June, 2016

Practice and Perception of Team Code Ownership

Todd Sedano
Paul Ralph
Cécile Péraire

Practice and Perception of Team Code Ownership

Todd Sedano
Pivotal
3495 Deer Creak Road
Palo Alto, CA
professor@gmail.com

Paul Ralph
University of Auckland
Auckland
New Zealand
paul@paulralph.name

Cécile Péraire
Carnegie Mellon Unveristy
Silicon Valley Campus
Moffett Field, CA 94035, USA
cecile.peraire@sv.cmu.edu

ABSTRACT

Context: Team code ownership is a software development practice where any team member can modify any part of the team's code. However, many factors beyond official policy affect a developer's sense of ownership.

Objective: The purpose of this paper is to understand the factors that affect a team's sense of code ownership.

Method: Following Constructivist Grounded Theory, the first author conducted participant-observation of several software development projects, and interviewed 21 software engineers, interaction designers, and product managers. Iterating between theoretical sampling and analysis continued until achieving theoretical saturation.

Results: Team code ownership is a feeling. Developers feel team code ownership more when they understand the system context, have contributed to the code in question, perceive code quality as high, believe the product will satisfy the user needs, and perceive high team cohesion.

Limitations: Outcomes of grounded theory research are not statistically generalizable to defined populations, and may not apply to organizations with different software development cultures.

Conclusion: Team code ownership is rooted in numerous cognitive, emotional, contextual and technical factors and cannot be achieved simply by policy.

CCS Concepts

•Software and its engineering → Collaboration in software development;

Keywords

Extreme Programming, Collective code ownership, Team code ownership, Sustainable software development

1. INTRODUCTION

Team Code Ownership (which is similar to *Collective Code Ownership* and *Shared Code*) is a software development practice where any developer on a team has the right to change any of the team's code. Team code ownership is intended to accelerate development by allowing any developer to fix any team bug and by mitigating delays due to vacations, illness and other absence [3].

While some research has investigated the effects of different code ownership models, we are unaware of any studies that specifically investigate developers' sense of team code ownership; that is, the complex interactions between developers' knowledge, emotions, and approach to code ownership.

When team code ownership emerged as a core category in a grounded theory study, we therefore exploited this opportunity to investigate factors associated with perceived code ownership and related phenomena.

We quickly discovered that having the right to change a file does not mean that a specific developer will feel empowered to and justified in making a specific change. For example, a developer may feel reluctant to change code that he or she does not really understand. As we refined this core finding and allowed it to drive further data collection, we identified five factors associated with feelings of team code ownership.

This paper consequently reviews existing research connected to team code ownership (Section 2), describes our grounded theory approach (Section 3), and presents our emerging results: five factors associated with team code ownership (Section 4). Section 5 discusses the study's implications and limitations, followed by a summary of its contributions (Section 6).

EASE '16, June 01 - 03, 2016, Limerick, Ireland

ISBN 978-1-4503-3691-8/16/06…$15.00

DOI: http://dx.doi.org/10.1145/2915970.2916002

2. RELATED WORK

2.1 Team Code Ownership

In Extreme Programming [3], Kent Beck describes a set of interdependent practices for managing feature development and facilitating a collaborative team environment. One of these practices is *collective ownership*—"Anyone can change any piece of code in the system at any time." [2]. The book contrasts collective ownership against "no ownership" and "individual ownership." In 2004, collective ownership is renamed **shared code** [3].

In 2006, Martin Fowler defined *collective code ownership*, similarly to Beck [8], as a contrasting team position to "strong code ownership" where each file has one owner and "weak code ownership" where developers can change files, but an owner keeps an eye on files for which they are responsible.

Later, Bird et al. [4] contrasted the effects of strong- and

weak-ownership. They demonstrated that weak ownership leads to more defects than strong ownership for Windows Vista and Windows 7. The study defined ownership for a software component as a percentage of the version control commits for a single developer. They defined a major contributor as someone who has more than 5% of the git commits. A sensitivity analysis revealed that defining strong code ownership within the range from 2% to 10% produced similar results for the study.

