Reproducibility Debt: Challenges and Future Pathways

Zara Hassan
Australian National University
Canberra, Australia
zara.hassan@anu.edu.au

Christoph Treude
Singapore Management University
Singapore, Singapore
ctreude@smu.edu.sg

Michael Norrish
Australian National University
Canberra, Australia
michael.norrish@anu.edu.au

Graham Williams
Australian National University
Canberra, Australia
graham.williams@anu.edu.au

Alex Potanin
Australian National University
Canberra, Australia
alex.potanin@anu.edu.au

ABSTRACT

Reproducibility of scientific computation is a critical factor in validating its underlying process, but it is often elusive. Complexity and continuous evolution in software systems have introduced new challenges for reproducibility across a myriad of computational sciences, resulting in growing debt. This requires a comprehensive domain-agnostic study to define and asses Reproducibility Debt (RpD) in scientific software, thus uncovering and classifying all underlying factors attributed towards its emergence and identification i.e., *causes and effects*. Moreover, an organised map of prevention strategies is imperative to guide researchers for its proactive management. This vision paper highlights the challenges that hinder effective management of RpD in scientific software, with preliminary results from our ongoing work and an agenda for future research.

CCS CONCEPTS

• **Software and its engineering** → **Open source model**; **Reusability**; **Traceability**.

KEYWORDS

Reproducibility Debt, Technical Debt, Scientific Software, Scientific Computing, Computational Software

ACM Reference Format:
Zara Hassan, Christoph Treude, Michael Norrish, Graham Williams, and Alex Potanin. 2024. Reproducibility Debt: Challenges and Future Pathways. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE Companion '24), July 15–19, 2024, Porto de Galinhas, Brazil.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3663529.3663778

1 INTRODUCTION

Technical Debt (TD) is defined as a "collection of design or implementation constructs that are beneficial in the short-term but set up a technical context that can make future changes more costly or impossible" [2], meaning that developers adopt practices that are expedient in the short term but compromise the overall quality of software [31, 35]. Multiple TD types have been identified, along with their occurrences (namely, smells) and corresponding management strategies [2, 29]. Scientists implementing scientific software[1] can also incorporate TD; even though they are domain experts, they often lack knowledge of required Software Engineering (SE) practices to develop high-quality software [18, 19, 27]. As a result, TD is not exclusive to traditional and commercial software but also manifests in scientific software, where its adverse effects can become more concerning. This added complication is caused by scientific software's purpose–to produce research outcomes in a myriad of disciplines, which form the basis for future scientific research [37].

In scientific research, reproducibility is an essential instrument allowing researchers to continue and extend previously published results [33]. In this context, *Nature* surveyed 'Reproducibility in Scientific Research', revealing that more than 70% of researchers were unable to reproduce published results and concluded it as a 'Reproducibility Crisis' [3]. The impact of the crisis also extends to computational space and raises questions on the reliability and credibility of scientific software [18, 33]. In scientific software, the "inability to reproduce results can often be attributed to technical issues and challenges around reliably recreating the full computational workflow from the original analysis" [6, 15, 26] which we have conceptualised as Reproducibility Debt (RpD).

We hereby argue that scientific software are intrinsically liable to inherit RpD. Therefore, using scientific software under the assumption that it has stable dependencies, consistent response to data, and adequate tests (reportedly proven to be of poor quality [19, 36]), poses a threat to the validity of scientific results. Our vision is that thorough research is required to define and assess RpD and other aspects of reproducibility in scientific software from a SE perspective. In this context, this paper highlights the challenges that hinder the management of RpD in scientific software projects and provides an agenda for future research.

2 CHALLENGES

This section provides an overview of the identified challenges, discussed under four main headings.

This work is licensed under a Creative Commons Attribution 4.0 International License.

