Technical Debt in the Peer-Review Documentation of R Packages: a rOpenSci Case Study

Zadia Codabux
University of Saskatchewan
zadiacodabux@ieee.org

Melina Vidoni
RMIT University
melina.vidoni@rmit.edu.au

Fatemeh H. Fard
University of British Columbia
fatemeh.fard@ubc.ca

Abstract*—*Context: **Technical Debt (TD) is a metaphor used to describe code that is "not quite right." Although TD studies have gained momentum, TD has yet to be studied as thoroughly in non-Object-Oriented (OO) or scientific software such as R. R is a multi-paradigm programming language, whose popularity in data science and statistical applications has amplified in recent years. Due to R's inherent ability to expand through user-contributed packages, several community-led organizations were created to organize and peer-review packages in a concerted effort to increase their quality. Nonetheless, it is well-known that most R users do not have a technical programming background, being from multiple disciplines. *Objective:* The goal of this study is to investigate TD in the documentation of the peer-review of R packages led by *rOpenSci*. *Method:* We collected over 5,000 comments from 157 packages that had been reviewed and approved to be published at rOpenSci. We manually analyzed a sample dataset of these comments posted by package authors, editors of *rOpenSci*, and reviewers during the review process to investigate the types of TD present in these reviews. *Results:* The findings of our study include (i) a taxonomy of TD derived from our analysis of the peer-reviews (ii) documentation debt as being the most prevalent type of debt (iii) different user roles are concerned with different types of TD. For instance, reviewers tend to report some types of TD more than other roles, and the types of TD they report are different from those reported by the authors of a package. *Conclusion:* TD analysis in scientific software or peer-review is almost non-existent. Our study is a pioneer but within the context of R packages. However, our findings can serve as a starting point for replication studies, given our public datasets, to perform similar analyses in other scientific software or to investigate the rationale behind our findings.**

***Index Terms*—Technical Debt, R Programming, rOpensci, Technical Debt Taxonomy, Mining Software Repositories**

I. Introduction

Technical Debt (TD) describes the problem of balancing near-term value with long-term quality [1]. The relevance of this metaphor in software development has been widely acknowledged, and researchers have studied this phenomenon from different perspectives [2]. Nonetheless, the artifacts used to study TD have remained somewhat the same [3], with only recent efforts venturing beyond source code [4, 5]. Managing technical debt is crucial as software containing debt are often more prone to defects, changes, vulnerabilities, and other software issues [6, 7, 8, 9, 10].

This work is partially supported by Natural Sciences and Engineering Research Council of Canada RGPIN-2019-05175.

The role of software in data science and statistics has increased considerably over the years [11]. As a result, scientific organizations identified the production and maintenance of scientific software as an area of concern, as it can affect the validity of results [12]. The relevance of this issue has increased recently, as top conferences have started reviewing the reproducibility and transparency of software tools used in academic production[1]. However, the process of reviewing scientific production is demanding, requiring extensive amounts of human effort [13]. Moreover, regardless of its popularity, there is not much research in R from a software engineering perspective, and technical debt has not been explored in this context [14]. Additionally, most R programmers do not have a solid programming background like a Software Engineer may have [15].

One of the most widely used programming languages for scientific software is R. It is a multi-paradigm language that has grown considerably due to its package-based structure and developers' ability to contribute packages publicly available as extensions of the "base" language [16]. It is commonly used in scientific applications, such as statistics and data science [17]. CRAN, the Comprehensive Archive Network, was created for users to suggest improvements and report bugs and, nowadays, it is the official venue to submit user-generated R packages [14]. However, although it has a growing community, most package contributors are not software engineers by training [15], and only a few of them are mindful of the inner concepts of the language [18]. This lack of formal programming training can lead to lower code quality [19].

Several community-led organisations were created to organize and review packages - among them, *rOpenSci* [20] and *BioConductor* [21]. In particular, *rOpenSci* has established a thorough peer-review process for R packages, based on the intersection of academic peer-reviews and software reviews. *rOpenSci* is a well-known organization that conducts a thorough peer-review process, which has become an integral part of the R community [20]. Moreover, they have forged partnerships with academic periodicals to complete their review process[2]. Their peer-review process is also open (i.e. not blinded[3]), and conducted in public

[1]https://bit.ly/2IsRFoW
[2]https://ropensci.org/blog/2017/11/29/review-collaboration-mee/
[3]https://devguide.ropensci.org/

978-1-7281-8710-5/21/$31.00 ©2021 IEEE
DOI 10.1109/MSR52588.2021.00032

GitHub issues[4]. *rOpenSci*'s scope is far more extensive than *BioConductor*'s. The latter is limited to specific bioinformatics packages only and interacts with existing packages and reuses their data structures[5]. Moreover, although CRAN allows for the submission of packages, it does not perform a thorough review of the submissions. Instead, it opts for generalized and partially automated checks.

This study aims to investigate TD in scientific software reviews, specifically in the context of R packages. We focused on the *rOpenSci*'s open peer-review process, which is handled using GitHub issues. We extracted 157 completed and accepted package reviews, consisting of more than 5,000 individual comments. The 157 reviews were broken down into individual comments and classified by the comment's author's role in the peer-review process. A sample of 358 comments, further broken down into 600 phrases, was inspected and manually analyzed by two authors to extract TD instances.

Our findings indicate:

- Several types of TD were found in the peer-review documentation, and the definitions were adjusted to fit the inherent characteristics of the R programming language. Furthermore, the types of debt were grouped according to three perspectives (namely users, developers and CRAN), representing "who" is the most affected by the debt type.
- Regardless of the participants' roles, *documentation* is the most prominent and *versioning* is the least encountered type of debt in the peer-review process.
- The participants (or "user roles" as referred to in this study) of the review process (i.e. *authors*, *editors* of *rOpenSci* and *reviewers* of the submitted packages) appear to focus on different types of debt. The percentage of the phrases with *documentation debt* is the highest among the *reviewers* and *editors*. However, *authors* tend to report more *defect debt*. Moreover, the distribution of debt types found in the comments is significantly different between *reviewers*, and other roles, while the distribution of TD types are not different among the comments written by *authors* or *editors*. Furthermore, the *reviewers* reported most of the TD instances.

Overall, the contributions of this study are:

- This is the first study that investigates TD in the R package peer-reviewing documentation. We use a new source of data to explore TD, compared to most studies from the literature which analyze source code.
- We propose a taxonomy of TD, extended to incorporate specific concepts related to R packages.
- A dataset of manually labelled TD instances in R peer-reviews documentation is made available publicly.

Paper Structure. Section II highlights related works on TD in traditional software and data science applications, and R-related research. Next, Section III describes our methodology by outlining our goal and research questions, discussing the dataset's construction, and elaborating on the data analysis. Section IV presents the results of our research questions, and Section V discusses the implications of our results for the broader area of R programming and TD. Finally, Section VI acknowledges the threats to the validity of the study, and Section VII concludes this work.

[4]https://github.com/ropensci/software-review/

[5]https://www.bioconductor.org/developers/package-submission/

II. Related Work

This section gives an overview of the relevant literature regarding TD taxonomy and classification, TD in data science, and software engineering research in R.

Technical Debt Taxonomy. The most common types of TD include *code, design, architecture, documentation, test (and test automation),* and *defect* [22], referring to debt in the various phases of the software development life cycle. However, despite being criticised for "diluting" the TD metaphor [23], other aspects of software that causes issues in the software development process have created additional types of debt, namely, *build* [24], *service* [25], *versioning* [26], *usability* [27], *people* [23], *process* [28], *social* [29], *database (design)* [30], *environmental* [31], *data* [28], and *infrastructure* [32]. However, according to surveys of the literature on TD, these debts have generated interest among the research community and have been adopted as additional types of TD in the software development process [22, 33].

