Code Ownership and Quality: Building a "Component Health Service" at Appian Corporation

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Ethan Gahm

Spring, 2023

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Rosanne Vrugtman, Technical Writing Advisor

Abstract
Appian Corporation, a Virginia-based tech company focusing on low-code automation software, has struggled for some time to track the ownership and maintenance status of its code repositories. Over the summer of 2022, as a software engineering intern in Appian Engineering's "Development Insights" squad, I helped to build a new cloud-based service for tracking code contributions and code health across the company's infrastructure. Working with a team of five full-time engineers and one other intern, I developed a Python application, deployed on a Kubernetes cluster, which fetched, parsed, aggregated and stored metrics related to test coverage, code quality, code contributions, and more. Our aim was to provide company leadership with insight into which components were being actively maintained, by whom, and to what extent best practices were being employed by the company's engineers. During my time at the company, I was not able to see the completion of the service. However, a bare-bones backend was built out, establishing a strong base from which to develop a complete application, including a front-end interface.

1. Introduction

Appian Corporation's software engineering division is divided into a number of Strategic Business Units (SBUs) each of which is subdivided into teams and squads. Each team and its component squads focus on particular long-term, mission-critical goals. The frequent dissolution, division, merging, and renaming of squads means it is not always clear who is responsible for maintaining a portion of the company's code base. Frequently, ownership of code is informally given to whichever engineer(s) originally authored a particular file or project, but this can become problematic when engineers leave the company or move to a different squad with a different focus. Engineers may be called on to update or address issues in projects far outside their current squad's domain.

Meanwhile, Appian has begun adoption of a static code analysis tool called SonarQube which programmatically examines source code and provides reports on code quality. Each repository or sub-module must be specially configured for SonarQube analysis to be performed, and configuration must be done carefully to ensure that SonarQube can view and analyze all relevant files in a project. So far, SonarQube has been adopted by only a subset of Appian Engineering's squads on a portion of its repositories.

The interplay between these two seemingly unrelated issues becomes apparent when it is necessary to assign a responsible party for configuring a project for SonarQube. Once configured, if SonarQube detects bugs, code smells, or other signs of poor code health, a knowledge of code ownership is necessary to determine who should address the issue(s).

2. Related Works

Some earlier scholarly work has been dedicated to a) measuring the importance of code ownership in large organizations, and b) measuring and valuing code health and code quality. Greiler, et. al. (2015) [1] replicated and extended results from earlier studies showing that strong code ownership correlates with high code quality. Breaking from earlier work, this study examined ownership of code directories, a level of granularity between source files and entire binaries, which happens to match the target organizational structure for code ownership within Appian Corporation. The study approximated a measure of code quality by counting bug reports and bug fixes in version control systems and determined

degree of ownership through counts of code contributions by individuals. In essence, Greiler, et. al. found that a small number of highly-involved contributors to a directory resulted in low total bug counts.

Sedano, et. al. (2016) [2] delve further into the idea of code ownership and attempt to identify which factors contribute to a strong sense of code ownership. They conclude that in order to feel that they "own" their code, engineers need to make frequent contributions, hold a high degree of understanding of the code base, and believe the overall code quality to be high. This may suggest that the best way to assign code ownership is to measure existing rates of contribution. The study also highlights how code quality and code ownership may be mutually reinforcing: healthy code may be better understood by developers who in turn feel a stronger sense of ownership and are better able to maintain and update repositories.

Tornhill and Borg (2022) [3] examine the impact of poor code quality on overall business productivity. Using a code health analysis tool from CodeScene, they conclude that poor code quality is responsible for wasting 42% of software developers' time. Furthermore, code deemed to be of "low quality" by the CodeScene tool is found to contain 15 times more defects than high quality code. These findings attribute a high value to tracking and enforcing high code quality within organizations.

3. System Design

We built the Component Health Service as a cloud-based application to be deployed on Appian's internally managed Kubernetes cluster.

3.1 System Architecture

We wrote the backend of the in Python and utilized the FastAPI library to construct endpoints. We built different endpoints in order to retrieve information regarding code contributions to particular repositories or to check on code health metrics as measured by SonarQube. Ultimately, we did not use these endpoints in the production system, instead making calls to the relevant functions through an internal scheduler rather than via external API calls. However, the endpoints still proved useful for testing purposes.

Alongside the main application we deployed a PostgreSQL database as an adjacent Kubernetes pod and used it to record metrics over time. The database included two tables: one for recording repository contributions and another for storing SonarQube metrics. Additionally, we constructed a third "component health" table by combining data from each of the other two. This third table consolidated core metrics and attempted to summarize the overall health of each repository.

3.2 Deployment

We deployed the application to an internal Kubernetes cluster using the Helmfile library and a set of "GitLab Runners." This system worked on a push-based model, whereby each time a change was made to the application's remote repository on GitLab, a runner would be triggered which would push the code changes up to a remotely-hosted Kubernetes node.

Early in the development process, we grappled with a pull-based deployment approach using an open-source tool called Argo. We were prompted to transition to Argo by a corporation-wide effort to move away from GitLab runners for security and reliability reasons. We eventually abandoned Argo, however, due to difficulties that we

experienced with the adaptability and reliability of the company's internal implementation of the tool.

3.3 Challenges

Testing was by far the most difficult part of the development process. The other intern on our squad and I had little prior experience with writing tests for production grade enterprise software and there was a difficult learning curve associated with the process. In particular, we found it challenging to effectively mock inputs for functions so as to produce complete code coverage while sufficiently isolating independent code components with their own targeted tests.

