5-2019

An empirical study on discovering a new self-admitted technical debt type - API-debt

Ahmed Aljohani
aha3089@rit.edu

An empirical study on discovering a new self-admitted technical debt type - API-debt

by

Ahmed Aljohani

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Software Engineering

Supervised by
Dr. Mohamed Wiem Mkaouer

Department of Software Engineering
B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York

May 2019

The thesis “An empirical study on discovering a new self-admitted technical debt type - API-debt” by Ahmed Aljohani has been examined and approved by the following Examination Committee:

Dr. Mohamed Wiem Mkaouer
Assistant Professor
Thesis Committee Chair

Dr. Christian Newman
Assistant Professor

Dr. Yasmin El-Glaly
Assistant Professor

Dr. J. Scott Hawker
Associate Professor
Graduate Program Director

Dedication

Dedicated to my family and friends.

Abstract

An empirical study on discovering a new self-admitted technical debt type - API-debt

Ahmed Aljohani

Supervising Professor: Dr. Mohamed Wiem Mkaouer

Self-Admitted Technical Debt (SATD) is when developers intentionally choose to take short-cuts, non-optimal solutions (e.g. temporary fix or rush code development) that negatively contribute to long-term source-code quality in order to achieve short-term goals such as product deadline. Several studies have successfully identified SATD through the source-comments, classified them into five types (design debt, defect debt, documentation debt, requirement debt, and test debt) based on how they negatively affect different parts of the source-code and proposed a tool that automatically detects SATD using the source-comments as input. However, few papers deeply investigate the types of SATD and their effects on software projects. In this paper, we introduce a new type of SATD - we call it API debt - that is related to core API or third-party libraries. In addition, we quantify the amount of API-debt that are found in our selected data-sets, why it is introduced and finally measuring the amount of API-debt removal.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

While building a software project, developers might encounter challenges like tight deadlines, time-to-market, and meeting customer needs. In order to tackle the mentioned challenges, developers tend to produce non-optimal solutions, work-around or shortcuts to the software product. Although these non-optimal solutions speed up the process of delivering the product at the right deadline, it increases the difficulty and impacts negatively on the maintenance of the software in the long run. The non-optimal solutions, work-around or shortcuts are called **" Technical debt "**.

Technical debt (TD) is a term that is used to express non-optimal or workaround solutions. It occurs in software projects to achieve short-term goals that can vary between time-to-market or meeting the customer needs [13].

The previous study by [9] proved that technical debt impacts the software quality in terms of the high amount of *code re-work*, *software errors*, and *bugs*. In other words, technical debt can degrade the overall quality of the software.

Technical debt can be introduced in a software project deliberately or non-deliberately by developers [9]. Non-deliberate TD is when a developer unintentionally introduces technical debt to the source-code; on the other hands, TD can also deliberately take place in the source-code where the developer intentionally incorporate short-cuts and bad practices. This type of technical debt is called Self-Admitted Technical Debt (SATD) which is the domain of our study.

The paper elaborates on a process to detect SATD, as introduced in [9] that describe a process to detect SATD via the source-comments. The paper also identifies the amount of

SATD found in studied projects, and the reason of SATD existence as well as quantifying the amount of removed SATD in the projects under consideration. The paper elaborates on a process to detect SATD by using textual terms and phrases that indicate whether source-comments contain SATD. A list of the terms and phases are cited in [4].

Another study by [7] categorized self-admitted technical debt into five types as design debt, defect debt, documentation debt, requirement debt and test debt each impacting a particular aspect of the software. For instance, test debt is a self-admitted technical debt that indicates a code portion is not thoroughly tested or poorly tested. Therefore, this type of comment shows the need for testing. Their findings show that the design debt is the most occurring SATD type with **42%** to **84%** against the other types.

While investigating SATD types that are proposed by [7], we observed that - based on our best knowledge - SATD types can extend to have a new type that is related to **Application programming interface (API)** and **third-party libraries** issues. As the SATD types have an impact on some aspects of the software, we believe that certain SATD comments could indicate to APIs issues.

Our motive for introducing a new type of SATD is because in the previous study [12] 93.3% of large-number of projects on GitHub rely on API and third-party libraries. Indeed, APIs facilitate and speed up the process of projects creation. Even though these libraries are often reliable and tested, in many cases, they could be misused, or they do not correctly fit to the project and its functionality. However, since the developer tends to introduce low-quality solutions to achieve short-term goals, these solutions could include or related to core APIs and third-party libraries that need to be detected and classified as a new type of SATD. Moreover, based on our literature review about Self-admitted technical debt, studying SATD in terms of APIs is the untouched territory.

Therefore, in this study, we will investigate the existence of a new type of SATD - we call it Self-admitted API-debt, how wide-spread it is, misused and whether or not it is removed.

Our contributions of this study can be summarized as follow:

- Identify and quantify a new type of SATD - API-debt.
- Provide data-set that contains API-debt comments.
- Investigating the introduction and the removal of API-debt.

The paper structure is as follow: in chapter 2 we present a background of Self-admitted technical debt types and their effect upon software projects. Chapter 3 is a light literature review on SATD and APIs. In chapter 4, we introduced relevant related studies about SATD identification, detection, and consequence of SATD . The methodology of our study is presented in chapter 5, and the findings of our methodology are explained in chapter 6. Lastly, we summarized the limitation of our study and the conclusion of the paper in chapter 7 and chapter 8 respectively.

Chapter 2

Background

2.1 Self-admitted technical debt

Self-admitted technical debt can be identified through the source code-comments which are written by the developers indicating a design problem, code with unexpected behavior or a code that is hard to test and change. This is called self-admitted technical debt where the developers deliberately acknowledge one of the aforementioned situations. Prior work by [7] identified five types of self-admitted technical debts which are (design debt, test debt, defect debt, document debt, and requirement debt). Below, the five types of self-admitted technical debt comments along with their explanations and examples.

- **Self-admitted design debt:** Is a comment that indicates poor code design including code smells, duplicated code, lack of abstraction, complex code or a temporary implementation. Fig 2.1 is an example of self-admitted technical debt comments that indicate a design problem in the source-code.

- **Self-admitted defect debt:** When a developer or an author of the code, mentions some unexpected behavior of the code or explicitly mentioned there is a bug that needs to be fixed. Fig 2.2, shows defect-debt comments that clearly mention a bug in a method and unexpected output from using OutputStream version.

