Making simulation results reproducible - Survey, guidelines, and examples based on Gradle and Docker (#37926)

First revision

Guidance from your Editor

Please submit by **24 Aug 2019** for the benefit of the authors (and your $200 publishing discount).

Structure and Criteria
Please read the 'Structure and Criteria' page for general guidance.

Author notes
Have you read the author notes on the guidance page?

Raw data check
Review the raw data. Download from the location described by the author.

Image check
Check that figures and images have not been inappropriately manipulated.

Privacy reminder: If uploading an annotated PDF, remove identifiable information to remain anonymous.

Files

Download and review all files from the materials page.

1 Tracked changes manuscript(s)
1 Rebuttal letter(s)
8 Figure file(s)
1 Latex file(s)
1 Table file(s)
1 Other file(s)

Structure and Criteria

2

Structure your review

The review form is divided into 5 sections. Please consider these when composing your review:

1. **BASIC REPORTING**
2. **EXPERIMENTAL DESIGN**
3. **VALIDITY OF THE FINDINGS**
4. General comments
5. Confidential notes to the editor

You can also annotate this PDF and upload it as part of your review

When ready submit online.

Editorial Criteria

Use these criteria points to structure your review. The full detailed editorial criteria is on your guidance page.

BASIC REPORTING

- ✓ Clear, unambiguous, professional English language used throughout.
- ✓ Intro & background to show context. Literature well referenced & relevant.
- ✓ Structure conforms to PeerJ standards, discipline norm, or improved for clarity.
- ✓ Figures are relevant, high quality, well labelled & described.
- ✓ Raw data supplied (see PeerJ policy).

EXPERIMENTAL DESIGN

- ✓ Original primary research within Scope of the journal.
- ✓ Research question well defined, relevant & meaningful. It is stated how the research fills an identified knowledge gap.
- ✓ Rigorous investigation performed to a high technical & ethical standard.
- ✓ Methods described with sufficient detail & information to replicate.

VALIDITY OF THE FINDINGS

- ℹ Impact and novelty not assessed. Negative/inconclusive results accepted. *Meaningful* replication encouraged where rationale & benefit to literature is clearly stated.
- ✓ All underlying data have been provided; they are robust, statistically sound, & controlled.
- ✓ Speculation is welcome, but should be identified as such.
- ✓ Conclusions are well stated, linked to original research question & limited to supporting results.

Standout reviewing tips

The best reviewers use these techniques

Tip	*Example*
Support criticisms with evidence from the text or from other sources	*Smith et al (J of Methodology, 2005, V3, pp 123) have shown that the analysis you use in Lines 241-250 is not the most appropriate for this situation. Please explain why you used this method.*
Give specific suggestions on how to improve the manuscript	*Your introduction needs more detail. I suggest that you improve the description at lines 57- 86 to provide more justification for your study (specifically, you should expand upon the knowledge gap being filled).*
Comment on language and grammar issues	*The English language should be improved to ensure that an international audience can clearly understand your text. Some examples where the language could be improved include lines 23, 77, 121, 128 – the current phrasing makes comprehension difficult.*
Organize by importance of the issues, and number your points	*1. Your most important issue* *2. The next most important item* *3. …* *4. The least important points*
Please provide constructive criticism, and avoid personal opinions	*I thank you for providing the raw data, however your supplemental files need more descriptive metadata identifiers to be useful to future readers. Although your results are compelling, the data analysis should be improved in the following ways: AA, BB, CC*
Comment on strengths (as well as weaknesses) of the manuscript	*I commend the authors for their extensive data set, compiled over many years of detailed fieldwork. In addition, the manuscript is clearly written in professional, unambiguous language. If there is a weakness, it is in the statistical analysis (as I have noted above) which should be improved upon before Acceptance.*

Making simulation results reproducible - Survey, guidelines, and examples based on Gradle and Docker

Wilfried Elmenreich Corresp., [1] , **Philipp Moll** [1] , **Sebastian Theuermann** [1] , **Mathias Lux** [1]

[1] Universität Klagenfurt, Klagenfurt, Austria

Corresponding Author: Wilfried Elmenreich
Email address: wilfried.elmenreich@aau.at

This paper addresses two research questions related to reproducibility within the context of research related to computer science. First a survey on reproducibility addressed to researchers in the academic and private sectors is described and evaluated. The survey indicates a strong need for open and easily accessible results, in particular, reproducing an experiment should not require too much effort. The results of the survey are then used to formulate guidelines for making research results reproducible. In addition, this paper explores four approaches based on software tools that could bring forward reproducibility in research results. After a general analysis of tools, three examples are further investigated based on actual research projects which are used to evaluate previously introduced tools. Results indicate that the evaluated tools contribute well to making simulation results reproducible but due to conflicting requirements, none of the presented solutions fulfills all intended goals perfectly.

Making Simulation Results Reproducible – Survey, Guidelines, and Examples based on Gradle and Docker

Wilfried Elmenreich[1]**, Philipp Moll**[1]**, Sebastian Theuermann**[1]**, and Mathias Lux**[1]

[1]**Universität Klagenfurt, Austria**

Corresponding author:
Wilfried Elmenreich[1]

Email address: wilfried.elmenreich@aau.at

ABSTRACT

This paper addresses two research questions related to reproducibility within the context of research related to computer science. First a survey on reproducibility addressed to researchers in the academic and private sectors is described and evaluated. The survey indicates a strong need for open and easily accessible results, in particular, reproducing an experiment should not require too much effort. The results of the survey are then used to formulate guidelines for making research results reproducible. In addition, this paper explores four approaches based on software tools that could bring forward reproducibility in research results. After a general analysis of tools, three examples are further investigated based on actual research projects which are used to evaluate previously introduced tools. Results indicate that the evaluated tools contribute well to making simulation results reproducible but due to conflicting requirements, none of the presented solutions fulfills all intended goals perfectly.

1 INTRODUCTION

Reproducibility of experimental results is fundamental in all scientific disciplines. Reproducing results of published experiments, however, is often a cumbersome and unrewarding task. Casadevall and Fang (2010) report that some fields, for example biology, are concerned with complex and chaotic systems which are difficult to reproduce. At the same time, we would expect digital software-based experiments to be easily reproducible, because digital data can be easily provided and computer algorithms on these data are typically well-described and deterministic. However, this is often not the case due to a lack of disclosure of relevant software and data that would be necessary to reproduce results. Ongoing open science initiatives aim to have researchers provide access to data and software together with their publications in order to allow reviewers to make well-informed decisions and to provide other researchers with the information and necessary means to build upon and extend original research (Ram (2013)).

This paper addresses two research questions related to reproducibility. The first research question: “*To what extent is reproducibility of results based on software artifacts important in the field of computer science and in related research areas?*” with the aspects of relevance to a researcher’s field, willingness to contribute to make one’s own work reproducible, and possible concerns. With a focus on the disciplines computer science, computer engineering and electrical engineering the current practice, subject awareness, and possible concerns have been assessed by using an online survey, which addressed researchers at different positions in universities, research institutions, and companies. The second research question is “*What tools can be used to support reproducibility?*”~~.~~ To answer this question, we present three examples where three different types of software projects are packaged to provide an accurate and easy possibility for reproducing results in a controlled environment and analyze how these solutions address the requirements derived from the survey.

The responses to our online survey confirm our initial assumption that reproducibility of research results is an important concern in computer science research. One of the researchers’ main reasons for publishing software artifacts along with scientific publications is improved credibility and reliability of results. Based on the survey's results presented in Section 3, we infer requirements and general guidelines assisting researchers in making their research reproducible in Section 4. Finally, we discuss how different tools comply with the created requirements and guidelines. We find that due to conflicting requirements, none of the presented solutions fulfills all intended goals perfectly. One of the most pressing challenges is achieving long-term availability of results while respecting licensing issues of required third-party dependencies. An in-depth discussion of open issues is elaborated in Section 7 and we conclude the paper and highlight our major findings in Section 8.

2 RELATED WORK

Walters (2013) notes that *it is often difficult to reproduce the work described in molecular modeling and chemoinformatics papers*. For the most part, this is due to the absence of a disclosure requirement in many scientific publication venues thus far. Morin et al. (2012) report that in 2010 only three of the 20 most cited journals had editorial policies requiring availability of source code after publication. Fortunately, this situation is changing for the better, for example *Science* introduced a policy requiring authors to make data and code related to their publication available whenever possible (Witten and Tibshirani (2013); Peng (2011); Hanson et al. (2011)). Commenting on this policy, Shrader-Frechette and Oreskes (2011) brought up the issue that although privately funded science may be of high quality, it is not subject to the same requirements for transparency as publicly funded science. Another obstacle is the use of closed-source tools and undisclosed software results in publicly funded research software development projects as discussed by Morin et al. (2012). Vitek and Kalibera (2011) address the topic of repeatability and reproducibility for systems research and identify a particular difficulty for embedded systems due to companies being reluctant to release code and that implementations are often bound to specific hardware.

