Self-Admitted Technical Debt in R Packages: An Exploratory Study

***Abstract*—Self-Admitted Technical Debt (SATD) is a particular case of Technical Debt (TD) where developers explicitly acknowledge their sub-optimal implementation decisions. Though previous studies have demonstrated that SATD is common in software projects and negatively impacts their maintenance, they have mostly approached software systems coded in traditional object-oriented programming (OOP), such as Java, C++ or .NET. This paper studies SATD in R packages, and reports results of a three-part study. The first part mined more than 500 R packages available on GitHub, and manually analysed more than 164k of comments to generate a dataset. The second part administered a crowd-sourcing to analyse the quality of the extracted comments, while the third part conducted a survey to address developers' perspectives regarding SATD comments. The main findings indicate that a large amount of outdated code is left commented, with SATD accounting for about 3% of comments. Code Debt was the most common type, but there were also traces of Algorithm Debt, and there is a considerable amount of comments dedicated to circumventing CRAN checks. Moreover, package authors seldom address the SATD they encounter and often add it as self-reminders.**

***Index Terms*—Self-Admitted Technical Debt, Mining Software Repositories, Empirical Software Engineering, R Programming.**

I. Introduction

Delivering high-quality, defect-free software is the goal of all software projects; this is even more relevant in scientific software, where it is often used to obtain research results in a myriad of disciplines. Nonetheless, in both cases, developers are often rushed into completing tasks for various reasons, such as cost reduction, short deadlines, or even lack of knowledge [1].

Technical Debt (TD) is a metaphor that reflects the implied cost of additional rework caused by choosing an easy solution now instead of using a better approach that would take longer [2]. The notion of *Self-Admitted Technical Debt* (SATD) refers to the situation where the developers are aware that the current implementation is not optimal, and write comments alerting of the problems of the solution [3]. Though several studies have been conducted along these lines, they have mostly used the same domain: object-oriented programming (OOP) repositories [4].

Previous systematic mappings demonstrated a lack of software engineering research in R programming, both from an academic and practitioner's perspective alike [5]. R is a package-based programming ecosystem for statistical analysis, visualisations and datasets, with a unique mixture of paradigms: it is dynamically typed, vectorised, both lazy and side-effecting, fostering functional and interactive programming but also providing core object-oriented (OO) features [6]. Moreover, though it has an ever-growing community, most package contributors are not software engineers by trade [7], and only a few of them are mindful of the inner concepts of the language [8].

As a result, this paper conducted a three-part mixed-methods study to understand SATD in a new domain: R package programming. The first part mined 503 R packages from GitHub, and manually analysed more than 164k of comments to generate a dataset. The second part directed a crowd-sourcing to analyse the quality of the extracted comments, while the third part conducted a survey to address developers' perspectives regarding SATD comments. By analysing the collected data, this paper aims to answer the following research questions:

- **RQ1: How much SATD exists in R packages?** Focused on the type of content addressed on inline comments and how many of them represent SATD, depending on their location (package or test code). The resulting distribution was compared to that of traditional OOP projects. Following a systematic process, R packages were mined from GitHub, and internal comments were extracted and manually classified.
- **RQ2: What is the quality of comments in R packages?** In R, both code and comments may be different from those present in other studies. This RQ was answered through a crowd-sourcing approach implemented through a developers' survey (Q-Survey); a sample of comments was classified in terms of usefulness (familiarity and necessity) and clarity (precision and simplicity).
- **RQ3: What types of TD are most commonly admitted?** The comments identified as SATD from RQ1 were manually classified again, to determine which type of debt they represent. Selected SATD comments were compared to [3] dataset to uncover differences, and to define new patterns for R.
- **RQ4: Are developers aware of SATD?** Aimed to discover possible differences in the commenting practices of R programmers, to determine differences with OOP-centred programmers. The authors and maintainers of the mined packages were invited to participate in an anonymous survey (A-Survey); this survey's structure replicated one used by a previous work [9].

More than 164k of comments were manually read and classified into 12 types of TD. The A-Survey obtained about 102 responses, while Q-Survey surpassed the expected threshold to obtain 140 complete responses. This manuscript

978-1-7281-8710-5/21/$31.00 ©2021 IEEE
DOI 10.1109/MSR52588.2021.00030

makes the following contributions:

- An analysis of TD in R packages. Results showcase, an excessive amount of outdated code left commented out, poor testing quality with comments written to avoid automation of tests, and a tendency to bypass CRAN checks. Moreover, Code and Test are the types of TD most commonly admitted, though Algorithm Debt was also detected. Comment quality was often deemed low and obscure, and developers indicated they seldom address it in their packages.
- A comparison with results from other studies, also exploring source code comments, but in OOP projects. This includes SATD text patters, developers' perspectives, and types of datasets.
- A dataset of SATD comments in a large variety of R packages, to use in future studies, as well as the final list of text patterns discovered in this study.

Structure of the paper. Section II describes related works, and how this study differentiates itself from others. Section III presents the methodology, detailing the selection of repositories to mine, the dataset generation, the construction of surveys, the sample selection and the data analysis planning. After that, Section IV answers each proposed question, highlighting key findings. Section V provides the main implications of the findings, while Section VI discusses threats to validity. Finally, Section VII concludes this work.

II. Related Works

SATD and MSR studies have been commonly carried in empirical software engineering. A baseline study manually classified source code comments to obtain 62 SATD patterns [3], while another one explored projects to understand what types of TD were most commonly mentioned, and what heuristics can be followed to discover it [2]. These were later expanded by a large-scale automated replication across 159 studies [10]. Using the manually generated dataset, [1] used natural language processing to automatically mine and detect SATD occurrences in ten open source projects. A study created an automated prediction system to estimate SATD in comments in OOP software [11]. Finally, another mining study investigated the removal of SATD through a two-part study that combined mining repositories and developer surveys [9].