- **Self-admitted documentation debt:** In Fig 2.3, some of the written code are not appropriately documented for some reasons and they need to be documented for better understanding of what each method or class purpose.

- **Self-admitted requirement debt:** Is a comment that indicates a portion of the code (method or class) is in-complete or missing some functionalities that need to be added to the source-code artifact. For instance, Fig 2.4 express some missing methods that need to be implemented for certain classes.

- **Self-admitted test debt:** Comments that indicate a lack of a proper or a lot of testing to the code. Some of the code portions are not completely tested or poorly tested. Therefore this type of comment shows the need for testing. Fig 2.5, express the need for having either more test the code or conducting a proper test to the code.

"TODO: - This method is too complex, lets break it up" - [from ArgoUml]

"/ TODO: really should be a separate class */"* - [from ArgoUml]

Figure 2.1: An example of design-debt comments [7]

"// Bug in above method" - [from Apache Jmeter]

"// WARNING: the OutputStream version of this doesn't work!" - [from ArgoUml]

Figure 2.2: An example of defect-debt comments [7]

*"**FIXME** This function needs documentation"* - [from Columba]

"// TODO Document the reason for this" - [from Apache Jmeter]

Figure 2.3: An example of documentation-debt comments [7]

"/TODO no methods yet for getClassname" - [from Apache Ant]

"//TODO no method for newInstance using a reverse-classloader" - [from Apache Ant]

Figure 2.4: An example of requirement-debt comments [7]

"// TODO - need a lot more tests" - [from Apache Jmeter]

"//TODO enable some proper tests!!" - [from Apache Jmeter]

Figure 2.5: An example of test-debt comments [7]

Chapter 3

Literature survey on SATD and APIs

In this section, we conducted a literature survey to investigate the current state of self-admitted technical debt that is related to core APIs and third-party libraries. We selected *five electronic databases* suggested by [2] to perform our *search string* - in Fig.3 - that narrows down the number of relevant papers on self-admitted technical debt.

After applying the search terms, we retrieved **244** potential papers about SATD as shown in table 3.1. where the table lists the online databases that we used to apply our search phrases along with the initial number of papers from each database.

```
**Search String**
( SATD
OR self admitted
OR self-admitted
OR self admitted technical debt
OR self-admitted technical debt
OR technical debt )
```

Figure 3.1: Search String

However, the initial pool of 244 research papers is not fully related to self-admitted technical debt. Meaning that there are some papers that are absolutely out of the scope of SATD topics due to the accuracy of the search-engines of the five online databases we selected. To filter the initial pool from such papers, we conducted manual filtering based

Table 3.1: No. of related papers on self-admitted technical debt per database

Database	Search result
IEEE	213
ACM	6
ScienceDirect	1
Springer	23
ISI	1
Total	244

Table 3.2: literature survey summary after the manual filtering.

DB	No. of papers on SATD	No. of papers on SATD and APIs
IEEE	6	None
ACM	4	None
ScienceDirect	1	None
Springer	1	None
ISI	1	None

on reading each **paper's title** and **abstract**, and extracted papers that are relevant to SATD.

AS a result of the filtering process, we came up with **13** out of 244 papers that are strongly related to SATD topics that including but not limited to identifying, detecting and classifying self-admitted technical debt.

As far as our knowledge and our literature survey, we did not find any study addressing self-admitted technical debt and APIs. Table 3.2. presents a summary of our literature survey.

The first column of the table shows the database names that we used to insert our search string, and in the second column, we presents the number of papers in self-admitted technical debt after the manual filtering. In the third column, we present the number of papers that address self-admitted technical debt and API.

It is clearly shown that the study of self-admitted technical debt on core APIs and third-party libraries is still unexplored topic and it needs further investigation.

Chapter 4

Related Work

Our work focus on three different areas which are the source code comment, self-admitted technical debt, and Android API. Therefore, the related work is divided upon the aforementioned areas.

4.1 Source code comments usage

Source code comment is an essential component of the software where it helps developers to manage their code easily and understand their implementation. Takang et al.[11] and Lawrie et al. [5] empirically investigated the use and the quality of source code comments and how it would affect the code understandably and the software maintainability. The common finding of both studies is that commented code is more understandable and maintainable that non-commented code. For instance, Storey et al.[10] investigate a set of comment annotations that support developers to manage their code along with the team tasks. Annotations such as (TODO, FIXME) in the code not only helps engineers to manage their tasks but also can be a way of communications among developers. Yet, these annotations are used as well to identify comments that have self-admitted technical debt.

4.2 Self-admitted technical debt detection and identification

Technical debt an approach to apply non-optimal or workaround solution to achieve short-term goal of the project. Self-admitted technical debt is a form of technical debt where

developers intentionally mentioned their short-cut solutions via source code comments. Several papers investigated different approaches to identifying and detecting Self-admitted technical debt (SATD).

In 2014, Potdar et al. [9] use the source comments to identify Self-admitted technical debt along with the amount of SATD in most used software projects, the reason for their introduction and removal. One of their contributions is that they identify common patterns that indicate SATD. They manually read 101.762K code comments to find patterns that identify SATD. As a result, they identified 62 common patterns/phrases that indicate SATD.

Similar work conducted by Bavota et al. [1] where they conducted a differentiated replication work that investigates SATD diffusion in software projects, how it does evolve in software projects as well as whether low-quality modules are more exposed to SATD. They found that code debt is most speared debt in software projects than the other types. SATD increases throughout all the projects by 57% due to adding new instance/ feature to the code and low-quality modules are not necessary to have SATD.

Other studies showed that self-admitted technical debt has several types and each type has its own impact and spread. Maldonado et al. [7] manually examined 60K comments to detect and quantify SATD in order to investigate different types of SATD. Their findings show that SATD is common in most software projects with five different types namely: design debt, defect debt, documentation debt, requirement debt and test debt. The most common debt that is design debt with 42% to 84% out of the 60K comments.

In 2017, Maldonado et al. [3] contributed to his previous work [7] by identifying self-admitted technical debt automatically using Natural Language Processing (NLP) classifier. Their study showed that NLP classier achieved 90% accuracy in identifying self-admitted technical debt. In this study, NLP classifier will be used to determine files that have self-admitted technical debt instead of the common patterns that proposed by Potdar et al. [9].