FSE Companion '24, July 15–19, 2024, Porto de Galinhas, Brazil
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0658-5/24/07
https://doi.org/10.1145/3663529.3663778

[1]"End-user application software that is written to achieve scientific objectives (e.g., Climate models), tools that support writing code that expresses a scientific model and the execution of scientific code, research software written to publish papers, production software written for real users" [19]

2.1 Diverse Definitions of Reproducibility

The importance of reproducibility is acknowledged across all computational sciences. Therefore, scientists across domains have defined reproducibility with respect to their discipline-specific needs, resulting in diverse definitions. For instance, according to Drummond [9] *"reproducibility requires changes"*—obtaining the same results from a markedly different experiment. Two slightly more refined definitions interpreted reproducibility as *"the ability of a study to be reproduced, in whole or in part, by an independent research team."* [13], and *"...the ability of an independent team to recreate the same qualitative results"* [28]. In a concise approach to defining elements needed to achieve reproducibility, Barba [4] briefly stated that *"same data + same method = same results"*. In a more polished definition, Peng [26] stated that *"reproducibility is a continuous variable ranging from only a paper describing an experiment being shared, to the linked executable code and data being shared along with the paper"*. According to Essawy et al. [10], *"reproducible software" in hydro-logic modelling requires sharing both the software and data, but also their associated metadata*—a detail omitted in other definitions.

Conclusion. The lack of consensus on a single definition has caused a negative impact towards building a common understanding among scientists and researchers about what reproducibility entails. Thus, hindering the development of a unified approach towards its effective management; also acknowledged by the Federation of American Societies for Experimental Biology (FASEB [11]) and the National Academy of Sciences Committee on Reproducibility and Replicability of Science (NASEM [25]) who stated that "lack of consensus on a single definition [is] among the causes of the [reproducibility] crisis".

2.2 Domain-Specific Strategies

The first step in TD management is the identification of TD items i.e., a list of issues or bad practices which create debt [16, 21, 29]. In the case of RpD, various scientists and researchers have made domain-specific efforts regarding reproducibility issues. This has led to a myriad of positions highlighting various factors influencing reproducibility i.e., missing dependencies, inadequately documented workflows, poorly written code, version upgradation in programming languages, libraries, operating systems, and non-deterministic order of execution in parallel systems [5, 6, 15, 22, 32]. However, most of the identified issues and proposed solutions are domain-specific, resulting in isolated proposals that are either not applicable or unacknowledged in other disciplines.

For instance, in one approach, existing SE tools and technologies are suggested as a solution to ensure the reproducibility of scientific workflows i.e., the use of containers, literate programming notebooks, version control, and open-source repositories [6, 15, 17, 33, 34]. In a more sophisticated approach, specialised platforms and infrastructures were developed by researchers, to meet their problem and disciplines' specific needs. For instance, the *Whole Tale* [5] platform facilitates the linkage of data and codes with publications by strengthening the three layers of scholarly publication i.e., scholarly process, data, and computational analysis. *N3phele* [8] is a cloud-based solution for managing the reproducibility of complex biological data processing pipelines. *Osiris* [38] is an automated approach to make Jupyter notebooks reproducible. *PyDFix* [24] detects and fixes dependency errors in Python builds. Maneage [1] is designed to manage data lineage and longevity issues in tools such as Docker and Jupyter Notebooks. *Invarient framework* [23] enables reproducibility by capturing and preserving the dependencies and configurations used by the program.

Conclusion. Despite various attempts to highlight the underlying causes of RpD, the suggested initiatives are primarily domain-specific and lack coordination, limiting their identification, measurement and prevention. Given the broad and growing reliance on scientific software across a myriad of computational science disciplines, a comprehensive domain-agnostic study from a SE lens is imperative.

2.3 Inadequate Use of SE Practices

Scientific software is mainly developed by or under the supervision of domain engineers/scientists, less acquainted with state-of-the-art SE practices and tools. Johanson and Hasselbring [18] argued that computational scientists rarely adopt SE practices, resulting in productivity and credibility crises. Pinto et al. [27] and Vidoni [37] report that research scientists do not employ formal Requirements Engineering practices, and instead of documenting requirements, they rely on source code comments, thus incorporating TD. Heaton and Carver [14] report that scientific software developers, regardless of realising the importance of software verification, validation, and testing, can only partially adopt these practices due to limited knowledge about their application in scientific software.