Technical Debt in Data Science Applications. TD in Machine Learning (ML) systems was first discussed by Sculley et al. [34]. They explored multiple hidden TD in ML systems, including entanglement, data dependencies and configuration issues, which can adversely impact the system design. Breck et al. [35] proposed specific tests to reduce TD and ensure ML systems' production readiness. Self-Admitted Technical Debt (SATD) in deep learning frameworks has been investigated to show that the frameworks contain *design debt* as the major TD type, followed by debts in the *defect, documentation, test, requirement, compatibility,* and *algorithm* categories [36]. Although not explicitly mentioning TD, several studies have developed guidelines for the quality assurance of machine learning systems [37], developed tools for data validation [38], or studied the software engineering practices for machine learning [39, 40].

Software Engineering Research in R Programming. A recurrent topic of research in R is the analysis of package and dependency networks, e.g., dependency management and the impact of GitHub in non-CRAN packages [17], measuring package activity and lifecycle [41] and the growth and maintainability capabilities of CRAN packages [42]. Another popular area, given R's mix of programming paradigms, is programming language theory, addressing problems such as code profiling [43], wrappers to other languages (e.g., C++) [44], type systems [45], and lazy evaluation [46].

Several studies have been conducted in traditional software code reviews [13, 47] but only a few have focused on scientific software reviews [11, 48, 49]. However, this study is focused on R's peer review documentation and code review is outside the scope of this work.

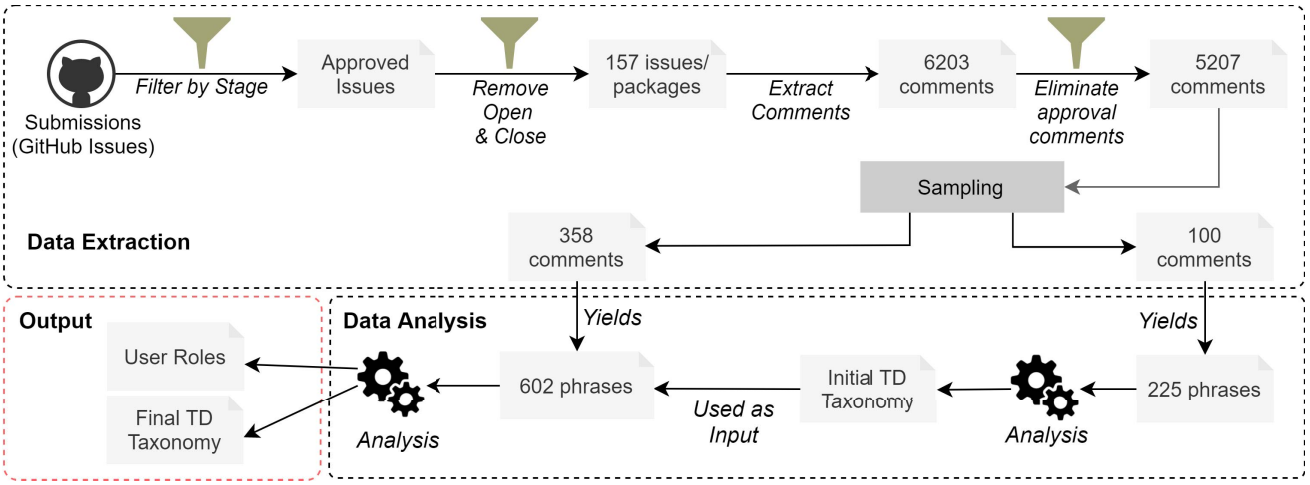

Fig. 1. Schematic Diagram of the Study

The study of TD in machine learning goes back to 2015 but has since gained popularity, with recent studies on SATD in deep learning applications. Although R is one of the main languages used in data science applications, we are not aware of any research investigating TD in the R language. Besides, many of the TD-related researches explore the TD in source code. However, in this study, we are exploring TD in the documentation of R packages' peer-reviews.

III. Methodology

A. Goal and Research Questions

This study investigates the presence of TD in the documentation of the peer-review process of scientific software, particularly in the *rOpenSci* case. Specifically, this led us to the following research questions:

- **RQ1: What types of TD are analyzed during the peer-review process?** The rationale behind this question is to determine the different types of TD that will emerge from the peer-review process documentation, as reported by the different user roles. We will use the types of TD associated with traditional software (as listed in Section II) as a starting point to build a taxonomy specific to scientific software.
- **RQ2: What is the distribution of TD according to the types identified in RQ1?** We want to determine the distribution of the different types of TD in the documentation of the peer-reviews and analyze which debt is the most and least prevalent in the peer-reviews.
- **RQ3: Are identified TD types independent of the user roles?** We investigate the association between the user roles (as described in Table I) and TD types. We explore the differences among the TD types identified for each user role. This will give us insights into the differences among user roles in terms of the TD types they report.

B. Data Extraction

rOpenSci handles every submission to be peer-reviewed as a GitHub issue in their Software Review Repository[4]. For more information about *rOpenSci*'s review process, the Peer-Review Guide is openly available[6]. Overall, there are two types of submissions: *pre-submissions* (in which the authors enquire about the fit of the proposed package to *rOpenSci*'s scope), and *submissions* (referring to the software peer-review process) [20]. For this study, the latter was considered. GitHub "labels" are used to indicate the stage in which the review is at.

A submission starts with an *author* creating a new GitHub issue using the "package submission" template[7]; after some initial checks and questions from the *handling editor*, at least two *reviewers* are selected and invited by the *handling editor*. The *reviewers* must have experience both on the topic and R. Each of them needs to conduct a review of the package and submit their revision using the review template[8]. The process continues as a back-and-forth communication between *reviewers* and *author*, moderated by the *handling editor*. Once the package is approved, the handling editor posts an approval comment[9] and the issue is closed afterwards. Additional communication related to the administrative on-boarding process, but not any package discussion, may happen.

A label identifies whether a submitted package for peer-review is still `under review` (and in which step of the process), `withdrawn`, `on-hold` or `approved` to be published by *rOpenSci*. The approved packages are the ones for which the review process is completed, and the package authors have applied the feedback they have received. We developed an R script using the GitHub API and mined the reviews from the GitHub issues of the approved submissions.

[6]https://devguide.ropensci.org/softwarereviewintro.html
[7]https://bit.ly/38sR7bt
[8]https://devguide.ropensci.org/reviewtemplate.html#reviewtemplate
[9]https://devguide.ropensci.org/approvaltemplate.html

This process is described next and can be visualized in the "Data Extraction" phase of Figure 1.

First, we extracted issues tagged as `approved`. This indicates that an issue is a *submission* for peer-review and its review has been completed and the package is approved for "onboarding" (i.e. akin to "publication") in *rOpenSci*. The packages with `open` or `close` tags were disregarded, as *rOpenSci* issues can remain open for reasons unrelated to the review. Following this, we extracted **157 peer-review issues to be studied**, which correspond to 157 different packages. This dataset was converted to a CSV file, which contains data about: *labels assigned, the issue opening comments, dates for creation, update and closing, number of comments of the issue, the handle of the opening author, title, issue number, issue ID and URL*. Every row of this dataset belongs to a single issue.

The 157 approved issues have a timestamp for the creation and the time the issue is closed. We use the difference between these two timestamps to determine how many days it takes for packages to be reviewed and approved after they are submitted. The median duration for issue approvals is 104.7 days, with the 0.25 percentile of 69.7 days and 0.75 percentile of 193.1 days. In terms of outliers, there are five packages for which the duration is between 400 and 550 days.