Focusing on the current state of reproducibility, ACM SIGCOMM Computer Communication Review (CCR) conducted a survey on reproducibility with 77 responses from authors of papers published in CCR and the SIGCOMM sponsored conferences (Bonaventure (2017)). The responses showed that there is a good awareness of the need for reproducibility and a majority of authors either considered their paper self-contained or have released the software used to perform experiments. However, there were only few releases of experimental data or of modifications of existing open source software. An The open question part of the survey indicated a need for encouragement for publishing reproducible work or for papers that attempt to reproduce or refute earlier results.

Flittner et al. (2018) conducted an analysis of papers from four different ACM conferences held in 2017. This study found that the type of artifacts can differ significantly between different communities. The analysis further indicates that even if researchers state that their work is reproducible, the majority of analyzed papers do not provide the complete toolset to reproduce all results. Most importantly, the study shows that published artifacts are indeed reused, which is why the authors encourage others to release artifacts.

A critical aspect when releasing artifacts is to decide on tools supporting researchers in the process of making research reproducible. Several papers report on case studies for data repositories in the context of reproducibility including fields such as geographic information systems (Steiniger and Hunter (2013)), astrophysics (Allen and Schmidt (2015)), microbiome census (McMurdie and Holmes (2013)), and neuroimaging (Poline et al. (2012)). These examples are promising, but it cannot be expected that the approaches are going to be used beyond the field they have been introduced. Simflowny (Arbona et al. (2013)) is a platform for formalizing and managing the main elements of a simulation flow, tied not to a field, but to a specified simulation architecture. The Whole Tale approach (Brinckman et al. (2018)) aims at linking data, code, and digital scholarly objects to publications and integrating all parts of the research story. Other works focus on code and data management, such as Ram (2013) suggesting very general version control systems such as Git for transparent data management in order to enable reproducibility of scientific work. The CARE approach (Janin et al. (2014)) extends the archiving concept with an execution system for Linux systems, which also takes software installation and dependencies into account. Docker (Boettiger (2015)), which will be examined more closely in this paper, provides an even more generic approach by utilizing virtualization for providing cross-platform portability. A tutorial for using Docker to improve reproducibility in software and web engineering research was published in Cito et al. (2016). Reprozip by Chirigati et al. (2013) provided a packing and unpacking mechanism for Linux systems allowing the creation of a package from a computer experiment which can be unpacked on another target machine, including support for unpacking into a Docker image. In contrast to the work presented above, our work focuses on the researchers' requirements regarding reproducibility independent of the capabilities of individual tools. Based on survey responses, we infer requirements and guidelines for making research reproducible and further analyze how different tools for packaging software artifacts comply with the researchers' needs. We further identify limitations of current tools and raise awareness of researchers on the pros and cons of using different types of applications for making research reproducible.

3 SURVEY

In computer science, a large amount of research is backed up by prototypes, implementations of algorithms, benchmarking and evaluation tools, and data generated in the course of research. A critical factor for cutting edge research is to be able to build upon the results of other researchers or groups by extending their ideas, applying them to new domains or by reflecting them from a new angle. This is easily done with scientific publications, which are mostly available online. While the hypotheses, findings, models, processes and equations are published, the data generated and the tools used for generating data and evaluating new approaches are sometimes only pointed out, but have to be found elsewhere.

Our hypothesis in that direction is that there is a gap between scientific publishing on one hand and the publication of software artifacts and data for making results reproducible for other researchers on the other. In that sense,we created a survey asking researchers in the computer science field for their approach and opinion.

3.1 Methodology

The survey design is driven by our first research question "*To what extent is reproducibility of results based on software artifacts important in the field of computer science and in related research areas?*"~~.~~ The survey consists of five parts. First, basic demographic information, including the type of research, the area of research, the typical research contribution, and the type of organization the researchers are working for, is collected. Second, the common practice of the researchers for publishing software artifacts and data is surveyed. Third, we focus on the researchers' expectations regarding the procedure of reproducing scientific results. Fourth, we ask for opinions on integrating the question of reproducibility in the peer review process. Finally, we collect additional thoughts with open questions.

Five-point Likert scales are used to indicate the level of agreement in the survey. For questions where Likert scales are not applicable, single-choice or multiple-choice questions (e.g.:, "*What are the typical results of your research work?*"), or numerical inputs without predefined range or categories (e.g.,"*How much time (in hours) are you willing to invest to make the results of a paper reproducible?*") are used. For single-choice and multiple-choice questions,we discussed the nominal scales based on related work as well as the authors' experience. Pretests with people neither involved with the questionnaire nor taking part in the final survey were conducted to reduce the chance of leaving out important options. For the sake of completeness custom values are allowed in addition to the given options, to allow researchers to report on their practice. Open-ended questions are only used where other types of questions might limit the spectrum of answers.

The survey was set up as an anonymous online survey, with no partial answers allowed as all questions were mandatory and only the final submission at the end of the survey would save the results. The survey was distributed via a scientific mailing lists and via personal contacts with the request to distribute the survey among colleagues[1]. The full survey and all responses are included in the supplementary material of the paper.

3.2 Demographics

In total, we received 125 responses, mostly from academic researchers. 74 out of the 125 participants were working or studying at a university and 35 of 125 of research institutes. 13 participants noted that they were mainly working for a company, two were private researchers, one from a school. With their position, 30% of the participants were PhD students, 28% were professors or group leaders, 17% worked as researchers within a project, 12% were principal investigators, and 9% were bachelor or master students at the time of the study. Three participants were heads of departments or organizations, and two participants indicated that they were postdoctoral researchers. Computer science or computer engineering was the

[1] The online survey was distributed on the following channels: Information-Centric Networking research group discussion list (`https://www.irtf.org/mailman/listinfo/icnrg`); the Google Group *comp.simulation* (`https://groups.google.com/forum/#!forum/comp.simulation`); the authors' Facebook and Twitter profiles; and via personal contacts.

area of research for 72% of the participants. 7.2% of the participants came from electrical engineering, 4% from information systems, 3.3% from (applied) mathematics, and 1.6% from simulation. Furthermore, singular mentions were applied informatics, ciencias sociales, computational biology, computational biology/numerical simulations, computer networks, data analysis, economics, management, materials science, mathematical modeling, medical informatics, physics, scientific computing, and user experience. The population also includes researcher for whom publishing software is common practice; 28% of the participants have indicated that they have not published any software artifact at the time of the study.

3.3 What Researchers Want

Four aspects of the survey responses were analyzed. First, the relevance of reproducibility for the research community is analyzed. Second, we investigate what people are willing to do in order to achieve reproducible research. Third, we discuss the researchers' opinions on reproducibility in the peer review process. Finally, we highlight concerns regarding sharing scientific software artifacts.

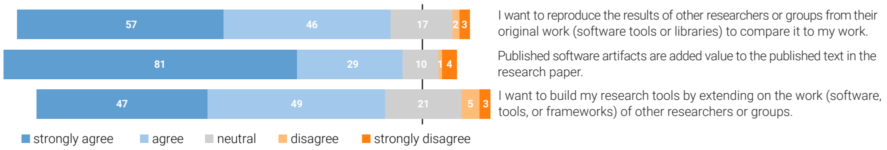


Figure 1. Responses to questions focusing on the general relevance of reproducibility.

Fig. 1 summarizes the responses to questions showing the relevance of reproducibility in research. As can be seen, the majority of people wants to reproduce results from other researchers or groups: 103 of 125 indicated agreement. Even more (110 out of 125) considered reproducible results as added value for research papers. It can be seen that the majority of researchers (96 out of 125) wants to build their research on the work of others, which requires others to share scientific artifacts.

It can be seen from the researchers' demographics in Fig. 2 that the relevance of reproducibility is independent of a researcher's position, research area, and research environment. The results of the question "*I want to reproduce the results of other researchers or groups from their original work (software tools or libraries) to compare it to my work*" were grouped by position, research area, and research environment. These distributions look very similar for all questions from Fig. 1. A full collection of graphical illustrations of these distributions is included in the supplementary material of the paper.

An open-ended question asking why software artifacts should be published yielded diverse answers. The most frequent answers were improvements in credibility and reliability of results, building trust, and improving understanding of the results of others. Besides, answers included the benefit of a practical approach by fostering task-based research, increasing visibility for your research by making tools available and open communication to foster research in general.

