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.5108218 under CC-BY-4.0 license. This version was compiled on 2021-07-15 based on git commit `f228110e` 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 unit to increase transparency, reproducibility, and reusability of computational research. This article introduces the ERC reproducibility service (ERS), which supports publication workflows 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 we 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 investigation 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

As computers and algorithms infuse all scientific disciplines, Open Science and reproducibility are enormous challenges for research in geography and geosciences (David *et al.*, 2016; Nüst and Pebesma, 2020), and the typical scientific paper falls short of communicating the actual scholarship (Brammer *et al.*, 2011; Marwick, 2015; Gil *et al.*, 2016). The relevance of openness and reproducible, reusable research are undisputed, but the problems in applying them in daily work and the challenges around reproducibility 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 being made on openness in geospatial sciences, reproducibility has not been systematically addressed, and more stringent reproducibility requirements for publication have only recently been implemented (Minghini *et al.*, 2020; cf. Peng and Hicks, 2021). Thus, further development of infrastructure for supporting reproducibility is needed (Peng and Hicks, 2021).

To achieve sufficient openness and reproducibility, all aspects of science take on the common goal of changing the existing culture. Only a general broad change can support and motivate researchers to shift towards good Open Science and reproducible research practices. Examples for areas where change is needed are (i) requirements established by funders and journals (cf. Hardwicke *et al.*, 2018, and Stodden *et al.* (2018); Nüst *et al.*, 2018), (ii) mechanisms to award recognition to all types of research outputs (Piwowar, 2013), and (iii) 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 a demonstration of its 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 of scholarly publications and serves the particular needs of geospatial data sciences. Examples and applications are taken from these domains, 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. Locating 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). As previous work has presented the ERC's 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 we conclude with a summary and an outlook on future work.

2. Related work

Containerisation is widely adopted as a technology to capture general computing environments around computational workflows (Boettiger, 2015), but it can also be used 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 researchers' computers and cloud infrastructures; and *ease of use*, i.e., abstracting away the complexities of managing the environment, enabling use by researchers. Workflow tools can automate the process of capturing 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 available for capturing environments are manifold, and once the respective package exists, portability is a given. However, the approaches do vary considerably in their usability and accessibility.

One approach, *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 allowing for 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 of 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 it also requires that authors are familiar with a number of complex tools. In one work, Chuah *et al.* (2020) bridge between tracing and declarative approaches and also generate `Dockerfiles` for workflows, but by 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 they emphasise 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 preserving 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 the 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, for example capturing contributions pre-publication and extensions post-publication relying on GNU Make, but it requires familiarity with low-level tools. Finally, *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 can be downloaded and executed on a local computer. The main differences are the capturing of reviewer comments and ratings within the `.pm` file and the use of VMs. Another tool, *Inkling* (Castleberry *et al.*, 2013), has own file formats for documents and workflow configuration based on LaTeX and for 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 the scientific contents of software into so-called pure computations (Hinsen, 2015). Hinsen presents extensive requirements, two prototypical implementations, and important lessons learned. Similar to ERC and ERS, ActivePapers demonstrates the feasibility of packaging reproducible research, but with a different approach without containerisation. That works's strong theoretical base and implementation from the ground up 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 researchers to more deeply and more often delve into the tools, and, thus, represents a more long-term change than the current scope of the ERS.

None of these related projects and ideas have found

[1] https://repo2docker.readthedocs.io/en/latest/specification.html

[2] https://www.nist.gov/programs-projects/cloud-reproducible-records

considerable uptake outside of specific groups or communities; this is also true for the ERC and ERS. 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; Editorial, 2018), and others develop their 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 the 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 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.