However, most studies in the area of SATD are often conducted in projects using traditional OOP. This limitation was highlighted by a previous systematic literature review that identified the need for more sources [4]. An example along these lines still worked with OOP software but mined SATD from GitHub issues rather than inline comments [12].

Exploring R contributes to explore SATD in a domain that, to the author's best knowledge, has not yet been explored, but also expands research regarding software engineering in this unique programming paradigm.

However, studies related to R programming from a software engineering perspective are scarce. A mining study explored how the use of GitHub influences the R ecosystem, regarding the distribution of R packages and for inter-repository package dependencies [13]. Another mining project studied the maintenance performed in R's statistical packages to explore change frequency, and variability between versions [14]. At the same time, in terms of programming theory, another one assessed the success of different R features to evaluate the fundamental choices behind the language design [15].

III. Methodology

This empirical study was divided into three parts. First, mining of R package repositories in GitHub with manual analysis, second, an open anonymous survey to peruse the quality of comments, and lastly another anonymous survey directed to package authors and maintainers. The next subsections describe the various steps followed to build the dataset.

A. Dataset Generation

The repositories to be mined were selected following the systematic process outlined by Kalliamvakou et al. [16].

First, inclusion and exclusion criteria were defined to determine which packages should be considered. Overall, **included R packages** had to be public, open-source repositories written in R, with a basic R package structure (as defined by [17]), and including unit testing. On the contrary, **excluded** packages were forks from other packages, written in a natural language other than English, created before 2010 or with no commits after 2018. Personal, deprecated, archived or non-maintained packages were also disregarded, as well as those that were books, data packages or collections of other packages unified into a single one.

Six control packages were selected by experts' recommendation. These were: `httr`, `pkgdown`, `ggally`, `roxygen2`, `igraph`, and `raster`. These packages were used to define the search string employed to collect the packages. This string followed GitHub's search syntax[1]. The process was iterative, and control papers had to be found up to the fifth search page. The string was approved after completing the search four times during the same week, and obtaining the control packages before the fifth page.

The search was completed using GitHub's Advanced Search[2], with the string *package NOT personal created:>2010-01-01 pushed:>2018-01-01 language:R*. It also excluded forks, any type of license, and were sorted by "best match". The search produced 630 repositories that were manually checked to ensure the inclusion/exclusion criteria. **503 unique repositories of R packages were used in this study**.

Results of the search were stored as a list of "slugs": a combination of the type `accountName/repositoryName`. An R script was used to download the repositories using the GitHub API automatically. After that, the same script inspected all repositories, extracting all the internal comment lines (those starting with `#`); documentation comments (i.e.

[1]See: https://docs.github.com/en/github/searching-for-information-on-github/understanding-the-search-syntax

[2]See: https://github.com/search/advanced

Roxygen's) were excluded from this study. This produced a first dataset that included: package name, location (R code, or test code), file name, start and end line of the comment, comment body, and if it was a full line or not. The last column was a true/false value used to indicate if that line of code contained only the comment, or if it started with code and the comment was on the last part.

R does not have a comment syntax to allow a multi-line comment (i.e. an equivalent to Java's $\backslash * \ldots * \backslash$, so developers often write several related single-line comments one after another. Therefore, the first dataset was automatically parsed to detect this situation (only on full-line comments) and to merge the start/end line, as well as the comment body. This second dataset had the same structure as the first one. **Overall, 164349 lines of comments were mined.**

B. Package Authors' Survey (A-Survey)

R packages show information about their authors and maintainers in the "description" file, often disclosing their email addresses. The selected repositories were automatically explored with an R script, to extract this information and consolidate the records; i.e., each email was associated with a list of package names in which it was found. **845 email addresses were mined through this process**.

The survey's questionnaire was constructed as an adjusted replica of [9]; this decision was deliberate in order to be capable of comparing results. Given R's diverse disciplinary background[7], the wording of the questionnaire was simplified, and two questions were added: one regarding what TD is, and a "free space" comment. It also included: 1) demographics regarding role and development tasks of the project, 2) Likert-scale questions about frequency of finding or addressing SATD comments, and 3) open-ended questions asking how they act upon those comments.

The survey was implemented in Qualtrics, and its automated distribution by email was used to send the questionnaire to the selected participants. From the 845 emails sent, 64 bounced back, and 102 developers completed the questionnaire. **The analysis was conducted only using these 102 submissions.**

The full questionnaire and the unparsed responses are available in the accompanying dataset[3].

C. Comments' Quality Survey (Q-Survey)

To analyse the quality and usefulness of the comments, each sampled comment had to be quantified in a 5-points Likert-scale in two categories. These represented qualities intrinsic to "technical writing"[4], namely:

- **Usefulness**. Given by the *familiarity* (use of straightforward language, such as plain English) and *necessity* (if it provides what is required for comprehension and nothing more) of the comment.
- **Clarity**. Related to effectively communicating with its intended audience, and evaluated through *precision* (specificity and exactness) and *simplicity*.

A crowd-sourcing approach, implemented through an anonymous online survey, was used to obtain this classification. In here, participants were not paid to classify the comments manually but were filtered after a short demographic section regarding if they had any type of programming experience. This allowed classifying their responses according to their knowledge in R, or other languages. After that, the questionnaire showcased two blocks, each of them with comments to be classified.

The generation of this survey posed a statistical sampling challenge, as several variables were taken into account.

