On the Developers' Attitude Towards CRAN Checks

Pranjay Kumar
Davin Ie
RMIT University
Melbourne, Australia

Melina Vidoni
melina.vidoni@anu.edu.au
Australian National University
Australia

ABSTRACT

R is a package-based, multi-paradigm programming language for scientific software. It provides an easy way to install third-party code, datasets, tests, documentation and examples through CRAN (Comprehensive R Archive Network). Prior works indicated developers tend to code workarounds to bypass CRAN's automated checks (performed when submitting a package) instead of fixing the code–doing so reduces packages' quality. It may become a threat to those analyses written in R that rely on miss-checked code. This preliminary study card-sorted source code comments and analysed StackOverflow (SO) conversations discussing CRAN checks to understand developers' attitudes. We determined that about a quarter of SO posts aim to bypass a check with a workaround; the most affected are code-related problems, package dependencies, installation and feasibility. We analyse these checks and outline future steps to improve similar automated analyses.

CCS CONCEPTS

• **General and reference** → **Empirical studies**; • **Software and its engineering** → *Software libraries and repositories*; Software defect analysis.

KEYWORDS

Software Ecosystems, Package-Based Environment, R Programming, CRAN Checks

ACM Reference Format:
Pranjay Kumar, Davin Ie, and Melina Vidoni. 2022. On the Developers' Attitude Towards CRAN Checks. In *30th International Conference on Program Comprehension (ICPC '22), May 16–17, 2022, Virtual Event, USA*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3524610.3528389

1 INTRODUCTION

R is a multi-paradigm, package-based language for scientific software [15]. R developers contribute extension packages to the 'base' language [16] to CRAN (Comprehensive R Archive Network), which allows installing and importing packages with a single command. CRAN submissions are automatically checked [9, 18] on a process meant to filter submissions[1]. These checks are essential since organisations like rOpenSci or BioConductor adopted them in their quality control [3].

Prior works addressing R packages' source code comments determined developers tended to bypass CRAN's checks, creating workarounds instead of fixing the code [16]. This practice is problematic–it reduces package quality and introduces technical debt, discrediting such automated revision. In particular, "part of the complexity in measuring the scientific software ecosystem comes from how those different pieces of software are brought together and recombined into workflows" [8]. This complexity is critical for R because, given its package-based structure, new packages 'run off' previous packages, whose maintainability and quality are affected by the packages it relies on [1]. This is problematic since R's popularity grew over the last few years. The IEEE Spectrum ranked R the 7th most popular language in 2021[2], being considered among the fastest-growing programming languages. Lai et al. [11] demonstrated R grew from being used in 11.4% of the ecology papers to 58% (in 2019) and given BioConductor's impact, R is extensively used in bioinformatics research. [6].

This paper is a preliminary investigation of developers' intentions when approaching CRAN checks. We analysed source code comments and StackOverflow (SO) discussions to determine intention to avoid, fix or inquire about particular CRAN checks and which are affected. Given the relevance of CRAN for the R ecosystem [15], understanding this phenomenon can yield light about the "Build Debt" of R packages. "Build Debt" are issues making the build task harder and time-consuming, such as ill-defined dependencies; for R, it also encompasses CRAN (and its checks) code coverage, among other tools [3].

Preliminary results indicate coverage calculations are prone to manipulation with `nocov` tags in comments. Source code comments indicate the introduction of "Build Debt", "Code", and "Defect Debt" (in order), with checks related to "R Code Quality" being the most exposed. In StackOverflow, checks about "Description", "R Code", and "Package Structure" are the most queried, with about a third of enquiries resulting in an *avoidance* behaviour. Moreover, there was a peak of enquiries of any type during 2020.

The study dataset is available for reproduction purposes [10].

2 RELATED WORKS

Different studies assessed CRAN packages. One analysed over 5000 CRAN packages according to check-status, determining most errors are resolved within a few days without developer intervention [2]. Another evaluated GitHub's influence in CRAN, concluding that R packages hosted on GitHub suffer from inter-repository dependency problems, with CRAN-package updates causing backward-incompatible changes [7]. Another evaluated CRAN,

[1] Details on CRAN checks are available on: https://r-pkgs.org/r-cmd-check.html

[2] https://spectrum.ieee.org/top-programming-languages-2021

ICPC 2022, May 21–22, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9298-3/22/05…$15.00
https://doi.org/10.1145/3524610.3528389

BioConductor and R-Forge, determining that most packages depend on CRAN but had builds directly affected by the check-status.

