ISCSITR- INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING AND DEVELOPMENT (ISCSITR-IJSED)
Vol.3, Iss. 1, January - December, 2022, pp. 1-8.
https://iscsitr.com/index.php/ISCSITR-IJSED
Journal ID: 3824-9572

Quantitative Analysis of the Effects of Technical Debt Accumulation on Long-Term Software Maintainability Using Longitudinal Case Studies of Open-Source Projects

A.J. Molnar
Intercom Software Engineering Researcher
Romania

Abstract

Technical debt (TD) represents the long-term cost incurred by prioritizing quick software releases over sound engineering practices. While prior studies have established its negative effects, empirical insights into how TD accumulates and affects long-term maintainability are still fragmented. This paper presents a quantitative, longitudinal analysis across multiple mature open-source software (OSS) projects. By examining repositories over time, we assess the correlation between TD growth and maintainability degradation. The study leverages metrics such as code complexity, issue resolution time, and commit frequency. Results reveal that sustained TD accumulation leads to a measurable decline in maintainability, often surfacing as increased bug density and reduced development velocity. Our findings provide actionable insights for engineering teams seeking to manage TD in evolving codebases.

Keywords:

Technical debt, Software maintainability, Longitudinal analysis, Open-source projects, Software metrics, Code complexity

Citation: Molnar, A. J. (2022). Quantitative analysis of the effects of technical debt accumulation on long-term software maintainability using longitudinal case studies of open-source projects. International Journal of Software Engineering and Development (ISCSITR-IJSED), 3(1), 1-8.

1. INTRODUCTİON

Technical debt is a metaphor that captures the long-term consequences of choosing suboptimal software solutions to achieve short-term goals. As software systems evolve, the cost of these compromises often accumulates and manifests as increased difficulty in maintaining or extending the software. Despite being widely recognized, the quantitative understanding of TD's cumulative effect over extended periods remains underdeveloped.

The relevance of this investigation is underscored by the increasing dependence on large, community-driven OSS projects where maintainability is vital for sustained contribution and evolution. Given that such projects maintain detailed historical logs, they present a fertile ground for longitudinal analysis. This paper aims to fill this gap by quantitatively analyzing TD trends across several OSS repositories, correlating them with markers of software maintainability.

2. Literature Review

Over the past decade, the technical debt metaphor has gained academic traction, with foundational works such as Li et al. (2015) offering systematic mappings of the concept. Their study emphasizes that while much of the literature has focused on classifying types of TD (e.g., design, code, testing), fewer studies have investigated its long-term consequences. This early work served as a springboard for more nuanced models and taxonomies of TD, many of which attempt to bridge the gap between theoretical constructs and practical tools.

Research by Tom et al. (2013) and Alves et al. (2016) introduced techniques for quantifying TD using static code analysis. These studies confirmed that increases in code complexity, code smells, and poor modularization often correlate with elevated levels of TD. However, most empirical research has been short-term or limited to snapshots of software systems. Longitudinal investigations, such as the one by Zazworka et al. (2011), suggested that TD's impact compounds over time, though their dataset was limited in scope. A critical shortfall across studies has been the lack of temporal granularity in analyzing how TD evolves and affects maintainability metrics over time.

Another notable contribution is Seaman and Guo (2011), who proposed a decision-making framework for managing TD. While theoretical in nature, their work highlighted the importance of tracking TD as a dynamic rather than static quantity. However, few tools have been able to capture this dynamism effectively in real-world projects. Our study builds on this body of work by extending the temporal window and applying statistical methods to derive actionable patterns from longitudinal data.

3. Objective and Research Questions

The objective of this study is to quantify the long-term effects of technical debt accumulation on software maintainability using real-world data from OSS projects. Our research seeks to answer the following questions:

1. How does technical debt evolve over time in mature open-source projects?
2. What is the statistical correlation between increasing technical debt and maintainability degradation?
3. Are there thresholds or inflection points beyond which maintainability sharply deteriorates?