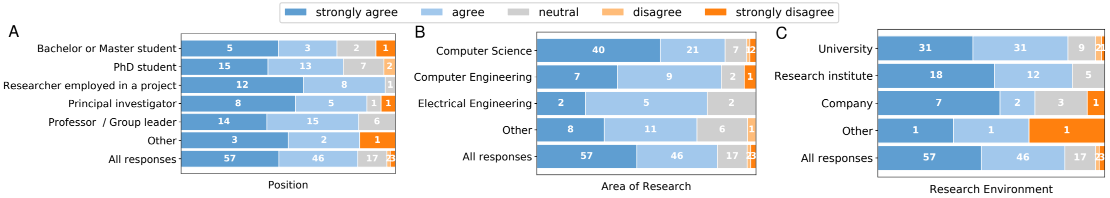


Figure 2. Responses to the question "*I want to reproduce the results of other researchers or groups from their original work (software tools or libraries) to compare it to my work.*" grouped by researchers' positions (A), research area (B) and research environment (C).

After showing the researchers' interests in reproducibility, which are aligned with the results from other published surveys, we now evaluate what researchers are willing to do to make their results reproducible for others and how much effort they are willing to invest to reproduce the results of others. Focusing on Fig. 3, we see that about half of the researchers typically try to reproduce the results of others by running their tools (53 out of 125). This again shows the demand for publishing scientific software artifacts. The average amount of time participants would invest in making software of others work to reproduce results was 23.12 hours, neglecting 2 two outliers who would spend 10^5 and 10^{35} or more hours. The responses of to the corresponding survey questions are visualized in Fig. 4 (A). Four 4 participants noted that they would invest 100-360 hours to reproduce results of others, 18 participants noted that they would invest 30-80 hours, 32 participants would invest 10-24 hours, 47 would invest 1-8 hours and only three 3 participants would not invest any time at all.

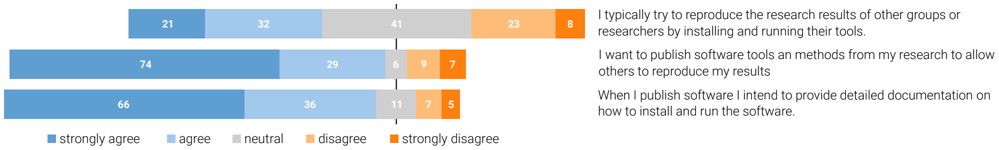
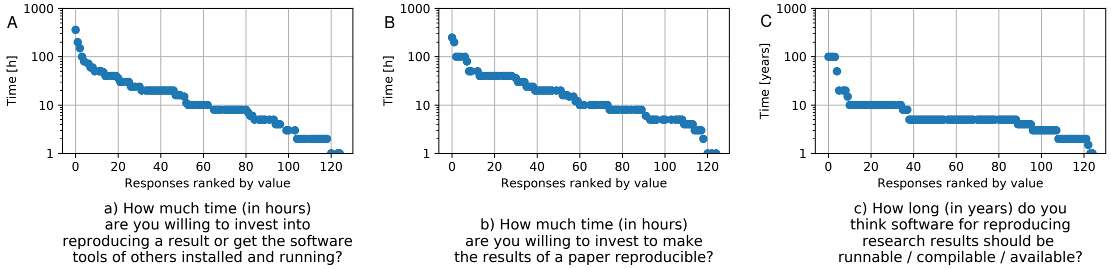


Figure 3. Responses to questions focusing on what researchers are willing to do to achieve reproducible results or to share artifacts.

Most researchers agreed they would like to publish their software to aid reproducibility. The question of whether researchers want to publish their software tools to allow others to reproduce their results was answered with agreement from the majority of researchers (103 out of 125) with only 16 disagreeing. When publishing software, 102 out of 125 researchers intend to provide detailed documentation on how to install and run the published software artifact. The question of how many hours researchers want to invest into making their results reproducible led to an average of 24.4 hours. We excluded three outliers with answers of 1000, 10^6, and 10^{25} hours as we agreed that the answer of 1000 hours – in other words 25 work weeks – and more is more likely to be a misunderstanding of the question and may include the original research work in addition to the extra work of making the results reproducible. The results can be seen in Fig. 4 (B). Summarizing the results in clusters results in six 6 participants investing 100-250 hours, 37 participants indicating they would invest 20-80 hours, 55 participants reporting to invest from 1-16 hours, and only four 4 indicating that they would not invest any time. Interpreting these numbers, we see that researchers are willing to invest more time to make their own research reproducible than to reproduce the results of others.



a) How much time (in hours) are you willing to invest into reproducing a result or get the software tools of others installed and running?

b) How much time (in hours) are you willing to invest to make the results of a paper reproducible?

c) How long (in years) do you think software for reproducing research results should be runnable / compilable / available?

Figure 4. Responses to the above questions on how much hours researcher would invest into reproducing and making reproducible as well as how many years result should be reproducible. Note that the y-axis is logarithmic.

The results of a multiple-choice question asking for the typical composition of research results shows that software implementations and datasets are already common artifacts of today's research, indicating the potential utility of making research reproducible. Besides results in written form – 107 researchers mentioned *published papers* and 37 participants *reports with detailed results* – a *software implementation* is part of the research results for 87 participants and 47 participants mentioned a *dataset* being part of their results.

Another important aspect for reproducible research is the long-term availability of results and artifacts. The effort of preparing and publishing software artifacts and results would ultimately be in vain if the artifacts later become inaccessible. Participants were asked about their opinion on how long results and necessary software artifacts should be available after initial publication. The results can be seen in Fig. 4 (C). With the exception of five outliers (with answers of 0, 10^6, 10^9, and 10^{25} years), the participants stated that software for reproducing results should be available for an average of 9.1 years. Summarizing through clusters 30 participants stated it should be from 0.5-3 years, 55 indicated it should be 4-5, 26 state 8-10, and 9 nine think it should be more than ten years available.

Asked about how they share research artifacts or make results reproducible, 90 out of 125 participants stated to have already published software at the time of their participation in the survey. Means of making their results reproducible were – multiple means could be specified – detailed instructions (68), make scripts (54), installation scripts (34), virtualization software (29), and container frameworks (15). There were two mentions of hosting web front ends as means of making results available and three mentions of public source code repositories as platforms for distribution.

Now that we are aware of current practices for making results reproducible, we focus on the role of reproducibility in the peer review process. Our assumption is that testing for reproducibility during the peer review process could enhance the credibility of published results and thereby increase the quality of a paper. This opinion is shared by the survey participants as visualized in Fig. 5: 87 out of 125 participants stated that checking for reproducibility should be part of the peer review process. Furthermore, 79 out of 125 participants would be willing to check results in addition to the traditional peer review process.

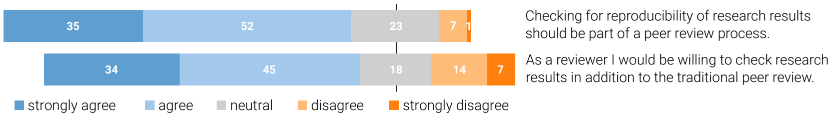


Figure 5. Responses on questions focusing on the role of reproducibility in the peer review process.

Here, differences among different positions and research areas can be found (see Fig. 6). When focusing on the researchers' position, nine 9 out of 10 bachelor or master students showed agreement, with none indicating disagreement. Principal investigators indicated the lowest agreement. Differences can also be seen regarding different research areas. Researchers from computer engineering showed the least agreement, whereas electrical engineers indicated the most agreement. Researchers from other research areas, including computer science, indicated a similar interest. We could not identify significant differences between different research environments.

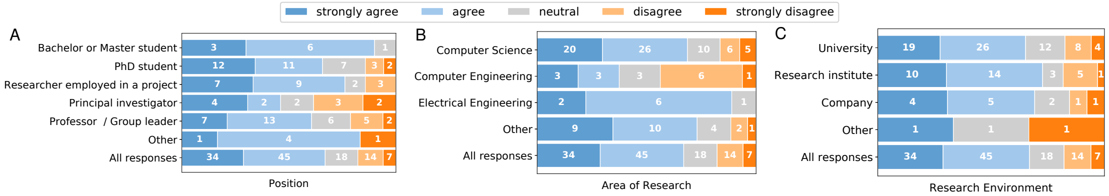


Figure 6. Responses to the question ≪*As a reviewer I would be willing to check research results in addition to the traditional peer review.*≫ grouped by the researchers' position (A), research area (B) and research environment (C).

