for Scientific Computing (July):42–47.

Whitesides, G. M. 2004. “Whitesides’ Group: Writing a Paper.” *Advanced Materials* 16 (15):1375–7. https://doi.org/10.1002/adma.200400767.

Wilson, Greg, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, and Tracy K. Teal. 2017. “Good Enough Practices in Scientific Computing.” *PLOS Computational Biology* 13 (6):e1005510. https://doi.org/10.1371/journal.pcbi.1005510.

17 Synopsis

17.1 Infrastructure & user experience

The infrastructure designed and built during this dissertation ~~must~~ serve the needs of the users. Therefore, research question **IUE1** addresses these user groups, and we discuss how reproducibility practices and the concrete realisation of the ERC as a packaging mechanism helps each type of user.

How can packaging of computational analyses serve the needs of authors, publishers, readers, and preservationists?

For **authors**, notebooks and containers are an important tool for reproducibility, and using them is a good idea even without ERCs as the final product of research. As shown in Chapter 4, packaging a complex multi-step analysis openly creates new possibilities, for example parametrisation of an analysis in an accessible way by extending a graphical UI for container management with a model option form. However, as the examples in Chapter 10 show, ~~the~~ reproducibility largely depends on the authors' individual motivations and skills. That chapter presents a number of challenges for practical reproducibility in geography and geosciences: the complex combinations of many tools, some of which are proprietary, as well as the artistic process of creating maps hamper reproducibility, processing infrastructures can be opaque, and partly irreproducible methods (e.g., due to sensitive data) are sometimes seen as a get-out-of-jail card to dismiss reproducibility altogether. However, the chapter also shows that solutions do exist, though establishing them is not in the sole responsibility of individual authors, because different efforts by authors should not be disadvantageous compared to other members of their community ~~despite.~~ Individual efforts for better reproducibility are wrongly perceived as additional work when reproducible research actually is more efficient. Furthermore, these challenges can be mitigated by infrastructure, such as mechanisms that provide controlled access to sensitive data or to the reproducibility service. The method of automated environment capturing offered by the reproducibility service is crucial to serve the broader community of geographers and geoscientists, who are generally not (yet) experts in applying computational methods in a reproducible way. A core feature of the ERC is its minimalist approach, which allows authors not versed in computational reproducibility to just rely on the ERC reproducibility service to create their ERC, but it also allows authors more familiar with these tools to create handcrafted workflows and write the computing environment manifest and ERC configuration (Dockerfile and erc.yml) themselves. The ERC values simplicity and transparency over complexity. It does not introduce abstractions that merely hide the uncertainty. Further, with suitable default commands for the container, readers could still easily inspect and interact with features of the ERC reproducibility service when manually built ERCs are examined. Yet, the minimum requirement for this ERC approach still requires authors to provide a fully scripted workflow. And, considering the conclusions of Chapter 11, which presents a vision for the skill set that authors of the future will have, a fully scripted workflow is the foundation for more effective and fulfilling scientific authorship. Authors benefit from workflows that foster collaboration with their readers and are more convincing to reviewers.

Readers are the stakeholders with the most obvious benefits, as the ERC should help them understand a piece of research ~~much~~ more ~~readily~~ because they will have access to literate programming documents, can inspect all data and code, and will be able to interact with workflows based on bindings. A particularly considered reader for the o2r project is the **reviewer**. Chapter 13 argues that matching skills for reviewing computational parts of research articles is hard and reviewer time precious. The infrastructure and user experience provided by the ERC can, to some extent, mitigate these problems and make reviews more efficient. Naturally, the accessibility of work must be a core concern of all stakeholders, but the burden to serve the needs of readers and reviewers always lies with the author and can be facilitated by infrastructure (and incentives and policy). As the ERC is the foundation for more collaboration, even as early in the publication process as between authors and reviewers, it can turn readers into contributors and, thereby, provide benefits for all. The required openness also removes the barriers of proprietary software, making the criterion of reproducibility a means to ensure reusability, e.g., scripts over point-and-click GIS tools (cf. Chapter 10). The GEOBIA workflow (see Chapter 4) was a breakthrough for transparency, reproducibility, and reusability in a domain largely relying on one commercial tool. The article demonstrates that containers can be sued for a specific open and reproducible geoscientific analysis in the domain of remote sensing. It showcases the potential of containerisation to ease reproducibility and understandability while simultaneously presenting the successful creation of a reproducible workflow based on a very complex combination of FOSS tools.

Publishers are in a challenging situation in academia, but different types of publishers face different challenges and for different reasons. For small publishers or independent journals, it is extremely hard to innovate and to finance one-click reproductions of articles. Nevertheless, these journals as well as independent publishers may attempt reproducible articles for the mere reason of wanting to do science properly. It is no surprise that a non-profit organisation with a mandate to innovate openly, eLife[11] is one of the leaders when it comes to novel ways to publish reproducible research. While large publishers should have the funds for innovation, they face organisational inertia and the growing conflict between commercial publishing and free academic labour. Especially commercial publishers have no incentive to invest in complex technical infrastructure when there is no clear monetary benefit. What unites both publisher types (and everything in-between) is the general current practice of academic publishing (simplified: research, submit, review, publish, repeat), and the presented infrastructure matching this practice. However, the reproducibility service and ERC user interface are also suitable for novel more flexible, piece-wise, continuous, or open publication patterns. Even for post-publication or preprint peer review, or for separated reviews of research ideas and methods, a snapshot must always be taken for which the review is conducted. The ERC concept and tools (Chapters 3 and 5) support working with such snapshots and provide a novel approach to enhance scientific papers, peer review procedures, and mid-term preservation of computational research than can be connected and integrated in existing platforms and procedures in an evolutionary way. The systematic review of platforms (Chapter 6) shows that while publishers do have options to choose from, the real costs remain hard to estimate. Potentially costly activities include both maintaining the infrastructure for reproducibility and integrating reproducibility requirements into existing workflows and systems, and comprise technological and training investments. These observations are corroborated by many direct conversations with publishers and illustrated by the slow pace of innovation in scientific publishing. Hrynaszkiewicz (2020) provides an excellent analysis of how scholarly publishers can help to increase reproducible research, and the identified key areas can be directly linked to chapters of this dissertation. At the intersection of publishers, authors, and readers, **editors** take a key position in establishing new practices, as they can engage all levels of the culture change pyramid.

Preservationists are stakeholders that are most distant from the regular realm of researchers and have the least leverage, one might even say the least thankful role. The preservation of workflows was especially considered in the early phase of this dissertation (cf. Chapter 3). The two-container concept of the ERC (see Figure 1 on page 71), with the inner (Docker) container holding the computing environment and the outer container holding all workflow-specific code and data, ensures that no relevant information is hidden inside the computational container. The outer packaging based on BagIt provides a

[11] https://elifesciences.org/about

connection to preservation workflows, and the preservation of containers reflects ongoing research. Regular preservation mechanisms, such as emulation, can leverage containers. However, the perspective to "preserve" or "archive" ERCs was refined over the course of the dissertation because putting something into an archive means not *using* it anymore; importantly, usage and extension is where ERCs provide the most benefits. The plain-text simple nature of the ERC configuration and the semantically meaningful bindings as entry points into a workflow are helpful for humans even when the ERC tools and platforms may no longer be available. Even though preservation of ERCs could be solved in future work, the more interesting aspect is short to mid-term reproducibility, say around 10 years. This might already suffice to enable more reuse and transparency when it matters, and, ultimately, better quality of the preserved works. While running code that is older is a good thing, actual reuse and expansion become unlikely and maintenance requires large efforts (Peer et al., 2021), thereby making a reimplementation more reasonable. As notebooks are even explored as an alternative to classic papers[12], preservationists may turn to the ERC concept and its implementation because they can facilitate creating snapshots of the notebook environment. This way, they can preserve the interactive nature of notebooks when they are first-level items in scholarly communication.

Although researchers will inevitably need more software skills to conduct their work in a transparent and reusable way, the current situation in terms of education, publishing, and assessment will not be quickly resolved; instead, it is a generational question. Until research software topics and reproducibility have permeated education and evaluation of all scientific disciplines, authors and infrastructure providers will have to find a middle ground regarding what the author must provide so that editors, reviewers, and publishers can still check and support reproducibility. Therefore, research question **IUE2** concerns the level of automation needed to capture workflows. The presented infrastructure and key tool, containerit, presented in Chapters 5 and 7, show that if an author can be asked to provide a fully scripted workflow, then *one can fully automate the ERC creation.* Manual steps might be needed only to collect correct metadata. This scripted workflow may also be a regular R script, and R Markdown documents may also be used to execute workflows based on other languages than R, serving as a, possibly thin, integration layer. The sharing of computational research around big datasets remains a large challenge. The architecture allows for avoiding data duplication if data is too large and for connecting to existing established (open) data infrastructures, in which a community might have made a strong investment. The creation of ERC is only possible with modern containerisation technology. Containers are an immensely useful technology for improving reproducibility of data-based research, as demonstrated by the large survey paper (Chapter 9) aimed at understanding the whole breadth of use cases based on containers in R. The same advantages that are crucial for publishing computationally reproducible papers apply to education or production environments: Across use cases and approaches, the reproducibility, portability, well-definedness, and ease of use of computing environments form the common thread. Nevertheless, the same paper also points out that the fragmentation and doubling of related efforts hamper the sustainability of a diverse collection of largely unfunded tools. And while the paper ends with a call for consolidation, researcher freedom mandates that we must also explore the long tail of geospatial workflows based on the R language and how containers can capture them (Sunni, 2020). So, while automation can work in most cases, the system needs to allow handcrafted environments. The question of how to make using both the automatically and the bespoke environments safe for readers and operators was not answered in this work beyond authenticating ERC users with real profiles based on ORCID.

To what extent can the process of capturing the runtime, software, data, and metadata of reproducible research packages be automated in geoscientific analyses?

[12] https://www.earthcube.org/2021-earthcube-annual-meeting

The ERC was not created in a vacuum. Alternatives to the ERC web service (some of which are commercial or closed platforms) have been developed and adopted over recent years, which points towards the increasing awareness of the need to share computational workflows with papers. At the same time, the debate about financing scholarly publishing (e.g., increasing use of free preprint and review servers, overlay/independent journals with minimal infrastructure costs, transformative agreements, and transition to Green/Gold/Diamond OA) changes the landscape of services and platforms. Therefore, research question **IUE3** asks how the ERC fits into the existing publishing practices, and we should extend the question to review how it fits the continuously evolving publishing norms and practices. Of course, the ERC does not solve all issues around communicating research results, namely the sharing of private data, but that is an organisational challenge not a technical one (cf. Chapter 10). o2r's fully open approach, i.e., all specifications and software are openly shared, and the option to substitute inputs and easily create interactive figures sets it apart from all but one other solution. Further, the ERC is still unique in its approach for capturing the computing environment through executing full workflows and storing both manifest and image, and for exposing key parameters through bindings. However, the o2r system is also still a research prototype, while other services and tools are commercial products actively deployed by large publishers or are much more widely used. And even a generic technology like containerisation is subject to technological drift and even popularity, and needs to be successful to be supported and maintained (Peer et al., 2021).