Second, we used the unique *issue number* in combination with the GitHub API to extract the comments for each issue. It should be noted that there are multiple comments associated with each issue, posted by package authors, reviewers, or editors of *rOpenSci*. The minimum number of comments per issue is one, the maximum is 113, and the median is 37. We obtained **6203 individual comments for this complementary dataset**. In particular, it includes the following fields: *unique comment ID and URL, issue number, dates for create and update, user handle of the author, and the comment body*. Here, each row belongs to a comment, and multiple rows may link back to the same GitHub issue.

It is worth mentioning that:

- Selected reviews were not discriminated by year, considering issues from *rOpenSci*'s peer-review dates back to 2011; however, the early reviews (about the first six months since *rOpenSci*'s foundation) did not have a well-defined peer-review process.
- *rOpenSci* uses a pre-built template for issue openings, where the authors have to input specific data, akin to the submission form in an academic journal. Therefore, it was established that these opening posts did not contain any discussion regarding the package and were excluded from the analysis.

Often, the discussion may continue after the package's acceptance; however, these comments are related to the "publication" of the package in *rOpenSci*, as well as its promotion on different academic networks. As a result, these discussions do not focus on the quality of the code and do not include any additional TD instances.

The comments were filtered in a two-fold process: first, an automated search for the "approval template,"[9] eliminating all "default" and second, a manual search eliminating comments indicating approval without using the template. Although the "approval template" is suggested by *rOpenSci*, it is not enforced, and editors may customize it. This process reduced the comments to 5207.

Lastly, we sampled[10] a subset of the 5207 comments with a confidence level of 95% and confidence interval of 5, resulting in 358 comments. This sample is used for manual investigation and further analysis. We extracted a random sample of 100 comments initially as a learning phase for RQ1 before analyzing the 358 comments. The two authors who applied the manual investigation have previous experience with TD. One of the authors has extensive research experience with TD (8+ years), and the other is an R expert with research experience in TD.

C. Data Analysis

In this section, we describe how we extracted instances of TD and user roles from the comments. This process can be visualized in the "Data Analysis" phase of Figure 1.

1) Comment Classification: To classify the phrases, we used a *card sorting technique* [50]. Card sorting is commonly used to derive taxonomies from data. Our card sort was closed, that is, we used a pre-defined list of TD types. We extracted the different types of TD from the literature and compiled a list of the 17 types of debt, as enumerated in Section II.

We performed two iterations to classify the comments. In the first iteration, we randomly picked 100 comments from the 5207 comments. This phase resulted in an initial taxonomy of 10 different types of TD. The purpose of this phase is two-fold: (1) familiarizing the authors with the R terminologies and the peer-review process, (2) extracting an initial taxonomy of TD. In the first iteration, out of the 100 comments, 36 had TD and were further broken down into 225 phrases for analysis. The rationale for breaking the comments into phrases is that the comments were rather lengthy and contained different TD types. Each comment was broken down first into paragraphs and then into sentences (by either checking for a period or a semi-colon). If two sentences (or more) were related and referred to the same point, they were kept together as one phrase. Two of the authors separately manually classified them according to the 17 types of TD using the closed card sorting technique. Out of the 225 phrases, the manual classification led to 44 disagreements. They discussed the disagreements and finalized the classification of these 44 phrases. This first iteration reduced the list of 17 types of TD to 10, namely: *architecture, build, code, defect, design, documentation, requirements, test, usability and versioning*.

In the second iteration, we randomly sampled 358 comments from the remaining in the documentation, ensuring that they do not have an intersection with the sample of 100 used in the first phase. Out of the 358 sampled comments, we manually extracted 95 that contained TD and discarded the remaining 264 with no TD. The 95 comments were broken down into 602 phrases. The first two authors manually classified

[10]https://www.surveysystem.com/sscalc.htm

the phrases, but this time, according to the taxonomy in the first iteration. If a phrase could not be classified, it is marked with a question mark to be discussed later. Then, the authors discussed all of the disagreements and the cases they were unsure about its category. In the discussions, the authors decided on the final classification of the disagreements. During the second iteration, no additional types of debt were uncovered, suggesting that the 100 comments were pretty representative of the dataset regarding the different types of TD present. Therefore, the taxonomy of TD was finalized. Two phrases out of the 602 were removed as the authors could not decide whether it was a debt or not, yielding a total of 600 phrases that were classified.

Revision "comments" in *rOpenSci* are large since the process is derived from academic peer-review; they are organized in paragraphs and sentences. Each comment was broken down into paragraphs and then into sentences (by either checking for a period or a semi-colon). If two sentences (or more) were related and referred to the same point, they were kept together as a *phrase*. Here, the word 'comment' refers to GitHub's 'comment' on an issue (where the review process is conducted) and is not representative of the text's length.

TABLE I
ASSIGNED ROLES AND THE PROCESS USED TO EXTRACT THEM

Role	Source
op_author. Equivalent to a corresponding author.	Assigned to the author that opened the issue. This is extracted from the dataset during the Data Extraction phase.
handling_editor. The associate editor handling the issue in the same sense as in academic journals.	*rOpenSci* assigns (in GitHub terms) this user to the issue. This was extracted from the first dataset and automatically labelled.
author. Other authors or maintainers of the package.	Often mentioned in the opening post, they were only labelled if their GitHub handles were mentioned. The assigned roles of the GitHub handles were extracted by reading all the comments for an issue.
editors. Other editors, such as the Editor in Chief	They were manually extracted from the dataset; this role often checks the scope suitability of the package and assigns a *handling_editor*.
reviewers. The experts assigned as referees for the submission.	They were found by inspecting the *handling editor* post, in which these are often mentioned, and by inspecting the reviews in search for the "review template"[8].

Then, we calculated the inter-rater reliability using Cohen's Kappa coefficient [51]. Cohen's Kappa is a test that measures the raters' agreement in studies that have two or more raters responsible for measuring a categorical scale variable. The test results in a number between -1 and $+1$, where -1 shows the highest disagreement and $+1$ shows the most agreement rate. The threshold cutoff for deciding on the high agreement varies based on the fields [51]. We use the score of above 0.79 as a high agreement rate, which is used in previous and similar studies in software engineering [36, 52]. We obtain the result of $+0.80$, which indicates the high rate of agreement and reliability of our coding schema.

2) User Roles: Each comment in the collected dataset is written by a user (identified by GitHub handles), and each user has a different role. The values of the user roles are either pre-defined or null. In this step, for each comment of an issue, we extract the user roles for each of the user handles, as the same handles can have different roles in multiple issues. For example, a user can be an *author* of a package in one issue and a *reviewer* of another package. The pre-defined roles and their extraction process, are summarised in Table I.

D. Replication Package

All the data used in our study are publicly available.[11] Specifically, we provide the R scripts that we used to extract the comments and perform the data analysis and processing, and the working data sets (phrases with TD type and user roles reporting them) used to run the statistical analysis.

IV. RESULTS & DISCUSSION

The following subsections present the results organized by research questions and a brief discussion.

A. RQ1: TD Taxonomy for R

Table II presents the adapted definitions of the types of TD found through the manual analysis process (see Section III-C). The definitions are extracted from well-known academic sources, cited in the table, and then expanded or generalized to fit R concepts. For each type of debt, the table also presents an example extracted from the manual sample.

The manual inspection highlighted three *perspectives* that group the TD types in the domain of R programming in terms of 'who' is affected by each type of debt the most. They were extracted during the manual inspection process considering R's intrinsic characteristics and previous research findings that linked different TD types together. They were devised according to 'who' was affected by the debt, according to what the person wrote–this was obtained by exploring the context of each sentence. Often, they mentioned why something was problematic. For example, they would say, "it is better for the user if...", or "a user may find it confusing if...". Therefore, this classification into perspectives comes from the data, but there is a possibility that there are overlaps. Moreover, note that the reviews did not mention or reference TD explicitly. Figure 2 presents the grouping as a graph, with perspectives in the mid-level and types of debts as leaves. These perspectives are:

- **User:** TD types grouped under this category are *usability*, *documentation* and *requirements*. These debts ultimately affect mostly the user of the package – i.e., the person who installs and imports the package into their own project – than the developers. Previous work have established that *documentation debt* is strongly related to (and sometimes caused by) *requirement debt* [59]. Moreover, other studies have demonstrated that poor implementation of requirements affects the *usability* [60].

