

# Does Technical Debt Lead to the Rejection of Pull Requests?

Marcelino Campos  
Oliveira Silva  
Chemtech - A Siemens  
Business  
Belo Horizonte, Brazil  
marcelino.silva@siemens.com

Marco Tulio Valente  
Departamento de Ciência da  
Computação  
Universidade Federal de  
Minas Gerais  
Belo Horizonte, Brazil  
mtov@dcc.ufmg.br

Ricardo Terra  
Departamento de Ciência da  
Computação  
Universidade Federal de  
Lavras  
Lavras, Brazil  
terra@dcc.ufla.br

## ABSTRACT

Technical Debt is a term used to classify non-optimal solutions during software development. These solutions cause several maintenance problems and hence they should be avoided or at least documented. Although there are a considered number of studies that focus on the identification of Technical Debt, we focus on the identification of Technical Debt in pull requests. Specifically, we conduct an investigation to reveal the different types of Technical Debt that can lead to the rejection of pull requests. From the analysis of 1,722 pull requests, we classify Technical Debt in seven categories namely design, documentation, test, build, project convention, performance, or security debt. Our results indicate that the most common category of Technical Debt is design with 39.34%, followed by test with 23.70% and project convention with 15.64%. We also note that the type of Technical Debt influences on the size of push request discussions, e.g., security and project convention debts instigate more discussion than the other types.

## Categories and Subject Descriptors

D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement—*Version control*.

## General Terms

Design, Experimentation.

## Keywords

Technical Debt, Pull Request, GitHub.

## 1. INTRODUCTION

Software developers usually face challenges to build complex systems in a short deadline. More relevant, in order to accomplish such deadlines, they may implement their tasks

using sub-optimal technical approaches, which increases the costs of the maintenance in the future. When this situation ascends, the project acquires Technical Debt [4].

In this paper, we focus on Technical Debt through code submission via pull requests. Our goal, thereupon, is to investigate what are the most common Technical Debts that cause rejection of pull requests.

Such investigation provides open source developers with identification of the most frequent types of Technical Debts, which leads at least to the following contributions: (i) the identification of a high occurrence of one type of problem can alert both reviewers and developers; (ii) developers can produce a checklist of the most common Technical Debts before submitting pull requests; and (iii) reviewers can better evaluate pull requests when they are aware of some types of problems, e.g., a reviewer may approve a code that successfully solves a particular bug, but in return, brings problems in the management of memory, which will only be realized in the future.

This paper is organized as follows. Section 2 provides background information about pull-based development and technical debt. Section 3 describes the methodology of our study, stating four research questions and presenting our dataset. Section 4 answers and discusses each research question. Finally, Section 5 states threats to validity and Section 6 concludes.

## 2. BACKGROUND

### 2.1 Distributed Software Development

Distributed software development allows developers to work on a software project even being geographically dispersed. There are a bunch of integration techniques that help to produce software in a distribute way, but a particular one has become very popular nowadays: pull-based development [8].

The pull-based development process has been widely used and is quickly becoming the default one. Large source code hosting sites (e.g., GitHub, BitBucket, and Gitorious) support pull-based development, totalizing more than a million projects, which indicates its current relevance. In pull-based development, a developer pushes changes to a repository and describes the underlying changes, e.g., new features, bug fixes, improvements, etc. Soon after, developers submit a pull request, interested parties evaluate it in order to accept or reject its integration to the project.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.  
SBSI 2016, May 17th-20th, 2016, Florianópolis, Santa Catarina, Brazil  
Copyright SBC 2016.



Figure 1: Study Design Overview

In GitHub<sup>1</sup>, for instance, contributors and reviewers can interact through three kinds of comment views: (i) comments on the pull request itself, (ii) comments on a specific line within the pull request, and (iii) comments on a specific commit within the pull request. The acceptance or rejection of a pull request can trigger many interactions between the contributors and reviewers. This process is not trivial since it involves both technical and social communication aspects [15].