How can the ERC fit into the existing infrastructure of services and platforms for research and publishing in geography and geosciences?

Nevertheless, in Nüst & Pebesma (2020) we show that the technical challenges to create, share, and review reproducible computational workflows can be overcome with today's technology. Containerisation and computational notebooks in general are very well suited to realise a highly reproducible scholarly discourse based on data and code, yet they should be accompanied by good documentation—a README is always important. These tools are also likely to be actively maintained for the foreseeable future. This finding also applies to geography and geosciences. The *specifics of geospatial sciences* for reproducibility do exists, but for the most part they are not any harder to solve or any different than in other disciplines. Maps, as an important means to communicate results, are still often created with point-and-click software and not scripted, but that is a social and educational question, as technological alternatives exist. The history of standardised data access and the more recent availability of big Earth data complicate things and might make it necessary to more routinely "break up" the ERC to allow access to trusted repositories, but that is also feasible. Notebooks and other open source script-based tools are increasingly used for geospatial visualisation and are well suited for sharing reproducible computational workflows if used right (Rowe et al., 2020; University of Southern California et al., 2021).

In conclusion, practically all chapters of this dissertation call for an adoption of the ready-to-use methods to overcome barriers and transform scientific communication in the geospatial disciplines, and we can extend this call to include research compendia in general and the ERC specifically. *To summarise, the ERC concept and tools can be integrated into publishing platforms in geography and geosciences to evaluate computational snapshots of research works in a way that suits all stakeholders.*

17.2 Communities, incentives & policy

The top-down perspective on reproducibility in geography, geoscience, and GIScience is guided by the communities and their self-given practices, including norms for mutual recognition of research outputs and written policies. The studies to assess the state of reproducibility in GIScience (Chapters 11 and 12) do not leave much room for interpretation—reproducibility is quite low, even when looking back just a few years. Both articles investigating the state-of-the-art in GIScience conference publications show that, at the time of investigation, none of the assessed 107 papers provided sufficient details and references to deposits of data or code to allow a reproduction to be practical, i.e., without intensive interaction with the authors or without substantial efforts to recreate large parts of the workflow. The chapters' main Figures 3 on page 171 respectively 1 on page 191 illustrate these results clearly. For the vast majority of papers, this is the case even at the time of publication. Figure 4 shows the main patterns for assessed papers of both conferences. Noteworthy are the complete lack of papers that achieve the highest level of 3 (open and published permanently) in any of the categories. While the majority of papers describe methods and results sufficiently to make a recreation in principle possible (level 1), about half of the papers do not give enough details to access data and about two thirds do not document the computational environment at all (level 0). Within the smaller groups that achieve levels of 2 (i.e., access provided, but not permanently and open) the figure shows that there's plenty of jumping between levels and most combinations seem to exist. This is a sign of the lack of common practice and the differing opinions and understanding of authors concerning good and open enough communication.

The assessments' core takeaway is that unavailability of data and code and incompleteness of methodological details are insurmountable barriers to reproductions. We largely attribute this to a lack of recognition and requirements, so the articles give concrete recommendations for improvement to authors and conference organisers and speak directly to the community members. It should be noted that both works clearly separate the validity of results from the reproducibility of workflows (see also Riet et al., 2019). The chapter's recommendations are, to a large extent, based on the lessons learned from the assessments itself and the experiences made throughout working on this dissertation, but also on the literature from other disciplines. The need for communities to step up is also argued clearly in the conclusions of the above chapters, but especially in Chapter 10, which concludes with goals that might seem lofty, but are without any alternative: *"[..] the benefits of working reproducibly are by now clear. Technical, systemic, and cultural barriers are conquerable. The scientific community should embrace the disruptions in scholarly publishing [..] by setting up new [..] standards for scholarly communication. The maxim [..] should be open source software implementing an open and self-correcting public infrastructure controlled by scientists."* (Nüst & Pebesma, 2020).

The Reproducible AGILE initiative shows how quickly practices can change, provided that the topic finds champions in the community as well as institutional support. Initiated by a series of workshops, a community adopted policies that require transparency and reward openness and reproducibility in just a few years—the start of cultural change happens at an observable speed. In its first two years, the reproducibility review confirmed 14 successful reproductions[13], and all published full papers in 2021 had to make data and software (un)availability explicit. The fact that these 14 papers could be reproduced strongly suggests that all of them, unless data could only shared privately, would likely reach at least a level of 2 across all categories, and at least a few properly deposited material to score 3s—a clear improvement over the earlier assessed works. The main findings of the reproducibility review were that a large spectrum of reproducibility exists, that the direct communication between author and reproducibility reviewer and the subsequent immediate impact of recommendations on submissions (i.e., changes made before final publication) is beneficial and rewarding, and that the main challenge for reproducibility reviewers remains documentation (e.g., undocumented runtime, lack of links between code/data artefacts and paper, incompleteness) but not at all unassertive behaviour or doubts on the side of authors (Nüst, Ostermann, et al., 2020). With the organisation of workshops, the assessment of the state of reproducibility and lobbying with the community leadership, the need for change and the practicality

[13]https://agile-giss.copernicus.org/articles/

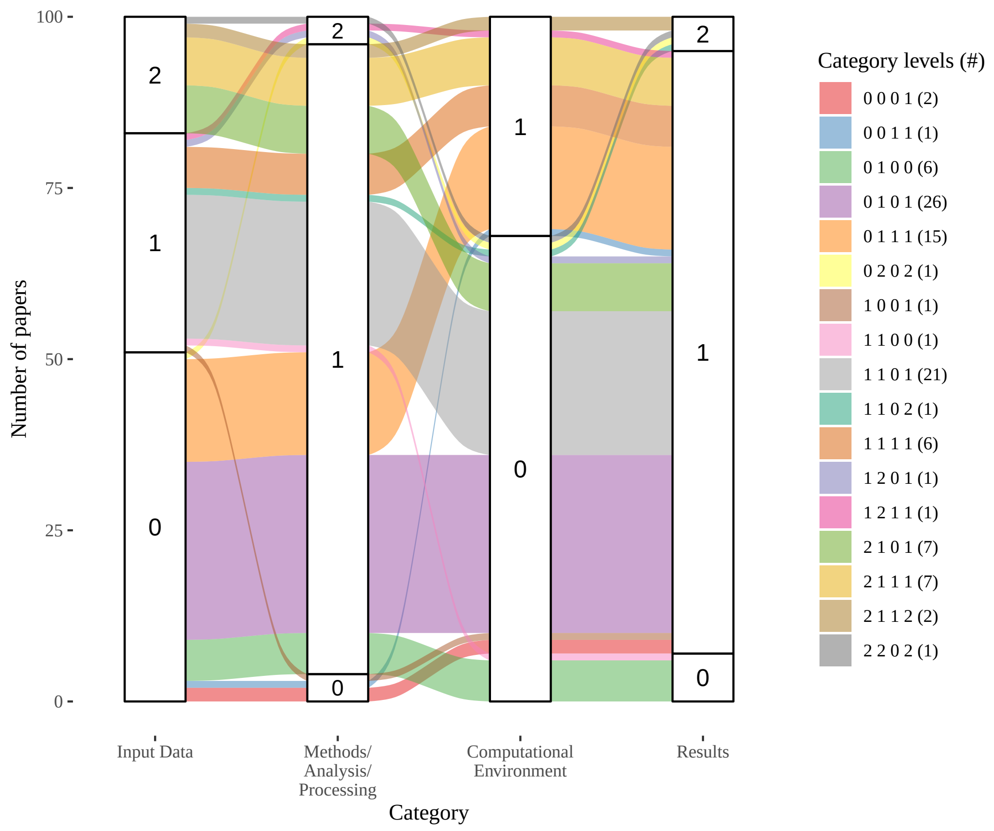


Figure 4: Combined alluvial diagram. Includes groups of papers across four categories for the merged AGILE (Chapter 11) and GIScience (Chapter 12) datasets; the category *Preprocessing* was dropped because of difficulties to clearly assess it; included are 100 papers without any “not applicable” value from 2010 to 2018; see analysis notebook.

of reproducible methods was conclusively demonstrated to the community, and the focus on positive encouragement works at least so far and for this particular setting (Nüst, Ostermann, et al., 2020; Nüst, 2021d). Nevertheless, the discourse in the community continues and what “reproducibility” means concretely for the discipline and event is evolving.

Because the recommendations given across all chapters draw from the lessons learned from other disciplines, they are strong and promise sustainability through widespread adoption across all scientific disciplines. Only small domain-specific challenges were identified (cf. Chapter 10), and the solutions are, in general, not specific to geospatial sciences, answering **CIP1**. The domain-specific practices resolve around common practices of map-making, e.g., the broad application of largely point-and-click GIS tools, and shortcomings in education and the habit of little scrutiny of published works. Even for the AGILE conference and community, the recommendations and conclusions presented in Chapter 11 leave some goals yet not reached. Tobler’s first law of geography[14] does lead to interesting questions on the replication of geolocation-based research, but, generally, the challenges and solutions—both technical and social—do not set geography, geosciences, and GIScience apart from other disciplines.

What are domain-specific challenges and solutions for the geosciences domain in the context of reproducible publications?

Since most challenges are shared with other disciplines that use observational or simulated data, and because more and more disciplines are employing geospatial data and looking to geospatial communities for guidance, it is reasonable to connect with interdisciplinary approaches to improve reproducibility. One such initiative is CODECHECK, which employs a more evolutionary and much less technical approach to increase awareness and recognition for reproducibility, and, thereby, complements the developed solutions driven by technological advances. Chapter 13 analyses the options of integrating code execution into peer review, the required roles, and barriers as well as opportunities for involved stakeholders. These provide a toolbox for stakeholders in (geospatial) publishing (e.g., editors, publishers) to start requiring and evaluating the reproducibility of submissions. To date, 25 CODECHECK certificates have been created. These certificates demonstrate the value and a reasonable scope of the execution of workflows by an independent person as part of the peer review process. Both Reproducible AGILE and CODECHECK show how important it is to complement technical solutions with contributions to education and lobbying for policy changes.