[11] See: http://doi.org/10.5281/zenodo.4589573

TABLE II
TAXONOMY OF EXTENDED TD DEFINITIONS FOR R, WITH AN EXAMPLE EXTRACTED FROM THE MANUALLY CLASSIFIED SAMPLES.

Debt Type	R Definition	Example
Architecture	Refers to the problems encountered in product architecture, for example, violation of modularity, which can affect architectural requirements (e.g. performance, robustness) [53, 54].	"A common solution is to split the general tool and the specific application into separate packages. This isn't required, but it is important to clearly separate the components of the code base and the documentation for both the EHR manipulation and the CCHIC use-case."
Build	Refers to issues that make the build task harder, and unnecessarily time consuming. The build process can involve code that does not contribute to value to the customer. Moreover, if the build process needs to run ill-defined dependencies, the process becomes unnecessarily slow. When this occurs, one can identify build debt [24]. In the context of R, build debt encompasses anything related to Travis, Codcov.io, GitHub Actions, CI, AppVeyor, CRAN, CMD checks, devtools::check.	"No problems installing from a local tarball, but problems arose after installing using `devtools::install_all()`"
Code	Refers to the problems found in the source code that can negatively affect the legibility of the code making it more difficult to maintain. Usually, this debt can be identified by examining the source code for issues related to bad coding practices [55]. In the context of R, code debt encompasses anything related to renaming classes and functions, <- vs. = , parameters and arguments in functions, FALSE/TRUE vs. F/T, print vs warning/message.	"There were some instances where you used system variable names as local variable names in functions, e.g. line 36 in `auk_time.r`, `time <- paste0(ifelse(nchar(time) == 4, "0", ""), time)`. It obviously doesn't cause an issue as is, but maybe it could down the line."
Defect	Refers to known defects, usually identified by testing activities or by the user and reported on bug tracking systems [56].	"A similar strategy underlies `safe_hex_color()`, so the method has the same flaws. Plus, something mysterious happens sometimes:"
Design	Refers to debt that can be discovered by analyzing the source code and identifying violations of the principles of good object-oriented design (e.g. very large or tightly coupled classes) [54, 57]. In the context of R, design debt encompasses anything related to S3 classes and S4 methods, exporting functions with @export or the name pattern (visibility), internal functions with coupling issues, location of functions in the same file, selective importing @import (whole package) or @importFrom (a specific function), notations packageName::functionName and package:::function, returning objects (dataframes or tibbles), and Tidyverse vs. baseR.	"**Internal functions** do not need to be exported (e.g. `parse.info`, `read.prov`). Remove the `@export` tag for these functions and add the tags `@keywords internal` and `@noRd`."
Documentation	Refers to the problems found in software project documentation and can be identified by looking for missing, inadequate, or incomplete documentation of any type [58]. In the context of R, documentation debt encompasses anything related to Roxygen2 (e.g., @param @return, @example), Pkgdown, Readme files, and Vignettes.	"It is not documented that the date-times are integers converted to POSIX times, so they are all after 1970."
Requirements	Refers to trade offs made with respect to what requirements the development team needs to implement or how to implement them. Some examples of this type of debt are: requirements that are only partially implemented, requirements that are implemented but not for all cases, requirements that are implemented but in a way that does not fully satisfy all the non-functional requirements (e.g. security, performance) [23].	"Uploading multiples files+directories, however, is still very much a work in progress (i.e., not functional). It's something I would personally use quite a bit in my own work."
Test	Refers to issues found in testing activities that can affect the quality of those activities. Examples of this type of debt are planned tests that were not run, or known deficiencies in the test suite (e.g. low code coverage) [58]. In the context of R, test debt encompasses anything related to coverage, covr, unit testing (e.g., testthat), and test automation.	"Test coverage is good overall but sticks mostly to 'happy path'. Untested error paths appear in `sync.R`, `utils.R`, `wget.R`, `postprocess.R` etc. I feel like a jerk for bringing this up. I know how unfun it is to write test cases for these."
Usability	Refers to inappropriate usability decisions that will need to be adjusted later. Examples of this debt are lack of usability standard and inconsistency among navigational aspects of the software [27]. In the context of R, test debt encompasses anything related to usability, interfaces, visualization and so on.	"A better error might say, `see ?ch_generate` or print out the `all_choices` vector."
Versioning	Refers to problems in source code versioning, such as unnecessary code forks [26].	"I note only that four levels of versioning might be overkill - current version is *0.0.4-1*. Up to the authors, but with just 3 levels you'd now be up to *0.0.11* (or maybe *0.2.x*), which still allows an approximately infinite future before version numbers get too high, and makes it easier for others to track version numbers."

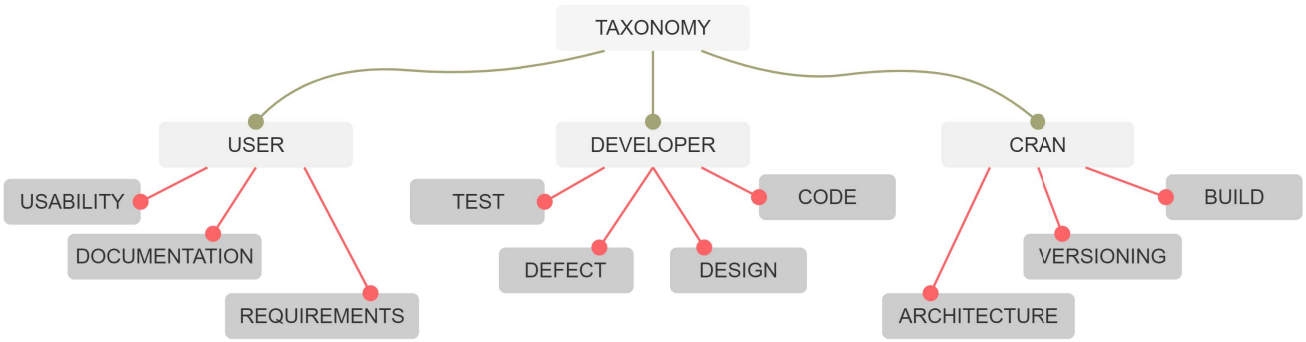

Fig. 2. Summary of the proposed taxonomy of TD for R, grouping each *type* (leaf level) in *perspective* (mid level).

- **Developer:** *Design* and *code debt* are often viewed as 'two sides of the same coin' [61], intrinsically related in terms of code quality [62]. A key point in common of *code, design, test,* and *defect* debt is that they are introduced and addressed during the development, testing and maintenance phases of the software development lifecycle [56]. Previous studies have also demonstrated that developers focus on these four types of debt the most [63], and that *defect* tends to be introduced as a result of the other three types of debt [58].
- **CRAN:** CRAN performs automated checks on package size, structure, and versioning, among others when a package is uploaded[12]; thus, even if a package will not eventually be published there, it still goes through the automated check. Overall, these types of debt affect the maintainability of the software, which is what CRAN checks attempt to improve as well [42].

Findings #1. We extracted ten different technical debt types from the peer-review documentation and proposed a taxonomy pertaining to three different perspectives, namely: user, developer, and CRAN.

