15th Annual Conference on Systems Engineering Research
Disciplinary Convergence: Implications for Systems Engineering Research

Eds.: Azad M. Madni, Barry Boehm
Daniel A. Erwin, Roger Ghanem; University of Southern California
Marilee J. Wheaton, The Aerospace Corporation
Redondo Beach, CA, March 23-25, 2017

# An Empirical Study of Technical Debt in Open-Source Software Systems

*Reem Alfayez, Celia Chen, Pooyan Behnamghader, Kamonphop Srisopha, Barry Boehm*
**Center for Systems and Software Engineering, University of Southern California, Los Angeles, USA {alfayez, qianqiac, pbehnamg, srisopha, boehm}@usc.edu**

***Abstract***

***Technical debt (TD) is a term coined by agile software pioneer Ward Cunningham to account for the added software-system effort or cost resulting from taking early software project shortcuts. The debt metaphor reflects that debt accumulates interest: the later it is paid, the more it costs. The TD concept has achieved extensive visibility and usage in the software field, but it applies at least as strongly to cyber-physical systems. In researching the TD phenomena, we have found that open-source software projects are particularly good subjects, as they keep records of the timing, content, and rationale for each update. In this paper, we concentrate on the analysis of open source software projects to evaluate the relationships between multiple software system characteristics and TD and the relationships between software process factors and TD.***

***Keywords: Open source software; software domain; software metrics; software quality; software size; technical debt***

## 1. Introduction

Technical debt (TD) was defined by Cunningham [1] as the added software-system effort or cost resulting from taking early software project shortcuts or when short-term needs are addressed first. As with financial debt, it accumulates interest: the later it is paid, the more it costs. Clearly, the debt metaphor applies at least as strongly to cyber-physical systems, with even greater impact if the recovery involves millions of automobiles or exploding cellphones.

There are more and less responsible ways of incurring technical debt. The more responsible ways to treat technical debt as an investment are to enable more rapid delivery in time-critical situations, such as competing to become first-to-market for a new product; rapid fielding of defenses to terrorist threats; or trying to build the riskiest parts first to determine whether continuing the project is feasible. The less-responsible ways often suffer from a lack of foresight, such as in building the easiest parts first, neglecting non-functional requirements, or skimping on systems engineering or maintainability preparation.

A particularly attractive area for researching the causes and effects of incurring TD is in the open-source software area. This form of software development involves distributed individuals or groups of developers participating in concurrently developing and improving software systems, many of which (e.g., Linux, Apache, JBoss, and Tomcat) are reliably used around the world. They are a rich source of consistent data, as they keep records of the timing, change content, and rationale for each upgrade.

In this paper, we perform an empirical analysis study on Apache Java systems to understand how

the system characteristics such as domain and size relate to its TD and TD density, and how system process factors relate to its TD and TD density. Our goal is to get an understanding of the relations between these factors to provide guidance for decision makers and system engineers to incorporate it into their trade-space studies. Section 2 summarizes the research approach and key analysis tools. Section 3 elaborates on the research questions and data collection choices. Section 4 summarizes the data analysis and results. Section 5 presents the threats to validity. Section 6 the related work. Section 7 the conclusions and planned future work. Lastly, section 8 the acknowledgements.

## 2. Background

### 2.1. Technical Debt

*Definition:* Technical debt was originally defined as "not quite right code which we postpone making it right."[2] As the metaphor evolved, it has been used to describe other kind of debts such as test debt, personal debt, design debt, requirement debt, documentation debt, etc. [2,3]

*Measurement and Calculation*: Technical debt is measured using the following software qualities [4]:

- Robustness describes how stable the application is and its ability to recover from failure.
- Performance efficiency measures how the application uses its resources efficiently and how responsive it is.
- Security indicates the system's ability to protect confidential information and preventing unauthorized intrusions.
- Transferability measures the ease with which a new team can understand the software and become productive.
- Changeability measures the software adaptability and modifiability.

