Data Science -1 (2021) 1–21
DOI 10.3233/DS-210031
IOS Press

WORCS: A workflow for open reproducible code in science

Caspar J. Van Lissa [a,*,**], Andreas M. Brandmaier [b], Loek Brinkman [c], Anna-Lena Lamprecht [d], Aaron Peikert [e], Marijn E. Struiksma [f] and Barbara M.I. Vreede [g]

[a] *Department of Methodology & Statistics, Utrecht University, The Netherlands and Open Science Community Utrecht, The Netherlands*
E-mail: c.j.vanlissa@uu.nl; ORCID: https://orcid.org/0000-0002-0808-5024

[b] *Center for Lifespan Psychology, Max Planck Institute for Human Development, Berlin, Germany and Max Planck UCL Centre for Computational Psychiatry and Ageing Research, Berlin, Germany and London, UK*
ORCID: https://orcid.org/0000-0001-8765-6982

[c] *Open Science Community Utrecht, The Netherlands and University Medical Center, Utrecht University, The Netherlands*
ORCID: https://orcid.org/0000-0003-3997-1173

[d] *Open Science Community Utrecht, The Netherlands and Department of Information and Computing Sciences, Utrecht University, The Netherlands*
ORCID: https://orcid.org/0000-0003-1953-5606

[e] *Center for Lifespan Psychology, Max Planck Institute for Human Development, Berlin, Germany*
ORCID: https://orcid.org/0000-0001-7813-818X

[f] *Open Science Community Utrecht, The Netherlands and Utrecht Institute of Linguistics OTS, Utrecht University, The Netherlands*
ORCID: https://orcid.org/0000-0002-1166-1424

[g] *University Library, Utrecht University, The Netherlands and Netherlands eScience Center, The Netherlands*
ORCID: https://orcid.org/0000-0002-5023-4601

Editor: Tobias Kuhn (https://orcid.org/0000-0002-1267-0234)
Solicited reviews: Daniel Garijo (https://orcid.org/0000-0003-0454-7145); Remzi Celebi (https://orcid.org/0000-0001-7769-4272); Katherine Wolstencroft (https://orcid.org/0000-0002-1279-5133); one anonymous reviewer

Received 25 September 2020
Accepted 11 December 2020

Abstract. Adopting open science principles can be challenging, requiring conceptual education and training in the use of new tools. This paper introduces the Workflow for Open Reproducible Code in Science (WORCS): A step-by-step procedure

*Corresponding author. E-mail: c.j.vanlissa@uu.nl.
**All authors of this paper, except the first, are listed in alphabetical order. Author contributions are detailed in the document 'authors.csv' on the project repository, at https://github.com/cjvanlissa/worcs.

that researchers can follow to make a research project open and reproducible. This workflow intends to lower the threshold for adoption of open science principles. It is based on established best practices, and can be used either in parallel to, or in absence of, top-down requirements by journals, institutions, and funding bodies. To facilitate widespread adoption, the WORCS principles have been implemented in the R package `worcs`, which offers an RStudio project template and utility functions for specific workflow steps. This paper introduces the conceptual workflow, discusses how it meets different standards for open science, and addresses the functionality provided by the `R` implementation, `worcs`. This paper is primarily targeted towards scholars conducting research projects in R, conducting research that involves academic prose, analysis code, and tabular data. However, the workflow is flexible enough to accommodate other scenarios, and offers a starting point for customized solutions. The source code for the `R` package and manuscript, and a list of examples of WORCS projects, are available at https://github.com/cjvanlissa/worcs.

Keywords: Open science, reproducibility, r, dynamic document generation, version control

1. Introduction

Academia is arguably past the tipping point of a paradigm shift towards open science. Support for this transition was first motivated by several highly publicized cases of scientific fraud (Levelt et al. [20]), increasing awareness of questionable research practices (John et al. [16]), and the replication crisis (Shrout and Rodgers [36]). However, open science should not be seen as a cure (or punishment) for this crisis. As the late dr. Jonathan Tennant put it: "Open science is just good science" (Tennant [39]). Open science creates opportunities for researchers to more easily conduct reliable, cumulative, and collaborative science (Adolph et al. [3]; Nosek and Bar-Anan [24]). Open science also promotes inclusivity, because it removes barriers for participation. Capitalizing on these advances has the potential to accelerate scientific progress (see also Coyne [11]), as has been aptly demonstrated by the role of open science in the response to the coronavirus pandemic (Sondervan et al. [37]).

Many researchers are motivated to adopt current best practices for open science and enjoy these benefits. And yet, the question of how to get started can be daunting. Making the transition requires researchers to become knowledgeable about different open science challenges and solutions, and to become proficient with new and unfamiliar tools. This paper is designed to ease that transition by presenting a simple workflow, based on established best practices, that meets most requirements open science: The *Workflow for Open Reproducible Code in Science* (WORCS).

This paper introduces the conceptual workflow (referred to in upper case, "WORCS"), and discusses its underlying principles and how it meets different standards for open science. The principles underlying this workflow are universal and largely platform independent. We also present a software implementation of WORCS for `R` users (R Core Team [31]): The `R` package `worcs` (referred to in monospace font, Van Lissa et al. [41]). This package offers a project template for RStudio (RStudio Team [35]), and several convenience functions to automate specific workflow steps. This paper is most relevant for scholars using R to conduct research that involves academic prose, analysis code, and tabular data. For other readers, it can serve as a primer on open science workflows, and provide a sensible starting point for a customized solution.

The WORCS workflow constitutes a step-by-step procedure that researchers can follow to make a research project open and reproducible. These steps are elaborated in greater detail below. See Fig. 1 for a flowchart that highlights important steps in different stages of the research process. WORCS is compatible with open science requirements already implemented by journals and institutions, and will help fulfill them. It can also be used by individual authors to produce work in accordance with best practices in the absence of top-down support or guidelines (i.e., "grass roots" open science).

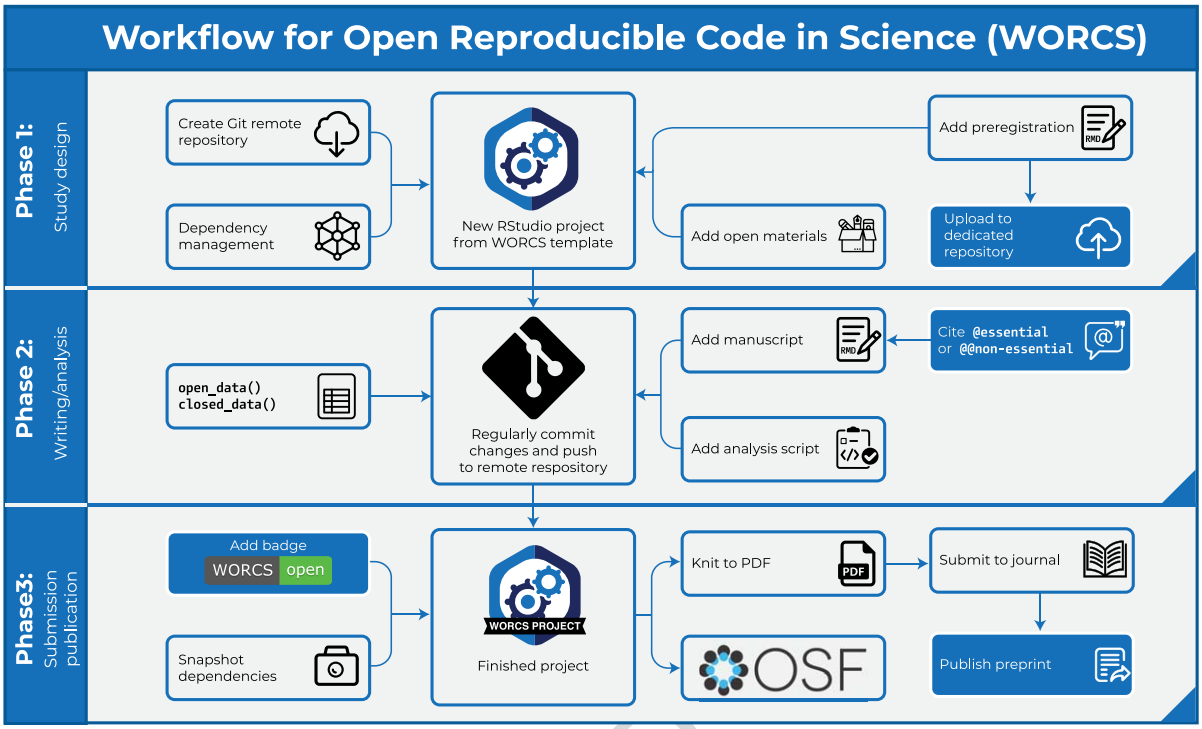

Fig. 1. Schematic illustration of important steps in different stages of the WORCS procedure.