We contend that the inadequate use of SE practices in scientific software development can lead to the accumulation of RpD. Scientific software often involves complex computations and data processing; therefore, researchers may prioritise short-term scientific goals, i.e., getting faster results and speeding up publications, over SE practices, i.e., detailed documentation and rigorous testing [18, 19]. This approach can lead to software that is difficult to maintain and reproduce, resulting in long-term consequences such as the lack of trust in published works, erratic comparisons to other works, and the inability to build upon others' work, grossly hindering progress in science.

Conclusion. We stand by the argument of Heaton and Carver [14] that the use of SE practices will benefit scientific software development by improving accuracy and streamlining the development process. However, there is a need to customise existing SE practices to better align with the requirements of scientific software development for their better adoption and utilisation.

2.4 Undervalued Human-Centric Aspects

Most of the literature on reproducibility focuses solely on technical aspects and provides proposals (tested or not) to resolve them [5, 6, 15] as discussed in Section 2.2. We argue that building a reproducible package should not be confined to the technical aspects only; at the same time, it should encompass human-centred considerations, institutional stances and support, and ethical protocols.

Reproducibility relies mainly on the scientist whose work is to be reproduced and is biased by an individual's perception of reproducibility and how pertinent institutions support them. For example, scientists may consider that there are no direct benefits or incentives to have their work reproduced, that the effort required to clean code and data is unwarranted, that they may lose competitive advantage over other researchers, or they may be concerned or deterred by intellectual property issues [33, 34]. Although efficient collaboration is essential for scientists who rely on scientific software, little has been achieved for policies and incentive schemes to create a culture of collaborative and reproducible research.

Conclusion. Comprehensive analysis of RpD in scientific software can not be achieved by restricting our focus to its technical and structural concerns. Since humans are the originators of every research work, human-centric aspects [7] —including cultural, psychological, and social factors—warrant detailed exploration.

3 RESEARCH OBJECTIVES

Given the highlighted challenges, we begin our study to consolidate existing knowledge on 'Reproducibility in Scientific Software' across all scientific disciplines using a Systematic Literature Review (SLR) based on the guidelines of Kitchenham and Charters [20]. The goal of the study is to: *formulate an integrated definition of 'Scientific Software Reproducibility'* and *'Characterise RpD'* based on issues (causes and effects) discussed in the existing literature. Our detailed agenda and research questions are stated in Section 4. However, preliminary results from our ongoing SLR that satisfy the following two RQs are presented in this section with a short description of the opted methodology.

(1) *How can the concept of Reproducibility be defined and characterised in the context of scientific software?*
(2) *What are the main categories of issues that contribute to RpD, and what is their relation with established TD types?*

3.1 Methodology

`Step1:` Inclusion/Exclusion Criteria were formulated based on the primary research objective to ensure the inclusion of quality literature (e.g., peer-reviewed), that discusses open science, reproducibility and/or replicability (under this name or another) of scientific software (or aspects related to it) in any discipline. Moreover, we did not exclude any publications based on the year to mitigate the risk of missing relevant papers. `Step2:` The following Search String was used to search the literature from *ACM Digital Library, IEEE Xplore, ScienceDirect, Springer, Wiley and Taylor & Francis*. To avoid publisher bias *Google Scholar* was also added as given in guidelines by Wohlin [39]:

```
(“reproducibility” OR “replicability” OR
“repeatability”) AND (“open source” OR “open
science” OR “open code”) AND (“scientific
software” OR “scientific computing” OR
“computational software”)
```

`Step3:` After collecting all primary studies (2198 papers), Zotero[2] was used to remove duplicates. `Step4:` Primary studies were selected using a three-stage filtering process applying pre-defined IECs and Quality Criteria. `Step5:` The requisite data was extracted from 214 primary studies in a spreadsheet. `Step6:` The qualitative approach to synthesise the extracted data was adopted. *(A list of selected papers, IECs, Quality Criteria and Data Extraction form can be viewed here.*[3])

3.2 Preliminary Results

3.2.1 Defining Scientific Software Reproducibility. We identified that out of 214 selected papers, 104 provided the definition of reproducibility (proposed, re-used, or discussed). A qualitative analysis of all extracted definitions was performed to identify common themes and patterns among them. We identified that the 'ability to re-perform experiment' is a prominent theme among all definitions, other four concepts that delimit reproducibility in the context of scientific software are: *"use of computational methods"*, *"using open knowledge"*, *"individual analysis and interpretations of results"*, and *"independent replication"*. Based on the identified themes, we devised the following definition of 'Scientific Software Reproducibility'.

