Wilhelm Hasselbring*, Leslie Carr, Simon Hettrick, Heather Packer, and Thanassis Tiropanis

From FAIR research data toward FAIR and open research software

https://doi.org/10.1515/itit-2019-0040
Received October 10, 2019; revised November 30, 2019; accepted January 25, 2020

Abstract: The Open Science agenda holds that science advances faster when we can build on existing results. Therefore, research data must be FAIR (Findable, Accessible, Interoperable, and Reusable) in order to advance the findability, reproducibility and reuse of research results. Besides the research data, all the processing steps on these data – as basis of scientific publications – have to be available, too.

For good scientific practice, the resulting research software should be both open and adhere to the FAIR principles to allow full repeatability, reproducibility, and reuse. As compared to research data, research software should be both archived for reproducibility and actively maintained for reusability.

The FAIR data principles do not require openness, but research software should be open source software. Established open source software licenses provide sufficient licensing options, such that it should be the rare exception to keep research software closed.

We review and analyze the current state in this area in order to give recommendations for making research software FAIR and open.

Keywords: FAIR principles, research software, open source software

ACM CCS: Software and its engineering → Software creation and management → Collaboration in software development → Open source model, Information systems → Information retrieval → Retrieval tasks and goals

***Corresponding author: Wilhelm Hasselbring,**
Christian-Albrechts-Universität zu Kiel, D-24098 Kiel, Germany, e-mail: hasselbring@email.uni-kiel.de, ORCID: https://orcid.org/0000-0001-6625-4335
Leslie Carr, Simon Hettrick, Heather Packer, Thanassis Tiropanis, University of Southampton, SO17 1TW Southampton, UK, e-mails: lac@ecs.soton.ac.uk, sjh@ecs.soton.ac.uk, hp3@ecs.soton.ac.uk, t.tiropanis@southampton.ac.uk

1 Introduction

Replicability and reproducibility are the ultimate standards by which scientific claims are judged [34]. Reproducibility and reuse of research can be improved by increasing transparency of the research process and products via an open science culture [32].

Of the variations of open science [13], in this paper, we consider the *pragmatic* and *infrastructure* views. The *pragmatic* view regards open science as a method to make research and knowledge dissemination more efficient. It thereby considers science and research as a process that can be optimized by, for instance, modularizing the process of knowledge creation, opening the scientific value chain, including external knowledge and allowing collaboration through online tools such as Github, etc. The *infrastructure* view is concerned with the technical infrastructure that enables emerging research practices on the Internet, for the most part software tools and applications, as well as (high-performance) computing systems. Examples for such infrastructures are the European Open Science Cloud EOSC [1] and the Generic Research Data Infrastructure GeRDI [43].

Research data must be FAIR (Findable, Accessible, Interoperable, and Reusable) in order to advance the findability, reproducibility and reuse of research results. Besides the research data, all the processing steps on these data – as basis of scientific publications – have to be available, too.

To review and analyze the current state in this area we first introduce research software and research software engineering in Sections 2 and 3, respectively. The FAIR principles are originally intended for research *data*. Section 4 explains how the FAIR principles apply to research *software*. For findability and accessibility, approaches to software publishing are relevant, as will be discussed in Section 5. For reproducibility and reusability, so-called artifact evaluation serves as a review mechanism, see Section 6. Our recommendations to make research software FAIR and open are summarized in Section 7. We propose the deployment of *research software observatories* in Section 8 to better support both research software retrieval and analysis. Section 9 concludes the paper with an outlook to future work.

2 Research software

Research software is software that is employed in the scientific discovery process or a research object itself. Computational science (also scientific computing) involves the development of research software for model simulations and data analytics to understand natural systems answering questions that neither theory nor experiment alone are equipped to answer. Computational science is a multidisciplinary field lying at the intersection of mathematics and statistics, computer science, and core disciplines of science and research.

Many researchers spend significant time creating and contributing to software, an activity which is currently not well represented in the scholarly record. This situation creates some problems:

1. Trust in research relies on the peer review system but without ready access to the software used to perform a given experiment or analysis, it is difficult for readers to check a paper's validity.
2. Lack of access to the software underlying research makes it significantly more difficult to build new research results on top of existing ones: it is difficult to 'stand on the shoulders of giants'.
3. Promotion and hiring in academic research are highly dependent on building a portfolio of well-cited papers, and researchers whose main work is software development often have fewer research papers published.