2. Defining open science practices

Although open science is advocated by many, it does not have a unitary definition. Throughout this paper, we adhere to the definition of open science as "the practice of science in such a way that others can collaborate and contribute, where research data, lab notes and other research processes are freely available, under terms that enable reuse, redistribution and reproduction of the research and its underlying data and methods" (the FOSTER Open Science Definition, as discussed in Bezjak et al. [8]). In this context, the term "science" refers to all scholarly disciplines. We further define the term *reproducible*, as used in the WORCS acronym, as being able to re-perform "the same analysis with the same code using a different analyst" (Patil et al. [27]).

The "TOP-guidelines" are one of the most influential concrete operationalisations of general open science principles (Nosek et al. [23]). These guidelines describe eight standards for open science, the first seven of which "account for reproducibility of the reported results based on the originating data, and for sharing sufficient information to conduct an independent replication" (Nosek et al. [23]). These guidelines are: 1) Comprehensive citation of literature, data, materials, and methods; 2) sharing data, 3) sharing the code required to reproduce analyses, 4) sharing new research materials, and 5) sharing details of the design and analysis; 6) pre-registration of studies before data collection, and 7) pre-registration of the analysis plan prior to analysis. The eighth criterion is "not formally a transparency standard for authors", but "addresses journal guidelines for consideration of independent replications for publication". In this context, reproducibility can be defined as "re-performing the same analysis with the same code using a different analyst", and replicability as "re-performing the experiment and collecting new data" (Patil et al. [27]). WORCS defines the goals of open science in terms of the first seven TOP-guidelines,

and the workflow and its R implementation are designed to facilitate compliance therewith. We group these guidelines into three categories: citation (1), sharing (2–5), and preregistration (6–7).

2.1. Introducing the tools

WORCS relies solely on a set of free, open source software solutions which we will discuss before introducing the workflow.

2.1.1. Dynamic document generation

The first is *dynamic document generation* (DDG): Writing scientific reports in a format that interleaves written reports with blocks of code used to conduct the analyses. The text is automatically formatted as a scientific paper in several potential styles. When the text is formatted, the code blocks are evaluated and their results are insterted in the text, or rendered as figures and tables. Dynamic document generation supersedes the classical approach of using separate programs to write prose and conduct analyses, and then manually copy-pasting analysis results into the text. This paper is an example of DDG, and its source code is available here.

Although transitioning to DDG involves a slight learning curve, we strongly believe that the investment will pay off. Time saved from painstakingly copy-pasting output and manually formatting text soon outweighs the investment of switching to a new program. Moreover, human error in manually copying results is eliminated. When revisions require major changes to the analyses, all results, figures and tables are automatically updated. The flexibility in output formats also means that a manuscript can be rendered to presentation format, or even to a website or blog post. Moreover, the fact that code is evaluated each time the document is compiled requires researchers to work reproducibly, and allows reviewers and/or readers verify reproducibility simply by re-compiling the document. While writing academic papers in a programming environment might seem counter-intuitive at first, this approach is much more amenable to the needs of academics than most word processing software. It prevents mistakes, and saves time.

In the R implementation of WORCS, we recommend centering a research project around one dynamically generated RMarkdown document (Xie et al. [47]). RMarkdown is based on Markdown, which uses plain-text commands to typeset a document. It enhances this format with support for R-code, and is compatible with LaTeX math typesetting. For a full overview of the functionality of RMarkdown, see Xie et al. [47].

The RMarkdown document can incorporate blocks of analysis code, which can be used to render results directly to Tables and Figures, which can be dynamically referenced in the text. Long scripts can be stored in .R files, and included in the main document using the read_chunk() function from the knitr package, which comes together with the rmarkdown package. When a reader or reviewer compiles this document, all code is run automatically, thus verifying computational replicability. The RMarkdown document can be automatically formatted in many styles, including APA style (thanks to the R package papaja, Aust and Barth [7]), a host of other scientific formats (see Allaire et al. [4]), and as plain Markdown.

2.1.2. Version control

The second solution is *version control*: Maintaining an indelible log of every change to all project files. Version control is a near-essential tool for scientific reproducibility, as anyone learns who has had the experience of accidentally deleting a crucial file, or of being unable to reproduce analyses because they ran some analyses interactively, instead of documenting the syntax in a script file (see also Blischak et al. [9]). Many scientists use some form of *implicit* version control; for example, by renaming files

after major changes (e.g., "manuscript_final_2.2-2019-10-12.doc"), tracking changes in word processing software, or using cloud hosting services that retain backups of previous versions.

An integral part of WORCS is the *explicit* version control software Git (www.git-scm.com). A project version controlled with Git is called a "repository", or repo. Git tracks changes to files in the repository on a line-by-line basis. The user can store these changes by making a "commit": a snapshot of the version controlled files. Each "commit" can contain as many changes as desired; for example, a whole new paragraph, or many small changes made to address a single reviewer comment. Each commit is further tagged with a message, describing the changes. The `R` implementation of `worcs` contains a convenience function, `git_update("your commit message")`, which creates a new commit for all changes since the previous commit, labels it with "your commit message", and pushes these changes to the remote repository. It is, effectively, a Git "quick save" command.

From an open science perspective, Git is particularly appealing because it retains a complete historical backlog of all commits. This can be used, for example, to verify that authors executed the steps outlined in a preregistration. Users can also compare changes between different commits, or go back to a previous version of the code (for example, after making a mistake, or to replicate a previous version of the results). Git only version controls files explicitly committed by the user. Moreover, it is possible to prevent files from being version controlled – which is useful for privacy sensitive data. A `.gitignore` file lists files that should not be version-controlled. These files thus exist only on the user's computer.

The functionality of Git is enhanced by services such as GitHub (https://github.com). GitHub is best understood as a cloud storage service with social networking functionality. The cloud storage aspect of GitHub allows you to "clone" (copy) a local Git repository to the GitHub website, as a backup or research archive. The GitHub platform offers additional functionality over Git. One such feature is that specific stages in the lifecycle of a project can be tagged as a "release": A named downloadable snapshot of a specific state of the project, that is prominently featured on the GitHub project page. The WORCS procedure encourages users to create releases to demarcate project milestones such as preregistration and submission to a journal. Releases are also useful to mark the submission of a revised manuscript, and publication.

The social network aspect of GitHub comes into play when a repository is made "public": This allows other researchers to peruse the repository and see how the work was done; clone it to their own computer to replicate the original work or apply the methods to their own data; open "Issues" to ask questions or give feedback on the project, or even send a "Pull request" with suggested changes to the text or code for your consideration. Git and GitHub shine as tools for collaboration, because different people can simultaneously work on different parts of a project, and their changes can be compared and automatically merged on the website. Even on solo projects, working with Git/GitHub has many benefits: Staying organized, being able to start a new study with a clone of an old, similar repository, or splitting off an "experimental branch" to try something new, while retaining the ability to "revert" (return) to a previous state of the project, or to "merge" (incorporate) the experimental branch.

Although GitHub is the most widely used remote repository, `worcs` supports alternative services. We will refer to all such services as "remote repositories". The `worcs` package uses `gert` (Ooms [26]) to connect a local project to a remote repository. It also includes user-friendly functions to set user credentials, `git_user()`, and to add, commit, and push changed files to a remote repository in one step, `git_update()`.

2.1.3. Dependency management

The third solution is *dependency management*: Keeping track of exactly what software was used to conduct the analyses. At first glance, it might seem sufficient to state that analyses were conducted in

Program X. However, every program is susceptible to changes, updates, and bugfixes. Open Source software, in particular, is regularly updated because there is an active community of developers contributing functionality and bugfixes. Potentially, any such update could change the results of the code, thus rendering the analysis computationally non-reproducible.