'Scientific Software Reproducibility' is the ability to re-perform experiments using computational methods and open knowledge to obtain results that can be analysed, interpreted, and replicated independently.

3.2.2 Issues Contributing towards RpD. Identification of TD items is usually the first step in the management of TD in any software systems [12, 16, 29]. *RpD items in scientific software refer to the technical issues, sub-optimal practices or challenges within the development and usage of scientific software that can hinder the reproducibility of research results.* To characterise RpD in scientific software, we developed a high-level taxonomy of RpD items based on qualitative analysis of data obtained from selected primary studies. The proposed taxonomy provides a structured overview of issues attributed towards the emergence and identification of RpD (See Table 1).

Moreover, from the ongoing analysis of the obtained data, we identified several underlying causes and effects of RpD under each category, which includes (but are not limited to) *non-systematic data processing and analysis, unorganised data and analysis files, incomplete or selective reporting of datasets, non-Standardised data and metadata formats for storage, sharing and reuse, poorly organised code, undocumented scientific workflows, missing dependencies, inadequate data and code documentation, lack of programming and computing skills among researchers, lack of knowledge in research data management skills, and lack of trusted infrastructure for storing, processing and distributing large datasets and scientific software code.*

We also established a relationship between identified RpD items and existing TD types by concurrently analysing our emerged factors (causes and effects) and the list of situations (smells) discussed by Alves et al. [2], Rios et al. [29], Sculley et al. [32] where these debts can occur in software systems also shown in Table 1.

[2]https://www.zotero.org/

[3]https://doi.org/10.6084/m9.figshare.25771689.v2

Table 1: High level taxonomy of RpD items and their hypothesised relation with existing TD types

RpD Item	Description	Related Debt [2, 29, 32]
Data-Centric Issues	These are related to scientific data processing, storage, and dissemination.	Data, Documentation, Infrastructure
Code-Centric Issues	These are related to developing, organising, and disseminating scientific software code.	Code, Versioning, Infrastructure, Build, Test
Documentation Issues	Incomplete or unclear documentation can hinder understanding the processes required to reproduce research findings. This includes missing details about data sources, software dependencies, or specific configurations used in analysis.	Documentation, Data, Code
Infrastructure and Tools-Centric Issues	These are associated with the infrastructure, tools, and technologies used to develop and store scientific software.	Infrastructure, Build, Architecture
Versioning Issues	Arise when the version of software, code, or data used in the original research is unavailable or incompatible with other software used in replication.	Data, Versioning, Code
Human-Centric Issues	These are associated with individuals involved in scientific research, including software developers, domain researchers, reviewers, and funding organisations.	People, Cultural
Legal Issues	Refer to the intellectual property and ownership issues in open research software and data.	–

4 FUTURE AVENUES OF RESEARCH

We have formulated an integrated definition of 'Scientific Software Reproducibility' and a high-level taxonomy of RpD items to serve as a baseline to guide our future work, discussed under three main headings. The laid down research questions will be investigated by conducting SLR, surveys, and interview-based case studies, involving researchers and scientific software developers, as demonstrated by prior works to investigate TD [2, 29, 30].

4.1 Unified RpD Framework

The following research questions aim towards the development of a unified framework for the identification and management of RpD in scientific software projects.

(3) *What are the main causes of RpD in scientific software projects?*
(4) *What effects does RpD have on scientific software projects?*
(5) *How does RpD manifest in scientific software?*
(6) *What are the mitigation strategies to effectively manage RpD in scientific software?*

We intend to formulate an evolved taxonomy of RpD, identify a list of situations in which debt/smells can be found in scientific software projects, and create an organised map of activities. This results in a theoretical framework that will include specific guidelines and processes to mitigate and manage RpD (akin to the one produced by Rios et al. [30] for Documentation Debt).

