Code Ownership in Organizations: Policy, Practice, and Common Problems

A Research Paper submitted to the Department of Engineering and Society

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

Introduction

Within organizations that produce software, *code ownership* refers to which specific developers are allowed to modify and contribute to particular pieces of code (Bird, et. al., 2011). Team code ownership is a specific model whereby portions of a code base are assigned to a particular team or organizational subdivision, and all members of that team are allowed to make changes to any part of the team's code (Sedano et. al., 2016).

Technical Paper 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 could not 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.

STS Research Abstract

My STS research examines the code ownership methodologies currently in use at major technology companies. I draw conclusions about the effectiveness of different approaches to

code ownership for ensuring high code quality while maintaining development efficiency and mitigating developer headaches. I begin with an in-depth survey of existing scholarship. I augment past research with a set of interviews with current software engineers at major technology companies. I develop the following conclusions and recommendations: i) code ownership and code ownership policy are two different things and strong code ownership may be achieved with loose code ownership policy; ii) companies should ensure that developers only contribute to areas of a code base with which they are highly familiar; iii) code ownership policies can create challenges around "inter-team dependencies" which can be resolved with flexible code ownership policy or through intelligent assignment of task priorities. Because code ownership is correlated with software quality, and because software quality has far-reaching impacts, the findings of this paper are broadly relevant to society as a whole.

Literature Review

Several past studies have explored the correlation between code ownership and software quality. Additionally, some qualitative research has focused on organizational strategies within software companies, examining the benefits and drawbacks of collective versus individual code ownership on developer experience and development efficiency.

Defining Code Ownership

There is no universally accepted definition for *code ownership*. Bird et al. (2011) state that "ownership is a general term used to describe whether one person has responsibility for a software component, or if there is no clearly responsible developer." The same study, however, chooses to measure code ownership as the proportion of edits made to a software component by a single developer—a related yet somewhat imprecise indicator of "responsibility." *Collective code ownership* is a loose code ownership model in which anyone (within a team) is allowed to

modify any part of a code base at any time (Ribeiro et al., 2016). Sedano et al. (2016) defines the term *team code ownership* as an approximate synonym to collective code ownership, and the two terms may be used interchangeably in this paper. Both Riberio et al. and Sedano et al. highlight the "collective" or "shared" nature of this ownership model, but it is worth noting that team code ownership can still be quite rigid, with each section of a code base explicitly assigned to a small group of developers.

Effects of Code Ownership on Software Quality

Bird et al. (2011) were among the first to take a highly analytical approach to correlating code ownership with code quality. They measured code ownership based on the proportion of commits made by a particular developer. If a single developer made a high proportion of the edits to a piece of code, then that piece of code was deemed to have high ownership. Conversely, if a large number of developers touched the same component, then it was assumed that there were many "non-experts" working on that component, and it was deemed to have low ownership. The study examined code ownership in Windows Vista and Windows 7 and found that up to 72% of the variance in pre and post-release code failures could be explained by simple code ownership metrics. In other words, the higher the number of minor (non-expert) contributors to a software component, the higher its failure rate.

Foucault et al. (2014) conducted a replication study of Bird et al.'s work, examining code ownership in open-source software. In contrast to the emphatic findings of Bird. et al., they found little correlation between the degree of code ownership and the fault-proneness of the software projects they examined. They conclude that Bird et al.'s findings, which focused on commercial, closed-source software binaries, did not generalize to open-source projects.

Greiler et al. (2015) conducted a second replication study, this time adjusting the focus of their research to smaller software entities. Rather than observing failure rates in large-scale software binaries (namely Windows Vista and Windows 7), they looked at directories, a smaller sub-division of software systems. The study confirmed the findings of Bird et al., detecting positive correlations between the number of contributors to a directory and the number of bugs and defects.

Thongtanunam et al. (2016) argued that Bird. et al. and others missed a critical element of modern development processes by choosing to omit code reviews from their definitions of a "code contribution." They argue that a developer who has extensively reviewed code commits may exert a high degree of ownership over a piece of code without having themselves made explicit line edits. They recreate Bird et al.'s finding that high code ownership is correlated with fewer software defects, but further posit that the proportion of code *reviewers* who lack expertise has a strong, positive correlation with the likelihood of post-release software defects.