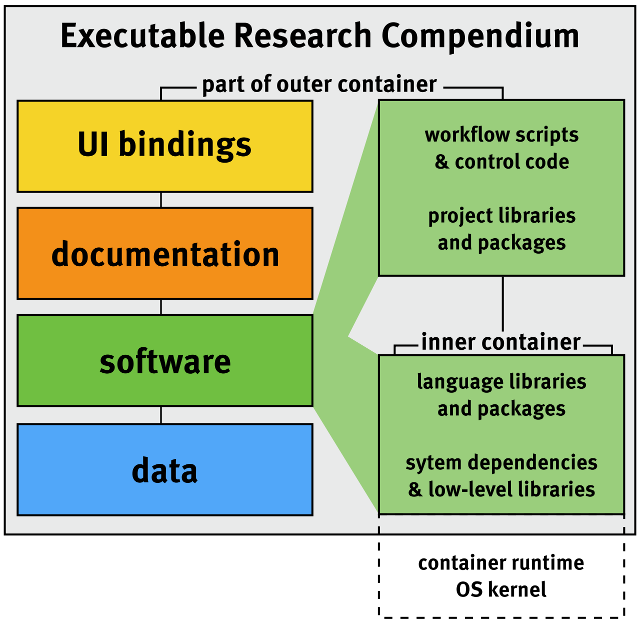
More practically, the ERC technical specification should support the goals of the ERC as described in Nüst *et al.* (2017) and serve as the foundation for the implementation of a Web service for creating, examining, and finding ERCs. The realisation is guided by several design goals. All these goals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

1. Simplicity and convention over configuration. The specification should not re-do something which already exists, e.g., in the form of an open specification or tool, and should 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 generally be 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", meaning before submission or after completing a research project. While it would be beneficial to steer researchers' workflows on a highly reproducible track from the beginning, because it provides a basis for real collaboration (Whitehouse, 2019), this is unrealistic because of researcher freedom, diversity in previous knowledge, and the evolutionary slow change of habits and practices. The majority of cases should be covered by following regular conventions, whereas special cases should be supported with configuration.

2. Nested containers. We acknowledge the 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 an executable runtime container and a 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 configurations and, as much as possible, also the content should be based on plain text files. Plain text files are usable

[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.



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 UI bindings, documentation, data, workflow code and project libraries.

by both humans and computers, ensure that ERCs are acceptable by users with varying backgrounds and levels of expertise, guarantee that ERCs remain understandable in the future, and enable ERCs to be easily extended. 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.

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 usersÿ 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 comprise the background on goals, design decisions, and the development process. The specification document uses technical language to clearly identify requirements and optional features, but it 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 is archived in Nüst (2021). 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 it 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 build and must be tagged matching to the ERC ID. To enable controlling the workflow through tools, the default commands of the image 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:

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

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

```
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 it 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 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 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 that ERC-supporting tools can implement. An ERC check comprises the execution of the workflow and the comparison of the ERC's files after the execution. The 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 (2021). 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 between the *reproducibility service*, i.e., the ERS, and 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 a 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 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 users, it is important that these features are provided in a guided process with excellent user experience, without too much exposure 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 it can be scrutinised and will not put the rigorousness of the actual ERCs into question.

The considered **stakeholders** in the architecture are the author (scientist), who publishes an ERC as part of a scientific publication process to build a convincing argument, the reviewer or editor (scientist), who examines an ERC during a review process to assess reproducibility and validity of results, the reader (scientist), who views and interacts with an ERC on a journal website to understand methods and build upon results, the publisher, who offers ERC-based publishing to increase the quality of publications, the curator or preservationist, who ensures research is complete and archivable using the ERC, the operator, who provides infrastructure to researchers at their own university or the publisher to communicate high-quality research using an ERC, and the developer, who uses and extends 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 not 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, allow 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 medi-

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

ately 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 provide software artefacts during ERC creation, 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 provide content to the reproducibility service for ERC creation but they 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 of their scope, e.g., the archives must ensure long-term access rights or technical accessibility. Therefore, these preservation 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 is described by the **architectural decisions**. First, the developed solution is set in an existing system of services, and first and foremost, it 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, to support the large variety of skills available on 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, whereby the database's operation log is used for notifications and events across microservices, e.g., for enabling 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. Fourth, 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 for both integration into microservices and standalone use cases. 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 they also allow running 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 a black box, where only the outside appearance and interaction options are described, and a 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 ERS as shown in Figure 3. The ERS itself consists of a web server to distribute incoming API calls to the microservices as a reverse proxy and to serve the static files of the user interface. The web server also manages secure communication via HTTPS. The microservices run in containers. They 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 that is mounted into every microservice container. The web server 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 does not have to 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 possibly *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

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

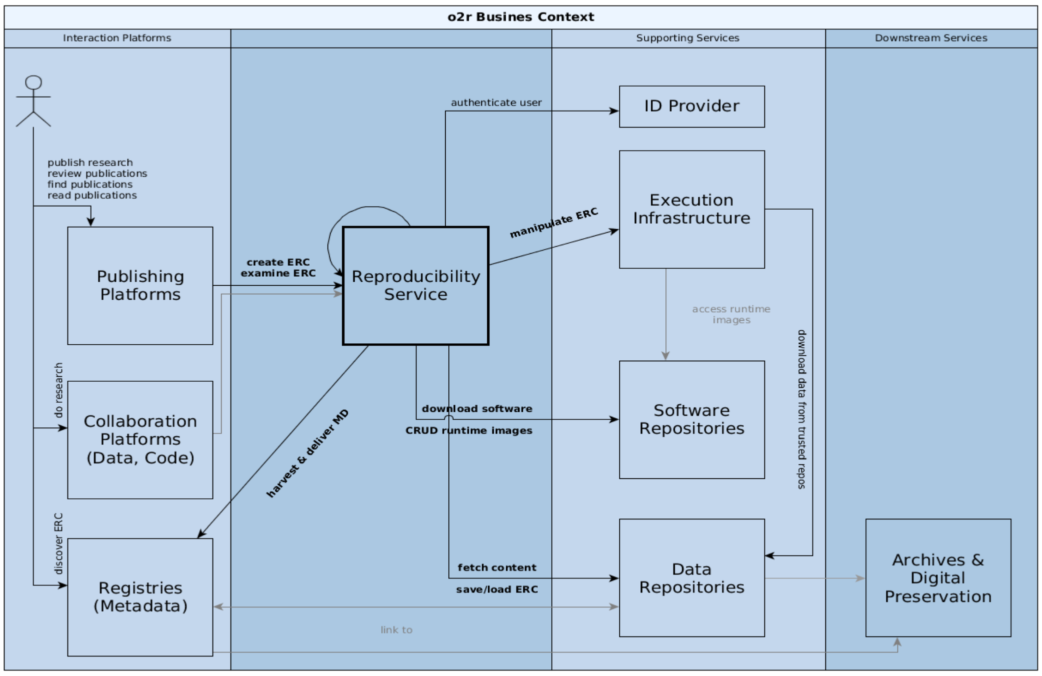


Fig. 2. Business context. Full-scale image online at https://o2r.info/architecture/#31-business-context.

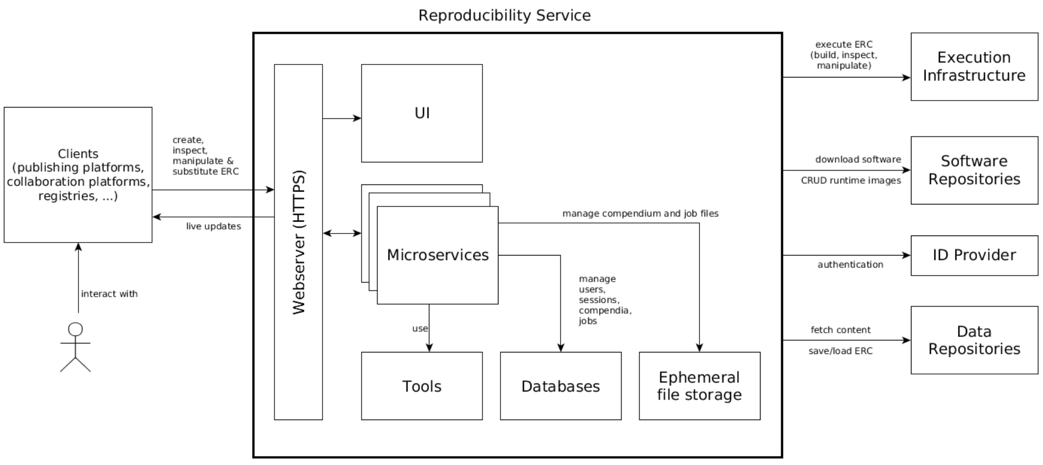


Fig. 3. Reproducibility service. Full-scale image online at https://o2r.info/architecture/#527-whitebox-reproducibility-service.

containerit tool and the execution infrastructure. The primary means to create a complete manifest is executing the whole workflow. Alternatively, containerit also detects existing computing environment information that is stored in the provided workspace, namely a `sessionInfo` object for R saved in a file `sessionInfo.Rda`. 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, the step for generating them 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 allow the user to check that the ERC is correctly uploaded to the repository before publication, the use must take an extra action to publish the shipment.