4.2 Reproducibility-Oriented SE

As discussed in Section 2.3, there is a need for customised SE practices for scientific software development, thereby enforcing their full adoption and utilisation. Therefore, we propose the notion of *Reproducibility-Oriented SE, which refers to an approach to scientific software development that incorporates best practices and principles from SE and scientific research to produce transparent, reliable, and reproducible software.* Hence, our next two questions are focused on exploring SE practices for developing reproducible scientific software:

(7) *What are the key SE practices and tools supporting reproducibility, and what is their current state of adoption among computational scientists?*
(8) *What are the most effective SE practices and tools towards achieving fast results and publications?*

We intend to evolve a custom SE process model for scientific software, namely Reproducibility-Oriented Process (ROP) by concurrently analysing RQ 7 and 8, thus guiding researchers to achieve short-term benefits through improved results and long-term benefits through more maintainable and reproducible software.

4.3 Human Aspects of Reproducibility

Human aspects of SE bring attention to the psychological, social, and cultural aspects of a better software development process [7]. As stated in Section 2.4 reproducibility is a multifaceted debt directly affected by the perceptions of scientists whose work is to be reproduced. Therefore, understanding the human aspects of RpD in the scientific software development process is essential for its management. Hence, our following two research questions focus on exploring the human aspects of reproducibility:

(9) *What factors lead researchers and scientific software developers to de-prioritise reproducibility?*
(10) *How can funding agencies and institutions support researchers and developers towards Reproducibility-Oriented Research?*

We intend to develop a culture of reproducible research that needs to be nurtured by combining the efforts of the SE community, publishers of scientific research, and scientific software developers. To achieve this, we will explore already established work on human aspects of SE from a reproducibility perspective. The goal is to formulate a comprehensive set of guidelines suitable for developers and reviewers, covering all aspects of reproducibility.

REFERENCES

[1] Mohammad Akhlaghi, Raul Infante-Sainz, Boudewijn F. Roukema, Mohammadreza Khellat, David Valls-Gabaud, and Roberto Baena-Galle. 2021. Toward Long-Term and Archivable Reproducibility. *Computing in Science & Engineering* 23, 3 (2021), 82–91. https://doi.org/10.1109/mcse.2021.3072860

[2] Nicolli S.R. Alves, Thiago S. Mendes, Manoel G. de Mendonça, Rodrigo O. Spínola, Forrest Shull, and Carolyn Seaman. 2016. Identification and Management of Technical Debt: A Systematic Mapping Study. *Information and Software Technology* 70 (2016), 100–121. https://doi.org/10.1016/j.infsof.2015.10.008

[3] Monya Baker. 2016. 1,500 Scientists Lift the Lid on Reproducibility. *Nature* 533, 7604 (2016), 452–454. https://doi.org/10.1038/533452a

[4] Lorena A. Barba. 2018. Terminologies for Reproducible Research. https://doi.org/10.48550/ARXIV.1802.03311

[5] Adam Brinckman, Kyle Chard, Niall Gaffney, Mihael Hategan, Matthew B Jones, Kacper Kowalik, Sivakumar Kulasekaran, Bertram Ludäscher, Bryce D Mecum, and Jarek Nabrzyski. 2019. Computing Environments for Reproducibility: Capturing the "Whole Tale". *Future Generation Computer Systems* 94 (2019), 854–867.

[6] R. Shane Canon. 2020. The Role of Containers in Reproducibility. In *2020 2nd International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*. 19–25. https://doi.org/10.1109/CANOPIEHPC51917.2020.00008

[7] L. Fernando Capretz. 2014. Bringing the Human Factor to Software Engineering. *IEEE Software* 31, 02 (mar 2014), 104–104. https://doi.org/10.1109/MS.2014.30

[8] Nigel Cook, Dejan Milojicic, Richard Kaufmann, and Joel Sevinsky. 2012. N3phele: Open Science-as-a-Service Workbench for Cloud-based Scientific Computing. In *2012 7th Open Cirrus Summit*. 1–5. https://doi.org/10.1109/OCS.2012.30

[9] Chris Drummond. 2009. Replicability is not Reproducibility: Nor is it Good Science. In *Evaluation Methods for Machine Learning Workshop at the 26th ICML*, Vol. 1. National Research Council of Canada Montreal, Canada.