Collective Code Ownership and Organizational Strategies

Maruping et al. (2009) conducted a comprehensive study of 509 software developers across 56 software projects. They found that collective code ownership practices allowed for better coordination of expertise within software teams. In other words, by allowing multiple members of each team to examine and contribute to all parts of the code base, developers experienced in certain areas were more likely to be able to apply their particular talents, reducing the number of discovered bugs and improving code quality. They also found that maintaining strict coding standards, defined as a requirement that "all developers write and maintain software code in a common and consistent format," correlated with fewer software defects.

Ribeiro et al. (2016) discussed the advantages and disadvantages of collective code ownership. They referenced and acknowledged Bird et al.'s findings relating code ownership to code quality, but also pointed out that a stricter code ownership policy can create bottlenecks where developers must wait to "get permission" before making changes. They executed a qualitative study in which they interviewed nineteen employees of three different companies and gathered their subjective opinions on code ownership and its impacts on developer experience. Among the advantages they found to collective code ownership were greater member backup (the ability of team members to cover for each other when someone goes on leave or switches teams), greater individual learning, and greater breadth of understanding. Among the disadvantages were increased interpersonal conflict and worse depth of understanding when it came to particular segments of a codebase. Meanwhile, among the interviewees, there was disagreement on whether collective code ownership had a positive or negative impact on code quality.

A qualitative study by Leite et al. (2020) defined four broad organizational structures to which most modern software engineering divisions conform. They were i) siloed departments, ii) classical DevOps, iii) cross-functional teams, iv) platform teams. They posited that siloed department structures, in which collaboration was limited between engineers with different roles, were damaging to productivity and security. This might indicate a preference for more open and collaborative organizational structures, which may include looser, collective code ownership models.

Gap in the Literature: Code Ownership Between Teams

Although extensive qualitative and quantitative research has been done on code ownership and its effects on code quality and developer experience, this work has usually

focused on code ownership within teams. Bird et al. and others measured code ownership purely based on the number of (individual) contributors to a piece of code. Maruping et al. and Ribeiro et al. examined collective code ownership, but only within a single team. They apparently assume that even under a collective code ownership model, developers would only touch code that was owned by their assigned project team. Many questions related to inter-team dynamics remain unanswered. Among these questions are: how should it be determined which portions of a code base are assigned to which team? What specific responsibilities does a team assume when they are assigned as owners to a section of a code base? Should developers ever be allowed to contribute to code not owned by their team? And how should knowledge and/or ownership be transferred when a team is dissolved or when a developer leaves a company? This paper attempts to answer some of these questions as well as provide an overview of which code ownership models are currently in use at major software companies.

Methodology

To augment existing research, I conducted a total of five informal interviews with software engineers with work experience at various software companies. In these interviews, I asked interviewees four open-ended questions (Appendix A) about their understanding of code ownership and how code ownership was organized at their company. I then asked interviewees to describe how often they encountered each of four different "problems" associated with code ownership (Appendix B). The goals of these interviews were as follows:

1) To gain an improved understanding of how modern software companies handle code ownership.
2) To better understand software engineers' feelings around different code ownership policies.

3) To draw loose correlations and form connections between code ownership policies and the type and frequency of developer headaches.

Interviewees were found through personal connections. I attempted to interview engineers at different stages in their career and who had worked at different types of software companies. Between the five individual engineers interviewed, a total of eight different software companies were discussed. In several cases, interviewed developers spoke not only about their current jobs, but also about previous employers. In these cases I asked them specifically to compare and contrast their experiences at each company, which provided valuable insight into the unique problems that can be brought about by different code ownership models.

The data gleaned from these interviews were entirely qualitative. Because each developer had a unique work experience and perspective, it was impossible to gather any sort of quantitative data on their experiences. Furthermore, the low sample size (five developers and eight companies) combined with a biased sample (interviewees were largely individuals to whom I had some personal connection) mean that all conclusions drawn must be taken with a grain of salt. Still, I found that multiple developers had remarkably similar experiences to one another and I could identify definite trends.

Findings

Company	Summary of Code Ownership Model
A	● Team code ownership. ● Edit permissions set programmatically at directory or repository level. ● No formal protocol for knowledge sharing when an individual leaves the company. If a team is dissolved or an individual leaves the company, new owners are assigned, as appropriate, on a case-by-case basis. ● Only vague understanding of what duties come with code ownership.
B	● No formal assignments of code ownership. ● Informal ownership exists at the team level. ● Developers seldom modify code with which they are not highly familiar.