Chapter 5

Methodology

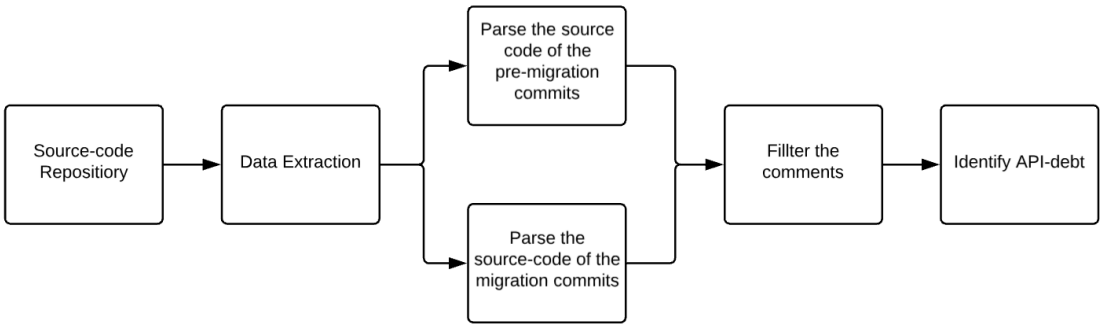

Figure 5.1: Approach Overview.

The purpose of this study is to investigate a new type of self-admitted technical debt that is mainly related to either a core API or a third-party API library. We called this type a self-admitted API debt. We narrowed down our research to understand the nature of self-admitted API debt, determining the amount of API debt in large open-source projects, why it is introduced and determining the amount of self-admitted API debt that is removed after it is introduced. To achieve the goal of this research, we formalized three research questions as follows:

- **RQ1: Do self-admitted API-debt exist in the selected software projects?**
- **RQ2: Why is self-admitted API-debt introduced to the source-code?**
- **RQ3: How much self-admitted API-debt instances are removed after they exist?**

In order to answer our research questions, we established an overall approach as shown in Fig.5.1. The overall approach starts with mining the source-code repository of 55 open-source Java projects as well as extracting all the migration commits IDs for each project. After that, we extracted the source-code-comments in two phases: *1) getting all the source-comments of pre-migration commits. 2) getting all the source-comments at the migration commits.*

Once extracting all the source-comments, we leveraged a tool called "SATD-detector" that is proposed by [3] to automatically detect self-admitted technical debt comments. The tool takes the regular source-comments as an input and returns self-admitted technical debt comments that will be used to manually investigate API-debt comments.

5.1 Data extraction

To perform our study, we used two large data-sets of source-comments. The first data-set contains the latest public release of 55 open-source Java projects provided by [8]. These projects are randomly selected out of 300 projects along with their migration commits IDs. The selection process is based on shuffling, once a project gets selected, we shuffle the provided list of projects again to select another project and keep repeating this until we have 55 projects selected. After that, we extracted all the migration commits unique IDs for each software project. The unique commit ID is the hash code of the commit that allows the developers to keep a record of when and what changes they have made. This allows us to check-out each project version before and at the migration commit. The reason for using all the migration commits for a project is to understand the amount of API-debt comments (RQ1) along with determining the amount of API-debt removal (RQ3). It is also important to highlight that the commit usually contains a commit message which is a brief description of what and why changes were made. The commits messages of the projects help us to understand the reason behind why developers produced API-debts in the source-code (RQ2).

Only the projects written in Java programming language were selected due to our parser

Table 5.1: Analyzed projects information.

Total number of software projects	Total number of Java files	Total number of migration commits
55	11,171	8,065

Table 5.2: Samples of SATD comments and their classification from SATD data-set

Project name	**classification**	**SATD comment**
apache-ant-1.7.0	DEFECT	//FIXME formatters are not thread-safe
apache-ant-1.7.0	DESIGN	//XXX could perhaps also call thread.stop(); not sure if anyone cares
ArgoUML	TEST	//We tested this above - do we need to test again?
ArgoUML	IMPLEMENTATION	//TODO implementation?
apache-ant-1.7.0	WITHOUT CLASSIFICATION	//Success! The XML-commons resolver library is available, so use it.

tool's capability of only parsing Java files. These projects are well-documented by the source-comments, and they have a significant development history in terms of the number of migration commits - 8,065 migration commits. As for the projects files and components, we focused only on extracting Java files due to the high number of source-comments they have, and thus, files such as (CSS, JS, and HTML) were excluded since they do not have a significant amount of source-comments. Table 5.1 shows a brief description of our projects characteristics. From the 55 software projects we extracted 11,171 Java files along with 8,065 migration commit IDs.

To support our study on API-debt, we also leveraged a second data-set that is proposed by [7]. The data-set contains roughly 60K SATD comments that are manually detected as self-admitted technical debt comments. In table 5.3. We removed the duplicate SATD comments from 60K to 37,557 SATD comments to facilitate the manual analysis of our study. This data-set is not only reliable to use in our study but also each SATD comment

Table 5.3: No. of SATD comments in the SATD data-set

Total number of SATD comments	Total number of non-duplicate SATD comments
60 K	37,557

in that data-set has been classified into five different types (design debt, requirement debt, test debt, defect debt, documentation debt) as discussed in chapter 2.

The classification of SATD will help us to understand which type will API-debt falls in and whether or not the prior work was able to classify some SATD comments as API-debt. An example of the content of the published data-set in Table 5.2. The first column of the table is the software project's name, the type of SATD comment is listed in the second column, and lastly, the actual SATD comments that is written by the developer. We refer to this data-set as "SATD data-set".

5.2 Scanning and parsing the code

After obtaining the Java source-code of the 55 software projects, we parsed them using a Java parser tool to extract all the source-comments. We also used SATD-detector tool to detect comments that are self-admitted technical debt. Both tools work hand in hand where the Java parser tool parses the Java files, extracts all the source-comments and feeds it to SATD-detector in which it determines whether or not a comment is a self-admitted technical debt. In Fig 5.3, we present the output of using the mentioned tools together. The tools not only extract the source-comments but also provide other details such as the source-comment type (inline or block comments), the file's name and the path of the java source-code, and most importantly whether or not a comment is self-admitted technical debt.

The parsing process has two phases in order to address RQ1 and RQ3. The first phase focused on parsing the source-code of the pre-migration commits for each projects. On the other hands, phase two focuses on parsing the source-code of the migration commits.

