Using Analytics to Quantify the Interest of Self-Admitted Technical Debt

Yasutaka Kamei†, Everton Maldonado††, Emad Shihab††, and Naoyasu Ubayashi†
†Principles of Software Languages Group (POSL), Kyushu University, Fukuoka, Japan
††Department of Computer Science and Software Engineering, Concordia University, Montréal, Canada
Email: †{kamei, ubayashi}@ait.kyushu-u.ac.jp, ††{e_silvam, eshihab}@encs.concordia.ca

***Abstract*—Technical debt refers to the phenomena of taking a shortcut to achieve short term development gain at the cost of increased maintenance effort in the future. The concept of debt, in particular, the cost of debt has not been widely studied. Therefore, the goal of this paper is to determine ways to measure the 'interest' on the debt and use these measures to see how much of the technical debt incurs positive interest, i.e., debt that indeed costs more to pay off in the future. To measure interest, we use the LOC and Fan-In measures. We perform a case study on the Apache JMeter project and find that approximately 42 - 44% of the technical debt incurs positive interest.**

I. Introduction

Technical debt was first coined by Cunningham in 1992 to refer to the phenomena of taking a shortcut to achieve short term development gain at the cost of increased maintenance effort in the future [3]. The technical debt community, organized through the managing technical debt workshop [1], has studied many aspects of technical debt, including its detection [12], impact [11] and the appearance of technical debt in the form of code smells [4]. Most recently, we developed an approach to identify technical debt from code comments, referred to as self-admitted technical debt (SATD). SATD refers to the situation where developers know that the current implementation is not optimal and write comments alerting the inadequacy of the solution.

In the last few years, an increasing amount of work has focused on SATD. In particular, our prior work focused on the detection of SATD [6] and the classification of different types of SATD and the development of datasets to enable future studies on SATD [5]. Other work by Bavota and Russo [2] performed an empirical study of SATD on a large number of Apache projects showed that SATD is prevalent in open source projects, is long lived and is increasing over time. A study by Wehaibi et al. [10] examined the impact of SATD on quality and found that SATD does not necessarily relate to more defects, however, it does make the software system more complex.

Although the metaphor of technical debt has been well studied, to the best of our knowledge, the cost of debt/interest has not been extensively studied. Measuring the interest of the technical debt is one of the challenges in the field, since it requires for the detection of the technical debt, the tracking of the debt over time and the development of measures to accurately quantify this debt. Given that SATD allows us to know the exact method the technical debt exists in, we are able to perform fine-grained analysis of the code, which enables us to quantify interest of the debt. In this paper, the interest refers to the additional difficulty in repaying the debt.

We first propose the use of code metrics, in particular the well-known Lines of Code (LOC) and Fan-In, to measure interest. We use LOC since it highly correlates with most code complexity metrics and Fan-In[1] since it allows us to measure how much a piece of code is depended on by other code. Then, we use the developed measure to determine how much of the SATD incurs positive interest. In a case study on the Apache JMeter project, we find that using LOC, 44.2% and using Fan-In 42.2% of the SATD in JMeter incurs positive interest.

The rest of the paper is organized as follows; Section II introduces our approach to quantify interest of SATD. Section III describes a preliminary study using the developed measure. Finally, Section IV draws conclusions and our future work.

II. Approach

To perform our study, we need to determine the SATD in the codebase, locate when the SATD was introduced in the project and when it was later removed. Then, we use our measures of interest, i.e., LOC and Fan-In, to compare the size of the code and the amount of dependence other code had on the TD code in order to quantify interest (Figure 1).

1. SATD Extraction. In order to measure interest of the TD, our first step is to identify where it exists. Since we focus particularly on SATD, we use code comments found in the source code. We extract and parse the source code of JMeter version 2.10. To perform the parsing, we use the JDeodorant tool [9], which allows us to extract a comment and map it to its corresponding method. Then, we apply a series of filters to remove irrelevant comments, e.g., copyright-related comments. Finally, the 2nd author[2] manually classified all comments to determine if they are SATD comments or not and mapped these comments to their respective methods. In this study, we assume that SATD exists in the method where

[1]Understand calculates Fan-In as the number of inputs a function uses plus the number of unique subprograms calling the function [7].