Technical Debt (TD) describes the problem of balancing short-term value with long-term quality. An analysis of rOpenSci's peer-review process for R packages demonstrated the reviewers depend on CRAN checks and are concerned with "Build TD" [3]. A study on SATD (Self-Admitted Technical Debt) concluded that a considerable amount of source code comments admitted bypassing a check to prevent CRAN from detecting the situation instead of fixing it [16]. Using SATD comments to analyse developers' motivations has drawn attention from the research community [12]. This includes studying developers' reasons and purpose for introducing "Build TD" [19], determining intention through contextualised vocabulary [5], and finding priority of tasks to understand developers' logic [13].

3 METHODOLOGY

Our goal is to analyse how developers approach CRAN checks problems, determine if they are fixed or bypassed/avoided (with a workaround), which checks are most affected, and how this trend evolved. Our research questions (RQ) are: *RQ1) What CRAN checks are more discussed in source code comments, and which debt do they affect the most? RQ2) Are developers prone to bypass checks instead of fixing them?* Since there is no previous work on this domain, understanding *why* the avoidance happens is a future work dependent on answering the above RQs.

3.1 Comments Analysis

We analysed SATD source comments using the publicly available dataset of a prior work, which contains 4,962 source code comments (of 500 R packages) classified into TD types, also listing comment's length in line, file, and package name [16]. Our analysis of this dataset was done in phases:

Phase 1.1. Two authors independently classified a representative sample (95% of confidence, 5% of error) of 357 comments (without code) as *CRAN-related/not CRAN-related*. After, the authors discussed and analysed the differing classifications, reaching a consensus. They had a Cohen-Kappa (CK) agreement of 0.85, which supported a reliable classification [3].

Phase 1.2. The remaining 4605 comments were analysed similarly. About 50 were deemed 'explicitly CRAN-related' since they mentioned 'CRAN check' or 'CMD check'. The remaining were read and manually filtered. Both authors discussed their classification ($CK = 0.846$), reaching a consensus on the differences. We obtained dataset $D_1 = 393$ of 'possibly CRAN-related' comments (including the 'explicit'), and $D_2 = 411$ of `nocov` comments.

Phase 1.3. One author collected the corresponding functions or lines of code for each of the 393 comments of Phase-1.2.

Phase 1.4. Two authors analysed D_1 comments (with code) to remove false positives. The final discussion CK was 0.87. D_1' had 117 CRAN-related comments (including 'explicit'); about 2.35% of the original dataset. Many comments were ambiguously worded and required code inspection but were ultimately false positives. For example, `hacky fix for visible binding warning` was a developer warning, not CRAN-related.

Phase 1.5 Each D_1' record was classified into the categories used by CRAN to typify checks[1]; namely: metadata, structure, description, namespace, R code, data, documentation, demos, compiled code, tests and vignettes. Most comments had a single category, but 13.6% had two because the code could not be installed to reproduce it, or both were affected. This was decided through continuous discussion, and no CK was calculated.

3.2 StackOverflow Analysis

We mined SO discussions tagged as `cran` (totalling 717), irrespective of the publication year, following mining guidelines [4], and analysed them in phases:

Phase 2.1. Two authors analysed 85 posts (95% confidence, 5% error), using previously-agreed clear examples (title and question post) to define keywords representing CRAN-related posts. These were: CRAN, R CMD, check, package, submission, and devtools check (since `devtools::check()` runs CRAN checks locally).

Phase 2.2. We automatically labelled the posts by keywords and obtained six groups (determined by keyword count). We sampled 20% of each group, except for none or all keywords. Two authors independently analysed the sub-samples to assess the keywords' accuracy and determined that: 1/6 were posts unrelated to CRAN-checks (mostly only having 'package'), while 2/6 and 3/6 had false positives (not check-related posts). Thus, we applied the negative keyword 'install' to remove false positives. This produced $D_3 = 313$, of SO posts with 2-6 keywords each.

Phase 2.3. Two authors independently read each D_3 post, labelling them into: *avoid*, *fix*, or *not check-related*, according to the post's wording; e.g., 'how do I silence the warning' or 'ignore this warning and do X' were considered an avoidance. After, they discussed their classification ($CK = 0.832$), to reach consensus. Both authors agreed on a third category, labelled *process* (i.e., enquiries about the checking process, but not about a particular check). Finally, D_3' kept 200 posts.