Technical debt is calculated using two main components: the principal and the interest [5]. The principal measures the cost or effort for refactoring a quality rule violation: for example, the effort to change a meaningless variable name to a meaningful name. The interest measures the decreased productivity or extra defects occurrence due to violating a software quality rule or not fixing a violated rule.

In this study, we measured technical debt principal using SonarQube (http://www.sonarqube.org/). SonarQube implements the SQALE method, which is the leading-edge method developed by DNV ITGS France to assess technical debt while conforming to the ISO 9126 standard for software quality [6]. SonarQube uses the following formula to measure Technical Debt:

$$\begin{aligned}\text{Debt(in man days)} = {} & \text{cost\_to\_fix\_duplications} \\ & + \text{cost\_to\_fix\_violations} \\ & + \text{cost\_to\_comment\_public\_API} \\ & + \text{cost\_to\_fix\_uncoverd\_complexity} \\ & + \text{cost\_to\_bring\_complexiy\_below\_threshold}\end{aligned}$$

Where:

$$\begin{aligned}Duplications &= cost\_to\_fix\_one\_block * duplicated\_blocks \\ Violations &= \cos t\_to fix\_one\_violation * mandatory\_violations \\ Comments &= \cos t\_to\_comment\_one\_API * public\_undocumented\_api \\ Coverage &= cost\_to\_cover\_one\_of\_complexity \\ &\quad * uncovered\_complexity\_by\_tests\ (\text{80\% of coverage is the objective})\end{aligned}$$

$$Complexity = cost\_to\_split\_a\_method * (function\_complexity\_distribution >= 8) + cost\_to\_split\_a\_class * (class\_complexity\_distribution >= 60)$$

After retrieving technical debt, we calculated technical debt density using the following formula:

$$TechnicalDebtDensity = \frac{TechnicalDebt\ (in\_man\_hours)}{KLOC}$$

**3. Empirical Study Setup**

In order to provide decision makers with guidelines to assist them in the decision making process, we want to understand how different system characteristics and system process factors relate to technical debt. We studied the relations between each factor and the total system technical debt to assess their relations on a system level, and we calculated technical debt density by normalizing the numerator total technical debt with the denominator KLOC in order to evaluate the correlations of the different variables in our study and the quality per capita in our subject systems.

3.1 Research Questions

- **RQ1:** Does source code size relate to the total technical debt and the technical debt density?
- **RQ2:** Do the total technical debt and the technical debt density in software vary among domains?
- **RQ3:** Do system process factors, including number of commits, releases, branches, and contributors relate to the total technical debt and the technical debt density?

3.2 Data Collection

Data was collected from the Apache Software Foundation (http://www.apache.org) and only Java systems were considered for this empirical study since there are sufficient numbers of Java systems from different domains. The six following software domains of Apache systems were selected in this paper:

- Big Data
- Database
- Library
- Network Server
- Web Framework
- XML

Table 1 and 2 show the characteristics of the selected systems in each domain.

| Domains | Number of Systems | Average LOC |
|---|---|---|
| Big Data | 16 | 44992 |
| Database | 13 | 52610 |
| Library | 35 | 113612 |
| Network Server | 9 | 20624 |
| Web Framework | 11 | 31164 |
| XML | 7 | 51569 |

*Table 1. Characteristics of system data source.*

| Domains | [1,999] | [1000,5000] | [5001,10000] | >10000 |
|---|---|---|---|---|
| Big Data | 0 | 1 | 0 | 15 |
| Database | 0 | 0 | 2 | 11 |
| Library | 2 | 11 | 7 | 15 |
| Network Server | 0 | 2 | 3 | 4 |
| Web Framework | 0 | 0 | 2 | 7 |
| XML | 0 | 0 | 3 | 4 |

*Table 2. Classification of number of systems by LOC in each domain*

The data collection process involved establishing and applying consistent criteria for inclusion of well-known systems to ensure the quality of this study. We excluded systems that fall under multiple domains or systems that have empty repositories. Source code under example, sample, and tutorial folders were also excluded. Systems that fall under all of the following criteria were considered:

- There are more than one official releases.
- The latest stable release source code is available.
- The source code has well-established sizing.
- The source code is fully accessible.

