Spring 2015

Longitudinal Analysis of Technical Debt for Strategic Platform Adoption

J. Yates Monteith
Clemson University

John D. McGregor
Clemson University

Mike Finney
Clemson University

Longitudinal Analysis of Technical Debt in Support of Strategic Software Platform Adoption

J. Yate Monteith, Mike Finney, John D. McGregor
{jymonte[1], johnmc[1], mfinney[2]}@clemson.edu
School of Computing[1] and Department of Mathematical Sciences[2]

Abstract

Increasingly, software producing organizations utilize a common software platform; however, little expertise exists on selecting which platform to use when presented a number of different platforms. While technical debt can be used to examine the quality of a software platform by the organization that produces the software, a single discrete data point does not provide sufficient context for analysis. In this paper, we seek to resolve this difficulty by applying linear regression analysis to technical debt data collected by the SonarQube static analyzer. We apply this method to a case study on Cytoscape network analysis platform to perform a pedagogical investigation on the longitudinal technical debt found in that platform. We present our case study on the longitudinal technical debt of the Cytoscape network analysis platform, utilizing the data and analysis generated from our method.

Introduction

Technical debt is a metaphor describing the difference between `good' code and `not-quite-right' code. While software development is rife with not-quite-right code, research in technical debt seeks to provide a context and rationale for what is technical debt, why it is incurred, and how it is introduced into a project. Much like software architecture, technical debt exists and its effects are felt in development regardless whether or not it is acknowledged by developers

Strategic decision makers tasked with selecting a platform for development may be able to glean useful information from analyzing the technical debt of a prospective platform. However, tools used to measure technical debt only capture the state of the software platform at the time its analyzed. A single data point measuring the technical debt of platform only provides a limited amount of information to the strategic decision maker regarding the rates and frequencies at which technical debt is incurred. By taking a longitudinal approach to analyzing the technical debt of the platform, strategic decision makers can glean better insight into trends in the development process, as well as the rate at which technical debt is incurred in development.

Software Platforms

For the purposes of our work, we are taking on the perspective of a developer or project manager considering using the platform as a basis for a product. This perspective helps provide context for our method and analysis. When developing a software product based on a software platform the pace of development is bounded by the pace of development of the software platform. Furthermore, the quality of product developed relies heavily on the quality of the software platform. One example of a software platform is the iOS and Android mobile operating systems. Another example is the Cytoscape network analysis platform, which provides assets to developers for producing plug-ins that aid in network analysis in chemistry, biology and other related scientific and data centric fields.

Choosing a software platform that is not burdened by an unmanageable level of technical debt, as well as having an auspicious development history is an important factor for software platform adoption.

Technical Debt

Technical debt is incurred when a development artifact is produced that is known or suspected to be not-quite-right. However, if there is sufficient value in waiting to correct these errors, these imperfections do not need to be fixed right away. These imperfections represent a loan, the principle of which is the cost to remedy. If development artifact is produced that depends on the imperfect artifact, it is likely that reworking the imperfect artifact will necessitate rework on the second, dependent artifact. The additional work that is done as a result of the initial technical debt can be viewed as interest incurred on the principle technical debt. The following are some examples of decisions surrounding technical debt:

- When faced with changing requirements, do we develop for what may change, or delay development until uncertainty is resolved?
- Do we utilize design patterns that enhance our design today, or benefit the evolution of our design tomorrow?
- When behind schedule with an impending deadline, do we ship a project missing a key feature, or without thorough documentation?

Many of these options have no clear optimal decision: one choice is not clearly better than the other, but rather a lateral move, defined by the trade-offs that decision provides. Each of these decisions brings off a trade-off, between the cost and value of performing one set of actions over another set of actions.

Metrics for Technical Debt

Minor Violations: Simple violations that usually concern style and syntax of source code.
Major Violations: Violations that may have semantic repercussions and meaning.
Critical Violations: Critical violations indicate that serious errors exist in programming semantics.

Technical Debt Ratio (TDR): The technical debt ratio provides a ratio between the actual technical debt and theoretical maximum technical debt that exists within a project. This metric provides a weighted measure on how much technical debt exists in a project compared to how much technical debt could exist in a project.

Rules Compliance Index (RCI): The RCI of a project encodes the percentage of rules that are not violated out of all the triggered rules in analysis. While similar to TDR, it is important to note that TDR encodes a weighted aggregate of technical debt, while RCI encodes an instance-based unweighted aggregate of technical debt.