Phase 2.4.. Every avoid/fix D_3' record was labelled by category of CRAN checks [18], independently by two authors. They discussed results ($CK = 0.84$), reaching a consensus on the disagreements. Posts labelled as *process* were not classified as CRAN-checks, except some explicitly inquiring about a specific check.

4 RESULTS

4.1 RQ1: Admission in Comments

D_2 contained 411 comments tagging `nocov`, originally labelled as "Testing TD" [16]. `nocov` is used by the package `covr` to exclude a path or function from the coverage calculation[3]. This manipulation *excludes* functions from the coverage, falsely increasing it, therefore related to the CRAN-checks. `nocov` comments are about 8.3% of the SATD comments. CRAN checks for testing are completed through `devtools::test()` andbased on `covr` tests[4], thus including coverage calculations. Since poor testing and over-reliance in coverage is a known issue [15], this was regarded as an admission of bypassing test-related checks.

The comments dataset (D_1') was originally classified by TD type [16]. Leveraging this data, Figure 1 displays the discussed checks (y-axis) per TD type (colour).

[3] See: https://cran.r-project.org/web/packages/covr/readme/README.html
[4] https://rdrr.io/cran/devtools/man/test.html

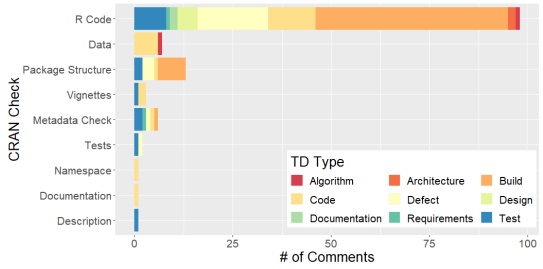

Figure 1: CRAN-check comments by TD type.

RQ Answer. About 43% comments admit "Build Debt", which appears almost every TD type uncovered in SATD comments. It is closely followed by "Code" and "Defect Debt" (each accounting for 18%). "Test Debt" comments in D_1' were either `nocov` with an explanation or related to another test issue. Regarding admission, "R Code" CRAN-checks are the most affected since it relates to syntax errors, incorrect dependencies, object-oriented usage, loading/unloading the package, visibility access, global variables, among others. With considerable difference in the comment numbers, the second affected check-type is "Package Structure". Given the dataset was small, we did not classify into specific evaluations per check-type.

Examples (given space limitations): `Useless assignments to pass R CMD check`, and `Dirty trick to avoid the warning of "no visible binding for global variable"`.

4.2 RQ2: Community Trends

The mean answer count was 1.045, with 23.7% having no answers. From the total, 59.6% posts did not have an approved answer (thus, some answers were not approved as such). On average, each question had about 2.5 comments. Most posts had four keywords (41.4%), two (40%), and five (13.1%).

RQ Answer. Regarding attitudes, about 23.3% was *avoidance*, with only 5.5% enquiring about specific checks (type *process*). Although somewhat positive, the proportion of avoidance is concerning. Figure 2 summarises the attitude of the post (fix, avoid or process enquiry) regarding each coded check.

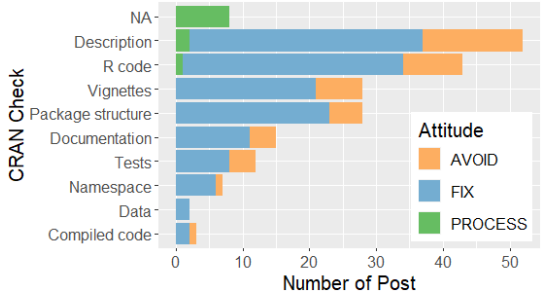

Figure 2: SO posts by CRAN check type and attitude.

Regarding *avoidance*, "Description" checks are prone to workarounds (about 28.9%; i.e., stop the warning/note, without fixing it). "R Code" checks accounted for almost 20.9%, and for "Vignettes", about 25% aimed to *avoid*, which is concerning, since vignettes are documentation to assist developers during package usage. 33% of "Tests" checks also indicated *avoidance*, aligned to the trend of tampering coverage calculations through comments.

Using the question's posting year, Figure 3 summarises the attitudinal trend (mined in March 2021, reflecting a proportion of that year). Enquiries increased in 2015 and 2020 when the posts doubled compared to previous years. The posts did not provide enough data to understand the reasons behind the increase. In 2014, avoidance and fixing attitudes were almost equalled.