3.3 Data Collection Challenges

Different open-source systems have different folder structures. While some systems reconcile with the standard conventions, other systems do not have any clear folder naming schema or folder structure. We overcame this challenge by reading system documentations and manuals, looking into commit logs, and reading the release notes. By taking these steps, we were able to exclude files that are not source code such as examples and tutorials. These steps are necessary to insure that we only measured source code technical debt.

## 4. Data analysis and results

4.1 Evaluation of size hypothesis.

In order to reject the null hypothesis and support the size hypothesis, we clustered the data using various clustering algorithms to examine whether total technical debt and technical debt density differ across different size clusters.

Cluster analysis: we used K-means clustering analysis to group the data into clusters based on the size of systems. K-means clustering does not provide any predicted outcomes, rather the algorithm is

intended to find patterns in the data and cluster them based on their similarity. Here we clustered the data based on the size of systems to examine whether total technical debt and the technical debt density differ significantly among each cluster.

Since K-means clustering requires a specific number of clusters to be generated, we need to know the optimal number of clusters in the dataset. The proposed solution is to first compute a clustering algorithm of interest using different values of clusters k. Next, the Silhouette width is drawn according to the number of clusters. The location of a peak is generally considered as an indicator of the appropriate number of clusters. Two clusters were suggested for both datasets.

Total technical debt clustering results: the results of K -means clustering consist of 2 clusters of size 11 and 80 respectively. Table 3 lists the means of technical debt and means of LOC of each cluster and Figure 1 visually represents the cluster solution.

| Cluster | Technical Debt Mean | LOC Mean |
|---|---|---|
| 1 | 2074.1818 | 120362.64 |
| 2 | 346.2475 | 18070.42 |

*Table 3. Cluster means*

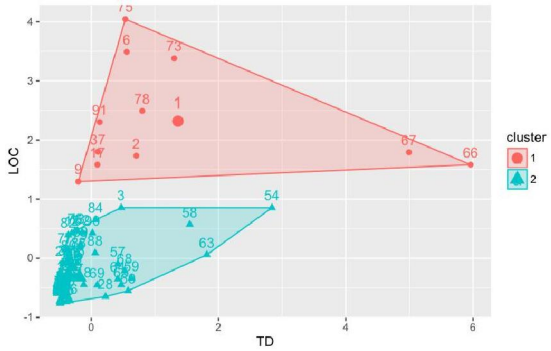


Figure 1. 2D representation of the cluster solution of Technical Debt

Technical debt density clustering results: the results of K -means clustering consist of 2 clusters of size 11 and 80 respectively. Table 4 lists the means of technical debt density and means of LOC of each cluster and Figure 2 visually represents the cluster solution.

| Cluster | Technical Debt Density Mean | LOC Mean |
|---|---|---|
| 1 | 19.48511 | 120362.64 |
| 2 | 20.06992 | 18070.42 |

*Table 4. Cluster means*

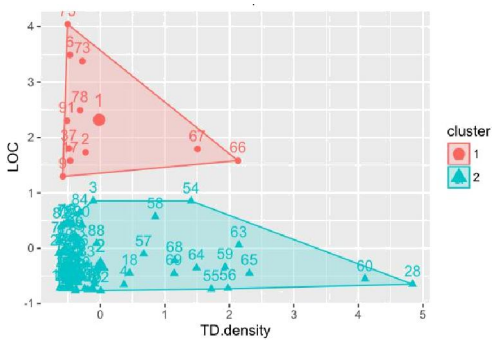


Figure 2. 2D representation of the cluster solution of Technical Debt Density.

Overall, larger systems had more technical debt in total but less technical debt density, while smaller systems had less technical debt in total but higher technical debt density. This may seem like a surprise, as one would expect that larger systems would be harder to understand, leading to higher debt density. However, there may be other explanations, at least for this sample. It could be that the larger projects had more capable teams and a more serious concern with high quality [7], or it could be that the larger projects had a larger proportion of simple, easy to understand components.