When analyzing the survey results on the researchers' concerns regarding publishing scientific software artifacts, we can see that the traditional payment models of scientific publishers used for research papers are seen as critical for publishing software artifacts. Fig. 7 shows that 104 out of 125 researchers indicate that their results can be reproduced with free and open source software. This goes hand in hand with researchers' reluctance to pay for publishing or accessing software artifacts. Only 24 out of 125 researchers are willing to pay for making software tools, frameworks, and subsequently their results to be available to other researchers. A few more, but still only 28 out of 125 researchers indicate agreement with paying for being able to reproduce the results of others. These responses indicate the importance of possibilities for sharing software artifacts free of charge regardless of the platform.

Continuing in this vein, we asked why results cannot be reproduced using open source tools. 50 participants indicated the use of paid-for programming language environments, 35 the use of licensed operating systems, 19 the use of copyrighted materials, and 11 the use of commercial tools.

Computer security, when installing programs from others, is not a major concern for 69 out of 125 participants, which is alarming when reflecting on possible security issues. An explanation could be that the researchers' awareness is low because they themselves would not harm others and believe others to be benevolent as well. However, this mindset does not account for security issues that do not originate from other researchers, but from used third-party libraries. Therefore, software from unknown sources, or with unknown dependencies, should always be handled with care.

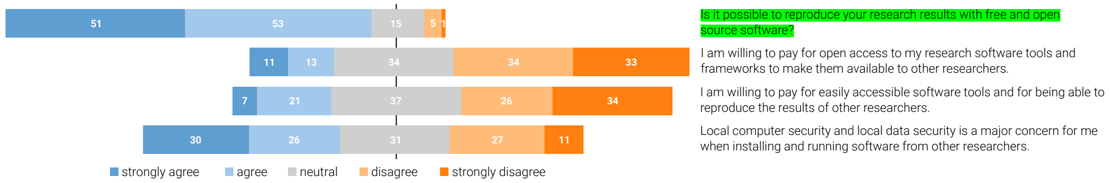


Figure 7. Responses to questions focusing on additional concerns when publishing scientific artifacts.

We further see that security awareness depends on the researchers' position (see Fig. 8). Undergraduate and master students indicate the highest awareness of security risks, while professors and principal investigators the lowest. A possible interpretation is that researchers in higher positions neglect security issues because of the high pressure to progress research. Students, in contrast, focus on smaller tasks and complete them more carefully. Regarding security awareness across different research areas, computer engineers have the highest awareness with 12 out of 19 researchers indicating agreement on the question "*Local computer security and local data security is a major concern for me when installing and running software from other researchers.*"~~.~~ For other fields, the awareness or lack thereof is almost equally low.

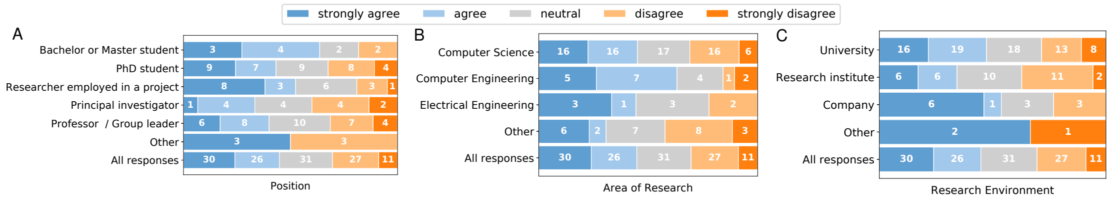


Figure 8. Responses to the question "*Local computer security and local data security is a major concern for me when installing and running software from other researchers.*" grouped by the researchers' position (A), research area (B) and research environment (C).

Besides economical and security concerns, we also asked researchers about additional reservations. A multiple-choice question on major concerns showed that when installing and running software from other researchers. This questions allowed for multiple choice as well as an other option, where participants could voice their concerns. Answers included:

- Ease of the installation (without major barriers),
- Hardware requirements like computation power, memory, or specialized equipment,
- License issues,
- Size of the download and installation,
- Used harddisk space after installation,
- I don't see additional concerns,
- Other (with the option of giving text here).

Participants primarily mentioned the ease of installation (104), license issues (72), and hardware requirements (71). Fewer participants noted the size of the download (27). Five other answers were entered~~, being~~:

- "I am sure it does not run on the first try 9 out of 10 times,"
- "External dependencies and their updateability / patchability in case of security fixes (should never depend on the initial publisher for third party libraries because they'd have to maintain their old packages for a long time); Also important: Downards [sic!] compatibility of "new" versions with old data & tools,"
- "Analytical reporoducability [sic!] & mathematical clarity (or correctness) is my main concern.",

- “Conflicting versions of additional required software ,“
- “complex build dependencies~~.~~”

A final open-ended question was about reservations towards publishing data and software: “*I have reservations for publishing software artifacts and data in research because ...*“For analysis following the approach of open coding the answers were labeled manually by the authors and the assigned labels were discussed until agreement was reached. The most common cluster of answers noted legal or privacy issues (14). Others pointed out the additional effort needed (8), commercial interests (8), missing reward or support for doing so (3), and that publishing artifacts is not part of the job, i.e., not supported by the group or organization (2). Note at that point that the raw data with all the responses is included in the supplementary material of the paper.

Regarding the aforementioned legal issues, it would be an interesting hypothesis that researchers would be more willing to share if legal issues and efforts are reduced. This may be achieved by license constraints (only licenses others can build upon) or exceptions for publishing research (leaving license issues aside for research by general agreement).

3.4 Correlation Analysis

Given the Likert scales for the answers,we did investigate the correlation (Spearman’s rank correlation) between answers to see if (i) intuitive and expected correlations exist and (ii) new and surprising correlations can be found. A table of all correlations with $|\rho| > 0.4$ is given in Tab. 1. The strongest correlation to be found with a coefficient of $\rho = 0.734$ and a p-value < 0.0001 was between the questions *Checking for reproducibility of research results should be part of a peer review process* and *As a reviewer I would be willing to check research results in addition to the traditional peer review*. Hence, people who stated to be willing to do reproducibility checks were more likely to find the idea of a review process with mandatory reproducibility checks attractive.

Another strong correlation ($\rho = 0.723$, $p < 0.0001$) was found between the questions *How much time (in hours) are you willing to invest into reproducing a result or get the software tools of others installed and running?* and *How much time (in hours) are you willing to invest to make the results of a paper reproducible?*~~.~~ With that correlation one can hypothesize that researchers with reproducibility in mind invest time in reproducing results as well as making their results reproducible.

A less strong but still rather interesting correlation ($\rho = 0.55$, $p < 0.0001$) was found between *I am willing to pay for open access to my research software tools and frameworks to make them available to other researchers.* and *I am willing to pay for easily accessible software tools and for being able to reproduce results of other researchers.*~~.~~ So with the overhead of participants not willing to pay for access and publishing of in context of reproducibility as indicated in Fig. 7, it is likely that researchers either like the idea of either paying for both, publishing and access, or none.

3.5 Threats to Validity

While a minor bias is assumed to be caused by the study’s title as participants may have been attracted by the title if they could identify with the topic of reproducibility, it is still valid to create hypothesis from the findings.

One possible limitation of the survey is the missing geographical distribution of the participants. We did not include questions on where participants are located or work primarily, and did not collect IP addresses. Hence, we cannot conclude if the survey result indicate a global trend, or if the preferences of researchers from different geographic regions differ. Similarly, a possible gender gap of the survey’s participants can not be evaluated.

For single and multiple choice questions with a pre-defined answer set in the survey the set of answer can introduce a certain bias to the results. Therefore, it was decided to avoid such questions if the risk of bias was high. In that sense,we also avoided quantizing numeric input, e.g.,the hours people spend ~~on~~ making their work reproducible. If it was necessary, we always included an open-answer option. ~~and~~ The pilot, pretests, survey of related work, and critical reflection by the authors were used as tools to minimize the bias. In one single case, i.e., the question *Which of the following are additional concerns when installing and running software from other researchers?*~~,~~ the open-answer option showed that at least one pre-determined answer was missing. Several participants noted complex build dependencies (also mentioned as conflicting versions of additional required libraries or external dependencies) are likely to be another major concern.

Table 1. Correlations in the survey answers with $| \rho | > 0.4$ using Spearman's rank correlation.