In the **ERC examination sequence**, the user initiates the opening of an existing ERC by providing a reference such as a 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 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 the ERS 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 sent 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 Figures 6 and 8. Based on these aids, the reader, e.g., the reviewer, can quickly determine whether deviations in the outputs are relevant or not, e.g., if they are only graphical artefacts or acceptable numerical variation as in Figure 6, or incorrect results as in Figure 8.

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 notifications 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 (2021). 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 (who undergo a manual check following 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 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 `../job? limit=10&compendium_id=abc12 &status=success&fields=user` (spaces added for readability) 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` facilitates 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.

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

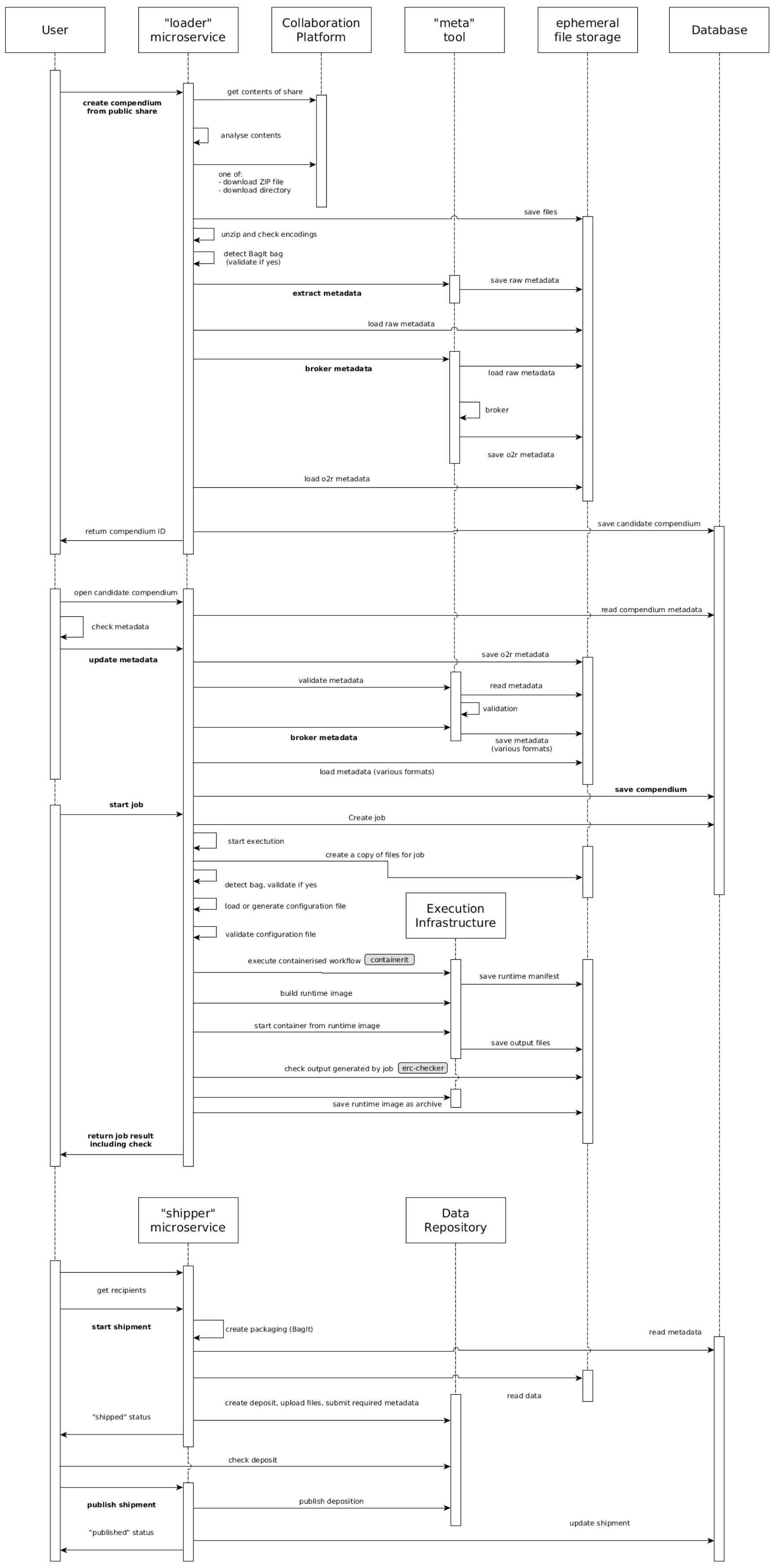


Fig. 4. Runtime view. ERC Creation; full-scale image online at https://o2r.info/architecture/#61-erc-creation.

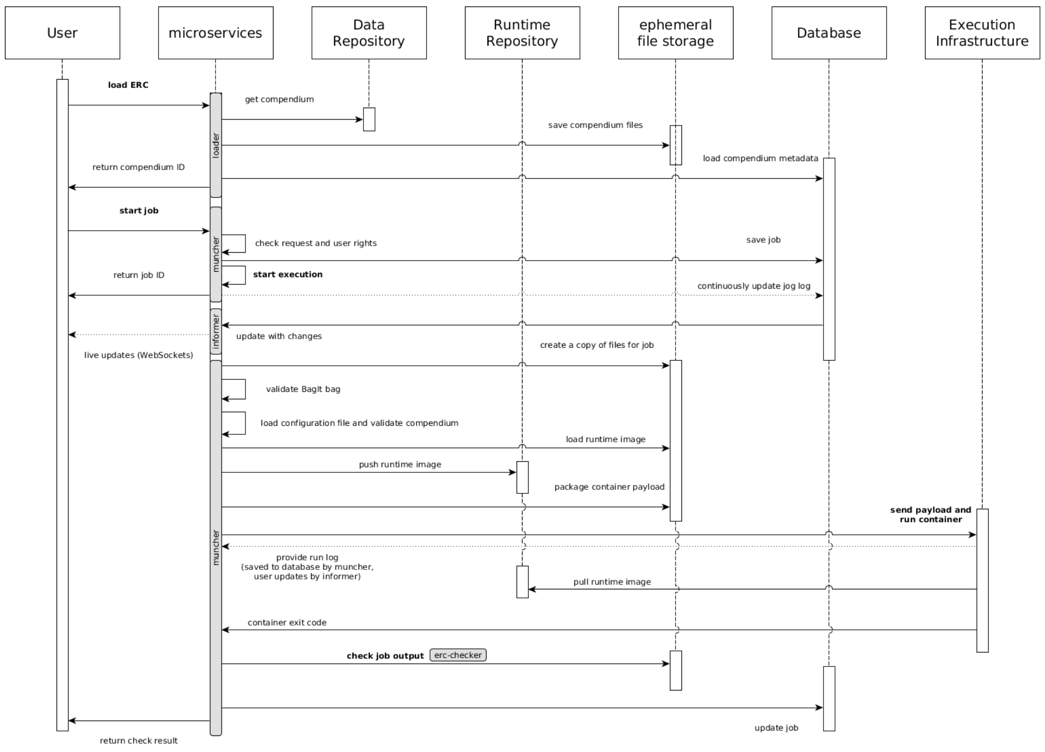
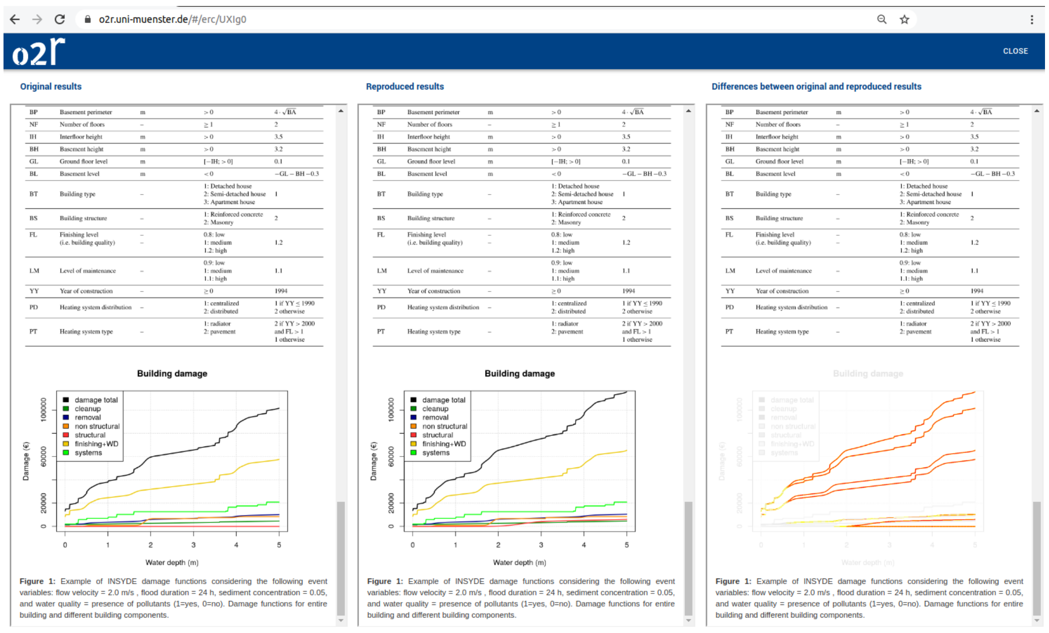


Fig. 5. Runtime view ERC examination; full scale image online at https://o2r.info/architecture/#62-erc-inspection.



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 ≤ 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 ≤ 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 ≤ 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.

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 one step is a `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 as follows (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)
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 (2021). 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.