Publishing research software as open source is an established practice in science; a popular open source platform is GitHub [5]. Researchers also disseminate the code and data for their experiments as virtual machines on repositories such as DockerHub [10]. Open source software practices have enabled the more general open science agenda. Open Science principles affect the research life-cycle, in the way science is performed, its results – including software – published, assessed, discovered, and monitored, as will be discussed in the following section.

3 Research software engineering for sustainable research software

Despite the increasing importance of research software to the scientific discovery process, well-established software engineering practices are rarely adopted in computational science [24], but the computational science community has started to appreciate that software engineering is central to any effort to increase its research software productivity [37]. Computer science, in particular software engineering, may help with reproducibility and reuse to advance computational science.

The above-mentioned pragmatic and infrastructure views of open science require software engineering to enable *sustainable* research software. We need proper research software *engineering* for obtaining sustainable research software. Research Software Engineers (RSE) combine an intimate understanding of research with expertise in software engineering [36]. It is a relatively new role in academia, but a highly popular one that has shown significant growth in countries around the world [35]. The RSE community has displayed a particular interest in adopting and promoting open science, and as such are perfectly placed to help researchers adopt FAIR and open software practices.

One of the biggest obstacles to making research software sustainable is ensuring appropriate credit and recognition for researchers who develop and maintain such software. The goal is to address the challenges that researchers face regarding *sustainable* research software. It is also essential to sustain software by sustaining its communities (researchers, developers, maintainers, managers, active users). If we would achieve that software citations appear somehow very close to publication citations, this could help with giving appropriate credit and recognition for researchers who develop and maintain research software. Approaches to software citation and software observatories may promote this, as we will discuss below.

4 The FAIR principles for research software

The FAIR principles are originally intended to make *data* findable, accessible, interoperable, and reusable [48]. However, for open science it is essential to publish research software in addition to research data. Extended to research software, the FAIR principles can be summarized as follows:

Findable: The first step in (re)using data and software is to find it.

Accessible: Once the user finds the required data and software, she or he needs to know how to access it, possibly including authentication and authorization if data is involved.

Interoperable: The data and software often need to be integrated with other data and software.

Reusable: For reusability, metadata, data and software should be well-described such that they can be reused, combined and extended in different settings.

Some communities also use source code itself as data. For example, the Mining Software Repositories community analyzes the rich data available in software repositories to uncover interesting information about software systems and projects [19]. Data from GitHub, Stackoverflow etc. is harvested into repositories such as GHTorrent to be employed in research [27]. Thus, the FAIR principles can also be applied to software, which can be stored and treated as data.

However, at present research software is typically not published and archived using the same practices as FAIR data, with a common vocabulary to describe the artifacts with metadata and in a citable way with a persistent identifier. GitHub is not a platform for archival publishing. Zenodo supports archiving and publishing snapshots from GitHub with persistent DOIs [16], however, it remains a great challenge to collect, preserve, and share all the software source code. Research software is the result of creative work that can continue to evolve over time. In general, software must be continuously maintained to function.

Computer science and software engineering play an important role in the implementation of the FAIR principles, which usually have a focus on helping other disciplines to be FAIR. However, computer science research itself is often also based on software; thus, computer science research software should also consider the FAIR principles.

5 Approaches to software publishing

For findability and accessibility, approaches to software publishing are relevant. Traditionally, the scientific paper is central to the research communication process. Guidelines for authors define what aspects of the research process should be made available to the community to evaluate, critique, reuse, and extend. Scientists recognize the value of transparency, openness, and reproducibility. However, it remains unclear, how this may be achieved with software.

Various journals allow one to add supplementary material to an article, which may include software. Such additional material is usually just put into zip archives, whose content is neither reviewed nor further explained with metadata information or other documentation. Thus, this may fail to promote reuse.

