Empirical Software Engineering (2027) 32:54
https://doi.org/10.1007/s10664-026-10973-6

Reproducibility debt: insights from practitioners

A mixed-methods study of causes, effects, and mitigation strategies

Zara Hassan[1] · **Christoph Treude**[2] · **Michael Norrish**[1] · **Graham Williams**[1] · **Alex Potanin**[1]

Received: 10 December 2025 / Accepted: 15 September 2026
© The Author(s) 2026

Abstract

Context Reproducibility Debt (RpD) refers to the accumulative issues and challenges that hinder the ability to reproduce scientific outcomes, stemming from challenges inherent in scientific software code and data, including development, organisation, dissemination, and documentation. RpD may accumulate when researchers and scientific software practitioners engage in sub-optimal activities, often for short-term benefits, which ultimately compromise the reproducibility of research outcomes. Hence, it is essential to understand the factors that could lead scientific software teams to incur RpD and the strategies they adopt to prevent RpD in their projects.

Objective This research aims to understand practitioners' perspectives on RpD, the factors they perceive as contributing to it, their consequences, and the practices they adopt to address it.

Methodology We present a mixed-methods study combining exploratory interviews with scientific software practitioners and a complementary survey conducted using our *Insight-RpD* survey tool.

Results In total, 23 practitioners participated in interviews and 59 completed the survey. We identified 69 practitioner-reported causes and 80 practitioner-reported effects associated with RpD, confirming and extending current knowledge. Of the 38 causes reported in our previous Systematic Literature Review (SLR), 31 were supported by evidence from this study. The empirical data also highlights 38 additional causes and 47 additional effects not captured in prior literature, which are associated with evolving software practices, such as automation, decentralised teams, and the use of generative AI. Additionally, we explore the mitigation strategies employed by practitioners to address and reduce the impact of RpD.

Conclusion By identifying factors contributing to RpD, its associated consequences, and mitigation strategies from practitioner insights, this study provides evidence-based diagrams of practitioner-reported cause-effect relationships and practical approaches that advance the understanding and management of RpD, while informing future research and practice in Research Software Engineering (RSE) and software sustainability.

Keywords Reproducibility debt · Technical debt · Scientific software · Research software · Research software engineering · Mixed-methods

Extended author information available on the last page of the article

Published online: 03 November 2026

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" (Ernst et al. 2021). Developers frequently encounter TD when prioritising quick solutions over sustainable practices, which can ultimately degrade the quality of the software (Rocha et al. 2017; Tom et al. 2013). Multiple TD types have been identified, along with their occurrences and corresponding management strategies (Alves et al. 2016; Rios et al. 2018a). Addressing TD early is critical, as effective management processes can limit its long-term consequences and maintain project integrity (Freire et al. 2020; Fernández-Sánchez et al. 2017; Lenarduzzi et al. 2021). Recently, Avgeriou et al. (2023) examined the current state of TD management and found that many TD types are mentioned in the literature without a clear definition or explanation of their relationship to TD. This lack of clarity contributes to fragmentation in research and hampers scientific progress in TD.

In the realm of scientific software[1] development (often referred to as research software in contemporary Research Software Engineering (RSE) literature[2]), TD poses unique challenges. Scientific software is frequently developed by domain experts who may lack formal training in Software Engineering (SE) (Pinto et al. 2018; Heaton and Carver 2015). Furthermore, the pressure to produce publishable research results often leads to a focus on rapid results rather than comprehensive documentation or testing (Johanson and Hasselbring 2018; Kanewala and Bieman 2014). Consequently, TD is not exclusive to traditional and commercial software but also manifests itself in scientific software, where its adverse effects can become even more concerning. This added complication is caused by the purpose of scientific software, that is, to produce research results in a variety of disciplines, which form the basis for future scientific research (Vidoni 2021; Hannay et al. 2009).

The widespread use of scientific software across various computational science disciplines has amplified the significance of open science practices, "a movement to make all research artefacts available to the public, thus ensuring the transparency and reproducibility of scientific processes" (Daniel Méndez Fernández et al. 2019). In response, several organisations are actively promoting open science and reproducibility by offering tools and frameworks to share scientific software and associated datasets. Examples include the Research Software Alliance (Resa n.d.), which has chapters in regions including Australia and New Zealand; the Australian Research Data Commons (ARDC n.d.), which provides open access to categorised datasets; the Australian Research Council (ARC n.d.), which mandates open science commitments in its discovery grants; and the National Science Foundation (NSF n.d.), which advocates for publishing source code and data to support reproducibility. The growing emphasis on these initiatives has also contributed to the emergence of RSE, as a recognised field concerned with the development, maintenance, sustainability, and reproducibility of software used in research.

Despite these efforts, reproducibility in scientific software and research results remains a persistent challenge (Hassan et al. 2024). Sculley et al. (2015) and Abubakar et al. (2020) first used the term RpD in the context of Machine Learning (ML) systems, highlighting

[1] "Software developed to achieve specific scientific objectives, support scientific workflows, or enable research publications" (Kanewala and Bieman 2014).

[2] https://zenodo.org/records/5504016

challenges arising from randomised algorithms and non-determinism in parallel learning. Geiger et al. (2018) extended the concept to data-intensive research, emphasising that RpD can accumulate when teams neglect best practices, such as adhering to established workflows and proper documentation.

RpD is a type of TD that affects the reproducibility of computational research. It refers to the "accumulative issues and challenges that hinder the ability to reproduce scientific outcomes, stemming from challenges inherent in scientific software code and data, including development, organisation, dissemination, and documentation. RpD may accumulate when researchers and scientific software developers engage in sub-optimal activities, often for short-term benefits, which ultimately compromise the reproducibility of research outcomes" (Hassan et al. 2025a). Our Systematic Literature Review (SLR) identified several issues leading to RpD, which we named 'RpD items'. These items are related to one or more tangible artefacts of software-supported research, including code, data, and documentation (Hassan et al. 2024). Although our SLR provides a comprehensive overview of RpD, many of the studies identified did not consider the nuanced experiences and perspectives of practitioners involved in scientific software development. Hence, it is imperative to understand directly from these practitioners how they perceive the concept of RpD and the factors contributing to its accumulation.

Prior work (de Toledo et al. 2021; Rios et al. 2018b, 2020b) demonstrates that considering the viewpoints and experiences of software development teams establishes a good understanding of TD issues and their management practices. As RpD is a newly recognised type of TD, there is limited research on the factors that lead scientific software development teams to incur RpD. This gap presents an opportunity for further inquiry to enhance our understanding of RpD and inform approaches for its mitigation. In this regard, this paper presents the results of a mixed-methods study that combines interviews followed by a survey to analyse technical issues, developers' challenges, and other sub-optimal activities (collectively referred to as contributing factors) adopted by practitioners involved in scientific software development that contribute to RpD and inform strategies for its mitigation. This research is designed with the following main objectives in mind:

1. Understand how practitioners perceive the concept of RpD;
2. Identify factors contributing to RpD and their associated impacts;
3. Examine strategies adopted by practitioners to prevent/mitigate RpD.

To achieve these objectives, we employed a mixed-methods approach. First, we conducted semi-structured interviews with 23 practitioners involved in scientific software development to explore their experiences and perceptions of RpD. The qualitative insights from the interviews informed the design of a follow-up survey, which received responses from 59 participants and aimed to validate and extend the interview findings across a broader population. The combination of qualitative and quantitative methods enabled a comprehensive understanding of practitioner perspectives on RpD, its contributing factors, associated consequences, and mitigation strategies.

The rest of this paper is organised as follows: Section 2 provides an overview of the background and related work; Section 3 presents our research questions; Section 4 outlines the methodology adopted for the study; Section 5 presents the consolidated results of both studies. Section 6 discusses the findings, including a cause-effect diagram of RpD, and their

implications. This is followed by a comparison of our results with findings from the SLR. Finally, Section 8 summarises the key insights and outlines directions for future work. All non-sensitive data and analysis code used in this research are openly available in permanent repositories to support reproducibility.[3][4]

2 Background

2.1 Technical Debt

Since Cunningham (1992) first proposed the metaphor of TD numerous studies have identified different types of TD and have devised processes and strategies for their management (Avgeriou et al. 2016; Mcconnell 2008; Li et al. 2014). A substantial body of TD research has focused on developing taxonomies of debt types and approaches for their identification and management. Alves et al. (2016) conducted a systematic mapping study, considering 100 papers published from 2010 to 2014, to propose an initial taxonomy of TD types together with a list of existing strategies for their identification and management. Rios et al. (2018a) then conducted a systematic tertiary study between 2012 and 2018, considering 13 secondary studies. They evolved the taxonomy of TD types and produced a list of indicators of their occurrence, organising a map of activities, strategies and tools to manage TD. Although the results of both studies encompassed several TD types, they worked with primary studies centred on traditional software development and did not consider RpD or scientific software. Likewise, Lacerda et al. (2020) only focused on identifying Code Smells and Code Debt to provide an assessment of which detection and refactoring tools and processes could be applied to counteract those smells.

Beyond debt classification and management, researchers have also examined the human and technical aspects of TD. From a human-centred perspective, Freire et al. (2020) used a validated survey system (InsighTD) to study the relationships between preventive actions and TD types. Although they did assess a large array of debts, they also did not consider RpD. Codabux and Vidoni (2021) explored open peer-reviews written as GitHub issues to identify the TD in R packages submitted to *rOpensci*. They manually analysed 5000 comments from 157 packages and evolved a taxonomy of TD specific to R packages, including the perspective of editors, developers and reviewers; however, RpD was not discussed.

Several other studies identified TD through source code inspection (Brown et al. 2010; Zazworka et al. 2014), examining traditional software written in the Java and C programming languages. Tsoukalas et al. (2021) applied machine learning (ML) to identify and assess TD in Java projects. They used 18 metrics related to each Java class for statistical and ML models to classify them as 'High-TD or not' and proposed a prototype tool for auto-assessment of TD in Java projects. Subsequently, Tsoukalas et al. (2022) introduced a tool named 'TD Classifier' based on earlier work. TD Classifier incorporates the knowledge extracted by accumulating the results of three widely used TD assessment tools Squore (Baldassari 2016), SonarQube (Saarimaki et al. 2019) and CAST (Curtis et al. 2012), and relied on other open-source tools to spot high-TD classes in Java projects from their Git

[3] https://doi.org/10.6084/m9.figshare.30821456

[4] https://github.com/ZaraHassan35/Reproducibility-Debt

repository. While TD Classifier assists the developer in TD management activities, it is limited to the Java programming language and would need to be expanded to support other languages used in the development of scientific software. Tang et al. (2021) formulated a taxonomy of refactoring in ML and deep learning (DL) systems. They introduced 14 ML-specific and 7 TD-specific categories by analysing 26 ML projects. As a result, they suggested best practices and anti-patterns to assist practitioners and developers in evolving long-lasting ML systems and to assist educators in teaching methods for tackling TD in ML and DL systems.

In summary, while TD has been extensively explored within software development, RpD has not yet received any significant attention. The software development community has recognised the importance of addressing TD to ensure the long-term viability and maintainability of software systems. The scientific community is now increasingly grappling with issues related to the reproducibility of scientific software and associated results.

2.2 Reproducibility Debt

The term RpD was formally used by Sculley et al. (2015) and Abubakar et al. (2020) for reproducibility issues in ML systems. They argued that RpD in these systems arises from factors such as randomised algorithms and non-determinism in parallel learning. Later, Geiger et al. (2018) argued that RpD can accumulate in data-intensive research when teams overlook best practices, such as adhering to established workflows and proper documentation, often in order to meet strict research deadlines. Hence, reproducibility was mentioned as a TD in existing literature but without a clear definition and scope. This serves as a baseline for our research on investigating RpD in scientific software.

In our initial exploration of RpD, we conducted a systematic review following rigorous protocols outlined by Kitchenham and Charters (2007). We analysed data from 214 primary studies published until January 2024 to present a comprehensive taxonomy of issues contributing to RpD in scientific software. This effort led to the establishment of the first formal definition of RpD (Hassan et al. 2025a). Our review identified 37 causes related to the emergence of RpD and their 63 corresponding effects. In particular, the findings underscore that issues related to data management and the people involved in the scientific software development process are significant contributors to RpD. RpD poses a challenge across research domains. Information and computing sciences, biomedical and clinical sciences, biological sciences, physical sciences, and earth sciences were the five disciplines that reported the largest number of factors associated with RpD.

Furthermore, we compiled a set of 29 prevention strategies aimed at mitigating the risks associated with RpD. Our research also identified 39 specialised tools and frameworks designed to enhance reproducibility in scientific work. This comprehensive examination enriches the understanding of RpD and offers practical insights for researchers tackling its challenges. However, our findings were solely based on existing literature, indicating a need to explore human perspectives on RpD further. Engaging with practitioners involved in scientific software development was essential to gain a deeper understanding of the unique challenges and developing effective strategies for managing RpD. We report on this engagement in this paper.

3 Research Questions (RQs)

This study investigates how practitioners understand, experience, and manage RpD in scientific software development. We address four RQs, developed based on our primary research objectives and explored through a mixed-methods study involving exploratory interviews and a follow-up *InsightRpD* survey. Below, we present each RQ along with the rationale for its inclusion and the method used to investigate it.

RQ1: How do practitioners understand and perceive reproducibility and RpD in scientific software development?

Our study began with an interview phase aimed at exploring the experiences of practitioners with RpD. However, early findings revealed that a foundational understanding of how practitioners conceptualise reproducibility and RpD was needed before addressing more detailed themes. To establish this baseline, we formulated RQ1 and addressed it through survey data, specifically responses to SQ12–SQ16 of our *InsightRpD* questionnaire, which is presented later in this paper in Table 3. These survey questions assess practitioners' definitions and recognition of reproducibility and RpD in their work. By gathering these insights, we aim to validate and refine the existing frameworks for understanding reproducibility and RpD in scientific software development (Hassan et al. 2024, 2025a).

RQ2: What do practitioners perceive to be the causes and effects of RpD in scientific software development?

This RQ investigates the underlying factors that lead practitioners to encounter or accumulate RpD, as well as its effects. Causes may include technical issues, developer challenges, or sub-optimal development practices (Hassan et al. 2025a). While RpD, like other types of TD, is usually linked to negative impacts, some forms may be acceptable in the short term or taken on purpose for strategic reasons. Still, understanding its effects is important for prioritising mitigation efforts and maintaining the reliability of scientific software (Hassan et al. 2025b).

We employed a mixed-methods approach to answer this question. First, we conducted interviews with practitioners involved in scientific software development to collect qualitative data. The responses were analysed using thematic analysis to uncover common patterns and themes. Based on these findings, we developed follow-up survey questions to validate and quantify the identified themes across a larger sample. The survey responses were analysed using descriptive statistics and thematic coding of open-ended answers. Findings from both sources were then synthesised to provide a comprehensive understanding of perceived causes and effects of RpD.

RQ3: What do practitioners perceive as best practices for the management and mitigation of RpD in scientific software projects?

TD items in any software project can either be prevented or managed (Rios et al. 2018b; Avgeriou et al. 2016). In this context, this question aims to understand which path a development team should follow, i.e., whether it is better to prevent debt or incur it and pay it off later, along with identifying possible strategies for RpD prevention and payment. This RQ was addressed using both interview and survey data. The interviews provided in-depth insights into practitioners' reasoning behind their preferred approaches, while the survey enabled us to validate and quantify these strategies across a broader set of participants. Together, these methods offered a comprehensive view of best practices for managing and mitigating RpD in scientific software projects.

RQ4: Are there any organisational guidelines, methodologies, or tools in practice to mitigate RpD?

While previous research has proposed several guidelines and tools to support reproducibility (Botvinik et al. 2023; Maghami et al. 2023; Brinckman et al. 2019; Goecks et al. 2010; Ziemann et al. 2023; Canon and Younge 2019), it remains unclear how widely these are adopted in practice. Our SLR (Hassan et al. 2025a) identified and organised these existing solutions, but further validation from practitioners was required. To explore this, we added IQ19-IQ22 to our interview questionnaire (See Table 1). Insights from these interviews guided the creation of closed survey questions (SQ24–SQ27 in Table 3) to investigate the adoption and practical use of guidelines, methodologies, and tools across a broader participant population.

4 Methodology

We employed a mixed-methods approach combining interviews and a survey. The interviews provided detailed practitioner insights into RpD, with analysis including identification and counting of recurring themes. The survey complemented this by collecting broader input from a larger group of participants, helping to confirm and extend the interview findings. This combined approach allowed us to gain both depth and breadth in understanding the causes, effects, and mitigation strategies related to RpD. Each method is discussed in detail in the following subsections.

4.1 Interviews with Practitioners

This section details the complete process used for the interview component of our multi-method study

4.1.1 Interview Questions

We designed the interview questions (See Table 1) as a first step of this study, guided by our SLR on RpD (Hassan et al. 2025a) and the *InsighTD* questionnaire (Rios et al. 2018b). *InsighTD* is a set of questions used by several researchers to inquire about TD causes, effects and mitigation strategies (Rios et al. 2020a; de Toledo et al. 2021; Pérez et al. 2021). Our interview questionnaire includes a total of 22 questions. The first ten questions gather demographic information to characterise the participants and the scientific software they develop. The next five focus on RpD, its causes and effects. Questions 16, 17, and 18 explore how RpD is managed in practice, including strategies for prevention. Questions 19, 20, 21 and 22 identify specialised tools, platforms, resources, or organisational guidelines that participants find helpful in preventing RpD.