```
# To use the images from Zenodo, download all files form the
# Zenodo deposit, then:
make reproduce

# To build from the source code in the git repository and
# run the ERS:
clone https://github.com/o2r-project/reference-implementation
cd reference-implementation
make local
```

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 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
```

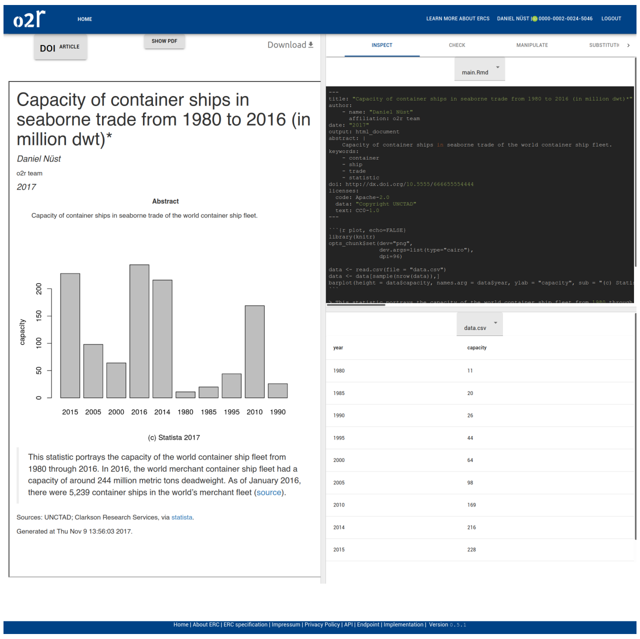
The `Dockerfile` defines the computational environment (extra line breaks for readability).

```
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 this example is contrived, 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 from the outer container into the inner 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

[16] https://github.com/o2r-project/
[17] https://github.com/o2r-project/reference-implementation
[18] https://github.com/o2r-project/erc-examples/
[19] ſ 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.

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) Statis
~~~

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

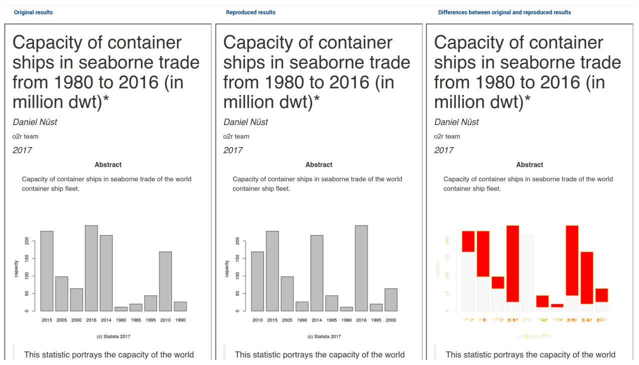
Fig. 7. Screenshot of ERC examination view in the ERS. The left hand side shows the display file rendering in HTML, the right hand side allows to inspect the source RăMarkdown document and the input data.