More publishing-oriented practices were also explored. For instance, Elsevier conducted the "Executable Paper Grand Challenge" to enhance how scientific information is used and communicated, addressing both computational science and computer science [15]. Several projects presented their concepts of "Executable Papers" which were published in the corresponding conference proceedings. The example paper [30] from this competition uses literate programming to present a Curry program within the paper. Thus, it contains the complete concise source code of their software application, which is directly executable, together with sufficient documentation to be understandable.

Research software may also be published in software journals such as the Journal of Open Source Software [33], the Journal of Open Research Software [40] or Software Impacts [12] but this is rarely adopted in computer science.

Some research communities are also building online platforms for sharing research software services. The SoBigData Lab [39], for instance, provides a cloud service for data analytics, with a focus on social mining research. OceanTEA provides an online service for analyzing ocean observation data [22]. The integrated toolchain LabPal for running, processing, and including the results of computer experiments in scientific publications is presented in [18]. The tool Qresp for curating, discovering and exploring reproducible scientific papers is presented in [17]. Generic services such as BinderHub [44] and Code Ocean [7] support online execution of reproducible code.

6 Artifact evaluation as a review mechanism

For reproducibility and reusability, so-called artifact evaluation may serve as a review mechanism. The code quality of research software often is a hindrance for reuse. For example, there are typically no tests, documentation is often lacking, and the code does not usually adhere to any coding standards. This is not necessarily caused by the scientist's bad work, but rather it is the natural result of what scientists are judged on, namely the scientific quality of the papers they put out, as opposed to the quality of software that enables such papers.

To address these issues, several ACM conferences initiated artifact evaluation processes, in which supplementary material is part of the review process [28]. The ACM distinguishes between repeatability (same team, same experimental setup), replicability (different team, same experimental setup), reproducibility (different team, different experimental setup), or reusability (artifacts are carefully documented and well-structured to the extent that reuse and repurposing are facilitated) of research artifacts [4].

Artifacts can be software systems, scripts used to run experiments, input datasets, data collected in experiments, or scripts used to analyze results. Artifact evaluation processes help to check their quality via peer review. In some subdisciplines of Computer Science these artifact evaluation processes have been established: Databases (ACM SIGMOD), Software Engineering (ACM SIGSOFT) and Programming Languages (ACM SIGPLAN), see [28]. SIGMOD calls the process *reproducibility evaluation* and also offers the 'Most Reproducible Paper Award'. The Super Computing Conference Series introduced a reproducibility initiative with an artifact evaluation process in 2016 [42]. However, some subdisciplines of Computer Science are still discussing whether they should adopt the artifact evaluation process. Such a subdiscipline is Information Retrieval (ACM SIGIR), which started an initiative to implement the ACM artifact review and badging process [14].

Recently, the Empirical Software Engineering journal initiated an open science initiative including an artifact evaluation process [31]. Once a manuscript gets *minor revision*, the authors are encouraged to prepare a replication package. When the manuscript gets accepted, the authors are invited to submit the replication package for evaluation.

Childers and Chrysanthis [6] examine how artifact evaluation has incentivized authors, and whether the process is having a measurable impact. They observe a statistical correlation between successfully evaluated artifacts and higher citation counts of the associated papers. This correlation does not imply a cause-and-effect conclusion, but the hypothesis is that authors who participate in artifact evaluations for whatever reason may have a tendency to be more active and visible in the community.

It is important to appreciate that the artifact evaluation process requires significant effort and expertise, similar to the review process for scientific papers. To address these challenges, program chairs and artifact evaluation chairs usually appoint different committees: members of program committees are mostly senior researchers while members of artifact evaluation committees are mostly early-stage researchers such as postdocs and experienced PhD students. Senior researchers contribute their long-term research experience and early-stage researchers contribute their knowledge of state-of-the-art software engineering practices.

7 Recommendations to make research software FAIR and open

Publishing research software in an archival repository is currently not common in all areas of science. Research software is usually managed in GitHub or similar platforms, where it can be maintained and re-used, but not published for scientific reward and proper citation. An approach to addressing these issues is by enabling and standardizing citation of software. Software citation brings the effort of software development into the current model of academic credit, while simultaneously enhancing the scholarly record by linking together software with publications, datasets, methods, and other research objects. Therefore, our recommendations along to the FAIR principles are the following:

For findability, challenges to be addressed for FAIR publication of research software are methods for software citation and software retrieval. To support findability, computer science sub-disciplines may adopt approaches that are currently under exploration for research software in general. However, appropriate software metadata remains a great challenge.