Certain incentives for researchers to work more reproducibly and for communities to adopt reproducible practices and introduce them into policies include novel services and features that would not be possible with classic, irreproducible sharing of research outputs and merely text-based communication of results. To answer **CIP2**, this dissertation presented the ERC itself and a platform for ERC creation and inspection. Furthermore, it looked into downstream applications that demonstrate advantages of applications based on sharing research works in the form of (E)RCs. *What would be possible when researchers create and publishers publish ERCs but not PDFs?*

What new services and features can be built upon reproducible workflows, e.g., when packaged as an ERC?

Chapter 14 is based on a supervised thesis (Niers, 2020) and gives a preview into the options available when using an ERC by presenting the manual capture of geospatial metadata for articles in a journal and showing how these can be displayed on a journal website. With ERC, such a process could be completely automated. The research project OPTIMETA will extend the early *geoOJS* prototype presented in this work into a full-featured OJS plugin with search using spatial indexes and a cross-journal discovery platform.
Extending on the blog post Reproducible Research Badges and presenting the results of a study project of the same name, Chapter 15 describes an independent solution to integrate

[14]https://en.wikipedia.org/wiki/Tobler%27s_first_law_of_geography

metadata as badges on articles and reproducible research across a number of online portals to improve discovery of scientific works. It demonstrates the relevant contributions that a reproducibility service, described in Chapter ??, can make to scientific communication, namely by providing relevant metadata regarding executability. It also shows how new technologies can impact established practices on the user interface side, which is otherwise out of the scope of this dissertation. A supervised thesis demonstrated the scalable on-demand creation of badges based on complex metadata (Graupner & Nüst, 2020) and could be adapted to ERC badges.
The third downstream usage of ERCs, Chapter 16, builds on the idea that when research compendia gain more traction, readers need to change the ways they approach a piece of literature, which applies not just to reading but to interacting, using, extending, etc. The preprint attempts to provide guidance for both authors and readers of a research compendium. The text offers reflections on the potential impact that (executable) research compendia can have on science communication and contributes to their advancement. These showcases merely scratch the surface of ERC-based scholarly communication, and more ideas are presented in the Outlook.

The balance between technological solutions and adopting new ways to communicate and collaborate is deeply embedded in the culture change pyramid. This balance is reflected in this dissertation by the inclusion of both the innovative, technology-driven approach for scientific commons based on ERCs and the "less is more" approach that highlights the human interaction and community practices with CODECHECK and Reproducible AGILE. Besides, a number of talks, events, posts and collaborations[15] (Nüst, 2019a, 2021a, 2021b, 2018, 2017b, 2019b, 2020b, 2020a, 2021e; Nüst, Drost, et al., 2021; Nüst, Schutzeichel, et al., 2018; Nüst & Konkol, 2020; Sochat & Nüst, 2021; Steeves et al., 2017) gave us many opportunities to promote reproducibility in general, talk about RCs (without the E), advertise useful practices, or contribute to projects in other disciplines (Breznau et al., 2019, 2021). These texts, software contributions, and presentations are not formally part of this dissertation, but they provided interactions and discussions that were an important source of ideas and a creator of opportunities.

17.3 Conclusion

Cultural change is very slow. The technological innovations and means to influence norms which were identified in this dissertation nevertheless contribute to a wider-scale adoption of reproducible research practices. While the fundamental ideas for reproducibility have been around for a long time, notably with Claerbout & Karrenbach almost 30 years ago (Claerbout & Karrenbach, 1992) and with research compendia first presented in 2007 by Gentleman and Temple Lang (Gentleman & Temple Lang, 2007), the modern technologies transferred to reproducibility tools and scholarly practices in this work provide a new level of accessibility and pathway for reproducible methods. However, the age of the landmark papers also shows that adoption is too slow and a technology-driven approach does not suffice, despite all individual and collective benefits, and despite the progress on Openness (e.g., in geospatial science, Minghini et al., 2020). Therefore, concerted social actions are required as suggested and initiated by this work, so that publishing of research results can catch up

[15]Over 40 entries were contributed to the o2r project blog. The website https://research-compendium.science/ was started to collect all information around research compendia and own research outputs as well as reproducibility basics were presented in a number of keynotes and talks, see https://o2r.info/publications/. Furthermore, the Reproducible Research Support Services (R2S2, <https://go.wwu.de/r2s2>) is part of the o2r project, offering members of the University of Münster consulting services on setting up reproducible workflows, creating research compendia, and it reproduces workflows of manuscripts before submissions to increase trust in the results.

with the methods of conducting research. The technology adoption lifecycle[16] indicates five stages of adoption that any innovation typically goes through: innovators, early adopters, early majority, late majority, and laggards. Even over the short time frame of this dissertation, containerisation technology and reproducibility review practices have arguably left the innovators stage and are being picked up by early adopters. As communities start to shift norms in a way that could lead to a shift in policy, it is not out of line to hope for a majority adoption and to expect more reproducible practices within the next 5-10 years. While working towards these practices, communities will have to continue their discourses about what reproducibility means for them, and how close to original results a reproduction must be to be deemed successful (cf. Riet et al., 2019). The technological advancements and the increase in complexity of research outputs that contributed to the "falling behind" of reproducibility concerns will never stop again, so it is paramount that all ideas and solutions presented here let a human make the final call about reproducibility and judge quality and validity in interplay with computational reproducibility.

To achieve this culture change, a few open questions remain and are presented in the outlook below. To conclude, one must also consider the question of *what is the geoinformatics in all this?* In my understanding, as a field of "hyphenated informatics," geoinformatics researches solutions to geospatial problems using information technology (IT). Geospatial problems often are the questions and issues that geographers and geoscientists face in their work. In the case of the research presented in this dissertation, the issue is reproducibility. Domain-specific solutions using IT regularly require the adaptation and transfer of novel concepts and tools from computer sciences and general IT. In this case, containerisation was applied and made available for a broader community of geospatial disciplines. Of course, the designed infrastructure could just as well be applied to other natural sciences, and the methods to understand and shift community practice are transferable metascience, too. However, directly addressing geographers, geoscientists, and GIScience researchers is needed to communicate the challenges and approaches of reproducible research successfully. Activities like research on reproducibility have a very generic component, nevertheless they are important to carry out in a specific domain so they do not remain a theoretical exercise. The communication with researchers and the need for a practical evaluation require the bridging between informatics and geo-disciplines and the consideration of barriers and opportunities across all levels of the culture change pyramid. That requirement is met by this dissertation's **key contributions**:

- This work realises a functional infrastructure for more reproducible scholarly communication based on the concept of the Executable Research Compendium and demonstrates more open solutions and practices this infrastructure can support.
- It offers innovations in the application of containerisation to the practices of reproducible research, including handcrafted as well as automatically captured computing environments, and it gives overviews of common practices based on containers.
- It highlights the state of reproducibility in GIScience and gives clear steps for stakeholders to improve transparency, reproducibility, and potential for collaboration of research in geography, geoscience, and GIScience.
- It provides approaches to introduce reproducibility in peer review and publishing practices based on numerous reproductions and community involvement with demonstrated success.

[16]https://en.wikipedia.org/wiki/Technology_adoption_life_cycle

17.4 Outlook

Building upon the key contributions and lessons presented in the previous chapters, we see the following promising directions for future work.

Notebook- or ERC-based research is better from an individual perspective, but it also facilitates strategic research building to accumulate evidence, which can more effectively deal with reproducibility challenges (Nichols et al., 2021). With ERCs, we don't just publish results but collaborate on the advancement of science. The idea of the ERC as a self-contained publishable piece of research output has not lost any appeal, because the scenarios and applications that would benefit from the ERC as the "unit of publishing" are manifold. Beyond the downstream applications presented above, most benefits lie in the **opportunities to discover, re-combine, and examine ERCs**. In particular, realising the idea of adapted representations depending on the use case as presented in a vision paper (Kray et al., 2019) would be breaking new ground for communication among scientists and with the public. New discovery solutions can take advantage of the meaningful metadata about the method (used software, libraries, computing environment) and data (spatial extent, temporal extent) that can be automatically derived from the contents of ERCs. Based on these metadata, a supervised thesis (Lohoff, 2018) demonstrates a starting point for exploring advanced recommendations of related works that go beyond current standards of keywords and full text search. For journals not yet ready to make the switch to ERCs, an upcoming project will introduce geospatial metadata for journal articles (Hauschke et al., 2021). A remaining open question is the cost of reproducibility. First, *who pays for the computing?* Publishers or universities? Can computational costs be made transparent, or will they be paid through indirect fees, such as publication charges? With limited computing time, one might also ask how to determine what should be reproduced and what should not. Second, *who pays for the maintenance?*. As Peer et al. (2021) point out, reproducibility is dynamic and active maintenance of datasets and code projects is not rewarded but needed, and requires resources, infrastructure, standards, and policies (Peer et al., 2021). The costs, unsurprisingly, reach across all levels of culture change.

With the recent awareness on reproducibility, it is unsurprising that more and more journals and conferences are considering an evaluation computational workflows. Even though hard requirements are still far away, the next generation of researchers will find the sharing and evaluation of data and code much more natural. To support scientific venues in the uptake of reproducibility reviews, we have conceptualised a **survey on code execution during peer review in journals and conferences** (Nüst, Seibold, et al., 2021). The idea is to conduct semi-structured interviews with reproducibility editors of conferences and journals to collect data on how the different processes for workflow evaluation work. The lessons captured in this work will be a great source to develop a shared understanding of what is reasonable and useful to expect and to demand from authors, reviewers, and editors.

The ideas and plans for the **continued development of the o2r platform and ERC concept** are of course too numerous to discuss here. The biggest challenge for a wider adoption are scalability and stability, which are less of a research challenge and more of a software development task. One core idea here would be to re-evaluate other platforms as a basis, e.g., support ERCs in BinderHub (Project Jupyter et al., 2018), rather than implementing yet another scheduling platform for containerised workspaces. The idea of wizard-based creation of interactive bindings is still unique and potentially very useful for a transition to more computation-focused means of publication, such as eLife's ERA[17] or peer-reviewed notebooks[18]. Furthermore, the concept of the ERC could be explored not only to capture

[17]https://elifesciences.org/labs/dc5acbde/welcome-to-a-new-era-of-reproducible-publishing
[18]https://www.earthcube.org/2021-earthcube-annual-meeting

just notebooks and research papers but to enable the portability and archival of other interactive works with spatial data, such as atlases[19] or textbooks.
However, the ERC is not free of challenges. The question of how to preserve complex compounds such as the ERC is not yet answered, and it is still unknown whether archiving computing environments is possible in a useful way. Considering the fact that "archival" normally means "not going to be used anymore," at some age ERCs would become the object of investigation for science historians. Is it feasible to actively check the "health" of a containerised analysis in ERC as part of a deep integrity check in an ERC archive (cf. Peer et al., 2021)? The impact and compatibility of ERCs with existing work on *preservation* of containers and advances in providing easy access to emulation[20] should be explored. Furthermore, the question of effectively and transparently capturing and managing the *licenses* of the myriad parts in an ERC was only barely addressed. The ERC is not the only idea that promises to solve issues related to sharing computational research. It should be a worthwhile exercise to explore *conversions between approaches to package computational workflows* and possibly identify a minimal common ground, so that no "winner" has to be declared but instead the most suitable of compatible tools is used. The shared foundation of containers is a promising start for connecting, e.g., *Whole Tales*, *ERAs*, *ERCs*, and *ReproZip packages*.