Many solutions exist to ensure computational reproducibility, which differ in user-friendliness and effectiveness. These solutions typically work by enveloping your research project in a distinct "environment" that only has access to programs that are explicitly installed, and maintaining a record of these programs. When choosing an appropriate solution for dependency management, there is a tradeoff between ease-of-use on the one hand, and robustness on the other. In making this tradeoff, it is important to consider that all solutions have a limited shelf life (Brown [10]). Therefore, the best approach might be to use a "good-enough" solution, and acknowledge that all code requires some maintenance if you want to reproduce it in the future.

For dependency management, the `R` implementation of `worcs` relies on the recently released package `renv` (Ushey [40]). This package strikes a great balance between user-friendliness and robustness. Developed by the team behind RStudio, `renv` maintains a text-based, human-readable log of all packages used, their version numbers, and where they were installed from (e.g., CRAN, Bioconductor, GitHub). This text-based log file can be version controlled with Git. When someone else loads your project, `renv` will install all of the required packages from this list. This lightweight approach safeguards reproducibility as long as these repositories maintain their archives. The `worcs` project template in RStudio automatically sets up `renv` for dependency management if the checkbox labeled "renv" is selected during project creation.

All solutions to computational reproducibility require users to make their dependencies explicit, often by reinstalling all software for each project in a controlled environment. This can lead to long installation times and large memory requirements. Similarly, `renv` requires users to call `install.packages()` for each library used – but it does not reinstall packages for each new project. Instead, packages reside in a cache that is shared between all projects on the computer. If a project uses a package that is already in the cache, it is not reinstalled, but linked from the cache. This improves the user experience by saving time and memory. As opposed to alternative solutions, `renv` is tightly integrated with the native R package management system. It monitors the scripts in the project directory for `library()` calls, and notifies the user when it is necessary to update the dependencies file using the `snapshot()` function. Therefore, `renv` does not require the users to change their workflow of installing/removing and using packages, and requires little technical skill to setup. It is not a fail-safe method to ensure strict computational reproducibility, but it meets most reproducibility requirements using only tools native to `R` and RStudio.

When a project is conducted in a different software environment than `R`, or if an R-project has external dependencies not tracked by `renv`, then users might choose to use Docker for dependency management.

Docker instantiates a "Docker container": An environment that behaves like a virtual computer, which can be stored like a time capsule, and identically reinstated on a user computer, or in the cloud. Although the use of Docker falls outside of the scope of the present paper, for existing users, a Docker build running `worcs` and its dependencies is available. The stetup with Docker vignette explains how to instantiate this build. For more information on this topic, Peikert and Brandmaier [28] describe a Docker-based workflow for computational reproducibility that is conceptually and practically compatible with WORCS.

As Docker is somewhat challenging to set up for novice users, an `R` package is currently in development to facilitate Docker integration (`repro`, Peikert et al. [29]). Another example of a Docker-based

workflow is the cloud-based collaboration platform "Code Ocean". This platform greatly simplifies setting up a reproducible computing environment with version control and dependency management. A key feature is that collaborators can log in to the server session, rather than each having to set up Docker on a local computer. One important consideration when using any cloud-computing solution is the necessity to upload human participant data for analysis. Such a platform would need to comply with relevant laws and ethical requirements. For example, as of this writing, the terms of use for Code Ocean do not address the GDPR, nor is it clear where data are stored. This is a potential liability for European scholars.

2.2. Text-based files are better

A key consideration when developing a research project is what filetypes to use. WORCS encourages the use of text-based files whenever possible, instead of binary files. Text-based files can be read by machines and humans alike. Binary files, such as Word (`.docx`) or SPSS (`.sav`, `.spo`) files, must be decoded first, often requiring commercial software. Although binary files can be version controlled with Git, their change log is uninterpretable. Binary files are also often larger than text-based files, which means they take up more space on cloud hosting services. For these reasons, uploading large binary files to remote repositories is frowned upon.

Two additional points are worth noting: First, as Git tracks line-by-line changes, it is recommended to start each sentence on a new line. That way, the change log will indicate which specific sentence was edited, instead of replacing an entire paragraph. As RMarkdown does not parse a single line break as the end of a paragraph, this practice does not disrupt the flow of a paragraph in the rendered version. Second, the remote repository GitHub renders certain text-based filetypes for online viewing: For example, `.md` (Markdown) files are displayed as web pages, and `.csv` files as spreadsheets. The `worcs` package uses this feature to, for example, render a codebook for the data as `.md` so it will be shown as a web page on GitHub.

3. Introducing the workflow

We provide a conceptual outline of the WORCS procedure below, based on WORCS Version 0.1.6. The conceptual workflow recommends actions to take in the different phases of a research project, in order to work openly and transparently in compliance with the TOP-guidelines and best practices for open science. Note that, although the steps are numbered for reference purposes, we acknowledge that the process of conducting research is not always linear. We refer users of the `worcs` implementation in `R` to the workflow vignette, which describes in more detail which steps are automated by the `R` package and which actions must be taken by the user. The vignette is also updated with future developments of the package. For examples of how the workflow is used in practice, we maintain a list of public `worcs` projects on the WORCS GitHub page.

3.1. Phase 1: Study design

1. Create a (Public or Private) remote repository on a "Git" hosting service
2. When using R, initialize a new RStudio project using the WORCS template. Otherwise, clone the remote repository to your local project folder.
3. Add a README.md file, explaining how users should interact with the project, and a LICENSE to explain users' rights and limit your liability. This is automated by the `worcs` package.

4. Optional: Preregister your analysis by committing a plain-text preregistration and tag this commit with the label "preregistration".
5. Optional: Upload the preregistration to a dedicated preregistration server
6. Optional: Add study materials to the repository

3.2. Phase 2: Writing and analysis

7. Create an executable script documenting the code required to load the raw data into a tabular format, and de-identify human subjects if applicable
8. Save the data into a plain-text tabular format like `.csv`. When using open data, commit this file to "Git". When using closed data, commit a checksum of the file, and a synthetic copy of the data.
9. Write the manuscript using a dynamic document generation format, with code chunks to perform the analyses.
10. Commit every small change to the "Git" repository
11. Use comprehensive citation

3.3. Phase 3: Submission and publication

12. Use dependency management to make the computational environment fully reproducible
13. Optional: Add a WORCS-badge to your project's README file
14. Make a Private "Git" remote repository Public
15. Create a project page on the Open Science Framework (OSF) and connect it to the "Git" remote repository
16. Generate a Digital Object Identifier (DOI) for the OSF project
17. Add an open science statement to the Abstract or Author notes, which links to the "OSF" project page and/or the "Git" remote repository
18. Render the dynamic document to PDF
19. Optional: Publish the PDF as a preprint, and add it to the OSF project
20. Submit the paper, and tag the commit of the submitted paper as a release of the submitted paper as a release, as in Step 4.

3.4. Notes for cautious researchers

Some researchers might want to share their work only once the paper is accepted for publication. In this case, we recommend creating a "Private" repository in Step 1, and completing Steps 13-18 upon acceptance.

3.5. The R implementation of WORCS

WORCS is, first and foremost, a conceptual workflow that could be implemented in any software environment. As of this writing, the workflow has been implemented for `R` users in the package `worcs` (Van Lissa et al. [41]). Several arguments support our choice to implement this workflow first in `R` and RStudio, although we encourage developers to port the software to other languages. First, `R` and all of it extensions are free open source software, which make it a tool of choice for open science. Second, all tools required for an open science workflow are implemented in `R`, most of these tools are directly accessible through the RStudio user interface, and some of them are actively developed by the team

behind RStudio. Third, RStudio is a Public Benefit Corporation, whose commitment to open science is codified in the company charter:

> RStudio's primary purpose is to create free and open-source software for data science, scientific research, and technical communication. This allows anyone with access to a computer to participate freely in a global economy that rewards data literacy; enhances the production and consumption of knowledge; and facilitates collaboration and reproducible research in science, education and industry.

Fourth, `R` is the second-most cited statistical software package (Muenchen [22]), following SPSS, which has no support for any of the open science tools discussed in this paper. Fifth, `R` is well-supported by a vibrant and inclusive online community, which develops new methods and packages, and provides support and tutorials for existing ones. Finally, `R` is highly interoperable: Packages are available to read and write nearly every filetype, including DOCX, PDF (in APA style), and HTML. Moreover, wrappers are available for many tools developed in other programming languages, and code written in other programming languages can be evaluated from `R`, including `C++`, `Fortran`, and `Python` (Allaire et al. [5]). There are excellent free resources for learning `R` (e.g., Grolemund and Wickham [14]).