Pull request evaluation varies from a project to another. Gousios et al. [9] investigated quantitatively what factors are behind the acceptance in pull-based development. They report that all projects have a similar process, but the dominating factors (stickiness of project area and social distance, respectively) are massively different. On one hand, this difference suggests that there is no common process for evaluating contributions in pull-based development. On the other hand, there are cases in which the owner relies on bots to inspect and reject pull requests that do not comply with contributor license agreement.

Bacchelli and Bird [2] investigate a specific technique used in pull-based development named “modern” code review. Although it is very similar to the traditional code review, “modern” code review determines *a priori* the process for accepting a contribution and requires sign-off by a specific number of collaborators. They argue that, although the main purpose of “modern” code review is to find defects, in practice the benefits of knowledge transfer and team awareness counterpoise these purposes.

## 2.2 Technical Debt

In our study, we rely on the terminology coined by Fowler [7] which defines that Technical Debt occurs when developers adopt an un-optimal or sub-optimal technical decision. Therefore, solutions that do not solve the whole problem are not considered in our study. Alves et al. [1], for example, consider bugs and unimplemented requirements as types of Technical Debt. In contrast, we do not consider them as Technical Debt since they do not solve the original problem they have been proposed to. We also rely on the definition of Curtis et al. [5] that state that Technical Debt should be distinguished from defects or failures. Failures during testing or operation can be symptoms of Technical Debt, but most structural issues related to Technical Debt may not lead to test or operational failures. Instead, they may cause a less efficient use, less scalable, more difficult to maintain, or more vulnerable system. In essence, Technical Debt emerges from poor design quality and affects business as well as IT costs and business risks.

Similarly to our study, there are studies that focus on the identification of Technical Debt. For example, Santos et

<sup>1</sup><https://developer.github.com/guides/working-with-comments/>

al. [16], Letouzey et al. [11] and Marinescu [13] use metrics based on cohesion, coupling, code duplication, lack of comments, coding rules violation, potential bugs, and the absence of unit tests to reveal the existence of Technical Debt.

There also are other studies that identify Technical Debt in a restricted way. Zazworka et al. [18] focus on a particular kind of design debt, namely God Classes. Fontana et al. [6] investigate design Technical Debts that appear in the form of bad smells. Potdar and Shihab [14], and Maldonado and Shihab [12] investigate the detection of Technical Debt by manually inspecting code comments.

Zazworka et al. [4] conduct an experiment to compare the efficiency of automated tools in comparison with human elicitation, regarding the detection of Technical Debt. They report that there is a small overlap between the two approaches and hence the best solution is to combine them. They conclude that automated tools are more efficient in finding defect Technical Debt, whereas developers can realize more abstract categories of Technical Debt.

## 3. STUDY DESIGN

As illustrated in Figure 1, we define the following protocol to conduct our study:

- (a) Define research goals and questions,
- (b) Define dataset and resources,
- (c) Define study data filtering procedure and exclusion criteria,
- (d) Conduct manual inspection, and
- (e) Analyze the results.

The next subsections describe in details each aforementioned step.

### 3.1 Research Questions

This study aims to understand how the rejection of pull requests relates to Technical Debt. Using the Goal-Question-Metric format proposed by Basili et al. [17], we defined the following research questions:

#### **RQ #1: How often pull requests are rejected?**

This question investigates the percentage of rejection in pull requests.

#### **RQ #2: Does Technical Debt lead to the rejection of pull requests?**

This question investigates whether reviewers and contributors list Technical Debt as the main reason to reject pull requests.

Table 1: Target Projects

System	URL	Domain	LOC	PR's
Elastic Search	https://github.com/elastic/elasticsearch	Full-text search engine	428,550	6,115
IntelliJ	https://github.com/JetBrains/intellij-community	Programming IDE	504,505	323
Iosched	https://github.com/google/iosched	Conference Application	41,666	72
Picasso	https://github.com/square/picasso	Image downloading library	9,238	344
Retrofit	https://github.com/square/retrofit	HTTP client for Android	6,013	523
Storm	https://github.com/apache/storm/	Distributed real-time system	46,306	262
<b>Total</b>				<b>7,639</b>

**RQ #3: What are the most common types of Technical Debt that lead to the rejection of pull requests?**

Assuming a positive answer to RQ #2, this research question goes further. Although Technical Debt is a general term, some studies reported different types of Technical Debt [1]. Nevertheless, the types of Technical Debts that occur in pull requests remain unknown.