Table 5.4: Overview of SATD detection.

Parsing phase	#comments	#TD detected by SATD detector	#non-duplicate TD
Pre-migration commits	1,297,345	4,771	1,529
Migration commits	1,290,718	5,711	1,396

To simplify the parsing process, in Fig 5.2. we labeled the project version right before the migration commit as **C1**, and we labeled the project version of the migration commit as **C2**. In order to answer RQ1, we parsed all the source-comments of pre-migration commits(**C1**), and to answer RQ3, we parsed all the source-code of the migration commits(**C2**).

The results of the two parsing phases are listed in table 5.3. The parsing information table shows **1,297,345** and **1,290,718** of raw source-comments that we found in both pre-migration, and migration commits respectively. The total number of SATD comments of the pre-migration commits is 1,529 SATD comments, while we got 1,396 SATD comments from the migration commits. We used **1,529** SATD comments for answering RQ1 and we used **1,396** SATD comments to answer RQ3.

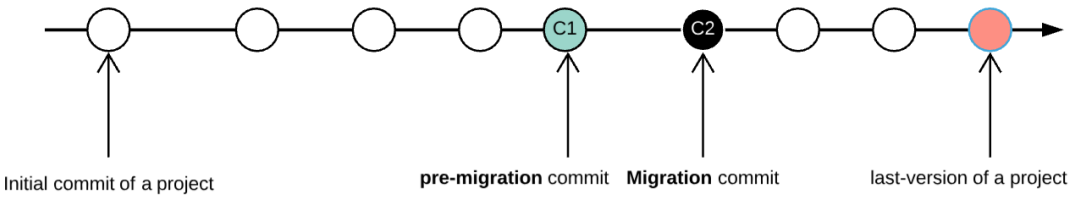

Figure 5.2: Commits processing for the experimental setup of RQ1 and RQ3.

App	FilePath	RelativeFilePath	FileName	IsSATD	CommentType	CommentedNodeType	Comment
wicket	/Users/ahme	Clone/wicket/jdk-1.5	EncodingTest.java	FALSE	JavaDoc	ClassOrInterfaceDeclaration	" The javascript 'history' variable is not supported by"
wicket	/Users/ahme	Clone/wicket/jdk-1.5	EncodingTest.java	FALSE	JavaDoc	MethodDeclaration	" httpunit and we don't want httpunit to throw an"
wicket	/Users/ahme	Clone/wicket/jdk-1.5	EncodingTest.java	FALSE	JavaDoc	ConstructorDeclaration	" exception just because they can not handle it."
wicket	/Users/ahme	Clone/wicket/jdk-1.5	EncodingTest.java	FALSE	JavaDoc	MethodDeclaration	" assertXPath(//SPAN , Hello World!);"
wicket	/Users/ahme	Clone/wicket/jdk-1.5	AllTests.java	FALSE	JavaDoc	ClassOrInterfaceDeclaration	" assertWicketIdTagText(message , Hello World);"

Figure 5.3: Example of the processed commit output.

5.2.1 Parse the source-code of the pre-migration commits

Parsing the source-code of the pre-migration commit is the first parsing phase we did in our methodology. We got **1,529** SATD comments as a final result of this parsing phase that are used to manually investigate API-debt comments.

5.2.2 Parse the source-code of the migration commit

Parsing the source-code of the migration commits is required to understand whether or not the introduced API-debt comments are removed. We used the final result of this parsing phase which is **1,396** SATD comments. These comments are used to see whether the API-debt comments that are found in the first parsing phase are still present after the migration commits take place. This phase of parsing gives us the ability to track the API-debt comments present in SATD comments that are found before and on the migration comments.

5.3 Identify self-admitted technical debt and API debt

The domain of our research is to investigate a new type of SATD, and hence we only focused on comments that are considered as self-admitted technical debt comments.

5.3.1 Identifying self-admitted technical debt

In prior studies [1] [9], identification of self-admitted technical debt was done manually by going through each comment and applying textual SATD phrases that are proposed by [9] to capture whether a comment is self-admitted technical debt or not.

However, [3] proposed a tool called SATD-detector which automatically detect SATD comments using machine learning algorithms. The SATD-detector takes the source-comments as input and returns whether or not the comment is SATD.

Since our empirical study focuses only on SATD comments to find a new type of self-admitted technical debt, we leveraged the SATD-detector with our Java parser tool, so once the parser extracts the source-comments of a project, it uses the project comments as an

input to the SATD-detector.

5.3.2 Identifying API-debt

Identifying API-debt using 55 projects: In total, we went through **1,529** SATD comments as a result of parsing the source-code before the migration commits occur. The process of investigating whether a SATD comment is API-debt is by looking for any indication of core APIs or third-party libraries issues, API upgrade or downgrade. Meaning that, we manually search for SATD comments that indicate to any API issues such as a bug introduced by an API, an API upgrade or API downgrade.

Identifying API-debt using SATD data-set: To further investigate API-debt, we used a SATD data-set from prior work by Maldon et. el [7]. The data-set has 60K source-comments detected manually as Self-admitted technical debt. However, the main contribution of Maldon et. el is classifying SATD comments into five categorize upon their negative affect on the source-code artifact. The process of their manual classification is conducted by going through each comment and labeling it either (design, defect, test, document or without classification). For instance, if a comment is a design debt, it will be labeled ”design”, whereas if a comment is not one of the five types, it will be classified as ”without classification”. Meaning that, the comment is not related to one of their defined types, yet it could be an API-debt.

As for identifying API-debt using SATD data-set, since all the comments in that data-set are a self-admitted technical debt, we did not use the SATD-detector for that. Instead, we removed the duplicate SATD comments from 60K to 37,557K SATD comments. After that, we read through **37,557** SATD comments to find any evidence of API-debt along with the classification that is provided by [7].

Chapter 6

Experimental Results

In this chapter, we present the results of our empirical study on self-admitted API debt along with other aspects such as the amount of API-debt that exit in data-sets, the reasoning of introducing the API-debt and finally the amount of API-debt comments that are removed after they were introduced. For each research question, we provide statistical results as well as graphical presentations that provide more details.

For each research question, we present the motivation, the approach of answering the research question and lastly the findings.