4.1.2 Context and Population

As the study involved human participants, ethics approval was obtained before the study. The interview data collection period ran from 6 May 2024 to 12 June 2024. Interviews were conducted at a large government research organisation (through convenience sampling),

Table 1 Interview questionnaire

Tell us about:	
Demographics	
IQ1.	Organisation and its divisions
IQ2.	Which type of Scientific Software are you developing in your organisation?
IQ3.	Size (LOC) and Age (Years) software you are working on
IQ4.	Stage of development (initial development, testing, evolution, migration)
IQ5.	Is your software generic or specific to a scientific discipline (for example, earth science, physical science, medical etc.)
IQ6.	Current research field or area of expertise
IQ7.	Years of Experience and highest education received
IQ8.	Size of the team you are working with
IQ9.	Current position in your organisation or role in your team (requirement engineer, analyst, developer, researcher, PM, Team leader)
IQ10.	How do you rate your experience/skill level in this role (at the time)
Causes and effects	
IQ11.	Can you briefly describe what RpD means to you and how often you encounter it?
IQ12.	What was the cause that contributed to RpD?
IQ13.	What additional factors or challenges have you encountered that contribute to RpD?
IQ14.	What are the most significant effects or consequences of the causes you described above?
IQ15.	Any other impacts of RpD felt in the project or other projects you have worked on previously.
Prevention and payment	
IQ16.	In your opinion, can RpD be prevented? If 'Yes' How? If 'No' Why?
IQ17.	In terms of effort, is it better to prevent debt, or incur it and pay it off later?
IQ18.	What practices or solutions have you adopted to prevent the future occurrence of RpD in your project?
IQ19.	Did the organisation define processes OR any precise guidelines to ensure reproducibility of code and results? If so what are they?
IQ20.	Does the organisation adopt any project management methodology (traditional/agile)? If so, which one?
IQ21.	Are you using version control and container technologies? What is their state of adoption in your team? Do these tools ensure reproducibility? How?
IQ22.	Do you use any specialised framework or tool to ensure reproducibility?

having four main research and development divisions and over 600 employees in different roles in several independent or collaborative teams. The organisation focuses on applied research and supports large-scale industrial projects, serving as a technical advisor to both government and industry. Its environment is distinctly non-academic and not affiliated with

any university. For privacy all organisational and participant details are anonymised. In the rest of the document, we will refer to the organisation as CompanyX. The research teams in CompanyX were working on a wide range of scientific software from data acquisition workflows where scientific software is used to acquire the raw data, through data processing software where the scientific software processes the raw data into meaningful insights, onto post-processing software used to visualise and present those findings for the scientists to infer meaningful insights.

We identified five key roles of individuals involved in the development and management of scientific software in CompanyX.

1. **Research Scientist**: A domain expert specialising in their scientific field. They often write short scripts to describe scientific phenomena and interpret results from visualised data. Additionally, they may take on roles as team leaders or project managers on a project-by-project basis.
2. **Data Manager**: Responsible for overseeing the data acquisition process and ensuring the proper storage of data along with metadata. They facilitate the successful handover of raw data to the data analysis team.
3. **Data Analyst**: Handles the analysis of data, extracting meaningful insights and assisting in the interpretation of results to support scientific research.
4. **Research Software Engineer**: A specialised software engineer who works closely with research scientists to develop and maintain research software. They ensure that the software meets requirements and supports scientific objectives, and promotes software quality, sustainability, and reproducibility.
5. **Director**: Oversees the overall coordination of processes and budgets, working in close collaboration with senior management to ensure strategic alignment and effective execution of projects.

To recruit the participants for interviews, we initially contacted the client services team of CompanyX. They directed us to one of their research scientists for assistance. After an initial meeting, the research scientist agreed to participate in an interview and assist in the recruitment of more participants. We refer to this research scientist as **A**. After completing the interview with **A**, they suggested 12 individuals working in different teams and sent them emails inviting them to participate in the study. Eventually, 8 agreed to participate in the study. These 8 participants, on completing the interviews, reported finding the study interesting and recommended other suitable members for interviewing. Our main selection criteria were individuals involved in the development of scientific software at any stage (requirements specification, development, testing, evolution, or migration) and for any research objective (data acquisition, data processing, modelling, or analysis). We eventually conducted 23 interviews with individuals across different teams, working in the five main roles identified above. Most teams were working independently on separate projects, while a few teams collaborated specifically on data. We also had an opportunity to talk to the centralised IT team, who were engaged with members across all teams in CompanyX, and offered a well-rounded perspective.

In developing scientific software, developers aim to achieve specific scientific objectives, such as data management, processing, modelling, and analysis across all scientific disciplines (Kanewala and Bieman 2014). Through our interviews, we found that each par-

ticipant was developing software tailored to their unique scientific goals. Such a variety of scientific software applications within a single organisation provided rich insights into RpD in scientific software projects. The software they worked on varied widely in size. The largest project had 1.2 million lines of code (LOC) whilst smaller Python scripts might consist of just 200 LOC. Some participants did not track LOC. Python was the most common programming language used, though R, MATLAB, and Julia were also represented. The software discussed was somewhat evenly split between generic applications and those specific to particular research fields. Participants were involved in a range of projects, from single large-scale endeavours to multiple smaller endeavours. This diverse sample provided a comprehensive overview of varied scientific software projects within CompanyX. Table 2 presents a characterisation of scientific software on which participants were working.

4.1.3 Data Collection

Before starting the interviews, we shared the Participant Information Sheet (PIS) and consent form with all participants. The PIS described the purpose and objectives of the study but did not provide a formal definition of RpD. This was intentional, as one objective of the interview study was to explore participants' own understanding and experiences of reproducibility-related challenges in scientific software. Each participant gave consent for the interviews to be recorded and transcribed. For participant convenience, we conducted both in-person and online interviews. In total, 18 participants were interviewed in person and 5 participants were interviewed online.

Table 2 A characterisation of the scientific software participants are working on. The columns are the Participant ID, Type of Scientific Software, the Software Age in years, the Software Size in Lines of Code, the Programming Language, and the Development Stage. **DModel&A** is Data Modelling and Analysis; **DP** is Data Processing; **DModel** is Data Modelling; **DMgmt&P** is Data Management and Processing; **DA** is Data Analysis Participant **R** utilised Python, C, FORTRAN, Shell. **T** utilised MATLAB, Python, Julia. **Py&Apps** indicates from small Python scripts to large GUI-based applciations

P.ID	Type	Years	LOC	Language	Stage
A	DModel&A	5	14 000	Julia	Evolution
B	DP	≥ 15	Py&Apps	Python	Evolution
C	DP	20	Py&Apps	Python	Evolution
D	DModel	7	100 000	Python	Evolution
E	DP	6	10 000	Python	Evolution
F	DMgmt&P	10	–	Python	Dev&Test
G	–	–	–	Python	–
H	–	–	–	Python	–
I	DA	10	–	Python	Dev&Evol
J	DMgmt&P	Multiple	Various	Python, R	All stages
K	DModel&A	5	Py&Apps	Python, R	Dev&Evol
L	DA	5	10 000	Python	Evol&Migr
M	DModel	13	–	Python	Repro&Migr
N	DModel&A	5	200–300	Python	All stages
O	DA	2	200–300	Python	Dev&Test
P	DMgmt&P	4	700 000	Python	Dev
Q	DModel	6	Py&Apps	Python	Dev&Test
R	DMgmt &P	8	1 200 000	Python +	All stages
S	DA	0.5	2000	Python	Test
T	DA	Multiple	Py&Apps	Python +	Evolve
U	DP	2	200	Python	Evol
V	DModel	0.5	200–300	Python	Dev&Test
W	DModel&A	0.5	1000	Python	Dev&Test

The interviews were conducted across four non-consecutive weeks within the interview data collection period, with scheduling determined by participant availability and organisational constraints. In week 1, the first author visited CompanyX and collaborated with their assigned representative, who later became Participant A. Only 2 interviews were conducted that day. On the second day, 2 more interviews were conducted. The data from these four participants were then discussed among all authors to ensure consistency in interpretation and alignment on key themes. In the second week, 9 more interviews were conducted (3 per day), and the collected data were reviewed. A steady pace of up to 3 interviews a day was maintained to manage the workload and availability of the participants. Each interview lasted 40-50 minutes. During the interviews, participants were not interrupted and were provided time to reflect before answering questions about RpD causes. While we found that the participants were comfortable answering all questions about RpD and mitigation strategies, a few asked for an explanation of the demographic questions. IQ2, about the type of scientific software they were working with, particularly required explanation. Following a little explanation, they were able to answer the question. When asked about RpD (IQ11) all participants answered that they learned this term from our shared PIS and found the study interesting in the context of reproducibility issues they have encountered. We did not notice any discomfort during the interviews.

4.1.4 Data Analysis

Step 1: The first step in our data analysis involved transcribing audio recordings of the interviews. An automated transcription tool called WhisperAI[5] was used to transcribe the recorded audio. To ensure the accuracy of the transcription, we manually reviewed the transcribed texts.

Step 2: In the second step we applied open coding, a method from grounded theory (Corbin and Strauss 2008), to analyse the transcribed data. Open coding is typically the initial phase of coding in exploratory studies and is aimed at developing a set of concepts that are consistent with the data. For this purpose a qualitative data analysis software called NVivo[6] was used. All transcribed text files were imported into NVivo for analysis. We devised a preliminary coding scheme to address the proposed RQs. Identified factors (technical issues and developer challenges) and mitigation strategies were grouped under the main themes 'RpD Causes' and 'Mitigation Strategies'. In our SLR study, we proposed a taxonomy of issues contributing towards RpD. We used this established structure to further guide the coding process. All identified codes were categorised under seven predefined categories: data-centric, code-centric, documentation-centric, infrastructure and tool-centric, versioning, human-centric, and legal issues.

During the interviews, we asked specific questions about the causes of RpD (IQ12, IQ13) and their corresponding effects (IQ14, IQ15), which made identifying and coding these causes and effects relatively straightforward. The effect of most of the mentioned causes emerged naturally as participants discussed the causes in detail, often stating something like, "As a result of this issue (cause), we faced this problem (effect)". These instances were coded as effects associated with the respective causes. To obtain the codes, from each text

[5] https://openai.com/index/whisper/

[6] https://lumivero.com/products/nvivo/

fragment, we first identified the cause and effects by using the terms used by the participant who mentioned it. Subsequently, to standardise the terminology, we adjusted the terms according to the findings from the SLR, ensuring that the meanings of the labels remained unchanged. Regarding the association between causes and effects, all effects identified in the coding are consequences of a corresponding cause. The mentions of causes and effects were counted on the basis of their frequency of occurrence in all responses.

The first author conducted the qualitative coding, guided by the predefined coding structure (Hassan et al. 2025a). Emerging codes were discussed and reviewed with a second author. Following an initial review of a small subset of the coded data, the analysis proceeded through an iterative, discussion-driven process in which codes, categories, and interpretations were continuously refined through researcher discussion and consensus. As the analysis progressed, all authors collaboratively reviewed the evolving codebook, coded excerpts, and themes to ensure interpretive depth, consistency, and to minimise individual researcher bias. To further enhance the trustworthiness of the findings, we conducted member checking by inviting participants to review and validate the results. A summary of the findings was sent to eight interviewees from different teams. Seven participants fully agreed with the findings and considered them practically appropriate. One senior developer suggested a minor refinement to the mitigation strategies, which we incorporated into the final descriptions.

4.2 InsightRpD-A Global Survey

To develop the *InsightRpD* Survey Tool, we followed a design process used by Rios et al. (2020b) for the *InsighTD* questionnaire. This method is effective because *InsighTD* is an established tool that many researchers rely on for investigating TD (Pérez et al. 2021; de Toledo et al. 2021; Rios et al. 2020a), making it a valuable reference for our work. Our survey questionnaire incorporates knowledge gained from our SLR and interviews, allowing us to create relevant and targeted questions that consider the roles and software development knowledge of the participants involved. The survey was made accessible from 12 December 2024 to 03 March 2025. We conducted the survey by soliciting participation through social media platforms, specifically *LinkedIn* and *X*. The survey link was shared through the authors' professional networks and public posts on these platforms.

In the following, we discuss the methodology adopted for survey design, the survey questionnaire, participant identification and recruitment, and the data analysis process.

4.2.1 Survey Design Methodology

The survey design followed an iterative development and validation process. The questionnaire was initially drafted by the first author based on findings from the SLR, interview study, and the structure of the *InsighTD* survey. It then underwent three rounds of internal review involving all authors, whose expertise spans scientific software, machine learning, programming languages, technical debt, and software engineering. Additional feedback from two independent reviewers was incorporated to improve clarity, relevance, and technical accuracy. Following these revisions, the final questionnaire was prepared for pilot testing.

Pilot Survey A pilot survey involving five participants was conducted prior to the main survey. Our primary objective was to assess participant responses to each question, evaluate the effectiveness of the survey questions in gathering high-quality data, and determine the time required for participants to complete the survey. Additionally, we offered a brief feedback form at the end of the survey to gather suggestions and identify areas for potential improvements. Based on pilot participant feedback, SQ1-SQ11, which were previously open-ended, were changed to closed-ended questions to reduce completion time and help participants focus on the main questions related to RpD. Additionally, SQ24-SQ27 were revised to closed-ended questions to account for participants' limited familiarity with certain software engineering practices. Some minor adjustments were also made to the sequencing of questions.

4.2.2 *InsightRpD* Questionnaire

The survey consists of 28 questions organised into five main blocks, as shown in Table 3. The first block (SQ1–SQ11) gathers demographic and contextual information. The term reproducibility is often used interchangeably with related concepts such as replicability and repeatability (Hassan et al. 2024; Barba 2018), and multiple definitions have been proposed across disciplines (Maghami et al. 2023; Peng 2011; Drummond 2009; Bakinam et al. 2020). Therefore, before inquiring about RpD, SQ12–SQ14 explore participants' understanding of scientific software reproducibility. Participants were first asked about their familiarity with reproducibility (SQ12), then invited to define scientific software reproducibility in their own words (SQ13). Subsequently, we presented our integrated definition of scientific software reproducibility (Hassan et al. 2024) and asked participants to assess its clarity (SQ14).

As RpD is a newly recognised type of TD, SQ15–SQ16 investigate participants' understanding of RpD. Participants were first asked about their perception of RpD (SQ15), after which we presented our definition of RpD (Hassan et al. 2025a) and asked how frequently they had encountered RpD in their scientific software projects (SQ16). Responses to SQ12–SQ16 were used to address RQ1 and inform subsequent survey questions. Questions SQ17–SQ21 explore factors contributing to RpD and their associated impacts. Participants were asked to describe factors they believed contributed to RpD, provide examples from past projects, and discuss the consequences they had observed. These questions primarily support RQ2. Questions SQ22–SQ23 focus on how RpD is managed in practice, including its prevention and remediation. Finally, SQ24–SQ28 investigate the use of software engineering practices and tools in scientific software development, including development processes, software engineering tools, testing practices, barriers to adoption, and suggestions for improving scientific software reproducibility. These questions primarily support RQ3 and RQ4.

4.2.3 Population Definition and Survey Distribution

This study aims to investigate the state of practice in scientific software reproducibility. The questionnaire was completed by practitioners actively involved in the development and use of scientific software. To capture a comprehensive view, we sought participants with diverse roles within scientific software projects, including: *1. Software Developers* as individuals who design and implement scientific software tools and libraries; *2. Research Scientists* as

Table 3 Survey questions

No	Question	Type
SQ1.	In which country are you currently working?	Closed
SQ2.	What is the size of your research organisation? (Number of employees)	Closed
SQ3.	What is the type of your research organisation?	Closed
SQ4.	Which type of Scientific Software are you developing or using?	Closed
SQ5.	Is your software generic or specific to a scientific discipline (for example, earth science, physical science, medical etc.)?	Closed
SQ6.	What is the size (LOC) of the software you are working on?	Closed
SQ7.	What is the age of the software (Years) you are working on?	Closed
SQ8.	What is the current size of the team working on this software?	Closed
SQ9.	What is your current position or role in your research or development team?	Closed
SQ10.	How do you rate your experience/skill level in this role (at the time)?	Closed
SQ11.	What is the highest level of education you have completed?	Closed
SQ12.	How familiar are you with the concept of scientific software reproducibility?	Closed
SQ13.	In your words, how would you describe 'Scientific Software Reproducibility'?	Open
SQ14.	On a scale from 1 to 5, how much did the provided definition of scientific software reproducibility help clarify the concept for you? (1=Low and 5=High)	Closed
SQ15.	Are you familiar with the concept of Reproducibility Debt? If 'yes' Explain in a few words	Open
SQ16.	How often have you encountered RpD in your Scientific Software project?	Closed
SQ17.	When (if) you have encountered RpD, what was the cause of the RpD?	Open
SQ18.	What other motives or reasons or causes contributed either directly or indirectly to the cause you described above?	Open
SQ19.	Recalling the cases of RpD you have encountered across different scientific software projects you have worked on, and the causes of those cases. Which causes would you say are the most likely to lead to RpD?	Open
SQ20.	What are the most significant effects or consequences of the causes you described above?	Open
SQ21.	Any other impacts of RpD felt in the project or other projects you have worked on.	Open
SQ22.	In your opinion, can RpD be prevented? If 'Yes' How? If 'No' Why?	Open
SQ23.	What practices or solutions have you adopted to prevent the future occurrence of RpD in your project?	Open
SQ24.	Have you ever heard of or adopted any of the software engineering (SE) process models (*traditional, agile, hybrid*)?	Closed
SQ25.	Are you using any SE tools and technologies in the development process of scientific software, for example Version Control, Containerisation, Virtual Machines, and Literate Programming?	Closed
SQ26.	Do you perform comprehensive testing (unit, integration, and regression tests) of developed software?	Closed
SQ27.	Can you list the reasons behind not using above mentioned tools and testing practices?	Closed
SQ28.	What improvements or changes do you believe could be made at the institutional or organisational level to better support reproducibility efforts?	Open

scientists in various disciplines who develop and use software in their research workflows; *3. Data Scientists* as individuals who analyse data and build models; and *4. Academic and Industry Participants* as faculty members, students, and industry employees who contribute to or use scientific software in their work.