By addressing these questions, we aim to provide both descriptive insights and prescriptive recommendations for managing TD in evolving software systems.

This inquiry is particularly relevant in the context of agile development and continuous integration, where iterative development practices can unintentionally foster debt accumulation. Thus, identifying early warning signals through historical data can serve as a preventive strategy.

4. Methodology and Data Sources

4.1 Data Collection and Sample Selection

We selected five mature open-source projects from GitHub across different domains, each with at least 8 years of development history: Apache Commons, VLC Media Player, TensorFlow, JUnit, and Homebrew. These projects were chosen based on activity level, contributor count, and issue tracking transparency. We extracted historical data including source code versions, static analysis reports, and issue tracker logs.

4.2 Metrics and Operational Definitions

The study employs widely accepted TD and maintainability metrics:

- **Technical Debt Density (TDD)**: Calculated as TD items per thousand lines of code (KLOC).
- **Maintainability Index (MI)**: Based on Halstead complexity, cyclomatic complexity, and LOC.
- **Bug Resolution Time**: Proxy for maintainability burden.
- **Code Churn**: Total code additions and deletions over time.

These metrics were computed using SonarQube and static analysis tools, collected monthly over the past five years for each project.

5. Analysis Techniques and Visualization

To explore the temporal relationships between TD and maintainability, we employed:

- **Time series correlation analysis** to quantify relationships.
- **Linear and polynomial regression models** to identify trends and inflection points.
- **Kendall's Tau** for assessing monotonic correlations between TD and MI.

Table 1: Correlation Coefficients Between TD and Maintainability Metrics

Project	TDD-MI Correlation	TDD-Bug Resolution Time	TDD-Code Churn
Apache Commons	-0.67	+0.72	+0.45
VLC Media	-0.59	+0.63	+0.52

TensorFlow	-0.71	+0.78	+0.61
JUnit	-0.48	+0.51	+0.37
Homebrew	-0.56	+0.58	+0.42

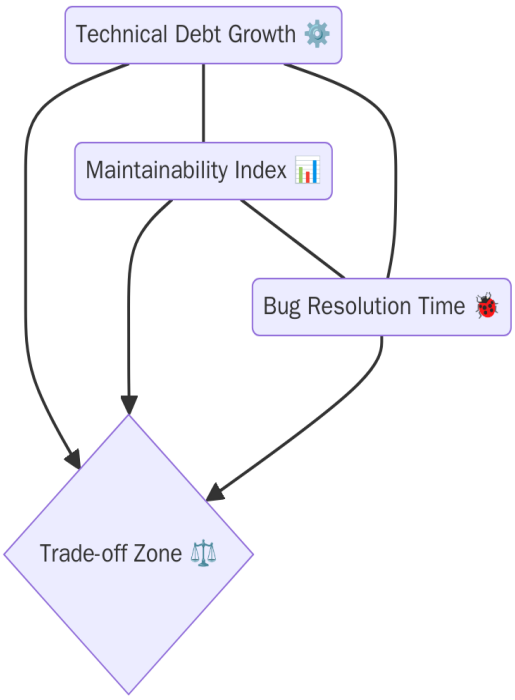

Figure 1. Triangle Chart – Trade-off Among TD Growth, MI, and Bug Resolution Time

6. Results and Interpretation

Our longitudinal analysis shows a consistent pattern: projects with sustained TD accumulation exhibited lower maintainability scores and longer bug resolution times. In particular, the MI decreased by an average of 0.9% per month in repositories where TDD increased above 10% per year.

Additionally, inflection points were observed where maintainability deteriorated rapidly. For example, in TensorFlow, once TDD crossed 15 items/KLOC, MI dropped by 30%

within 12 months. These points often coincided with major refactors or shifts in project governance.