Working with `R` is simplified immensely by using the RStudio integrated development engine (IDE) for `R` (RStudio Team [35]). RStudio automates and streamlines tedious or complicated aspects of working with the tools used in WORCS, and many are embedded directly into the visual user interface. This makes RStudio a comprehensive solution for open science research projects. Another important feature of RStudio is project management. A project bundles writing, analyses, data, references, et cetera, into a self-contained folder, that can be uploaded entirely to a Git remote repository and downloaded by future users. The `worcs R` package installs a new RStudio project template. When a new project is initialized from this template, the bookkeeping required to set up an open science project is performed automatically.

3.5.1. Preparing your system

Before you can use the `R` implementation of the WORCS workflow, you have to install the required software. This 30 minute procedure is documented in the setup vignette. After setting up your system, you can use `worcs` for all of your projects. Alternatively, Docker users can follow the stetup with Docker vignette. The `R` implementation of the workflow introduced earlier is detailed, step-by-step, in the workflow vignette.

4. How WORCS helps meet the TOP-guidelines

4.1. Comprehensive citation

The TOP-guidelines encourage comprehensive citation of literature, data, materials, methods, and software. In principle, researchers can meet this requirement by simply citing every reference used. Unfortunately, citation of data and software is less commonplace than citation of literature and materials. Crediting these resources is important, because it incentivizes data sharing and the development of open-source software, supports the open science efforts of others, and helps researchers receive credit for *all* of their research output, in line with the San Francisco Declaration on Research Assessment, DORA.

To facilitate citing datasets, researchers sometimes publish *data papers*; documents that detail the procedure, sample, and codebook. Specialized journals, such as the *Journal of Open Psychology Data*,

aid in the publication of these data papers. For smaller projects, researchers often simply share the data in a remote repository, along with a text file with the preferred citation for the data (which can be a substantive paper), and the license that applies to the data, such as Creative Commons BY-SA or BY-NC-SA. When in doubt, one can always contact the data creators and ask what the preferred citation is.

References for software are sometimes provided within the software environment; for example, in `R`, package references can be found by calling `citation("packagename")`. This returns an APA-style reference, and a BibTeX entry. Software papers are also common, and are sometimes called "Application Notes". The Journal of Open Source Software (JOSS) offers programmers a way to generate a publication based on well-documented software; in this process, the code itself is peer reviewed, and writing an actual paper is optional. For an example, see Rosenberg et al. [34]. Software citations can also sometimes be found in the source code repository (e.g., on Git remote repositories), using the "Citation File Format". The *Software Heritage Project* recently published a BibLaTeX software citation style that will further facilitate accurate citation of software in dynamically generated documents.

One important impediment to comprehensive citation is the fact that print journals operate with space constraints. Print journals often discourage comprehensive citation, either actively, or passively by including the reference list in the manuscript word count. Researchers can overcome this impediment by preparing two versions of the manuscript: One version with comprehensive citations for online dissemination, and another version for print, with only the essential citations. The print version should reference the online version, so interested readers can find the comprehensive reference list. The WORCS procedure suggests uploading the online version to a preprint server. This is important because most major preprint servers – including arXiv.org and all preprint services hosted by the Open Science Framework (OSF) – are indexed by Google Scholar. This means that authors will receive credit for cited work; even if they are cited only in the online version. Moreover, preprint servers ensure that the online version will have a persistent DOI, and will remain reliably accessible, just like the print version.

4.2. Implementation in worcs

The `worcs` package includes a vignette on citation to explain the fundamentals for novice users, and offer recommendations specific to `worcs`. We recommend using the free, open-source reference manager Zotero; it is feature-rich, user-friendly, and highly interoperable with other reference managers. A tutorial for using Zotero with RMarkdown exists here.

The `worcs` package also offers original solutions for comprehensive citation. Firstly, during the writing process, authors can mark the distinction between essential and non-essential references. It is easier to do this from the start, instead of going back to cut non-essential references just prior to publication. Standard Markdown uses the at-symbol (`@`) to cite a reference. `worcs` additionally reserves the "double at"-symbol (`@`) to cite a non-essential reference. Users can render the manuscript either with, or without, comprehensive citations by adapting the `knit` command in the front matter, setting it to `knit: worcs::cite_all` to render all citations, and to `knit: worcs::cite_essential` to remove all *non-essential* citations.

With regard to the citation of `R` packages, it is worth noting that the `papaja` package (Aust and Barth [7]), which provides the `APA6` template used in `worcs`, also includes functions to automatically cite all packages used in the `R` session, and add their references to the bibliography file.

4.3. Data sharing

Data sharing is important for computational reproducibility and secondary analysis. Computational reproducibility means that a third party can exactly recreate the results from the original data, using the published analysis code. Secondary analysis means that a third party can conduct sensitivity analyses, explore alternative explanations, or even use existing data to answer a different research question. If it is possible to share the data, these can simply be pushed to a Git remote repository, along with documentation (such as a codebook) and the analysis code. This way, others can download the entire repository and reproduce the analyses from start to finish. From an open science perspective, data sharing is always desirable. From a practical point of view, it is not always possible.

Data sharing may be impeded by several concerns. One technical concern is whether data require special storage provisions, as is often the case with "big data". Most Git remote repositories do not support files larger than 100 MB, but it is possible to enhance a Git repository with Git Large File Storage to add remotely stored files as large as several GB. Files exceeding this size require custom solutions beyond the scope of this paper.

Sharing human participant data additionally entails legal and ethical concerns. Before sharing any human participant data, it is recommended to obtain approval from qualified ethical and legal advisory organs, guidance from Research Data Management Support, and informed consent from participants. Many legislatures require researchers to "de-identify" human subject data upon collection, by removing or deleting any sensitive personal information and contact details. However, when researchers have an imperfect understanding of the legal obligations and dispensations, they can end up over- or undersharing, and risk either placing participants at risk of being identified, or undermine the replicability of their own work and potential reuse value of their data (Phillips and Knoppers [30]). Furthermore, it is important to note that the European GDPR prohibits storing "personal data" (information which can identify a natural person whether directly or indirectly) on a server outside the EU, unless it offers an "adequate level of protection". Although different rules may apply to pseudonimized data, there are many repositories that are GDPR compliant, such as the European servers of the Open Science Framework. Different Universities, countries, and funding bodies also have their own repositories that are complient with local legistation. In sum, data should only be shared after consult qualified legal and ethical advisory organs, obtaining informed consent from participants, and de-identifying or pseudononimizing data.

If data cannot be shared, researchers should aim to safeguard the potential for computational reproducibility and secondary analysis as much as possible. WORCS recommends two solutions to accomplish this goal. The first solution is to publish a *checksum* of the original data file (Rivest [33]). Think of a checksum as a 32-character unique identifier,[1] or as a lossy "summary" of the contents of a file. Any change to the file will result in a different checksum. Thus, one can use a checksum to verify the identity of a file, and ensure that its contents are unchanged. When data cannot be shared, the risk of fraud or abuse can be mitigated by publishing a checksum for the original data, in addition to the complete analysis code. Using the checksum to verify the identity of a private dataset, researchers can prove to an independent auditor that running the public analysis code on the private data results in the published results of their work.

A second solution is to share a synthetic dataset with similar characteristics to the real data. Synthetic data mimic the level of measurement and (conditional) distributions of the real data (see Nowok et al. [25]). Sharing synthetic data allows any third party to 1) verify that the published code works, 2) debug

[1] It is theoretically possible but improbable that random changes to a file will result in the same checksum.

the code, and 3) write valid code for alternative analyses. It is important to note that complex multivariate relationships present in the real data are often lost in synthetic data. Thus, findings from the real data might not be replicated in the synthetic data, and findings in the synthetic data *should not be substantively interpreted*. Still, sharing synthetic data facilitates data requests from third parties. A third party can write analysis code based on the synthetic data, and send it to the authors who evaluate it on the real data and send back the results. The `worcs` package offers a simple but flexible function to generate synthetic data, `synthetic()`. This function is based on the work by Nowok et al. [25] (see the `R` package `synthpop`), but with a simpler and more flexible user interface. Specifically, the `worcs` function `synthetic()` accepts any function that can be used to model marginal associations in the data. By default, this function uses random forests, as implemented in the `ranger` package, to generate a model for synthesis (Wright and Ziegler [46]). Note that the `ranger()` function does not handle all available data types in `R`. Users can call custom functions to build a prediction model for unsupported data types, using the argument `model_expression`. For further details, see the function documentation.