These technical approaches must always be accompanied by education and community interaction, otherwise they won't be able to encroach upon the higher layers of the culture change pyramid. The challenge of **achieving policy changes through recognition and wide adoption of code execution** as part of the CODECHECK initiative and Reproducible AGILE, e.g., by providing a codechecking infrastructure, mentoring or support for training ERCs, and building a pool of experienced diverse codecheckers for independent scholarly led diamond OA journals or to preprint servers, reaches across all layers of the culture change pyramid. Just as o2r, it attempts to change academia through peer-review practices, but it builds on top of the technological progress made since the beginning of o2r and this dissertation. A longitudinal study of the impact of CODECHECKs and guidelines for reproducible papers, e.g., by repeating the assessment of GIScience papers in 2023 or by comparing discipline journals with and without code execution regularly, could powerfully test and document the effects of policy changes and community discourse.

How can the case for reproducibility be made stronger? While wider adoption of reproducibility practices will increase transparency and reusability, and thereby reduce duplicated and positively impact quality, improving reproducibility could be combined with complementary efforts to increase the number of *replications*, i.e., research to investigate the original question with new data and methods (code). Therefore, as suggested in Chapter 12, the **reproduction and replication of fundamental works in geography, geoscience, and GIScience** could be a game-changing activity. For example, the encouragement of replications and reproductions as topics of undergraduate theses would be beneficial both for the disciplines, which could increase the trust in their foundations, and for younger generations of researchers, who work on relevant questions and learn from excellent pieces of research.

Finally, the work of this dissertation comes to an end in a time when reproducible research is en vogue. Similar to Sören Auer's concerns about research data management[21], where he warns about the *"lowlands of a burned RDM vision"* in connection with the technology

[19]Cf. presentation at Werkstattgespräch "Atlas-Zukünfte" (Forschungsbereich Geovisualisierung des Leipniz-Institut für Länderkunde, IfL), Leipzig, Deutschland, in 2016; http://www.slideshare.net/nuest/atlas-zuknfte.

[20]E.g., EaaSI, https://www.softwarepreservationnetwork.org/emulation-as-a-service-infrastructure/.

[21]Presentation at https://docs.google.com/presentation/d/1-RFf-JK5gpudvQvqzF6LdCV5FfGttXU6v0qC2-n3xHw/edit# summarised in https://twitter.com/SoerenAuer/status/1323899294827556864.

adoption lifecycle, one must be careful to identify sustainable motivations and incentives for the broad majority of researchers. It is not sustainable to convince only the altruistic, innovative, or simply well-positioned early adopters of open practices. Only in a culture change that reaches everybody can the burdens as well as the benefits be shared equally between individual researchers and science in general. This change must include long-term funding for maintaining core data, software, and infrastructure (Anzt et al., 2021; Ficarra et al., 2020; Mons, 2020). Luckily, research communities themselves, with a special burden on seniors and leaders, have the power to set their laws and norms and to shape their markets and infrastructures. Hopefully, this work contributes to these forces to be used for more open and much more reproducible geospatial sciences. Reproducible research is just a small building block in a much larger transformation in science that needs to tackle challenges around, for example, equity/diversity/inclusion, openness, research assessment & evaluation, metrics & incentives, predatory publishing, misinformation, involving the Global South, healthy work environments, career opportunities for software experts, publication pressure & bias, and valuing reuse over piecewise publishing (Mejlgaard et al., 2020; Schimanski & Alperin, 2018; West & Bergstrom, 2021). Nevertheless, reproducibility is one topic that individuals and communities can relate to can use to connect with each other to advance science; therefore it is a conquerable challenge for the foreseeable future.

18 References

Anzt, H., Bach, F., Druskat, S., Löffler, F., Loewe, A., Renard, B. Y., Seemann, G., Struck, A., Achhammer, E., Aggarwal, P., Appel, F., Bader, M., Brusch, L., Busse, C., Chourdakis, G., Dabrowski, P. W., Ebert, P., Flemisch, B., Friedl, S., Fritzsch, B., Funk, M. D., Gast, V., Goth, F., Grad, J.-N., Hegewald, J., Hermann, S., Hohmann, F., Janosch, S., Kutra, D., Linxweiler, J., Muth, T., Peters-Kottig, W., Rack, F., Raters, F. H. C., Rave, S., Reina, G., Reißig, M., Ropinski, T., Schaarschmidt, J., Seibold, H., Thiele, J. P., Uekermann, B., Unger, S., & Weeber, R. (2021). An environment for sustainable research software in Germany and beyond: Current state, open challenges, and call for action. *F1000Research*, *9*, 295. https://doi.org/10.12688/f1000research.23224.2

Appel, M., Nüst, D., & Pebesma, E. (2017). *Reproducible Earth observation analytics: Challenges, ideas, and a studycase on containerized land use change detection.* Geophysical Research Abstracts, Vol. 19. https://meetingorganizer.copernicus.org/EGU2017/EGU2017-8525.pdf

Brachman, M. L. (2020). Don't forget about geography. *Journal of Spatial Information Science*, *21*, 263–266. https://doi.org/10.5311/JOSIS.2020.21.727

Breznau, N., Rinke, E. M., Wuttke, A., Adem, M., Adriaans, J., Alvarez-Benjumea, A., Andersen, H. K., Auer, D., Azevedo, F., Bahnsen, O., Balzer, D., Bauer, G., Bauer, P., Baumann, M., Baute, S., Benoit, V., Berning, C., Bernauer, J., Berthold, A., ... Nguyen, H. H. V. (2019). *The crowdsourced replication initiative: Investigating immigration and social policy preferences. Executive report.* https://doi.org/10.31235/osf.io/6j9qb

Breznau, N., Rinke, E. M., Wuttke, A., Adem, M., Adriaans, J., Alvarez-Benjumea, A., Andersen, H. K., Auer, D., Azevedo, F., Bahnsen, O., Balzer, D., Bauer, G., Bauer, P. C., Baumann, M., Baute, S., Benoit, V., Bernauer, J., Berning, C., Berthold, A., ... Nguyen, H. H. V. (2021). *Observing many researchers using the same data and hypothesis reveals a hidden universe of uncertainty.* https://doi.org/10.31222/osf.io/cd5j9

Claerbout, J., & Karrenbach, M. (1992). *Electronic documents give reproducible research a new meaning*. 601–604. https://doi.org/10.1190/1.1822162

Eglen, S., & Nüst, D. (2019). CODECHECK: An open-science initiative to facilitate sharing of computer programs and results presented in scientific publications. *Septentrio Conference Series*, *1*. https://doi.org/10.7557/5.4910

Ficarra, V., Fosci, M., Chiarelli, A., Kramer, B., & Proudman, V. (2020). *Scoping the Open Science Infrastructure Landscape in Europe.* Zenodo. https://doi.org/10.5281/zenodo.4159838

Gentleman, R., & Temple Lang, D. (2007). Statistical analyses and reproducible research. *Journal of Computational and Graphical Statistics*, *16*(1), 1–23. https://doi.org/10.1198/106186007X178663

Goodchild, M. F., Fotheringham, A. S., Kedron, P., & Li, W. (2020). Introduction: Forum on Reproducibility and Replicability in Geography. *Annals of the American Association of Geographers*, 1–4. https://doi.org/10.1080/24694452.2020.1806030

Granell, C., Nüst, D., Ostermann, F. O., & Sileryte, R. (2018). *Reproducible Research is like riding a bike.* https://doi.org/10.7287/peerj.preprints.27216v1

Granell, C., Sileryte, R., & Nüst, D. (2020). *Reproducible graduate theses in GIScience.* https://doi.org/10.17605/osf.io/wcexy

Graupner, A., & Nüst, D. (2020). *Serverless GEO labels for the semantic sensor web* (K. Janowicz & J. A. Verstegen, Eds.; Vol. 177, p. 4:14:14). Schloss Dagstuhl–Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.GIScience.2021.I.4

Hauschke, C., Nüst, D., Cordts, A., & Lilienthal, S. (2021). OPTIMETA – Strengthening the Open Access publishing system through open citations and spatiotemporal metadata. *Research Ideas and Outcomes*, *7*, e66264. https://doi.org/10.3897/rio.7.e66264

Hrynaszkiewicz, I. (2020). Publishers' Responsibilities in Promoting Data Quality and Reproducibility. In A. Bespalov, M. C. Michel, & T. Steckler (Eds.), *Good Research Practice in Non-Clinical Pharmacology and Biomedicine* (pp. 319–348). Springer International Publishing. https://doi.org/10.1007/164_2019_290

Knoth, C., & Nüst, D. (2017). Reproducibility and Practical Adoption of GEOBIA with Open-Source Software in Docker Containers. *Remote Sensing*, *9*(3), 290. https://doi.org/10.3390/rs9030290

Konkol, M. (2019). *Publishing Reproducible Geoscientific Papers: Status quo, benefits, and opportunities*. https://doi.org/10.31237/osf.io/mcdrn

Konkol, M., & Kray, C. (2019). In-depth examination of spatiotemporal figures in open reproducible research. *Cartography and Geographic Information Science*, *46*(5), 412–427. https://doi.org/10.1080/15230406.2018.1512421

Konkol, M., Kray, C., & Pfeiffer, M. (2019). Computational reproducibility in geoscientific papers: Insights from a series of studies with geoscientists and a reproduction study. *International Journal of Geographical Information Science*, *33*(2), 408–429. https://doi.org/10.1080/13658816.2018.1508687

Konkol, M., Kray, C., & Suleiman, J. (2019). Creating Interactive Scientific Publications using Bindings. *Proceedings of the ACM on Human-Computer Interaction*, *3*(EICS), 16:116:18. https://doi.org/10.1145/3331158

Konkol, M., Nüst, D., & Goulier, L. (2020). Publishing computational research - a review of infrastructures for reproducible and transparent scholarly communication. *Research Integrity and Peer Review*, *5*(1), 10. https://doi.org/10.1186/s41073-020-00095-y