B. RQ2: TD Distribution by Type

Figure 3 depicts the distribution of the 600 phrases according to the ten TD types identified earlier. We found that the most prominent type of debt is *documentation debt*. We had 186 occurrences out of 600, making up for almost 30% of all the instances of debt. This is followed by *code debt* (representing 15% of the phrases) and *design and defect debt* (each representing 12% of the phrases). It is worth noticing that every occurrence of the same type of TD (if referring to different cases) is accounted for in this calculation.

The other types of debt which were sparse were *test debt, requirements debt, architecture debt, build debt, usability debt, and versioning debt*. They each represented 4−7% of the phrases. *Versioning debt* was the least reported type of TD

[12]https://r-pkgs.org/r-cmd-check.html

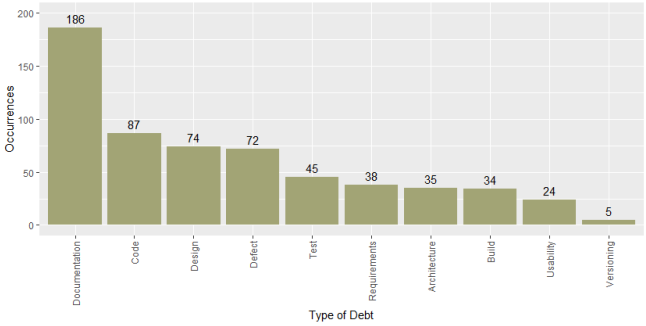

Fig. 3. TD Distribution by Type

with only five instances out of 600. On average, each comment has six phrases which we encountered as TD, with a minimum of one, maximum of 50, and median of two. We did not notice any same TD phrase appearing in the same review.

Findings #2. Documentation debt is the most prominent type of debt, while versioning debt is the least prominent type of debt in the dataset.

C. RQ3: Association of User Roles and TD Types

We manually reviewed the issues' comments to identify the user roles. This manual investigation ensures that we extract all the roles for a comment since a user can have different roles in different issues. Next, we use the comment's unique identifier to label the roles for each of the 600 phrases. The distribution of phrases for user role is shown in Table III.

TABLE III
TECHNICAL DEBT COUNTS BY USER ROLES

Role	TD Phrase Count
op_author	48
handling_editor	68
author	5
editors	5
reviewers	474

As the number of phrases written by *authors* and *editors* are low, we combined the *author* with *op_author*, and the *editors* with *handling_editor* phrase counts. The *op_author* and *author* are developers of a package; and the responsibilities of *handling_editors* and *editors* are also similar to each other. For RQ3, we refer to these combined roles as *author* and *editor* respectively. This combination is also confirmed by one of the current editors of the *rOpenSci*. Therefore, combining these roles will not affect the reliability of the results.

We applied the Chi-Square test and Cramer's V to find the association between the three user roles (i.e. *author*, *editor*, and *reviewer*) and the TD types. The Chi-Square test is used to find the independence among two categories, where each category has multiple levels (more than two), and Cramer's V shows the strength of this association [64]. A main assumption of the Chi-Square test is that the expected frequencies in the contingency table are not less than 5 for more than 20% of the cell values [64, 65]. This assumption is not satisfied in our case, as the number of phrases labelled as *design*, *versioning*, and *usability* by *authors* and *editors* are low. Therefore, we use the three TD perspectives discussed earlier (i.e. User, Developer, and CRAN) instead of the TD types. Our null hypothesis is as follows:

Null Hypothesis H_0: The TD types in the comments do not depend on the user role associated with the comment's writer.

This hypothesis is rejected with a p-value of $4.2E-08$ for $\alpha = 0.05$ and a Cramer's V value of 0.18. Therefore, the TD types and the user roles are dependent but with a small association. We also applied Post Hoc tests [64] to identify the differences between the user roles pairwise: *author* vs. *editor*, *author* vs. *reviewer*, and *editor* vs. *reviewer*. As the number of comparisons is more than two, Bonferroni Correction [64] is applied to adjust the alpha value ($\alpha = 0.016$). The test results show that there is not a significant difference between the *author* and *editor* in terms of TD types in the comments they wrote. However, the *reviewer*'s role affect the TD types we identified in the comments. A significant difference is found between the *editor* and *reviewer* roles, and *author* and *reviewer* roles with p-values of $5.5E-04$ and $5.8E-08$, respectively. The effect size (i.e. strength of the association) using Cramer's V score shows a small association between the *editors* vs. *reviewers* (0.16) and a medium association among the *authors* and *reviewers* (0.25).

Figure 4 represents the percentage of TD phrases in each category, where we separated all phrases of each TD by user roles. For all TD types except *versioning debt*, *reviewers* have the highest number of comments, with *design debt*, *code debt*, and *usability debt* having the highest percentages, 90.5%, 89.6%, and 87.5% respectively. This is followed by 85.4% for *documentation debt*, 79% *requirements debt*, and 72% for *defect debt*. *Test debt*, *architecture debt* and *build debt* reported by *reviewers* are 60%, 60%, and 50% respectively. 40% of the *versioning debt* that we identified are reported by the *reviewers*, 40% by the *editors* and 20% by the *authors*.

The percentage of the phrases identified as *documentation debt*, *test debt*, and *usability debt* written by *editors* are the highest (12%, 22%, and 12.5% respectively). The next highest are the ones written by the *reviewers*. We found a small percentage of *documentation debt* and no instances of *usability debt* among the phrases written by *authors*. The percentage of phrases with *requirements debt* written by the *authors* is approximately twice as many as the *editors*. The percentage of phrases classified as *architecture debt* and *defect debt* are the same for *authors* and *editors* - 20% and 13.8% respectively. The percentages for the *authors* and *editors* are very close for *design debt* and *code debt* (4-6%). The highest percentages belong to *build debt* with 23% and 26% encountered by *editors* and *authors* respectively.

We also separated the phrases written by each user role to determine the percentage of each TD type for each user role. These TD distributions are shown in Figure 5. The *authors* have *defect debt* as the highest TD (18.8%) instances reported, followed by *build debt*, *test debt*, and *architecture debt* with 16.9%, 15%, and 13.2%, respectively. The next two highest percentage of the TD phrases for *authors* belong to *code debt* and *requirements debt* with 9.4% each. There is no phrases with *usability debt* and the *versioning debt* percentage is also very small ($< 2\%$) for the *authors*. The debt instances identified by the *editors* are mostly *documentation* (31.5%), followed by *defect* and *test* (13.7% each), *build* (10.9%) and *architecture* (9.5%). The percentages of phrases with *code*, *design*, *requirements*, and *usability* are almost the same (4-5 percent). *Versioning* is the least reported (2.7%) by *editors*. *Reviewers* report mostly *documentation* (33.5%), followed by *code*, *design*, and *defect* with 16.4%, 14.1%, and 10.9% respectively. The percentage for the other categories is less than 10%. Similar to other roles, *versioning* is the least reported with only 2 instances (approx. 0.4%).

> **Findings #3.** The user roles affect the quantity and the TD types that are reported. The *reviewers* report more TDs than other user roles. Moreover, the type of TD encountered by each user role is different. The *reviewers* focus on the *documentation*, *code* and *design* debts more than others. The *authors* concentrated mostly on *defect*, *test*, and *architecture* debts. The *editors* pay more attention to *documentation*, *defect*, and *requirements* debts.