Questions	ρ	p-value
Checking for reproducibility of research results should be part of a peer review process. As a reviewer I would be willing to check research results in addition to the traditional peer review.	0.734	< 0.0001
How much time (in hours) are you willing to invest into reproducing a result or get the software tools of others installed and running? How much time (in hours) are you willing to invest to make the results of a paper reproducible?	0.723	< 0.0001
I am willing to pay for open access to my research software tools and frameworks to make them available to other researchers. I am willing to pay for easily accessible software tools and for being able to reproduce results of other researchers.	0.550	< 0.0001
I want to publish software tools and methods from my research to allow others to reproduce my results. When I publish software I intend to provide detailed documentation on how to install and run the software.	0.490	< 0.0001
I want to reproduce the results of other researchers or groups from their original work (software tools or libraries) to compare it to my work. I want to build my research tools by extending on the work (software, tools or frameworks) of other researchers or groups.	0.482	< 0.0001
I want to reproduce the results of other researchers or groups from their original work (software tools or libraries) to compare it to my work. I typically try to reproduce the research results of other groups or researchers by installing and running their tools.	0.477	< 0.0001
When I publish software I intend to provide detailed documentation on how to install and run the software. I want to build my research tools by extending on the work (software, tools or frameworks) of other researchers or groups.	0.412	< 0.0001
When I publish software I intend to provide detailed documentation on how to install and run the software. I typically try to reproduce the research results of other groups or researchers by installing and running their tools.	0.410	< 0.0001
Published software artifacts are added value to the published text in the research paper. I want to build my research tools by extending on the work (software, tools or frameworks) of other researchers or groups.	0.408	< 0.0001

Participants could have had overlaps in the categorization of positions, for example a person could be a PhD student and an employed researcher in a project at the same time. In this case, participants might have selected the category randomly or selected the category they appreciate more. Despite this, as long as no intentional or unintentional mistakes are made in the answers, each category will contain samples that are member of this category.

English as the only language for the survey might be a further limitation. Nevertheless, English is the working language of the target audience, and consequently, we assume the influence by the survey~~'s~~ language to be negligible.

4 REQUIREMENTS AND GENERAL GUIDELINES

The survey results indicate that a majority of researchers of all levels consider reproducibility as very relevant. There is further a strong interest in doing work to make one's own results reproducible, a strong interest to use results of others for comparison to own work, and to some extent, a motivation to try to reproduce work for review purposes.

To achieve this, it is necessary to make all information that is necessary to reproduce the results available together with a publication. Additionally, the effort necessary to reproduce the results needs to match the value of doing the work. Work reproducing or refuting previous results is overall much less appreciated than original work, so the effort a researcher is willing to invest in order to reproduce previous results is much lower than the effort they are willing to put in to produce new work. On the other hand, when planning to build own research on top of other results the investment can be higher. The most critical case is in reviewing, when reproducibility is intended to be checked as part of the reviewing process. Reviewers have a strict timeline to perform their review, so there is a need for a straightforward, mostly automated process to reproduce results. Moreover, despite contributing to verifying the results of a paper, reviewers are not mentioned in connection with the work. As reviewers work voluntarily, they are probably the least motivated to reproduce results.

Moreover, researchers have responded critically to commercial systems introducing payments, either from the publishing researcher side or from the consumer side. A majority of participants also name security as a concern in this context, which highlights an issue to be addressed for researchers being security-aware as well as for those who are less concerned about security.

In order to address these issues, the following guidelines are proposed:

- Code, data, and information on how to conduct an experiment should be gathered in a single place (a single container), which can be found in connection with the paper.
- The reproduction process should be highly automated (for example, by an easy-to-handle build and execution script).
- To address security issues, the published artifacts should be provided as source code and scripts allowing for running the code in a virtual environment should be provided.
- Commercial libraries and other components that require reviewers to pay for access should be avoided.
- Since research papers tend to create some interest even long after they have been published, it is necessary to ensure that software and environment for the reproduction process remain available, either by packing all necessary components into a container or by referring to well-archived open source tools.
- The time and necessary information to reproduce results should be tested with an independent person. Unless the size of the project requires it, the reproduction process should take at most two days.

5 EXISTING TOOLS

Most tools for sharing software artifacts are also used in the development of software artifacts. These could be either tools for simple tasks such as compiling software projects, but also more complex tools for tasks such as automated dependency installation and software packaging. To prevent unnecessarily complex configuration, it is wise to select tools based upon the complexity of the software artifact. Software artifacts which are complex to run require more sophisticated tools with high levels of abstraction, whereas simple artifacts do not require complex tools to run.

In this section, we tackle our second research question by presenting four open source tools for sharing software artifacts, ranging from tools for compiling simple artifacts to sophisticated frameworks for sharing self-contained software environments. The tools have been selected despite of their different scopes because of their potential to support reproducible research. It has to be noted~~,~~ that a complex project might even incorporate multiple tools, for example a build system within a virtual environment.

We begin with a discussion of simple tools, such as *CMake*, which are used for build management and continue by discussing tools utilizing a higher level of abstraction. For discussion purposes, well-known tools, each representing a class of tools with similar functionality, were selected~~;~~ Discussed pros and cons are valid not only for the discussed tool itself, but for the complete class represented by the tool. Finally, we summarize the features of the different tools and discuss the importance of their benefits, according to the survey~~s'~~ results presented in Section 3.

5.1 CMake

CMake is a cross-platform build tool based on C++. It is designed to be used with native build environments such as *make*. Platform-independent build files are used to generate compiler-specific build instructions, which define the actual build process. Main features of CMake are tools for testing, building, and packaging software, as well as the support of hierarchical structures, automatic resolution of dependencies and parallel builds.

One drawback of CMake and similar build management systems is that required libraries or other dependencies of software artifacts must be available and installed in the required version on the host system in order to successfully build the project. This could lead to extensive preparations for a build which is mandatory for executing software artifacts.

CMake has been chosen for discussion because it is one of the most used tools of this type. Tools with similar functionality are *configure scripts*, the *GNU Build System* and the *WAF* build automation system.

5.2 Gradle

Gradle is a general purpose build tool based on Java and Groovy. Gradle integrates the build tools Maven as well as Ant and can replace or extend both systems. Main features of Gradle are the support for Maven repositories for dependency resolution and installation and the out of the box support for common tasks, i.e., building, packaging and reporting. Gradle supports multiple programming languages, but has a strong focus on Java, especially as it is the recommended build system for Android development. An integrated wrapper system allows to use Gradle for building software artifacts without installing Gradle. Dependency installations and versions are maintained automatically. If a build requires a new version of a library, it is downloaded and installed automatically.

The automated dependency installation is a great benefit of Gradle, although there are still some challenges to overcome. One issue is that automated dependency installation only works, if the required libraries are offered in an online repository. If the required dependency is removed from the online repository, building any software depending on this library becomes impossible.

For other programming languages, tools with similar functionality are available, i.e. the *Node Package Manager* (NPM) for JavaScript projects or *pip* for Python projects.

5.3 Docker

The open source software *Docker* allows packaging software applications including code, system tools, and system libraries into a single *Docker image*. The resulting image can be published, downloaded and executed on various systems without operating system restrictions in a virtualized environment. This way, an application embedded in a Docker image will execute in a predefined way, independent of the software environment installed on the host computer. The only requirement for the host system is the installed Docker engine.

A Docker image is a kind of lightweight virtual machine image. It could contain the runtime environment for a single application with or without graphical user interface, but it could also contain a ready to deploy server application for web services or even environments for heavy calculations or simulations. When running the Docker image, a *Docker container* is launched. A Docker container can be seen as an isolated runtime environment, which uses the kernel of the host operating system and thereby becomes more lightweight than traditional virtual machines. A running Docker container can be accessed via a terminal or a graphical user interface allowing for a broad range of applications.

Docker images can be shared in two different ways. The first way is to export a running container including all files and executables as image and to share it as a single file. This file can be large in size but is fast to launch by others. The second way is to create a so-called *Dockerfile*. Dockerfiles contain the building instructions for Docker images. These instructions include commands for installing required dependencies and for installing the shared software artifact itself. When building a Docker image from a Dockerfile, all instructions from the Dockerfile are automatically executed. This leads to a small Dockerfiles, but a more complex import process. In addition, when using Dockerfiles, all dependencies need to be available either in online repositories, or locally on the machine building the image.

The major difference between Docker and the previously presented tools is that Docker is not usually used for the development of an artifact. In most cases, a Docker image is created for sharing a predefined environment in a team. This means that the image is created and the software artifact is deployed in the container afterwards.

Table 2. Comparison of tools for sharing scientific software artifacts

Tool	CMake	Gradle	Docker	VirtualBox
Security	no security mechanisms	no security mechanisms	sandboxed environment	sandboxed environment
Supported platforms	Linux, MacOS, Windows	Java VM	Linux, MacOS, Windows	Linux, MacOS, Windows
Required knowledge for sharing	little	little	moderate	little
Effort for sharing	little	little	moderate	high
Required knowledge for installation and execution	moderate	moderate	little	little
Effort for installation and execution	moderate/high	little	little	little
Size of shared object	small	small	up to multiple GBs	up to multiple GBs
Limitations	Installation could be exhausting	Specific Gradle project structure recommended	GUI requires extra effort	Images always include the entire operating system

An alternative to Docker is using Linux Containers (LXC), which allow to run multiple isolated Linux systems on a single host.