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 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 creating 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, but researchers would have to educate themselves on how to create and test them (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, which a reader can use as an entrypoint to inspect the workflow code, and the display file to open for reading. The workflow code is trivial in this example but in complex workflows can spread across multiple files. 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.



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

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

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. 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 in the least because of the confining context of a research project. Nevertheless, the implementation points out important aspects and taught valuable lessons, which can help to adapt concepts, specifications, or even software for a productive infrastructure.

6.1. Project set-up, maintainability, and security. On the project setup 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 does it have measures to evaluate quality of data, software, or scientific merit. This narrow aim, even

[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[ˆ14].

though the internal tools are complex, improves the usability and extensibility of the ERC and ERS.

The complexity introduced through the many microservices was good at the start, as it provided flexibility, but the need for consolidation for sustainability has led 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. One mitigation that would work, at least until the ERS itself breaks, could be the integration of the ERS with `repo2docker` (cf. Project Jupyter *et al.*, 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 examine 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, at this point the ERS is fully dependent on the Docker container runtime—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 that containers are used in research, the more likely it will become that a special container and image specification, which can be maintained long term and 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 were not designed with preservation in mind[25]. An alternative mitigation could be multiple (cf. Glatard *et al.*, 2018) container engines, which as of yet could not be realised for the ERS. 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 it should be understandable by all **researchers** using computational methods. The core concepts of computational notebooks and containerisation are becoming more widespread across researchers, improving the reproducibility of their works, such that the combination of both these approaches 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 that researchers 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. Hinsen, 2018) that are suitable for their work, as long as 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

[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

configuration of the time zone, which led to check failures because times were 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 an 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 be 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 they also go further as other reproducibility formats in explicitly modelling them. This is a compromise to at least provide compatible licenses for important parts, but does not do enough justice ti highlighting the importance of software citation (Katz *et al.*, 2021) and giving contributors credit. 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 unaltered 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 the API as reasonably good for a version 1, though it might be helpful to more clearly distinguish between the loading of workspace and the opening of an ERC, which currently is realised with requests to the same API endpoint. The integration of ERS into publishing systems has not been realised, e.g., regarding user authentication; the 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).

[27] https://www.orcid.org/

[28] https://o2r.info/api/evaluation/load_test.html

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 for taking snapshots at the point of submission and making these snapshots available to peer reviewers do they can **assist examination of manuscripts** in several ways: First, the visual comparison of the display files created by the authors and the ERS itself; second, the UI bindings interact with specific parts of the workflow; and 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 judgement calls 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 from a challenge in human-computer interaction (HCI). The latter perspective focuses on usage and reasoning, which is more important for verifiability. In the same sense, UI bindings aid verification on the basis of a reproducible computation.

One 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 (i) it made the reproducibility service development and installation more complex, and (ii) discovery through the reproducibility service is not a long-term solution, as it only offers short-term storage of ERCs. Leveraging the connections between the ERC's parts and the exposure of the main document and software stack for search and discovery should be placed at the repositories storing ERCs.

The ERC as a **snapshot** is naturally a compromise between reliability—something that works now for a specific purpose—and reusability—something that can be extended and built 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 snapshot 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 it 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 allow reproducibility services to be able to decide whether 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. However, the nature of secured communication leads to these backups being black boxes 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 during and after peer review (cf. Nüst and Pebesma, 2020).