6.1 RQ1: Do self-admitted API-debt exist in the selected software projects?

6.1.1 Motivation:

Recently, in [7], the paper classified self-admitted technical debt into different types and each type of self-admitted technical debt is associated to a particular software issue. Their finding shows that **design-debt** is the most common debt found with *42%* out of the 60K SATD comments. An example of self-admitted technical debt types shown in chapter 2.

The domain of this paper is to investigate self-admitted technical debt that is related to core APIs and third-party libraries. Our motive for that is 93.3% of software projects on Github are built using core and third-part libraries [12]. We believe that improper solutions, short-cuts or hacks can be involved using APIs and third-part libraries. In this research question, we want to investigate a new type of self-admitted technical debt that is related

to API issues - we call it API debt. We also investigate the amount of API-debt using two data-sets.

6.1.2 Approach:

To answer RQ1, we performed *two types of manual analysis*. **First,** we measured how common API-debt exists in our selected software projects. **Second,** we investigate the amount of API-debt using the published data-set that contains 60K SATD comments along with their types as explained earlier in chapter 5.

The first manual analysis addresses whether or not API-debt exists in our selected projects by manually going through 1,529 SATD comments that resulted from the pre-migration commits parsing process. Once we detect API-debt comments, we compute the disruption of API-debt along with an absolute number of API-debt instances.

For the second manual analysis, the same approach was applied, where we read through 37,589 SATD comments along with SATD classifications. Once we detect API-debt comments in that data-set, we measured how API-debt is wide-spread, grouped by the SATD types/classification.

6.1.3 Result:

The results of our manual analysis on API-debt comments are listed in table 6.1. The table shows the number of self-admitted technical debt comments and the number of API-debts that we found in both mentioned data-sets. Our findings show that 266 out of 1,529 and 331 out of 37,101 comments captured as API-debt. Table 6.4 lists some of API-debt comments found in both data-sets.

Since we preformed two types of manual analysis, we split the result of RQ1 as follows:

Result - API-debt in 55 software projects: The result of using the 1,529 SATD source-comments from 55 projects shows that 14.8% of SATD comments are actually API-debts as shown in Fig 6.3. The raw number of API-debts that are captured manually is 266 source-comments. The API-debt comments that we found clearly showcase issues related

Table 6.1: No. of API-debt comments found in both data-sets

Comments From	#SATD comments	#API-debt comments
Pre-migration commits	1,529	266
SATD dataset	37,101	331

to APIs that the developers encountered and documented as self-admitted technical debt.

"//TODO: Rewrite to use ArrayList and .add semantics".

The comment above is an example of API-debt that is captured by using the source-comments of our selected projects. It is clear that the code portion of this comment indicates that a developer asks for rewriting a code portion using one of the core Java APIs (ArrayList) instead of the current proposed solution of that code.

Result - API-debt in SATD data-set:

We used the SATD data-set - as explained in chapter 5 to understand how the prior work classifies comments that considered as API-debt. After going through 37,101 SATD comments, we found 331 comments that can be classified as API-debt which represent 0.9% of the total number of the SATD comments as shown in Fig 6.2.

"//Java API call wasnt successful, lets try some tricks, similar to MR" is a concrete example of a developer that has an issue to call a Java API but the call was not successful due to unknown reason. However, in the same comments the developer mentioned to have a workaround - as it considered a self-admitted technical debt - to solve the issue temporary.

To give an insight into how the prior classified the 331 API-debt comments, we count the number of API-debt and grouped them by the SATD type. Table 6.3 illustrates the raw number of SATD types of the API-debt comments we captured. From Fig 6.1, we noticed that 67.1% of API-debt comments were labeled as "without classification", yet 222 API-debt comments are clearly express self-admitted technical debt that related to API. 91 of the comments were labeled as design debt which represents 27.5% of the total number of the API-debt comments. The reason for that is because the 91 comments express both API issues and design issues in the source code. Finally, 2.1% and 3.3% API-debts instances classified to be a defect and implementation debt respectively.

Table 6.2: API-debt vs SATD types.

Example of API-debt	SATD type
//Java API call wasn't successful, let's try some tricks, similar to MRI	WITHOUT CLASSIFICATION
//needed for SOAP libraries, etc	WITHOUT CLASSIFICATION
//as new versions of java come out, add them to this test	WITHOUT CLASSIFICATION
//Check API for these - it's how CVS does it...	WITHOUT CLASSIFICATION
// remove this when 4.1 plugin API is deprecated	WITHOUT CLASSIFICATION
// WARNING: the OutputStream version of this doesn't work! - tfm	DEFECT
// TODO: Rework to use UML 1.4 TagDefinitions - tfm	IMPLEMENTATION
// remove this when 4.1 plugin is deprecated	WITHOUT CLASSIFICATION

Table 6.3: # API-debt comments and their classification using SATD data-set

No. of API-debt	SATD Types
7	DEFECT
91	DESIGN
11	IMPLEMENTATION
222	WITHOUT CLASSIFICATION
0	TEST
0	DOCUMENTATION
331	Total

6.2 RQ2: Why is self-admitted API-debt introduced to the source-code?

6.2.1 Motivation:

After detecting and quantifying of API-debt in two different data-sets, we want to understand the reason for introducing such API-debt comments. To achieve the goal of understating why developers bring API-debt to software projects, we used the migration commit messages. The migration commit messages are textual text written by the developers to

Table 6.4: Example of API-debt comments from two data-sets

API-debt comment	API-debt source
”TODO add to library org.skife.csv”	55 software projects
” hack to discover jstl libraries”	55 software projects
” TODO: 4alf: migrate it to ExperimentDAO?”	55 software projects
” TODO: Rewrite to use ArrayList and .add semantics: see”	55 software projects
// in UML2, we don’t want to see element imports in profiles	SATD data-set
// add all native jars	SATD data-set
// XXX: we should use JCVS (www.ice.com/JCVS) instead of command line execution so that we don’t rely on having native CVS stuff around (SM)	SATD data-set

resolve API-related issues. Thus, by going through these messages, we got an insight into why API-debt is introduced.

We leveraged 55 open-source java projects that have 8,506 migration commits. We extracted all the commit messages from the 8506 migration commits in order to manually investigate them to get an insight why the API-debt is introduced.