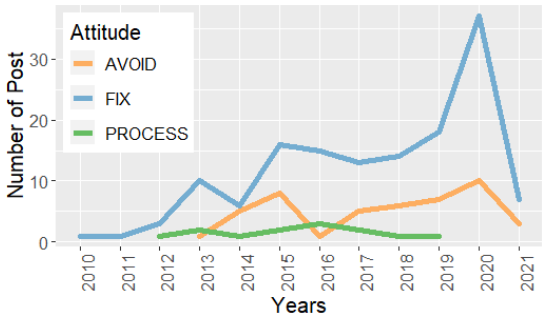

Figure 3: Attitudes about CRAN checks per year.

Table 1 summarises the most affected checks by attitude. The results are concerning since CRAN's checks are mandatory for submission, and both rOpenSci and BioConductor use them in their review process (ran automatically upon submission, or through `BioChecks`, respectively). In both cases, checks are a pre-filter before reviewers peer-review the package and occasionally steer the discussion (e.g., regarding formatting guidelines) [3].

Table 1: Common checks in SO, by attitude and posts counts.

Check	Fix (Posts #)	Avoid (Posts #)
R code for possible problems	27	7
Package dependencies	21	11
Whether package 'XYZ' can be installed	17	4
CRAN incoming feasibility	9	4

5 DISCUSSION & PROPOSALS

Overall, 8.3% of SATD comments reveal miscalculations of **coverage**. Although a coverage threshold for R packages remains undefined, their testing quality is low [15]. Given R's data science nature, this knowledge would benefit the community. Proposal: *Coverage* is not a good measurement for testing quality [17], thus, CRAN-checking the number of tested alternative paths would be more suitable, reinforcing the testing of edge/uncommon cases [15].

"R Code" is the most problematic and discussed check since it tackles code quality and introduces "Build", "Code", and "Defect Debt". Most are not aligned with R's `tidyverse` notation, forcing developers to do a workaround to avoid the note if using this syntax. Proposal: These checks are the widest, including design aspects (e.g., object-oriented consistency, replacement features), build aspects (e.g., dependencies), and generic code problems (e.g., syntax errors, non-ASCII characters). These should be specialised to give developers more specific information about the problem.

Aligning these checks with known TD types will be valuable given that there are R-specific taxonomies [3]. Proposal: `tidyverse`'s usage has grown considerably; by May 2020, `dplyr` was the third most-downloaded package, with other `tidyverse` packages being in April's 2021 Top 20. Including `tidyverse`-friendly check to avoid false-positive warnings would be benefit R developers.

Other checks appear briefly alongside SATD, but are prominent in SO. Checks such as "Description" and "Namespace" relate to standardised aspects and CRAN's standards [18], and may not be perceived until the developers decide to publish in CRAN. While some developers do not comment about these fixes, there may be minor discrepancies between the CRAN and GitHub versions of the same package. Proposal: Investigating developers' perceptions about CRAN checks through surveys or interviews. Corroborate if the avoidance of continuous fixing for specific checks leads to outdated GitHub repositories compared to CRAN versions.

"Check package dependencies" (inside "Description") considers dependencies installation, packages numbers, cyclic dependencies, consistency with the Namespace file [18]; near a third of queries tended to *avoid*. Previous studies demonstrated that in R's package-based environment, installation issues are transitive (a package no longer passing checks breaks the packages depending on it) [1, 7]. Others demonstrated the continuous growth of links between packages [14], while SATD comments indicate imported packages may force developers to bypass the problems of imported packages [16]. This relates to "check whether package XYZ can be installed" since all dependencies must be successfully installed, which a quarter of enquires tended to *avoid*.

Proposal: The Namespace file is automatically modified (through `roxygen2`'s system), but the Description is updated through a command and read in the checks reports. Incorporating automated suggestions (like Maven/Gradle projects do) would be helpful. Proposal: Package installation issues depend on the operative system–a check passing in CRAN's servers may not pass on a private environment or vice-versa. Although tools were proposed [2], no recent efforts assess this issue. Investigating "Installation" and "Build Debt" in packages is essential, given its package-based nature and its effects in package builds [1].

"Check R code for possible problems" compounds syntax quality issues [18], and about a fifth SATD comments tended to *avoid*. CRAN interprets imported packages' variables as global if not called with the double-colon operator (`package::variable`)[5]. Although somewhat related to dependencies, the check's message is unclear. Proposal: Many notes have unrelated messages, leading to a challenging debugging process. Clear messages with error codes would assist the developers, possibly developed and refined through a community effort.