C	● Every piece of code is strictly owned by a single project team. ● Code is very rarely shared between teams and across the company, and as such, the concept of code ownership is rarely discussed.
D	● Team code ownership. ● Edit permissions set programmatically for directories through owner files. ● Responsibilities of ownership are maintenance and issue handling. ● Code owners must sign off on all changes made to code. ● Teams rarely dissolved without code deprecation, avoiding "orphaned code." ● Company-wide spreadsheet assigns each service to a tech lead (owner).
E	● No formal assignments of code ownership. ● Informal ownership exists, but not always made clear. ● Only requirement to make changes anywhere in the company's codebase is sign-offs from any two other engineers.
F	● Team code ownership. ● No universal protocol for how edit permissions are set. Each team is responsible for setting the edit permissions on the code they own. ● Each deployment pipeline has a contact email attached to facilitate locating the owner of each directory. However, emails are not always kept up to date. ● No formal protocol for knowledge sharing when an individual leaves the company, but is handled on a case-by-case basis. ● Internal effort to document code ownership in a spreadsheet.
G	● Team code ownership. ● Currently, no standardized ownership practices across teams. ● Company is small and young.
H	● Every piece of code is strictly owned by a single project team. ● Code is infrequently shared between teams. ● In some cases, code ownership may occur within teams, meaning that permission is required from a single individual to change a piece of code.

Table 1. Summaries of code ownership models at surveyed companies

Overview of Current Code Ownership Methodologies

Table 1 shows a summary of the code ownership policies employed by each company, as discussed in the interview process. Several broad trends are immediately clear. For one, none of the companies surveyed frequently rely on an "individual" code ownership model. Even in the case of Company D, where the interviewee described the code ownership model as "extremely

strict," code is owned collectively by a team. Only at Company H is individual code ownership occasionally used, however the interviewee made it clear that this was not the norm.

The majority of companies surveyed use some form of team code ownership. Differences arise in how strictly and/or clearly ownership is assigned. At Company D, ownership is always assigned very explicitly and ambiguity is rare. At companies A and F, code ownership is assigned formally when possible, but protocols around code ownership are more fluid and ambiguity sometimes exists. At Company B and Company E, code ownership is not assigned explicitly to teams, but informal code ownership may arise. Interviewees from both Company B and Company E explained that they would frequently examine commit histories, identifying potential "owners" based on who had most recently modified a piece of code.

Code Ownership Vs. Code Ownership Policy

Bird et al. and others measured the strength of code ownership based on the number of unique developers touching a piece of code. A possible inference is that stronger code ownership as measured in this way correlates with stronger code ownership policies. In other words, one might guess that formal assignment of teams or individuals to particular repositories might result in fewer editors on each file. My interviews, however, call this notion into question. Companies B and E share a similar "weak" code ownership model, which includes no formal assignments of ownership. Nonetheless, at Company B, informal code owners exist for nearly every repository, and determining which individual or individuals are likely responsible for any given piece of code is rarely a challenge. The interviewee from Company B explained that developers were generally "scared" of breaking code with which they were not familiar, and thus would seldom make changes to repositories with which they did not have a high degree of expertise. In contrast, ambiguity over code ownership is more common at Company E, with an interviewee

explaining that they were frequently unsure to whom any given piece of code belonged and that problems often arose when developers made changes in areas with which they were not very familiar.

Coincidentally, companies A, D, and F all prescribe code ownership at the team level, with each repository formally assigned to a single set of developers. Only at Company D, however, does this code ownership model entirely resolve ambiguity. At companies A and F, assignments of code ownership are sometimes outdated or unclear. At Company F, code ownership is sometimes marked only by an email address attached to a deployment pipeline, and these emails are not always up to date. At Company A, even when ownership is clear, the specific duties that come with ownership are not always clear. Developers are not always made aware of what their specific responsibilities are as owners of a repository. It follows that "strong code ownership" and "strong code ownership policy" are not the same. It is possible to achieve strong code ownership with weak code ownership policies and vice versa.

The Importance of Developer Expertise

An interviewee from Company B emphasized that, although their company did not employ formal assignments of code ownership, developer expertise was an assumed prerequisite to making changes. In other words, no developer would change a piece of code unless they had a high degree of understanding and experience with that piece of code. Similarly, an interviewee from Company A explained that, with increasing seniority and experience, developers were allowed to make changes to larger portions of the company's code base with somewhat less oversight. These practices make sense in the context of findings by Thongtanunam et al. (2016) that "the proportion of reviewers without expertise shares a strong, increasing relationship with the likelihood of having post-release defects." Furthermore, Sedano et al. (2016) found that

developers do not tend to feel that they truly own a piece of code unless they have a strong understanding of how it works. This further suggests a difference between code ownership and code ownership policy. Good code ownership policy should match developers with sections of a company's code base that they are highly familiar with and should discourage contributions by non-expert developers.