4.2.4 Data Analysis

In total, 63 participants initiated the survey, of whom 59 completed the survey and were included in the final analysis, resulting in a completion rate of 94%. Responses were included if participants completed approximately 75% of questions across the survey sections. Four submissions containing only partial demographic information and no substantive responses were excluded. While respondents who did not complete the survey may differ from those who completed it, the small number of exclusions suggests limited risk of attrition bias. For the 59 participants included in the analysis, missing responses to individual questions were excluded only from analyses involving those questions; participants remained included in other analyses for which they provided the required responses. No imputation or weighting procedures were applied. The data was analysed using quantitative and qualitative methods, tailored to the nature of each question. For closed-ended questions with nominal, ordinal, or continuous data, descriptive statistics such as frequencies and percentages were calculated to summarise participants' responses. For open-ended questions, a qualitative analysis was conducted to identify themes and patterns in participants' descriptions of RpD. The qualitative analysis process for addressing each RQ varied slightly, depending on the type and length of the responses, as well as the specific insights we aimed to extract from the data.

The survey questions designed to address RQ1 primarily consisted of closed-ended questions, facilitating straightforward quantitative analysis using descriptive statistics. During the analysis of identifying the RpD causes and effects (RQ2), we found that participants' responses about the causes of RpD were concise and clear. Therefore, instead of directly assigning open codes to the data, we first categorised the responses from SQ17, SQ18, and SQ19 into predefined categories: Data-Centric Causes, Code-Centric Causes, Documentation-Centric Causes, and others, based on insights from the SLR. This process was carried out by the first author, in collaboration with a practitioner from the industry. Together, we reviewed all responses and placed them into the predefined categories. Any disagreements or discrepancies were discussed to ensure consistency and mitigate researcher bias in categorisation. In the second stage, we employed an abductive coding approach to the categorised data (Vila-Henninger et al. 2024; Tavory and Timmermans 2014). Abductive coding is a hybrid qualitative data analysis technique, that combines inductive and deductive approaches. This method allowed us to explore data with an open mind, generating new insights, while simultaneously considering our existing theories and concepts derived from the SLR and interviews to develop more comprehensive explanations. We applied the same abductive coding approach to identify the effects of the causes that emerged in the previous step.

For SQ20 and SQ21, where participants were directly asked about the effects of the causes described in SQ17, SQ18, and SQ19, we did not categorise the effects immediately. Instead, we first cross-referenced participants' IDs to determine which individuals had described each cause. For example, if Cause 1 (C1) was highlighted by Participants P23, P34, and P42, we directly coded the effects they described and categorised them according to the corresponding cause. Furthermore, we observed that while participants clearly identified the causes of RpD, the same level of clarity was not present in their descriptions of the effects. This is where the deductive coding approach proved valuable. By drawing on insights from the SLR and interviews, we were able to identify and code the effects of each cause that contributes to RpD.

We used both quantitative and qualitative analysis to gain insights from the data collected in SQ22, SQ23, and SQ28 to answer RQ3. For SQ22, we applied both qualitative and quantitative approaches to understand participants' viewpoints on RpD prevention. For SQ23 and SQ28, we employed an abductive coding approach (Vila-Henninger et al. 2024). Strategies closely aligned with those previously identified were coded accordingly, while we also remained open to identifying new strategies that had not been anticipated. This combined approach allowed us to capture the deeper insights from participants' responses, providing a comprehensive understanding of RpD prevention.

The analysis protocol was defined at the outset and applied consistently. Initial coding was performed by the first author and reviewed collaboratively with an industry practitioner. Emerging codes were discussed, refined, and agreed upon throughout the coding process. The final list of codes for each research question was subsequently reviewed and discussed among all authors of this paper. This collaborative coding approach helped ensure consistency in the interpretation and categorisation of responses while reducing the influence of individual researcher bias.

5 Results

This section presents the consolidated results from both studies. Although the data analyses for both studies were conducted independently, the findings were later compared and integrated using qualitative mapping. This involves identifying common themes, patterns, and relationships between practitioner-reported causes and effects across both studies. By aligning these elements qualitatively, the results become comparable and more meaningful. Additionally, we report the frequency of occurrence for each identified cause and effect to indicate the extent of supporting evidence.

5.1 Participant Demographics

In total, 23 participants took part in the interviews. The sample included one expert with 40 years of experience, several individuals with 15–20 years of experience in their respective fields, and a number of early-career participants with 1–5 years of experience. Most participants held PhD degrees in different scientific disciplines, while others held Master's or undergraduate qualifications. Only two participants had formal education in Computer Science or Software Engineering, and one participant was entirely self-taught in software development. The complete characterisation of the interview participants is presented in Table 4.

Figure 1 presents the demographics of the 59 survey participants included in the final analysis, including their region, scientific discipline, and type and size of organisation. As shown in *Plot a* of Fig. 1, participants come from four main regions, with the largest proportion from Asia (55%), followed by Australia (24%), Europe (16%), and North America (4%). This regional spread highlights significant representation from the Asia-Pacific and European research communities. Participants represent a variety of organisational sizes *(Plot c)*, with 53% working in large organisations, 26% in medium-sized ones, and 20% in small organisations. In terms of organisation type *(Plot d)*, 39% are affiliated with government entities, 37% with academic institutions, and 24% with the private sector. Furthermore, as

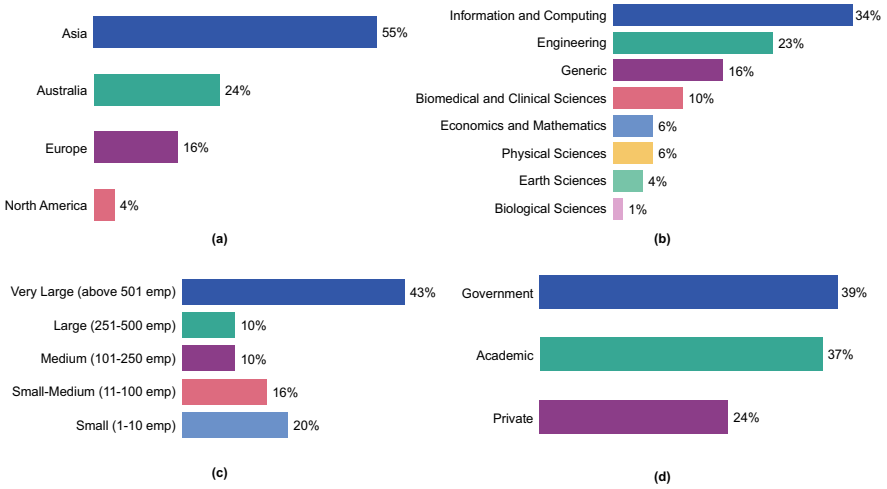
Table 4 Chracterisation of participants for the interview study

Id	Experience (years)	Education	Team size	Role in team
A	17	PhD	8	Senior research scientist
				Project manager
				Developer
B	17	Masters	12–14	Data analyst
C	12	PhD	12–14	Data analyst
				Project manager
D	30	PhD	4	Research scientist
				Project manager
E	8	PhD	5	Research scientist
				Developer
F	40	PhD	10	Research scientist
G	22	PhD	14	Director
H	24	PhD	10	Data analyst
				Project manager
I	12	PhD	30	Data analyst
J	16	Self-taught	4	Senior developer
K	18	PhD	4	Senior research scientist
				Project manager
				Developer
L	5	PhD	5	Research scientist
M	17	PhD	4–5	Research scientist
N	1	Masters'	4–5	Research scientist
O	30	Undergraduate	8	Database manager
P	21	Undergraduate	6	Senior developer
Q	7	Undergraduate	8	Research scientist
R	20	PhD	4–5	Senior developer
S	17	Undergraduate	3	Senior Data analyst
T	23	PhD	8	Data analyst
U	30	Masters'	30	Data manager
V	0.5	Undergraduate	28–30	Data analyst
W	8	Undergraduate	10	Data analyst

illustrated in *Plot b*, participants come from seven distinct scientific disciplines, underscoring the interdisciplinary nature of the study. This diversity ensures that our findings capture a broad range of experiences, challenges, and practices related to RpD in scientific software development and can be used across various fields of research.

5.2 Additional Participant Characteristics

In addition to demographic information, our survey captured further contextual characteristics of the participants and the scientific software they work on. Questions 4–11 in the survey questionnaire provide details about the types of scientific software in use, their technical characteristics, and the roles participants assume within their projects.



.

Fig. 1 Participants demographics: **a**). distribution across geographic regions, **b**). distribution across scientific disciplines, **c**). distribution by organisation size, **d**). distribution by organisation type

5.2.1 Scientific Software Characterisation and Classification

Responses to Questions 4, 6, 7, and 8 describe how participants categorise the scientific software they work with. *Plot b* in Fig. 2 shows that *Data Analysis Software* is the most frequently mentioned category (33 mentions), followed by *Data Processing Software* (25 mentions), *Data Management Software* (22 mentions), and *Data Modelling Software* (21 mentions). These categories align with those identified in the interview study (Table 2).

Participants also report the technical characteristics of their software, including its size, age, and team composition, as shown in *Plots a*, *c*, and *d* of Fig. 2. In terms of codebase size, 42% work on software containing 1,000–10,000 LOC, 31% on 10,000–100,000 LOC, 13% on software under 1,000 LOC, 11% on 100,000–1,000,000 LOC, and 2% on systems exceeding 1,000,000 LOC. Regarding team size, 42% report working in teams of 1–5 members and 36% in teams of 6–10, with only a small proportion working in larger teams. Software age varies: 38% of systems are 1–3 years old, 18% are 4–7 years old, 16% are 8–15 years old, and 16% exceed 15 years.

5.2.2 Practitioners' Roles and Backgrounds

Responses to Questions 9, 10, and 11 provide information about the roles participants assume in scientific software development. As shown in *Plot a* of Fig. 3, the most frequently reported role is project lead (15 participants), followed by research scientists (12), data scientists and analysts (11), scientific software developers (11), and data managers (7). A new role not observed in interviews, *system engineers*, also appears in the survey, representing around 5% of responses. Participants report varied experience levels (*Plot b*), with 31%

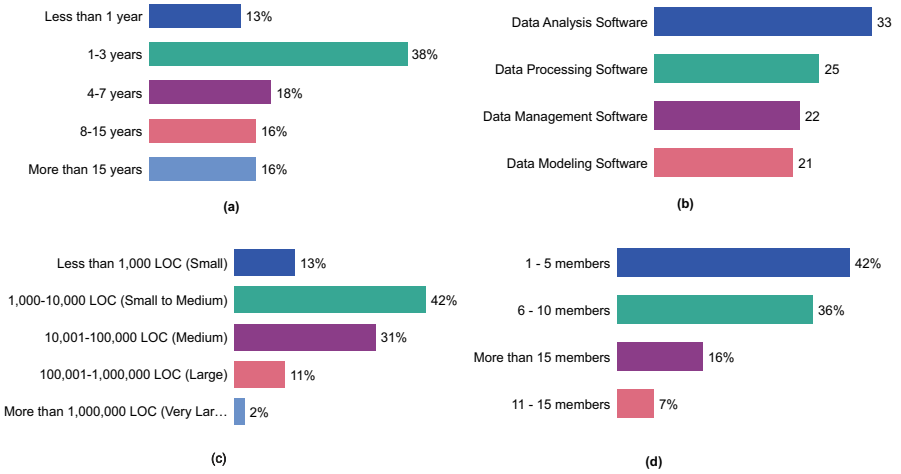


Fig. 2 Scientific software characterisation and classification: (**a**) age of the software participants work on, (**b**) taxonomy of scientific software in practice, (**c**) size of the software participants work on, and (**d**) size of the team developing the software

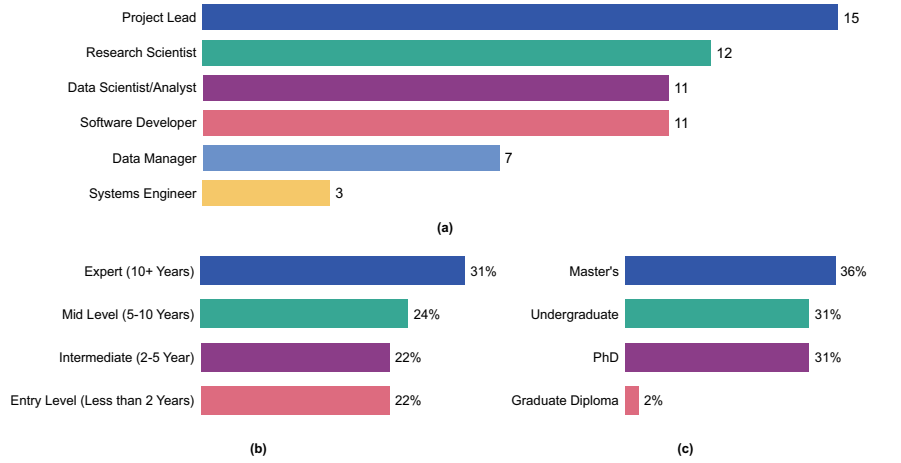


Fig. 3 Practitioners' roles in scientific software development: **a**). Participants' roles (number of mentions), **b**). Distribution of participants by skill level, **c**). Distribution of participants by qualification

identifying as experts with more than 10 years of experience, 24% as mid-level practitioners with 5–10 years, and 44% as entry-level or intermediate practitioners. Educational qualifications similarly vary: 36% hold a Master's degree, 31% a PhD, 31% an undergraduate degree, and 2% a graduate diploma.

5.3 Scientific Software Reproducibility and RpD (RQ1)

5.3.1 Defining Scientific Software Reproducibility

In our previous SLR, we synthesised evidence from 103 primary studies to develop an integrated definition of scientific software reproducibility (Hassan et al. 2024). We define scientific software reproducibility as: *"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."* ur definition is conceptually aligned with ACM's terminology of repeatability, reproducibility, and replicability.[7] In particular, the ability to *re-perform experiments* aligns with ACM's emphasis on repeatability and reproducibility, where computational experiments can be rerun using the same procedures and artifacts to obtain consistent results. Similarly, the ability to obtain results that can be *replicated independently* aligns with ACM's notion of replicability, where independent teams reproduce outcomes using independently developed artifacts. However, our definition extends ACM's framework by explicitly incorporating *computational methods*, *open knowledge*, and the ability to *analyse and interpret* results, reflecting the broader socio-technical challenges of achieving reproducibility in scientific and research software environments.

In this study, we examine how practitioners understand this concept and evaluate the relevance of the proposed definition in practice. To achieve this, SQ12 assessed participants' familiarity with scientific software reproducibility, SQ13 explored their own interpretations of the concept, and SQ14 asked participants to evaluate the usefulness of our proposed definition on a five-point scale. Together, these questions enabled us to assess the alignment between practitioner perspectives and the proposed definition.

The responses to SQ12 indicate varying levels of awareness and organisational support for reproducibility (Fig. 4). While 39% of participants report that reproducibility is well recognised within their organisations and supported by formal guidelines, 30% indicate personal familiarity with the concept despite limited organisational support. In addition, 16% report confusion between reproducibility and replicability, and a further 16% report limited personal understanding despite recognising its organisational importance. These findings highlight considerable variation in how reproducibility is understood and supported across research environments.

Qualitative analysis of SQ13 responses shows strong alignment between practitioners' definitions and the five themes identified in our previous literature review (Hassan et al. 2024): *ability to re-perform experiments, use of computational methods, open knowledge, and independent interpretation, analysis, and replication of results*. Participants frequently referenced these themes, providing further support for the framework developed in our earlier work. Similarly, responses to SQ14 indicate broad agreement with our proposed definition, with 65% of participants expressing strong or good agreement and a further 24% reporting moderate agreement. Only 12% expressed disagreement to varying degrees. Overall, the findings suggest substantial alignment between practitioner perspectives and our proposed framework for scientific software reproducibility (Hassan et al. 2024).