At first glance, the ERS seems to be severely limited by the focus on **R Markdown** for the main file and HTML for the display file. Yet, R is the lingua franca of statistics and is used more and more over proprietary alternatives, but more importantly, using R Markdown as a common ground format for reproducible research is second to none when it comes to creating publication-ready display documents. It supports citation management, templates, both Web and print output formats (i.e., PDF), and transparency due to its plain text nature[29]. 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). Templates for R Markdown could be provided by publishers, though today most are 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 approaches such as ERA (Guizzardi *et al.*, 2021) that connect computational notebooks with standardised publishing formats could be easier to adopt, but they lack some of the ERS's features.

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 is 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 for capturing 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 explicitly does 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 could be extended to 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 while disregarding general ignorance of how quickly digital resources and free services may decay or disappear[31]. First, we cannot imagine today what computers will look like in 50 or 60 years. Yet, science historians might still find a lot of valuable information in ERC. Second, even though some software (e.g., FORTRAN, GNU Make) has been around a long time, for the

[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/.

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 will still be 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 break 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 already a declining availability of data (Vines *et al.*, 2014).

Peng (2017) suggests introducing limiting principles so that practical implications do not stall the goals of 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 10 GB 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 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, five, 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 of whether to package, e.g., a reduced example and preprocessed data.

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 subspectrum is determined by 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) devise 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 for checking whether 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, but the ERC is far from the explicit level of detail 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 "messy" shortcomings Thain et al. describe for virtual machines and container technology, but the ERC cannot capture distributed systems (machines, file systems). Regarding the preservation objectives, the ERC focuses on *Identical Verification*,, verifying that 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 for updating 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 readable to both human and machine and to *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 seven levels in three main categories: accessibility, executability, and interactivity. 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.

Finally, Hinsen (2020) formulates *four essential possibilities* from a scientists point of view as the basis for constructive discussion about reproducible scientific computations. The ERC and ERS can enable the first three possibilities: Code and data can be inspected, workflows can run for verification both in the ERS and locally, and intermediate results or modifications can be explored. The fourth possibility to verify that executable software and source code match can be achieved for the workflow code which is available in source, but not completely for the libraries in the inner container. The ERS assumes these dependencies to simply be "correct", though where available the source packages of tools could be installed alongside the executable binaries. The blog post rightly argues the complexity of today's software stacks make not using precompiled binaries a hassle. Hinsen further discusses the variation over time of the four possibilities, which can be reduced by the ERS's snapshotting approach, and that the degrees to which the possibilities need to be fulfilled may depend on the research at hand.

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 sharing snapshots of research workflows for review and reading, and they can be integrated in a scholarly publication process. Further, 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, such as piecemeal approaches for publication, review, and publishing 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 whether the technical and organisational innovations can benefit from each other or are better introduced successively.

However, an uptake of ERC and ERS by third parties has not yet been achieved 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 be attributed to the many moving parts for adapting to more reproducible and transparent processes, 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 for promoting reproducibility are widely discussed (e.g., Hrynaszkiewicz, 2020; Eglen *et al.*, 2018). However, existing pilots yield promising results with 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 making a "reproducibility package" as well as reports about reproductions (cf. Nüst and Eglen, 2021) first level citizens or types of research output in databases such as CrossRef or repositories such as Zenodo. This would give recognition to and aid discovery of data, software, and reproducible computational workflows. Nevertheless, the presented software does provide a basis for further testing with stakeholders, in order 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, but 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 orchestrating 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—this problem 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 towards which open community-owned research infrastructures 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 like the executable research compendium reproducibility service that can provide transparency, reproducibility, and reusability.

[32] https://www.earthcube.org/notebooks

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

[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).

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 and Celeste R. Brennecka from the Scientific Editing Service, University of Münster, for her editorial review. 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 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.

Editorial (2018). “Easing the burden of code review.” *Nature Methods*, **15**(9), 641–641. ISSN 1548-7105. doi:10.1038/s41592-018-0137-5.

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). “FORCE—Landsat + 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.

Hinsen K (2020). "The four possibilities of reproducible scientific computations." URL https://blog.khinsen.net/posts/2020/11/20/the-four-possibilities-of-reproducible-scientific-computations/.

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 (2021). "Reproducibility Service for Executable Research Compendia: Technical Specifications and Reference Implementation (Version 1.1.0)." *Technical report*. doi:10.5281/zenodo.5106499.

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.