4.2 Evaluation of domain hypotheses

In this section, we performed various statistical analyses to examine whether total technical debt and technical debt density differ among different software domains.

*4.2.1 Levene's Test*

In order to perform one-way ANOVA to examine whether technical debt and technical debt density of the six domains differs significantly, we first performed Levene's test to examine whether the variances of the six domains are significantly different. Levene's test result indicates unequal variances ($F = 6.117$, $p = 6.912e{-}05$) for total technical debt and unequal variances ($F = 4.9892$, $p = 4.695e{-}04$) for technical debt density. Since the p-value of Levene's test is much less than 0.05, we concluded that the variances of the six domains are significantly different, thus variances are unequal across domains. This would mean that we violated one of the assumptions of one-way ANOVA. Therefore, we could not perform one-way ANOVA to see if technical debt and technical debt density differ across domains. However, there are a number of ways to deal with unequal variances, such as transformations, robust regression and so on. In this study, we used Welch ANOVA [8].

*4.2.2 Welch ANOVA*

The Welch ANOVA is based on the usual ANOVA F -test. However, the means are weighted by the reciprocal of the group mean variances. Therefore, it is suitable to handle unequal variances. An alpha level of 0.05 was used for all subsequent analyses.

The Welch ANOVA of total technical debt of six different domains reveals a statistically significant difference, Welch's $F(5, 23.508) = 4.2964$, $p = 0.006408$, indicating that not all domains have the same total technical debt. The estimated omega squared ( $\omega 2 = 0.15$) indicates that approximately 15% of the total variation in total technical debt is attributable to differences among the six domains.

The Welch ANOVA of technical debt density of six different domains also reveals a statistically significant difference, Welch's $F(5, 25.47) = 5.2781$, $p = 0.001848$, indicating that not all domains have the same technical debt density. The estimated omega squared ($\omega 2 = 0.19$) indicates that approximately 19% of the total variation in technical debt density is attributable to differences among the six domains.

In conclusion, we have enough evidence to reject the null hypothesis and accept that both technical debt and technical debt density vary across the different domains.

4.2.3 Games-Howell Test

We concluded that at least two of the six domains differ significantly through Welch's ANOVA analysis, however, beyond that, we still couldn't tell the differences between all unique pairwise comparisons. Therefore, we performed a Games-Howell Test to figure out which parings differ significantly.

Post- hoc Games-Howell results indicate that systems in Library and Big Data have significantly different total technical debt at the 0.1 level of significance while other comparisons are not significant. However, more pairings appear to have significantly different technical debt density at the 0.1 level of significance, including XML with Network, Web Framework with Network, Web Framework with Database, Library with Network, Network with Big Data, and Database with Big Data.

The variation of technical debt and technical debt density with different domains might be due to the variation in domains complexity levels, development methods, personnel and specialists [9]. A further work will investigate what factors contribute to technical debt variation more.

4.3 Evaluation of system process factors hypotheses

In this section, we performed Pearson Correlation tests to examine how different process factors relate to technical debt and technical debt density.

In order to understand how system process factors relate to technical debt and technical debt density, we used Pearson correlation to assess the relationships between each factor and the total technical debt and technical debt density. The significance level was set to 0.05, which is equal to a confidence level of 95%. Any p-value that is well below that threshold is concluded as a strong relationship.

Figure 3 shows the correlation between total technical debt and the system process factors. Table 5 lists the correlation coefficients and the significance levels of each system process factors and technical debt. While the results show a strong positive correlation between the number of commits and the total technical debt and the number of releases and the total technical debt, there is no significance correlation between number of contributors and technical debt and number of branches and technical debt.