[7] https://www.acm.org/publications/policies/artifact-review-and-badging-current

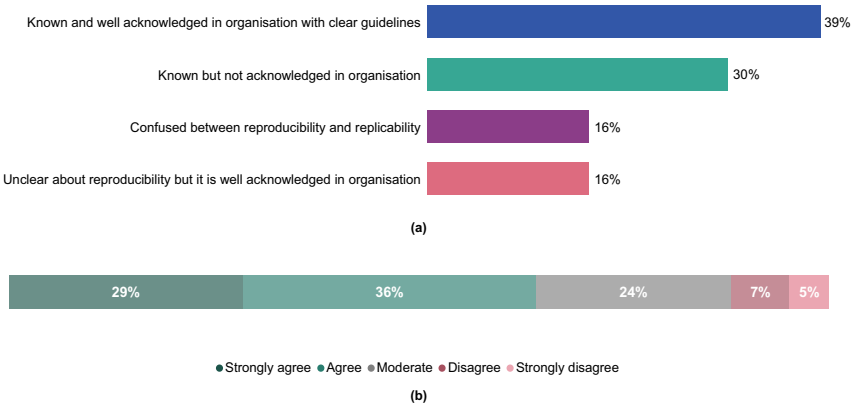


Fig. 4 Participants' perspectives on scientific software reproducibility: **a**). Participants' knowledge on Reproducibility, **b**). Participants' agreement with our proposed definition

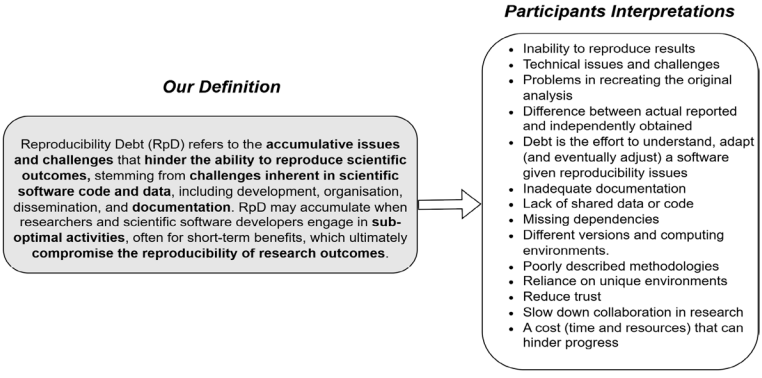


Fig. 5 Participants' interpretations of RpD in scientific software

5.3.2 RpD in Scientific Software

We next examine practitioners' understanding of RpD and how they recognise it in practice using responses to SQ15 and SQ16. Of the 59 survey participants, 19 (32%) report familiarity with the concept and provide their own definitions. Qualitative analysis of these responses (Fig. 5) reveals several recurring themes, including 'inability to reproduce results', 'technical issues and challenges', 'inadequate documentation', 'missing data and code', and 'lack of time and resources'. These themes closely align with the categories underpinning our proposed definition of RpD (Hassan et al. 2025a). While practitioners often describe RpD in terms of difficulties reproducing results, our definition focuses more broadly on the underlying issues that contribute to such difficulties. Despite this difference

in emphasis, the strong overlap between practitioner responses and our framework provides further support for the validity and practical relevance of the proposed definition.

5.3.3 RpD in Scientific Software

We next examine practitioners' understanding of RpD and how they recognise it in practice using responses to SQ15 and SQ16. Of the 59 survey participants, 19 (32%) report familiarity with the concept and provide their own definitions. Qualitative analysis of these responses (Fig. 5) reveals several recurring themes, including 'inability to reproduce results', 'technical issues and challenges', 'inadequate documentation', 'missing data and code', and 'lack of time and resources'. These themes closely align with the categories underpinning our proposed definition of RpD (Hassan et al. 2025a), where scientific outcomes include both computational results and the broader findings derived from them.

Based on this understanding, responses to SQ16 were analysed to assess how frequently practitioners encounter RpD in their scientific software projects. The results indicate that RpD is a common challenge, with 14% of participants reporting that they always encounter it, 26% frequently, and 38% occasionally. Only 22% report rarely experiencing RpD. These findings suggest that RpD is widely encountered across scientific software projects, although its frequency and impact vary between contexts.

Practitioners exhibit varying levels of awareness and institutional support for reproducibility, yet their perspectives largely align with our previously proposed framework. Regarding RpD, it is a nuanced and widely encountered challenge in scientific computing, experienced to different degrees across projects.

5.4 RpD Causes and Effects (RQ2)

Our study identifies a comprehensive set of practitioner-reported causes and effects associated with RpD in scientific software projects, drawing upon data from both interviews and surveys. Across both sources, a total of 69 practitioner-reported causes and 80 practitioner-reported effects of RpD are documented, grouped into seven main categories. Human-centric, documentation-centric, and data-centric factors emerge as the most prominent contributors to RpD, collectively accounting for over 70% of all reported causes and more than 80% of all reported effects. In contrast, code-, tool-, version-, and legal-related factors are reported less frequently but remain important contributors to reproducibility challenges. These findings provide evidence of the systemic and multifaceted nature of RpD, with strong cross-validation between interviews and surveys.

5.4.1 Human-Centric Factors

Human-centric issues are the most frequently mentioned contributors to RpD across both studies. This category includes 17 causes (113 mentions) and 15 effects (66 mentions) as presented in Table 5, accounting for 29% of all (378) cause mentions and 24% of all (276) effect mentions, the highest across all categories. Our interview data reveal 13 causes (81

mentions) and 9 effects (45 mentions), reported by 91% of participants. A vast majority of the interview participants believe that the major causes of RpD in scientific software projects are related to people (such as human, time, and funding) constraints, pressure to publish results, lack of training, inconsistent practices, and a lack of realisation of the importance of reproducible research. For example, one research scientist states:

> *"when we look at our immediate peers and we think yes, he can understand that, but we don't look really broadly what about our peers outside who might be using it and the debt that we are instilling in that software for them to propagate so I would say it is lack of awareness* **[Cause: lack of realisation of reproducible research importance]***.....In my 20 plus years of career have I ever thought about incorporating things about reducing reproducibility debt in the projects that I have done....I can't think of that actually that much because the drive has always been towards producing a scientific result rather than making sure that the tools that we're using are well documented for other people down the lane to be used upon* **[Effect: Insufficient efforts to produce reproducible research]**.

In the survey, we identify 13 causes and 10 effects (32 and 21 mentions, respectively). Of these, 10 causes and 7 effects validate the findings from the interviews and the prior SLR, while 3 causes and 3 effects are newly uncovered. Validated causes include: lack of training in software engineering practices; insufficient incentives for reproducibility-oriented efforts; the absence of formal guidelines; fragmented collaboration between domain experts and developers; and resource and time constraints under publication pressure. Participants also confirm challenges such as: resistance from senior researchers; competing priorities; and a lack of awareness about the value of reproducibility. These findings underscore the central role of motivation, awareness, and organisational culture in shaping reproducibility practices.

5.4.2 Documentation-Centric Factors

Documentation issues are also highly prevalent, with 10 identified causes and 16 effects, accounting for 22% of all cause mentions (85 out of 378 mentions) and 22% of all effect mentions (60 out of 276 mentions), as presented in Table 6. This makes documentation issues the second most significant contributor to RpD. Both the interview and survey studies show strong alignment in this category.

From the interviews, we identify 8 causes (57 mentions) and 14 effects (41 mentions) related to documentation. The most prominent issue reported is non-documented scripts and applications written by scientists. This leads to substantial consequences, including re-work required to re-run scripts, the need to write developer and user documentation, and difficulties understanding how code operates. Another major concern raised by participants is the disconnect between scholarly publications and the underlying code, data, models, and methodologies, which impairs others' ability to reproduce or build upon previous research. Additional key issues include inadequate documentation for open-source projects and undocumented dependencies or environment settings.

Here one senior developer states:

Table 5 RpD causes-effects in human-centric category as appeared in interview and survey

ID	Cause/Effects (#)	I	Sr
C1: Lack of knowledge, training and expertise in data management (2)		✓	
C2: Lack of training or inadequate application of software engineering practices and tools–version control and containerisation technologies (16)		✓	✓
E1	Inability to track changes in code and software configurations over time (2)	✓	–
E2	Rework (3)	✓	–
C3: Lack of recognition, reward and incentives for reproducibility-oriented practices – data, code sharing, exhaustive testing, documentation (6)		✓	✓
E3	Reduced priority of reproducibility practices (1)	–	✓
E4	Prolonged research cycles (2)	–	✓
C4: Lack of formal training and precise reproducibility guidelines at an organisation level (14)		✓	✓
E5	Lack of confidence in developed software for sharing and scrutiny (2)	–	✓
E6	Possible adoption of ad-hoc software development practices (6)	✓	–
E7	Delays in the deadline (4)	–	✓
C5: Fragmented Collaboration in large-sized projects between domain researchers and developers (5)		✓	–
E8	Inconsistent standards and practices (1)	–	✓
C6: Resources (human, time, funding) constraints and publication pressure (31)		✓	✓
E9	Incomplete documentation (10)	✓	–
E10	Insufficient validation (10)	✓	–
E11	Degraded research quality (10)	✓	✓
E12	Reduced collaborations (3)	–	✓
C7: Resistance to adopting reproducibility-oriented tools and practices by experienced researchers (1)		–	✓
C8: Lack of realisation of reproducible research's importance (14)		✓	✓
E13	Insufficient efforts to produce reproducible research (9)	✓	✓
C9: Competing interests/priorities among scientists and developers (3)		✓	✓
E14	Trained developers switch to another team or another job (1)	✓	–
C10: Employee turnover (1)		✓	–
E15	Confusion and inefficiency (1)	✓	–
C11: A large number of stakeholders are involved in the scientific software development process (1)		✓	–

Table 5 (continued)

ID	Cause/Effects (#)	I	Sr
C12: People do not realise the importance of scientific software in the research process (4)		✓	✓
C13: People don't share software to make money from it (10)		✓	✓
E16	Slowed Development Due to Re-inventing Solutions (1)	–	✓
C14: Self-training using generative AI (1)		✓	–
C15: Fear of failure in replication leading to hiding processes- deliberate obfuscation (1)		–	✓
C16: Gap in skills due to Employee turnover (1)		–	✓
C17: Lack of a unified framework to ensure RpD mitigation in research and development (1)		–	✓

> *"So there's been many times where we don't have enough documentation and we have to create that documentation based on what the scientists have written"* **[Cause: Non-documented scripts and application written by scientists]**. They continued: *"So we have to go and, you know, review the application, look at the code, and generally it's usually one big script where we have to modularise it, refactor the code base. We have to write that documentation. We have to write developer docs, user guides, you know, all that sort of stuff, extra work"*. **[Effect: Lot of re-work to re-run the scripts, writing developer docs and user guides]**.

In the survey, 7 causes of RpD (28 mentions) are reported in the documentation category. The survey strongly validates causes identified in the interviews and earlier SLR. These include inadequate or outdated documentation, missing dependency and environment information, and unstructured or outdated comments in large codebases. It also reinforces earlier findings about the disconnect between publications and underlying materials, publication bias, and restrictions in publication formats that limit detailed documentation. Furthermore, manual and inconsistent documentation practices and a lack of incentives to maintain documentation in open-source projects are confirmed as ongoing challenges. In terms of effects, the survey validates 6 effects from prior findings and identifies 3 new effects, with 19 mentions in total. These effects emphasise the compounding burden that poor documentation imposes on reproducibility, including inefficiencies in collaboration, onboarding difficulties, and reduced long-term usability of scientific software.

5.4.3 Data-Centric Factors

Data-related challenges rank third in terms of cause mentions, with 16 causes and 27 effects, contributing 19% of all cause mentions (73 out of 378 mentions). However, they have the greatest impact in terms of effects, accounting for 35% of all effect mentions (98 out of 276 mentions)—the highest among all categories as presented in Table 7.

These issues are identified by 18 interview participants (86%) as significant contributors to RpD. A total of 12 causes (52 mentions) are documented. The most prominent problems include inadequate or outdated data documentation and incomplete or selective reporting of datasets. Inadequate documentation often leads to missing or unclear information about data preprocessing and analytical steps, making it difficult to track changes in datasets over time. Incomplete or selective reporting is associated with missing values and degraded data quality. Additionally, the use of vintage datasets with missing or outdated values is cited as

Table 6 RpD causes-effects in documentation-centric category as appeared in interview and survey

ID	Cause/Effects (#)	I	Sr
C18: Inadequate or outdated data documentation (18)		✓	✓
E17	Missing data preprocessing and analysis details (11)	✓	–
E18	Loss of data provenance (1)	–	✓
E19	Unable to understand the changes in the datasets (4)	✓	–
C19: Inadequately documented dependencies, versions, environment settings, and workflows (13)		✓	✓
E20	Lack of understanding about the used software and versions (1)	–	✓
E21	Unable to recreate the environment (8)	✓	✓
E22	Difficulty and extra work in locating missing dependencies (6)	✓	✓
C20: Missing/outdated/unstructured documentation and code comments for large growing projects (21)		✓	✓
E23	Reduced code understandability (6)	✓	–
E24	Lot of re-work to re-run the scripts, writing developer docs and user guides (6)	✓	✓
E25	Questionable research practices (1)	–	✓
C21: Disconnect between scholarly publications and their underlying data, models, code and methodology used to produce the findings (10)		✓	✓
E26	Questionable research practices (1)	–	✓
E27	Impairs researchers' ability to build upon results (4)	✓	–
C22: Restricted length and format of research articles(1)		–	✓
C23: Scholarly publications are based on narrow datasets with only positive results (2)		✓	–
E28	Increased effort (1)	✓	–
C24: Manual ad-hoc methods of preparing and sharing documentation (4)		–	✓
E29	Affects verify-ability, re-usability and maintainability of software (2)	–	✓
C25: No compensation or reward for open source software — Inadequate documentation for open source projects (7)		✓	–
E30	Black box science (1)	✓	–
C26: Inadequate documentation of scientific process (8)		✓	✓
E31	Unable to keep track of the complete scientific process (5)	✓	–
E32	Inability to Replicate Results (2)	–	✓
C27: Undocumented unit tests (1)		✓	–

a recurring issue, often requiring extra effort or financial resources to recover lost data. Collectively, these data-related shortcomings directly impact the reliability and reproducibility of research findings.

Here, one senior data analyst states:

> *"I think in terms of reproducibility... I've seen it over the years...Someone made a change to the underlying data, and there's nothing that documents why that change*

Table 7 RpD causes-effects in data-centric category as appeared in interview and survey

ID	Cause/Effects (#)	I	Sr
C28: Non-systematic data acquisition, processing and analysis (11)		✓	–
E33	Unorganised data and analysis files (5)	✓	–
E34	Limited traceability of data and metadata (5)	✓	–
E35	Loss of data provenance (9)	✓	–
E36	Lot of money is wasted and extra effort is required (9)	✓	✓
E37	Missing data preprocessing and analysis details (9)	✓	–
C29: Incomplete or selective reporting of datasets (16)		✓	✓
E38	Missing values (8)	✓	✓
E39	Low-quality datasets (7)	✓	–
E40	Frustrating process of replication (1)	–	✓
E41	Partial Reproducibility and biased results (2)	–	✓
C30: Large number of stakeholders involved in data acquisition/processing (2)		✓	✓
E42	Low-quality datasets (7)	✓	–
E43	Incomplete/Inconsistent data documentation (1)	–	✓
C31: Non-standardised data/metadata formats for storage, sharing and reuse (2)		–	✓
E44	Inconsistent results (2)	–	✓
C32: Heterogeneous and complex datasets (2)		✓	–
E45	Challenging data understanding and re-usability (2)	✓	–
C33: Data formats and models evolve over time (5)		✓	–
E46	Limited backward compatibility (2)	✓	–
C34: Data size changes over time (2)		✓	–
E47	Original code break with new data (1)	✓	–
E48	Recurring debt (1)	✓	–
C35: Lack of control on shared data files (4)		✓	–
E49	No data versioning for growing amount of data (3)	✓	–
C36: Lack of data acquisition standards (3)		✓	✓
E50	No accountability on data acquisition and processing workflows (3)	✓	–
C37: Manual recording of metadata–Loss of metadata files (5)		✓	✓
E51	Complexity in code used to handle such missing data and metadata (3)	✓	–
E52	Lot of rework (4)	✓	–
E53	Inconsistent metadata (1)	–	✓
C38: Multiple databases and storage options (3)		✓	–
E54	Duplication of datasets (2)	✓	–
E55	Non-discoverable datasets (2)	✓	–
C39: Multiple versions of slightly different datasets (3)		✓	–

Table 7 (continued)

ID	Cause/Effects (#)	I	Sr
E56	Challenging data understanding and re-usability (2)	✓	–
C40: Vintage datasets with missing information (8)		✓	–
E57	Extra effort and money to recover missing data (5)	✓	–
E58	Loss of data integrity (2)	✓	–
C41: Assuming Determinism in External Data Sources (1)		–	✓
C42: Data obfuscation/anonymisation (4)		–	✓
C43: Inconsistent data points (1)		–	✓
E59	Variable results (1)	–	✓

was made. there's no record of why (reason) those changes were made." **[Cause: Inadequate or outdated data documentation]**. *"So, the biggest, the hardest thing to tackle is the change logs of data and how they've changed".* **[Effect: Unable to understand the changes in datasets]**.

In the survey, 9 data-related causes (21 mentions) are reported, including 6 causes that validate those identified in the interviews and SLR, and 3 newly reported causes. Validated issues reinforce previously established concerns such as inconsistent data collection and analysis methods, non-standardised formats for data and metadata, selective reporting, and insufficient infrastructure for data storage and sharing. Additional challenges include the involvement of multiple stakeholders and increasing dataset complexity. Newly identified causes expand this category further. Participants describe problematic assumptions such as overreliance on external data sources presumed to be stable and trustworthy, anonymisation practices that strip essential information, and inconsistencies within datasets that impair reproducibility. On the effects side, the survey identifies 6 effects, of which 4 confirm prior findings and 2 are newly discovered. These findings affirm the ongoing relevance of known issues while also highlighting new and evolving data practices that further contribute to RpD in modern scientific workflows.