Meanwhile, Murphy [14] argued that the concept of code ownership must be unpacked and expanded. He argued that the complexities of code ownership are missed by merely examining git commits to determine who modified which files.

Our paper renames *collective code ownership* to *team code ownership*. For small systems and teams, these terms are synonymous. For a large system with multiple teams, in practice, teams would have strong ownership of their portion of the system. Allowing any pair to modify any part of Microsoft Windows or Pivotal Cloud Foundry is impractical.

Team code ownership requires more than a team saying, "everyone can modify anything." Instead, this paper examines how a team feels that they own the code. We define "sense of team code ownership" as the degree to which individual members of the team feel collective ownership.

2.2 Sustainable Software Development

We describe the theory of Sustainable Software Development through Overlapping Pair Rotation in the paper by the same name [16] and summarize it in Table 1. The theory describes how teams can continue to deliver value in spite of team disruptions. The theory is a collection of synergistic principles, policies, and practices encouraging a positive attitude towards team disruption, knowledge sharing and continuity, as well as caring about code quality. The practices actively remove knowledge silos and caretake the code so that any pair can work on any story in the backlog.

2.3 Psychological Ownership

Psychological ownership refers to "the feeling of possessiveness and of being psychologically tied to an object" [15]. Targets of ownership, whether physical or immaterial, become the extension of one's self: "What is mine becomes (in my feelings) part of ME" [11]. Ownership can be attached to a part or the whole. Psychological ownership occurs when the object becomes part of the psychological owner's identity. Psychological ownership answers the question, "What do I feel is mine?"

Changes in ownership can have strong effects on our self-identity. An increase in the number of possessions can produce positive effects [7], while a diminish can lead to a personality shrinkage [12]. Someone threatening a person's ownership can trigger strong emotions and responses.

Peirce [15] identifies three sources or "roots" of psychological ownership: efficacy and effectance, self-identity, and having a place. A major reason for possession of physical goods or abstract ideas is rooted in the innate human desire to be in control; being able to alter one's environment creates feelings of efficacy and pleasure. Ownership fulfills the need for self-identification as people define themselves, express themselves, and ensure their own survival by what they own. Ownership fulfills the need to have a place and a territory to possess. "Each motive facilitates the development of psychological ownership, rather than directly causes this state to occur." Psychological ownership occurs with code because creating software can satisfy the desire for efficacy and effectance, self-identity, and having a place.

Peirce identifies three paths or "routes" to ownership: controlling the target, coming to intimately know the target, and investing the self into the target. With *controlling the target*, targets that can be controlled are perceived to be part of the self. As individuals repeatedly exercise control of an object, eventually this leads to "feelings of ownership toward that object." The higher the autonomy of the job task, the more likely ownership develops toward the activity. When a person has little control over an activity, psychological ownership is unlikely to develop. With *coming to intimately know the target*, the association with the object creates feelings of ownership. One example is when a gardener feels that the garden belongs to the gardener. (This happens routinely with software developers who feel that they own part of the code base, when in reality, the company owns the software.) Feelings of ownership increase as one becomes intimately familiar with the object and associated with it. With "investing the self into the target," we feel that we own what we create, shape or produce. Spending time, energy, and effort enables us to alter our view of ourselves to include identity with the object. The more investing in the object, the stronger the psychological ownership. Nonroutine, complex jobs infuse more of our own ideas resulting in increased ownership.

3. RESEARCH METHOD

3.1 Constructivist Grounded Theory

We followed Charmaz' approach to Grounded Theory [6], which provides an iterative approach to data collection, data coding, and analysis resulting in an emergent theory. The two primary data sources were field notes collected during continuous participant observations of a 7.5-month project and interviews with 21 Pivotal software engineers, interaction designers, and product managers. Interviews were recorded, transcribed, coded, and analyzed using constant comparison. Our presentation is informed by Stol et al.'s reporting guidelines for grounded theory studies in software engineering [17].