Note that for data synthesis to work properly, it is essential that the `class` (data type) of variables is defined correctly. As of `worcs` version 0.1.6, numeric, integer, factor, and logical data are supported out of the box. Other types of variables should be converted to one of these types. It is important to consider this when using special data types, such as the labeled vectors created by the `haven` package when reading an SPSS file. We further caution that the default random forests algorithm is computationally intensive, and may be unfeasible for use with "large" datasets. Users can provide use a custom `model_expression` and `predict_expression` to use a different algorithm when calling `synthetic()`. For example, users could use `lm()` as a `model_expression` to use linear regression, which preserves linear marginal relationships but can give rise to values out of range of the original data. Or users could call `sample()` as a `predict_expression` to bootstrap each variable, a very quick solution that maintains univariate distributions but loses all marginal relationships.

4.3.1. Processing data in WORCS

When initializing a new `worcs` project, an empty `R` script called `prepare_data.R` is generated. As soon as raw data are collected, researchers should use this file to document all steps necessary to load the data into `R`, pseudonimize it, and prepare it for analysis. As data should be shared in a format as close to raw as possible, this script should be as short as possible and only document the minimum necessary steps. For example, if the data was originally in SPSS format with IP addresses and GPS location, this file might just contain the code required to read the SPSS file, and to remove those columns of sensitive personal information. As soon as the data are pseudonimized and processed into tabular (spreadsheet-like) format, the researcher should version control some indelible record of the raw data. The concept of "tidy" data is a helpful standard; i.e., clean, tabular data that is formatted in a way that facilitates further analysis (see Wickham [44]).

One issue of concern is the practice of "fixing" mistakes in the data. It is not uncommon for researchers to perform such corrections manually in the raw data, even if all other analysis steps are documented. Regardless of whether the data will be open or closed, it is important that the raw data be left unchanged as much as possible. This eliminates human error. Any alterations to the data – including processing steps and even error corrections – should be documented in the code, instead of applied to the raw data. This way, the code will be a pipeline from the raw data to the published results. Thus, for instance, if data have been entered manually in a spreadsheet, and a researcher discovers that in one of the variables, missing values were coded as `-99` whereas they were coded as `-9` in all other variables, then we would recommend adding a line of code to recode the missing values – thereby documenting the mistake – instead of "fixing" it by editing the raw spreadsheet.

The `worcs` package offers two functions for version controlling a record of the data: One for open, and one for closed data. Researchers should assume that the decision to make data open is irreversible. Thus, the decision should be made well before arriving at this stage in the workflow. If there is any doubt, it is prudent to proceed with closed data and make the data open once all doubt is assuaged.

In `worcs`, researchers can call the function `open_data(data)` to make the data publicly available. This function stores the `data` object (such as a `data.frame` or `matrix`) in a tabular data file. It also generates an accompanying codebook – a human-readable document that serves as a legend for the tabular data file – that can be viewed online. All changed files should now be added to the Git repository, committed, and pushed to the remote repository. This can be done through the Git panel in RStudio, or by running the `worcs` function `git_update("commit message")`. Once the remote repository is made public, these data will be open to the public. Assume that, once this is done, the data cannot be un-shared.

Alternatively, if the project requires data to remain closed, researchers can call `closed_data(data)`. This function also stores the data in a local `.csv` file and generates a codebook, but the original data file is added to `.gitignore` so it cannot be accidentally added to the Git repository. The function also computes a checksum for the original data, and logs it in the `.worcs` project file. This checksum serves as the indelible record of the state of the raw data. The function `closed_data()` also calls the aforementioned `synthetic()` function to create a synthetic dataset. The user should push all changed files to the remote repository. Once the remote repository is made public, people will have access to a checksum for the original data that exist on your local machine, a codebook describing those data, and a synthetic copy of the data. Keep in mind that, when using closed data, the original data file exists only on the user's computer. Adequate provisions to back up the data, while respecting principles of responsible data stewardship, should be made.

As the purpose of the `prepare_data.R` file is to prepare data for sharing, the final line of this file should typically be either `open_data()` or `closed_data()`.

After generating tidy, shareable data, this data is then loaded in the analysis code with the function `load_data()`. This function only loads the real data if it is available on the user's computer, and otherwise, loads the synthetic data. This will have the effect that third party users who copy a remote repository with closed data will automatically load the synthetic dataset, whereas the study authors, who have the real data stored locally, will automatically load the real data. This makes it possible for reviewers, coauthors, and auditors, to write analysis scripts without requiring access to the original data. They can simply start the script with `load_data()`, and write their code based on the synthetic data. Then, they can submit their code to you – by email or as a "Pull request". When you run their code on your system, `load_data()` will load the original data, and use that to run their script. You can then return the results to the third party.

Note that these functions can be used to store multiple data files, if necessary. As of version 0.1.2, the `worcs` package only stores data as `.csv` files. As explained above, this is recommended, because `.csv` files are human- and machine readable, and because data typically need to be in tabular format for analysis. Many types of data can be represented as a table; for example, text corpora can have one row per document, and EEG- or ECG waveforms can have a row per measurement occasion and a column per channel. The `prepare_data.R` file should document any steps required to convert these data into tabular format. If this is not an appropriate format for the data, readers are encouraged to follow the development of the `repro` package (Peikert et al. [29]), which will offer greater support for reproducibly storing and retrieving diverse data formats.

Some users may intend to make their data openly available through a restricted access platform, such as an institutional repository, but not publicly through a Git remote repository. In this case, it is recommended to use `closed_data()`, and to manually upload the original `data.csv` file to the restricted access platform. If users wish to share their data through the Open Science Framework, it is sufficient to connect the OSF page to the Git remote repository as an Add-on.

4.4. Sharing code, research materials, design and analysis

When writing a manuscript created using dynamic document generation, analysis code is embedded in the prose of the paper. Thus, the TOP-guideline of sharing analysis code can be met simply by committing the source code of the manuscript to the Git repository, and making this remote repository Public. If authors additionally use open data and a reproducible environment (as suggested in WORCS), then a third party can simply replicate all analyses by copying the entire repository from the Git hosting service, and Knitting the manuscript on their local computer.

Aside from analysis code, the TOP-guidelines also encourage sharing new research materials, and details of the study design and analysis. These goals can be accomplished by placing any such documents in the Git repository folder, committing them, and pushing to a cloud hosting service. As with any document version controlled in this way, it is advisable (but not required) to use plain text only.

4.5. Preregistration

Lindsay et al. [21] define preregistration as "creating a permanent record of your study plans before you look at the data. The plan is stored in a date-stamped, uneditable file in a secure online archive." Two such archives are well-known in the social sciences: AsPredicted.org, and OSF.io. However, Git cloud hosting services also conform to these standards. Thus, it is possible to preregister a study simply by committing a preregistration document to the local Git repository, and pushing it to the remote repository. Subsequently taging the release as "Preregistration" on the remote repository renders it distinct from all other commits, and easily findable by people and programs. This approach is simple, quick, and robust. Moreover, it is compatible with formal preregistration through services such as AsPredicted.org or OSF.io: If the preregistration is written using dynamic document generation, as recommended in WORCS, this file can be rendered to PDF and uploaded as an attachment to the formal preregistration service.

The advantages, disadvantages, and pitfalls for preregistering different types of studies have been extensively debated elsewhere (see Lindsay et al. [21]). For example, because the practice of preregistration has historically been closely tied to experimental research, it has been a matter of some debate whether secondary data analyses can be preregistered (but see Weston et al. [43] for an excellent discussion of the topic).

When analyzing existing data, it is difficult to *prove* that a researcher did not have direct (or indirect, through collaborators or by reading studies using the same data) exposure to the data, before composing the preregistration. However, the question of proof is only relevant from a perspective of preventing scientific misconduct. Preregistration is a good way to ensure that a researcher is not "HARKing": Hypothesizing after the results are known (Kerr [17]).

Good faith preregistration efforts always improve the quality of deductive (theory-testing) research, because they avoid HARKing, ensure reliable significance tests, avoid overfitting noise in the data, and limit the number of forking paths researchers wander during data analysis (Gelman and Loken [12]).