6.2.2 Approach:

Using the **RQ1s** results was not sufficient to understand why API-debt is introduced due to the lack of information in the API-debt comments. Most of API-debt comments do not clearly explain the reasoning of having API-debt, yet some of them do. However, to fully understand why API-debt is introduced is by using the migration commit messages since the primary purpose of these commits is to fix API issues-related. The commit migration messages have a high value to this question in which developers clearly mention the reasons for having the migration commits.

We leveraged 55 open-source java projects that have **8,506 migration commits**. The total number of migration commit messages is **8093**. With removing the duplicate commit messages we come up with **323** migration commit messages as shown in table 6.5. This

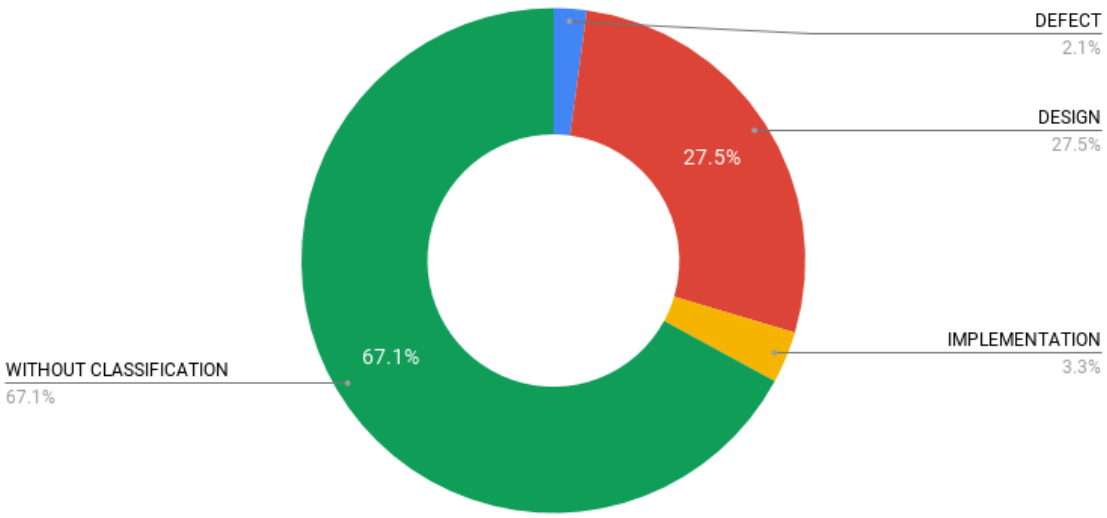

Figure 6.1: The distribution of API-debt comments along with pre-classified SATD types

selection will not affect the investigation of RQ2 since we are computing the disruption in percentage.

Table 6.5: Number of migration commit messages.

commit messages	non-duplicate commit messages
8,065	323

In our analysis of the migration commit messages, we noted that developers do API migration for five reasons. We considered these reason as *factors* of why the developers introduce API-debt comments. We categorized the commit messages into five factors as following:

1. **Better functionality**
2. **Updating the current API**
3. **Fix issues**
4. **Refactoring**
5. **Backwards functionally**

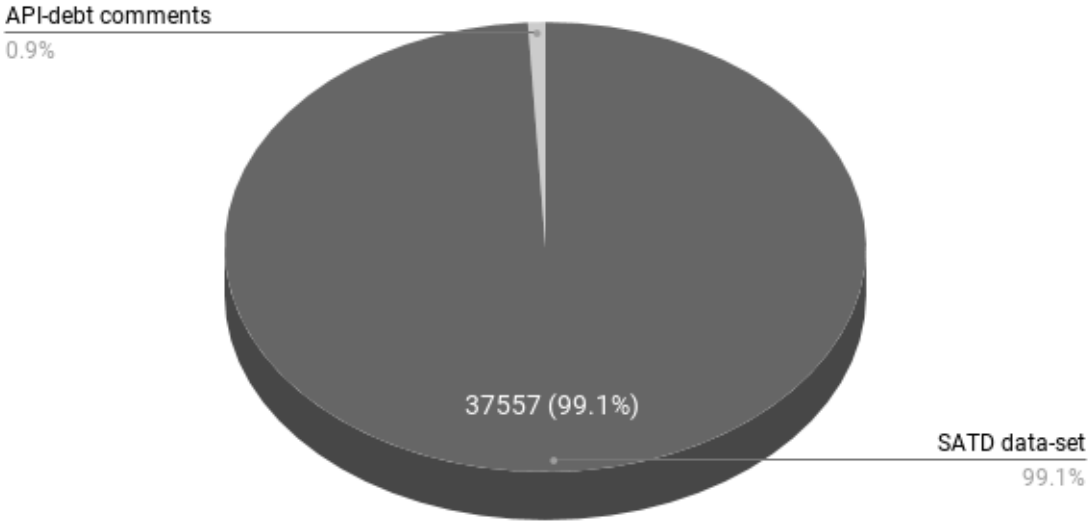

Figure 6.2: Distribution of API-debt using SATD data-set

Table 6.6 shows examples of some of the migration commit messages categorized by one of the five factors.

Table 6.6: Examples of Migration commit messages and Factors.

Factor	Commit Message Example
Better functionality	witched to JUnit over TestNG
Update current API	Update to latest httpclient 4.x
Fix	Fixed a number of bundle bugs
None	????.
Refactoring	massive refactoring
Backward functionality	Fallback to log4j - OpenFire is using it :(

6.2.3 Result:

Fig 6.4 shows that 64.4% of the API-debt comments are introduced due to moving from an old API to a new API. In other words, the majority of developers introduce API-debt to