- *How many comments to include per survey*. Aiming to create a survey that required less than 10 minutes to complete, a dummy questionnaire (with the correct structure) was loaded into Qualtrics. The automated time estimation determined that **36 comments (plus demographics) yielded an 8.5 minutes-long survey**. This was further tested with two test participants. Comments were divided into two groups: SATD and "valuable non-SATD" (see Section IV-A). They would be equally distributed, with **16 comments for each group**.
- *Which comments to include for classification.* A conservative sample size calculation was used to determine the size of the sample for each group. With confidence of 95%, and a margin of error of 0.05, the sample size of each subgroup (SATD and non-SATD) **had a size of 358 unique comments each**. A random subset was sampled from each group and loaded in the survey.
- *Which comments to show.* Qualtrics offers advanced randomisation with even distribution[5]. This allowed fixing an "explanation" statement above each group, and then randomly showing 16 comments of each group to each participant. This ensured that comments would be evenly included, obtaining a similar number of responses. All questions were mandatory responses.
- *Minimal number of responses to obtain*. In order to account for respondent's bias, each comment had to be classified multiple times. Aiming for a minimum of **10 classifications per comment**, the survey had to **obtain at least 204 responses**. This was calculated as: $sampleSize * 2[groups] * minResponses/surveySize$.

Once loaded, the survey was distributed online in social networks, using an anonymous link, and hashtags commonly used by the R community. This approach ensured that the comment analysis was done mostly by members of the R community, with some knowledge of programming, while obtaining repeated classifications that minimised the respondents' bias. The survey's questionnaire and survey responses are available as part of the dataset[3].

[3]See: https://tinyurl.com/yyfwvs94

[4]See: https://www.hurleywrite.com/Blog/294209/The-4-Pillars-of-Clarity-in-Technical-Writing.

[5]See: https://www.qualtrics.com/support/survey-platform/survey-module/block-options/question-randomization/

IV. Study Results

This Section introduces the answers and key findings of each of the research questions.

A. RQ1: SATD Frequency

In total, 164363 lines of code were manually inspected. Out of them, 87 were found to be (totally, or partially), written in a language different than English (specifically, Portuguese, German and Spanish). Though the selected packages were entirely written in English, these lines were "internal exceptions", a common practice among non-native English speaking developers [18]. Those 87 lines were disregarded, and the rest of the analysis was conducted over the remaining 164262 English comments.

In terms of comments "demographics", three proportions were calculated. First, about 89.7% (exactly 147407) comments were *full-line comments*, occupying the whole line of the code, and not just a part of it; the remaining 16855 comments (10.3%) were partial lines. Second, out of the total, 77.% (127283 comments) were *single-line* -i.e. starting and ending on the same line-, while the remaining 22.5% (or 36979 comments) were *multi-line*. Third and last, 87.8% (exactly 144173 comments) appeared in the R code, while only 12.2% (20089 comments) were found in the tests suits; all of the selected packages had test suits.

An initial set of possible groups was developed in consultation with two expert R developers, based on their personal experiences; this was used for the manual classification. Table I summarises the classification statistics, by number of comments and percentage of the total.

TABLE I
Filtering of comments, divided by multi and single line.

	Multilines		Single-lines	
Type	# Comments	%	# Comments	%
Explanation	17170	10.45	91820	55.89
Code	7846	4.77	9832	5.98
Title	5222	3.17	15432	9.39
SATD	2043	1.24	2919	1.77
Credits	2563	1.56	978	0.59
Exceptions	582	0.35	2252	1.37
Block Closures	237	0.14	1183	0.72
Return Notes	365	0.22	1593	0.96
Example Data	315	0.19	484	0.29
Fixes	636	0.38	790	0.48

Overall, there are several "minor" classifications. **Return Notes** indicate comments that explain when a function is returning a value; while sometimes they are meaningful, most of the time, they are not (e.g. `#Return results`). **Block Closures** are often used to highlight the end of a block when the function has too many nested blocks (e.g. `#end while`). **Credits** were comments used to give credit to a particular developer, to link to a StackOverflow post, or cite a research paper that explained the implementation of a method. **Example Data** indicated comments that explicitly indicated data, such as a usage example or a dataset location, without locating it in the Roxygen documentation.

Titles marked comments that mostly had strings of characters (such as dashes, bars, hashes), used to separate blocks of code; though without meaningful content, it accounted for 12.56% of the comments. **Fixes** often pointed to GitHub issues or errors that had been long corrected, such as `#Breaking works after escapes (#265)`; they had to state that everything was fixed and working. Finally, **Exceptions** were also minor comments that simply indicated that an error was being thrown, or a warning was being written in the console -e.g. `#Warnings for illegal groupings`.

The remaining three categories were considered extremely relevant. These are:

- **Code:** These comment out fragments of code, and accounted for almost 11% of the comments. Leaving unused or outdated code in a software project is considered a bad practice, often deemed *Code Debt* [19], specifically code smells related to outdated code. These have already been identified as a type of debt in a previous study [20]. Examples span from a single line to full functions being left out.
- **SATD:** Specific lines of comments that indicate a self-admission of TD by stating a known bug, lack of knowledge, displeasure of the solution, or even outlining future actions to be taken. These accounted for 4962 lines, representing slightly more than 3% of all lines.
- **Explanations:** Grouped general comments used to explain what the code was doing, without being part of any of the other categories. This is, as expected, the most common type, accounting for more than 66% of the comments, or 108990 comments.