[10] Bakinam T. Essawy, Jonathan L. Goodall, Hao Xu, and Yolanda Gil. 2017. Evaluation of the OntoSoft Ontology for Describing Metadata for Legacy Hydrologic Modeling Software. *Environmental Modelling & Software* 92 (2017), 317–329. https://doi.org/10.1016/j.envsoft.2017.01.024

[11] FASEB. 2019. National Academies Releases Report on Reproducibility and Replicability in Science. https://www.faseb.org/journals-and-news/washington-update/national-academies-releases-report-on-reproducibility-and-replicability-in-science

[12] Carlos Fernández-Sánchez, Juan Garbajosa, Agustín Yagüe, and Jennifer Perez. 2017. Identification and analysis of the elements required to manage technical debt by means of a systematic mapping study. *Journal of Systems and Software* 124 (2017), 22–38. https://doi.org/10.1016/j.jss.2016.10.018

[13] Jesús M. González-Barahona and Gregorio Robles. 2012. On the Reproducibility of Empirical Software Engineering Studies Based on Data Retrieved From Development Repositories. *Empirical Software Engineering* 17, 1 (2012), 75–89. https://doi.org/10.1007/s10664-011-9181-9

[14] Dustin Heaton and Jeffrey C. Carver. 2015. Claims about the use of software engineering practices in science: A systematic literature review. *Information and Software Technology* 67 (2015), 207–219. https://doi.org/10.1016/j.infsof.2015.07.011

[15] Peter Ivie and Douglas Thain. 2019. Reproducibility in Scientific Computing. *Comput. Surveys* 51, 3 (2019), 1–36. https://doi.org/10.1145/3186266

[16] Clemente Izurieta, Ipek Ozkaya, Carolyn Budinger Seaman, Philippe B Kruchten, Robert L. Nord, Will Snipes, and Paris Avgeriou. 2016. Perspectives on Managing Technical Debt: A Transition Point and Roadmap from Dagstuhl. In *QuASoQ/TDA@APSEC*.

[17] Ivo Jimenez, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, Jay Lofstead, Carlos Maltzahn, Kathryn Mohror, and Robert Ricci. 2017. PopperCI: Automated reproducibility validation. In *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 450–455. https://doi.org/10.1109/INFCOMW.2017.8116418

[18] Arne N. Johanson and Wilhelm Hasselbring. 2018. Software Engineering for Computational Science: Past, Present, Future. *Computing in Science & Engineering* 20 (2018), 90–109. https://doi.org/10.1109/MCSE.2018.021651343

[19] Upulee Kanewala and James M. Bieman. 2014. Testing scientific software: A systematic literature review. *Information and Software Technology* 56, 10 (2014), 1219–1232. https://doi.org/10.1016/j.infsof.2014.05.006

[20] Barbara Ann Kitchenham and Stuart Charters. 2007. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Technical Report EBSE 2007-001. Keele University and Durham University Joint Report.

[21] Valentina Lenarduzzi, Francesco Lomio, Sergio Moreschini, Davide Taibi, and Damian Andrew Tamburri. 2021. Software Quality for AI: Where We Are Now?. In *Software Quality: Future Perspectives on Software Engineering Quality*, Dietmar Winkler, Stefan Biffl, Daniel Mendez, Manuel Wimmer, and Johannes Bergsmann (Eds.). Springer, Cham, 43–53.

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

[23] Haiyan Meng, Rupa Kommineni, Quan Pham, Robert Gardner, Tanu Malik, and Douglas Thain. 2015. An invariant framework for conducting reproducible computational science. *Journal of Computational Science* 9 (2015), 137–142. https://doi.org/10.1016/j.jocs.2015.04.012 Computational Science at the Gates of Nature.