5.4 VirtualBox

VirtualBox is an open source software for the virtualization of an entire operating system. VirtualBox emulates a predefined hardware environment, where multiple operation systems, like Windows, Mac OS and most Unix Systems can be installed. The installed operating system is stored as persistent image, which allows the installation and configuration of software. Once the image is created, it can be shared and executed on multiple machines.

As mentioned before, VirtualBox emulates the entire hardware of a computer resulting in higher execution overhead as well as higher setup effort. Before the scientific software artifact can be installed in a VirtualBox container, an operating system and all dependencies have to be installed.

A non-open source alternative to VirtualBox is VMWare, which has similar functionality.

5.5 Comparison of Analyzed Tools

After the presentation of selected tools in the last section, we now want to compare their features for sharing scientific software artifacts. As criteria for the comparison, we focus in this section on important aspects of software for researchers, according to the survey presented in Section 3. Table 2 briefly summarizes our findings; a description of each criteria is found throughout this section. The ratings in Table 2 are based on qualitative comparisons, as well as on our experience from using the tools for making three different research projects reproducible, as elaborated in Section 6.

Security: As indicated by the survey, local computer and data security is a major concern for many researchers. Some software artifacts require administrator access rights on the local machine in order to be executed. These access rights allow malicious behavior, which could lead to unwanted consequences on the local machine or on the local network.

VirtualBox and Docker execute software artifacts in sandboxed environments and therefore allow the secure execution of software artifacts. Tools like CMake and Gradle do not offer this security mechanism. When executing a shared software artifact from untrusted sources, a sandboxed environment is recommended.

Supported platforms: CMake, Docker, and VirtualBox are compatible with most Linux platforms, recent versions of MacOS, and selected versions of Windows 10. Gradle works as long as the Java Virtual Machine is available. Besides this platform support it has to be kept in mind that the software artifacts itself could require a certain operating system. This problem can be mitigated through virtualization of Docker and VirtualBox.

Required knowledge for sharing: If a build management tool is used in the development of a scientific software artifact, we assume that the researchers are familiarized with the build management tool during the development phase. Therefore, no additional knowledge for the researcher who is sharing the artifact is required. VirtualBox also does not require a lot of additional background information. Everybody who is able to install an operating system is able to share a software artifact embedded in a VirtualBox image. The terminology of Docker seems to be confusing at first glance, requiring some time to become familiar with Docker~~’s~~ concepts.

Effort for sharing: CMake, Gradle,and other build management systems are intended to define a standardized build process. If a build management system is used during the development of the scientific software artifact, no additional effort arises for sharing. The configuration file for the build management system can be shared along the source code of the software artifact.

Docker and VirtualBox are usually not directly involved in software development. In most cases, a Docker or VirtualBox image has to be created explicitly for sharing the software artifact. The structured process of building a Docker container allows easy reuse of existing Docker containers for other software artifacts. In the case of VirtualBox, the whole VirtualBox image has to be shared on a file server. Docker containers can be shared on the free to use Docker Hub or on a file server. Alternatively, a Dockerfile, which contains the building instructions for a Docker container, can be created and shared as a text file. However, using a Dockerfile requires all dependencies being available in repositories, adding additional complexity to the overall process.

Required knowledge for installation and execution: Researchers are often not familiar with the tools used for the creation of software artifacts. Reading the documentation of build management tools can be exhausting and time-consuming for the short test of an artifact. CMake and Gradle require some knowledge in order to build a software artifact, especially if errors appear.

VirtualBox and Docker are easier to use. If a Docker image is hosted on DockerHub, a single command is sufficient for downloading and running the image. If this command is provided, no additional knowledge is required. Due to a graphical user interface, running a virtual box image is even easier.

Effort for installation and execution: According to the survey results, ease of installation is a major consideration for most researchers (104 of 125 participants). Regarding the installation of the used tool itself, Gradle has the lowest requirements. The Gradle Wrapper allows installing dependencies and the build of artifacts without installing Gradle itself. For installing and executing the shared software artifact, the highest effort arises when using CMake, where required dependencies have to be installed manually. For building and executing software artifacts with Gradle only a few commands are required. Docker and VirtualBox require the least effort; the shared image only needs to be executed.

Size of shared object: When using CMake or Gradle, the source code of the software artifact and the configuration file of the build management tool have to be shared, which usually leads to small shared objects.

The shared container of Docker or VirtualBox has to contain the source code and all other tools which are required for execution, such as the operating system. This results in large shared objects, in some cases the size of a Docker container exceeds one Gigabyte.

Alternatively, Docker provides an option allowing for smaller shared objects – Dockerfiles. A Dockerfile contains only text instructions for building a Docker image. Therefore, the size of a Dockerfile is only a few kilobytes, but once executed, Docker automatically pulls the artifact~~’s~~ source code from provided repositories and builds the software artifact, resulting in a large Docker image on the local machine. Nevertheless, the size of the download is not a major concern for the majority of the survey participants.

Limitations: All analyzed tools have limitations. CMake is a lightweight tool for software development, but the effort for installing the dependencies of a software artifact could be extensive. Furthermore, it is only applicable for a handful of programming languages such as C or C++.

When Gradle is chosen as build system early in the development phase, it is perfectly suited for Java projects. Using Gradle for existing projects can be cumbersome because it requires additional configuration for projects that do not comply with Gradle~~'s~~ default project structure. Especially for researchers that are not familiar with Gradle, the time spent for this additional configuration step should not be neglected.

Docker is perfectly suited for command-line or web applications, which is the case for a huge amount of scientific software artifacts. Additional configuration is required to support GUIs of desktop applications. FREVO (see section 6.2), used in one of our examples, demonstrates GUI support for desktop applications with Docker.

VirtualBox is applicable for all types of software artifacts, but the overhead of creating and sharing a VirtualBox image could be huge. For sharing an artifact, independent of its size and complexity, a complete operating system has to be installed and shared.

6 EXAMPLES

After introducing background information in the last sections, three examples are presented analyzing the applicability of various tools for sharing software artifacts. Three scientific artifacts from different computer science research areas allowed us to focus on various types of artifacts with different build systems and procedures for sharing. The first example – Stochastic Adaptive Forwarding – is a simulation scenario, which can be executed on a command line in order to conduct performance evaluations. Second, FREVO is a simulation tool, mainly controlled via a graphical user interface. The third example – LireSolr – is a server-based application used for image retrieval.

6.1 Stochastic Adaptive Forwarding

Stochastic Adaptive Forwarding (SAF) (Posch et al. (2017)) is a forwarding strategy for the novel Internet architecture Named Data Networking (NDN) (Zhang et al. (2014)). Forwarding strategies in NDN are responsible for forwarding packets to neighboring nodes and therefore select the paths of traffic flows in the network.

The Network Forwarding Daemon (NFD) implements the networking functionalities of NDN. It is written in C++ and uses the WAF build automation system. The network simulator ns-3/ndnSIM (Mastorakis et al. (2016)) is used for testing purposes, which also uses the WAF build system. For testing SAF in the simulation environment three steps are required: i) Installation of the NFD; ii) installation of the network simulator ns-3/ndnSIM and finally iii) patching SAF into a compatible version of the NFD. The installation of SAF was tested and analyzed in the standard way by using WAF and Docker.

SAF with WAF: The standard way of developing NDN forwarding strategies is by using the WAF build automation system. The functionality of the WAF build system is similar to the functionality of CMake. This means that WAF automatically resolves dependencies, but the installation of dependencies must be performed manually. Although extensive installation instructions were published[2], it is tricky to install the simulator and its dependencies. Furthermore, there are slightly undocumented differences when installing the NDN framework on different Unix versions. Once the NDN framework is compiled in the correct version, it is easy to patch SAF. Nevertheless, it can take up to several hours to initially install and compile the NDN framework with SAF.