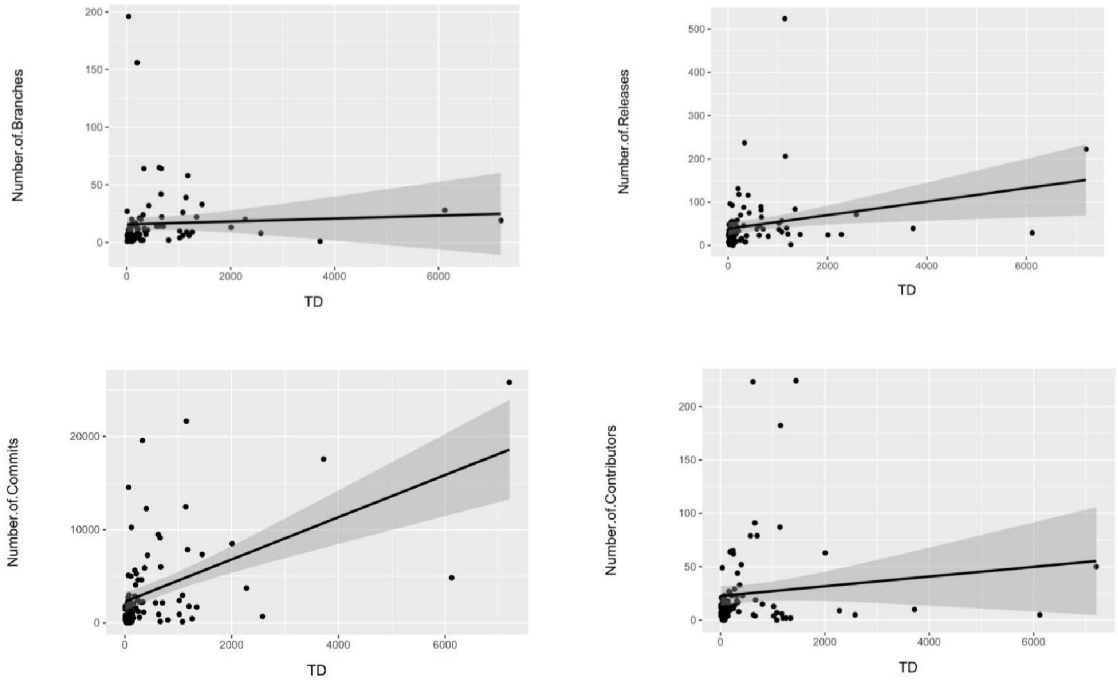


Figure 3. TD and System Process Factors Correlation

| | N | R-value | p-value |
|---|---|---|---|
| Number of branches | 91 | 0.05071 | 0.63308 |
| Number of releases | 91 | 0.26114 | 0.01241 |
| Number of commits | 91 | 0.51619 | 1.63135e-7 |
| Number of contributors | 91 | 0.12618 | 0.23335 |

*Table 5. Correlation coefficients matrix between total technical debt and system process factor*

Figure 4 shows the correlation between technical debt density and the system process factors. Table 6 shows that there is no significance correlation between technical debt density and any of the system process factors. This could be a side effect of the counter-intuitive results above on the correlations between larger and smaller systems and technical debt.