These results were compared to other studies. For example, Potdar and Shihab [3] obtained a 7.42% of SATD comments, while Bavota and Russo [10] identified a varying range between 0.2% to 2.6% of SATD comments. A range between 3.77% and 20.13% per project was also found through an automated classification by Flisar and Podgorelec [11]. Moreover, the overall rate found in this study is also aligned with what was reported by Maldonado and Shihab [2], Potdar and Shihab [3]. This is quite relevant, as the programming language or its paradigm does not appear to affect SATD frequency of occurrence.

> **Findings #1.1.** About 11% of the comments represented code commented out, indicating *Code Debt*.
> **Findings #1.2.** SATD comments were about 3% of the total, inline to proportions found in other studies [2, 3]. R's paradigm or type of programming do not appear to influence this frequency.

B. RQ2: Comments Quality

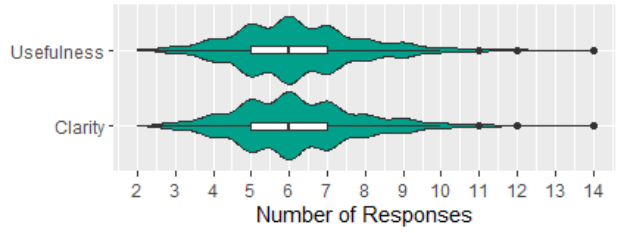
An anonymous online survey was used as a crowd-sourcing approach to evaluate the quality of a sample of comments. The structure and distribution of this survey were discussed

in Section III-C. Overall, out of a threshold of 204 responses required, we obtained 140 complete submissions.

In terms of demographics, participants appear to be experienced programmers, with 29.4% reporting more than 10 years of experience, and 34.3% having between 5-10 years; the remainder had between 2-5 years (close to 29%), and only 7.2% had less than two years of experience. The *programming language experience* was rated using a 5-point Likert-scale; the most popular choices with their Likert mean are R (Likert 3.82), Python (Likert 1.85), C/C++ (Likert 1.4) and others (Likert 2.26). Finally, regarding *background disciplines*, statistics account for about 29.3% of participants, with social sciences (about 14.6%) and computer science (11.2%) close behind, followed by others (15.5%); moreover, most people selected at least two disciplines.

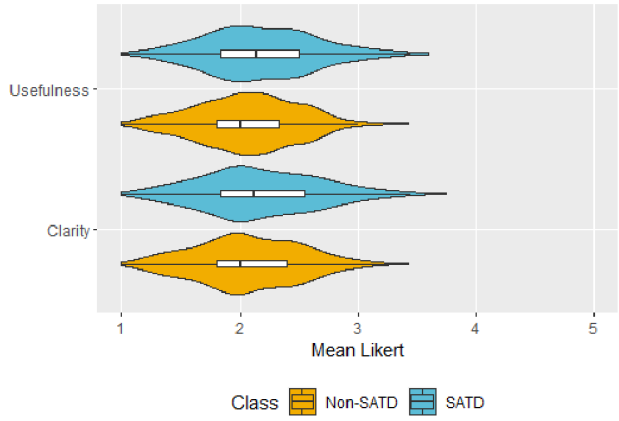
The survey was left open for over a month, during October/November 2020. Figure 1 uses a violin plot to showcase how many classifications received each comment on average; they are divided by usefulness and clarity only. As can be seen, the majority of sampled comments received about six responses, which was below number considered during the planning of the sample size (see Section III-C). This may increase the trustworthiness of the results, compensating for a smaller sample.

Fig. 1. Frequency of assessment for all sampled comments, in terms of Usefulness and Clarity.



Overall, Figure 2 summarises the results of the classification, by the class of comment (SATD or non-SATD), and by type of evaluation (Usefulness or Clarity). As can be seen, comments tend to have low quality, as both the usefulness and the clarity was averaged to Likert 2: "Poor". In general, the mean for each type and category oscillates between 2.06 and 2.18, which is close enough to be negligible.

Fig. 2. Distribution of mean Likert values for all sampled comments, in terms of Usefulness and Clarity, for each block of comments (SATD and non-SATD.



Several points should be discussed. Though the mean number of obtained assessments per comment is below the expected, the respondents declare having many years of experience, with 63.8% of them being *senior developers* with more than five years of experience. They also self-report an above-average level of R programming, with knowledge on other languages such as Python. This compensates the lack of responses, as senior developers are prone to have a better understanding of the language, having encountered this type of comments more often than a junior developer.

From this, it can be concluded that comments inside R packages have a low quality: they are not easily understandable, and their clarity is also low. However, the questionnaire was not planned to understand *why* this was happening, as it focused on obtaining the first classification. Therefore, the following work should tackle research questions such as "why do developers consider that the comments have such low quality?", and "what can be done to increase the usefulness and clarity of said comments?".

Findings #2. Both SATD and non-SATD comments have low quality, with "poor" (Likert 2) usefulness and clarity, as per the assessment conducted by mostly senior R developers.

C. RQ3: Admitted TD

From the 164k lines of code, about 4962 were identified as SATD comments. Since the comments are written in natural language, and they may contain technical terms specific to data science or statistics (which may affect an automated search using existing datasets), the comments were analysed by manually reading each of them. As a starting point, the classification was done using Alves et al. [19] ontology, as in previous works [10, 11].

1) Types of Debt: The categorisation was only done at a high level, without addressing specific smells. The types of debt discovered, their frequency in lines of code, and the percentage of lines is reported in Table II. In the following, the most relevant type is described by reporting and commenting representative SATD instances that were found; this is done due to space limitations. The full classification is available in the dataset[3].