SAF with Docker: NDN and SAF are licensed under GPL V3, meaning that there are no legal concerns over packaging the software. Technically, Docker provides two options for creating and sharing images. The first is to check out a preconfigured image like Ubuntu Linux from the Docker website and connect to it via terminal. All required changes can be made in the terminal and finally persisted with a commit. The altered image can be shared via the Docker website or as binary file. The second possibility to create the image is by using Dockerfiles. These files contain simple creation instructions for images and can be shared easily due to their small size. To build an image, the Dockerfile can be executed on any host with the Docker framework installed. Both variants were tested for SAF. The resulting images, containing all dependencies and the compiled software artifacts, have a size of about 4.6 GB with the size of the Dockerfile being about 2 KB. Using the precompiled image[3], running the image only takes an instant. The execution of the Dockerfile takes, depending on the Internet connection and the computing power of the host system, between 15 and 60 minutes. Once the image is running, the results of the paper can be reproduced or new experiments with SAF can be conducted using the provided network simulator.

[2] https://github.com/danposch/SAF, last visited 2019-07-08

[3] https://hub.docker.com/r/phmoll/saf-prebuild/, last visited 2019-07-08

6.2 FREVO

FREVO (Sobe et al. (2012)) is an open source framework to help engineers and scientists in evolutionary design or optimization tasks to create a desired swarm behavior. The major feature of FREVO is the component-wise decomposition and separation of the key building blocks for an optimization task. This structure enables the components to be designed separately allowing the user to easily swap and evaluate different configurations and methods or to connect an external simulation tool. FREVO is typically used for evolving swarm behavior for a given simulation (Fehervari and Elmenreich (2010); Monacchi et al. (2014)). FREVO is a mid-sized project with 50k lines of mostly Java code, having a graphical interface as well as a mode for pure command line operation, e.g., to be used on a simulation server. The component-based structure allows to easily extend and remove components (e.g., a simulation, a type of neural network, an optimization algorithm), which sometimes creates some effort in newly setting up FREVO.

FREVO was tested and analyzed with the following three tools:

FREVO with build script: Until recently, FREVO was provided as a download zip file[4] that included sources of the main program and additional components together with an ant build file. However, there had been problems in the past with different language versions of Java. A further problem can be dependencies on third party tools or libraries, which are not automatically maintained by this type of build script.

FREVO with Gradle: An analysis showed that the current structure of FREVO, especially due to its component-plug-in-architecture, conflicts with the expected and possible project structure for Gradle.

FREVO with Docker: Since FREVO and its components are open source under GPL V3, there was neither a legal nor a technical problem to put it into a virtual Docker container. We used an Ubuntu Linux system that was provided by Docker. Openjdk8 was installed as Java Runtime environment. After installing FREVO in the Docker system, it was pushed onto the free Docker Hub hosting platform.[5] To reproduce a result made with Frevo it thus possible to (given that Docker is installed) download and execute the respective Docker container. In general, the result was easily usable, apart from some effort to get a graphical display working. The parallelization of simulation, which is a natural ability of FREVO, works fine as well inside a Docker container. The FREVO container has a compressed size of 223 MB, which is mostly due to the files of Ubuntu Linux.

6.3 LireSolr

LireSolr (Lux and Macstravic (2014)) is an extension for the popular Apache Solr[6] text retrieval server to add functionality for visual information retrieval. It adds support for indexing and searching images based on image features and is for instance in use by the World Intellectual Property Organisation, a UN agency, within the Global Brand DB[7] for retrieval of similar visual trademarks.

LireSolr brings the functionality of the LIRE library (Lux and Marques (2013)) to the popular search server. While LIRE is a library for visual information retrieval based on inverted indexes, it is research driven and intented to be integrated with local Java applications. Apache Solr is more popular than the underlying inverted index system, Lucene, as it allows to modularize retrieval functionality by providing a specific retrieval server with cloud functionality and multiple APIs to access it for practical use.

LireSolr is intended for people who need out of the box visual retrieval methods without the need for integrating a library in their applications. It can be called from any mobile, server, or desktop platform and runs on systems with a Java 8 runtime. This flexibility is valued among researchers as well as practitioners. LireSolr is hosted on Github[8]. Gradle and Docker build files are part of the repository.

LireSolr with Gradle: The standard method of building LireSolr is by using Gradle. Current IDEs can import Gradle build files; any task can be done from within the IDE. While Gradle makes sure that the right version for each library is downloaded and everything is ready to build, installing the new features to the Solr server has to be done manually. The supporting task in Gradle just exports the necessary JAR files. The user or developer has to install Solr, then create a Solr core, change two configuration files, copy

[4] `http://frevo.sourceforge.net/`, last visited: 2019-07-08
[5] `https://hub.docker.com/r/frodewin/frevo/`, last visited: 2019-07-08
[6] `http://lucene.apache.org/solr/`, last visited 2019-07-08
[7] `http://www.wipo.int/branddb/en/`, last visited 2019-07-08
[8] `https://github.com/dermotte/liresolr`, last visited 2019-07-08

the JARs and restart the server to complete the installation. While these steps are extensively described in the documentation, it still presents a major effort for new users without prior experience of retrieval in general or using Apache Solr.

LireSolr with Docker: As LireSolr is extending Solr by adding additional functionality, the intuitive way to create a Docker container is to extend the Solr Docker container. The *Dockerfile* defining the build of the Docker container is part of the LireSolr repository, where a specific Gradle task is building and preparing relevant files for the creation of the image. This includes the aforementioned JARs and config files as well as a pre prepared Solr core and a small web application as a client. The Docker container can easily be run and provides basic functionality for digital image search. Developers who just want to test LireSolr can get it running within seconds using Docker Hub[9].

THE ONE TOOL TO REPRODUCE THEM ALL?

In the previous sections, we presented tools for sharing software artifacts and examples showing how the tools can be applied in order to share scientific software artifacts. In this section, we now reflect on the advantages and shortcomings of the tools with respect to the results from our survey presented in Section 3.

Each of the presented tools has its pros and cons. For instance, the additional effort for sharing an artifact when using a build management tool is very low because in most cases a build management tool is used during the creation of the artifact. In contrast, it can be challenging and time-consuming for other researchers to get the build management tool up and running because required dependencies or the installation process may not be documented in detail. Software artifacts, which are provided as virtualized containers are easy to run and provide a high degree of security but are inconvenient in case a researcher wants to build upon previously published software artifacts.

When weighing these advantages and shortcomings we quickly see that *the one tool to reproduce all our scientific results* does not exist. Nevertheless, based on our findings from the survey we now want to give recommendations for creating reproducible results and scientific software artifacts which can be easily used by other researchers.

The survey clearly showed that many researchers are interested in building their research on the work of others, which becomes much easier, when published software artifacts can be reused. Furthermore, we saw that the average time researchers are willing to invest to get artifacts running is only about two days. Thus, we assume that it is very important for researchers to get the artifact running quickly, otherwise, researchers lose interest in using the artifact and start developing their own solution. When taking the demand for security into account, we see that virtualized containers appear to be a good choice. The provided software artifact can be executed without the overhead of installing it, by simply running the container. Furthermore, it is possible to become familiar with the artifact in the virtualized environment and check if the artifact is suitable to base own work on it.

When researchers decide to build on the artifact, it may be cumbersome to continue using a virtualized container, because altering a software artifact is more convenient on a local system. This means that the researcher has to install the artifact locally, without virtualized container. According to our study, researchers currently prefer providing detailed instructions and build tools. Solely relying on this information, it could be challenging to install the artifact, as already discussed.

Dockerfiles are one solution to overcome this issue. As already explained, a Dockerfile is a kind of a construction guideline for Docker containers. It contains all command line directives, which are required to build a Docker container and can therefore be seen as exact procedure for the local installation of an artifact. Following the commands listed in the Dockerfile, local installation of a software artifact is relatively easy. These commands ensure that all dependencies are installed correctly, otherwise it would not be possible to create a Docker container. This means that by providing a Dockerfile, both options become possible, software artifacts can be executed in a secure container, but can also be easily installed by following instructions from the Dockerfile.

Another finding of our survey is that the long-term availability of software artifacts is important for researchers and should be around 10 years. In addition, the ACM Artifact Reviewing and Badging guideline[10] emphasizes the importance of being able to reproduce results after a long time, by providing a

[9] `https://hub.docker.com/r/dermotte/liresolr/`, last visited: 2019-07-08

[10] `https://www.acm.org/publications/policies/artifact-review-badging`, last visited 2019-07-08

separate badge for artifacts which are archived in archival repositories. When looking at our presented tools, we can see technical, as well as legal issues on the way to achieve long term availability. Although services, such as Code Ocean[11] or Dryad[12], are available for archiving software artifacts, the following points should be kept in mind. Tools such as Gradle rely on online repositories for downloading required dependencies. If only one of the required dependencies becomes unavailable, the build fails. This means that all dependencies, as well as all required tools have to be included when the artifact is archived. This leads to technical issues, because the amount of required tools to reproduce a result could be tremendously high. For instance, if a required operating system or compiler is no longer available, the results can not be reproduced, which means that even these tools must to be archived. Besides this technical issue, packaging these tools could lead to legal issues as well when tools with limiting licenses are used. Furthermore, operators of platforms for archiving software could decide to discontinue service. This would result in loss of all artifacts archived by this provider.