5.4.4 Code-Centric Factors

Code-related factors include 12 causes and 9 effects presented in Table 8, accounting for 10% of all cause mentions (40 out of 378 mentions) and 8% of all effect mentions (22 out of 276 mentions). In the interviews, 7 causes (18 mentions) fall under this category, with lack of unit testing and low-quality scripts arising from non-standardised programming practices being the most prominent. The survey adds 11 causes and 7 effects (22 and 8 mentions), validating 8 causes from the interviews and SLR and introducing 3 new ones, such as insufficient explanation of algorithm logic and obfuscated or unavailable code. These insights emphasise that suboptimal coding practices, especially in non-software-engineering-focused teams, remain a persistent source of RpD.

Table 8 RpD causes-effects in code-centric category as appeared in interview and survey

ID	Cause/Effects (#)	I	Sr
C44: Lack of good and standardised programming practices by developers and scientists (12)		✓	✓
E60	Low-quality code/scripts non-reproducible results (8)	✓	✓
C45: Lack of formal and rigorous code review process before publishing (2)		✓	✓
C46: Complex algorithms and code (1)		–	✓
C47: Lack of comprehensive testing (unit, integration, and regression tests) (3)		–	✓
E61	Low maintainability (1)	–	✓
E62	Wasted time and resources (1)	–	✓
C48: Programming language limitations–software not optimised (2)		✓	✓
E63	Software or algorithm breaks with a large dataset (1)	✓	–
E64	Time consumption on trivial code corrections (1)	–	✓
C49: Limited Resources to write unit tests (3)		✓	–
E65	Limited unit testing (7)	✓	–
C50: Partial sharing of code to retain competitive advantage (2)		✓	✓
C51: Mathematical errors and bugs in program code (7)		✓	✓
E66	Time Delays (1)	✓	–
C52: Non-deterministic scripts (2)		✓	✓
C53: Lack of translating engineering requirements to the code (1)		–	✓
C54: Inadequate documentation/ explanation of the algorithms (3)		–	✓
E67	Threat of misleading future work (1)	–	✓
C55: Code unavailability and obfuscation (2)		–	✓
E68	Recurring debt (1)	–	✓

5.4.5 Tool-Centric Factors

Tool-related issues comprise 8 causes and 6 effects, as presented in Table 9, accounting for 9.52% of all cause mentions (36 out of 378 mentions) and 7.25% of all effect mentions (20 out of 276 mentions). In the interviews, 3 causes (11 mentions) are identified, including the use of outdated tools, non-deterministic behaviour in scripts, and challenges in manag-

Table 9 RpD causes-effects in tool-centric category as appeared in interview and survey

ID	Cause/Effects (#)	I	Sr
C56: Missing or poorly managed dependencies for complex software systems (5)		–	✓
E69	Inability to re-execute the program (3)	–	✓
E70	Inconsistent results (1)	–	✓
C57: Inconsistent software/hardware settings in different computing environments (12)		✓	✓
E71	Unexpected behaviour or even failure to build (7)	✓	✓
C58: Lack of integrated and trusted infrastructure for permanent storage, sharing and execution of scientific workflows (4)		–	✓
E72	Uncertain long-term re-execution of workflows (2)	–	✓
C59: Outdated or non-trivial software tools/infrastructure usage (1)		–	✓
C60: High resource requirement for replication–HPCs or GPUs (2)		–	✓
C61: Non-deterministic order of execution in parallel systems (1)		–	✓
C62: Lack of consensus on a single tool among team members (2)		✓	✓
E73	Inconsistent results (2)	✓	✓
C63: Continuous evolution of technology (9)		✓	✓
E74	Require extra time and effort to learn new technology (5)	✓	✓

ing execution environments. These are associated with 3 effects (6 mentions), highlighting recurring technical barriers to reproducibility. The survey reinforces these findings, reporting 8 causes and 6 effects (25 and 14 mentions, respectively), all of which validate earlier results. Notable issues include inconsistent hardware/software configurations, lack of consensus on toolchains, and infrastructure limitations. Collectively, these findings emphasise the technical complexity involved in maintaining reproducibility in the context of rapidly evolving tool ecosystems.

5.4.6 Version-Centric Factors

Versioning challenges are another source of RpD, adding to the technical burden that undermines reproducibility. Although cited less frequently than other categories, this factor still accounts for 2 causes and 4 effects, as presented in Table 10, comprising 4.76% of all cause mentions (18 out of 378 mentions) and 2.54% of all effect mentions (7 out of 276 men-

Table 10 RpD causes-effects in version-centric category as appeared in interview and survey

ID	Cause/Effects (#)	I	Sr
C64: Version upgrading in programming languages, libraries, operating systems, and other dependencies (17)		✓	✓
E75	Different sets of results and previous inference become invalid (3)	✓	✓
E76	Random number generators behave differently depending on the kernel (1)	✓	–
E77	Limited usability, portability and backwards compatibility (2)	–	✓
C65:Unavailability of previous versions of tools (1)		–	✓
E78	Project Delays (1)	–	✓

Table 11 RpD causes-effects in legal category as appeared in interview and survey

ID	Cause/Effects (#)	I	Sr
C66 Use of proprietary software with proprietary formats and costly licence (8)		✓	✓
E79	Data cannot be transferred from one software to another (1)	✓	–
E80	Unable to access the software (2)	✓	–
C67: Intellectual property and ownership issues in Open Research Software and data (2)		–	✓
C68: Fear of legal consequences over sharing of sensitive data involving human subjects (2)		–	✓
C69: Mixture of open source data with proprietary data, and their mutual non-conformities (1)		–	✓

tions). In the interviews, only one cause is identified (12 mentions), focused on difficulties related to software versioning and upgrades. The survey introduces one additional cause—the unavailability of previous tool versions (6 mentions). These findings highlight how even minor version discrepancies can significantly disrupt reproducibility, particularly when execution environments are not fully preserved or documented.

5.4.7 Legal and Policy Factors

Legal and policy-related issues are the least frequently reported but remain significant in certain contexts, comprising 3% of cause mentions (13 out of 378 mentions) and 1% of effect mentions (3 out of 276 mentions), as presented in Table 11. The interviews identify one cause (5 mentions) related to legal restrictions in data sharing. The survey expands this to four causes (8 mentions), confirming concerns about restrictive software licenses, intellectual property issues, and fear of legal consequences. A new cause, challenges from mixing open-source and proprietary data, is also identified. While less prominent overall, these constraints can completely block reproducibility in specific scenarios.

We identified 69 practitioner-reported causes of RpD in scientific software projects. Of these, 31 causes reported in our prior SLR were confirmed by both empirical studies, while 38 additional causes emerged from practitioner experiences. Human-, documentation-, and data-related factors were the most frequently reported contributors to RpD.

5.5 Mitigation and Prevention of RpD (RQ3)

Across both the interview and survey phases, participants share in-depth perspectives on how RpD can be prevented or mitigated in scientific software projects. From the interview responses to IQ16-IQ18, a total of 28 mitigation strategies are identified. In parallel, the survey responses to SQ22, SQ23, and SQ28 yield 35 strategies, of which 14 overlap with those identified in the interviews, while 21 represent new contributions offering fresh insights. Together, these 49 unique strategies are summarised in Table 12. In addition to outlining specific practices, both interview and survey participants also share their perspectives on the feasibility of RpD prevention and the trade-offs involved in proactively preventing versus paying off RpD later, discussed in detail below.

Perspectives on Prevention: Can RpD Be Avoided Entirely? In both the interviews and the survey, participants are asked whether they believe RpD can be completely prevented. Responses reflect divergent views, with some expressing cautious optimism, others emphasising its inevitability, and a few adopting a case-by-case stance.

In the interviews, out of 23 participants, 14 state that RpD cannot be fully prevented due to the inherent complexities of scientific research and software development. One senior research scientist remarks: *"I think it can be mitigated. I don't think it can ever be 100% prevented because 100% prevention would mean that you have to have everything under your control, including the data, the processing and the outputs. If the data changes, it's not reproducible. If the code changes, it's not reproducible. At an organisational level, if it's a large enough organisation, various things are under various people's control. So I think you can mitigate it to a large extent"*. Another echoes this view: *"Well, maybe not entirely but you can minimise it and limit it as best you can".* Similarly, a senior developer emphasises the limitations of current tools: *"Yeah, we would definitely hope that it could be prevented, but with the current toolchains and toolkits that people use...there's no perfect or comprehensive way to actually avoid it. I mean it will always be there but I guess we can minimise it"*.

However, 7 interview participants express their belief in the possibility of prevention, provided adequate effort, standardisation, and management is in place. One participant notes: *"Yes, it is (prevented), With the passage of time we are trying to make the documentation more and more standardised for each step. So, yes it can be prevented but effort is required and proper management is required at each step. Each data acquisition and processing step that proper quality assurance has to be maintained"*. Another participant states simply: *"Yes, of course, by following meticulously set guidance"*.

Table 12 RpD mitigation strategies as appeared in the interview and survey

ID	Mitigation strategies	I	Sr
	Human/Organisational/Policy		
S1	Provide training and resources on reproducibility skills for developers and reviewers.	–	✓
S2	Create incentives and recognition systems through citations, awards or promotions.	–	✓
S3	Allocate sufficient resources (time, human and funding) to support reproducibility efforts.	✓	✓
S4	Academia should take the responsibility of teaching reproducibility practices (coding, documentation, version control) as mandatory in every scientific discipline.	✓	✓
S5	Development of scientific software by following the software development life cycle model.	✓	–
S6	Close collaboration between research scientists and the development team.	✓	✓
S7	A centralised software development team that works in close collaboration with scientists and develops an organised workflow with the best programming and testing practices and associated documentation.	✓	✓
S8	Trial and error approach to prevent RpD.	–	✓
S9	Negotiate requirements and design implications with stakeholders	✓	–
S10	Leadership should communicate about RpD.	–	✓
S11	Introduce reproducibility experts in organisations.	–	✓
S12	Risk awareness and mitigation plans and reviews in general and in particular with respect to annual code/software/documentation audits.	–	✓
S13	Establish reproducibility guidelines:	✓	✓
S13.1	Established and organised frameworks to ensure developers enforce reproducibility guidelines.		
S13.2	Supervision and monitoring of the complete process.		
S13.3	Long-term policies should be adopted.		
S13.4	Clear expectations and checklist for reproducibility.		
	Documentation-Centric		
S14	Thoroughly document research methods, including data collection, processing and analysis from the start of a project and link them to the fragments of the code implementing the method.	✓	
S15	Document computational environment as README– should indicate the operating system(s), software dependencies(packages, libraries, versions) and hardware used.	–	✓
S16	Use modern programming languages and tools to generate documentation from code comments, Doxygen, Sphinx, and MKDocs.	✓	–
S17	Document complete testing environment and document unit tests.	✓	–
S18	Define system scope, requirements and plan reproducibility.	–	✓
S19	Document and communicate all failures and limitations.	✓	–
S20	Standardised data documentation and quality assurance for each step.	✓	–
	Data-Centric		
S21	Use open trusted repositories for storing, sharing, and publishing data as a package (data, metadata, provenance, documentation of processing-analysis steps and relevant scripts)	–	✓
S22	Select repository that guarantees long-term storage, provides versioning support and assigns DOIs with citation templates	–	✓
S23	Unified storage with standard file names and paths to avoid duplication	✓	–
S24	Data and metadata format standardization and harmonization	–	✓
S25	Integrate metadata with data	✓	–
S26	Provide access to the underlying data along with the paper–No data No publication rule –Provide accurate test data	✓	✓
S27	Use of standardised workflow that can be used by different teams to process different datasets	✓	✓

Table 12 (continued)

ID	Mitigation strategies	I	Sr
S28	Convert proprietary file format to open source formats before publishing of datasets	✓	✓
S29	Learned and experienced people should monitor data acquisition process	✓	–
S30	Prioritise data logging and data documentation as part of the acquisition process	✓	–
S31	Use version control for both data as well as software and associated documentation	✓	✓
	Code-Centric		
S32	Use of Virtual Machine or Container technologies (Docker, Singularity, Sciunit, Kubernetes, Mesos, Oracle VirtualBox, VMware) to encapsulate code and complete computational environment *Combine it with Continuous Integration (CI)	–	✓
S33	Share clean, readable, well-organised, and documented code developed using a version control system –Continuous refactoring, minimise the use of scripts, write code as functions	✓	✓
S34	Minimise external dependencies and make the code base self-sufficient	✓	–
S35	Avoid using large packages that add complexity to the codebase	✓	–
S36	Code documentation or comprehensive code comments	✓	–
S37	Use of code hosting services with integrated version control to share code, associated workflows and link it to publication	–	✓
S38	Comprehensive testing (unit and integration), incorporating rigorous code reviews, automated testing, and continuous integration practices –Test code with real-world data sets	✓	✓
S39	Provide training such as software carpentry to produce better quality code	–	✓
S40	Prefer the use of open-source libraries	✓	✓
S41	Minimise the use of GUIs	✓	–
	Tool/Version and Legal		
S42	Obtain and distribute appropriate open source licences (permissive or copyleft) to allow re-use of software and data.	–	✓
S43	Be conscious of the diverse technology ecosystem and carefully choose toolchains and toolkits that remain stable over time.	✓	✓
S44	Clear framework of RpD, highlighting its impacts.	–	✓
S45	Building ‘playbooks’ of what to do when confronted with the type of situation that invites RpD.	–	✓
S46	Use version control for both data as well as software, and associated documentation.	–	✓
S47	Development of advanced AI-based tools to ensure reproducibility.	–	✓
S48	Provide access to standardised tools to avoid compatibility issues.	–	✓
S49	Recognise software as a first-class research output.	–	✓

In the survey, 38 out of 59 participants respond to SQ22 about RpD prevention. Among them, 17 participants say that RpD cannot be fully prevented. Their reasons include small team sizes, inconsistent standards, environmental variability, and practical enforcement challenges. For example: *“RpD cannot be prevented when a small team is working on modules of datasets or driven software”*. *“It will always be there—competing priorities”*. *“To prevent it absolutely, all team members need to be aware of the issue and educated against it”*. *“Standards need to be enforced consistently among all team members. This is rarely practical”*. Environmental unpredictability is also highlighted: *“Computers or software might work differently in different places, and some problems only show up in certain situations”*. And one participant draws a powerful analogy: *“Just like criminal [activity], it can only be limited or decreased, but cannot be prevented”*.

Twenty-one participants, in contrast, believe RpD can be prevented, offering a variety of practical strategies. One participant suggests that RpD prevention can be achieved through

"the use of standard practices, frameworks, tools, libraries, team coordination, and an emphasis on reproducibility". Two participants emphasise the importance of *"recordkeeping"* and *"proper definition of the engineering problem along with adequate documentation"*. Another pair of participants highlight the need to *"make all necessary data available"* and stress the importance of *"proper documentation, clean coding, frequent code reviews, and documentation of test conditions"*. One participant particularly stresses the importance of *"proper project planning"*, advocating for the need to *"make a plan for reproducibility at the very beginning of the project or software development"*. These insights underline the varied and multifaceted approaches that participants believe could help prevent the occurrence of RpD in scientific software projects.

Prevent or Pay Later: Cost-Benefit Views on RpD Management Question 17 (interviews) specifically focuses on whether preventing RpD early is more effective than dealing with it later. Eleven participants strongly support proactive prevention, noting that although it requires effort upfront, it ultimately saves time and cost:

One research scientist states: *"I would say yes, [less effort is required in proactive prevention] but I don't see any mechanism that really can be enforced"*. Another compares RpD to financial debt: *"It's definitely better to prevent the debt. If you inherit the debt, then it will sit there earning interest until it comes to really hurt."*. A third participant highlights habit formation: *"I think it's a lot harder to pay it off later. I don't think much effort is required in developing habits and systems that reduce a large chunk of reproducibility debt"*.

Three participants suggest that the decision depends on context, including project size, deadline, and likelihood of future change. One senior developer states: *"Things (scope and data) that are unknown and that might change in the future—if you try to over-future-proof something (complex design) and it never happens, then you might incur costs by doing something that might never happen. So it's a fine line and I would say you have to do it case by case"*. Another says: *"If there is a project with only one-month deadline... In that case if a code is not documented in great detail, we say it's okay, we can do this in two months' time or whenever this project is done. We do delay documentation for about one to two months."* Nine interview participants state they are unsure or unable to answer this question definitively.

The collective insights from both the interviews and surveys indicate that participants view RpD as a challenge that can be mitigated, even if complete prevention may not always be achievable. Across both studies, participants identify 49 mitigation strategies, emphasising practices such as documentation, testing, clean code, version control, project planning, and team coordination. These strategies span both individual and organisational efforts and reflect the diverse technical and socio-technical approaches currently employed within research software engineering environments to address reproducibility challenges. Together, these findings provide a broad overview of the practices that practitioners associate with the prevention and management of RpD in scientific software projects.

While RpD may not be entirely avoidable, it can be effectively managed with structured practices. We identified 49 mitigation strategies spanning both individual and organisational efforts, demonstrating increased capability and awareness in managing reproducibility challenges.

5.6 Organisational Guidelines, Methodology, or Tools to Mitigate RpD (RQ4)

To answer RQ4, we explore current practices through interviews (IQ19–IQ22) and a follow-up survey (SQ25–SQ29), both grounded in insights from our SLR (Hassan et al. 2025a).