Authors sometimes want their users to cite something other than the piece of software directly. Examples include citing a paper that introduces the software, a published software manual or book, a 'software paper' created specifically as a citation target, or a benchmarking paper.

However, there exists guidelines for software citation and identification [38], and already some metadata standards for software citation exist [26]:

– The Citation File Format (CFF) is a human- and machine-readable file format in YAML which provides citation metadata for software [11].
– A CodeMeta instance file describes the metadata associated with a software object using JSON's linked data (JSON-LD) notation [3].

The CiteAs.org online service links between software repositories and their requested citations, exploiting the above standards. What is missing, are search engines that exploit this metadata and, more importantly, widespread annotation of research software with citation information.

For accessibility, software artifacts should be published with preservation in mind. GitHub, for example, does not directly support the preservation of software "snapshots" which were used to achieve some research results. This may, for instance be achieved via taking a snapshot from GitHub to be archived on Zenodo.org [16]:

– GitHub serves for use, reuse, and active involvement of researchers.
– Zenodo serves for archival and reproducibility of published research results.

An open question is whether computer science research needs its own discipline-specific data repository and whether the combination of GitHub and Zenodo is sufficient.

A snapshot that has been archived at Zenodo can help with reproducing a published research result, but it represents only the state of a possibly outdated software at this time. Concurrent version control systems such as Git (for which GitHub is a development platform) also allow tagging of releases. Such tags allow you to identify specific release versions of your code, which may also support reproducibility of some specific research result.

The Software Heritage archive could be another option for software preservation [9]. However, be aware that long-term preservation of software is difficult and expensive. It will be a challenge archiving, for instance, complete virtual machines or Docker images in a way that we can execute them even after decades. Approaches such as CernVM intend to address these challenges [47].

For interoperability, research software engineers should adhere to established software and data standards allowing for interoperable software components [20]. Proper interface definitions in modular software architectures are essential for interoperable research software components.

Software virtualization techniques such as Docker containers and online services help to support portability, and thus interoperability and reusability across platforms. To achieve this, researchers must be competent and willing to learn to provide their software in executable ways for whatever infrastructure they would like to have impact via reuse through other researchers.

Artifact evaluation processes may also support interoperability, if the reviewers take this concern into account.

For reusability, artifact evaluation processes review replicability and reproducibility and, if successful, reusability of research software. This way, the reusability of research software may be improved significantly.

From a software engineering point of view, modular software architectures allow for reusing parts of research software systems [21]. So far, many research software systems are not structured in a modular architecture, what should be improved in the future. Domain-specific languages may also help with the comprehensibility and modularity of research software [23].

It is vital for reusability to follow good software engineering practices to ensure that the software can be built on by others [8]. Adequate documentation is important, but so are engineering practices such as providing testing frameworks and test data for continuous integration to ensure that future adaptations can be tested to ensure that they work correctly.

As mentioned above, software virtualization techniques also support reusability across platforms. However, most current technologies and platforms are not yet sufficient to enable reusability, e. g. for a complex, steadily growing C++ research prototype running in a specialized HPC environment of some university. To improve this situation, more innovations on the infrastructure level are required. It may be useful to distinguish between Software-as-Code (e. g., via GitHub) and Software-as-a-Service (e. g., via some online cloud service on which the software is executed, such as BinderHub [44]).

8 Observatories for FAIR and open research software

Based on our experience with analyzing the relationships of research software and research publications, we propose the deployment of *Research Software Observatories* to better support both research software retrieval and analysis; thus FAIR research software. Discovery and analysis of data resources have been considered in the conceptualization of web observatories [46] and later in data observatories [45]. A data observatory is a catalogue of data resources and of observations (analytic applications) on those resources. Data observatories envisage decentralized, interoperable catalogues of data resources, hosted by different organizations.[1] Thus, data observatories are dis-

[1] Reference implementations have already emerged; e. g. https://github.com/webobservatory/

Table 1: Recommendations for FAIR Research Software.