Code Debt. *"Problems found in the source code which can affect the legibility of the code negatively, making it difficult to be maintained"* [19]. No further classification was made in this particular category. This decision was made due to R's specific paradigm, and the strong link between some smells to OOP (e.g. the "god class" or "feature envy"). This debt

TABLE II
TYPES OF TD FOUND IN SATD COMMENTS.

TD Type	# Lines	% Total
Code	2015	40.6%
Test	784	15.8%
Defect	693	13.97%
Requirements	355	7.15%
Architecture	291	5.86%
Algorithm	276	5.56%
Design	221	4.45%
Build	160	3.22%
Usability	71	1.4%
Documentation	53	1.07%
People	21	0.42%
Versioning	21	0.42%

represents 40.6% of total comments, though about 1.81% was present in the test suit, with the remainder in the code itself. In terms of R code itself, some relevant examples are:

- `#Such an ugly hack to format misc typed column`. The comment referred to a dataframe that often had columns incorrectly named, causing an issue with functions that depended on column names.
- `# hacky barebones recreation of system2`. The developer was apparently having issues with a specific system function[6] and decided to *duplicate the code* to be capable of altering the function behaviour, rather than to continue addressing the importing problem.

Test Debt. *"Issues which can affect the quality of testing activities"* [19]. It is possible to assume this debt is frequently addressed due to the importance given to testing in many communities that peer-review R packages, such as rOpenSci [21] or BioConductor [22]. CRAN submissions do not require tests, but they are automatically checked nonetheless. Some relevant examples are:

- Those indicating that a test was needed. Such as `#TODO: test for the existence of objective method`.
- Those indicating the inadequacy of the test suit, likely `# this is only tested for walktrap, should work for other methods`.
- Comments referring to problems caused by the tests when building the package. Examples are `# why doesn't vcov(x) work here???`.

Defect Debt. *"Known defects [...] that should be fixed but due to competing priorities and limited resources have to be deferred to a later time"* [19]. This definition was limited only to defects related to the R package "as a piece of software", without including those that discussed the algorithms' implementation. Almost 14% of the comments discussed defects, which is aligned with the authors survey responses: they reported that in most cases, SATD comments are used as self-reminders of incomplete or buggy features that should be addressed later (see Section IV-D). Some interesting examples are:

[6]See: https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/system2

- Known workarounds to ensure the system keeps working while the bug is unfixed, such as `# HACK: Circumvents a bug in flowClust.`.
- Placeholders or reminders to fix a specific behaviour after a given milestone. Examples are: `# FIXME handle this better in multicoloc.data`.
- Most notoriously, it was often used to highlight places in which the code was not behaving as it should be: `# FIXME still not quite right`.

Algorithm Debt. This type of debt is not often found on "traditional" software, but more in "research software". In particular, a previous work stated that *"Algorithm debt corresponds to sub-optimal implementations of algorithm logic in deep learning frameworks. Algorithm debt can pull down the performance of a system"* [23]. This definition was extrapolated to refer to "sub-optimal implementations of **specific** algorithm logic in a **data science or statistical** framework", as it was commonly found in R packages. Though it can be considered a sub-type of Defect Debt, it was purposefully kept separated due to using similar words but for a different purpose. Noteworthy examples are:

- Indicating possible defects, or logic not working as intended. Such as `#It seems this over-estimates the truth`, and also `# FIXME assumes correlation prior under srs is dbeta`.
- Acknowledging possible missing features regarding the algorithm, such as `# should we estimate the relative bias?`.
- Comments discussing pre-fixed values, their fitness or worthiness, such as `# some initial values, probably not optimal...` and also `# how high do we need to set this? 1/5/10/100?`.

Overall, the findings of this study correlate with others by determining Code Debt as the most common type [10]. Though all types of debt identified in other SATD studies were present (namely, Requirements, Design, Defect, Documentation and Test) [1, 2, 3, 10], the occurrences are different in R packages. Algorithm Debt was also discovered, in alignment with studies conducted in other research-centred software [23].

> **Findings #3.1**. Code debt accounts for 40% of SATD comments. Other common types are Test, Defect and Requirements. *Algorithm Debt* was also discovered, accounting for about 5.56% of comments, in alignment to other research-centred types of software.

2) SATD Patterns: The current approach for SATD analysis uses 62 comment patterns determined by Potdar and Shihab [3] obtained through a manual analysis on four OOP systems (i.e. Eclipse, Chromium OS, ArgoUML and Apache). Therefore, this part of the analysis implied four steps:

1) Finding these patterns [3] in the existing comments.
2) Distilling comments (e.g. removing symbols and stop words), and conducting an n-gram analysis to determine new patterns.
3) Re-assemble the "cleaned" patterns to include stop-words again, and generate the list.
4) Manual inspection of remaining comments, in search for less reliable, ad-hoc patterns.

First, in terms of Potdar and Shihab [3], there were 62 patterns, but only 17 were found among the R SATD comments; none of them were present in the non-SATD comments. Almost half of them had a single occurrence (nine patterns), and only four were frequent: `fixme` (934 comments), `hack` (210 comments), `ugly` (24 comments) and `stupid` (with 19 occurrences).

Second, an n-gram analysis was conducted. Comments that already had existing patterns were not considered. The remainder was cleaned from symbols, numbers and stop-words, and then grouped in n-grams of one to four words. Every set was subset three times, keeping only the fourth quantile (i.e. the one with higher frequencies). After that, starting from the single-word gram, every dataset was manually analysed to determine newer patterns. In particular, ten new (clean) patterns were defined.

Third, these patterns were search the "full comments" (the as-is version, not the "cleaned up" of the second step), to add additional stop words. For example, the new pattern `lack memory` was converted to `lack of memory`. Moreover, some patterns were detected to be written with and without spaces in between, probably due to incorrect grammar (e.g. `workaround` and `work around`). These patterns were added to the list in all common variants.