Organisational Guidelines In response to IQ19, interview participants reveal a widespread absence of formal organisational guidelines for reproducibility. Good practices are often shared informally within teams but are not codified across research units or institutions. This lack of structure was particularly evident in responses from early-career researchers, who described difficulties arising from inconsistent practices and inherited reproducibility challenges. One junior data analyst described their experience: *"I am probably not understanding what I'm actually doing with the data... what I am a little bit anxious about when I'm creating... I always have to check with people who are a bit more experienced than I am, but just because I am inexperienced, that will cause a dataset that I made to be not reproducible"*. Of the 23 interview participants, 22 reported having no official reproducibility guidelines in place, while one participant reported following FAIR principles.

Project Management Methodology IQ20 explored whether teams follow structured project management methodologies. Eight participants reported using Agile practices, and one participant reported using a plan-driven approach for a long-term, complex project. The remaining 14 participants reported not using a formal methodology. Their responses reflected a range of reasons, including uncertainty regarding project requirements and team-specific practices: *'We do not feel the need to use it"*, *'We do not know what we are going to develop and how to develop"*, *'We manage our projects in our own way"*, and *'We are a small team"*.

SE Tools-Version Control and Containers In IQ21, 20 out of 23 interview participants reported using version control systems (e.g., Git), indicating broad adoption among the interviewed practitioners. Container technologies (e.g., Docker and Singularity), however, were reported by only three participants, all of whom belonged to specialised software development teams. These participants described using containers to manage reproducibility issues associated with scientific software developed by collaborators. Most participants reported being unfamiliar with containerisation technologies. This observation is consistent with findings from our SLR (Hassan et al. 2025a), which also reported limited adoption of reproducibility-supporting technologies in practice.

Awareness of Specialised Tools Our SLR identified 39 tools designed to support reproducibility in scientific software (Hassan et al. 2025a). However, responses to IQ22 indicated that none of the 23 interview participants had used or heard of these tools.

Survey Insights-Validation and Broader Patterns Drawing on the interview findings, we developed several closed-ended survey questions (SQ25–SQ29) to assess the broader adoption of software engineering practices and tools. When asked about familiarity with software development process models (Agile, Plan-Driven, Hybrid), most participants reported awareness of these approaches but indicated that they were not routinely applied in practice (Fig. 6). In terms of testing practices, a majority of participants reported using approaches

such as unit testing, integration testing, and code reviews. Participants also reported the use of software engineering tools including version control systems, virtual machines, and containers. Among respondents who did not use such tools, commonly reported barriers included lack of time and resources, insufficient training, and limited incentives for reproducibility-oriented practices (Fig. 7).

While tools and methodologies to mitigate RpD are increasingly recognised, their adoption is limited by organisational and human factors. Promoting reproducibility requires cultural change, supported through targeted training, institutional policies, and community-driven initiatives.

6 Discussion

In this section, we discuss the use of cause-effect diagrams to represent the relationships between the identified causes and effects of RpD, using aggregated practitioner-reported data gathered from our interview and survey studies. We analyse the relationships among the most frequently reported factors and highlight their relative prominence across the empirical dataset. We also discuss the relationship between RpD and traditional forms of TD, highlighting the reproducibility-specific characteristics, stakeholder perspectives, and research software engineering context that distinguish RpD from existing TD concepts. Finally, we reflect on the practical and research implications of our findings and compare our results with related work on RpD in scientific software. This includes a comparison with our prior SLR, highlighting how practitioner insights confirm, extend, or diverge from existing knowledge.

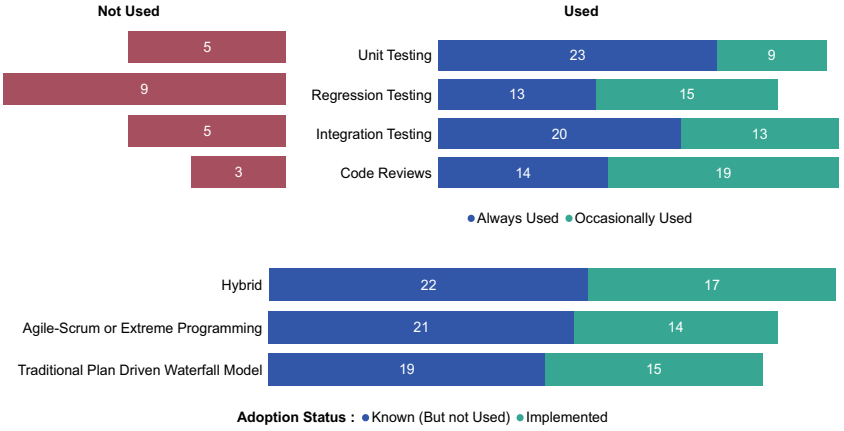


Fig. 6 Knowledge and adoption of SE process models and testing practices among practitioners

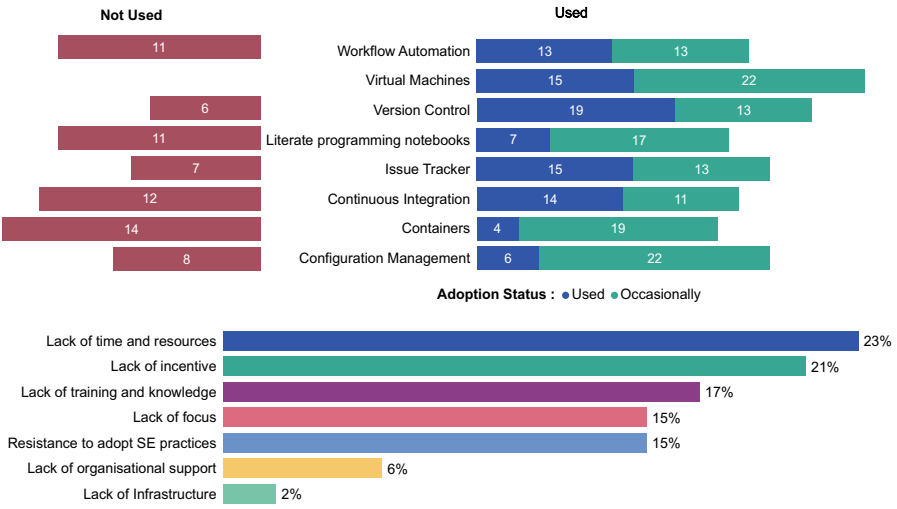


Fig. 7 Adoption status of software engineering tools among practitioners and the primary reasons for non-adoption

6.1 Practitioner-Reported Cause-Effect Diagrams

This study identified a large number of causes and effects of RpD, which can be difficult to interpret without a consolidated view. To summarise these findings, we map them into cause-effect diagrams. Following the approach introduced by Kalinowski et al. (2008), and inspired by Ishikawa's classical cause-effect diagrams (Ishikawa 1986), these visualisations present each cause and effect within its corresponding category and indicate its relative prominence based on practitioner-reported evidence. Darker shading and closer proximity to the central axis reflect more frequently reported items, allowing practitioners to quickly identify the most prominent factors. These diagrams have been shown in prior TD research to support understanding and decision-making (Rios et al. 2019, 2020b), and here they serve as a concise summary of the main patterns observed in our empirical data. Our adaptation focuses specifically on RpD and provides a practical overview of the most prominent causes and commonly experienced effects reported by practitioners across the interview and survey studies.

6.1.1 Building the Diagrams

To calculate the relative prominence values shown in the diagrams (Figs. 8 and 9), we first combined the number of mentions for each cause and effect identified in both the interviews and the *InsightRpD* survey. These counts were then normalised against the total number of identified mentions across causes and effects, respectively. The resulting values represent relative prominence scores within our empirical dataset and are intended to support prioritisation and comparative analysis.

Figure 8 presents a practitioner-reported cause-effect diagram that visualises the causes of RpD in scientific software projects based on our empirical data. Each identified cause is

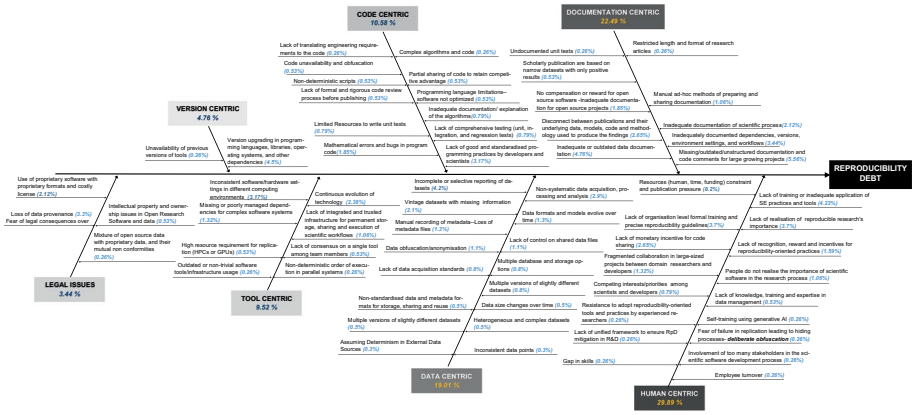


Fig. 8 Practitioner reported cause-effect diagram for the causes attributed towards the emergence of RpD

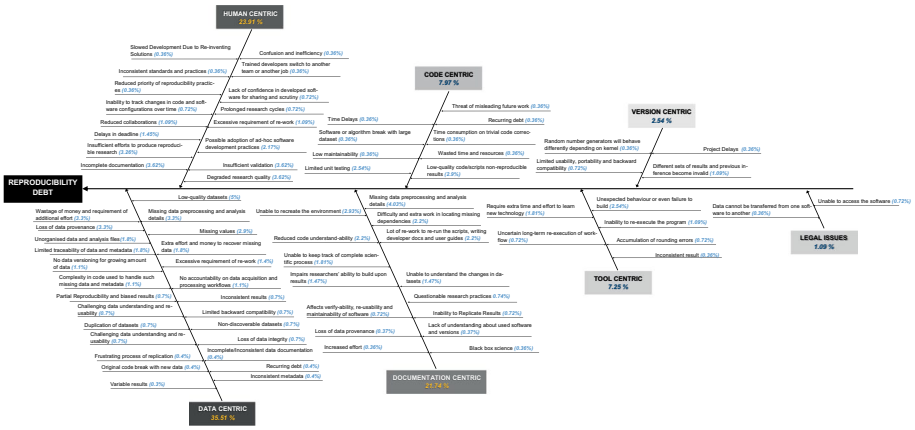


Fig. 9 Practitioner reported cause-effect diagram for the effects of RpD

placed within its relevant category. The proximity of a cause to the central axis indicates how frequently it was reported, i.e., causes closer to the centre were cited more often. For example, the diagram highlights that the Human-Centric category accounts for the largest share of causes (29%, or 113 out of 378 citations). Within this category, the most frequently reported cause is resource constraints (e.g., lack of personnel, time, funding) combined with publication pressure. This is followed by insufficient training or inadequate use of software engineering practices and tools, such as version control systems and containerisation technologies. Figure 9 shows the corresponding diagram for the effects of RpD. As with the causes, each effect is grouped by category and positioned according to how often it was reported by practitioners. Effects placed closer to the main axis represent those most commonly associated with RpD in scientific software projects.

6.1.2 Using the Diagrams

Cause–effect diagrams provide a visual summary of the main causes and effects of RpD across the interviewed and surveyed practitioners. They help identify which causes most commonly contribute to RpD and which effects are most frequently experienced. Teams can use them to answer questions such as: *'Based on practitioner experience or prior studies, which causes are most commonly associated with RpD?"* or *'Which outcomes are most frequently reported when these issues persist?"* While these diagrams are based on aggregated empirical data, research teams can adapt them to reflect their own contexts and update them as new observations emerge.

6.2 Reproducibility Debt vs Technical Debt

Many issues contributing to RpD appear similar to traditional forms of TD, as both arise from deferred actions and decisions that create future costs. However, our findings suggest that RpD represents a distinct form of debt. While TD primarily concerns software quality attributes such as maintainability, reliability, and development efficiency, RpD focuses on the reproducibility of scientific software, computational results, and the scientific outcomes derived from them (Hassan et al. 2025a; Johanson and Hasselbring 2018). Consequently, actions that compromise reproducibility may incur RpD even when they have limited impact on conventional software quality measures.

Our findings further show that RpD extends beyond the software artefacts traditionally considered in TD research. Whereas TD commonly focuses on source code, architecture, testing, requirements, and documentation, RpD encompasses a broader set of reproducibility-critical artefacts, including data, metadata, experimental workflows, computational environments, software dependencies, and research practices. The prominence of data- and documentation-related issues in our findings highlights the importance of these artefacts in research software engineering. A further distinction concerns the stakeholder ecosystem. TD research typically focuses on software practitioners such as developers, architects, and testers. In contrast, scientific software development involves research scientists, data analysts, data managers, and research software engineers, many of whom are domain specialists rather than formally trained software engineers. These differences influence how reproducibility-related issues emerge, are perceived, and are managed.

Finally, RpD is shaped by the research context in which scientific software is developed. Evolving research objectives, changing datasets, experimental workflows, funding constraints, and publication pressures introduce reproducibility risks that are not fully captured by existing TD frameworks. While RpD shares conceptual foundations with TD, its focus on scientific reproducibility, broader set of artefacts, distinct stakeholders, and research-driven context establish RpD as a distinct and complementary debt type within software engineering.

6.3 Implications

The findings of our mixed-methods study reveal the widespread and multifaceted nature of RpD in scientific software development. These implications span technical, organisa-

tional, cultural, and policy dimensions, offering practical and strategic insights to a variety of stakeholders involved in research and software development.

6.3.1 For Practitioners

1. **Evidence-Based Awareness and Prevention:** The identification of 69 causes of RpD presented in Section 5.4 provides practitioners with a concrete, evidence-based foundation to better understand the most pressing reproducibility challenges in scientific software. These causes, ranging from undocumented scientific workflows *(C19)* and unstable environments *(C56, C63, C64)* to insufficient training and informal practices *(C1, C2, C4, C7, C14)*, serve as a guide for teams to reflect on their own processes and proactively implement preventive measures. By referencing these causes, research teams can detect early warning signs of debt accumulation and take informed action before reproducibility problems emerge. In particular, technical challenges alone do not cause RpD; human, documentation, and cultural factors such as poor coding practices *(C44)*, inadequate documentation *(C18, C19, C20, C26)*, and inconsistent data handling *(C28)*, often linked to the lack of training and incentives, also contribute significantly.
2. **Guidance for Remediation through Anticipated Effects:** Understanding 80 identified effects of RpD presented in Tables 5 - 11 including delays in validation, loss of trust, increased rework, and technical fragility, gives practitioners a robust view of downstream consequences. This awareness guides project leaders in justifying resources for remediation, setting realistic timelines, and mitigating organisational risks.
3. **Actionable Prevention Strategies Grounded in Practice:** A comprehensive list of 49 strategies presented in Table 12 has important implications for practitioners to prevent and reduce RpD. These include adopting systematic data management and version control practices, using open and trusted repositories that support versioning and long-term access, and standardising data and metadata formats. Employing container technologies like Docker in combination with continuous integration (CI) pipelines encapsulates computational environments and automates reproducibility checks. Emphasis should also be placed on writing clean, well-documented code, minimising external dependencies, conducting thorough testing and peer code reviews, and clearly documenting research methods and computational environments. Beyond technical measures, fostering a reproducibility-aware culture through training, incentives, resource allocation, and leadership support is essential to improving software quality and sustainability.
4. **Decision Support through Practitioner-Reported Cause-Effect Diagrams:** The cause-effect diagrams (See Figs. 8 and 9) created in this study serve as a practical decision-support tool that makes the often intangible problem of RpD more visible and actionable. These diagrams can be used during project planning, retrospectives, or reproducibility audits to prioritise the most prominent causes and effects requiring attention. By enabling teams to move from intuition-based to evidence-informed decision-making, these diagrams enhance the effectiveness of preventive and corrective measures.

6.3.2 For Researchers

1. **Foundation for RpD as a Distinct Research Construct:** This work positions RpD as a clearly defined and empirically grounded concept within software engineering for science. The structured taxonomy of causes and effects, together with the practitioner-reported cause-effect framework, establishes RpD as an actionable research construct, enabling systematic empirical investigation, theoretical advancement, and practical intervention.
2. **Availability of a Comprehensive Empirical Dataset:** The dataset comprising 69 causes, 80 effects, and several mitigation strategies serves as a valuable empirical resource for secondary analyses, cross-disciplinary comparisons, and reproducibility assessment tools. It provides a solid foundation for reproducibility studies aimed at validation or extension in diverse domains or regions.
3. **Support for Comparative and Longitudinal Studies:** The practitioner-reported cause-effect framework developed in this study enables researchers to conduct comparative analyses of RpD across disciplines, project types, and team structures. By applying this framework, researchers can assess and benchmark reproducibility risk profiles in different scientific settings and observe how RpD patterns evolve over time. This opens the door for longitudinal studies that track the effectiveness of interventions, policies, or tooling improvements in reducing RpD across the software lifecycle.
4. **Pathway for Expanding Global Understanding of RpD:** This work lays groundwork for a globally coordinated effort, similar to the InsighTD initiative for TD, to understand and manage RpD. Encouraging replications, adaptations, and localised studies invites community participation to refine models and represent the diversity of reproducibility challenges worldwide.
5. **Catalyst for Interdisciplinary Collaboration and Policy Development:** Findings highlight the need for collaboration among domain scientists, research software engineers, infrastructure providers, and policymakers. Addressing RpD requires systemic solutions spanning training, tooling, culture, and incentives. The structured insights can inform reproducibility policies at funding agencies, educational institutions, and scientific journals, advancing sustainable, scalable solutions.