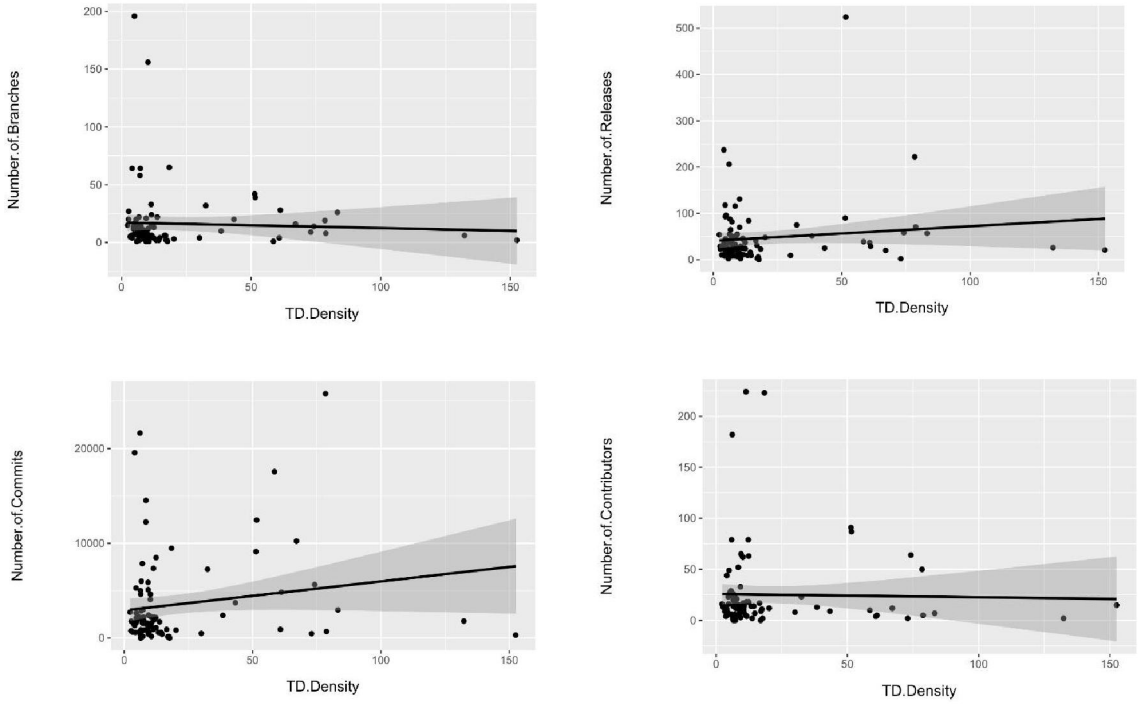


Figure 4. TD Density and System Process Factors Correlation

| | N | R-value | p-value |
|---|---|---|---|
| Number of branches | 91 | -0.04658 | 0.66108 |
| Number of releases | 91 | 0.12826 | 0.22567 |
| Number of commits | 91 | 0.17108 | 0.10493 |
| Number of contributors | 91 | -0.02188 | 0.83688 |

*Table 6. Correlation coefficients matrix between technical debt density and system process factors*

## 5. Threats to validity

The key threats to external validity involve our subject systems. Although we only used ninety-one java systems from six different domains, we selected the most popular systems and domains in one of the leading open-source software communities. The systems also vary along multiple dimensions, such as the number of versions, size, domain, and timeframe. The other domains do not have enough Apache java projects that satisfy the selection criteria of our systems.

Further, we were unable to find sources of data outside of the open source community to test hypotheses about the effects of various forms of closed-source process strategies on technical debt. Some companies are performing such studies, but generally prefer to keep the results private.

The construct validity of our study is mainly threatened by the accuracy of the amount of technical debt. To mitigate this threat, we chose SonarQube which is one of the most trusted tools to calculate technical debt. SonarQube is widely used, and to the best of our knowledge is the only open source tool that implements the SQALE method for evaluating technical debt which is currently the most widely used approach to manage technical debt [6].

## 6. Related Work

Past literatures have been focusing mainly on code smells and their impact on software quality [10, 11]. Our work looks into numerous aspects of software system and the correlation each of them has with technical debt and technical debt density.

Marinescu [12] proposed a framework that can assess technical debt by detecting several design flaws in software systems, for example, specific violations of well -established design principles. Zazworka et al. [13] also focused on the impact of design debt on software quality. They investigated how design debt, in the form of god classes, affects the maintainability and the correctness of a software project. However, Sterling [14] mentioned that technical debt does not include only design debt, but also configuration management debt, quality debt, platform experience debt, etc. In this paper, we look at that other different aspects of software system that could potentially relate to technical debt.

Seaman et al. [15] discuss proposes, benefits of managing technical debt, and several decision making approaches to technical debt. Similarly, our paper aims to help system engineers and decision makers to clearly identify what relates to technical debt and thus supports better management and decision-making with regards to technical debt, again subject to the limitation of our results to open-source software development.