Kray, C., Pebesma, E., Konkol, M., & Nüst, D. (2019). *Reproducible Research in Geoinformatics: Concepts, Challenges and Benefits (Vision Paper)* (S. Timpf, C. Schlieder, M. Kattenbeck, B. Ludwig, & K. Stewart, Eds.; Vol. 142, p. 8:18:13). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. https://doi.org/10.4230/LIPIcs.COSIT.2019.8

Lohoff, L. (2018). Similarity Measurements for Executable Research Compendia. *MSc Geoinformatics Thesis, Ifgi, WWU Münster.*

Mejlgaard, N., Bouter, L. M., Gaskell, G., Kavouras, P., Allum, N., Bendtsen, A.-K., Charitidis, C. A., Claesen, N., Dierickx, K., Domaradzka, A., Elizondo, A. R., Foeger, N., Hiney, M., Kaltenbrunner, W., Labib, K., Marušić, A., Sørensen, M. P., Ravn, T., Ščepanović, R., Tijdink, J. K., & Veltri, G. A. (2020). Research integrity: Nine ways to move from talk to walk. *Nature*, *586*(7829), 358–360. https://doi.org/10.1038/d41586-020-02847-8

Minghini, M., Mobasheri, A., Rautenbach, V., & Brovelli, M. A. (2020). Geospatial openness: From software to standards & data. *Open Geospatial Data, Software and Standards*, *5*(1), 1. https://doi.org/10.1186/s40965-020-0074-y

Mons, B. (2020). Invest 5% of research funds in ensuring data are reusable. *Nature*, *578*(7796), 491–491. https://doi.org/10.1038/d41586-020-00505-7

Nichols, J. D., Oli, M. K., Kendall, W. L., & Boomer, G. S. (2021). Opinion: A better approach for dealing with reproducibility and replicability in science. *Proceedings of the National Academy of Sciences*, *118*(7). https://doi.org/10.1073/pnas.2100769118

Niers, T. (2020). Geospatial Metadata for Discovery in Scholarly Publishing. *MSc Geoinformatics Thesis, Ifgi, WWU Münster.* http://nbn-resolving.de/urn:nbn:de:hbz:6-69029469735

Niers, T., & Nüst, D. (2020). Geospatial Metadata for Discovery in Scholarly Publishing. *Septentrio Conference Series*, *4*. https://doi.org/10.7557/5.5590

Nosek, B. (2019). *Strategy for Culture Change*. COS blog. https://www.cos.io/blog/strategy-for-culture-change

Nosek, B. A., Hardwicke, T. E., Moshontz, H., Allard, A., Corker, K. S., Almenberg, A. D., Fidler, F., Hilgard, J., Struhl, M. K., Nuijten, M. B., Rohrer, J. M., Romero, F., Scheel, A. M., Scherer, L., Schönbrodt, F., & Vazire, S. (2021). *Replicability, Robustness, and Reproducibility in Psychological Science*. PsyArXiv. https://doi.org/10.31234/osf.io/ksfvq

Nüst, D. (2019a). *How to increase reproducibility and transparency in your research*. EGU GeoLog; archived as part of https://doi.org/10.5281/zenodo.1485437. https://blogs.egu.eu/geolog/2019/02/01/reproducibility-and-transparency-in-research/

Nüst, D. (2021a). *Practical reproducibility and reproducibility vs. Peer review*. Presented at Spatial Data Science Hangout (online) by Center for Spatial Studies, University of California, Santa Barbara, CA, USA. http://bit.ly/hangout21-repro

Nüst, D. (2021b). *Executing workflows during peer review for transparency, reproducibility, and reusability*. Presented at Thuringian RDM-Days 2021 (online). https://doi.org/10.5281/zenodo.5018144

Nüst, D. (2018). *Integrating Binder and Stencila – the building blocks to increased open communication and transparency*. Cross-posted on eLife blog, Stencila blog and Jupyter blog; archived as part of https://doi.org/10.5281/zenodo.1485437.

Nüst, D. (2017a). *Executable research compendia in geoscience research infrastructures*. Geophysical Research Abstracts, Vol. 19. https://meetingorganizer.copernicus.org/EGU2017/EGU2017-8525.pdf

Nüst, D. (2017b). *Reproducible Research in Geostatistics and Geosciences with Notebooks and Containers*. Course at GEOSTAT Split 2017 Summer School, Split, Croatia. https://doi.org/10.5281/zenodo.2477154

Nüst, D. (2019b). *Packaging Research for Open Scholarship*. Presented at Kolloquium des Lehrstuhls für Geoinformatik, Institut für Geographie, Friedrich Schiller Universität Jena. https://gitlab.com/nuest/packaging-research-for-open-scholarship

Nüst, D. (2020a). *Container images for research librarians 101*. Presented at Librarians Building Momentum for Reproducibility on 2020-01-28, Online, organised by NYI, New York. https://doi.org/10.17605/osf.io/xta6d

Nüst, D. (2020b). *Research compendia enable code review during peer review*. Presented at Remote ReproHack (N8 CIR); slides. https://doi.org/10.5281/zenodo.3855440

Nüst, D. (2021c). *A web service for executable research compendia enables reproducible publications and transparent reviews in geospatial sciences*. https://zivgitlab.uni-muenster.de/d_nues01/architecture-paper/

Nüst, D. (2021d). *AGILE Conference Reproducibility Review 2021 (talk).* https://doi.org/10.5281/ZENODO.4926269

Nüst, D. (2021e). *Reproducibility and Peer Review.* Presented at Virtual kick off meeting of the Open Reproducible Data Science and Statistics (ORDS, Graduate Academy of the University of Rostock) network on 2020-12-01, University of Rostock. https://doi.org/10.5281/zenodo.4292263

Nüst, D., & Bartoschek, T. (2018). *Open Environmental Data Analysis.* Geophysical Research Abstracts, Vol. 20; archived on Zenodo; ERC: https://o2r.uni-muenster.de/erc/PhbIa. https://doi.org/10.5281/zenodo.1217912

Nüst, D., Boettiger, C., & Eddelbuettel, D. (2018). *rocker/geospatial: A flexible runtime environment for geoscientific data analysis.* Geophysical Research Abstracts, Vol. 20; archived on Zenodo. https://doi.org/10.5281/zenodo.1216751

Nüst, D., Boettiger, C., & Marwick, B. (2018). How to Read a Research Compendium. *arXiv:1806.09525 [Cs].* http://arxiv.org/abs/1806.09525

Nüst, D., Drost, N., Topping, D., & Wyborn, L. (2021). Improving Research Software in the Geosciences. In *EGU General Assembly 2021 Great Debate.* Co-sponsored by EGU and AGU. Materials at https://great-debate-research-software.github.io/. https://meetingorganizer.copernicus.org/EGU21/session/39993

Nüst, D., Eddelbuettel, D., Bennett, D., Cannoodt, R., Clark, D., Daróczi, G., Edmondson, M., Fay, C., Hughes, E., Kjeldgaard, L., Lopp, S., Marwick, B., Nolis, H., Nolis, J., Ooi, H., Ram, K., Ross, N., Shepherd, L., Sólymos, P., Swetnam, T. L., Turaga, N., Petegem, C. V., Williams, J., Willis, C., & Xiao, N. (2020). The Rockerverse: Packages and Applications for Containerisation with R. *The R Journal*, *12*(1). https://doi.org/10.32614/RJ-2020-007

Nüst, D., & Eglen, S. J. (2021). CODECHECK: An Open Science initiative for the independent execution of computations underlying research articles during peer review to improve reproducibility. *F1000Research*, *10*, 253. https://doi.org/10.12688/f1000research.51738.1

Nüst, D., Granell, C., Hofer, B., Konkol, M., Ostermann, F. O., Sileryte, R., & Cerutti, V. (2018). Reproducible research and GIScience: An evaluation using AGILE conference papers. *PeerJ*, *6*, e5072. https://doi.org/10.7717/peerj.5072

Nüst, D., & Hinz, M. (2017). *Automatically archiving reproducible studies with Docker.* Presented at useR!2017, Brussels, Belgium. https://doi.org/10.5281/zenodo.824007

Nüst, D., & Hinz, M. (2019). Containerit: Generating Dockerfiles for reproducible research with R. *Journal of Open Source Software*, *4*(40), 1603. https://doi.org/10.21105/joss.01603

Nüst, D., & Konkol, M. (2020). *A Vision for Reproducible Research in Geoinformatics, Geography, and Geosciences.* GenR blog. https://genr.eu/wp/a-vision-for-reproducible-research-in-geoinformatics-geography-and-geosciences/

Nüst, D., Konkol, M., Pebesma, E., Kray, C., Schutzeichel, M., Przibytzin, H., & Lorenz, J. (2017). Opening the Publication Process with Executable Research Compendia. *D-Lib Magazine*, *23*(1/2). https://doi.org/10.1045/january2017-nuest

Nüst, D., Kray, C., Pebesma, E., Lorenz, J., Konkol, M., Niers, T., Przibytzin, H., Schutzeichel, M., Chaudhary, R., Garzon, J. S., Hinz, M., Jakuschona, N., Koppe, J., Kraft, T., Kühnel, T., Lohoff, L., Suleiman, J., & Qamaz, Y. (2018). *Reproducibility Service for Executable Research Compendia: Technical Specifications and Reference Implementation.* Zenodo. https://doi.org/10.5281/zenodo.2203844

Nüst, D., Lohoff, L., Einfeldt, L., Gavish, N., Götza, M., Jaswal, S. T., Khalid, S., Meierkort, L., Mohr, M., Rendel, C., & Eek, A. van. (2019, June). *Guerrilla Badges for Reproducible Geospatial Data Science (AGILE 2019 Short Paper).* https://doi.org/10.31223/osf.io/xtsqh

Nüst, D., Ostermann, F., Granell, C., & Kmoch, A. (2020). Improving reproducibility of geospatial conference papers – lessons learned from a first implementation of reproducibility reviews. *Septentrio Conference Series*, *4*. https://doi.org/10.7557/5.5601

Nüst, D., Ostermann, F., Sileryte, R., Hofer, B., Granell, C., Teperek, M., Graser, A., Broman, K., Hettne, K., Clare, C., Belliard, F., & Wang, Y. (2021). *AGILE Reproducible Paper Guidelines (December 2020).* https://doi.org/10.17605/osf.io/cb7z8

Nüst, D., & Pebesma, E. (2020). Practical reproducibility in geography and geosciences. *Annals of the American Association of Geographers*, *0*(0), 1–11. https://doi.org/10.1080/24694452.2020.1806028

