CBR Insight:
Measure and Visualize Source Code Quality

Jeremy Ludwig & Devin Cline
Stottler Henke Associates, Inc.
San Mateo, CA, US
ludwig, dcline @ stottlerhenke.com

***Abstract*—A critical aspect of software development is creating high-quality source code that is reliable, maintainable, and has limited technical debt. Software development teams generally employ a variety of design techniques, processes, and tools to continually work towards quality code while balancing the overall time and budget demands of the project. The goal of CBR Insight (CBRI) is to provide an objective and understandable measure of software quality that can help guide decisions and direct limited resources during software acquisition, development, and sustainment. CBRI supports the ability of technical and non-technical decision makers to verify that a project's software implementation follows through on promises around developing and sustaining reliable and maintainable software while managing technical debt.**

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

I. Introduction

Creating and maintaining high-quality software is especially important for critical systems such as those made for NASA and the DoD, and for software product lines where long-lived, reusable modules are intended to be shared by multiple systems. The goal of CBR Insight (CBRI) is to provide an objective and understandable measure of software quality that can help guide decisions during software acquisition, development, and sustainment.

CBRI performs four distinct tasks in fulfilling this goal. First, a small set of source code metrics highly related to software reliability, maintainability, and preventable technical debt are calculated. Second, realistic targets are developed for these metrics based on similar, successful, 'peer' projects. Third, an aggregated score is generated by comparing the calculated metrics to the target values. Fourth, the results are presented to decision makers in an accessible dashboard overview. The remainder of this abstract includes an overview of related work, a closer look at CBRI, and a brief discussion of ongoing work.

II. Background and Related Work

CBRI calculates a small, essential set of static software code metrics linked to the software product quality characteristics of reliability and maintainability [1], [2] and to the most commonly identified sources of technical debt [3]. Architectural decisions, overly complex code, and lack of code documentation are the top three avoidable sources of technical debt in practice. CBRI uses a plugin to Understand [4] to calculate metrics in each of three given areas. While CBRI focuses on presenting an overview to decision makers, software developers can use Understand to calculate the same metrics and address identified deficiencies.

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 [5], [6] as a starting point.

III. CBR Insight

The CBR Insight dashboard (Figure 1) focuses on measuring and visualizing software code quality across multiple projects in three important areas: architecture, complexity, and clarity.

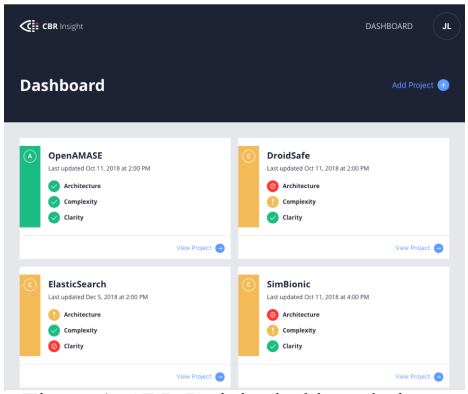

Figure 1. CBR Insight dashboard view.

CBRI calculates static source code metrics for each of these three areas. A brief description is given for non-standard metrics. See the citations for more information. The architecture metrics are Core Size and Propagation Cost [7]. Files in the Core architecture group generally contain more defects and cost more to maintain, so a smaller core size is better. The two complexity metrics are Duplicate Lines of Code and Overly Complex Files. An overly complex file is one that exceeds 4 of 5 thresholds from a set of standard software metrics [1] including LOC, WMC-Unweighted, WMC-McCabe, RFC, and CBO. The Code-To-Comment ratio is used as an initial measure of clarity. This metric has been well studied as part of earlier work on quality

models [8]. See [9] for a more detailed discussion of the specific architecture, complexity, and clarity metrics selected for use in CBRI.

Generating a target range for a metric involves identifying peer projects on Github that are similar to the project of interest, calculating all of the metrics for each of the peer projects, and then calculating the interquartile range of each metric across the peer projects.

An overall score is calculated for each project, along with scores of the architecture, complexity, and clarity aspects. These scores are calculated by comparing the calculated metric values of the project of interest against the generated target range. The overall scores are assigned to letter grades A thru F for visualization in the dashboard.