When starting a grounded theory research study, the core question is "What is happening here?" (Glaser, 1978) [9]. Our initial core question was: "What is happening at Pivotal when it comes to software development?" This question led to the Theory of Sustainable Software Development summarized in Section 2.2. When team code ownership emerged as one of the core categories of the theory, the researcher collected additional data in order to identify the factors affecting the sense of code ownership. The factors are introduced in Section 4 and are the main contributions of the paper.

3.2 Data Collection

The primary researcher relied on "intensive interviews," which Charmaz summarizes as "open-ended yet directed, shaped yet emergent, and paced yet unrestricted" [6]. The technique relies on open-ended questions. The purpose is for the researcher to enter into the participant's personal perspective within the context of the research question.

While exploring new emergent core categories, whenever possible, the researcher initiated subsequent interviews with

Table 1: Theory of Sustainable Software Development: Principles, Policies, and Practices

Sustainable Software Development			
Underlying Principles	Policies	Removing Knowledge Silos Practices	Caretaking the Code Practices
Keeping a Positive Attitude Toward Team Disruption	Team Code Ownership	Continuous Pair Programming	TDD / BDD
Encouraging Knowledge Sharing and Continuity	Shared Schedule	Overlapping Pair Rotation	Continuous Refactoring
Caring about Code Quality	Avoid Technical Debt	Knowledge Pollination	Supported by Live on Master

a goal of not forcing the issue. For example, "please draw your feelings about the code" often resulted in conversations about code ownership. After the interview, the interview was transcribed into a Word document with timecode stamps for each segment.

The primary researcher collected field notes while working as an engineer. The field notes comprise multiple paragraph entries recorded several times a week collected over a six month period. The notes describe individual and collective actions, captures what participants defined as interesting or problematic, and include anecdotes and observations.

3.3 Research Context: Pivotal

Pivotal is a large American company with 16 offices around the world. One of its divisions is Pivotal Labs. Pivotal Labs' mission is to both deliver highly-crafted software products and provide a transformative experience for their client's engineering cultures. To change a developer's way of working, Pivotal combines the client's software engineers with Pivotal's engineers at a Pivotal office where they can experience Extreme Programming in an environment conducive for agile development.

A common team size is six developers plus an interaction designer and a product manager. In the history of the Palo Alto office, the number of developers on a project ranges from 2 to 28. Larger projects are organized into smaller coordinating teams with one product manager per team and one or two interaction designers per team.

Pivotal Labs has followed Extreme Programming [3] since the late 1990s. While each team is autonomous in making its own decisions as to what is best for a particular project, the company culture strongly suggests following all of the core practices of Extreme Programming.

4. TEAM CODE OWNERSHIP

In the literature, collective code ownership is often treated as a policy statement. In this case, simply claiming that "anyone can modify any piece the code" was not sufficient to engender willingness to modify any file. Rather, ownership is an emotional or qualitative attribute that ties all developers on the team to the project and code base. It is a spectrum where, on one side, each individual has ownership of only their code, and on the other side, everyone on the team owns the entire code base. Some events appear to erode the team's sense of ownership over the project's duration, while some practices appear to counteract these erosions. This section details the five factors that appear most related to team code ownership and examples of events or tendencies that erode it.

4.1 System Context

Definition: System context is the knowledge and situational awareness about the code, including the discourse that surrounds the code. System context includes understanding existing design decisions, underlying technologies, the relationship between features and user needs, and the implementation of existing features.

Purpose: Developing an in-depth knowledge of the system exercises the "intimately knowing the target" path of psychological ownership.

For a pair to work efficiently on any part of the system, one of them needs to have enough context to know how that part of the system works. Without enough context, a pair might struggle, slow down, or be blocked in working on a feature.

Code ownership seems to vary with the context that the developer has about the code; the more the developer knows, the higher the sense of ownership. Knowledge silos, the size of the code base, or the number of developers working in parallel can make it difficult for a programmer to develop a deep system context level.