**RQ #4: Which types of Technical Debt spur more discussion?**

This research question, which also assumes a positive answer to RQ #2, verifies whether a particular type of Technical Debt instigates more discussion than the others. Therefore, we investigate whether contributors discuss more to reject and close a pull request incurring in a specific type of debt, rather than others.

### 3.2 Dataset

Our study relies on six open source projects, namely IntelliJ, Elastic Search, Iosched, Picasso, Retrofit, and Storm, as detailed in Table 1. We chose the aforementioned projects, since they (i) are publicly available in GitHub, (ii) belong to different application domains, (iii) have more than a thousand stars, and (iv) follows a pull-based development process, i.e., they accept pull requests from the community.

We extracted all pull requests from the six projects.<sup>2</sup> To carry out this activity, we developed an extractor of pull requests using the GitHub API, which allowed us to easily extract pull requests that were closed along a project's lifetime, with the three kinds of comment (as detailed in Section 2.1). Altogether, our study investigates 7,639 pull requests.

### 3.3 Selection procedure

Our study relies on the following inclusion/exclusion criteria in order to analyze only consistent data.

- (Inclusion) We only consider pull requests that have been closed;
- (Exclusion) We disregard pull requests that had no reviews because without comments we cannot figure out the reason for the rejection; and
- (Exclusion) We disregard pull requests that (i) had no comments from at least one project's owner or (ii) had only automatic comments from bots.

<sup>2</sup>The extraction was performed on December 11, 2015

As reported in Table 2, the aforementioned criteria reduced the number of pull requests in our initial dataset. The largest reduction occurred in Elastic Search, which reduced the number of pull requests from 6,115 to 584 (9.6%).

Table 2: Filtering Results

System	PR's (A)	PR's After Filtering (B)	Comments After Filtering	% of selected PR (B/A)
Elastic Search	6,115	584	1,374	9.6%
IntelliJ	323	226	603	70.0%
Iosched	72	48	62	66.7%
Picasso	344	339	509	98.5%
Retrofit	523	319	703	61.0%
Storm	262	206	512	78.6%
<b>Total</b>	<b>7,639</b>	<b>1,722</b>	<b>3,763</b>	<b>22.5%</b>

### 3.4 Manual Investigation

We have classified the 1,722 pull requests that remaining in our dataset after applying the selection procedure in one of the following seven different types of Technical Debts: (i) *design*, (ii) *documentation*, (iii) *test*, (iv) *build*, (v) *project convention*, (vi) *performance*, or (vii) *security* debt.

The classification was made by the first author of this paper, who has more than nine years of experience working as a software engineer, in particular with Java programming language. We argue that these qualifications provide him the necessary background to conduct the manual classification of the comments.

The first author manually read through pull requests as described in Section 2.1. While examining them, he classified each item by the nature of the debt. Some instances could be classified in more than one type of debt, e.g., a code with a poor implementation could be considered a *design* debt, but it could also be considered a *performance* debt since it causes excessive consumption of memory. Although this situation may have different interpretations depending of who is reading the comments, we defined that each pull request would have only one classification type for the sake of clarity. Basically, we considered the more meaningful type for each scenario. In total, we read and classified 1,772 pull requests from six open source projects.

Table 3: Percentage of rejected pull requests

System	Analyzed PR (A)	Rejected PR (B)	PR rejected due to TD (C)	% of rejection (B/A)	% of rejection due to TD (C/B)
IntelliJ	226	211	26	93.4%	12.3%
Iosched	48	22	1	45.8%	4.5%
Picasso	339	112	59	33.0%	52.7%
Storm	206	112	32	54.4%	28.6%
Retrofit	319	103	49	32.3%	47.6%
Elastic Search	584	119	39	20.4%	30.3%
<b>Total</b>	<b>1,722</b>	<b>679</b>	<b>206</b>	<b>39.4%</b>	<b>31.1%</b>