"Check CRAN incoming feasibility" performs format checks (e.g., versioning, licensing, spelling mistakes) including warnings for CRAN maintainers [18]. A third of the posts *avoided* or ignored these checks. Proposal: Three-quarters of avoiding posts discussed maintainers-specific checks[6]. These checks simplify the (manual) internal process but are misunderstood by new developers and clog checks' lists with unneeded information. Only maintainers should see this check. Proposal: Regarding spelling problems, CRAN detected non-ASCII authors' names as a grammatical error[7]. Non-ASCII characters in the Description file's `Authors` or `Authors@R` section should be ignored. Understandably, package titles should remain ASCII-only.

[5]Example: https://tinyurl.com/cran-example-1
[6]Example: https://tinyurl.com/cran-example-2

6 THREATS TO VALIDITY

Construct. D_1's TD types were validated in a prior study and subject to those threats [16].

Internal. The independent classification and agreement discussion between authors reduced researcher bias; CKs were always higher than 0.79, indicating a strong agreement [3]. The initial comments filtering (phases 1.1-1.2) excluded the code, but threats were minimised by having two authors analyse it. Through the number of comments is small, they are representative of current R packages (extracted from 500 packages, accounting for over 160k comments [16]). Regarding SO, our dataset included posts tagged (in SO) as `cran`. It is possible a miss-tag occurred (incorrectly tagged or a relevant post without the tag); however, we assumed the accuracy of SO's curators regarding tag edition.

External. The comments dataset was extracted from over 500 [16]; SO's dataset had about 200 records which, even if they represent all of SO's `cran`-tagged posts, remains a small sample. Regarding other communities, RStudio Community only had 36 CRAN posts without the check-related filtering, and the mailing list restricts who can participate. Although our results are generalisable for R packages, they do not apply to R scripts or RMarkdown. Likewise, though our findings are relevant for the R community, similar environments (e.g., npm or PyPi) cannot directly apply them.

7 CONCLUSION

This paper presents a preliminary investigation about R developers' approach to CRAN's checks, analysing source code comments and StackOverflow's (SO) posts.

We uncovered a tendency to tamper coverage measurements, affecting CRAN's checks results. Checks related to "R Code" are frequent in source code comments and SO posts, with "Description", "Vignettes", and "Package Structure" mentioned mainly in SO. About a quarter of SO discussions aim to bypass checks through a workaround instead of fixing it, and the most affected checks relate to dependencies, installation, feasibility and possible code problems; a third of the dependency-related discussions tend to *avoid*. We also proposed *nine* future works for the R community to improve or adjust CRAN checks.

Detecting issues through mandatory vetting tests is relevant to communities that currently do not enforce them (e.g., PyPi), as it uncovers issues prior to package distribution. Moreover, our results warn community-led efforts for package peer-review that are currently using automated checks as part of their review process.

The most critical **future work** is analysing how the developers' attitude (fix or avoid) affects packages' quality (i.e., "Build TD") and the transitivity of such effects (i.e., also impacting packages depending on the target one). Future works should extend into analysing non-compulsory, developer-led checks and their effect on other platforms (e.g., PyPi or npm).

[7]Example: https://tinyurl.com/cran-example-3

REFERENCES

[1] Elvira-Maria Arvanitou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, and Jeffrey C. Carver. 2021. Software Engineering Practices for Scientific Software Development: A Systematic Mapping Study. *Journal of Systems and Software* 172 (2021), 110848. https://doi.org/10.1016/j.jss.2020.110848

[2] Maëlick Claes, Tom Mens, and Philippe Grosjean. 2014. On the Maintainability of CRAN Packages. In *IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. IEEE, Antwerp, Belgium, 308–312. https://doi.org/10.1109/CSMR-WCRE.2014.6747183

[3] Zadia Codabux, Melina Vidoni, and Fatemeh Fard. 2021. Technical Debt in the Peer-Review Documentation of R Packages: a rOpenSci Case Study. In *International Conference on Mining Software Repositories*. IEEE, Madrid, Spain.