The selected patterns are presented in the list below, also available as a dataset[3]. From the total of SATD comments, these patterns categorised about 59% of the samples. Due to this being a first manual study in the R programming domain, this classification was deemed acceptable, though further studies and automation are considered for future works.

- Patterns from Potdar and Shihab [3] found in SATD comments: `barf`, `causes issue`, `crap`, `don't use this`, `fixme`, `hack` (or `hacky`), `inconsistency`, `is problematic`, `kludge`, `silly`, `stupid`, `take care`, `temporary solution`, `there is a problem`, `trial and error`, `ugly`.
- Proposed and newly detected patterns (step 3): `todo` (or `to do`, or `to-do`), `nocov`, `workaround` (or `work around`, or `work-around`), `hard coded` (or `hard-coded`, or `hardcoded`), `old code`, `trick r`, `lack of memory`, `r cmd check`.

On the fourth and last step, remaining SATD comments were manually inspected to determine additional patterns. Though these patterns only classified about 140 lines of comments, they complement the above set. It is worth noticing that these require further validation. They are: `can't do this`, `this should work`, `should work`, `trick` (or `tricky`), `quick fix`, `it doesn't work`, and `not working`.

In particular, it was detected that there is a considerable presence of comments related to skipping automated unit testing for a variety of reasons. This finding is aligned with a result of a previous work that determined that R packages have an excessive focus on coverage, but not so much test quality [24]. Moreover, there is a strong emphasis on circumventing CRAN checks, leading to doubt the usefulness of such automated revision. Nonetheless, for now, the latter remains as future work.

Findings #3.2. Only 17 out of the 62 patterns from Potdar and Shihab [3] were found in these comments, with most corresponding to single-word patterns. From there, 11 new patterns have been added, also accounting for grammatical variations.
Findings #3.3. There is a large number of SATD comments directed at stopping the execution of automated tests, in alignment with previous studies that highlighted low testing quality in R packages [24].
Findings #3.4. Many SATD comments indicated they were bypassing CRAN checks, leading to doubt the usefulness of such automated revision.

D. RQ4: Developer's Perspective

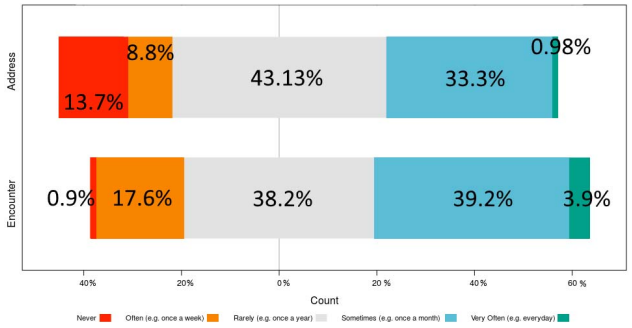
An anonymous online survey (A-Survey) was used to discover differences in the commenting practices of developers, and to compare the findings of this work to the perceptions of developers in real life. The structure and distribution of this survey were discussed in Section III-B. Overall, 845 authors and maintainers were identified, and 102 responses were received, which represents a response rate of 12%, acceptable in questionnaire-based software engineering surveys [25].

Two questions were concerned with demographics: In terms of *programming experience in R*, the respondents are deemed experts, as 47.06% declared more than 10 years of experience, followed up by 39.22% having between 5-10 years of experience; finally, about 12.75% had 2-5 years, with less than 1% having less than two years. These results indicate that the sample of practitioners was highly experienced, increasing the reliability of the obtained results [26]. The second question inquired about the *frequency of working with other people's code*. About 16.7% indicated "everyday", and 27.45% said "once a week"; after that, 38.24% reported to do it "once a month", and 17.65% said "once a year". These results are also quite positive, as they highlight that participants are mostly highly collaborative, and therefore more prone to having extensive experience.

In terms of questions regarding encountering and addressing TD, Figure 3 summarises results. As can be seen, about 43% of participants find SATD at least monthly, but less than 34% decided to address that SATD with the same

frequency. However, there is a large a considerable percentage of participants (about 22.5%) that seldom do something about the SATD they encounter. This challenges the quality of the code of the studied R packages.

Fig. 3. Survey responses on how often developers encounter or address SATD.



Given the diversity of technical backgrounds in the R community, a question asked "Do you know what technical debt is?". In response, about 55.9% replied "Yes, and I try to minimise it in my code", while only 8.9% stated "Yes, but I don't worry about it". From the remaining 35.3%, half of them did not know what it was, and the other half confessed searching the term in Google before replying. These replies give perspective when addressing the free-text answers.

Adding SATD was not enquired as a Likert scale, but as an open-ended question only; this was to minimise the negative impact of this question to R developers. Responses were manually analysed and classified. Overall, 13 groups were identified, but the five most popular are: *as self reminders* (about 28.4% of comments), for *future planning* (close to 16.7%), to obtain a *quick solution* first (slightly above 12.7%) or due to *lack of knowledge* or as a *warning* (both with 7.8% each). Regardless of the main reason, about 16.7% also mentioned *lack of time* as a key reason to admit TD. Some of the most striking comments are the following:

- "Other features prioritised higher; research software just needs to be 'good enough' not perfect; research software typically grows organically, and it is not really planned, so new ideas come up during development a lot."
- "The current solution is usually 'good enough', even though if it's not done the 'right' way. (And, the 'you ain't gonna need it' principle might suggest that the 'quick and dirty' solution will suffice anyhow)".
- "The cost of remembering is too high. Also, code my look logically sound, but a TODO about an unfixed corner case is a lifesaver".
- "They do not have an elegant solution at the time of a quick and easy fix but intend to return to the problem later".