We also experienced limitations related to the capabilities of Appian's internal infrastructure. Initially, we planned to take advantage of a library called "KNative" to deploy the Component Health Service as a serverless application. However, we discovered that Appian's internal Kubernetes cluster was not updated to accommodate several necessary KNative features. Judging that it would be too burdensome and slow to ask other teams at the company to update their infrastructure to accommodate the service, we had to abandon this plan.

3.4 Future Frontend Work

During the span of the summer, we completed the core of the Component Health Service's backend, but I left the company just before work on the frontend was to begin.

We planned to construct a frontend using Appian's own low-code web development platform, creating a dashboard using the company's "SAIL" programming language. This frontend would then be hooked up to the PostgreSQL database, cleanly displaying and summarizing information about code contributions and code quality. The dashboard would be visible from Appian's "Home" portal, accessible to all employees at the company.

4. Results

Through extensive use of automated unit tests, we achieved close to 100% code coverage on the application, ensuring a degree of reliability and robustness. We presented our progress on the Component Health Service at several meetings attended by all members of the squad's SBU. Engineers from outside the squad asked questions and provided feedback on the product. It became clear from these meetings that there was not a strong understanding of SonarQube, SonarQube configuration, or the importance of measuring and tracking component health. This reinforced the importance both of the project itself and of communicating the project's core goals and capabilities within the company.

Because the summer internship program ended before the application's frontend could be constructed and before the service could be put into production, I did not see the direct impact of the service on Appian's engineering department. However, should the project be completed and effectively deployed, it will prove useful in determining how to effectively allocate time and energy towards improving the quality and robustness of Appian's core infrastructure.

5. Conclusion

The Component Health Service represents an important step towards understanding, managing, and addressing issues related to code ownership and code quality at Appian Corporation. The project also aligns closely with the Development Insights squad's goal of developing a more metrics-driven approach to software development. By

measuring, recording, and presenting information about code contributions and code quality, the service will highlight weak points in the company's code and provide actionable steps that can be taken to remedy issues.

6. Future Work

I left Appian and the Development Insights Squad before the Component Health Service was completed, but our squad's next step would have been to construct a frontend for the application, viewable from the Appian Home portal.

Even after the application is entirely completed, it will undergo continual maintenance. Changes to Appian's infrastructure and discovery of bugs will necessitate changes and updates deployed on a rolling basis.

Additionally, the Component Health Service will only be effective if engineers and leaders at the company buy into it. Even if the service provides relevant, accurate, and actionable information, it may prove useless if engineers fail to act on it. For this reason, the squad will need to focus not only on perfecting the technical aspects of the service, but also on advocacy and education. The Development Insights squad will need to make all members of the engineering department aware of what the Component Health Service has to offer and instruct them on how to interpret and act on the data it presents.

7. UVA Evaluation

My coursework in UVA's department of computer science effectively equipped me with the mindset, analytical abilities, and other soft skills I needed to succeed at Appian. My work in CS 3240 Advanced Software Development, provided me with experience working with a team to collaboratively develop software. Courses like CS 2150 Program and Data Representation and CS 4414 Operating Systems taught me how to approach new and unfamiliar code bases and to extend existing projects.

My courses also provided me with several technical skills that proved essential in my internship. My work with git version control, GitHub, and CI/CD infrastructure in CS 3240 was highly relevant to almost everything I did at Appian. My knowledge of the Python programming language, which was strengthened by work in CS 1111 (Introduction to Programming) was also useful.

On the other hand, many core skills I needed in order to succeed over the summer had not been sufficiently covered in my coursework. For example, my classes had only briefly touched on the subject of software testing. While the department does offer a software testing elective course, it is not a requirement for undergraduate students. I believe software testing could be more effectively integrated into the curricula of core CS classes.

Similarly, a stronger knowledge of cloud infrastructure and cloud-based systems would have been extremely helpful. For example, the container orchestration system Kubernetes is widely used across the software development industry, but is difficult to understand and work with as a beginner. I suggest that Kubernetes or a similar tool should be covered and used by students in at least one required class at UVA.

8. Acknowledgments

I would like to acknowledge and thank the members of the Development Insights squad—Kyle Pinkham, Leonid Gaiazov, Jorge Arturo Rivera, Isabelle Stevens, and

David Grossman—who guided me as I took on new and challenging work this summer. I would also like to thank my co-intern, Nikhil Mahajan, for supporting me and collaborating closely over the course of the entire internship. Finally, I would like to thank Rosanne Vrugtman for providing guidance through the process of drafting, editing, and revising this paper.

References

Greiler, M., Herzig, K., and Czerwonka, J. 2015. "Code Ownership and Software Quality: A Replication Study," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, May 2015, 2-12. DOI: https://doi.org/10.1109/msr.2015.8

Sedano, T., Ralph, P., and Péraire, C. 2016. "Practice and Perception of Team Code Ownership," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, Article 36, June 2016, 1-6. DOI: https://doi.org/10.1145/2915970.2916002

Tornhill A,. and Borg, M. 2016. "Code red: the business impact of code quality - a quantitative study of 39 proprietary production codebases," in *Proceedings of the international Conference on Technical Debt*, May 2022, 11-20. DOI: https://doi.org/10.1145/3524843.3528091