Nüst, D., & Schutzeichel, M. (2017, June 12). *20th AGILE Conference for Geo-information Science.* https://doi.org/10.5281/zenodo.1478542

Nüst, D., Schutzeichel, M., & Konkol, M. (2018). *Opening Reproducible Research: A research project website and blog.* https://doi.org/10.5281/zenodo.4384840

Nüst, D., Seibold, H., Eglen, S. J., & Schulz-Vanheyden, L. (2021). *Code Execution in Peer Review.* https://doi.org/10.17605/osf.io/x32nc

Nüst, D., Sochat, V., Marwick, B., Eglen, S., Head, T., & Hirst, T. (2020). *Ten Simple Rules for Writing Dockerfiles for Reproducible Data Science.* https://doi.org/10.31219/osf.io/fsd7t

Ostermann, F. O., Nüst, D., Granell, C., Hofer, B., & Konkol, M. (2020). *Reproducible Research and GIScience: An evaluation using GIScience conference papers.* https://doi.org/10.31223/X5ZK5V

Pebesma, E. J., Kray, C., & Tröger, B. (2020). Opening Reproducible Research II: Infrastructure for Electronic Publications and Digital Scholarly Communication. *Project Proposal.* https://doi.org/10.17879/42149626934

Peer, L., Orr, L. V., & Coppock, A. (2021). Active Maintenance: A Proposal for the Long-Term Computational Reproducibility of Scientific Results. *PS: Political Science & Politics*, 1–5. https://doi.org/10.1017/S1049096521000366

Project Jupyter, Bussonnier, M., Forde, J., Freeman, J., Granger, B., Head, T., Holdgraf, C., Kelley, K., Nalvarte, G., Osheroff, A., Pacer, M., Panda, Y., Perez, F., Ragan-Kelley, B., & Willing, C. (2018). Binder 2.0 - Reproducible, interactive, sharable environments for science at scale. *Proceedings of the 17th Python in Science Conference*, 113–120. https://doi.org/10.25080/Majora-4af1f417-011

Riet, G. ter, Storosum, B. W. C., & Zwinderman, A. H. (2019). What is reproducibility? *F1000Research*, *8*, 36. https://doi.org/10.12688/f1000research.17615.1

Rowe, F., Maier, G., Arribas-Bel, D., & Rey, S. (2020). The Potential of Notebooks for Scientific Publication, Reproducibility and Dissemination. *REGION*, *7*(3), E1–E5. https://doi.org/10.18335/region.v7i3.357

Schimanski, L. A., & Alperin, J. P. (2018). The evaluation of scholarship in academic promotion and tenure processes: Past, present, and future. *F1000Research*, *7*, 1605. https://doi.org/10.12688/f1000research.16493.1

Sochat, V., & Nüst, D. (2021). *RSE Stories podcast episode "Are You Interested in Riddles?"*. Download MP3. https://us-rse.org/rse-stories/2021/daniel-nuest/

Steeves, V., Rampin, R., & Nüst, D. (2017). *Reproducible computational research in the publication cycle*. OSF; short course at EGU General Assembly 2017; see report. https://doi.org/10.17605/osf.io/umy6g

Sunni, I. (2020). Testing geospatial R packages on implementations of the R language and platforms. *MSc Geospatial Technologies*. http://hdl.handle.net/10362/95140

University of Southern California, Boeing, G., Arribas-Bel, D., & University of Liverpool. (2021). GIS and Computational Notebooks. *Geographic Information Science & Technology Body of Knowledge*, *2021*(Q1). https://doi.org/10.22224/gistbok/2021.1.2

West, J. D., & Bergstrom, C. T. (2021). Misinformation in and about science. *Proceedings of the National Academy of Sciences*, *118*(15). https://doi.org/10.1073/pnas.1912444117

Wikipedia contributors. (2020). *OS-level virtualization*. Wikipedia. https://en.wikipedia.org/w/index.php?title=OS-level_virtualization&oldid=935110975

Wikipedia contributors. (2021). *Literate programming*. Wikipedia. https://en.wikipedia.org/w/index.php?title=Literate_programming&oldid=1011343163

Daniel **Nüst**

RESEARCHER

email@danielnuest.de | danielnuest.de | 0000-0002-0024-5046 | nuest | nuest | danielnuest | nordholmen

Daniel is a research software engineer and PhD student at the Institute for Geoinformatics, University of Münster, Germany. He develops tools for reproducible geoscientific research in the project Opening Reproducible Research (https://o2r.info). Daniel is an advocate for open scholarship and reproducibility.

Date and place of birth: January 2nd, 1984, Gütersloh | **Nationality**: German

Education

Westfälische Wilhelms-Universität Münster *Münster, DE*
PhD *01/2016 (current)*

Westfälische Wilhelms-Universität Münster Institut für Geoinformatik *Münster, DE*
Diplom-Geoinformatiker *10/2004 - 11/2011*

Gymnasium Verl *Verl, DE*
Abitur *08/1995 - 06/2003*

Employment

University of Münster, Institute for Geoinformatics *Münster, DE*
researcher in project Opening Reproducible Research *01/2016 - 04/2023*

52°North Initiative for Geospatial Open Source Software GmbH *Münster, DE*
consultant and developer, i.a. in projects GeoViQua (FP7), GLUES (BMBF), ConnectinGEO (H2020) *10/2011 - 12/2015*

52°North Initiative for Geospatial Open Source Software GmbH *Münster, DE*
student assistant in project GENESIS (FP7) *02/2011 - 09/2011*

University of Münster, Institute for Geoinformatics *Münster, DE*
student assistant in project OSIRIS (FP6), lab tutor *11/2009 - 01/2011*

Memberships and service

NFDI4Culture Expert:innen-Forum "Nachhaltige Softwareentwicklung" *NFDI4Culture, DE*
expert NFDI4Culture expert forum on sustainable software development *2021 - 2022*

Knowledge Exchange *Bristol, GB*
expert Task and Finish Group "Publishing Reproducible Research Output" *2020 - 2021*

de-RSE e.V. - Society for Research Software *Berlin, DE*
Vice Chair *2018 - 2021*

Association of Geographic Information Laboratories in Europe *Europe, NL*
Reproducibility Committee Chair AGILE Conference *2020 - 2021*

Forum Citizen Science *Münster, DE*
Scientific Programme Committee Co-chair *2019 - 2019*

13 reviews conducted for the journals GigaScience, ISPRS International Journal of Geo-Information, The Journal of Open Source Software, Journal of Geographical Systems, The R Journal, Semantic Web, Sustainability, Data Science Journal, and International Journal of Digital Earth.

Funding

Mozilla Foundation (grant number: MF-1811-05959) *CA, US*
CODECHECK *2019 - 2020*

Association of Geographic Information Laboratories in Europe (AGILE) *Europe, NL*
Reproducible Publications at AGILE Conferences *2019 - 2019*

Bundesministerium für Bildung und Forschung (grant number: 16TOA028B) *Berlin, DE*
Strengthening the Open Access publishing system through open citations and spatiotemporal metadata (OPTIMETA) *2021 - 2023*

Works

A full list of publications can be found at https://orcid.org/0000-0002-0024-5046.

Juli, 2021 Daniel Nüst · CV/Resume based on orcid.org/0000-0002-0024-5046 1

Colophon

This document is compiled with R Markdown. All source files are published at https://zivgitlab.uni-muenster.de/d_nues01/phd-package. The current revision is f287175.