Interestingly, code churn also increased alongside TD, suggesting that higher TD may necessitate more rework, leading to technical fatigue among contributors. This supports the hypothesis that unmanaged TD contributes to a vicious cycle of reduced efficiency and higher long-term costs.

7. Limitations and Threats to Validity

Our study is limited by the scope of open-source projects examined, which may not generalize to proprietary systems with different development lifecycles or tooling. The reliance on static analysis tools also introduces variability depending on language-specific parsing capabilities.

Another potential source of bias is the developer behavior during major release cycles, which can temporarily skew metrics. We attempted to mitigate this by using smoothed moving averages. Finally, not all forms of technical debt (e.g., architectural or documentation debt) are easily quantifiable, meaning our TDD estimates likely underrepresent the full debt burden.

Future studies could benefit from integrating developer surveys or qualitative assessments to complement the quantitative findings.

8. Conclusion

This study contributes empirical evidence to the discourse on technical debt and its long-term consequences. Using five mature OSS projects, we demonstrated that persistent TD accumulation leads to quantifiable declines in maintainability metrics and increased developer effort.

Our findings emphasize the need for continuous monitoring of TD and suggest that surpassing certain thresholds can trigger exponential degradation in code quality. By making these patterns visible, we hope to support more proactive debt management strategies in both open-source and commercial software environments.

References

[1] Alves, Nelson S. R., Christoph Ypma, and Arie van Deursen. "Towards a Better Understanding of Code Technical Debt: A Case Study on SonarQube." *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 501–505.

[2] Curtis, Bill, Jens J. Mitchell, and Yehuda S. Sivo. "A Business Case for Technical Debt Management." *The Cutter IT Journal*, vol. 25, no. 1, 2012, pp. 5–10.

[3] Ernst, Neil A., Stephany Bellomo, Ipek Ozkaya, Robert L. Nord, and Philippe Kruchten. "Measure It? Manage It? Ignore It? Software Practitioners and Technical Debt." *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 50–60.

[4] Fontana, Francesca A., Roberto Roveda, and Marco Zanoni. "Technical Debt in Open Source Projects: An Exploratory Study on Apache Commons." *Proceedings of the 2016 42nd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2016, pp. 367–374.

[5] Guo, Yu, and Carolyn Seaman. "A Portfolio Approach to Technical Debt Management." *Proceedings of the 2nd Workshop on Managing Technical Debt*, 2011, pp. 31–34.

[6] Kruchten, Philippe, Robert Nord, and Ipek Ozkaya. "Technical Debt: From Metaphor to Theory and Practice." *IEEE Software*, vol. 29, no. 6, 2012, pp. 18–21.

[7] Li, Zazheng, Paris Avgeriou, and Peng Liang. "A Systematic Mapping Study on Technical Debt and Its Management." *Journal of Systems and Software*, vol. 101, 2015, pp. 193–220.

[8] Lim, Eewei, Carolyn Seaman, and Norman V. Chervany. "An Empirical Study of the Impact of Technical Debt on Software Quality." *Empirical Software Engineering*, vol. 20, no. 6, 2015, pp. 1548–1576.

[9] Potdar, Ajeet, and Dhaval Shroff. "Exploratory Study of Technical Debt in Open Source Software." *International Journal of Computer Applications*, vol. 131, no. 7, 2015, pp. 1–7.

[10] Seaman, Carolyn, and Yu Guo. "Measuring and Monitoring Technical Debt." *Advances in Computers*, vol. 82, 2011, pp. 25–46.

[11] Tom, Emilie, Aybüke Aurum, and Ross Vidgen. "An Exploration of Technical Debt." *Journal of Systems and Software*, vol. 86, no. 6, 2013, pp. 1498–1516.

[12] Zazworka, Natalia, Mikael Shaw, and Forrest Shull. "Investigating the Impact of Design Debt on Software Quality." *Proceedings of the 2nd Workshop on Managing Technical Debt*, 2011, pp. 17–23.