When enquired about the reasons for addressing SATD comments, the discovered topics were different. Out of 13 topics, there were five more relevant. Above 17.6% stated that they addressed a SATD comment *when the feature became needed*, while 15.7% said that it was done *to improve code quality*. After that, *errors becoming troublesome* and *having to complete work* covered about 10.8% each. Nonetheless, 12.7% of respondents confessed not knowing what SATD comments were completed. Some of the most relevant comments are:

- "Because they need for their own purposes to make the code work, or to add this one feature that was planned but not implemented".
- "They TODO comments are addressed when there is a pressing need to add or finalise the unfinished feature."
- "When they reach a point, using this code in their project(s), that they can 1) no longer rely on a workaround or 2) the issue is directly blocking their progress on their project".
- "Obtained a working version and moved on to other tasks."

Findings in this question correspond to those of Maldonado et al. [9]: respondents do not seem to indicate a systematic process for addressing SATD, delaying this activity until the last moment; they also appear to do it in an ad-hoc manner. Overall, it was notorious that developers were prone to address their own SATD, but not that of others unless it became necessary.

> **Findings #4**. Participants were highly collaborative and experienced. Though they often encounter TD, they rarely address it. If they do, it is delayed until the last possible moment, and completed in an ad-hoc manner. Most developers admit debt as a self-reminder, or for future planning of activities. Results align with studies in other programming paradigms.

V. Implications

This study has several implications for SE research and R programming alike. To the authors' knowledge, this is the first paper that analyses SATD in source code comments of R packages. Therefore, the findings help quantify and highlight challenges and opportunities for research in the intersection of these two areas.

Analysis of SATD in comments of R packages. This study showed that though there are certain similarities between OOP-centred classifications of TD and SATD, there are also several differences that should be further enquired in future works. First, there is a large amount (about 11%) of comments that "comment out" code that has not been removed and is no longer being used, which represents smells related to *outdated code* for *Code Debt*. Further studies should focus on understanding how long these fragments live out, why they are commented, who introduces them, and who removes them.

Second, though most common types of debt were found in, an in-depth study in terms of specific smells could not be conducted due to R's mixture of paradigms; this is another line for future works. Along this line, *Algorithm Debt* was also present in SATD comments. Its definition used

was adapted from its original proposal related to machine learning frameworks [23]; there is little-to-no information on specific smells for it, opening another door for future research.

Insight into developer's attitudes and perceptions. Regardless of their increased growth and popularity as a language [27], the R community has not been thoroughly studied. This study produced one of the first insights into their reasoning and perception, though limited to SATD in source comments. In correspondence to the findings of other studies [12], developers still prefer to implement sub-optimal solutions to deliver a minimal working product faster. Most of them consider the addition of SATD as a "personal reminder", to highlight something they did not finish, or a new feature. Finally, this translated into the admission that R developers rarely address the TD they encounter in comments, and only do so due to one of two reasons: either "they have time" to do it, or it is causing a fault that halts the development.

"Transitive" quality of R packages. Overall, what has been stated in the above implications can potentially have a more concerning effect: the *transitivity of quality and Technical Debt*. Because R is a package-based environment, it is reasonable to hypothesise that the TD of one package (and hence, its lower code quality) can have a negative impact in the packages that import it (i.e. depend on it). In that case, the quality of that first package could potentially become a threat to the validity of the studies whose results were calculated using it.

In addition to the former discussion, there are also several indications of low quality and concerning TD were uncovered in this study, that may have enabled such "transitive TD":

1) The existence of *Algorithm Debt*, which may affect the theoretical/algorithmic results given by a package.
2) A tendency to prevent the automated execution of specific tests, due to the awareness of the unsuitability of the test suit (in alignment to previous findings [24]).
3) A tendency to circumvent and bypass CRAN checks.
4) The perception that existing comments have low quality in terms of Usefulness and Clarity.

Therefore, future studies could also focus on addressing the transitivity of TD in R, as well as the reasons and effects of the four points mentioned above.

VI. Threats to Validity

Some threats may have influenced this empirical study—this section discusses and overviews how they were addressed.

Internal validity. This concerns factors that can influence the results. Given R's unique characteristics, along with comments written in natural language, a manual classification was completed. Though this was done in other studies [3], any manual process is prone to human error and subjectivity; moreover, they were analysed by the first author only. To alleviate this, an authors' survey (A-Survey) was conducted to ensure that the findings also reflected the developers' perspectives. In total, 845 developers were contacted, obtaining 102 responses; this represents a response rate over 12%, which is quite acceptable in questionnaire-based software engineering surveys [25].

Construct validity. These concern the relationship between theory and observation. A threat of using inline code comments is the consistency of changes between comments and code, as well as the use of natural language itself. Therefore, results may be impacted by the quantity and quality of comments in every R package. Besides, since this study parted from the definitions of TD used in OOP, it is possible that data science-specific types of TD were not evaluated. However, the overall focus of this paper was to provide an initial dataset and to highlight differences between SATD in a different programming paradigm.

External validity. Concerns the generalisation of the findings. This study used 503 unique packages, written by a myriad of authors of diverse backgrounds, and analysed more than 164k inline comments; however, these results may not generalise to all R packages, to other programming languages, or sources of SATD (i.e. GitHub issues). In terms of the survey's classification, the sampling calculation described in Section III-C helped alleviate these problems, by considering the confidence and error, as well as ensuring that multiple people re-classified the same comments.