D. Discussion

Our findings show the number of instances for each TD type encountered by each user role differ greatly. For example, in our sampled data, the *authors* pay more attention to *defect debt* and less attention to *usability* and *versioning debts*. This can be due to the familiarity of the developers with their own package and being aware of potential defects and limitations. The *editors* and *reviewers* on the other hand, pay more attention to the *documentation debt* (above 30% of their phrases is identified as documentation). This aligns with the reasons for the creation of R. R is a community created with the intention of being fully open source [14, 21]. As a result, its focus on

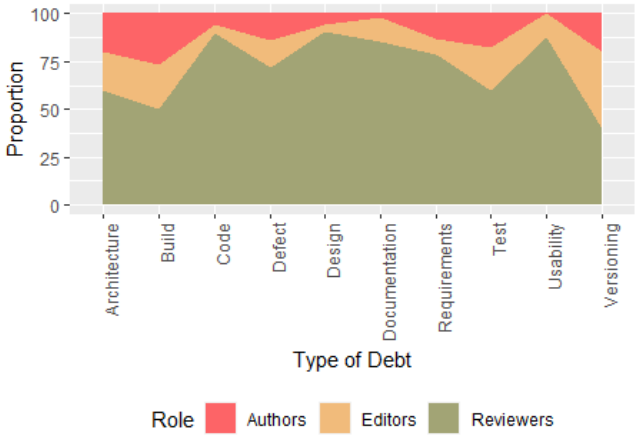

Fig. 4. The Percentage of TD Types Compared by User Roles

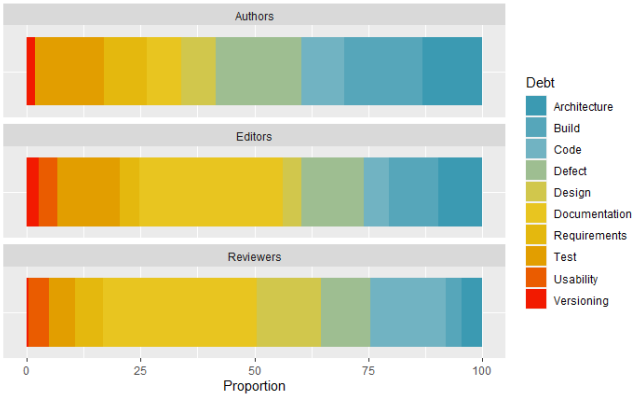

Fig. 5. The Distribution of TD Types Found in the Phrases Written by Each User Role (Shown in Percentages)

documentation is well-known [66], as well as the emphasis on the maintainability of contributed packages – for example, CRAN automatically compiles the package's documentation to provide a PDF-style document alongside the package [42]. The results mentioned above are an eye-opener for each of the user roles on *rOpenSci*. Although managing technical debt is vital to the software quality, not all users give the same importance to the different types of debt. As we expected, the *reviewers* reported more TDs compared to others. We relate this result to the fact that the reviewers' responsibility is to ensure that the approved packages achieve a certain quality. This is reflected both in the high number of TD phrases in our study that are written by *reviewers* and the significant difference of the distribution of TD types of *reviewers* and the two other roles.

The taxonomy of TD proposed in this paper is based on existing proposals for OO Programming. As a result, the definitions had to be adjusted to fit the characteristics of R, a functional dynamically-typed language. However, it is worth noting that several types of TD were not found in the discussions —examples are *service* and *installation debt*. This may be due to either difference in the language or the "topic scope" of *rOpenSci* which does not provide the opportunity to report such types of debt.

Other types of debt, specific to scientific software have been proposed in the literature, such as *algorithm* [36] or *reproducibility debt* [34]. However, these definitions have been lightly explored as they are recent proposals. A more in-depth study of R packages should be conducted to understand their impact on R fully. It is also worth noting that the proposed taxonomy and the frequencies of the different debt types may be different from those reported in SATD studies. SATD occurs when developers admit poor solutions in the source code comments [3]. Possible differences between the debt types in our work and the SATD studies may be hypothesized to be centered around the presence of other types of debt such as *code* and *design debt*.

V. Implications

Implications for Researchers. Our study shows that not all types of TD are equally prevalent in R packages. Our findings reveal that *documentation debt* is the most recurrent type of debt. In general, we know that documentation debt is one of the least studied types of debt. This reinforces the need for *documentation debt* to be studied more extensively. For instance, there is a need to establish what constitutes good documentation to minimize *documentation debt* or what guidelines to follow when writing software documentation, both traditional and scientific. This is the first study to explore a taxonomy for TD for scientific software to the best of our knowledge. We conducted our study in the context of R. This provides opportunities for researchers to investigate TD in other scientific software using our taxonomy as a starting point and augment it. The fact that comments from the different user roles focus on different types of debt is intriguing. *Reviewers* seem more concerned with creating a user-friendly package in terms of documentation, while *authors* seem focused on the package's structure. Further studies need to be conducted to understand the rationale behind these distinctions.

Implications for the R Community. To the authors' knowledge, this is the first taxonomy for R packages, including tailoring existing definitions to the intrinsic characteristics of the R language. For developers, this will allow the specification and elaboration of specific smells in the future, leading to better coding practices that will enhance the quality of the available packages. Moreover, these concepts can also be used when training new developers (i.e., by organizing workshops). Regarding the peer-review of R packages, the revision process (and specifically, the templates used by the referees) can be upgraded to include points that tackle those types of TD that are less encountered. The submission template used by authors can also be altered to emphasize on documentation and replication, as it is essential for the usability of the packages. Moreover, the results of our study reveal the importance of the reviewers' role in reviewing packages. Although none of the users discuss TD directly,

nor mention the metaphor, their commitment to quality is perceived through their comments. Although we did not directly trace the application of this feedback in the source code, this is validated through the review process of *rOpenSci*. Based on the number of instances found for each TD category, we recommend that researchers and practitioners require new solutions (such as updating the peer-review criteria and updating CRAN checks) to ensure all TD types are inspected in the peer-review process of R packages. Currently, the most prevalent debt types encountered are *documentation* and *requirements*, and the ones with least occurrences are *usability* and *versioning*. So, we suggest researching new ways to check how *usability* and *versioning* technical debts are addressed in R package peer-reviews.

VI. Threats to Validity

External Validity. External validity refers to the ability to generalize results. Our study was conducted on the documentation of the peer-review of R packages. We cannot draw general conclusions about the TD in other scientific software. Despite having a small sample size to start with (95 out of the 358 comments had TD), the comments were quite lengthy and needed to be broken down into phrases, each one pertaining to a different type of debt. Instead, we ended up with a sample of 600 phrases or instances of TD. However, the goal of our study is not to provide an analysis of TD that applies to all scientific software but to serve as a baseline for future studies in scientific software analysis or peer-reviews. It is worth noting that the results are based on the whole ecosystem of rOpenSci, but similar peer-review processes have been used in other communities (i.e. BioConductor, pyOpenSci and Journal of Open Source Software). Therefore, our results can be applied to those communities. However, these results would not be applicable to OO or functional languages.

Internal Validity. This threat refers to the possibility of having unwanted or unanticipated relationships. As we selected a sample of review comments in our analysis, sampling bias might impact our conclusions. To mitigate this threat, we randomly selected the comments with a confidence level of 95% and a confidence interval of 5 for our classification and did not have any restrictions on the timeline for the extraction. Therefore, our results are not bound to a certain period. In addition, to build our taxonomy, we used types of TD previously defined in the literature as a starting point. However, these debt types have been used extensively in both research and practice and validated by many studies.

Moreover, the manual process of classifying the comments according to debt types can be biased. To mitigate this threat, the classification was performed by two of the authors separately and then compared to finalize the classification. The card sorting method might be a potential source of bias, and despite the rigor followed by the authors to classify the comments, replication of this study might lead to different results. To mitigate this threat, the authors thoroughly discussed the conflicts and assigned the final debt type to the phrases which were the source of conflicts. Lastly, as we are using a different artifact in the format of the documentation of the peer-review of R packages as our dataset, there might be a threat associated with the authors being unfamiliar with the dataset as they usually deal with artifacts from traditional software development such as source code or code comments. To alleviate potential problems that this might introduce in our analysis, we used 100 records of the comments as a learning phase. This learning phase ensures that our authors become familiar with the dataset and the discussions for R package peer-reviews and ensure that the authors agree on the context of TD in each category.