Table 4: Results per category

Type of TD	IntelliJ	Iosched	Picasso	Storm	Retrofit	Elastic Search	Total	% per type
Build	0	1	2	2	0	0	<b>5</b>	2.37%
Design	15	0	9	19	27	13	<b>83</b>	39.34%
Documentation	0	0	4	3	0	5	<b>12</b>	5.69%
Performance	2	0	12	0	3	3	<b>20</b>	9.48%
Project Convention	5	0	10	4	6	8	<b>33</b>	15.64%
Security	0	0	2	0	1	0	<b>3</b>	1.42%
Tests	4	0	20	4	12	10	<b>50</b>	23.70%
Unclassified	185	21	53	80	54	80	<b>468</b>	—
<b>Total</b>	<b>211</b>	<b>22</b>	<b>112</b>	<b>112</b>	<b>103</b>	<b>119</b>	<b>679</b>	—

## 4. RESULTS

This section provides answers for the four research questions of our study.

### RQ #1: How often pull requests are rejected?

Our results indicate that 679 out of 1,722 pull requests were rejected. Therefore, 39.4% of pull requests are rejected, on average.

### RQ #2: Does Technical Debt lead to the rejection of pull requests?

Table 3 summarizes the data related to pull requests rejection. The last two columns show, respectively, the percentage of rejected pull requests and the percentage of rejected pull requests due to Technical Debt. The data of last column is the percentage of items with Technical Debt related to the rejected pull requests (instead of all ones). Our results indicated that 206 out of the 679 rejected pull requests are due to Technical Debt. Therefore, 30.3% of the rejected pull requests are due to Technical Debt.

It is worth noting that Picasso is the project with the highest percentage of pull requests rejected due to Technical Debt reasons (52.7%). Retrofit and Elastic Search immediately follows with respectively 47.6% and 32.8%. These high percentages suggest that Technical Debt is a very strong criterion for rejecting a pull request.

### RQ #3: What are the most common types of Technical Debt that lead to the rejection of pull requests?

As reported in Table 4, we found 206 pull requests rejected due to Technical Debt. These pull requests can be classified in one of the following seven types: *design*, *documentation*, *test*, *build*, *project convention*, *performance*, or *security* debt. The last column shows the percentage of each type of Technical Debt found in our study. The data is normalized, presenting the percentages of the different types rather than the raw numbers. For example, if a project has 100 pull requests that were rejected due to Technical Debt and 15 are due to design debt, then the percentage of *design* debt is 15%. Our results indicate that the most frequently Technical Debt encountered was *design* (39.34%), followed by *test* (23.70%), *project convention* (15.64%), *performance* (9.48%), *documentation* (5.69%), *build* (2.37%), and *security* (1.42%).

*Design Debt:* It refers to problems such as architecture violations, use of bad programming practices, violations of good object-oriented design principles, misuse of design-pattern, misplaced code, lack of abstraction, poor implementation, and workarounds or temporary solutions. This debt represents 39.34% of the rejection of pull requests. From our dataset, we can highlight the following examples:

“These  $X^*$  classes tend to be very painful to maintain when doing Lucene upgrades.”

(Elastic Search)

“Wouldn’t it be better to catch and suppress `FileNotFoundException` in `rmpath` itself? The ultimate goal of the function is to ensure that path is deleted, so it’s fine to suppress that error.”

(Storm)

*Design debt* was the most frequently Technical Debt encountered, although not in all projects. In Picasso, for example, the most common type was related to *test* debt. This may suggest that the major concerns of each team is different, as a group has a greater concern for correct use of best practices, others emphasize more well-tested systems.

*Documentation Debt*: It refers to problems found in the project documentation and can be identified by looking for missing, inadequate, or incomplete documentation of any type. It can also be related to architectural documents, code comments, Javadoc, release notes, etc. Besides, a poor description of the pull request reason is also considered a documentation problem. This debt was responsible for 5.69% of the rejection of pull requests. We can highlight the following examples:

“Closing this in favor of documenting `indices.get_gield_mapping` as ‘fields’, this to resemble to query string option present on several API’s.”

