Longitudinal Analysis of Technical Debt for Strategic Platform Adoption

J. Yates Monteith
Clemson University

John D. McGregor
Clemson University

Mike Finney
Clemson University

Follow this and additional works at: https://tigerprints.clemson.edu/grads_symposium

Recommended Citation
https://tigerprints.clemson.edu/grads_symposium/125

This Poster is brought to you for free and open access by the Research and Innovation Month at TigerPrints. It has been accepted for inclusion in Graduate Research and Discovery Symposium (GRADS) by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.
Abstract

Increasingly, software producing organizations utilize a common software platform; however, little expertise exists on selecting which platform to use when presented with 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.

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 departs from the imperfect artifact, it is likely that remedying 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 be fixed right away. These imperfections represent a loan, the principle of which is the cost to remedy. If development artifact is produced that departs from the imperfect artifact, it is likely that remedying 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:

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 basic 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.

Metrics for Technical Debt

Minor Violations: Simple violations that usually concern style and syntax of source code.

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.

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 correlating indicates that the TDR is decreasing with new introductions of 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 lead 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 28th, 2013. Each of these builds and revisions were 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 the 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 α = 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 to the metrics above. Lastly, correlation was calculated between the different metrics produced by SonarQube and each other, as well as size metrics of the codebase.

Results

The simplest metric that SonarQube produces, minor, major and critical violations, shown in Figures 1, 2 and 3. For each type of violation, the data is fit to a regression with respect to release version. 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, as 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 that the RCI is weighted and unweighted aggregate metrics. The TDR accounts for the number 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 has significantly reduced over time, incurring fewer rule violations, as indicated by the RCI, as well as rule violations being less severe, indicated by the TDR.