8 CONCLUSION

This paper focused on the reproducibility of research results in computer science. We collected the opinions and requirements of 125 researchers via an online survey. Analysis of the survey's results confirmed our initial assumption that the reproducibility of research results is an important concern in computer science research. In addition, researchers not only want to reproduce results, but also want to base their own work on the results of others. The main reasons for the importance of reproducibility are improved credibility and improved understanding of results. Using established commercial models, as they are common for scientific publications, was seen as critical. Moreover, the majority of survey participants showed a willingness to use open source tools for making their results accessible and reproducible. Based on the researchers' opinions, we created guidelines which aid researchers in making their research reproducible. The applicability of various tools for publishing software artifacts was discussed while keeping our guidelines in mind. Scientific artifacts of different research areas in computer science were used to test the applicability of discussed tools for sharing reproducible research.

We identified a conflict between comprehensibility and simplicity of using a tool versus security measures avoiding to compromise one's system when testing foreign code. Available tools provide a variety of possible solutions, however, we could not identify a single tool fulfilling all requirements.

Finally, we discussed our findings and concerns on the process of publishing reproducible research. According to our study, the long-term availability of reproducible results is of great importance to many researchers, but we identified open issues in achieving availability for longer periods. Even if reproducibility of research is not common practice yet, we recognized a strong positive shift towards reproducible research, backed not only by individual researchers, but also by renowned scientific journals and publishers.

With this work already leading to new insights regarding reproducibility, it also installs a beachhead for future research. With the survey as input and the discussions regarding the interpretation we identified the context of a researcher as a hypothetically highly influential factor on the view on reproducibility. So how do for instance not only cultural, geographical, and project background of a researcher, but also the research area as well as the research communities influence the willingness to investigate extra time in making results reproducible? Future work could also address the question whether and to what extend project size would influence the willingness to invest time into reproducing work.

ACKNOWLEDGMENTS

We would like to thank all participants of the survey for their valuable input and all colleagues who helped us by sharing their practical experience and discussion. We thank the anonymous reviewers for their constructive comments on a previous version of the paper. We are grateful to Lizyy Dawes for proofreading an earlier version of this paper.

REFERENCES

Allen, A. and Schmidt, J. (2015). Looking before Leaping: Creating a Software Registry. *Journal of Open Research Software, 3 (1):e15*, 3.

[11] `https://codeocean.com/`, last visited 2019-07-08

[12] `https://datadryad.org/`, last visited 2019-07-08

Arbona, A., Artigues, A., Bona-Casas, C., Massó, J., Miñano, B., Rigo, A., Trias, M., and Bona, C. (2013). Simflowny: A general-purpose platform for the management of physical models and simulation problems. *Computer Physics Communications*, 184(10):2321 – 2331.

Boettiger, C. (2015). An introduction to docker for reproducible research. *SIGOPS Oper. Syst. Rev.*, 49(1):71–79.

Bonaventure, O. (2017). The january 2017 issue. *SIGCOMM Comput. Commun. Rev.*, 47(1):1–3.

Brinckman, A., Chard, K., Gaffney, N., Hategan, M., Jones, M. B., Kowalik, K., Kulasekaran, S., Ludäscher, B., Mecum, B. D., Nabrzyski, J., Stodden, V., Taylor, I. J., Turk, M. J., and Turner, K. (2018). Computing environments for reproducibility: Capturing the "whole tale". *Future Generation Computer Systems*.

Casadevall, A. and Fang, F. C. (2010). Reproducible science? *Infection and Immunity*, 78(12):4972–4975.

Chirigati, F., Shasha, D., and Freire, J. (2013). Reprozip: Using provenance to support computational reproducibility. In *Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance*, TaPP '13, pages 1:1–1:4, Berkeley, CA, USA. USENIX Association.

Cito, J., Ferme, V., and C. Gall, H. (2016). Using docker containers to improve reproducibility in software and web engineering research. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pages 609–612.

Fehervari, I. and Elmenreich, W. (2010). Evolving neural network controllers for a team of self-organizing robots. *Journal of Robotics*.

Flittner, M., Mahfoudi, M. N., Saucez, D., Wählisch, M., Iannone, L., Bajpai, V., and Afanasyev, A. (2018). A Survey on Artifacts from CoNEXT, ICN, IMC, and SIGCOMM Conferences in 2017. *SIGCOMM Comput. Commun. Rev.*, 48(1):75–80.

Hanson, B., Sugden, A., and Alberts, B. (2011). Making data maximally available. *Science*, 331(6018):649.

Janin, Y., Vincent, C., and Duraffort, R. (2014). Care, the comprehensive archiver for reproducible execution. In *Proceedings of the 1st ACM SIGPLAN Workshop on Reproducible Research Methodologies and New Publication Models in Computer Engineering*, TRUST '14, pages 1:1–1:7, New York, NY, USA. ACM.

Lux, M. and Macstravic, G. (2014). The lire request handler: A solr plug-in for large scale content based image retrieval. In Gurrin, C., Hopfgartner, F., Hurst, W., Johansen, H., Lee, H., and O'Connor, N., editors, *MultiMedia Modeling: 20th Anniversary International Conference, MMM 2014, Dublin, Ireland, January 6-10, 2014, Proceedings, Part II*, pages 374–377, Cham. Springer International Publishing.

Lux, M. and Marques, O. (2013). Visual information retrieval using Java and LIRE. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 5(1):1–112.

Mastorakis, S., Afanasyev, A., Moiseenko, I., and Zhang, L. (2016). ndnSIM 2: An updated NDN simulator for NS-3. Technical Report NDN-0028, Revision 2, NDN.

McMurdie, P. J. and Holmes, S. (2013). phyloseq: An r package for reproducible interactive analysis and graphics of microbiome census data. *PLOS ONE*, 8(4):1–11.

Monacchi, A., Zhevzhyk, S., and Elmenreich, W. (2014). HEMS: A home energy market simulator. *Computer Science – Research and Development*.

Morin, A., Urban, J., Adams, P. D., Foster, I., Sali, A., Baker, D., and Sliz, P. (2012). Shining light into black boxes. *Science*, 336(6078):159–160.

Peng, R. D. (2011). Reproducible research in computational science. *Science*, 334(6060):1226–1227.

Poline, J.-B., Breeze, J. L., Ghosh, S., Gorgolewski, K., Halchenko, Y. O., Hanke, M., Haselgrove, C., Helmer, K. G., Keator, D. B., Marcus, D. S., Poldrack, R. A., Schwartz, Y., Ashburner, J., and Kennedy, D. N. (2012). Data sharing in neuroimaging research. *Front Neuroinform*, 6:9.

Posch, D., Rainer, B., and Hellwagner, H. (2017). SAF: Stochastic adaptive forwarding in named data networking. *IEEE/ACM Transactions on Networking*, 25(2):14.

Ram, K. (2013). Git can facilitate greater reproducibility and increased transparency in science. *Source Code for Biology and Medicine*, 8(1):7.

Shrader-Frechette, K. and Oreskes, N. (2011). Symmetrical transparency in science. *Science*, 332(6030):663–664.

Sobe, A., Fehérvári, I., and Elmenreich, W. (2012). FREVO: A tool for evolving and evaluating self-organizing systems. In *Proceedings of the 1st International Workshop on Evaluation for Self-Adaptive* *and Self-Organizing Systems*, Lyon, France.

Steiniger, S. and Hunter, A. J. (2013). The 2012 free and open source GIS software map – A guide to facilitate research, development, and adoption . *Computers, Environment and Urban Systems*, 39(0):136 – 150.

Vitek, J. and Kalibera, T. (2011). Repeatability, reproducibility and rigor in systems research. In *Proceedings of the 11th International Conference on Embedded Software EMSOFT 2011*, Taipei, Taiwan.

Walters, W. P. (2013). Modeling, informatics, and the quest for reproducibility. *Journal of Chemical Information and Modeling*, 53(7):1529–1530.

Witten, D. M. and Tibshirani, R. (2013). Scientific research in the age of omics: the good, the bad, and the sloppy. *J Am Med Inform Assoc*, 20(1):125–127. amiajnl-2012-000972[PII].

Zhang, L., Afanasyev, A., Burke, J., Jacobson, V., Claffy, K., Crowley, P., Papadopoulos, C., Wang, L., and Zhang, B. (2014). Named data networking. *SIGCOMM Comput. Commun. Rev.*, 44(3):66–73.