[24] Suchita Mukherjee, Abigail Almanza, and Cindy Rubio-González. 2021. Fixing dependency errors for Python build reproducibility. https://doi.org/10.1145/3460319.3464797

[25] NASEM. 2019. *Reproducibility and Replicability in Science*. The National Academies Press, Washington, DC. https://doi.org/10.17226/25303

[26] Roger D. Peng. 2011. Reproducible Research in Computational Science. *Science* 334, 6060 (2011), 1226–1227. https://doi.org/10.1126/science.1213847

[27] G. Pinto, I. Wiese, and L. F. Dias. 2018. How Do Scientists Develop Scientific Software? An External Replication. In *IEEE 25th International Conference on Software Analysis, Evolution and Reengineering*. IEEE, Campobasso, Italy, 582–591.

[28] Edward Raff and Andrew L. Farris. 2022. A Siren Song of Open Source Reproducibility. https://doi.org/10.48550/ARXIV.2204.04372

[29] Nicolli Rios, Manoel Gomes de Mendonça Neto, and Rodrigo Oliveira Spínola. 2018. A Tertiary Study on Technical Debt: Types, Management Strategies, Research Trends, and Base Information for Practitioners. *Information and Software Technology* 102 (2018), 117–145. https://doi.org/10.1016/j.infsof.2018.05.010

[30] Nicolli Rios, Leonardo Mendes, Cristina Cerdeiral, Ana Patrícia F. Magalhães, Boris Perez, Darío Correal, Hernán Astudillo, Carolyn Seaman, Clemente Izurieta, Gleison Santos, and Rodrigo Oliveira Spínola. 2020. Hearing the Voice of Software Practitioners on Causes, Effects, and Practices to Deal with Documentation Debt. In *Requirements Engineering: Foundation for Software Quality: 26th International Working Conference, REFSQ 2020, Pisa, Italy, March 24–27, 2020, Proceedings* (Pisa, Italy). Springer-Verlag, Berlin, Heidelberg, 55–70. https://doi.org/10.1007/978-3-030-44429-7_4

[31] Junior Cesar Rocha, Vanius Zapalowski, and Ingrid Nunes. 2017. *Understanding Technical Debt at the Code Level from the Perspective of Software Developers*. Association for Computing Machinery, New York, NY, USA, 64–73. https://doi.org/10.1145/3131151.3131164

[32] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in Machine learning systems. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2* (Montreal, Canada) *(NIPS'15)*. MIT Press, Cambridge, MA, USA, 2503–2511.

[33] Victoria Stodden. 2010. The Scientific Method in Practice: Reproducibility in the Computational Sciences. *SSRN Electronic Journal* (2010), 4773–10. https://doi.org/10.2139/ssrn.1550193

[34] Victoria Stodden and Sheila Miguez. 2014. Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research. *Journal of Open Research Software* 2, 1 (2014), e21. https://doi.org/10.5334/jors.ay

[35] Edith Tom, Aybüke Aurum, and Richard Vidgen. 2013. An exploration of Technical Debt. *Journal of Systems and Software* 86, 6 (2013), 1498–1516. https://doi.org/10.1016/j.jss.2012.12.052

[36] Melina Vidoni. 2021. Evaluating Unit Testing Practices in R Packages. In *Proceedings of the 43rd International Conference on Software Engineering* (Spain). IEEE Press, USA, 1523–1534. https://doi.org/10.1109/ICSE43902.2021.00136

[37] M. Vidoni. 2021. Self-Admitted Technical Debt in R Packages: An Exploratory Study. In *IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE Computer Society, Los Alamitos, CA, USA, 179–189. https://doi.org/10.1109/MSR52588.2021.00030

[38] Jiawei Wang, Tzu-yang Kuo, Li Li, and Andreas Zeller. 2020. Restoring reproducibility of Jupyter notebooks. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 288–289. https://doi.org/10.1145/3377812.3390803

[39] Claes Wohlin. 2014. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering* (London, England, United Kingdom) *(EASE '14)*. ACM, New York, NY, USA, Article 38, 10 pages. https://doi.org/10.1145/2601248.2601268

Received 28-JAN-2024; accepted 2024-04-09