Construct Validity. Construct validity refers to the degree to which a test measures what it claims to be measuring. Some of the debt types had a low number of occurrences after the classification and can be a potential bias for our findings in RQ3. However, the Chi-Square test does not have any assumption about the normality or the frequency of the sample data, and therefore, the test results are reliable. This ensures the association between the user roles and TD types, with an effect size (i.e. Cramer's V score). Besides, to further analyze the results and find out the differences among each pair of the TD type distributions, we applied Chi-Square Post Hoc Tests. This step identifies the pair-wise comparison among the distributions for user roles, reported in the results section.

VII. Conclusions and Future Works

In this study, we manually analyzed the technical debts in the documentation of R packages' peer-review comments. We proposed a taxonomy of ten different TD types pertaining to three different perspectives. Our findings reveal that some technical debts are prominent, and some appear with a low frequency. These TD occurrences significantly differ according to different user roles. The results of the statistical tests also confirm the dependency and differences between the TD types and user roles. The proposed taxonomy, investigating TD in the comments of the peer-review R packages, and differences among user roles are the main contributions of this study. In addition, we provide access to our scripts and datasets in the replication package (see Section III-D). We conclude with some implications for researchers and practitioners in the R community based on our findings.

The next step is to replicate this study but in a different domain. We plan to analyze the technical debt in *BioConductor* packages to investigate the differences among domain-specific and more general package reviews and extend our findings to be as generalizable as possible. *BioConductor* only onboards packages oriented to work with genomics data, while *rOpenSci* has a wider range of on-topic packages, including data munging, API connections, and geospatial-related packages. Following that, we will automate the identification of TD types in the comments of peer-review packages in both R and the *BioConductor* domain.

VIII. Acknowledgements

We thank Scott Chamberlain, Noam Ross and Karthik Ram, Associate Editors of *rOpenSci* and the MSR reviewers.

REFERENCES

[1] N. A. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord, and I. Gorton, "Measure it? manage it? ignore it? software practitioners and technical debt," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 50–60. [Online]. Available: https://doi.org/10.1145/2786805.2786848

[2] G. Bavota and B. Russo, "A large-scale empirical study on self-admitted technical debt," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, 2016, pp. 315–326.

[3] G. Sierra, E. Shihab, and Y. Kamei, "A survey of Self-Admitted Technical Debt," *Journal of Systems and Software*, vol. 152, pp. 70 – 82, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121219300457

[4] S. Bellomo, R. L. Nord, I. Ozkaya, and M. Popeck, "Got technical debt? surfacing elusive technical debt in issue trackers," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, 2016, pp. 327–338.

[5] L. Xavier, F. Ferreira, R. Brito, and M. T. Valente, *Beyond the Code: Mining Self-Admitted Technical Debt in Issue Tracker Systems*. New York, NY, USA: Association for Computing Machinery, 2020, p. 137–146. [Online]. Available: https://doi.org/10.1145/3379597.3387459

[6] F. Khomh, M. Di Penta, and Y.-G. Gueheneuc, "An exploratory study of the impact of code smells on software change-proneness," in *the 16th Working Conference on Reverse Engineering*, ser. WCRE '09, 2009, pp. 75–84.

[7] T. Hall, M. Zhang, D. Bowes, and Y. Sun, "Some code smells have a significant but small effect on faults," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 23, no. 4, pp. 1–39, 2014.

[8] Z. Codabux, K. Z. Sultana, and B. J. Williams, "The relationship between traceable code patterns and code smells." in *SEKE*, 2017, pp. 444–449.

[9] Z. Codabux, K. Z. Sultana, and B. J. Williams, "The relationship between code smells and traceable patterns—are they measuring the same thing?" *International Journal of Software Engineering and Knowledge Engineering*, vol. 27, no. 09n10, pp. 1529–1547, 2017.

[10] K. Z. Sultana, Z. Codabux, and B. J. Williams, "Examining the relationship of code and architectural smells with software vulnerabilities," in *27th Asia-Pacific Software Engineering Conference (APSEC)*, 2020.

[11] J. Howison and J. D. Herbsleb, "Scientific software production: Incentives and collaboration," in *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work*, ser. CSCW '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 513–522. [Online]. Available: https://doi.org/10.1145/1958824.1958904

[12] C. A. Stewart, G. T. Almes, and B. C. Wheeler, *NSF Cyberinfrastructure Software Sustainability and Reusability Workshop Report*. Indiana University, 2010.

[13] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 931–940.

[14] R. Ihaka, "The r project: A brief history and thoughts about the future," 2017. [Online]. Available: https://www.stat.auckland.ac.nz/\~{}ihaka/downloads/Massey.pdf

[15] D. M. German, B. Adams, and A. E. Hassan, "The Evolution of the R Software Ecosystem," in *2013 17th European Conference on Software Maintenance and Reengineering*, Mar. 2013, pp. 243–252, iSSN: 1534-5351.

[16] S. Theußl, U. Ligges, and K. Hornik, "Prospects and challenges in r package development," *Computational Statistics*, vol. 26, no. 3, pp. 395–404, Sep 2011. [Online]. Available: https://doi.org/10.1007/s00180-010-0205-5

[17] A. Decan, T. Mens, M. Claes, and P. Grosjean, "When github meets cran: An analysis of inter-repository package dependency problems," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, 2016, pp. 493–504.

[18] F. Morandat, B. Hill, L. Osvald, and J. Vitek, "Evaluating the Design of the R Language," in *ECOOP 2012 – Object-Oriented Programming*, ser. Lecture Notes in Computer Science, J. Noble, Ed. Berlin, Heidelberg: Springer, 2012, pp. 104–131.

[19] F. Gilson, M. Morales-Trujillo, and M. Mathews, *How Junior Developers Deal with Their Technical Debt?* New York, NY, USA: Association for Computing Machinery, 2020, p. 51–61. [Online]. Available: https://doi.org/10.1145/3387906.3388624

[20] C. Boettiger, S. Chamberlain, E. Hart, and K. Ram, "Building Software, Building Community: Lessons from the rOpenSci Project," *Journal of Open Research Software*, vol. 3, no. 1, p. e8, Nov. 2015, number: 1 Publisher: Ubiquity Press. [Online]. Available: http://openresearchsoftware.metajnl.com/articles/10.5334/jors.bu/

[21] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. Yang, and J. Zhang, "Bioconductor: open software development for computational biology and bioinformatics," *Genome Biology*, vol. 5, no. 10, p. R80, Sep 2004. [Online]. Available: https://doi.org/10.1186/gb-2004-5-10-r80

[22] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, vol. 101, pp. 193–220, 2015.

[23] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *Ieee software*, vol. 29, no. 6, pp. 18–21, 2012.

[24] J. D. Morgenthaler, M. Gridnev, R. Sauciuc, and S. Bhansali, "Searching for build debt: Experiences managing technical debt at google," in *2012 Third International Workshop on Managing Technical Debt (MTD)*. IEEE, 2012, pp. 1–6.

[25] E. Alzaghoul and R. Bahsoon, "Cloudmtd: Using real options to manage technical debt in cloud-based service selection," in *2013 4th International Workshop on Managing Technical Debt (MTD)*. IEEE, 2013, pp. 55–62.

[26] D. R. Greening, "Release duration and enterprise agility," in *2013 46th Hawaii International Conference on System Sciences*. IEEE, 2013, pp. 4835–4841.

[27] N. Zazworka, R. O. Spínola, A. Vetro', F. Shull, and C. Seaman, "A case study on effectively identifying technical debt," in *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, 2013, pp. 42–47.

[28] Z. Codabux, B. J. Williams, G. L. Bradshaw, and M. Cantor, "An empirical assessment of technical debt practices in industry," *Journal of Software: Evolution and Process*, vol. 29, no. 10, p. e1894, 2017.