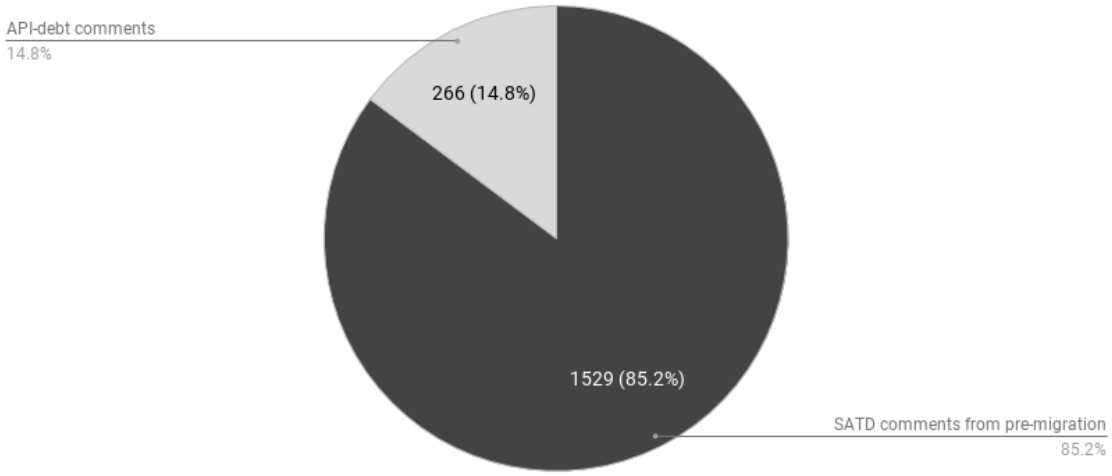

Figure 6.3: Distribution of API-debt using pre-migration commits

find a better functionality compared to the old API. As shown in table 6.6, 208 migration commit out of 323 commits are meant to move from one to another API.

Table 6.7: The total number of each factor

Factor	Number of commit Message
Better functionality	208
Update current API	37
Fix	7
None	65
Refactoring	4
Backward functionality	2

The second obvious reason for having API-debt is updating the current functionality of an API. Meaning that developers could introduce API-debt to look for newly added functionality or upgrade the current API that they have been used. In Fig 6.4, 11.5% of the migration commit took place to upgrade current APIs. Finally, fixing API issues and regular refactoring have the least percentages of why developers introduce API-debt with 2.2% and 1.2% respectively.

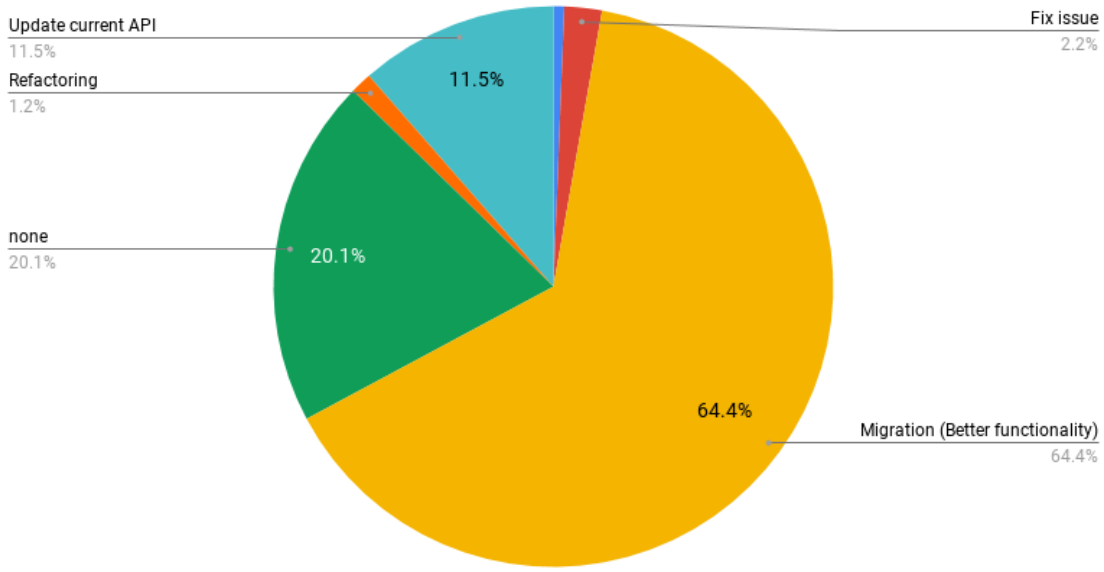

Figure 6.4: The distribution of factors

6.3 RQ3: How much self-admitted API-debt instances are removed after they exist?

6.3.1 Motivation:

Often when the developers present a self-admitted technical debt to the source-code, it meant to be fixed or removed later over the course of the project. Measuring how much SATD comments are removed has been studied in [6]. However, since we introduced a new type of self-admitted technical debt - API debt, we want to investigate how much API-debt comments are actually removed after the developers push the migration commits.

6.3.2 Approach:

The aim of the previous study [6] is to examine the amount of self-admitted technical debt that was removed in the code along with who removed it, and what type of activities that lead self-admitted technical debt to be removed. In other words, the paper identifies all the SATD comments in the first release of a project. Then, tracks each SATD comment in the

Table 6.8: The total number of API-debt and SATD comments

API-debt comments	SATD comments of migration commits
266	1,396

future releases of that project. Once that comment removed, they count that as self-admitted technical debt removal. This process is basically tracking each SATD comment from the first release of a project until the SATD comment disappeared or deleted by the developer. However the process of measuring the SATD removal differs from our study in which API-debt comments describe API issues that have not been addressed by the developers. We used the migration commits which meant to fix API issues. Therefore, instead of tracking API-debt comments form the first release of each project, we only compute the total number of API-debt comments that is introduced in the pr-migration commits, and removed after the migration commits.

We address this question is by using the source-comments that we obtained from parsing the source-code of the migration commits of the 55 software projects. Plus, the result of RQ1 which is 266 API-debt comments that we found from parsing the pre-migration commits. The total number of SATD comments of parsing the migration commits of all the projects is 1,396 SATD comments. Table 6.8 explains the total number of API-debt comments from RQ1 and the total number of SATD comments from parsing the source-code of the migration commits. We implemented a python script that takes the 266 API-debt comments and searches whether an API-debt comment has been removed from the 1,396 SATD comments. The tool counts the matched and removed comments from the 1,396 SATD comments.

6.3.3 Result:

In table 6.9, we present the total number of the API-debt comments that are removed after the migration commits with the total number of API-debt that remained after the migration commits. We found that 22 of the API-debt comments are removed whereas 244 API-

debt comments remained after the migration. In Fig 6.5, represents the percentage of the API-debt removal. We found that the migration commits contribute by 7.5% of API- debt removal. However, it seems that API-debt comments still exist with 92.5% after the migration commits.

