# Managing Reproducibility Debt in Scientific Software: A Practical Framework

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

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

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

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

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

## Abstract

Scientific software includes end-user applications, modelling tools, research software for publications, and production systems for real users. It plays a key role across various scientific disciplines by enabling large-scale computation, simulation, and data analysis. Unlike commercial software, scientific software is often developed in dynamic research environments with limited engineering practices, documentation, or testing. This makes it fragile and difficult to reproduce results, even when code and data are available, conditions in which Reproducibility Debt (RpD) accumulates. This paper presents the Reproducibility Debt Management Framework (RpD-MF), which is grounded in evidence from a systematic literature review, practitioner interviews, and a global survey. Central to the framework is a probabilistic cause-effect model that maps how technical, human, and organisational factors contribute to RpD. The framework is designed to help researchers and research software engineers identify, monitor, and prevent RpD, providing practical guidance to support sustainable and reproducible scientific software development.

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

***Keywords:*** Reproducibility, Technical Debt, Scientific Software, Reproducibility Debt Management, Computational Reproducibility

**ACM Reference Format:**
Zara Hassan, Christoph Treude, Graham Williams, Michael Norrish, and Alex Potanin. 2026. Managing Reproducibility Debt in Scientific Software: A Practical Framework. In *1st International Workshop on Software Engineering and Research Software (SERS '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3786172.3788369



*SERS '26, Rio de Janeiro, Brazil*

ACM ISBN 979-8-4007-2404-6/2026/04
https://doi.org/10.1145/3786172.3788369

## 1 Introduction

Scientific software encompasses a wide range of applications, including end-user applications and tools, modelling and simulation systems, research software developed for publications, and large-scale production systems for real users [5]. It underpins modern scientific inquiry by enabling computation, modelling, simulation, and data analysis across various disciplines. However, unlike commercial software, it is often developed in dynamic research environments with limited engineering practices, documentation, or testing [4]. As a result, researchers frequently face challenges in reproducing computational results due to software configuration, dependencies, versioning, and undocumented workflows [1, 6]. These limitations make scientific software fragile and difficult to reproduce, even when data and code are openly shared. In our previous work [2], we investigated and empirically characterised Reproducibility Debt (RpD), a type of Technical Debt (TD) that arises from accumulated human, organisational, and technical factors in the development, documentation, and dissemination of scientific software.

Building on our prior investigations on RpD [1–3], this paper presents a practical framework for managing RpD in scientific software projects. The framework is grounded in evidence collected through a sequential mixed-methods approach, including a systematic literature review (214 Primary Studies), practitioner interviews (23 Participants), and a global InsightRpD survey (59 Participants). Each study was conducted rigorously, with the findings of one stage informing the design and analysis of the next. To establish a foundation for the framework, we developed the first taxonomy of issues contributing to RpD, categorised into seven groups: data-centric, code-centric, documentation-centric, human& organisation-centric, infrastructure & tools-centric, versioning, and legal.

Across all three studies, we identified 75 causes and 110 effects of RpD. In the SLR, 37 causes and 63 effects were documented, and practitioner insights from interviews and

surveys validated 31 of these causes while adding 38 new ones. We quantified the frequency of each cause and effect, enabling the construction of probabilistic cause-effect diagrams [1], organised into the above-mentioned seven categories. The diagrams shared in our supplementary material visualise the complete landscape of RpD causes and effects, highlighting the most critical and high-risk factors. Our findings indicate that human and organisational practices are the largest contributors, followed by documentation-related issues, offering a quantitative foundation for prioritising mitigation efforts. Finally, our qualitative synthesis produced 61 mitigation strategies[2], derived from both literature and practitioner experience, encompassing all identified issue areas. Together, the taxonomies, RpD cause-effect diagrams, and mitigation strategies form the empirical foundation of the RpD-MF, introduced in the next section.