WORCS takes the pragmatic position that, in deductive (hypothesis-testing) research, it is beneficial to plan projects before executing them, to preregister these plans, and adhere to them. All deviations from this procedure should be disclosed. Researchers should minimize exposure to the data, and disclose any prior exposure, whether direct (e.g., by computing summary statistics) or indirect (e.g., by reading papers using the same data). Similarly, one can disclose any deviations from the analysis plan to handle unforeseen contingencies, such as violations of model assumptions; or additional exploratory analyses.

WORCS recommends documenting a verbal, conceptual description of the study plans in a text-based file. Optionally, an analysis script can be preregistered to document the planned analyses. If any changes must be made to the analysis code after obtaining the data, one can refer to the conceptual description to justify the changes. The ideal preregistered analysis script consists of a complete analysis that can be evaluated once the data are obtained. This ideal is often unattainable, because the data present researchers with unanticipated challenges; e.g., assumptions are violated, or analyses work differently than expected. Some of these challenges can be avoided by simulating the data one expects to obtain, and writing the analysis syntax based on the simulated data. This topic is beyond the scope of the present paper, but many user-friendly methods for simulating data are available in most statistical programming languages. For instance, `R` users can use the package `simstudy` (Goldfeld [13]).

As soon as a project is preregistered on a Public Git repository, it is visible to the world, and reviewers (both formal reviewers designated by a journal, and informal reviewers recruited by other means) can submit comments, e.g., through "Pull requests" on a Git remote repository. If the remote repository is Private, Reviewers can be invited as "Collaborators". It is important to note that contributing to the repository in any way will void reviewers' anonymity, as their contributions will be linked to a user name. Private remote repositories can be made public at a later date, along with their entire time-stamped history and tagged releases.

4.5.1. Implementation of preregistration in worcs

The `worcs` workflow facilitates preregistration, primarily by explicitly inviting a user to write the preregistration document. To this end, `worcs` imports several preregistration templates from the `R` package `prereg` (Aust [6]), including templates from organizations like AsPredicted.org and OSF.io, and from researchers (e.g., van 't Veer and Giner-Sorolla [42]). When initializing an RStudio project with the `worcs` project template, one of these preregistration templates can be selected, which will generate a file called `preregistration.Rmd`. This file should be used to document study plans, ideally prior to data collection. Within the workflow, the Git cloud hosting services can be used to preregister a study simply by pushing a text document with the study plans to a Git remote repository. By tagging the commit as a preregistration (see the workflow vignette), it is distinct from all other commits, and easily findable by people and machines.

4.6. Compatibility with other standards for open science

Aside from the TOP Guidelines, the FAIR Guiding Principles for scientific data management and stewardship (Wilkinson et al. [45]) are an important standard for open science. These principles advocate that digital research objects should be Findable, Accessible, Interoperable and Reusable. Initially mainly promoted as principles for data, they are increasingly applied as a standard for other types of research output as well, most notably software (Lamprecht et al. [19]).

WORCS was not designed to comprehensively address these principles, but the workflow does facilitate meeting them. In terms of Findability, all files in GitHub projects are searchable by content and

meta-data, both on the platform and through standard search engines. The workflow further recommends that users create a project page on the Open Science Framework – a recognized repository in the social sciences. When doing so, it is important to generate a Digital Object Identifier (DOI) for this project. A DOI is a persistent way to identify and connect to an object on the internet, which is crucial for findability. Additional DOIs can be generated through the OSF for specific resources, such as data sets. It is further worth noting that, optionally, GitHub repositories can be connected to Zenodo: On Zenodo, users can store a snapshot of the repository, provide metadata, and generate a DOI for the project or specific resources. When a project is connected to the OSF, as recommended, these steps may be redundant. Finally, each worcs project has a [YAML file] (https://yaml.org/) that lists the data objects (and other meta-data) in the project, which will allow web crawlers to index worcs projects. We use this metadata, for example, to identify public `worcs` projects on GitHub.

Accessibility is primarily safeguarded by hosting all project files in a public GitHub repository, which can be downloaded as a ZIP archive, and cloned or forked using the Git protocol. GitHub is committed to long term accessibility of these resources. We further encourage users to make their data accessible when possible. When data must be closed, the `worcs` function `closed_data()` adds a paragraph to the project readme, indicating how readers can contact the author for access.

With regard to interoperability, the workflow recommends using plain-text files whenever possible, which ensures that code and meta-data are human- and machine-readable. Furthermore, the `worcs` package is implemented in R, which is open source. To promote reusability, we recommend users to specify an appropriate license for their projects. The `worcs` package includes the Creative Commons (CC) licenses, and suggests a CC-BY 4.0 license by default. This license is featured in the GitHub repository, and we encourage users to also display it on their project's OSF page. Note that it is possible to license specific project resources separately under different licenses. For instance, whereas the CC-licenses are suitable for data, media, and academic papers – they are ill-suited for licensing software. It is thus prudent to consider separately licensing specific project resources, particularly when a project contains *original* source code (i.e., you write your own functions). For a more extensive discussion of appropriate licensing, see Peikert and Brandmaier [28], Stodden et al. [38], and the website choosealicense.com.

To further facilitate reusability, `worcs` automatically generates a codebook for each data file. We encourage users to elaborate on the default codebook by adding variable description and categories, which would – in the future – enable indexing data by topic area.

Incidentally, the FAIR principles have been applied to software as well, and `worcs` follows these recommendations for FAIR research software. It is hosted on a version controlled public remote repository GitHub, has an open source licence (GPL v3.0), is registered in a community registry (CRAN), enables the citation of the software (using the `citation("worcs")` command, or by citing this paper), and followed the *CII Best Practices* software quality checklist during development.

Sharing all research objects, as advocated in WORCS, also provides a thorough basis for research evaluation according to the San Francisco Declaration on Research Assessment (DORA; https://sfdora.org/), which plays an increasing role in grant funding, hiring, and promotion procedures. Direct access to research objects allows stakeholders to evaluate research quality based on content rather than relying on spurious surrogate indicators like journal impact factors, conference rankings, and h-indexes. The detailed version control and commit tracking of Git remote repositories furthermore make it possible to assess the relative contributions made by different researchers.

5. Discussion

In this tutorial paper, we have presented a workflow for open reproducible code in science. The workflow aims to lower the threshold for grass-roots adoption of open science principles. The workflow is supported by an `R` package with an RStudio project template and convenience functions. This relatively light-weight workflow meets most of the requirements for open science as detailed in the TOP-guidelines, and is compatible with other open science guidelines. The workflow helps researchers meet existing requirements for open science, and to reap the benefits of working openly and reproducibly, even when such requirements are absent.

5.1. Comparing WORCS to existing solutions

There have been several previous efforts to promote grass-roots adoption of open science principles. Each of these efforts has a different scope, strengths, and limitations that set it apart from WORCS. For example, there are "signalling solutions"; guidelines to structure and incentivize disclosure about open science practices. Specifically, Aalbersberg et al. [1] suggested publishing a "TOP-statement" as supplemental material, which discloses the authors' adherence to open science principles. Relatedly, Aczel et al. [2] developed a consensus-based Transparency Checklist that authors can complete online to generate a report. Such signalling solutions are very easy to adopt, and they address TOP-guidelines 1–7. Many journals now also offer authors the opportunity to earn "badges" for adhering to open science guidelines (Kidwell et al. [18]). These signalling solutions help structure authors' disclosures about, and incentivize adherence to, open science practices. WORCS, too, has its own checklist that calls attention to a number of concrete items contributing to an open and reproducible research project. Users of the `worcs` package can receive a badge, displayed on the `README.md` file, based on a semi-automated scoring of the checklist items, by calling the function `check_worcs()` within a `worcs` project directory.

A different class of solutions instead focuses on the practical issue of *how* researchers can meet the requirements of open science. A notable example is the workflow for reproducible analyses developed by Peikert and Brandmaier [28], which uses Docker to ensure strict computational reproducibility for even the most sophisticated analyses. There are some decisive differences with WORCS. First, concerning the scope of the workflow: WORCS is designed to address a unique issue not covered by other existing solutions, namely, to provide a workflow most conducive to satisfying TOP-guidelines 1–7 while adhering to the FAIR principles. Peikert and Brandmaier instead focus primarily on computational reproducibility, which is relevant for TOP-guidelines 2, 3, and 5. Second, with regard to ease of use, WORCS aims to bring down the learning curve by adopting sensible defaults for any decisions to be made. Peikert and Brandmaier, by contrast, designed their workflow to be very flexible and comprehensive, thus requiring substantially more background knowledge and technical sophistication from users. To sum up, WORCS builds upon the same general principles as Peikert and Brandmaier, and the two workflows are compatible. What sets WORCS apart is that it is more lightweight in terms of system and user requirements, thereby facilitating adoption, but still ensures computational reproducibility under *most circumstances*.