Threat: Increasing knowledge silos. When developers routinely work on one part of the code base, they can develop specific system context not shared by the team. Code specialization impedes anyone on the team from modifying any part of the team's code. One team said *"We need Marion on that story, only she knows the Apple watch code base,"* and *"Shea knows the ins-and-outs of the legacy integration, we need him to work on this story,"* which means there is a hindering imbalance between the individual and team understanding of the code.

Threat: Increasing code base size. The primary researcher participated on a team working with a large code base that was over eight years old and the team did not have a full understanding of the system. Initially, the team felt little ownership of the code, even though the team was responsible for it and agreed to 'team code ownership.' Often the team would need to ask a product manager why certain features exist in the code to understand the code's purpose and implementation. In time, as the team worked with the code and gained context, the team's sense of ownership improved.

Threat: Increasing team size. The primary researcher observed the relationship between team size and code context on five Pivotal projects as a participant-observer. As team size increases, the ability to gain system context decreases. Every day, all pairs are adding to the system. On a five pair team, so much work is happening each day that it becomes increasingly difficult to keep track of everything

that changes.

One developer on a ten-person project said, *"I feel that we don't have the context spread around fully. Having five, sometimes six, pairs on the project makes it go really fast, so it's hard to keep context."*

When developers do not have context about part of a system, or context about what remains to be done to finish a story, reluctance to start the next story at the top of the backlog emerges. It's easier to start a story that touches part of the system that they know. As one developer reflected, *"I am not entirely comfortable to jump into stories on certain aspects [of the system]."*

As a coping strategy, one developer, before the start of the work day, skimmed the git commits from the previous day to learn about new classes and changes in design and to understand the features the team added.

As team size grows, there is a potential risk of decreasing an individual developer's sense of team code ownership.

4.2 Code Contribution

Definition: Code contribution is the portion of the code that a given developer has worked on.

Purpose: Personally contributing to the code base increases a developer's sense of ownership by exercising "investing in the target" path of psychological ownership.

As a developer works on the code base, the developer's system context level increases. While code contribution level influences the system context level, it is not necessary related: developers might learn about the code through other means different from direct contribution, including conversations at stand-up, impromptu team huddles, or a pair saying *"Check out what we did yesterday."*

Threat: Inability to contribute. A developer's inability to contribute to the code base decreases the developer's sense of ownership.

This could happen, for instance, during a pair programming breakdown. When the pairing experience breaks down, one person drives the code development while the partner passively watches. (We call this dynamic "Performance Pair Programming," when one developer plows through a story and stops listening to the developer's partner.) When one person is writing all the code, individual code ownership replaces team code ownership.

In one situation, the partner took over and ignored the participant's input. The participant reflected, *"I would not be able to explain deeply what we had done. I would not be able to maintain it. I didn't really write it, so I feel very little ownership of it."*

Ideally, Pair Programming is a collaborative experience where both individuals are unable to tell who wrote which portions of the code.

4.3 Code Quality

Definition: Code quality relates to how well the code satisfies the project's desirable quality attributes. Desirable quality attributes might include design qualities, performance, reliability, scalability, security, testability, and usability [13].

Purpose: A high quality product satisfies the self-identity motivation of psychological ownership. Developers might not want to be identified with a low quality product.

Low-quality products also tend to involve a disproportionate amount of bug fixes. Developers need a balance between creating new features and fixing bugs each week. Working only on bugs for weeks affects their sense of ownership.