(Elastic Search)

“Documentation tends to become out of sync with the code, and it can become a maintenance nightmare. I’d rather we did this as a wiki page with links into the code, ala what I started doing here: <https://github.com/nathanmarz/storm/wiki/Implementation-docs>”

(Storm)

*Test Debt*: It specifies the need for implementation or improvement of the submitted tests. This debt was responsible for 23.70% of the rejection of pull requests. We can highlight the following examples:

“Do you think you’d be able to add a test for this behavior?”

(Picasso)

*Build Debt*: It refers to any issue that could compromise build or deployment. Changes that make the build or deploy harder, consuming more time/processing unnecessarily, or hindering the deploy in some way. For example, changes in the build process can make it use more dependencies than necessary. This debt was responsible for 2.37% of the rejection of pull requests. From our dataset, we can highlight the following examples:

“[...] looks like you’re having some weird quirks on your builds [...]”

(Iosched)

*Performance Debt*: It refers to any part of code that can delay or hinder the system’s performance. For example, it may include any piece of code that heavily use the network or increases the use of memory heap.

This debt was responsible for 9.48% of the rejection of pull requests. We can highlight the following examples:

“We are not going to buffer every download. This will destroy the heap.”

(Picasso)

“This should use double-checked locking. You’re optimizing for the worst case instead of the best case.”

(Picasso)

*Project Convention Debt*: It deals with the format and style conventions adopted by the project. For example, names of variables, method size, and indentation. It is worth noting that this debt was initially comprehended as *design*. However, during our classification process, we noted that this scenario is very important in some projects. Therefore, we considered these issues as a new type. It was responsible for 15.64% of the rejection of pull requests. We can highlight the following examples:

“Sorry, I’m not able to accept a pull request that renames all fields in a class so that they no longer match the IntelliJ IDEA coding style. Please make the minimum amount of changes required for your refactoring and follow the style of the codebase you’re contributing to.”

(IntelliJ)

*Security Debt*: It refers to a particular solution that compromises the security of the system, e.g., introducing a particular breach of security. This debt was responsible for 1.42% of the rejection of pull requests. We can highlight the following example:

“This class should be `package-private`. The constructor and factory method are `package-private`, and it’s not safe for general use.”

(Retrofit)

We did not classify the pull requests that did not comply with any of the defined types (*unclassified*). These pull requests were rejected due to other reasons than Technical Debt, e.g., they contained bugs or compilation errors, they did not attend a business requirement, they had not performed rebase, or they had not signed contributor license agreement.

#### ***RQ #4: Which types of Technical Debt spur more discussion?***

Last but not least, we investigated whether the type of Technical Debt influences on the size of a discussion. Table 5 reports the total number of comments per type of Technical Debt and also per system. For instance, *build* debt had a total of 11 comments in 5 pull requests, which results in 2.2 comments per pull request on average. Therefore, our results indicated that the *security* and *project convention*

Table 5: Total discussion per type of Technical Debt and per system

Type of TD	# of comments						Total per type (A)	Total of PR (B)	Average (A/B)
	IntelliJ	Iosched	Picasso	Storm	Retrofit	Elastic Search			
Build	0	3	4	4	0	0	11	5	2.2
Design	37	0	22	34	54	52	199	83	2.4
Documentation	0	0	10	10	0	12	32	12	2.7
Performance	12	0	40	0	6	12	70	20	3.5
Project Convention	14	0	35	7	54	24	134	33	4.1
Security	0	0	8	0	14	0	22	3	7.3
Tests	15	0	79	8	30	44	176	50	3.5
Total per system	78	3	198	63	158	144	644	206	3.1