Environment:

```
## wkhtmltopdf 0.12.5
## ----------
## requiremments.txt:
## panflute==2.0.5
## python-frontmatter==1.0.0


## Diagnostics Report [renv 0.13.2]
## ================================
##
## # Session Info =======================
## R version 4.1.0 (2021-05-18)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8          LC_NUMERIC=C
##  [3] LC_TIME=de_DE.UTF-8           LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=de_DE.UTF-8       LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=de_DE.UTF-8          LC_NAME=de_DE.UTF-8
##  [9] LC_ADDRESS=de_DE.UTF-8        LC_TELEPHONE=de_DE.UTF-8
## [11] LC_MEASUREMENT=de_DE.UTF-8    LC_IDENTIFICATION=de_DE.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices datasets  utils     methods   base
##
## other attached packages:
##  [1] rorcid_0.7.0       glue_1.4.2         vitae_0.4.2.9001   jsonlite_1.7.2
##  [5] ggfittext_0.9.1    ggalluvial_0.12.3  ggthemes_4.2.4     forcats_0.5.1
##  [9] purrr_0.3.4        readr_1.4.0        tidyr_1.1.3        tibble_3.1.2
## [13] ggplot2_3.3.3      tidyverse_1.3.1    wordcloud_2.6      RColorBrewer_1.1-2
## [17] quanteda_3.0.0     dplyr_1.0.6        tidytext_0.3.1     bib2df_1.1.2
## [21] stringr_1.4.0      pdftools_3.0.1     staplr_3.1.1       here_1.0.1
##
## loaded via a namespace (and not attached):
##  [1] fs_1.5.0           lubridate_1.7.10   httr_1.4.2         rprojroot_2.0.2
##  [5] SnowballC_0.7.0    tools_4.1.0        backports_1.2.1    utf8_1.2.1
##  [9] R6_2.5.0           DBI_1.1.1          colorspace_2.0-1   withr_2.4.2
## [13] tidyselect_1.1.1   curl_4.3.1         compiler_4.1.0     cli_2.5.0
## [17] rvest_1.0.0        xml2_1.3.2         humaniformat_0.6.0 triebeard_0.3.0
## [21] labeling_0.4.2     bookdown_0.22      scales_1.1.1       askpass_1.1
## [25] digest_0.6.27      rmarkdown_2.8      pkgconfig_2.0.3    htmltools_0.5.1.1
## [29] dbplyr_2.1.1       rlang_0.4.11       readxl_1.3.1       httpcode_0.3.0
## [33] rstudioapi_0.13    generics_0.1.0     farver_2.1.0       tokenizers_0.2.1
## [37] magrittr_2.0.1     fauxpas_0.5.0      Matrix_1.3-4       Rcpp_1.0.6
## [41] munsell_0.5.0      fansi_0.5.0        lifecycle_1.0.0    whisker_0.4
## [45] stringi_1.6.2      yaml_2.2.1         grid_4.1.0         crayon_1.4.1
## [49] lattice_0.20-44    haven_2.4.1        hms_1.1.0          knitr_1.33
## [53] pillar_1.6.1       tcltk_4.1.0        codetools_0.2-18   stopwords_2.2
## [57] fastmatch_1.1-0    crul_1.1.0         reprex_2.0.0       evaluate_0.14
## [61] qpdf_1.1           renv_0.13.2        RcppParallel_5.1.4 modelr_0.1.8
## [65] urltools_1.7.3     vctrs_0.3.8        cellranger_1.1.0   gtable_0.3.0
## [69] assertthat_0.2.1   xfun_0.23          broom_0.7.6        janeaustenr_0.1.5
## [73] rJava_1.0-4        tinytex_0.32       ellipsis_0.3.2
##
## # Project ============================
## Project path: "~/git/phd-package"
##
## # Status =============================
## * The project is already synchronized with the lockfile.
##
## # Packages ===========================
##                   Library Source   Lockfile Source Path Dependency
## BH               1.75.0-0   CRAN   1.75.0-0   CRAN  [1]   indirect
## DBI                 1.1.1   CRAN      1.1.1   CRAN  [1]   indirect
## ISOcodes       2021.02.24   CRAN 2021.02.24   CRAN  [1]   indirect
## KernSmooth        2.23-20   CRAN       <NA>   <NA>  [2]       <NA>
## MASS               7.3-54   CRAN     7.3-54   CRAN  [2]   indirect
## Matrix              1.3-4   CRAN      1.3-4   CRAN  [2]   indirect
## R6                  2.5.0   CRAN      2.5.0   CRAN  [1]   indirect
## RColorBrewer        1.1-2   CRAN      1.1-2   CRAN  [1]     direct
## Rcpp                1.0.6   CRAN      1.0.6   CRAN  [1]   indirect
## RcppArmadillo  0.10.5.0.0   CRAN 0.10.5.0.0   CRAN  [1]   indirect
## RcppParallel        5.1.4   CRAN      5.1.4   CRAN  [1]   indirect
## SnowballC           0.7.0   CRAN      0.7.0   CRAN  [1]   indirect
## XML              3.99-0.6   CRAN   3.99-0.6   CRAN  [1]   indirect
## askpass               1.1   CRAN        1.1   CRAN  [1]   indirect
## assertthat          0.2.1   CRAN      0.2.1   CRAN  [1]   indirect
## backports           1.2.1   CRAN      1.2.1   CRAN  [1]   indirect
## base64enc           0.1-3   CRAN      0.1-3   CRAN  [1]   indirect
## bib2df              1.1.2 GitHub      1.1.2 GitHub  [1]     direct
## blob                1.2.1   CRAN      1.2.1   CRAN  [1]   indirect
## bookdown             0.22   CRAN       0.22   CRAN  [1]     direct
## boot               1.3-28   CRAN       <NA>   <NA>  [2]       <NA>
## brew                1.0-6   CRAN       <NA>   <NA>  [1]       <NA>
## brio                1.1.2   CRAN       <NA>   <NA>  [1]       <NA>
## broom               0.7.6   CRAN      0.7.6   CRAN  [1]   indirect
## bslib             0.2.5.1  CRAN     <NA>  <NA>  [1]      <NA>
## cachem              1.0.5  CRAN     <NA>  <NA>  [1]      <NA>
## callr               3.7.0  CRAN    3.7.0  CRAN  [1]  indirect
## cellranger          1.1.0  CRAN    1.1.0  CRAN  [1]  indirect
## class              7.3-19  CRAN     <NA>  <NA>  [2]      <NA>
## classInt            0.4-3  CRAN     <NA>  <NA>  [1]      <NA>
## cli                 2.5.0  CRAN    2.5.0  CRAN  [1]  indirect
## clipr               0.7.1  CRAN    0.7.1  CRAN  [1]  indirect
## cluster             2.1.2  CRAN     <NA>  <NA>  [2]      <NA>
## codetools          0.2-18  CRAN     <NA>  <NA>  [2]      <NA>
## colorspace          2.0-1  CRAN    2.0-1  CRAN  [1]  indirect
## commonmark            1.7  CRAN     <NA>  <NA>  [1]      <NA>
## cpp11               0.2.7  CRAN    0.2.7  CRAN  [1]  indirect
## crayon              1.4.1  CRAN    1.4.1  CRAN  [1]  indirect
## credentials         1.3.0  CRAN     <NA>  <NA>  [1]      <NA>
## crosstalk           1.1.1  CRAN     <NA>  <NA>  [1]      <NA>
## crul                1.1.0  CRAN    1.1.0  CRAN  [1]  indirect
## curl                4.3.1  CRAN    4.3.1  CRAN  [1]  indirect
## data.table         1.14.0  CRAN   1.14.0  CRAN  [1]  indirect
## dbplyr              2.1.1  CRAN    2.1.1  CRAN  [1]  indirect
## desc                1.3.0  CRAN     <NA>  <NA>  [1]      <NA>
## diffobj             0.3.4  CRAN     <NA>  <NA>  [1]      <NA>
## digest             0.6.27  CRAN   0.6.27  CRAN  [1]  indirect
## dplyr               1.0.6  CRAN    1.0.6  CRAN  [1]    direct
## dtplyr              1.1.0  CRAN    1.1.0  CRAN  [1]  indirect
## e1071               1.7-7  CRAN     <NA>  <NA>  [1]      <NA>
## ellipsis            0.3.2  CRAN    0.3.2  CRAN  [1]  indirect
## evaluate             0.14  CRAN     0.14  CRAN  [1]  indirect
## fansi               0.5.0  CRAN    0.5.0  CRAN  [1]  indirect
## farver              2.1.0  CRAN    2.1.0  CRAN  [1]  indirect
## fastmap             1.1.0  CRAN     <NA>  <NA>  [1]      <NA>
## fastmatch           1.1-0  CRAN    1.1-0  CRAN  [1]  indirect
## fauxpas             0.5.0  CRAN    0.5.0  CRAN  [1]  indirect
## forcats             0.5.1  CRAN    0.5.1  CRAN  [1]    direct
## foreign            0.8-81  CRAN     <NA>  <NA>  [2]      <NA>
## fs                  1.5.0  CRAN    1.5.0  CRAN  [1]  indirect
## gargle              1.1.0  CRAN    1.1.0  CRAN  [1]  indirect
## generics            0.1.0  CRAN    0.1.0  CRAN  [1]  indirect
## gert                1.3.0  CRAN     <NA>  <NA>  [1]      <NA>
## ggalluvial         0.12.3  CRAN   0.12.3  CRAN  [1]    direct
## ggfittext           0.9.1  CRAN    0.9.1  CRAN  [1]    direct
## ggplot2             3.3.3  CRAN    3.3.3  CRAN  [1]    direct
## ggthemes            4.2.4  CRAN    4.2.4  CRAN  [1]    direct
## gh                  1.3.0  CRAN     <NA>  <NA>  [1]      <NA>
## gitcreds            0.1.1  CRAN     <NA>  <NA>  [1]      <NA>
## glue                1.4.2  CRAN    1.4.2  CRAN  [1]    direct
## googledrive         1.0.1  CRAN    1.0.1  CRAN  [1]  indirect
## googlesheets4       0.3.0  CRAN    0.3.0  CRAN  [1]  indirect
## grDevices            <NA>  <NA>     <NA>  <NA>  [2]  indirect
## graphics             <NA>  <NA>     <NA>  <NA>  [2]  indirect
## grid                 <NA>  <NA>     <NA>  <NA>  [2]  indirect
## gtable              0.3.0  CRAN    0.3.0  CRAN  [1]  indirect
## haven               2.4.1  CRAN    2.4.1  CRAN  [1]  indirect
## here                1.0.1  CRAN    1.0.1  CRAN  [1]    direct
## highr                 0.9  CRAN      0.9  CRAN  [1]  indirect
## hms                 1.1.0  CRAN    1.1.0  CRAN  [1]  indirect
## htmltools         0.5.1.1  CRAN  0.5.1.1  CRAN  [1]  indirect
## htmlwidgets         1.5.3  CRAN     <NA>  <NA>  [1]      <NA>
## httpcode            0.3.0  CRAN    0.3.0  CRAN  [1]  indirect
## httpuv              1.6.1  CRAN     <NA>  <NA>  [1]      <NA>
## httr                1.4.2  CRAN    1.4.2  CRAN  [1]  indirect
## humaniformat        0.6.0  CRAN    0.6.0  CRAN  [1]  indirect
## hunspell            3.0.1  CRAN    3.0.1  CRAN  [1]  indirect
## ids                 1.0.1  CRAN    1.0.1  CRAN  [1]  indirect
## ini                 0.3.1  CRAN     <NA>  <NA>  [1]      <NA>
## isoband             0.2.4  CRAN    0.2.4  CRAN  [1]  indirect
## janeaustenr         0.1.5  CRAN    0.1.5  CRAN  [1]  indirect
## jquerylib           0.1.4  CRAN     <NA>  <NA>  [1]      <NA>
## jsonlite            1.7.2  CRAN    1.7.2  CRAN  [1]    direct
## knitr                1.33  CRAN     1.33  CRAN  [1]    direct
## koRpus             0.13-8  CRAN     <NA>  <NA>  [1]      <NA>
## koRpus.lang.en      0.1-4  CRAN     <NA>  <NA>  [1]      <NA>
## labeling            0.4.2  CRAN    0.4.2  CRAN  [1]  indirect
## later               1.2.0  CRAN     <NA>  <NA>  [1]      <NA>
## lattice           