[2]The 2nd author who made the classification has more than 8 years of experience working in the industry as a software engineer, during this time he designed, implemented and maintained several programs using, in particular the Java programming language.

Fig. 1. The overview of our approach.

the comment is identified. Details regarding the dataset and the filtering applied can be found in our earlier work [5].

2. Identifying SATD Introduction and Removal. Since we are interested in measuring the interest, we need to determine the 'change' over time in these SATD methods. For each of the SATD comments identified by us, we use several git commands (e.g., `git log -- <PATH_TO_FILE>` and `git cat-file <SHA1>:<PATH_TO_FILE>`), to trace a comment back to the commit where it was introduced. We perform this task by replaying the history commit-by-commit. Using the same technique, we are also able to detect the removal of SATD. We detect the removal of SATD when we find that the commit is removed or changed.

3. Determining Metrics that Measure Interest. Once we are able to determine the SATD comments and their associated methods, we would like to calculate the interest that is incurred over time (i.e., from the introduction of the technical debt to its removal). To do so, we extracted 16 code metrics using the UNDERSTAND TOOL [8]. In particular, we selected all method-level complexity and size metrics that Understand is able to provide.

The reasons that we focused on complexity and size metrics are: 1) our intuition tells us that if a piece of code is introduced and then becomes more complex, then that is a good proxy for it being more difficult to deal with in the future, i.e., it incurred interest; and 2) prior work has shown that size metrics are typically highly correlated with complexity metrics, hence, we figured using size metrics (if they are highly correlated with complexity metrics in our case) would be an easier alternative to using complexity metrics.

We measured the Spearman correlation between the complexity and size metrics and found that indeed all metrics except Fan-In are highly correlated with LOC. Therefore, we decided to use the LOC metric as a measure of interest. In addition, since Fan-In is an indicator of how much a method is depended on, we decided to also include the Fan-In metric when calculating interest. The intuition being that if a method is depended on lightly when the SATD is introduced and then has many more dependencies in the future, then dealing with this SATD is much more difficult (since many dependencies may be affected). In the end, we settled on using the two metrics, LOC and Fan-In, as measures of interest.

4. Calculating Interest. Using our metrics, we consider the relative LOC and Fan-In values between the introduced and removed versions as interest. We calculate the interest per SATD instance. For example, if arbitrary metric values in the introduced and removed versions are 10 and 20 in the method where the SATD exists, the relative size is 100 (i.e, $100 * \frac{(20-10)}{10}$). In cases where the SATD is not yet removed, we use the numbers from the latest version of JMeter. Our assumption here is that if the SATD incurs positive interest, then it will be more difficult to remove in the future, e.g., if the code becomes more complex compared to when the debt was taken, then it will be more difficult to deal with.

While the paper tackles the research topic that accelerates a new research direction (i.e., quantifying interest of SATD), it also has the weakness of our current approach. We elaborate on the weakness of our current approach in Section IV.