FAIR Principle	Recommended Measure
Findability	Provide software metadata to improve software retrieval
	Use software citation to allow for proper reference and reward
	Employ research software observatories which may serve as retrieval service
Accessibility	Use software development platforms such as GitHub for code cloning
	Use repositories such as Zenodo to access archived software versions
	Use research software observatories as dedicated repository services
Interoperability	Provide proper interface definitions in modular software architectures
	Conform to established software standards
	Use software virtualization techniques for portability
	Participate in artifact evaluation processes to evaluate interoperability
Reusability	Use software development platforms such as GitHub for active involvement
	Build modular software architectures to allow for reusing parts of research software
	Use domain-specific languages for comprehensibility and modularity of research software
	Follow good software engineering practices to achieve high software quality
	Use software virtualization techniques such as Docker to support reusability across platforms
	Use software-as-a-service platforms such as BinderHub for immediate execution
	Use research software observatories for online analytics
	Participate in artifact evaluation processes to evaluate reusability

tributed, federated collections of datasets and tools for analyzing data, each with their own user community. Decentralization in this sense can provide for agile architectures in ways that centralized, one-size-fits-all solutions cannot.

In the context of open science and research software, research software observatories can be considered in three different ways. First, in terms of describing a research software observatory for FAIR and open research software, that will allow scientists to share software and observations on the status of this software. A research software observatory could support open science research and encourage best practice among research communities. Second, one could consider the research software used for processing scientific data and producing observations (analytics) in ways that respect the FAIR and open principles. Third, the opportunities and challenges of cataloging research software with appropriate citation links in observatories can be explored. Research software observatories need to support metadata for research software classification and citation to further empower researchers to find, access and reuse relevant and interoperable research software.

Related to such research software observatories is the *Research Software Directory*, which is a content management system for research software that aims to improve the findability, citability, and reproducibility of the software packages advertised in it, while enabling a qualitative assessment of their impact [41]. Collberg & Proebsting [8] studied the extent to which computer systems researchers share their code and data. Their focus is on re-building the research software for repeatability and on sharing contracts, not FAIR and open publishing. Lamprecht et al. [29] propose that research software and its associated metadata should be included in a searchable software registry. Research software observatories could serve this purpose.

9 Conclusions and future work

In this paper, we review and analyze the current state for publishing research software to give recommendations for making research software FAIR and open. For findability and accessibility, approaches to appropriate software publishing and citation aims at findability and accessibility. Artifact evaluation serves as a review mechanism to improve reproducibility and reusability. Research software observatories support research software retrieval and analysis. Table 1 summarizes our recommendations for FAIR research software.

Compared to research data, research software should be both archived for reproducibility and actively maintained for reusability. The combination of Zenodo (for archival and reproducibility) and GitHub (for maintenance and reuse) may be used to achieve this. Furthermore, research software should be open source software. Established open source software licenses [2] provide adequate licensing options such that there is no need to keep research software closed. In the vast majority of cases some existing license will be appropriate. For research data this is different. Research data may, for instance, be subject to privacy regulations. Thus, the FAIR *data* principles do not require openness, but accessibility that might include authentication and authorization. However, for research *software*, openness is to be expected [25]. Only in exceptional cases and for very good reasons should research software be closed.

Reproducibility and reusability are essential for good scientific practice. Future work should address the definition and establishment of appropriate metadata for citing both software code and software services. Such metadata could make research software also better searchable and discoverable. Research software observatories may provide such services for software retrieval and analysis.

Modularity is essential for maintainability, scalability and agility, but also for reusability. We suggest to further establish the concept of artifact evaluation to ensure the quality of published artifacts for better reusability.

Proper research software engineering enables reproducibility and reusability of research software in computational science. However, software engineering should also help software engineering and computer science research itself to support replicability and reproducibility of research software that is used in computer science experiments. This way, we may achieve FAIR and open computer science and software engineering research.

Acknowledgment: The presented work was conducted while the first author was on sabbatical leave at the University of Southampton, UK.