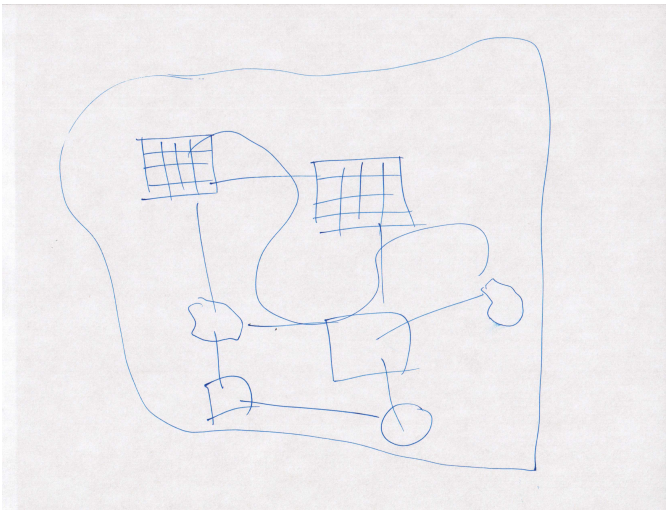

Figure 1: "Draw how you feel about the code"

Threat: Pressure to deliver and deprioritizing continuous refactoring. When developers are pressured to deliver more features at the expense of Continuous Refactoring, the code acquires technical debt, the code becomes more difficult to work with, and developers can begin to feel indifferent about the code. When developers begin to experience code apathy, this decreases their sense of team code ownership.

When the team neglects refactoring, new code is simply bolted onto the existing design. Each time the team bolts something else on, bolting on the next piece becomes more complicated. Thus, a dilemma arises for the programmers working on the next story that touches this part of the code: do they continue bolting on more code, or do they perform the pretermitted refactoring? A team's avoidance of refactoring may be a sign that code apathy is settling in. Code apathy results in reduced quality, as the developers become less invested in the craftsmanship of the code.

One developer felt *"proud and disgusted"* about the code base. He is simultaneously proud of each refactoring that the team performed and disgusted by the technical debt the team accrued by taking shortcuts to ship more features. The developer drew Figure 1 to show his feeling about the code, *"It is generally orderly with a few bits that maybe are not as orderly."*

Before the first launch of a product, the product manager suggested that the team deliver more features at the expense of technical debt. For some of the team, this was an unacceptable tradeoff, and those developers decided not to cut corners. Others on the team complied with the request and incurred technical debt. The entire team ended up paying the consequences with extensive refactors after the launch. On a communal code base, one pair adding tech debt affects everyone on the team.

When code apathy settles in, team members adopt the attitude that someone else will solve the problem with the code. When this attitude permeates a team, no one is solving the problems.

The team wants to feel pride in improving code quality. It

feels good to be improving the code design and readability. If the team starts neglecting these concerns, it can engender a sense of disgust and apathy for the code can spread throughout the team.

4.4 Product Fit

Definition: Product fit is developers believing that features of the product will satisfy the user's needs.

Purpose: Engineers want to create products that matter to the users. Delivering a product that matters to someone satisfies the self-identity motivation of psychological ownership.

Threat: Ignoring user feedback. When the product manager ignores feedback from user research and usability testing, developers may lose faith in the product's ability to achieve its goals. Developer motivation and engagement can decrease when developers perceive they are building a feature that users have explicitly said they do not want yet is built to solve a business goal.

Threat: Ignoring developers feedback about the product. Pivotal's balanced team approach is founded on collaboration between product managers, interaction designers, and developers. When product managers or other stakeholders ignore feedback from developers, developers can begin to feel less ownership in the product, and in turn, be less motivated to work on the project.

Feature apathy or product apathy can result in a poorly crafted product that does not meet the customer's needs.

4.5 Team Cohesion

Definition: Team cohesion is the degree to which team members identify as part of the team, stick together through adversity and take pride in the team's accomplishments [5, 1, 18].

Purpose: Team cohesion satisfies the "having a place" motivation of psychological ownership.

Threat: Distancing a developer from the team. Team apathy manifests when developers do not feel that they are a part of the team. Developers feel less ownership of the code base when they feel excluded from the team.

We observed several behaviors that can distance a developer from the team: interrupting the developer during discussions, using poor listening skills so that the developer feels unheard, or talking beyond the developer's level of technical expertise.