III. INITIAL CASE STUDY

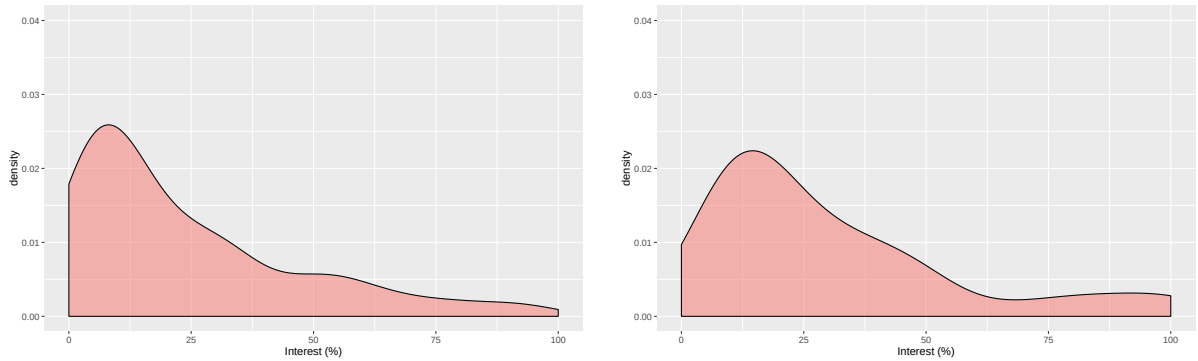
Motivation. There exist several previous studies that focused on understanding SATD (e.g., the detection of technical debt [6, 12] and the impact of SATD on software quality [10]). However, to the best of our knowledge, there are no studies that help in the quantification of SATD interest. Therefore, we would like to know how we can measure interest and if SATD actually incurs positive interest.

Datasets. To conduct our initial case study, we use data from the Apache JMeter open source. We use JMeter since we have used this dataset in the past [5, 6], and know that it cotains instances of SATD and uses Git as the version control system, which many of our tools are designed to work on. In particular, we use release v2.10 of JMeter, which contains 81,307 SLOC in 1,181 classes, contains 20,084 comments, and has 33 unique contributors.

Approach. To calculate interest of SATD, we follow the approach we explained in Section II. We show the number of SATD, the percentage of the technical debt that has positive interest, and the distribution of interest for technical debt that incurs an positive interest rate.

Results. We find that there is a high correlation between LOC and the other product metrics, except Fan-In. From the highly correlated metrics, we selected LOC as the metric to calculate interest, since intuitively it is easier to measure and comprehend. Therefore, we settled on using two product metrics (i.e., LOC and Fan-In) to measure interest.

Table I shows the number of SATD and the percentage of the technical debt that has positive interest in all technical debt. The table shows that 44.2% of technical debt incurs a positive interest rate in terms of LOC and 42.2% of the SATD has it in terms of Fan-In. We can see that in some cases, there can be negative interest (13.8% using LOC and 8.1% using Fan-In), where the SATD method gets smaller or have less Fan-In after the introduction of the SATD. There are also cases



(a) JMeter (LOC) (b) JMeter (Fan-In)

Fig. 2. Distribution of SATD Interest in JMeter. Interest is Measured Using LOC and Fan-In.

TABLE I
THE PERCENTAGE OF SATD THAT HAS POSITIVE, NEGATIVE AND NO CHANGE IN INTEREST

	# SATD instances	Positive	Negative	No Change
LOC	181	44.2%	13.8%	42.0%
Fan-In	161	42.2%	8.1%	49.7%

TABLE II
STATISTICS OF THE SATD THAT INCURS A POSITIVE RATE

	Min.	1st Qu.	Median	3rd Qu.	Max.
LOC	1.6	6.9	18.0	50.0	6667.0
Fan-In	5.6	12.4	25.0	50.0	900.0

where nothing changes in terms of LOC and Fan-In between the SATD introduction and removal. Lastly, it is important to note that there is not large difference between in the amount of positive and no change interest rates using LOC and Fan-In.

Next, we would like to know how high is the positive interest rate. This analysis provides us with more insight about the SATD that incurs a positive interest rate. Table II and Figure 2 show that the distribution of interest for the SATD that incurs a positive rate. We note that we limit the x-axis of Figure 2 to 100% for readability. We see from Figure 2, that the distributions are left-skewed, indicating that the majority of the SATD ranges between 6.9-12.4 and 50.0 in terms of LOC and Fan-In. Our findings clearly indicate that there is SATD that incurs a positive interest rate and different types of SATD have different values of interest, which shows that we should be prioritizing SATD based on its interest, i.e., all SATD is not equal.

44.2% of technical debt incurs a positive rate in terms of LOC and 42.2% of technical debt incurs it in terms of Fan-In.