0.20-44  CRAN  0.20-44  CRAN  [2]  indirect
## lazyeval            0.2.2  CRAN    0.2.2  CRAN  [1]  indirect
## lifecycle           1.0.0  CRAN    1.0.0  CRAN  [1]  indirect
## lubridate          1.7.10  CRAN   1.7.10  CRAN  [1]  indirect
## magrittr            2.0.1  CRAN    2.0.1  CRAN  [1]  indirect
## markdown              1.1  CRAN      1.1  CRAN  [1]  indirect
## memoise             2.0.0  CRAN     <NA>  <NA>  [1]      <NA>
## methods              <NA>  <NA>     <NA>  <NA>  [2]  indirect
## mgcv               1.8-36  CRAN   1.8-36  CRAN  [2]  indirect
## mime                 0.10  CRAN     0.10  CRAN  [1]  indirect
## miniUI            0.1.1.1  CRAN     <NA>  <NA>  [1]      <NA>
## modelr              0.1.8  CRAN    0.1.8  CRAN  [1]  indirect
## munsell             0.5.0  CRAN    0.5.0  CRAN  [1]  indirect
## nlme              3.1-152  CRAN  3.1-152  CRAN  [2]  indirect
## nnet               7.3-16  CRAN     <NA>  <NA>  [2]      <NA>
## openssl             1.4.4  CRAN    1.4.4  CRAN  [1]  indirect
## pdftools            3.0.1  CRAN    3.0.1  CRAN  [1]    direct
## pillar              1.6.1  CRAN    1.6.1  CRAN  [1]  indirect
## pkgbuild            1.2.0  CRAN     <NA>  <NA>  [1]      <NA>
## pkgconfig           2.0.3  CRAN    2.0.3  CRAN  [1]  indirect
## pkgload             1.2.1  CRAN     <NA>  <NA>  [1]      <NA>
## plyr                1.8.6  CRAN     <NA>  <NA>  [1]      <NA>
## praise              1.0.0  CRAN     <NA>  <NA>  [1]      <NA>
## prettyunits         1.1.1  CRAN    1.1.1  CRAN  [1]  indirect
## processx            3.5.2  CRAN    3.5.2  CRAN  [1]  indirect
## progress            1.2.2  CRAN    1.2.2  CRAN  [1]  indirect
## promises          1.2.0.1  CRAN     <NA>  <NA>  [1]      <NA>
## proxy               0.4-26 CRAN       <NA>   <NA> [1]      <NA>
## ps                   1.6.0 CRAN      1.6.0   CRAN [1]  indirect
## purrr                0.3.4 CRAN      0.3.4   CRAN [1]  indirect
## qpdf                   1.1 CRAN        1.1   CRAN [1]  indirect
## quanteda             3.0.0 CRAN      3.0.0   CRAN [1]    direct
## rJava                1.0-4 CRAN      1.0-4   CRAN [1]  indirect
## rappdirs             0.3.3 CRAN      0.3.3   CRAN [1]  indirect
## rcmdcheck            1.3.3 CRAN       <NA>   <NA> [1]      <NA>
## readr                1.4.0 CRAN      1.4.0   CRAN [1]    direct
## readxl               1.3.1 CRAN      1.3.1   CRAN [1]  indirect
## rematch              1.0.1 CRAN      1.0.1   CRAN [1]  indirect
## rematch2             2.1.2 CRAN      2.1.2   CRAN [1]  indirect
## remotes              2.4.0 CRAN       <NA>   <NA> [1]      <NA>
## renv                0.13.2 CRAN     0.13.2   CRAN [1]    direct
## reprex               2.0.0 CRAN      2.0.0   CRAN [1]  indirect
## rlang               0.4.11 CRAN     0.4.11   CRAN [1]  indirect
## rmarkdown              2.8 CRAN        2.8   CRAN [1]    direct
## rorcid               0.7.0 CRAN      0.7.0   CRAN [1]    direct
## roxygen2             7.1.1 CRAN       <NA>   <NA> [1]      <NA>
## rpart               4.1-15 CRAN       <NA>   <NA> [2]      <NA>
## rprojroot            2.0.2 CRAN      2.0.2   CRAN [1]  indirect
## rstudioapi            0.13 CRAN       0.13   CRAN [1]  indirect
## rversions            2.1.1 CRAN       <NA>   <NA> [1]      <NA>
## rvest                1.0.0 CRAN      1.0.0   CRAN [1]  indirect
## s2                   1.0.6 CRAN       <NA>   <NA> [1]      <NA>
## sass                 0.4.0 CRAN       <NA>   <NA> [1]      <NA>
## scales               1.1.1 CRAN      1.1.1   CRAN [1]  indirect
## selectr              0.4-2 CRAN      0.4-2   CRAN [1]  indirect
## sessioninfo          1.1.1 CRAN       <NA>   <NA> [1]      <NA>
## shades               1.4.0 CRAN      1.4.0   CRAN [1]  indirect
## shiny                1.6.0 CRAN       <NA>   <NA> [1]      <NA>
## sourcetools          0.1.7 CRAN       <NA>   <NA> [1]      <NA>
## sp                   1.4-5 CRAN       <NA>   <NA> [1]      <NA>
## spatial             7.3-12 CRAN       <NA>   <NA> [2]      <NA>
## splines               <NA> <NA>       <NA>   <NA> [2]  indirect
## staplr               3.1.1 CRAN      3.1.1   CRAN [1]    direct
## stats                 <NA> <NA>       <NA>   <NA> [2]  indirect
## stopwords              2.2 CRAN        2.2   CRAN [1]  indirect
## stringi              1.6.2 CRAN      1.6.2   CRAN [1]  indirect
## stringr              1.4.0 CRAN      1.4.0   CRAN [1]    direct
## survival            3.2-11 CRAN       <NA>   <NA> [2]      <NA>
## sylly                0.1-6 CRAN       <NA>   <NA> [1]      <NA>
## sylly.en             0.1-3 CRAN       <NA>   <NA> [1]      <NA>
## sys                    3.4 CRAN        3.4   CRAN [1]  indirect
## tcltk                 <NA> <NA>       <NA>   <NA> [2]  indirect
## testthat             3.0.3 CRAN       <NA>   <NA> [1]      <NA>
## tibble               3.1.2 CRAN      3.1.2   CRAN [1]    direct
## tidyr                1.1.3 CRAN      1.1.3   CRAN [1]    direct
## tidyselect           1.1.1 CRAN      1.1.1   CRAN [1]  indirect
## tidytext             0.3.1 CRAN      0.3.1   CRAN [1]    direct
## tidyverse            1.3.1 CRAN      1.3.1   CRAN [1]    direct
## tinytex               0.32 CRAN       0.32   CRAN [1]  indirect
## tokenizers           0.2.1 CRAN      0.2.1   CRAN [1]  indirect
## tools                 <NA> <NA>       <NA>   <NA> [2]  indirect
## triebeard            0.3.0 CRAN      0.3.0   CRAN [1]  indirect
## urltools             1.7.3 CRAN      1.7.3   CRAN [1]  indirect
## usethis              2.0.1 CRAN       <NA>   <NA> [1]      <NA>
## utf8                 1.2.1 CRAN      1.2.1   CRAN [1]  indirect
## utils                 <NA> <NA>       <NA>   <NA> [2]    direct
## uuid                 0.1-4 CRAN      0.1-4   CRAN [1]  indirect
## vctrs                0.3.8 CRAN      0.3.8   CRAN [1]  indirect
## viridisLite          0.4.0 CRAN      0.4.0   CRAN [1]  indirect
## vitae           0.4.2.9001 GitHub 0.4.2.9001 GitHub [1]    direct
## waldo                0.2.5 CRAN       <NA>   <NA> [1]      <NA>
## whisker                0.4 CRAN        0.4   CRAN [1]  indirect
## withr                2.4.2 CRAN      2.4.2   CRAN [1]  indirect
## wk                   0.4.1 CRAN       <NA>   <NA> [1]      <NA>
## wordcloud              2.6 CRAN        2.6   CRAN [1]    direct
## wordcountaddin  0.3.0.9000 GitHub     <NA>   <NA> [1]      <NA>
## xfun                  0.23 CRAN       0.23   CRAN [1]  indirect
## xml2                 1.3.2 CRAN      1.3.2   CRAN [1]  indirect
## xopen                1.0.0 CRAN       <NA>   <NA> [1]      <NA>
## xtable               1.8-4 CRAN       <NA>   <NA> [1]      <NA>
## yaml                 2.2.1 CRAN      2.2.1   CRAN [1]  indirect
## zip                  2.2.0 CRAN       <NA>   <NA> [1]      <NA>
##
## [1]: /home/daniel/git/phd-package/renv/library/R-4.1/x86_64-pc-linux-gnu
## [2]: /tmp/RtmpDn0EfB/renv-system-library
##
## # User Profile =======================
##                      Source    Package Require Version   Dev
## 1 /home/daniel/.Rprofile dadjokeapi                   FALSE
## 2 /home/daniel/.Rprofile      utils                   FALSE
##
## # Settings ===========================
## List of 8
##  $ external.libraries       : chr(0)
##  $ ignored.packages         : chr(0)
##  $ package.dependency.fields: chr [1:3] "Imports" "Depends" "LinkingTo"
##  $ r.version                : chr(0)
##  $ snapshot.type            : chr "implicit"
##  $ use.cache                : logi TRUE
##  $ vcs.ignore.library       : logi TRUE
##  $ vcs.ignore.local         : logi TRUE
##
## # Options ============================
## List of 6
##  $ defaultPackages     : chr [1:6] "datasets" "utils" "grDevices" "graphics" ...
##  $ download.file.method: NULL
##  $ download.file.extra : NULL
##  $ repos               : Named chr "https://cloud.r-project.org"
##    ..- attr(*, "names")= chr "CRAN"
##   $ renv.consent          : logi TRUE
##   $ renv.verbose          : logi TRUE
##
## # Environment Variables ==============
## HOME                          = /home/daniel
## LANG                          = en_US.UTF-8
## R_LIBS                        = /home/daniel/git/phd-package/renv/library/R-4.1/x86_64-pc-linux-gnu:/tmp/RtmpMqpjLb/renv-
        system-library
## R_LIBS_SITE                   = /usr/local/lib/R/site-library:/usr/lib/R/site-library:/usr/lib/R/library
## R_LIBS_USER                   = /home/daniel/git/phd-package/renv/library/R-4.1/x86_64-pc-linux-gnu:/tmp/RtmpDn0EfB/renv-
        system-library
## RENV_DEFAULT_R_ENVIRON        = <NA>
## RENV_DEFAULT_R_ENVIRON_USER   = <NA>
## RENV_DEFAULT_R_LIBS           = <NA>
## RENV_DEFAULT_R_LIBS_SITE      = /usr/local/lib/R/site-library:/usr/lib/R/site-library:/usr/lib/R/library
## RENV_DEFAULT_R_LIBS_USER      = ~/R/x86_64-pc-linux-gnu-library/4.1
## RENV_DEFAULT_R_PROFILE        = <NA>
## RENV_DEFAULT_R_PROFILE_USER   = <NA>
## RENV_PROJECT                  = /home/daniel/git/phd-package
##
## # PATH ===============================
## - /home/daniel/.poetry/bin
## - /home/daniel/.rvm/gems/ruby-2.7.1/bin
## - /home/daniel/.rvm/gems/ruby-2.7.1@global/bin
## - /home/daniel/.rvm/rubies/ruby-2.7.1/bin
## - /home/linuxbrew/.linuxbrew/bin
## - /home/daniel/.local/bin
## - /home/daniel/bin
## - /usr/local/sbin
## - /usr/local/bin
## - /usr/sbin
## - /usr/bin
## - /sbin
## - /bin
## - /usr/games
## - /usr/local/games
## - /snap/bin
## - /home/daniel/.rvm/bin
## - /usr/lib/rstudio/bin/postback
##
## # Cache ==============================
## There are a total of 182 package(s) installed in the renv cache.
## Cache path: "~/.local/share/renv/cache/v5/R-4.1/x86_64-pc-linux-gnu"
```