## 2 Reproducibility Debt Management Framework (RpD-MF)

Building on empirical insights from our sequential studies, we propose the RpD-MF to support systematic identification, prioritisation, monitoring, and prevention of RpD in scientific software projects. The framework helps researchers and research software engineers (RSEs) translate insights from RpD causes and effects into actionable interventions and continuous improvement. The framework is operationalised through the following five-step workflow:

***Step 1. Audit Your Scientific Workflow:*** Systematically examine all aspects of your computational workflow, including code, scripts, and data processing pipelines, to identify potential reproducibility risks. Review documentation, logs, and outputs for gaps such as missing datasets, undocumented preprocessing steps, untracked software dependencies, or inconsistent coding practices. Assess versioning of code and data, and verify that intermediate results are reproducible. This assessment establishes a baseline understanding of where RpD may exist, enabling targeted interventions in the next steps.

***Step 2. Map RpD Causes in Your Software Using Probabilistic Diagrams:*** Leverage probabilistic cause-effect diagrams to visualise and analyse the factors contributing to RpD. Organised into seven categories, these diagrams make patterns easier to recognise and highlight high-risk areas. For each factor observed in your project, record its occurrence to guide prioritisation and planning of interventions.

***Step 3. Prioritise and Plan Reproducibility Interventions:*** Focus on the most impactful factors identified in Step 2. High-priority areas typically include human and organisational practices, documentation gaps, or data quality issues. Document these risks in a reproducibility action plan to ensure systematic mitigation.

***Step 4. Implement Best Practices to Mitigate RpD:.*** Apply targeted strategies according to each issue category. Strategies may involve improving documentation, enhancing testing, standardising workflows, or adopting broader reproducibility practices. Systematic implementation reduces RpD, strengthens reproducibility, and aligns project outputs with scientific standards.

***Step 5. Monitor and Iterate for Continuous Reproducibility:*** Continuously track changes, update documentation, and revise cause-effect diagrams as new risks or patterns emerge. Conduct regular reviews to evaluate intervention effectiveness and adjust strategies as needed. Document any unresolved RpD items along with recommended prevention approaches so that future team members can address them efficiently.

By following these steps, the RpD Management Framework provides a structured, actionable approach to managing reproducibility risks, making scientific software more reliable, maintainable, and easier to reproduce across disciplines.

## 3 Conclusion and Future Work

The Reproducibility Debt Management Framework (RpD-MF) provides a practical, evidence-based approach to addressing reproducibility challenges in scientific software. By combining taxonomies of issues, probabilistic cause-effect diagrams, and actionable mitigation strategies, RpD-MF enables researchers and research software engineers to systematically identify, prioritise, and prevent RpD. The framework supports sustainable software practices and offers guidance applicable across disciplines. Beyond individual projects, RpD-MF provides a foundation for institutional policies, training programs, and collaborative practices that improve reproducibility at scale.

As future work, we propose the development of RpD-Manager, a software tool to support the systematic management of RpD in scientific software projects. Unlike existing tools that often require direct access to researchers' workflows, typically restricted due to proprietary or sensitive research contexts, RpD-Manager would focus on identifying the debt and tracking unresolved debt items, also guiding the application of mitigation strategies without full workflow access. The envisioned functionalities include: dashboards and summaries to visualise RpD risks; structured assessments to identify project-specific risk factors; dynamic tracking of unresolved RpD items with review prompts; guided mitigation through checklists and recommendations; adaptive refinement of strategies based on accumulated data (with potential for LLM-based support); collaborative monitoring through

[1] https://doi.org/10.6084/m9.figshare.30164266.v1
[2] https://doi.org/10.6084/m9.figshare.30185059.v1

shared logs and dashboards; optional integration with version control systems; and analytics to detect emerging risks for proactive management.

## References


[1] 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* (Porto de Galinhas, Brazil) *(FSE 2024)*. Association for Computing Machinery, New York, NY, USA, 462–466. doi:10.1145/3663529.3663778

[2] Zara Hassan, Christoph Treude, Michael Norrish, Graham Williams, and Alex Potanin. 2025. Characterising reproducibility debt in scientific software: A systematic literature review. *Journal of Systems and Software* 222 (2025), 112327. doi:10.1016/j.jss.2024.112327

[3] Zara Hassan, Christoph Treude, Graham Williams, Michael Norrish, and Alex Potanin. 2025. Reproducibility Debt in Scientific Software. In *Companion Proceedings of the 2025 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity* (Singapore, Singapore) *(SPLASH Companion '25)*. Association for Computing Machinery, New York, NY, USA, 50–51. doi:10.1145/3758316.3765482

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

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

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