On one team, during discussions, the team talked about code but never looked at the source code. One developer found these abstract discussions difficult to follow. Sometimes the team discussed parts of the code that the individual had not seen recently. When the team discussed two variants of coding practices without showing concrete examples, the programmer could not contribute. When the developer raised this issue to the team and the team continued with the status quo, the programmer felt marginalized by the team.

Poor onboarding of developers can contribute to feelings of isolation. On one project, there was a time crunch and the team was feeling the pressure to deliver stories. When the team added developers, the team had a "sink or swim" attitude, letting new team members figure things out on their own, hence making them feel unwelcome.

When developers feel that the team does not care about them, their sense of ownership can decrease.

5. DISCUSSION

5.1 Transitioning to team code ownership

The above results have numerous implications for teams attempting to transition to team code ownership.

Some developers effortlessly make the transition to team code ownership. They immediately see the benefits of being able to modify any part of the code base and quickly shift from "I made this" (personal ownership) to "we made this" (collective ownership.)

Others may struggle with team code ownership for several reasons:

- Developers may struggle to transition to a caretaker mindset. In one interview, a software engineer struggled to describe the developer's relationship with the code on a very challenging project and settled in on the caretaker metaphor: *"Sometimes I kind of feel like a janitor to [the code base]. Maybe caretaker would be better. Yeah, probably caretaker. I feel like a janitor just cleans up messes, but a caretaker makes things better."*
- A developer may be distraught at *"seeing my work slowly removed from the app."*
- Developers can no longer take pride in functionality that they exclusively develop.
- Existing knowledge silos, which hinder team code ownership, may be slow to break down.

New hires struggling with the transition slowly realize that *"Someone else is going to take over and they're going to do fine. I can move onto something else and that's okay."* They recognize the lack of long-term individual authorship, learn to expect their code to be transitory, develop trust in their teammates and thus loosely hold personal contributions. *"The code that I write today may be in the code base for a little while, and it will evolve into something better."* Eventually, they experience the benefits of a collaborative environment: *"People are a lot more flexible all across the board, with changing things or accepting feedback or collaborating,"* and the team can say "Hey, this is our code!"

Shifting from individual to team code ownership may requires multiple and complementary practices to actively remove knowledge silos. In this case, daily pair rotation helped combat knowledge silos. Moreover, for developers with strong individual ownership tendencies, sharing ownership first with a small group (where trust and communication come easier) may help. One Pivotal engineer uses improvisation and collaboration games to help teams practice letting go of control, trusting the team, and learning to be pleasantly surprised by what emerges.

5.2 Results Evaluation

The factors influencing team code ownership presented in Section 4, have emerged from the Grounded Theory research study introduced in Section 2.2. While other factors may influence team code ownership, we focus only on those that were observed during the study. Grounded Theory studies can be evaluated using the following criteria [6]:

Credibility: The 21 intensive open-ended interviews and numerous field notes from participant-observation serve as a rich and credible data set for the analysis.

Originality: The paper broadens the idea of team code ownership by acknowledging that collective code ownership is more than a policy statement, and by uniquely identifying factors that affect the team's sense of code ownership.

Resonance: Several participants reviewed our findings and indicated that both the factors and threats resonate with their experience.

Usefulness: The study identifies factors associated with ownership and suggests several ways of engendering team code ownership.

This work analyzed software projects at the Silicon Valley office of Pivotal following Extreme Programming. From an **external validity** perspective, grounded theory is non-statistical, non-sampling research. Our results therefore cannot be statistically generalized to a population. Rather, researchers and professionals can adapt the concepts and ideas to other contexts case-by-case.

Finally, our results might be influenced by **researcher bias** or **prior knowledge bias**. A risk of the participant-observer technique is that the researcher may lose perspective and become biased by being a member of the team. While a participant-observer gains perspective an outsider cannot, an outside observer might see something a participant observer will miss. Similarly, while prior knowledge helps the researcher interpret events and select lines of inquiry, prior knowledge may also blind the researcher to alternative explanations [10]. We mitigated these risks by recording interviews and having the second and third authors review the coding process.