6.3.3 For Research Institutions and Project Leaders

Institutions must recognise reproducibility as an organisational responsibility beyond a purely technical issue. Our study highlights the need for structured onboarding and training programs focused on reproducibility practices, the development of organisational guidelines enforcing reproducibility standards, and the allocation of dedicated resources (time, personnel, and funding) for tasks such as testing, documentation, and code cleanup. Strong leadership and clear institutional commitment are critical to fostering environments where reproducibility is a priority (See Table 12; *S1, S2, S3, S4, S10, S11, S12, S13*).

6.3.4 For Tool and Infrastructure Developers

Several RpD causes stem from tooling and infrastructural issues, such as missing dependencies, evolving environments, and insufficient integration of reproducibility practices (see *C56, C57, C58, C64* in Tables 9, 10). Our SLR identified several tools that already support reproducibility through containerisation, CI pipelines, workflow automation, stable versioning, and metadata preservation (Hassan et al. 2025a), but interviews and survey responses indicate that these tools are rarely adopted in practice. This highlights a gap between the availability of reproducibility-supporting infrastructure and its actual use, suggesting that the SE community should focus on understanding and addressing barriers to adoption to ensure these solutions are effectively integrated into scientific software workflows.

6.3.5 For Funders and Policy Makers

Current incentive structures often favour novelty over reproducibility. Time pressures, publication demands, and lack of recognition for reproducible practices contribute to RpD accumulation (See *C3, C6* in Table 5). Funding bodies and publishers should require reproducibility plans in grant proposals and publications, recognise and reward maintenance of software, documentation, and datasets, and promote open science through mandates and infrastructure investments. Adjusting incentives to value reproducibility is essential to addressing RpD effectively (See Table 12; *S2, S3, S44, S49*).

6.3.6 For the Broader Scientific Community

RpD impacts not only original research teams but also the broader community attempting to build upon or replicate prior work. The accumulation of RpD leads to rework, duplicated effort, and erosion of scientific trust. A cultural shift is needed that values transparency, reusability, and long-term stewardship of scientific artefacts. Community-driven initiatives, such as open-source projects, communicating reproducibility challenges, and knowledge-sharing platforms, can help standardise practices and foster collaboration across the scientific ecosystem (See Table 12; *S4, S12, S13, S28, S42, S44*).

6.4 Comparison to SLR

To deepen our understanding of RpD, we compare findings from our interviews and surveys with results from our previously conducted SLR (Hassan et al. 2025a). Across the three sources, we identify 75 causes and 110 effects of RpD. The SLR contributes 37 causes and 63 effects, of which 31 causes and 33 effects are empirically validated by practitioners in our interview-survey study. This overlap highlights strong alignment between challenges reported in the literature and those encountered in practice, lending credibility to the SLR findings and supporting their relevance across contexts

At the same time, our interview-survey study reveals 38 new causes and 47 new effects not present in the literature, underscoring important gaps in academic coverage. Figure 10 provides a visual summary of this overlap and divergence. Notably, the effects of RpD are underreported in existing literature, making it difficult to assess or prioritise RpD in scientific and research software. Our practitioner data helps fill this gap by identifying diverse

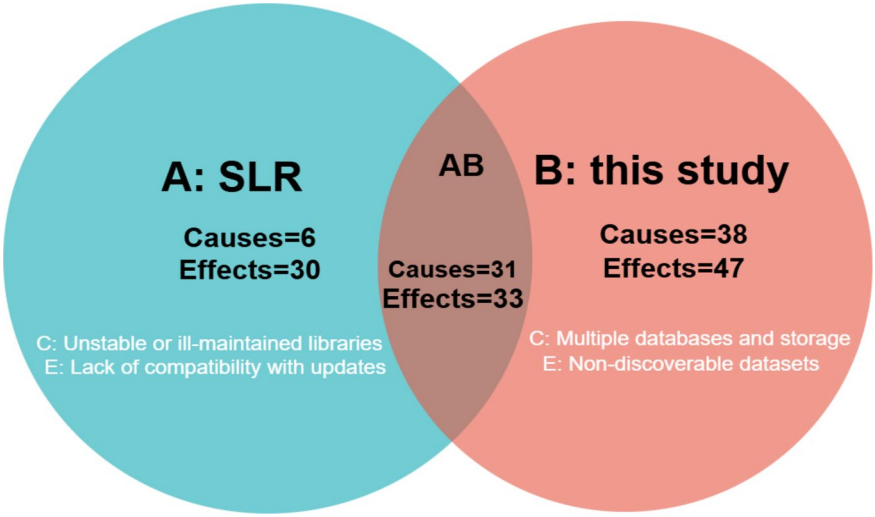


Fig. 10 Venn diagram showing overlap and divergence between SLR **(A)** and interview-survey study **(B)** for causes and effects of RpD

consequences of RpD that are rarely documented in prior work. This variation underscores the importance of capturing practitioner perspectives, as they reveal practical outcomes not visible through literature-based analysis alone. Many of the newly identified causes and effects are associated with evolving software practices, such as automation, decentralised teams, and use of generative AI for self training. Because our SLR (completed in January 2023) includes studies published over a broader timeframe, some more recent challenges may not have been captured. Conversely, our empirical study does not validate 6 causes and 20 effects from the SLR, which may reflect contextual differences between academic and industry settings, or limitations in practitioners' recall. Some issues may be more easily detected in retrospective academic analysis than by practitioners focused on day-to-day development.

In addition to causes and effects, we also examine prevention strategies for mitigating RpD. Our SLR identifies 29 strategies, alongside a set of specialised tools designed to support reproducibility. In practice, however, the number of strategies mentioned by practitioners expands to 49. Of these, 17 strategies align with those from the SLR, 20 are newly identified, and 12 are not supported by our empirical data. Interestingly, no interview participants mention using any specialised tools to ensure reproducibility, and only one survey respondent mentioned Windchill software, a tool not identified in the SLR. This suggests a disconnect between tools proposed in academic literature and those adopted in real-world settings.

Overall, this comparison underscores the value of integrating literature-based and empirical approaches when investigating RpD. The validated findings reinforce established challenges, while the newly uncovered causes, effects, and strategies reveal the evolving nature of reproducibility issues in modern software practices. The disconnect between literature-recommended tools and industry usage further emphasises the need for ongoing engagement with practitioners and continuous updating of the reproducibility knowledge base to ensure proposed solutions remain relevant and actionable.

7 Threats to Validity

As with any empirical study, our research on RpD in scientific and research software is subject to various validity threats. We mitigated these through established strategies, which we detail below.

7.1 Construct Validity

Construct validity concerns the extent to which a study accurately captures the theoretical constructs it aims to investigate (Sjøberg and Bergersen 2023). In this case, RpD causes, effects, and mitigation strategies are the key constructs under investigation. Given the evolving and multifaceted nature of RpD, particular care was taken to ensure that the construct was clearly and consistently defined. We grounded our conceptualisation in prior research, particularly insights derived from our SLR (Hassan et al. 2025a), which informed the design of both the interview protocol and survey instrument. To enhance validity, the survey was pilot-tested to evaluate clarity, relevance, and alignment with study objectives. Feedback from this pilot phase led to minor refinements in question wording and structure. To reduce participant bias, including social desirability effects and hypothesis guessing, participants were informed of the study's general purpose but not specific hypotheses, and anonymity was assured throughout (Wohlin et al. 2012). Member checking was employed not only to validate the accuracy of interview interpretations but also to reduce researcher bias in the coding and thematic analysis process. Additionally, to strengthen the objectivity of the coding framework, we involved a practitioner from industry during the qualitative coding phase. Their independent review and feedback on the initial coding schema helped surface potential blind spots and practical inconsistencies, and their input was incorporated into the final coding structure.

7.2 Internal Validity

Internal validity refers to the extent to which the findings of a study can be attributed to the research procedures rather than uncontrolled factors. In our study, the main threats to internal validity include instrumentation, researcher bias, and maturation.

Instrumentation refers to the influence of data collection instruments on the outcomes of the study (Wohlin et al. 2012). To mitigate this threat, both the interview and survey instruments were designed based on prior literature, including our SLR, and were reviewed by multiple researchers. The survey was subjected to three internal reviews and a pilot study was conducted to assess clarity, relevance, and potential ambiguity. Based on pilot feedback, minor refinements were made to the wording and structure of the questions to reduce the likelihood of misinterpretation.

Researcher bias is another potential threat, especially in the qualitative analysis phase. A potential source of bias arises from the researchers' prior work on RpD, including the development of the RpD concept and its associated framework. This prior knowledge may have influenced the interpretation of qualitative responses or the identification of RpD-related issues during analysis. For the interview study, this threat was mitigated by encouraging

participants to describe their experiences openly rather than directing them toward predetermined causes or effects. The qualitative coding was conducted iteratively, with regular peer debriefing sessions used to refine interpretations and improve consistency. In addition, member checking was conducted by inviting interview participants to review and validate the findings. For the qualitative analysis of the open-ended survey responses, researcher bias was mitigated through a predefined analysis protocol, collaborative review of the initial coding with an industry practitioner, and iterative discussion and refinement of codes. The final coding framework was subsequently reviewed by all authors to improve consistency and minimise individual researcher bias.

Maturation refers to the possibility that participants' responses might change over time or due to fatigue during data collection (Wohlin et al. 2012). We minimised this risk by designing the survey to be concise and easily completed within approximately 25 minutes, as confirmed during the pilot phase. To maintain participant involvement, we included a blend of open-ended and closed-ended questions. This approach not only helped keep participants engaged but also enabled us to draw meaningful inferences from both types of responses. Open-ended questions were answered with thoughtful detail; for example, in response to a question requesting definitions of RpD, the majority of participants provided responses averaging over 20 words. Similarly, when asked to identify key causes and effects of RpD, most respondents listed three or more items, indicating sustained engagement throughout the survey.

7.3 External Validity

External validity relates to the extent to which the results of a study can be generalised to other settings, populations, or contexts (Wohlin et al. 2012). In this study, we took deliberate steps to enhance the generalisability of our findings while also acknowledging their limitations. Our interview participants were drawn from a range of scientific domains, project types, and team structures, providing diverse perspectives on reproducibility challenges in scientific software. However, the interview study was conducted within a single large government research organisation. As organisational context may influence the reproducibility challenges and practices reported by participants, the findings may not fully reflect experiences in all research environments. To broaden the evidence base, we complemented the interview study with the *InsightRpD* survey, which included participants from diverse organisations, scientific disciplines, roles, and geographical regions. While this broader population strengthens the generalisability of the findings, it also introduces a mixed-methods limitation, as the interview and survey participants were drawn from different populations. Consequently, the integrated findings should be interpreted as complementary perspectives on RpD rather than direct replications across identical contexts.

We also note that simulation-intensive software was not explicitly captured or analysed as a separate category in either the interview or survey phases. While such software may be partially represented within categories such as Data Modelling, we cannot determine the extent to which simulation-intensive software is represented in our sample. As reproducibility challenges in simulation-intensive software may differ from those in other scientific software contexts, this should be considered a limitation when interpreting the generalisability of our findings. Additionally, as with most voluntary participation studies, the threat

of self-selection bias remains. It is possible that individuals with a pre-existing interest or concern about reproducibility issues were more likely to engage with the study, potentially skewing the data toward a more aware or motivated subset of the population. While we observed a mix of familiarity levels with reproducibility issues among respondents, we cannot fully rule out this bias.

Another limitation relates to the potential lack of representation from application software contexts or non-English-speaking communities. As the survey was conducted in English and promoted primarily among practitioners working with scientific and research software within academic and research-oriented networks, the generalisability of the results to industry or global software development communities should be interpreted with caution. Despite these limitations, the findings provide valuable insights into reproducibility challenges across a wide range of scientific software projects. To strengthen external validity, future work should involve replication studies in commercial settings and non-English-speaking regions, ideally using stratified sampling to ensure broader representation.

7.4 Qualitative Rigour and Trustworthiness

In addition to traditional validity considerations, we considered qualitative quality criteria informed by Tracy's eight "big-tent" criteria for excellent qualitative research (Tracy 2010). In particular, we focused on rich rigor, credibility, resonance, and meaningful coherence.

Rich rigour was supported through the mixed-methods design, diverse participant sample, and the use of both interview and survey data. Credibility was strengthened through triangulation across data sources, collaborative coding and review among the authors, practitioner involvement in the analysis process, and member checking with interview participants. Resonance is reflected in the practical relevance of the findings for scientific and research software communities, while meaningful coherence is demonstrated through the alignment between the study objectives, research design, analytical approach, and reported findings. We acknowledge that some qualitative quality criteria, such as sincerity and broader resonance across all contexts, were only partially addressed and remain subject to the limitations of the study design and sampling context.

7.5 Reliability and Reproducibility

Reliability, or dependability, refers to the consistency and reproducibility of the study's methods and findings. To ensure this, all data collection instruments and coding protocols were thoroughly documented to support transparency and facilitate replication. The survey instrument was pilot-tested to help ensure consistent interpretation of questions across participants. Collaborative coding sessions and regular cross-checks among researchers enhanced coding consistency and reduced subjectivity. Moreover, we reinforced reliability in our cause-effect analysis by recalculating category-specific relative prominence values for causes and effects, which supported the internal coherence of observed patterns. To further ensure reproducibility, the survey raw data, qualitative code files, interview-survey triangulation files, cause-effect analysis files, and the processed quantitative data have been made available in a permanent Figshare repository.[8] The program code used to process the quantitative data is provided as a fully reproducible workflow pipeline and is publicly

[8] https://doi.org/10.6084/m9.figshare.30821456

available in a dedicated GitHub repository.[9] Interview transcripts and related files cannot be shared due to ethical constraints, as they contain sensitive information about participants and specific project details.

In summary, we employed a range of strategies, including pilot testing, member checking, triangulation, coding transparency, and diverse data sources, to mitigate threats to validity across all dimensions. Nonetheless, we recognise that some limitations remain, including the inherent subjectivity in qualitative interpretation and the demographic scope of our sample. Future work could address these limitations by scaling replication efforts and broadening participation across additional scientific and regional contexts.

8 Conclusion and Future Work

This study provides a comprehensive empirical investigation into Reproducibility Debt (RpD) in scientific software development by synthesising insights from 23 interviews and 59 responses to the *InsightRpD* survey. We begin by exploring how practitioners understand reproducibility and RpD in their day-to-day work. While the term "Reproducibility Debt" is unfamiliar to many, most participants strongly relate to the concept once explained, and 88% of survey respondents agree with our proposed definition. This highlights a shared recognition of the challenges that accumulate when reproducibility is compromised during software development.

Through qualitative and empirical analysis, we identify 69 practitioner-reported causes and 80 practitioner-reported effects of RpD. These findings reveal the socio-technical complexity of reproducibility issues: human and organisational factors emerge as the dominant contributors, followed closely by documentation-related concerns. To structure and interpret the findings, we develop cause-effect diagrams that visualise the relationships between the most frequently reported causes and effects of RpD in scientific software projects. These visual tools help researchers and development teams better understand how reproducibility issues compound and where to focus mitigation efforts. Among individual causes, time and funding constraints stand out, representing 8.2% of all mentions. In response to recurring patterns across the data, we propose 49 practical mitigation strategies, many derived from real-world practitioner experience. These reflect a preference for early and continuous interventions, such as systematic documentation, continuous testing, onboarding protocols, and reproducibility checkpoints. Despite the availability of specialised tools and frameworks in the academic literature, our results suggest they are rarely adopted in practice. Many participants report the absence of organisational policies or dedicated resources for reproducibility, highlighting a gap between research-driven solutions and practical realities. These findings contribute to the growing body of Research Software Engineering (RSE) research by providing empirical evidence on the reproducibility challenges, contributing factors, consequences, and mitigation strategies associated with scientific software development and maintenance.

Looking ahead, future work should focus on replicating the *InsightRpD* survey across broader scientific, geographic, and institutional contexts to strengthen generalisability and identify underrepresented perspectives. Longitudinal studies are needed to examine how RpD evolves overtime and to evaluate the long-term effectiveness of mitigation strategies.

[9] https://github.com/ZaraHassan35/Reproducibility-Debt

There is also strong potential for developing tool-supported solutions to help teams identify, track, and manage RpD in their workflows. These might include lightweight self-assessment checklists, CI/CD integrations, or project dashboards that visualise reproducibility risks. Our practitioner-reported analytical framework could be embedded into such tools to enable evidence-informed decision-making. Additionally, future work should investigate how emerging technologies, particularly Large Language Models (LLMs) and generative AI, may introduce new forms of RpD. These technologies can generate code, documentation, and experimental artifacts that are difficult to verify, reproduce, or trace, potentially amplifying existing challenges in scientific software reproducibility. Understanding the risks, patterns, and mitigation strategies for RpD in LLM-assisted workflows will be critical as their adoption grows in research and development.

Finally, structural and policy-level factors deserve closer attention. Institutional incentives, funding models, and national research policies likely play a critical role in shaping reproducibility practices. Understanding these broader influences will be key to promoting long-term, systemic change. Future efforts should also explore the development of benchmarking frameworks, reproducibility maturity models, and community standards that support self-assessment and promote best practices for sustainable, reproducible scientific software.