Table 6.9: Number of the removed and remaining API-debt comments

The removed API-debt	The remaining API-debt
22	244

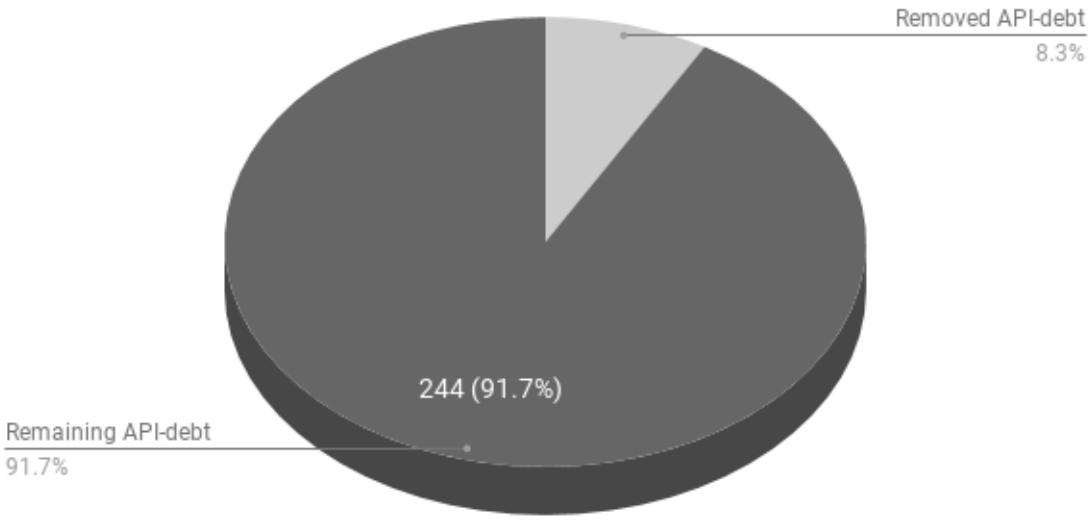

Figure 6.5: Disruption of API-debt removal

Chapter 7

Threats to Validity

There are threats to the validity of this study that we elaborate as follows:

- The manual investigation of API debt comments might be biased to some extent, however, while reading the self-admitted technical debt comments, we ensure that each comment must clearly be related to either a core or third-part API. Otherwise, we exclude the comment.

- We used the SATD detector tool to automatically detect SATD comments. The tool might skip some of the self-admitted technical debt comments while conducting our experiment. However, the tool can achieve 90% of detection performance.

- The python script to understand the removal of API debt, might not be the best solution to track the API comments comparing by the other studies. However, to ensure that some of API-debt comments are not missed, we examine the script by matching the comments in two ways. The first is by converting all the comments into lower case, and then we used natural language processing (NLP) library to remove all the stop words. The second approach is using the source-comments without any modification on the content of the source-comments. The result of both ways were identical.

- The literature review depends on the search keywords in chapter 3. Our search string might miss some important papers that could be related to technical debt and APIs. However, to make sure that we did not ignore some potential papers on SATD, we

used snowballing technique to limit the chance of missing such important papers on self-admitted technical debt.

Chapter 8

Conclusion

Sometimes developers commit code that is incomplete, introduces bugs, needs rework or consider as a temporal solution. These temporary workarounds are usually referred to as technical debt. Technical debt can also be introduced deliberately by the developer due to multiple reasons such as title deadline or time-to-market. This spacial case of technical debt is known as Self-admitted technical debt (SATD). SATD is when the developers intentionally commit code that has hacks, workaround solutions to the project. By using the source-comments, the developer can indicate that a portion of the code is SATD. Priory studies were able to detect and classify SATD into 5 different types. However, in our study, we introduce a new type of SATD that related to API issues and we call it API-debt. We two different data-sets and 323 migration commit to determine how much API-debt are existed, why it is introduced and how much of API-debt comments are removed. Our findings show that API-debt comments appear by 14.8% using 55 software projects and 0.9% using a published data-set. API-debt comments are introduced mainly due to moving from an old API to a new one and 8.6% of the API-debt comments are removed after the migration commit take place.

Bibliography

[1] Gabriele Bavota and Barbara Russo. A large-scale empirical study on self-admitted technical debt. In *Mining Software Repositories (MSR), 2016 IEEE/ACM 13th Working Conference on*, pages 315–326. IEEE, 2016.

[2] Lianping Chen, Muhammad Ali Babar, and He Zhang. Towards an evidence-based understanding of electronic data sources. 2010.

[3] Everton da Silva Maldonado, Emad Shihab, and Nikolaos Tsantalis. Using natural language processing to automatically detect self-admitted technical debt. *IEEE Transactions on Software Engineering*, 43(11):1044–1062, 2017.

[4] Qiao Huang, Emad Shihab, Xin Xia, David Lo, and Shanping Li. Identifying self-admitted technical debt in open source projects using text mining. *Empirical Software Engineering*, 23(1):418–451, 2018.

[5] Dawn J Lawrie, Henry Feild, and David Binkley. Leveraged quality assessment using information retrieval techniques. In *Program Comprehension, 2006. ICPC 2006. 14th IEEE International Conference on*, pages 149–158. IEEE, 2006.

[6] Everton da S Maldonado, Rabe Abdalkareem, Emad Shihab, and Alexander Serebrenik. An empirical study on the removal of self-admitted technical debt. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 238–248. IEEE, 2017.

[7] Everton da S Maldonado and Emad Shihab. Detecting and quantifying different types of self-admitted technical debt. In *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*, pages 9–15. IEEE, 2015.

[8] Hussein Alrubaye Mohamed Wiem Mkaouer and Ali Ouni. On the use of information retrieval to automate the detection of third-party java library migration at the function level. In *27th IEEE/ACM International Conference on Program Comprehension 2019*. IEEE, 2019.

[9] Aniket Potdar and Emad Shihab. An exploratory study on self-admitted technical debt. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pages 91–100. IEEE, 2014.

[10] Margaret-Anne Storey, Jody Ryall, R Ian Bull, Del Myers, and Janice Singer. Todo or to bug. In *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*, pages 251–260. IEEE, 2008.

[11] Armstrong A Takang, Penny A Grubb, and Robert D Macredie. The effects of comments and identifier names on program comprehensibility: an experimental investigation. *J. Prog. Lang.*, 4(3):143–167, 1996.

[12] Ferdian Thung. Api recommendation system for software development. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 896–899. IEEE, 2016.

[13] Sultan Wehaibi, Emad Shihab, and Latifa Guerrouj. Examining the impact of self-admitted technical debt on software quality. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 179–188. IEEE, 2016.