One initiative that WORCS is fully compatible with, is the *Scientific Paper of the Future*. This organization encourages geoscientists to document data provenance and availability in public repositories, document software used, and document the analysis steps taken to derive the results. All of these goals could be met using WORCS.

5.2. Limitations

WORCS is intended to substantially reduce the threshold for adopting best practices in open and reproducible research. However, several limitations and issues for future development remain. One potential challenge is the learning curve associated with the tools outlined in this paper. Learning to work with `R`, RMarkdown, and Git takes time – although tutorials such as this one substantially reduce the learning curve. Moreover, the time investment tends to pay off. Working with `R` opens the door to many cutting edge analysis techniques. Working with RMarkdown saves time and prevents mistakes by avoiding tedious copying of results into a text document. Working with Git keeps projects organized, prevents accidental loss of work, enables integrating changes by collaborators in a non-destructive manner, and ensures that entire research projects are archived and can be accessed or copied by third parties. Thus, the time investment is eminently worthwhile.

Another limitation is the fact that no single workflow can suit all research projects. Not all projects conform to the steps outlined here; some may skip steps, add steps, or follow similar steps in a different order. In principle, WORCS does not enforce a linear order, and allows extentensive user flexibility. A related concern is that some projects might require specialized solutions outside of the scope of this paper. For example, some projects might use very large data files, rely on data that reside on a protected server, or use proprietary software dependencies that cannot be version controlled using `renv` or Docker. In such cases, the workflow can serve as a starting point for a custom approach; offering a primer on best practices, and a discussion of relevant factors to consider. We urge users to inform us of such challenges and custom solutions, so that we may add them to the overview of WORCS-projects on GitHub, where they can serve as an example to others.

Another important challenge is managing collaborations when only the lead author uses RMarkdown, and the coauthors use Word. When using the `papaja` package to write APA-style papers, it is possible to Knit the manuscript to Word (.docx), by changing the line `output: papaja::apa6_pdf` in the manuscript's YAML header to `output: papaja::apa6_docx`. There are some limitations to the conversion, discussed here. When soliciting feedback from co-authors, ask them to use Track Changes and comment bubbles in Word. Then, manually incorporate their changes into the `manuscript.Rmd` file. In most cases, this is the most user-friendly approach, as most lead authors would review changes by co-authors anyway. A second approach is to ask collaborators to work in plain text. In this case, send collaborators the `manuscript.Rmd` file, and ask them to open (and save) it in Word or Notepad as a plain text file. When they send it back, make sure any changes to your local file are committed, and then simply overwrite your local version with their file. In RStudio, select the file in the Git tab, and click the Diff button to examine what changes the collaborator has made relative to your last committed version.

If all collaborators are committed to using WORCS, they can Fork the repository from the lead author on GitHub, clone it to their local device, make their own changes, and send a pull request to incorporate their changes. Working this way is extremely conducive to scientific collaboration (Ram [32]). Recall that, when using Git for collaborative writing, it is recommended to insert a line break after every sentence so that the change log will indicate which specific sentence was edited, and to prevent "merge conflicts" when two authors edit the same line. The resulting document will be rendered to PDF without spurious line breaks.

Being familiar with Git remote repositories opens doors to new forms of collaboration: In the open source software community, continuous peer review and voluntary collaborative acts by strangers who are interested in a project are commonplace (see Adolph et al. [3]). This kind of collaboration is budding in scientific software development as well; for example, the lead author of this paper became a co-author

on several R packages after submitting pull requests with bug fixes or additional functionality (Hallquist et al. [15]; Rosenberg et al. [34]), and two co-authors of this paper became involved by contributing pull requests to worcs. It is also possible to invite such collaboration by opening Issues for tasks that still need to be accomplished, and tag known collaborators to address them, or invite external collaborators to contribute their expertise.

A final limitation is that, as WORCS is relatively new, the number of current users is still limited. We therefore do not have elaborate insights in user experiences or feedback. We strive to make the uptake of worcs as easy as possible, by means of this background paper, extensive documentation and vignettes for the R implementation, online video tutorials, webinars and exemplar use cases. We encourage our users to provide us with feedback, feature requests, or bug reports through the channels indicated here.

5.3. Future developments

WORCS provides a user-friendly and lightweight workflow for open, reproducible research, that meets all TOP-guidelines pertaining to reproducibility (1–7). Nevertheless, there are clear directions for future developments. Firstly, although the workflow is currently implemented only in R, it is conceptually relevant for users of other statistical programming languages, and we welcome efforts to implement WORCS in other platforms. We welcome efforts to implement WORCS in other platforms. Secondly, even when a project is in R, it may have dependencies outside of the R environment that cannot be managed using renv. It is beyond the scope of the worcs package to support tools outside of R, or to containerize a project so that it can be identically reinstated on a different system or virtual machine. The forthcoming package repro (Peikert et al. [29]) will offer such extensions of the workflow, thus combining the strengths of WORCS with the solutions proposed by Peikert and Brandmaier [28].

5.4. Conclusion

WORCS offers a workflow for open reproducible code in science. The step-by-step procedure outlined in this tutorial helps researchers make an entire research project Open and Reproducible. The accompanying R package provides user-friendly support functions for several steps in the workflow, and an RStudio project template to get the project set up correctly.

WORCS encourages and simplifies the adoption of open science principles in daily scientific work. It helps researchers make all research output created throughout the scientific process – not just manuscripts, but data, code, and methods – open and publicly assessible. This enables other researchers to reproduce results, and facilitates cumulative science by allowing others to make direct use of these research objects.

Acknowledgements

The lead author is supported by a NWO Veni grant (NWO grant number VI.Veni.191G.090). We acknowledge Jeroen Ooms for offering feedback and suggesting the use of the R package gert.

References

[1] I.J. Aalbersberg, T. Appleyard, S. Brookhart, T. Carpenter, M. Clarke, S. Curry et al., Making science transparent by default; introducing the TOP statement, 2018. doi:10.31219/osf.io/sm78t.