[29] D. A. Tamburri, P. Kruchten, P. Lago, and H. van Vliet, "What is social debt in software engineering?" in *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 2013, pp. 93–96.

[30] M. Al-Barak and R. Bahsoon, "Database design debts through examining schema evolution," in *2016 IEEE 8th International Workshop on Managing Technical Debt (MTD)*. IEEE, 2016, pp. 17–23.

[31] D.-L. Magazine, "Technical debt as an indicator of library metadata quality," *D-Lib Magazine*, vol. 22, no. 11/12, 2016.

[32] P. Debois, "Agile infrastructure and operations: how infra-gile are you?" in *Agile 2008 Conference*. IEEE, 2008, pp. 202–207.

[33] N. S. Alves, L. F. Ribeiro, V. Caires, T. S. Mendes, and R. O. Spínola, "Towards an ontology of terms on technical debt," in *2014 Sixth International Workshop on Managing Technical Debt*. IEEE, 2014, pp. 1–7.

[34] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Advances in neural information processing systems*, 2015, pp. 2503–2511.

[35] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "The ml test score: A rubric for ml production readiness and technical debt reduction," in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 1123–1132.

[36] J. Liu, Q. Huang, X. Xia, E. Shihab, D. Lo, and S. Li, "Is using deep learning frameworks free? characterizing technical debt in deep learning frameworks," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Society*, 2020, pp. 1–10.

[37] K. Hamada, F. Ishikawa, S. Masuda, M. Matsuya, and Y. Ujita, "Guidelines for quality assurance of machine learning-based artificial intelligence," in *SEKE2020: the 32nd International Conference on Software Engineering & Knowledge Engineering*, 2020, pp. 335–341.

[38] N. Polyzotis, M. Zinkevich, S. Roy, E. Breck, and S. Whang, "Data validation for machine learning," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 334–347, 2019.

[39] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar,

N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 291–300.

[40] Z. Wan, X. Xia, D. Lo, and G. C. Murphy, "How does machine learning change software development practices?" *IEEE Transactions on Software Engineering*, 2019.

[41] K. Plakidas, D. Schall, and U. Zdun, "Evolution of the r software ecosystem: Metrics, relationships, and their impact on qualities," *Journal of Systems and Software*, vol. 132, pp. 119 – 146, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121217301371

[42] M. Claes, T. Mens, and P. Grosjean, "On the maintainability of cran packages," in *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, 2014, pp. 308–312.

[43] A. Rubio and F. de Villar, "Code Profiling in R: A Review of Existing Methods and an Introduction to Package GUIProfiler," *The R Journal*, vol. 7, no. 2, pp. 275–287, 2015. [Online]. Available: https://doi.org/10.32614/RJ-2015-036

[44] D. Eddelbuettel and R. Francois, "Rcpp: Seamless r and c++ integration," *Journal of Statistical Software, Articles*, vol. 40, no. 8, pp. 1–18, 2011. [Online]. Available: https://www.jstatsoft.org/v040/i08

[45] A. Turcotte and J. Vitek, "Towards a type system for r," in *Proceedings of the 14th Workshop on Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems*, ser. ICOOOLPS '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3340670.3342426

[46] O. Flückiger, G. Chari, J. Ječmen, M.-H. Yee, J. Hain, and J. Vitek, "R melts brains: An ir for first-class environments and lazy effectful arguments," in *Proceedings of the 15th ACM SIGPLAN International Symposium on Dynamic Languages*, ser. DLS 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 55–66. [Online]. Available: https://doi.org/10.1145/3359619.3359744

[47] F. E. Zanaty, T. Hirao, S. McIntosh, A. Ihara, and K. Matsumoto, "An empirical study of design discussions in code review," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3239235.3239525

[48] U. Kanewala and T. Yueh Chen, "Metamorphic testing: A simple yet effective approach for testing scientific software," *Computing in Science Engineering*, vol. 21, no. 1, pp. 66–72, 2019.

[49] B. A. Grüning, S. Lampa, M. Vaudel, and D. Blankenberg, "Software engineering for scientific big data analysis," *GigaScience*, vol. 8, no. 5, 05 2019, giz054. [Online]. Available: https://doi.org/10.1093/gigascience/giz054

[50] B. Whitworth, A. Ahmad, M. Soegaard, and R. Dam, "Encyclopedia of human computer interaction," *von C. Ghaoui. Hershey: Idea Group Reference. Kap. Socio-technical systems*, pp. 533–541, 2006.

[51] M. L. McHugh, "Interrater reliability: the kappa statistic," *Biochemia medica: Biochemia medica*, vol. 22, no. 3, pp. 276–282, 2012.

[52] Q. Huang, E. Shihab, X. Xia, D. Lo, and S. Li, "Identifying self-admitted technical debt in open source projects using text mining," *Empirical Software Engineering*, vol. 23, no. 1, pp. 418–451, 2018.

[53] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya *et al.*, "Managing technical debt in software-reliant systems," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*, 2010, pp. 47–52.

[54] C. Izurieta, A. Vetrò, N. Zazworka, Y. Cai, C. Seaman, and F. Shull, "Organizing the technical debt landscape," in *2012 Third International Workshop on Managing Technical Debt (MTD)*. IEEE, 2012, pp. 23–26.

[55] J. Bohnet and J. Döllner, "Monitoring code quality and development activity by software maps," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, 2011, pp. 9–16.

[56] W. Snipes, B. Robinson, Y. Guo, and C. Seaman, "Defining the decision factors for managing defects: a technical debt perspective," in *2012 Third International Workshop on Managing Technical Debt (MTD)*. IEEE, 2012, pp. 54–60.

[57] C. Seaman and Y. Guo, "Measuring and monitoring technical debt," in *Advances in Computers*. Elsevier, 2011, vol. 82, pp. 25–46.

[58] Y. Guo and C. Seaman, "A portfolio approach to technical debt management," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, ser. MTD '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 31–34. [Online]. Available: https://doi.org/10.1145/1985362.1985370

[59] N. Rios, L. Mendes, C. Cerdeiral, A. P. F. Magalhães, B. Perez, D. Correal, H. Astudillo, C. Seaman, C. Izurieta, G. Santos, and R. Oliveira Spínola, "Hearing the voice of software practitioners on causes, effects, and practices to deal with documentation debt," in *Requirements Engineering: Foundation for Software Quality*, N. Madhavji, L. Pasquale, A. Ferrari, and S. Gnesi, Eds. Cham: Springer International Publishing, 2020, pp. 55–70.

[60] L. C. d. F. Lage, M. Kalinowski, D. Trevisan, and R. Spinola, "Usability technical debt in software projects: A multi-case study," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2019, pp. 1–6.

[61] F. Buschmann, "To pay or not to pay technical debt," *IEEE Software*, vol. 28, no. 6, pp. 29–31, 2011.

[62] N. Zazworka, M. A. Shaw, F. Shull, and C. Seaman, "Investigating the impact of design debt on software quality," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, ser. MTD '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 17–23. [Online]. Available: https://doi.org/10.1145/1985362.1985366

[63] Z. Codabux and B. Williams, "Managing technical debt: An industrial case study," in *2013 4th International Workshop on Managing Technical Debt (MTD)*, 2013, pp. 8–15.

[64] J. H. McDonald, *Handbook of biological statistics*. sparky house publishing Baltimore, MD, 2009, vol. 2.

[65] P. Kroonenberg and A. Verbeek, "The tale of cochran's rule: My contingency table has so many expected values smaller than 5, what am i to do?" *The American Statistician*, vol. 72, no. 2, pp. 175–183, 2018.

[66] H. Wickham and G. Grolemund, *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*, 1st ed. O'Reilly Media, Inc., 2017.