Handling Inter-Team Dependencies

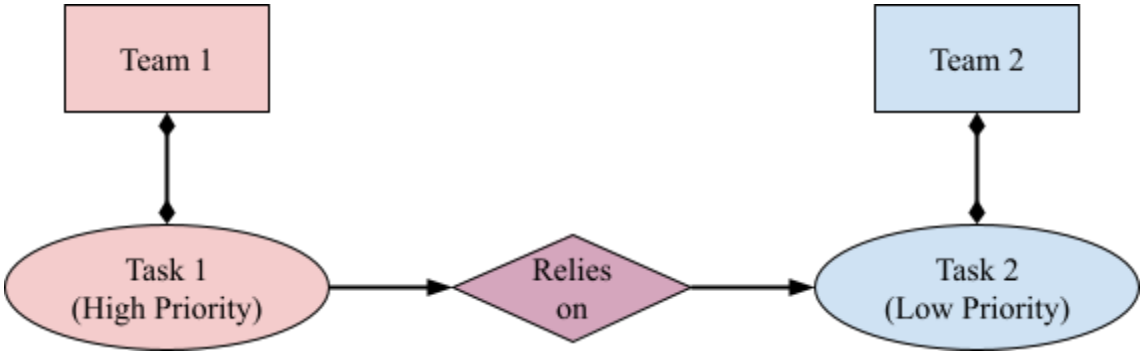

Figure 1. Illustration of a common, problematic, scenario

Figure 1 illustrates a common problematic scenario that was experienced by interviewees from companies A, B, D, F, G, and H. In this scenario, Team 1 has some mission-critical, high-priority task (Task 1) that they wish to complete. Task 1, however, relies on Team 2 completing a separate task (Task 2). Team 2 has little incentive to complete Task 2 because it is low priority (not mission critical) for Team 2. With strict code ownership policies, Team 1 may be stuck since they lack permission to complete Task 2 themselves. Multiple interviewees described the difficult politics that could arise in such a scenario, as members of Team 1 would have to negotiate with members of Team 2 to elevate the priority of Task 2.

In interviews, two possible solutions to this scenario presented themselves. At Company B, a lack of formal code ownership assignments means that a developer on Team 1 can simply inherit the task from Team 2. An interviewee from Company B explained that, while developers

would almost never make changes to code with which they were not highly familiar, they could themselves“take the time to become experts” in a new area of the code base before making changes. This model offers developers on Team 1 a choice: either wait for Team 2 to finish other tasks and turn their attention to Task 2, or take Task 2 off of Team 2’s hands and spend the time to complete Task 2 to a satisfactory standard.

A second solution to this scenario was presented by an interviewee at Company H. At Company H, the priority of each task for each team is set by a “lead developer” who considers the entire company’s long-term goals. In this scenario, developers from Team 1 would go to the lead developer on Team 2 and make a case for why Task 2 should have its priority elevated. Since the lead developer is concerned less with their own team’s progress than with the company’s progress, they may be more likely to agree.

Conclusion

Modern software companies employ various models for assigning and managing code ownership. While some companies opt for strict, one-to-one, or one-to-many assignments of teams to code repositories, other companies employ looser models, allowing developers to touch whichever parts of the company’s code base they deem necessary. I find that companies achieve success when each section of their code base is owned, either formally or informally, by a set of highly knowledgeable and experienced individuals. Furthermore, I find a definite distinction between strong code ownership (a clear correspondence between code and responsible developer) and strict code ownership policy (a formal or official assignment of code to a particular set of individuals). While strict code ownership policy can facilitate strong code ownership, strong code ownership can be achieved without it.

Societal implications

Because code ownership is correlated with software quality, code ownership practices have far-reaching societal implications. Dawson et al. (2010) examined the differences in cost associated with catching and addressing software defects at different stages in the software development lifecycle. They found that it was 100 times more costly to fix software bugs discovered during maintenance than those discovered during the design phase, and six to seven times more costly than fixing bugs discovered during testing.