[2] B. Aczel, B. Szaszi, A. Sarafoglou, Z. Kekecs, Š. Kucharský, D. Benjamin et al., A consensus-based transparency checklist, *Nature Human Behaviour* (2019), 1–3. doi:10.1038/s41562-019-0772-6.
[3] K.E. Adolph, R.O. Gilmore, C. Freeman, P. Sanderson and D. Millman, Toward open behavioral science, *Psychological Inquiry* **23**(3) (2012), 244–247. doi:10.1080/1047840X.2012.705133.
[4] J. Allaire, Y. Xie, R Foundation, H. Wickham, Journal of Statistical Software R. Vaidyanathan et al.Rticles: Article formats for r markdown 2020, Retrieved from https://CRAN.R-project.org/package=rticles.
[5] J.J. Allaire, K. Ushey, RStudio and Y. Tang, R Markdown Python engine, 2020, Retrieved January 13, 2020, from https://rstudio.github.io/reticulate/articles/r_markdown.html.
[6] F. Aust, Prereg: R Markdown templates to preregister scientific studies (version 0.4.0), 2019, Retrieved from https://CRAN.R-project.org/package=prereg.
[7] F. Aust and M. Barth, Papaja: Prepare reproducible APA journal articles with R Markdown, 2020, (Version 0.1.0.9842), (Original work published 2014), Retrieved from https://github.com/crsh/papaja.
[8] S. Bezjak, A. Clyburne-Sherin, P. Conzett, P. Fernandes, E. Görögh, K. Helbig et al., Open science training handbook (Version 1.0). Zenodo, 2018. doi:10.5281/zenodo.1212496.
[9] J.D. Blischak, E.R. Davenport and G. Wilson, A quick introduction to version control with Git and GitHub, *PLOS Computational Biology* **12**(1) (2016), e1004668. doi:10.1371/journal.pcbi.1004668.
[10] C.T. Brown, How I learned to stop worrying and love the coming archivability crisis in scientific software, 2017, Retrieved January 13, 2020, from http://ivory.idyll.org/blog/2017-pof-software-archivability.html.
[11] J.C. Coyne, Replication initiatives will not salvage the trustworthiness of psychology, *BMC Psychology* **4**(1) (2016), 28. doi:10.1186/s40359-016-0134-3.
[12] A. Gelman and E. Loken, The statistical crisis in science: Data-dependent analysis – a "Garden of forking paths" – explains why many statistically significant comparisons don't hold up, *American Scientist* **102**(6) (2014), 460–466, Retrieved from https://go.gale.com/ps/i.do?p=AONE&sw=w&issn=00030996&v=2.1&it=r&id=GALE%7CA389260653&sid=googleScholar&linkaccess=abs. doi:10.1511/2014.111.460.
[13] K. Goldfeld, Simstudy: Simulation of study data, 2020, Retrieved from https://CRAN.R-project.org/package=simstudy.
[14] G. Grolemund and H. Wickham, *R for Data Science*, O'Reilly, 2017, Retrieved from https://r4ds.had.co.nz/.
[15] M. Hallquist, J. Wiley and C.J. Van Lissa, MplusAutomation: An R package for facilitating large-scale latent variable analyses in Mplus (Version 0.7-3), 2018, Retrieved from https://CRAN.R-project.org/package=MplusAutomation.
[16] L.K. John, G. Loewenstein and D. Prelec, Measuring the prevalence of questionable research practices with incentives for truth telling, *Psychological Science* **23**(5) (2012), 524–532. doi:10.1177/0956797611430953.
[17] N.L. Kerr, HARKing: Hypothesizing after the results are known, *Personality and Social Psychology Review: An Official Journal of the Society for Personality and Social Psychology, Inc* **2**(3) (1998), 196–217. doi:10.1207/s15327957pspr0203_4.
[18] M.C. Kidwell, L.B. Lazarević, E. Baranski, T.E. Hardwicke, S. Piechowski, L.-S. Falkenberg and B.A. Nosek, Badges to acknowledge open practices: A simple, low-cost, effective method for increasing transparency, *PLOS Biology* **14**(5) (2016), e1002456. doi:10.1371/journal.pbio.1002456.
[19] A.-L. Lamprecht, L. Garcia, M. Kuzak, C. Martinez, R. Arcila, E. Martin Del Pico et al., Towards FAIR principles for research software, *Data Science* (2019), 1–23. doi:10.3233/DS-190026.
[20] W.J.M. Levelt, E. Noort and P.J.D. Drenth, Failing science: The fraudulent research practices of social psychologist Diederik Stapel (Falende wetenschap: De frauduleuze onderzoekspraktijken van sociaal-psycholoog Diederik Stapel), 2012, Retrieved from https://www.onderwijsbrabant.nl/sites/default/files/eindrapport_stapel_nov_2012.pdf.
[21] D.S. Lindsay, D.J. Simons and S.O. Lilienfeld, Research preregistration 101, *APS Observer* **29**(10) (2016), Retrieved from https://www.psychologicalscience.org/observer/research-preregistration-101.
[22] R.A. Muenchen, The popularity of data science software, 2012, April 25, Retrieved January 8, 2020, from http://r4stats.com/articles/popularity/.
[23] B.A. Nosek, G. Alter, G.C. Banks, D. Borsboom, S.D. Bowman, S.J. Breckler et al., Promoting an open research culture, *Science* **348**(6242) (2015), 1422–1425. doi:10.1126/science.aab2374.
[24] B.A. Nosek and Y. Bar-Anan, Scientific utopia: I. Opening scientific communication, *Psychological Inquiry* **23**(3) (2012), 217–243. doi:10.1080/1047840X.2012.692215.
[25] B. Nowok, G.M. Raab and C. Dibben, Synthpop: Bespoke creation of synthetic data in R, *Journal of Statistical Software* **74**(1,1) (2016), 1–26. doi:10.18637/jss.v074.i11.
[26] J. Ooms, Gert: Simple Git client for r, 2019, Retrieved from https://CRAN.R-project.org/package=gert.
[27] P. Patil, R.D. Peng and J.T. Leek, A visual tool for defining reproducibility and replicability, *Nature Human Behaviour* **3**(7,7) (2019), 650–652. doi:10.1038/s41562-019-0629-z.
[28] A. Peikert and A.M. Brandmaier, A reproducible data analysis workflow with R Markdown, Git, Make, and Docker. doi:10.31234/osf.io/8xzqy.
[29] A. Peikert, A.M. Brandmaier and C.J. Van Lissa, Repro: Automated setup of reproducible workflows and their dependencies, 2020, Retrieved from https://github.com/aaronpeikert/repro.

[30] M. Phillips and B.M. Knoppers, The discombobulation of de-identification, *Nature Biotechnology* **34**(11,11) (2016), 1102–1103. doi:10.1038/nbt.3696.
[31] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2020. Retrieved from https://www.R-project.org/.
[32] K. Ram, Git can facilitate greater reproducibility and increased transparency in science, *Source Code for Biology and Medicine* **8**(1) (2013), 7. doi:10.1186/1751-0473-8-7.
[33] R. Rivest, The MD5 message-digest algorithm, MIT Laboratory for Computer Science and RSA Data Security, Inc., 1992. Available at: https://doi.org/10.17487/RFC1321.
[34] J. Rosenberg, P. Beymer, D. Anderson, C.J. Van Lissa and J. Schmidt, tidyLPA: An R package to easily carry out latent profile analysis (LPA) using open-source or commercial software, *Journal of Open Source Software* **3**(30) (2018), 978. doi:10.21105/joss.00978.
[35] RStudio Team, *RStudio: Integrated Development Environment for R*, RStudio, Inc., Boston, MA, 2015. Retrieved from http://www.rstudio.com/.
[36] P.E. Shrout and J.L. Rodgers, Psychology, science, and knowledge construction: Broadening perspectives from the replication crisis, *Annual Review of Psychology* **69**(1) (2018), 487–510. doi:10.1146/annurev-psych-122216-011845.
[37] J. Sondervan, J. Bosman, B. Kramer, L. Brinkman, M. Imming and A. Versteeg, The COVID-19 pandemic stresses the societal importance of open science, *ScienceGuide*, 2020, April 3, Retrieved from https://www.scienceguide.nl/2020/04/dire-times-of-covid-19-stress-the-societal-importance-of-open-science/.
[38] V. Stodden, M. McNutt, D.H. Bailey, E. Deelman, Y. Gil, B. Hanson et al., Enhancing reproducibility for computational methods, *Science* **354**(6317) (2016), 1240–1241. doi:10.1126/science.aah6168.
[39] J. Tennant, Open science is just good science, 2018, TU Delft, Retrieved from https://figshare.com/articles/Open_Science_is_just_good_science_pptx/5783004.
[40] K. Ushey, 2020, Renv: Project environments (Version 0.12.0). Retrieved from https://CRAN.R-project.org/package=renv.
[41] C.J. Van Lissa, A. Peikert and A.M. Brandmaier, Worcs: Workflow for open reproducible code in science (Version 0.1.5), 2020, Retrieved from https://cran.r-project.org/web/packages/worcs/index.html.
[42] A.E. van 't Veer and R. Giner-Sorolla, Pre-registration in social psychology – a discussion and suggested template, *Journal of Experimental Social Psychology* **67** (2016), 2–12. doi:10.1016/j.jesp.2016.03.004.
[43] S.J. Weston, S.J. Ritchie, J.M. Rohrer and A.K. Przybylski, Recommendations for increasing the transparency of analysis of preexisting data sets, *Advances in Methods and Practices in Psychological Science* (2019). doi:10.1177/2515245919848684.
[44] H. Wickham, Tidy data, *Journal of Statistical Software* **59**(1,1) (2014), 1–23. doi:10.18637/jss.v059.i10.
[45] M.D. Wilkinson, M. Dumontier, I.J. Aalbersberg, G. Appleton, M. Axton, A. Baak et al., The FAIR guiding principles for scientific data management and stewardship, *Scientific Data* **3**(1) (2016), 160018. doi:10.1038/sdata.2016.18.
[46] M.N. Wright and A. Ziegler, Ranger: A fast implementation of random forests for high dimensional data in C++ and R, 2015, August 18, Retrieved from http://arxiv.org/abs/1508.04409.
[47] Y. Xie, J.J. Allaire and G. Grolemund, *R Markdown: The Definitive Guide*, Chapman and Hall/CRC, 2018. Retrieved from https://bookdown.org/yihui/rmarkdown/.