## 7. Conclusions and future work

We employed various statistical methods to investigate how technical debt and technical debt density relate to different system characteristics and process characteristics across a representative sample of ninety-one Apache Java open source projects. From the results of the data analysis on the hypotheses, we can conclude for similar systems that the size of a software system and the software domain it belongs to can correlate with its technical debt and technical debt density significantly. While the number of system releases and commits has a significant positive relationship with its technical debt, the results show no significant relationship between system technical debt and the number of its contributors and branches. In addition, our analyses show no significant relations between any of these system process factors and technical debt density.

In the future, we will further the study to understand the reasons behind these relations by conducting a biopsy analysis on these systems. Our ultimate goal is to provide guidelines for decision makers to help them study the trade space by providing what factors introduce more technical debt to the system and the quality per capita in the systems. We will also continue to search for similar sources of technical debt data outside the open-source community.

## 8. Acknowledgements

This material is based upon work supported in part by the U.S. Department of Defense through the Systems Engineering Research Center (SERC) under Contract H98230-08-D- 0171. SERC is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology. It is also

supported by the National Science Foundation grant CMMI-1408909, Developing a Constructive Logic-Based Theory of Value-Based Systems Engineering. We also acknowledge the support of National Natural Science Foundation of China No. 91318301, 61432001, 91218302.

## References

1. W. Cunningham, “The wycash portfolio management system,” ACM SIGPLAN OOPS Messenger, vol. 4, no. 2, pp. 29–30, 1993.
2. D. Falessi, P. Kruchten, R. L. Nord, and I. Ozkaya, “Technical debt at the crossroads of research and practice: report on the fifth international workshop on managing technical debt,” ACM SIGSOFT Software Engineering Notes, vol. 39, no. 2, pp. 31–33, 2014.
3. P. Kruchten, R. L. Nord, and I. Ozkaya, “Technical debt: From metaphor to theory and practice.” Ieee software, vol. 29, no. 6, 2012.
4. S. McConnell deSCribed, “Estimating the principal of an application’s technical debt,” 2012.
5. D. Falessi and A. Reichel, “Towards an open-source tool for measuring and visualizing the interest of technical debt,” in Managing Technical Debt (MTD), 2015 IEEE 7th International Workshop on. IEEE, 2015, pp. 1–8.
6. J.-L. Letouzey and M. Ilkiewicz, “Managing technical debt with the SQALE method,” IEEE software, vol. 29, no. 6, pp. 44–51, 2012.
7. Z. Jiang, P. Naudé, and B. Jiang, “The effects of software size on development effort and software quality,” International Journal of Computer and Information Science and Engineering, vol. 1, no. 4, pp. 230–234, 2007.
8. B. Welch, “On the comparison of several mean values: an alternative approach,” Biometrika, vol. 38, no. 3/4, pp. 330–336, 1951. 9.C. Jones, “Variations in software development practices,” IEEE software, vol. 20, no. 6, pp. 22–27, 2003.
10. S. Olbrich, D. S. Cruzes, V. Basili, and N. Zazworka, “The evolution and impact of code smells: A case study of two open source systems,” in Proceedings of the 2009 3rd international symposium on empirical software engineering and measurement. IEEE Computer Society, 2009, pp. 390–400.
11. N. Zazworka, R. O. Spínola, A. Vetro, F. Shull, and C. Seaman, “A case study on effectively identifying technical debt,” in Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering. ACM, 2013, pp. 42– 47.
12. R. Marinescu, “Assessing technical debt by identifying design flaws in software systems,” IBM Journal of Research and Development, vol. 56, no. 5, pp. 9–1, 2012.
13. N. Zazworka, M. A. Shaw, F. Shull, and C. Seaman, “Investigating the impact of design debt on software quality,” in Proceedings of the 2nd Workshop on Managing Technical Debt. ACM, 2011, pp. 17–23.
14. C. Sterling, Managing Software Debt: Building for Inevitable Change. Addison-Wesley Professional, 2010.
15. C. Seaman, Y. Guo, C. Izurieta, Y. Cai, N. Zazworka, F. Shull, and A. Vetrò, “Using technical debt data in decision making: Potential decision approaches,” in Proceedings of the Third International Workshop on Managing Technical Debt. IEEE Press, 2012, pp. 45–48.