Historically, code defects have been responsible for major economic failures, threats to cyber security, and have cost human lives. In 2008, a bug in Heathrow Airport's baggage reconciliation software resulted in travelers' bags being misplaced and hundreds of flight cancellations, ultimately costing the airport tens of millions of dollars (Reuters, 2008). A bug in the popular Java-based Log4j library revealed in 2021, allowed hackers to execute arbitrary code on remote servers, potentially compromising hundreds of millions of devices worldwide (Lyngaas, 2021). Although the bug was swiftly patched in a new release of the library, it was the responsibility of individual organizations to update their software to this newer version of the library, meaning that the bug continues to linger and threaten application security to this day. In 2018, a software bug in Uber's self-driving car software resulted in the death of one person when a car detected a pedestrian, but erroneously judged the detection to be a "false positive" (Lee, 2018). These are just a few of the many ways in which faulty software systems have had meaningful, negative impacts on society as a whole.

If we demand robust software, then we must do our best to build organizational systems around software developers that allow them to do their best work. Intelligent assignment and management of code ownership is a critical step in this process.

References

Bird, C., Nagappan, N., Murphy, B., Gall, H., & Devanbu, P. (2011). Don't touch my code! *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering - SIGSOFT/FSE '11*, 4–14. https://doi.org/10.1145/2025113.2025119

Dawson, M., Burrell, D., Rahim, E., & Brewster, S. (2010). Integrating Software Assurance into the Software Development Life Cycle (SDLC). *Journal of Information Systems Technology and Planning 3* , 49–53.

Foucault, M., Falleri, J.-R., & Blanc, X. (2014). Code ownership in open-source software. *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering - EASE '14*, 1–9. https://doi.org/10.1145/2601248.2601283

Greiler, M., Herzig, K., & Czerwonka, J. (2015). Code ownership and software quality: A replication study. *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. https://doi.org/10.1109/msr.2015.8

Leite, L., Kon, F., Pinto, G., & Meirelles, P. (2020). Building a theory of software teams organization in a continuous delivery context. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*. https://doi.org/10.1145/3377812.3390807

Lyngaas, S. (2021, December 14). *US warns hundreds of millions of devices at risk from newly revealed software vulnerability | CNN politics*. CNN. Retrieved October 30, 2022, from https://www.cnn.com/2021/12/13/politics/us-warning-software-vulnerability

Maruping, L. M., Zhang, X., & Venkatesh, V. (2009). Role of collective ownership and coding standards in coordinating expertise in software project teams. *European Journal of Information Systems*, *18*(4), 355–371. https://doi.org/10.1057/ejis.2009.24

Ribeiro, D. M., da Silva, F. Q., Valença, D., Freitas, E. L., & França, C. (2016). Advantages and disadvantages of using shared code from the developers perspective. *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. https://doi.org/10.1145/2961111.2962624

Sedano, T., Ralph, P., & Péraire, C. (2016). Practice and perception of team code ownership. *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*. https://doi.org/10.1145/2915970.2916002

Thomson Reuters. (2008, April 5). *Computer glitch causes Heathrow Terminal 5 woes*. Reuters. Retrieved October 30, 2022, from https://www.reuters.com/article/us-britain-heathrow/computer-glitch-causes-heathrow-terminal-5-woes-idUSL0551899620080405

Thongtanunam, P., McIntosh, S., Hassan, A. E., & Iida, H. (2016). Revisiting code ownership and its relationship with software quality in the scope of modern code review. *Proceedings of the 38th International Conference on Software Engineering*, 1039–1050. https://doi.org/10.1145/2884781.2884852

Tornhill, A., & Borg, M. (2022). Code red. *Proceedings of the International Conference on Technical Debt*. https://doi.org/10.1145/3524843.3528091

Appendix A

Interview Questions about Code Ownership and Code Ownership Policy

1) What does the term code ownership mean to you?
2) At your company, how is it determined who has permission to modify which pieces of code? In other words, how does your company handle code ownership?
3) Would you describe your company as having a strict, moderate, or loose code ownership model? (Define terms for interviewee as necessary).
4) At your company, if the original author of a piece of code has left the organization or has moved to a different team, who is usually responsible for maintaining and updating that piece of code?

Appendix B

Interview Questions about Code Ownership-Associated Problems

How often would you say you encounter each of the following issues related to code ownership? Feel free to provide examples/anecdotes should you wish.

1) You are slowed down in your workflow because you need to get permission or approval to access or modify a piece of code which you do not own.
2) You have a question about a piece of code, but are unsure who to ask because you do not know who at the company is familiar with and/or responsible for that piece of code.
3) You are asked questions about code that you wrote a long time ago and are no longer actively maintaining.
4) Someone else attempts to modify a piece of code that you are very familiar with, but they make mistakes or unwise decisions because they are not very familiar with it.