Static Software Metrics for Reliability and Maintainability

Jeremy Ludwig, Steven Xu
Stottler Henke Associates, Inc.
San Mateo, CA

Frederick Webber
Air Force Research Laboratory
711th HPW/RHAS
WPAFB, OH

ABSTRACT

This paper identifies a small, essential set of static software code metrics linked to the software product quality characteristics of reliability and maintainability and to the most commonly identified sources of technical debt. An open-source plug-in is created for the Understand code analysis tool that calculates and visualizes these metrics. The plug-in was developed as a first step in an ongoing project aimed at applying case-based reasoning to the issue of software product quality.[1]

CCS CONCEPTS

• Software and its engineering~Software creation and management

KEYWORDS

Software product quality, technical debt, reliability, maintainability, architecture, metrics, static code analysis

1 INTRODUCTION

There is a consistent push to improve software product quality, especially for components that are designed to be heavily re-used and extended, e.g. as a shared module of a software product line with an expected long life.

The first objective of this abstract is to identify a small, essential set of static software code metrics linked to the software product quality characteristics of reliability and maintainability [5, 13] and to the most commonly identified sources of technical debt [4]. While some technical debt is unavoidable [9], a large survey of software engineers and architects across multiple organizations provides a practical view of the causes and sources of avoidable technical debt [4]. Their results indicate that architectural decisions, overly complex code, and lack of code documentation are the top three avoidable sources of technical debt in practice. Specifically, this abstract identifies a small set of static source code metrics that are related to the three given areas of technical debt and have a strong empirical or applied relationship with reliability and maintainability.

The second objective of this paper is to describe the open source plug-in that was created for the Understand code visualization and static analysis tool, which calculates and visualizes these metrics in an interactive report.

There is an abundance of related work in software quality, technical debt, and automated code review that identifies specific source code metrics, describes how the measurements of these metrics are aggregated, and how the aggregations are used to assess characteristics of software quality and technical debt. Summarizing this work is outside the scope of this abstract, see [3, 6] as a starting point.

In the remainder of this abstract, the Methods section describes the work performed on metric identification, calculation, and visualization. Following this, the Conclusion sections summarizes the results and introduces future work.

2 METHODS

This section first discusses the static source code metrics that were selected to measure software product quality in each of the three areas of avoidable technical debt. Following this, the plug-in created for the Understand code analysis tool is briefly described.

2.1 Architectural Metrics

While there are numerous possible measures of software architecture [8, 16, 18], the propagation cost and core size metrics as defined by [1] were specified by the research agenda. Propagation cost is a system-wide metric that describes the proportion of software files that are directly or indirectly linked to each other. The core size metrics involves classifying every component (class or file) into one of five architecture groups based on the number of direct and indirect links of the components: core, shared, control, peripheral, and isolate. The core group represents the largest set of components that are interdependently linked to each other. A primary contribution of the plug-in is to calculate these two metrics and produce the Design Structure Matrix (DSM) graphs.

The utility of the propagation cost and core size measures of architectural complexity has been demonstrated in a number of studies [10, 11, 17]. These measures have been shown to relate significantly to defect density, programmer productivity, and programmer retention. Core files have been found to contain more defects and cost more to maintain [11].

TechDebt 2018, May 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5713-5/18/05.
https://doi.org/10.1145/3194164.3194184

2.2 Complexity Metrics

A small set of complexity, size, and coupling metrics were selected based on evidence supporting correlation with, or prediction of, the software characteristics of reliability and maintainability. Given the vast available literature on software metrics and an equally large variety of metrics, a reasonable starting point is systematic literature reviews. [7] reviews 99 primary studies and compares their work to several other surveys and systematic literature reviews (e.g., [15] [14]). The results indicate that the link from metric to reliability and maintainability across studies is strongest for: LOC, WMC-Unweighted/WMC-McCabe, RFC, and CBO. Standard definitions for these metrics can be found in [5], which generally match the descriptions for how the calculations are performed by Understand [12].

2.3 Comment Metrics

The Code-To-Comment ratio is used as an initial measure of code commenting. This metric has been well studied as part of earlier work on quality models [2]. Anecdotally, it is also one of the metrics most-used by developers utilizing the Understand software. The plug-in relies on existing functionality within Understand to calculate this metric.

2.4 Reporting Results

An interactive HTML report is generated as shown in Figure 1. The report includes all the generated metrics on the left and the various architecture groupings in a Design Structure Matrix graph in the center, as described in [1].

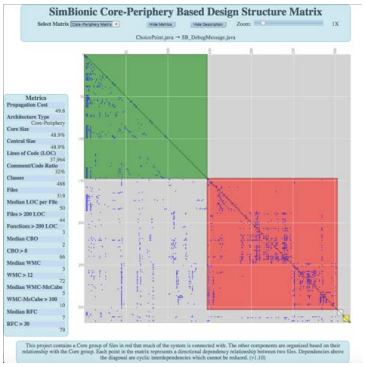

Figure 1. Interactive report generated for a repository.