References

1. Paul Ayris, Jean-Yves Berthou, Rachel Bruce, Stefanie Lindstaedt, Anna Monreale, Barend Mons, Yasuhiro Murayama, Caj Södergård, Klaus Tochtermann, and Ross Wilkinson. *Realising the European Open Science Cloud*. European Union, Luxembourg, 2016. doi:10.2777/940154.
2. Miriam Ballhausen. Free and open source software licenses explained. *Computer*, 52(06):82–86, June 2019. doi:10.1109/MC.2019.2907766.
3. Carl Boettiger. Generating CodeMeta metadata for R packages. *Journal of Open Source Software*, 2(19):454, 2017. URL: https://codemeta.github.io/, doi:10.21105/joss.00454.
4. Ronald F. Boisvert. Incentivizing reproducibility. *Communications of the ACM*, 59(10):5, September 2016. URL: https://www.acm.org/publications/policies/artifact-review-badging, doi:10.1145/2994031.
5. Hudson Borges, Andre Hora, and Marco Tulio Valente. Understanding the factors that impact the popularity of GitHub repositories. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 334–344, October 2016. doi:10.1109/ICSME.2016.31.
6. Bruce R. Childers, and Panos K. Chrysanthis. Artifact evaluation: Is it a real incentive? In *IEEE 13th International Conference on e-Science*, pages 488–489, October 2017. doi:10.1109/eScience.2017.79.
7. Code Ocean. Research collaboration platform, 2019. URL: https://codeocean.com/.
8. Christian Collberg, and Todd A. Proebsting. Repeatability in computer systems research. *Communications of the ACM*, 59(3):62–69, February 2016. doi:10.1145/2812803.
9. Roberto Di Cosmo, and Stefano Zacchiroli. Software Heritage: Why and How to Preserve Software Source Code. In *iPRES 2017 – 14th International Conference on Digital Preservation*, September 2017. URL: https://hal.archives-ouvertes.fr/hal-01590958.
10. Docker Inc. DockerHub: Build and Ship any Application Anywhere, 2019. URL: https://hub.docker.com/.
11. Stephan Druskat, Neil Chue Hong, Robert Haines, and James Baker. Citation File Format (CFF) – Specifications, August 2018. URL: https://citation-file-format.github.io, doi:10.5281/zenodo.1003149.
12. Elsevier. Software Impacts, 2019. URL: https://www.journals.elsevier.com/software-impacts/.
13. Benedikt Fecher, and Sascha Friesike. Open science: One term, five schools of thought. In *Opening Science: The Evolving Guide on How the Internet is Changing Research, Collaboration and Scholarly Publishing, chapter 2*, pages 17–47. Springer International Publishing, Cham, 2014. doi:10.1007/978-3-319-00026-8_2.
14. Nicola Ferro, and Diane Kelly. SIGIR initiative to implement ACM artifact review and badging. *SIGIR Forum*, 52(1):4–10, June 2018. doi:10.1145/3274784.3274786.
15. Ann Gabriel, and Rebecca Capone. Executable paper grand challenge workshop. *Procedia Computer Science*, 4:577–578, 2011. doi:10.1016/j.procs.2011.04.060.
16. GitHub. Making Your Code Citable, 2019. URL: https://guides.github.com/activities/citable-code/.
17. Marco Govoni et al. Qresp, a tool for curating, discovering and exploring reproducible scientific papers. *Scientific Data*, 6, 2019. doi:10.1038/sdata.2019.2.
18. Sylvain Halle, Raphael Khoury, and Mewena Awesso. Streamlining the inclusion of computer experiments in a research paper. *Computer*, 51(11):78–89, November 2018. doi:10.1109/MC.2018.2876075.
19. Ahmed E. Hassan. The road ahead for mining software repositories. In *2008 Frontiers of Software Maintenance*, pages 48–57, September 2008. URL: http://www.msrconf.org/, doi:10.1109/FOSM.2008.4659248.
20. Wilhelm Hasselbring. The role of standards for interoperating information systems. In *Information Technology Standards and Standardization: A Global Perspective*, pages 116–130. IGI Global, Hershey, PA, 2000. doi:10.4018/978-1-878289-70-4.ch008.
21. Wilhelm Hasselbring. Software architecture: Past, present, future. In *The Essence of Software Engineering*, pages 169–184. Springer International Publishing, Cham, 2018. doi:10.1007/978-3-319-73897-0_10.
22. Arne Johanson, Sascha Flögel, Christian Dullo, and Wilhelm Hasselbring. OceanTEA: exploring ocean-derived climate data

using microservices. In *Proceedings of the Sixth International Workshop on Climate Informatics (CI 2016)*, pages 25–28, September 2016. doi:10.5065/D6K072N6.