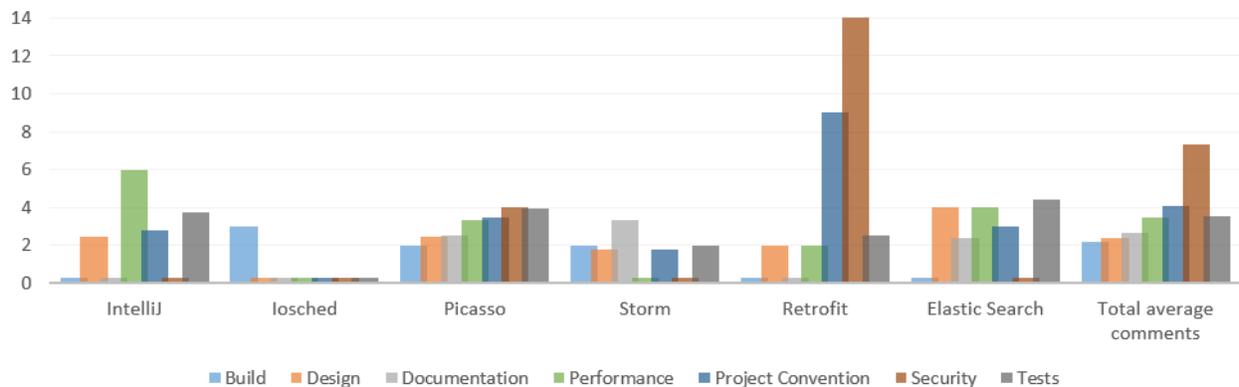


Figure 2: Average discussion per type of Technical Debt and per system

debts had the highest number of comments per pull request on average. While the overall average was 3.1 comments per pull request, *security* and *project convention* debts had on average 7.3 and 4.1 comments respectively.

As a complementary analysis, Figure 2 illustrates the average of comments by pull request per type of Technical Debt and also per system. For instance, while the *security* debt had the highest overall average of comments per pull request, particularly for IntelliJ, the highest average was the *performance* debt having 12 comments in 2 pull requests, which results in 6.0 comments per pull request on average.

We state that the *security* debt achieves the highest overall average, i.e., 7.3 comments per pull request. We conjecture that security is a very delicate topic because sometimes we must choose between increasing system security at the expense of performance, for example. Besides *security*, *project convention* and *test* debts are types that apparently do not have such a consensus. For example, software development conventions are not always very clear and also vary from project to project.

*Test debt* also generates a considerable number of discussions. Because tests are very important for a system, this is somewhat expected. Hetzel [10] stated that it is difficult

to say to what extent a system is completely covered by tests, which naturally triggers a lot of discussion. On the other hand, *design debt* had one of the lowest overall average (2.4 comments per pull request), which was unexpected since such finding contrasts to recent empirical study that reported that design is a topic that causes much debate [3]. After an investigation, we conjecture that since software design practices and patterns are well spread in the community, there are no longer much discussion on this topic.

## 5. THREATS TO VALIDITY

Our classification of pull request is manual because comments are written in natural language and therefore difficult to analyze by a machine. Like any human task, it is propitious to personal subjectivity. To reduce this threat, the main author double checked all classifications.

Our dataset relies on six open source projects. The projects are written in Java and publicly available for replication. We also chose projects from different domains to minimize external validity. However, as usual in empirical studies in software engineering, the results obtained may not be generalized to other projects, e.g., implemented in other languages.

## 6. CONCLUSION AND FUTURE WORK

Technical Debt is used to denote shortcuts and workarounds employed in software projects, which can impact the maintainability of the project and even hinder the development if not properly addressed. This paper explores identification and classification of Technical Debt introduced by the developers through pull-based development. We investigated pull requests of six open source projects namely IntelliJ, Elastic Search, Iosched, Picasso, Retrofit, and Storm. These projects are considered renowned and they have rigorous review processes.

We find that Technical Debt can be classified into seven types namely *design*, *documentation*, *test*, *build*, *project convention*, *performance*, and *security* debts. More important, we provide real examples of each type and the basis for its classification. Our results indicate that the most frequent Technical Debt is *design* (39.34%), followed by *test* (23.70%), *project convention* (15.64%), *performance* (9.48%), *documentation* (5.69%), *build* (2.37%), and *security* (1.42%). We also note that type of Technical Debt influences on the size of a discussion, i.e., *security* and *project convention* debts spur more comments prior to be rejected than the other types. As further work, we look forward to to identify how the type of Technical Debt associated with rejection influences the time of the problem solving. For this, we plan to examine pull requests that were rejected and were successfully resubmitted afterwards.