[4] Marco D'Ambros and Romain Robbes. 2011. Effective Mining of Software Repositories. In *27th IEEE International Conference on Software Maintenance*. IEEE, USA, 598–598. https://doi.org/10.1109/ICSM.2011.6080839

[5] Mário André de Freitas Farias, Manoel Gomes de Mendonça Neto, Marcos Kalinowski, and Rodrigo Oliveira Spínola. 2020. Identifying Self-Admitted Technical Debt Through Code Comment Analysis with a Contextualized Vocabulary. *Information and Software Technology* 121 (2020), 106270. https://doi.org/10.1016/j.infsof.2020.106270

[6] Ariel Deardorff. 2020. Why do Biomedical Researchers Learn to Program? An Exploratory Investigation. *Journal of the Medical Library Association : JMLA* 108, 1 (Jan 2020), 29–35. https://doi.org/10.5195/jmla.2020.819

[7] Alexandre Decan, Tom Mens, Maëlick Claes, and Philippe Grosjean. 2016. When GitHub Meets CRAN: An Analysis of Inter-Repository Package Dependency Problems. In *23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. IEEE, Osaka, Japan, 493–504. https://doi.org/10.1109/SANER.2016.12

[8] James Howison, Ewa Deelman, Michael J. McLennan, Rafael Ferreira da Silva, and James D. Herbsleb. 2015. Understanding the Scientific Software Ecosystem and its Impact: Current and Future Measures. *Research Evaluation* 24, 4 (07 2015), 454–470. https://doi.org/10.1093/reseval/rvv014

[9] James Howison and James D. Herbsleb. 2011. Scientific Software Production: Incentives and Collaboration. In *ACM Conference on Computer Supported Cooperative Work* (Hangzhou, China) *(CSCW '11)*. ACM, New York, USA, 513–522. https://doi.org/10.1145/1958824.1958904

[10] Pranjay Kumar, Davin Ie, and Melina Vidoni. 2022. *CRAN Checks' Avoidance*. https://doi.org/10.5281/zenodo.6342280

[11] Jiangshan Lai, Christopher J. Lortie, Robert A. Muenchen, Jian Yang, and Keping Ma. 2019. Evaluating the Popularity of R in Ecology. *Ecosphere* 10, 1 (2019), e02567. https://doi.org/10.1002/ecs2.2567

[12] Solomon Mensah, Jacky Keung, Jeffery Svajlenko, Kwabena Ebo Bennin, and Qing Mi. 2018. On the Value of a Prioritization Scheme for Resolving Self-Admitted Technical Debt. *Journal of Systems and Software* 135 (2018), 37–54. https://doi.org/10.1016/j.jss.2017.09.026

[13] Junior Cesar Rocha, Vanius Zapalowski, and Ingrid Nunes. 2017. Understanding Technical Debt at the Code Level from the Perspective of Software Developers. In *Proceedings of the 31st Brazilian Symposium on Software Engineering* (Fortaleza, CE, Brazil) *(SBES'17)*. Association for Computing Machinery, New York, NY, USA, 64–73. https://doi.org/10.1145/3131151.3131164

[14] Stefan Theußl, Uwe Ligges, and Kurt Hornik. 2011. Prospects and Challenges in R Package Development. *Computational Statistics* 26, 3 (01 Sep 2011), 395–404. https://doi.org/10.1007/s00180-010-0205-5

[15] M. Vidoni. 2021. Evaluating Unit Testing Practices in R Packages. In *43rd International Conference on Software Engineering (ICSE)*. IEEE, Madrid, Spain, 1–12.

[16] M. Vidoni. 2021. Self-Admitted Technical Debt in R Packages: An Exploratory Study. In *International Conference on Mining Software Repositories*. IEEE, Madrid, Spain.

[17] Tássio Virgínio, Railana Santana, Luana Almeida Martins, Larissa Rocha Soares, Heitor Costa, and Ivan Machado. 2019. On the Influence of Test Smells on Test Coverage. In *XXXIII Brazilian Symposium on Software Engineering* (Salvador, Brazil) *(SBES 2019)*. Association for Computing Machinery, USA, 467–471. https://doi.org/10.1145/3350768.3350775

[18] Hadley Wickham. 2015. *R Packages* (1st ed.). O'Reilly Media, Inc., USA.

[19] T. Xiao, D. Wang, S. Mcintosh, H. Hata, R. Kula, T. Ishio, and K. Matsumoto. 5555. Characterizing and Mitigating Self-Admitted Technical Debt in Build Systems. *IEEE Transactions on Software Engineering* X, 1 (sep 5555), 1–1. https://doi.org/10.1109/TSE.2021.3115772