23. Arne Johanson, and Wilhelm Hasselbring. Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment. *Empirical Software Engineering*, 22(4):2206–2236, August 2017. URL: http://rdcu.be/urXK, doi:10.1007/s10664-016-9483-z.
24. Arne Johanson, and Wilhelm Hasselbring. Software engineering for computational science: Past, present, future. *Computing in Science & Engineering*, 20(2):90–109, March 2018. doi:10.1109/MCSE.2018.021651343.
25. Daniel S. Katz, and Neil P. Chue Hong. FAIR is not fair enough, particularly for software citation, availability, or quality. *AGU Fall Meeting Abstracts*, December 2018. URL: http://adsabs.harvard.edu/abs/2018AGUFMIN41A..02K.
26. Daniel S. Katz, and Neil P. Chue Hong. Software citation in theory and practice. In *Mathematical Software – ICMS 2018*, pages 289–296, Springer International Publishing, Cham, 2018. doi:10.1007/978-3-319-96418-8_34.
27. Zoe Kotti, and Diomidis Spinellis. Standing on shoulders or feet? The usage of the MSR data papers. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 565–576, 2019. doi:10.1109/MSR.2019.00085.
28. Shriram Krishnamurthi, and Jan Vitek. The real software crisis: Repeatability as a core value. *Communications of the ACM*, 58(3):34–36, March 2015. doi:10.1145/2658987.
29. Anna-Lena Lamprecht, Leyla Garcia, Mateusz Kuzak, Carlos Martinez, Ricardo Arcila, Eva Martin Del Pico, Victoria Dominguez Del Angel, Stephanie van de Sandt, Jon Ison, Paula Andrea Martinez, Peter McQuilton, Alfonso Valencia, Jennifer Harrow, Fotis Psomopoulos, Josep Ll. Gelpi, Neil Chue Hong, Carole Goble, and Salvador Capella-Gutierrez. Towards FAIR principles for research software. *Data Science*, 1–23, November 2019. doi:10.3233/DS-190026.
30. Steffen Mazanek, and Michael Hanus. Constructing a bidirectional transformation between BPMN and BPEL with a functional logic programming language. *Journal of Visual Languages & Computing*, 22(1):66–89, 2011. doi:10.1016/j.jvlc.2010.11.005.
31. Daniel Méndez Fernández, Martin Monperrus, Robert Feldt, and Thomas Zimmermann. The open science initiative of the Empirical Software Engineering journal. *Empirical Software Engineering*, 24(3):1057–1060, June 2019. URL: https://github.com/emsejournal/openscience/, doi:10.1007/s10664-019-09712-x.
32. Brian A. Nosek et al. Promoting an open research culture. *Science*, 348(6242):1422–1425, 2015. doi:10.1126/science.aab2374.
33. Open Source Inititative. The Journal of Open Source Software, 2019. URL: https://joss.theoj.org/.
34. Roger D. Peng. Reproducible research in computational science. *Science*, 334(6060):1226–1227, 2011. doi:10.1126/science.1213847.
35. Olivier Philippe, Martin Hammitzsch, Stephan Janosch, Anelda van der Walt, Ben van Werkhoven, Simon Hettrick, Daniel S. Katz, Katrin Leinweber, Sandra Gesing, Stephan Druskat, Scott Henwood, Nicholas R. May, Nooriyah P. Lohani, and Manodeep Sinha. softwaresaved/international-survey: Public release for 2018 results, March 2019. doi:10.5281/zenodo.2585783.
36. Research Software Engineers Association. Who is a Research Software Engineer?, 2019. URL: https://rse.ac.uk/who/.
37. Ulrich Rüde, Karen Willcox, Lois Curfman McInnes, and Hans De Sterck. Research and education in computational science and engineering. *SIAM Review*, 60(3):707–754, 2018. doi:10.1137/16M1096840.
38. Arfon M. Smith, Daniel S. Katz, and Kyle E. Niemeyer. Software citation principles. *PeerJ Computer Science*, 2, September 2016. URL: https://www.force11.org/software-citation-principles, doi:10.7717/peerj-cs.86.
39. SoBigData. European research infrastructure for big data and social mining, 2019. URL: http://sobigdata.eu.
40. Software Sustainability Institute. The Journal of Open Research Software, 2019. URL: https://openresearchsoftware.metajnl.com/.
41. Jurriaan H. Spaaks, Jason Maassen, Tom Klaver, Stefan Verhoeven, Pushpanjali Pawar, Willem van Hage, Lars Ridder, Lode Kulik, Tom Bakker, Vincent van Hees, Laurens Bogaardt, Adrianne Mendrik, Bram van Es, Jisk Attema, Elena Ranguelova, and Rob van Nieuwpoort. Research Software Directory, March 2019. URL: https://github.com/research-software-directory/, doi:10.5281/zenodo.2609141.
42. Super Computing Conference Series. SC Reproducibility Initiative, 2018. URL: https://sc18.supercomputing.org/submit/sc-reproducibility-initiative/.
43. Nelson Tavares de Sousa, Wilhelm Hasselbring, Tobias Weber, and Dieter Kranzlmüller. Designing a generic research data infrastructure architecture with continuous software engineering. In *Software Engineering Workshops 2018*, pages 85–88, March 2018. URL: http://ceur-ws.org/Vol-2066/cse2018paper03.pdf.
44. The Jupyter Team. BinderHub, 2019. URL: https://binderhub.readthedocs.io/.
45. Thanassis Tiropanis. Data observatories: decentralised data and interdisciplinary research. In *Internet y Ciencia: Análisis Desde la Complejidad Estructural y Dinámica*, Ferrol, Spain, March 2019. URL: https://eprints.soton.ac.uk/428200/.
46. Thanassis Tiropanis, Wendy Hall, James Hendler, and Christian de Larrinaga. The Web Observatory: A middle layer for broad data. *Big Data*, 2(3):129–133, September 2014. doi:10.1089/big.2014.0035.
47. Dag Toppe Larsen, Jakob Blomer, Predrag Buncic, Ioannis Charalampidis, and Artem Haratyunyan. Long-term preservation of analysis software environment. *Journal of Physics: Conference Series*, 396(3):1–8, December 2012. doi:10.1088/1742-6596/396/3/032064.
48. Mark D. Wilkinson et al. The FAIR guiding principles for scientific data management and stewardship. *Scientific Data*, 3, March, 2016. URL: https://www.go-fair.org/fair-principles/, doi:10.1038/sdata.2016.18.