VII. Conclusions

This paper investigated *self-admitted technical debt* (SATD) in source comments of R packages. To do this, it conducted a three-part mixed-methods study to understand SATD in a new domain: R packages. The first part mined 503 R packages from GitHub, and manually analysed more than 164k of comments to generate a dataset. The second part administered a crowd-sourcing to analyse the quality of the extracted comments, while the third part comprised a survey to address developers' perspectives regarding SATD comments.

Overall, it was detected that about 11% of code comments represent outdated pieces of code that are no longer used, while additional 3% are SATD comments. Regardless of the type of comment (SATD or non-SATD), other R developers (mostly seniors with more than five years of programming experience) deem comment quality as low, with poor usefulness and clarity. The most common types of admitted TD are Code, Test and Defect, but this study also uncovered a considerable number of occurrences of Algorithm Debt; the least common types are Documentation, People and Versioning. In term of natural language patterns, of the well-known base of 62 patterns, only 17 were found, and 11 more were proposed by completing an n-gram analysis of the text; additional seven patterns were proposed after a final manual inspection. Finally, R developers also report focusing on a minimally-viable product first, and introducing SATD comments as "self-reminders"; due to this, they seldom address the TD they encounter.

Other general findings are the acknowledgement of inadequate tests suits (that result in developers' efforts for preventing these tests from being automatically executed), as

well as their tendency to bypass the automated CRAN checks, often introducing TD to do so.

Several lines of future works can be addressed. First, since developers concluded that comments have low quality, further studies are needed to understand why this categorisation is happening, and what to do to increase the quality of the comments. Second, the usefulness of CRAN checks and the reasons that lead developers to sidestep them should also be further inquired. Third and lastly, given R's mixture of paradigms, it is required to conduct further research into specific smells for each type of debt.

Acknowledgements

Blinded for the review process.

References

[1] E. d. S. Maldonado, E. Shihab, and N. Tsantalis, "Using natural language processing to automatically detect self-admitted technical debt," *IEEE Transactions on Software Engineering*, vol. 43, no. 11, pp. 1044–1062, 2017.

[2] E. Maldonado and E. Shihab, "Detecting and Quantifying Different Types of Self-Admitted Technical Debt," in *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*, 2015, pp. 9–15.

[3] A. Potdar and E. Shihab, "An exploratory study on self-admitted technical debt," in *2014 IEEE International Conference on Software Maintenance and Evolution*, 2014, pp. 91–100.

[4] 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

[5] Anonymised, "Anonymised manuscript - r mapping," *Anonymised Journal*, dec 2020.

[6] 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. London, United Kingdom: Association for Computing Machinery, Jul. 2019, pp. 1–5. [Online]. Available: https://doi.org/10.1145/3340670.3342426

[7] 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.

[8] 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.

[9] E. D. S. Maldonado, R. Abdalkareem, E. Shihab, and A. Serebrenik, "An empirical study on the removal of self-admitted technical debt," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2017, pp. 238–248.

[10] G. Bavota and B. Russo, "A large-scale empirical study on self-admitted technical debt," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 315–326. [Online]. Available: https://doi.org/10.1145/2901739.2901742

[11] J. Flisar and V. Podgorelec, "Identification of self-admitted technical debt using enhanced feature selection based on word embedding," *IEEE Access*, vol. 7, pp. 106 475–106 494, 2019.

[12] L. Xavier, F. Ferreira, R. Brito, and M. T. Valente, "Beyond the code: Mining self-admitted technical debt in issue tracker systems," in *Proceedings of the 17th International Conference on Mining Software Repositories*, ser. MSR '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 137–146. [Online]. Available: https://doi.org/10.1145/3379597.3387459

[13] 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.

[14] C. Ramirez, M. Nagappan, and M. Mirakhorli, "Studying the impact of evolution in r libraries on software engineering research," in *2015 IEEE 1st International Workshop on Software Analytics (SWAN)*, 2015, pp. 29–30.

[15] F. Morandat, B. Hill, L. Osvald, and J. Vitek, "Evaluating the design of the r language," in *ECOOP 2012 – Object-Oriented Programming*, J. Noble, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 104–131.

[16] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 92–101. [Online]. Available: https://doi.org/10.1145/2597073.2597074

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

[18] T. Pawelka and E. Juergens, "Is this code written in english? a study of the natural language of comments and identifiers in practice," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 401–410.

[19] N. S. R. 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*, 2014, pp. 1–7.

[20] M. A. de Freitas Farias, J. A. Santos, M. Kalinowski, M. Mendonça, and R. O. Spínola, "Investigating the identification of technical debt through code comment analysis," in *Enterprise Information Systems*, S. Hammoudi, L. A. Maciaszek, M. M. Missikoff, O. Camp, and J. Cordeiro, Eds. Cham: Springer International Publishing, 2017, pp. 284–309.

[21] 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/

[22] 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

[23] 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*, ser. ICSE-SEIS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–10. [Online]. Available: https://doi.org/10.1145/3377815.3381377

[24] Anonymised, "Anonymised manuscript - unit testing," *Anonymised Journal*, dec 2020.

[25] J. Singer, S. E. Sim, and T. C. Lethbridge, *Software Engineering Data Collection for Field Studies*. London: Springer London, 2008, pp. 9–34. [Online]. Available: https://doi.org/10.1007/978-1-84800-044-5_1

[26] E. Burmeister and L. M. Aitken, "Sample size: How many is enough?" *Australian Critical Care*, vol. 25, no. 4, pp. 271 – 274, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1036731412000847

[27] TIOBE, "Tiobe index - the software quality company," 2020. [Online]. Available: https://www.tiobe.com/tiobe-index/