We made our dataset publicly available.<sup>3</sup> We expect this dataset inspires future research in the area, e.g., using natural language processing techniques.

## Acknowledgments

Our research is supported by FAPEMIG, CNPq, Chemtech, and Siemens.

## 7. REFERENCES

- [1] Nicolli S.R. Alves, Leilane F. Ribeiro, Vivyane Caires, Thiago S. Mendes, and Rodrigo O. Spinola. Towards an ontology of terms on technical debt. In *6th International Workshop on Managing Technical Debt (MTD)*, pages 1–7, 2014.
- [2] Alberto Bacchelli and Christian Bird. Expectations, outcomes, and challenges of modern code review. In *33rd International Conference on Software Engineering (ICSE)*, pages 712–721, 2013.
- [3] Joao Brunet, Gail C. Murphy, Ricardo Terra, Jorge Figueiredo, and Dalton Serey. Do developers discuss design? In *11th Working Conference on Mining Software Repositories (MSR), Challenge Track*, pages 340–343, 2014.
- [4] Ward Cunningham. The wycash portfolio management system. In *Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum)*, 1992.
- [5] Bill Curtis, Jay Sappidi, and Alexandra Szykarski. Estimating the size, cost, and types of technical debt. In *3rd International Workshop on Managing Technical Debt (MTD)*, pages 49–53, 2012.
- [6] Francesca Arcelli Fontana, Vincenzo Ferme, and Stefano Spinelli. Investigating the impact of code smells debt on quality code evaluation. In *3rd International Workshop on Managing Technical Debt (MTD)*, pages 15–22, 2012.
- [7] Martin Fowler. Technicaldebt. <http://martinfowler.com/bliki/TechnicalDebt.html>, 2003.
- [8] Georgios Gousios, Martin Pinzger, and Arie van Deursen. An exploratory study of the pull-based software development model. In *36th International Conference on Software Engineering (ICSE)*, pages 345–355, 2014.
- [9] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie van Deursen. Work practices and challenges in pull-based development: The integrator’s perspective. In *37th International Conference on Software Engineering (ICSE)*, pages 358–368, 2015.
- [10] William C. Hetzel and Bill Hetzel. *The Complete Guide to Software Testing*. John Wiley & Sons, 1991.
- [11] Jean-Louis Letouzey. The SQALE method for evaluating technical debt. In *3rd International Workshop on Managing Technical Debt (MTD)*, pages 31–36, 2012.
- [12] Everton Maldonado and Emad Shihab. Detecting and quantifying different types of self-admitted technical debt. In *7th International Workshop on Managing Technical Debt (MTD)*, pages 9–15, 2015.
- [13] Radu Marinescu. Assessing technical debt by identifying design flaws in software systems. *IBM Journal of Research and Development*, 56(5), 2012.
- [14] Amit Potdar and Emad Shihab. An exploratory study on self-admitted technical debt. In *30th International Conference on Software Maintenance and Evolution (ICSME)*, pages 91–100, 2014.
- [15] Peter C. Rigby and Margaret-Anne Storey. Understanding broadcast based peer review on open source software projects. In *33rd International Conference on Software Engineering (ICSE)*, pages 541–550, 2011.
- [16] Paulo Sérgio Medeiros Santos, Amanda Varela, Cristine Ribeiro Dantas, and Daniel Beltrão Borges. Visualizing and managing technical debt in agile development: An experience report. In *14th International Conference on Agile Software Engineering (XP)*, pages 121–134, 2013.
- [17] Rini Van Solingen, Vic Basili, Gianluigi Caldiera, and H Dieter Rombach. Goal question metric (GQM) approach. *Encyclopedia of software engineering*, 2002.
- [18] Nico Zazworka, Michele A. Shaw, Forrest Shull, and Carolyn Seaman. Investigating the impact of design debt on software quality. In *2nd International Workshop on Managing Technical Debt (MTD)*, pages 17–23, 2011.

<sup>3</sup><https://github.com/aserg-ufmg/sbsi2016-data>