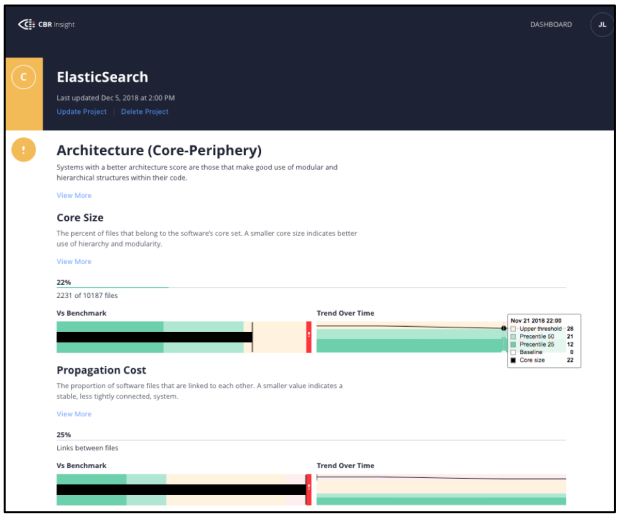

Figure 2. The architecture section of the Project View.

The Dashboard is a starting point for the user to drill down into the details of each project. The Project View (Figure 2) provides a description of the underlying metrics used to generate the scores for the project and visualizes the calculations over time. The visualizations include color-coded target ranges that were determined by analyzing successful peer projects as well as a tree-map of file size and complexity organized by the Core Size architecture set. Every section and metric contain accessible descriptions to assist the user in understanding the scores and measurements.

IV. CONCLUSION AND FUTURE WORK

Software code quality and technical debt have significant impact on a software product's reliability and maintainability. CBRI supports the ability of technical and non-technical decision makers to verify that a project's software implementation follows through on promises around developing and sustaining reliable and maintainable software while managing technical debt.

There is a long history of software engineering research in the area of software product quality and numerous existing tools aimed at performing automated code quality assessment. What makes CBR Insight a complementary addition to existing tools is: (i) the calculation of a small, essential set of metrics associated with maintainability, reliability, and technical debt, (ii) using peer projects to set the targets associated with each metric and (iii) presenting the information in a format preferred by decision makers. CBRI components are being released at https://github.com/StottlerHenkeAssociates as they are completed.

Ongoing work on CBRI is currently focused on a number of different issues. Some of these issues are in the inner workings of CBRI, investigating improvements to the library of peer projects and the score aggregation methods. Other issues include changes to the user interface: updating the graphical layout, displaying information on peer projects, and visualizing changes in the source code relative to a baseline measurement. The last issue is identifying additional metrics that gauge the clarity of software as it relates to reliability and maintainability [10].

ACKNOWLEDGMENT

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.

REFERENCES

[1] N. Fenton and J. Bieman, *Software Metrics: A Rigorous and Practical Approach, Third Edition*, 3rd ed. Boca Raton, FL, USA: CRC Press, Inc., 2014.

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

[3] N. A. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord, and I. Gorton, "Measure It? Manage It? Ignore It? Software Practitioners and Technical Debt," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, New York, NY, USA, 2015, pp. 50–60.

[4] "SciTools.com." [Online]. Available: https://scitools.com/. [Accessed: 24-Dec-2018].

[5] R. Ferenc, P. Hegedűs, and T. Gyimóthy, "Software Product Quality Models," in *Evolving Software Systems*, T. Mens, A. Serebrenik, and A. Cleve, Eds. Springer Berlin Heidelberg, 2014, pp. 65–100.

[6] B. Curtis, J. Sappidi, and A. Szynkarski, "Estimating the Principal of an Application's Technical Debt," *IEEE Softw.*, vol. 29, no. 6, pp. 34–42, Nov. 2012.

[7] C. Baldwin, A. MacCormack, and J. Rusnak, "Hidden Structure: Using Network Methods to Map System Architecture," 2014.

[8] D. Coleman, B. Lowther, and P. Oman, "The application of software maintainability models in industrial software systems," *J. Syst. Softw.*, vol. 29, no. 1, pp. 3–16, Apr. 1995.

[9] J. Ludwig, S. Xu, and F. Webber, "Compiling static software metrics for reliability and maintainability from GitHub repositories," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2017, pp. 5–9.

[10] C. Chen, R. Alfayez, K. Srisopha, B. Boehm, and L. Shi, "Why is It Important to Measure Maintainability, and What Are the Best Ways to Do It?," in *Proceedings of the 39th International Conference on Software Engineering Companion*, Piscataway, NJ, USA, 2017, pp. 377–378.