IV. CONCLUSION

In this paper, we introduced an approach to quantify interest of SATD. Our proposed approach uses software product metrics to lead to measure the interest from software projects. The results of our initial case study using the Apache JMeter project show that 44.2% of technical debt has a positive rate in terms of LOC and 42.2% of technical debt has it in terms of Fan-In.

Future work. This paper only shows an early idea to quantity the interest of SATD. Therefore, there remain many challenges to address in the future.

- To calculate interest, we use the relative size of metric values between two versions of SATD-introduction and removal. However, the period, in time, is not considered to calculate the interest. Therefore, in the future we would like to take the period into account when calculating the interest.
- There are several type of SATD, such as defect and design SATD. The previous study [5] shows that the percentage of SATD varies depending on the type of technical debt and the studied systems. For example, the projects that have limited time to develop features are likely to leave comments of features that need to be implemented in the future. To better understand the interest, in the future we would like to analyze the interest per type of SATD.
- The interest varies among technical debt. If we can understand the reason why some of SATD has larger interest, we can make use of such insights for future development. Therefore, we would like to manually investigate why some of the SATD has larger interest.
- Generally speaking, software systems are always evolving over time for implementing new functionality and fixing defects. Therefore, even if the size of the SATD method increases, it is not clear how best to evaluate the effects on the interest of SATD. We would like to compare the impact of software evolution on methods in the two groups, SATD v.s. non-SATD, to draw a relative comparison that controls for general evolution.

ACKNOWLEDGMENT

This research was partially supported by JSPS KAKENHI Grant Numbers 15H05306.

References

[1] International workshop on managing technical debt (MTD). https://www.sei.cmu.edu/community/td2016/. Accessed: 2016-10-16.

[2] G. Bavota and B. Russo. A large-scale empirical study on self-admitted technical debt. In *Proc. Int'l Conf. on Mining Software Repositories (MSR)*, pages 315–326, 2016.

[3] W. Cunningham. The WyCash portfolio management system. In *Addendum to the Proc. on Object-oriented Programming Systems, Languages, and Applications*, pages 29–30, 1992.

[4] F. Fontana, V. Ferme, and S. Spinelli. Investigating the impact of code smells debt on quality code evaluation. In *Proc. of the Int'l Workshop on Managing Technical Debt (MTD)*, pages 15–22, 2012.

[5] E. D. S. Maldonado and E. Shihab. Detecting and quantifying different types of self-admitted technical debt. In *Proc. of the Int'l Workshop on Managing Technical Debt (MTD)*, pages 9–15, 2015.

[6] A. Potdar and E. Shihab. An exploratory study on self-admitted technical debt. In *Proc. of the Int'l Conf. on Software Maintenance and Evolution (ICSME)*, pages 91–100, 2014.

[7] Scientific Toolworks, Inc. FANIN - Understand 2.6. https://scitools.com/support/metrics_list/?metricGroup=count.

[8] Scientific Toolworks, Inc. Understand 2.6. http://www.scitools.com/.

[9] N. Tsantalis, T. Chaikalis, and A. Chatzigeorgiou. Jdeodorant: Identification and removal of type-checking bad smells. In *Proc. European Conf. on Software Maintenance and Reengineering (CSMR)*, pages 329–331, 2008.

[10] S. Wehaibi, E. Shihab, and L. Guerrouj. Examining the impact of self-admitted technical debt on software quality. In *Proc. of the Int'l Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 179–188, 2016.

[11] N. Zazworka, M. A. Shaw, F. Shull, and C. Seaman. Investigating the impact of design debt on software quality. In *Proc. of the Int'l Workshop on Managing Technical Debt (MTD)*, pages 17–23, 2011.

[12] N. Zazworka, R. O. Spínola, A. Vetro', F. Shull, and C. Seaman. A case study on effectively identifying technical debt. In *Proc. of the Int'l Conf. on Evaluation and Assessment in Software Engineering (EASE)*, pages 42–47, 2013.