3 CONCLUSION

Software code quality and technical debt have significant impact on a software product's reliability and maintainability. This paper identifies a small, essential, set of static software code metrics linked to reliability and maintainability and to the most commonly identified sources of technical debt. This paper also describes an open source plug-in that was created for the Understand code analysis tool, which calculates these metrics and produces an interactive report (github.com/StottlerHenkeAssociates/Software-Architecture-Evaluation). While the plug-in is useful as-is, it was developed as a first step in an ongoing project aimed at applying case-based reasoning to the issue of software product quality. The next step in this project aims to use the described plug-in as part of a research effort to define and validate the aggregation of these metrics as part of a software product quality model.

ACKNOWLEDGMENTS

This material is based upon work supported by the United States Air Force Research Laboratory under Contract No. FA8650-16-M-6732. The views, opinions, and/or findings contained in this article/presentation are those of the author/presenter and should not be interpreted as representing the official views or policies, either expressed or implied, of the AFRL. DISTRIBUTION A. Approved for public release: distribution unlimited (case 88ABW-2017-2167).

REFERENCES

[1] Baldwin, C. et al. 2014. *Hidden Structure: Using Network Methods to Map System Architecture*.

[2] Coleman, D. et al. 1995. The application of software maintainability models in industrial software systems. *Journal of Systems and Software*. 29, 1 (Apr. 1995), 3–16. DOI:https://doi.org/10.1016/0164-1212(94)00125-7.

[3] Curtis, B. et al. 2012. Estimating the Principal of an Application's Technical Debt. *IEEE Software*. 29, 6 (Nov. 2012), 34–42. DOI:https://doi.org/10.1109/MS.2012.156.

[4] Ernst, N.A. et al. 2015. Measure It? Manage It? Ignore It? Software Practitioners and Technical Debt. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (New York, NY, USA, 2015), 50–60.

[5] Fenton, N. and Bieman, J. 2014. *Software Metrics: A Rigorous and Practical Approach, Third Edition*. CRC Press, Inc.

[6] Ferenc, R. et al. 2014. Software Product Quality Models. *Evolving Software Systems*. T. Mens et al., eds. Springer Berlin Heidelberg. 65–100.

[7] Jabangwe, R. et al. 2014. Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review. *Empirical Software Engineering*. 20, 3 (Mar. 2014), 640–693. DOI:https://doi.org/10.1007/s10664-013-9291-7.

[8] Kazman, R. et al. 2015. A Case Study in Locating the Architectural Roots of Technical Debt. *Proceedings of the 37th International Conference on Software Engineering - Volume 2* (Piscataway, NJ, USA, 2015), 179–188.

[9] Kruchten, P. et al. 2012. Technical Debt: From Metaphor to Theory and Practice. *IEEE Software*. 29, 6 (Nov. 2012), 18–21. DOI:https://doi.org/10.1109/MS.2012.167.

[10] MacCormack, A. et al. 2006. Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code. *Management Science*. 52, 7 (Jul. 2006), 1015–1030. DOI:https://doi.org/10.1287/mnsc.1060.0552.

[11] MacCormack, A. and Sturtevant, D.J. 2016. Technical debt and system architecture: The impact of coupling on defect-related activity. *Journal of Systems and Software*. 120, (Oct. 2016), 170–182. DOI:https://doi.org/10.1016/j.jss.2016.06.007.

[12] Metrics | SciTools.com: *https://scitools.com/feature/metrics/*. Accessed: 2017-03-07.

[13] Organización Internacional de Normalización 2011. *ISO-IEC 25010: 2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*. ISO.

[14] Radjenović, D. et al. 2013. Software fault prediction metrics: A systematic literature review. *Information and Software Technology*. 55, 8 (Aug. 2013), 1397–1418. DOI:https://doi.org/10.1016/j.infsof.2013.02.009.

[15] Riaz, M. et al. 2009. A Systematic Review of Software Maintainability Prediction and Metrics. *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement* (Washington, DC, USA, 2009), 367–377.

[16] Stevanetic, S. and Zdun, U. 2015. Software Metrics for Measuring the Understandability of Architectural Structures: A Systematic Mapping Study. *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering* (New York, NY, USA, 2015), 21:1–21:14.

[17] Sturtevant, D.J. 2013. *System design and the cost of architectural complexity*. Massachusetts Institute of Technology.

[18] Xiao, L. et al. 2016. Identifying and Quantifying Architectural Debt. *Proceedings of the 38th International Conference on Software Engineering* (New York, NY, USA, 2016), 488–498.