Acknowledgements The authors would like to thank all the participants who took part in the study.

Author Contributions Zara Hassan: Conceptualisation, Methodology, Data collection, Investigation, Data analysis, Visualisation, Paper writing – original draft, Paper writing – review and editing. Christoph Treude: Conceptualisation, Methodology, Investigation, Project administration, validation, Writing – review, and Supervision. Michael Norrish: Conceptualisation, Methodology, Validation, Writing – review, and Supervision. Graham Williams: Conceptualisation, Methodology, Validation, Project administration, Writing – review, and Supervision. Alex Potanin: Conceptualisation, Methodology, Validation, Project administration, Writing – review, and Supervision.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions. This work is supported by the Australian National University (ANU) through the ANU PhD scholarship within the ANU Research School of Computing.

Data Availability To ensure reproducibility, the survey raw data, qualitative code files, interview-survey triangulation files and the processed quantitative data have been made available in permanent Figshare repository (https://doi.org/10.6084/m9.figshare.30821456). The program code used to process the quantitative data is provided as a fully reproducible workflow pipeline and is publicly available in a dedicated GitHub repository (https://github.com/ZaraHassan35/Reproducibility-Debt). Interview transcripts and related files cannot be shared due to ethical constraints, as they contain sensitive information about participants and specific project details.

Declarations

Ethics Approval We obtained ethical approval from the Australian National University (ANU) Human Ethics Committee (Protocol: H/2023/1400).

Informed Consent All participants provided informed consent prior to taking part in the study, in accordance with the approved ethics protocol.

Clinical Trial Number Not applicable.

Conflict of Interest The authors have no competing interests to declare that are relevant to the content of this manuscript.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

Abubakar H, Obaidat MS, Gupta A, Bhattacharya P, Tanwar S (2020) Interplay of machine learning and software engineering for quality estimations. In: 2020 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI), pages 1–6, New York, NY, USA. IEEE. ISBN 1728173159

Alves NSR, Mendes TS, de Mendonça MG, Spínola RO, Shull F, Seaman C (2016) Identification and Management of Technical Debt: A Systematic Mapping Study. Inf Softw Technol 70:100–121. ISSN 0950-5849

ARC (n.d.). https://www.arc.gov.au/policies-strategies/policy/arc-open-access-policy

ARDC (n.d.). https://ardc.edu.au/resources/working-with-research-software/

Avgeriou P, Kruchten P, Ozkaya I, Seaman C (2016) Managing technical debt in software engineering (Dagstuhl Seminar 16162). Dagstuhl Reports 6(4):110–138. ISSN 2192-5283. https://doi.org/10.4230/DagRep.6.4.110, https://drops.dagstuhl.de/entities/document/10.4230/DagRep.6.4.110

Avgeriou P, Ozkaya I, Chatzigeorgiou A, Ciolkowski M, Ernst NA, Koontz RJ, Poort E, Shull F (2023) Technical debt management: The road ahead for successful software delivery. In: 2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE), pages 15–30, Los Alamitos, CA, USA. IEEE Computer Society. https://doi.org/10.1109/ICSE-FoSE59343.2023.00007

Baldassari B (2016) SQuORE: a new approach to software project assessment. In: International Conference on Software & Systems Engineering and their Applications, volume 6, New York, NY, USA. ACM

Barba LA (2018) Terminologies for reproducible research. ArXiv:1802.03311, https://api.semanticscholar.org/CorpusID:3639177

Botvinik-Nezer R, Wager TD (2023) Reproducibility in neuroimaging analysis: Challenges and solutions. Biological Psychiatry: Cognitive Neuroscience and Neuroimaging 8(8):780–788. ISSN 2451-9022. https://doi.org/10.1016/j.bpsc.2022.12.006. Reliability of Neurocognitive Measures for Mental Health

Brinckman A, Chard K, Gaffney N, Hategan M, Jones MB, Kowalik K, Kulasekaran S, Ludäscher B, Mecum BD, Nabrzyski J (2019) Computing environments for reproducibility: Capturing the "Whole Tale." Future Generation Computer Systems 94:854–867. ISSN 0167-739X

Brown N, Cai Y, Guo Y, Kazman R, Kim M, Kruchten P, Lim E, MacCormack A, Nord R, Ozkaya I, Sangwan R, Seaman C, Sullivan K, Zazworka N (2010) Managing technical debt in software-reliant systems. In: Workshop on Future of SE Research, FoSER '10, page 47–52, New York, NY, USA. Association for Computing Machinery. ISBN 9781450304276. https://doi.org/10.1145/1882362.1882373

Canon RS, Younge A (2019) A case for portability and reproducibility of HPC containers. In: 2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC), pages 49–54. https://doi.org/10.1109/CANOPIE-HPC49598.2019.00012

Codabux Z, Vidoni M, Fard F (2021) Technical debt in the peer-review documentation of R packages: a rOpenSci Case Study. In: International conference on mining software repositories, pages 1–11, Madrid, Spain. IEEE

Corbin JM, Strauss A (2008) Basics of Qualitative Research (3rd ed.): Techniques and Procedures for Developing Grounded Theory. SAGE Publications. https://api.semanticscholar.org/CorpusID:67424455

Cunningham W (1992) The WyCash portfolio management system. In: Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum), OOPSLA '92, page 29–30, New York, NY, USA. Association for Computing Machinery. ISBN 0897916107. https://doi.org/10.1145/157709.157715

Curtis B, Sappidi J, Szynkarski A (2012) Estimating the principal of an application's technical debt. IEEE Softw 29(6):34–42. https://doi.org/10.1109/MS.2012.156

de Toledo SS, Martini A, Sjøberg DIK (2021) Identifying architectural technical debt, principal, and interest in microservices: A multiple-case study. J Syst Softw 177:110968. ISSN 0164-1212. https://doi.org/10.1016/j.jss.2021.110968

Drummond C (2009) Replicability is not reproducibility: Nor is it good science. In: Proceedings of the Evaluation Methods for Machine Learning Workshop at the 26th ICML

Ernst N, Kazman R, Delange J (2021) Technical Debt in Practice: How to Find It and Fix It. MIT Press

Essawy BT, Goodall JL, Voce D, Morsy MM, Sadler JM, Choi YD, Tarboton DG, Malik T (2020) A taxonomy for reproducible and replicable research in environmental modelling. Environ Model Softw 134:104753. ISSN 1364-8152. https://doi.org/10.1016/j.envsoft.2020.104753

Fernández DM, Monperrus M, Feldt R, Zimmermann T (2019) The open science initiative of the empirical software engineering journal. Empir Softw Eng 24(3):1057–1060. ISSN 1573-7616

Fernández-Sánchez C, Garbajosa J, Yagüe A, Perez J (2017) Identification and analysis of the elements required to manage technical debt by means of a systematic mapping study. J Syst Softw 124:22–38. ISSN 0164-1212. https://doi.org/10.1016/j.jss.2016.10.018

Freire S, Rios N, Mendonça M, Falessi D, Seaman C, Izurieta C, Spínola RO (2020) Actions and impediments for technical debt prevention: Results from a global family of industrial surveys. In: 35th Annual ACM Symposium on Applied Computing, SAC '20, page 1548–1555, USA. ACM. ISBN 9781450368667

Geiger RS, Sholler D, Culich A, Martinez C, Hoces F, de la Guardia F, Lanusse KO, Stuart M, Vareth M, Varoquaux N et al (2018) Challenges of doing data-intensive research in teams, labs, and groups: Report from the BIDS best practices in data science series

Goecks J, Nekrutenko A, Taylor J, The Galaxy Team (2010) Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol 11(8):R86. ISSN 1465-6906. https://doi.org/10.1186/gb-2010-11-8-r86

Hannay JE, Dybå T, Arisholm E, Sjøberg DIK (2009) The effectiveness of pair programming: A meta-analysis. Inf Softw Technol 51(7):1110–1122

Hassan Z, Treude C, Norrish M, Williams G, Potanin A (2024) Reproducibility debt: Challenges and future pathways. In: Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, FSE 2024, page 462–466, New York, NY, USA. Association for Computing Machinery. ISBN 9798400706585. https://doi.org/10.1145/3663529.3663778

Hassan Z, Treude C, Norrish M, Williams G, Potanin A (2025a) Characterising reproducibility debt in scientific software: A systematic literature review. J Syst Softw 222:112327. ISSN 0164-1212. https://doi.org/10.1016/j.jss.2024.112327

Hassan Z, Treude C, Williams G, Norrish M, Potanin A (2025b) Reproducibility debt in scientific software. In: Companion Proceedings of the 2025 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity, SPLASH Companion '25, page 50–51, New York, NY, USA. Association for Computing Machinery. ISBN 9798400721410. https://doi.org/10.1145/3758316.3765482

Heaton D, Carver JC (2015) Claims about the use of software engineering practices in science: A systematic literature review. Inf Softw Technol 67:207–219. ISSN 0950-5849. https://doi.org/10.1016/j.infsof.2015.07.011

Ishikawa K (1986) Guide to Quality Control. Industrial engineering and technology, Asian Productivity Organization. ISBN 9789283310365. https://books.google.com.pk/books?id=POEeAQAAIAAJ

Johanson AN, Hasselbring W (2018) Software engineering for computational science: past, present, future. Comput Sci Eng 20:90–109. https://doi.org/10.1109/MCSE.2018.021651343

Kalinowski M, Travassos G, Card D (2008) Towards a defect prevention based process improvement approach. In: 2008 34th euromicro conference software engineering and advanced applications, pages 199–206. https://doi.org/10.1109/SEAA.2008.47

Kanewala U, Bieman JM (2014) Testing scientific software: A systematic literature review. Inf Softw Technol 56(10):1219–1232. ISSN 0950-5849. https://doi.org/10.1016/j.infsof.2014.05.006

Kitchenham BA, Charters S (2007) Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report

Lacerda G, Petrillo F, Pimenta M, Guéhéneuc YG (2020) Code smells and refactoring: A tertiary systematic review of challenges and observations. J Syst Softw 167:110610. ISSN 0164-1212

Lenarduzzi V, Besker T, Taibi D, Martini A, Fontana FA (2021) A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools. J Syst Softw 171:110827 ISSN 0164-1212. https://doi.org/10.1016/j.jss.2020.110827, https://www.sciencedirect.com/science/article/pii/S016412122030220X

Li Z, Liang P, Avgeriou P(2014) Chapter 9 - architectural debt management in value-oriented architecting. In: Mistrik I, Bahsoon R, Kazman R, Zhang Y (eds), Economics-Driven Software Architecture, pages 183–204. Morgan Kaufmann, Boston. ISBN 978-0-12-410464-8. https://doi.org/10.1016/B978-0-12-410464-8.00009-X

Maghami I, Van Beusekom A, Hay L, Li Z, Bennett A, Choi YD, Nijssen B, Wang S, Tarboton D, Goodall JL (2023) Building cyberinfrastructure for the reuse and reproducibility of complex hydrologic modeling studies. Environ Model Softw 164:105689. ISSN 1364-8152. https://doi.org/10.1016/j.envsoft.2023.105689

McConnell S (2008) Managing technical debt. Construx Software Builders, Inc, pages 1–14

NSF (n.d.). https://www.nsf.gov/pubs/2018/nsf18053/nsf18053.jsp

Peng RD (2011) Reproducible research in computational science. Science 334(6060):1226–1227. ISSN 0036-8075. https://doi.org/10.1126/science.1213847

Pérez B, Castellanos C, Correal D, Rios N, Freire S, Spínola R, Seaman C, Izurieta C (2021) Technical debt payment and prevention through the lenses of software architects. Inf Softw Technol 140:106692. ISSN 0950-5849. https://doi.org/10.1016/j.infsof.2021.106692

Pinto G, Wiese I, Dias LF (2018) How do scientists develop scientific software? an external replication. In: IEEE 25th international conference on software analysis, evolution and reengineering, pages 582–591, Campobasso, Italy. IEEE

ReSA (n.d.) ReSA — researchsoft.org. https://www.researchsoft.org/. Accessed 07 Feb 2024

Rios N, de Mendonça Neto MG, Spínola RO (2018a) A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. Inf Softw Technol 102:117–145. ISSN 0950-5849

Rios N, Spinola RO, de Mendonça MG, Neto CS (2018b) A study of factors that lead development teams to incur technical debt in software projects. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pages 429–436. https://doi.org/10.1109/SEAA.2018.00076

Rios N, Spínola RO, de Mendonça MG, Neto CS (2019) Supporting analysis of technical debt causes and effects with cross-company probabilistic cause-effect diagrams. In: Proceedings of the Second International Conference on Technical Debt, TechDebt '19, page 3–12. IEEE Press. https://doi.org/10.1109/TechDebt.2019.00009

Rios N, Mendes L, Cerdeiral C, Ana PF, Magalhães BP, Correal D, Astudillo H, Seaman C, Izurieta C, Santos G, Spínola RO (2020a) Hearing the voice of software practitioners on causes, effects, and practices to deal with documentation debt. In: Madhavji N, Pasquale L, Ferrari A, Gnesi S (eds), Requirements Engineering: Foundation for Software Quality, pages 55–70, Cham. Springer International Publishing. ISBN 978-3-030-44429-7

Rios N, Spínola RO, Mendonça MG, Seaman CB (2020b) The practitioners' point of view on the concept of technical debt and its causes and consequences: a design for a global family of industrial surveys and its first results from Brazil. Empir Softw Eng 25:3216–3287

Rocha JC, Zapalowski V, Nunes I (2017) Understanding Technical Debt at the Code Level from the Perspective of Software Developers, page 64–73. SBES'17. Association for Computing Machinery, New York, NY, USA. ISBN 9781450353267. https://doi.org/10.1145/3131151.3131164

Saarimaki N, Baldassarre MT, Lenarduzzi V, Romano S (2019) On the accuracy of SonarQube technical debt remediation time. In: 2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pages 317–324, New York, NY, USA. IEEE Xplore. https://doi.org/10.1109/SEAA.2019.00055

Sculley D, Holt G, Golovin D, Davydov E, Phillips T, Ebner D, Chaudhary V, Young M, Crespo J-F, Dennison D (2015) Hidden technical debt in machine learning systems. In: NIPS'15: proceedings of the 28th international conference on neural information processing systems, vol 2, pp 2503–2511, Montreal Canada. ACM

Sjøberg DIK, Bergersen GR (2023) Construct validity in software engineering. IEEE Trans Software Eng 49(3):1374–1396. https://doi.org/10.1109/TSE.2022.3176725

Geiger RS, Sholler D, Culich A, Martinez C, Hoces F, de la Guardia F, Lanusse KO, Stuart M, Vareth M, Varoquaux N et al (2018) Challenges of doing data-intensive research in teams, labs, and groups: Report from the BIDS best practices in data science series

Tavory I, Timmermans S (2014) Abductive analysis: Theorizing qualitative research. University of Chicago Press

Tom E, Aurum A, Vidgen R (2013) An exploration of Technical Debt. J Syst Softw 86(6):1498–1516. ISSN 0164-1212. https://doi.org/10.1016/j.jss.2012.12.052

Tracy S (2010) Qualitative quality: Eight "big-tent" criteria for excellent qualitative research. Qual Inq 16:837–851. https://doi.org/10.1177/1077800410383121

Tsoukalas D, Mittas N, Chatzigeorgiou A, Kehagias DD, Ampatzoglou A, Amanatidis T, Angelis L (2021) Machine learning for technical debt identification. IEEE Trans Softw Eng, pages 1–1. https://doi.org/10.1109/TSE.2021.3129355

Tsoukalas D, Chatzigeorgiou A, Ampatzoglou A, Mittas N, Kehagias D (2022) TD classifier: Automatic identification of Java classes with high technical debt. In: 2022 IEEE/ACM International Conference on Technical Debt (TechDebt), pages 76–80, New York, NY, USA. IEEE

Vidoni M (2021) Self-admitted technical debt in R packages: An exploratory study. In: 2021 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR) (MSR), pages 179–189, Los Alamitos, CA, USA. IEEE Computer Society. https://doi.org/10.1109/MSR52588.2021.00030

Vila-Henninger L, Dupuy C, Van Ingelgom V, Caprioli M, Teuber F, Pennetreau D, Bussi M, Le Gall C (2024) Abductive coding: Theory building and qualitative (re) analysis. Sociol Methods Res 53(2):968–1001

Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A et al (2012) Experimentation in software engineering, vol 236. Springer

Zazworka N, Vetro' A, Izurieta C, Wong S, Cai Y, Seaman C, Shull F (2014) Comparing four approaches for technical debt identification. Softw Quality J 22(3):403–426. ISSN 0963-9314

Ziemann M, Poulain P, Bora A (2023) The five pillars of computational reproducibility: bioinformatics and beyond. Brief Bioinform 24(6). ISSN 1467-5463. https://doi.org/10.1093/bib/bbad375

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Zara Hassan[1] · Christoph Treude[2] · Michael Norrish[1] · Graham Williams[1] · Alex Potanin[1]

✉ Zara Hassan
zara.hassan@anu.edu.au

Christoph Treude
ctreude@smu.edu.sg

Michael Norrish
michael.norrish@anu.edu.au

Graham Williams
graham.williams@anu.edu.au

Alex Potanin
alex.potanin@anu.edu.au

[1] The Australian National University (ANU), ANU School of Computing, 108 North Rd, Acton ACT 2601, Canberra, Australia

[2] Singapore Management University, Singapore, Singapore