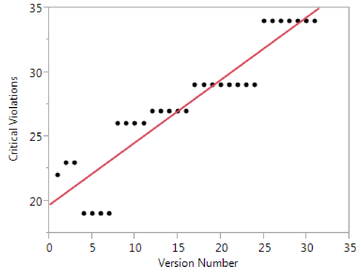

Figure 1. Critical Violations Per Version

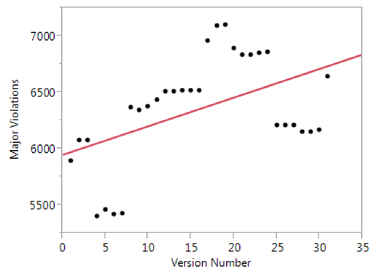

Figure 2. Major Violations Per Version

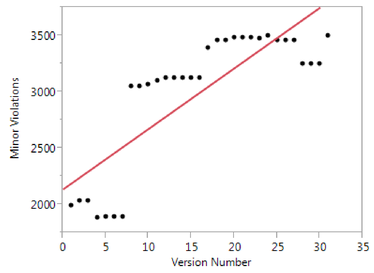

Figure 3. Minor Violations Per Version

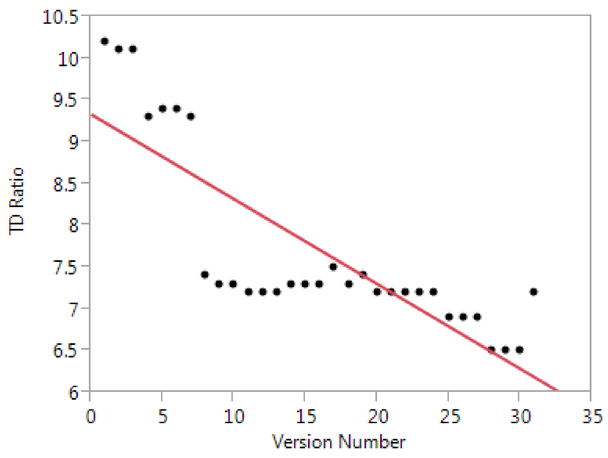

Figure 4. TDR per Version

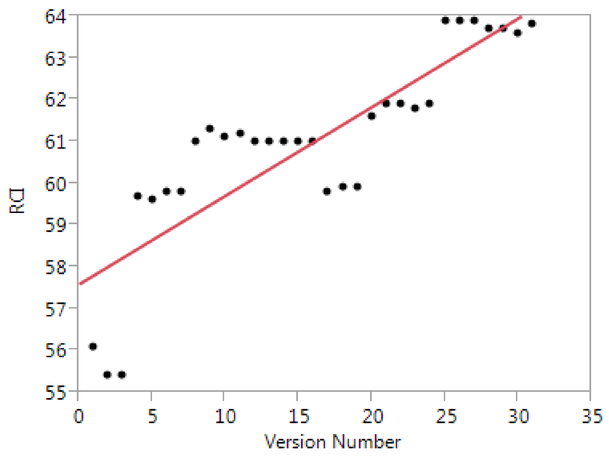

Figure 5. RCI per Version

Analysis and Conclusion

While the source code for the Cytoscape network analysis platform is rife with violations, and shows increasing technical debt at nearly every data point measured, the case for adopting is strong. The TDR and its rate of change indicates that the sources of technical debt introduced with new versions tend to be less severe, rather than more severe, as our weighted aggregate is decreasing. The RCI indicates that the development of Cytoscape is becoming more rigorous over time, incurring less violations with new introductions of source code. Finally, combining those metrics with the metric correlations indicates that the TDR is decreasing with introductions of new source code, while the RCI is increasing with new introductions of source code, and the two are inversely related, again, suggesting that the development is incurring fewer rule violations, with each newly introduced violation being less severe. These factors leads to a strong case for adopting the Cytoscape network analysis platform as the foundation for developing a software product.

Methodology

We have analyzed 31 release versions of the Cytoscape Desktop application, beginning with the 2.2-Pre Release, released 2008, and ending with 2.8.0 Alpha 1, the last release of the 2.x branch, released in 2012. We acquired this data through Cytoscape's webpage and code repository on Github between December 17th and 29th, 2013. Each of these builds are primarily written in Java with additional scaffolding provided by shell scripts and build files. Only the Java code was analyzed for technical debt. The 2.x release was chosen primarily because of a major shift in development, architecture and build process found in the 3.x release of Cytoscape.

The software was built using the provided Ant build scripts. Technical debt analysis was performed by SonarQube 3.7 using their Sonar-Runner 2.3 driver and the built-in plug-ins for analyzing Java code, version 1.3, and technical debt, 1.2.1.

After collecting data on code level occurrences of technical debt, we performed statistical analysis on the results using JMP 11. For each metric collected, we computed the linear regression with respect to version. The model's significance was measured using p-values corresponding to slope estimates. All linear model results are significant at $\alpha = 0.05$. Additionally, linear models were calculated to establish the relationship of total violations as a function of statements and classes. This was measured using adjusted R^2 values in addition the metrics above. Lastly, correlation was calculated between the different metrics produced by SonarQube and each other, as well as size metrics of the code base.

Results

The simplest metric that SonarQube provides, minor, major and critical violations, shown in Figures 1, 2 and 3. For each type of violation, the data is fit to a linear regression with a positive slope. This suggests a general positive trend implying one can expect the number of violations to increase over time. While these regressions show the number of violations is increasing with each release of Cytoscape, they do not describe the rate at which technical debt is growing with respect to the size of the code.

Table 1. Sum of Violations and Size Regressions

	Slope	P Value	Adjust R2
Classes	0.3376478	< 0.0001	0.942605
Statement	8.7323459	< 0.0001	0.906127

Table 1 provides a numerical computation of the R^2 values for the regression between the sum of all violations and the size of the codebase as measured in classes and statements. These values suggest that 94% and 90% of the variation in code violations can be attributed to the number of classes and statements.

TD Ratio describes the ratio between the current amount of technical debt in a project and the maximum value of technical debt in a project. Figure 4 shows the plot of TD Ratio of each version along with its linear regression. The negatively sloped regression line suggests that the technical debt ratio has been declining with each new release. We can gain additional understanding on these results by examining the RCI, the metric describing the percentage of rules that are not violated out of all the triggered rules in analysis. With each new version, the rule compliance index increases as well, as shown in Figure 5. The primary difference between TDR and RCI is weighted and unweighted aggregate metrics. The TDR accounts for the severity of violations, while RCI looks at the raw number of rules violated in a project. Together, the TDR and RCI suggest that the development of Cytoscape improved over time, incurring both fewer rule violations, indicated by the RCI, as well as rule violations being less severe, indicated by the TDR.