6. CONCLUSION

This paper reports results from a participant-observation, constructivist grounded theory study at Pivotal, a large American software company employing Extreme Programming practices. It provides three main contributions.

1) Our observations clearly indicate that **team code ownership is a feeling to be engendered not a policy to be decreed**.

2) Meanwhile, both discussions with and observations of participants suggest five factors associated with strong feelings team code ownership. Pivotal developers more acutely feel team code ownership when i) they understand the system context; ii) they have contributed to the code in question; iii) they perceive code quality as high; iv) they believe the product will satisfy user needs; and v) they perceive team cohesion as high.

3) Moreover, diverse events and trends can undermine sense of ownership, including: increasing knowledge silos, increasing code base size, increasing team size, inability to contribute, pressure to deliver and deprioritizing continuous refactoring, ignoring user feedback, ignoring developer feedback, and distancing a developer from the team.

In conclusion, Pivotal's developers find team code ownership highly advantageous; however, transitioning to a team code ownership model is easier for some than others. Some agile practices including continuous pair programming, overlapping pair rotation, continuous refactoring, and test driven development appear to help. Promising angles for future research include more nuanced explorations of the code ownership spectrum, further exploration of the roles of emotion and identity, as well as developing specific practices for facilitating ownership transitions.

Thank you to Rob Mee, David Goudreau, Ryan Richard, and Zach Larson for making this research possible.

7. REFERENCES

[1] D. J. Beal, R. R. Cohen, M. J. Burke, and C. L. McLendon. Cohesion and performance in groups: a meta-analytic clarification of construct relations. *Journal of Applied Psychology*, 88(6):989, 2003.

[2] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 2000.

[3] K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[4] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu. Don't touch my code!: Examining the effects of ownership on software quality. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*. ACM, 2011.

[5] K. A. Bollen and R. H. Hoyle. Perceived cohesion: A conceptual and empirical examination. *Social forces*, 69(2):479–504, 1990.

[6] K. Charmaz. *Constructing Grounded Theory*. SAGE Publications, 2014.

[7] R. Formanek. Why they collect: Collectors reveal their motivations. *Interpreting objects and collections*, 1994.

[8] M. Fowler. Code ownership, 2006. URL: http://martinfowler.com/bliki/CodeOwnership.html.

[9] B. Glaser. *Theoretical Sensitivity: Advances in the Methodology of Grounded Theory*. Sociology Press, 1978.

[10] B. G. Glaser. *Doing Grounded Theory: Issues and Discussions*. Sociology Press, 1998.

[11] S. Isaacs. Social development in young children. *British Journal of Educational Psychology*, 1933.

[12] W. James. *The Principles of Psychology*. Holt, 1980.

[13] J. Meier, D. Hill, A. Homer, T. Jason, P. Bansode, L. Wall, R. Boucher Jr, and A. Bogawat. *Microsoft Application Architecture Guide*. Microsoft Press Book, 2009.

[14] B. Murphy. Code ownership-more complex to understand than research implies. *Software, IEEE*, 32(6):19, Nov 2015.

[15] J. L. Pierce, T. Kostova, and K. T. Dirks. Toward a theory of psychological ownership in organizations. *Academy of Management Review*, 26(2):298–310, 2001.

[16] T. Sedano, P. Ralph, and C. Péraire. Sustainable software development through overlapping pair rotation. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement International Conference on Software Engineering*, ESEM, 2016.

[17] K.-J. Stol, P. Ralph, and B. Fitzgerald. Grounded theory in software engineering research: A critical review and guideline. In *Proceedings of the 2016 International Conference on Software Engineering*, ICSE '16, 2016.

[18] E. Whitworth and R. Biddle. Motivation and cohesion in agile teams. In *Agile Processes in Software Engineering and Extreme Programming*. Springer, 2007.