Bionotes

Wilhelm Hasselbring
Christian-Albrechts-Universität zu Kiel,
D-24098 Kiel, Germany
hasselbring@email.uni-kiel.de

Prof. Dr. Wilhelm Hasselbring is a full professor of software engineering in the Department of Computer Science at Kiel University.

Leslie Carr
University of Southampton, SO17 1TW
Southampton, UK
lac@ecs.soton.ac.uk

Prof. Dr. Leslie Carr is a full professor of web science in the Department of Electronics and Computer Science at the University of Southampton.

Simon Hettrick
University of Southampton, SO17 1TW
Southampton, UK
sjh@ecs.soton.ac.uk

Prof. Dr. Simon Hettrick is deputy director of UK's Software Sustainability Institute and a full professor in the Department of Electronics and Computer Science at the University of Southampton.

Heather Packer
University of Southampton, SO17 1TW
Southampton, UK
hp3@ecs.soton.ac.uk

Dr. Heather Packer is a New Frontier fellow in the Department of Electronics and Computer Science at the University of Southampton.

Thanassis Tiropanis
University of Southampton, SO17 1TW
Southampton, UK
t.tiropanis@southampton.ac.uk

Prof. Dr. Thanassis Tiropanis is an associate professor in the Department of Electronics and Computer Science at the University of Southampton.