May, 2017

Lessons Learned from an Extended Participant Observation Grounded Theory Study

Todd Sedano
Paul Ralph
Cécile Péraire

Lessons Learned from an Extended Participant Observation Grounded Theory Study

Todd Sedano
Pivotal
Palo Alto, CA, USA
Carnegie Mellon University
Silicon Valley Campus
Email: professor@gmail.com

Paul Ralph
University of Auckland
Auckland, New Zealand
University of British Columbia
Vancouver, BC, Canada
Email: paul@paulralph.name

Cécile Péraire
Carnegie Mellon University
Electrical and Computer Engineering
Silicon Valley Campus
Moffett Field, CA 94035, USA
Email: cecile.peraire@sv.cmu.edu

Abstract

Context: Conducting a Grounded Theory study is rigorous, demanding, and challenging. Misperceptions exist within the software engineering community [1].

Objective: The purpose of this paper is to describe one extended participant observation Grounded Theory study for aiding new empirical researchers wanting to run similar research studies.

Method: Following Constructivist Grounded Theory, we conducted a two-year five-month participant-observation of eight software development projects at Pivotal, a software development organization; interviewed 33 software engineers, interaction designers, and product managers; and analyzed one year of retrospection topics. We iterated between analysis and theoretical sampling until achieving theoretical saturation, publishing papers on team code ownership [2], sustainable software development through overlapping code ownership [3], and software development waste [4].

Results: This paper describes the missteps, challenges, and unique insights that occurred while conducting a Grounded Theory study.

Limitations: While the results are highly relevant to the researcher, the outcomes might not apply to other researchers.

Conclusion: Conducting my own Grounded Theory research study, attending Glaser's Seminar, and reading and rereading Charmaz's and Glaser's books helped the researcher overcome misperceptions about Grounded Theory research.

I. Introduction

For the past 2.5 years, I [1] have been conducting a full-time, participant observation, Grounded Theory study. Participant observation is a data collection method where the researcher participates in the project or organization being observed [5]. Grounded Theory is a comprehensive research method that generates theory from observations [6].

My first introduction to Grounded Theory came while reading Brown [7] in March 2013. Brown details her journey in applying Grounded Theory to her psychological research about processing shame. Grounded Theory's focus on interpersonal communication appealed to me. Later, in May, I attended the first International Workshop on Conducting Empirical Studies in Industry (CESI 2013) and quizzed attendees about how they used Grounded Theory in their research. During the main conference, The 35th International Conference on Software Engineering, I attended each Grounded Theory presentation to learn more about how the community used the method.

Lengthy participant observation studies are unusual in software engineering and in computer science more generally. Moreover, software engineering contexts present unique, unresolved challenges for longitudinal participant observation and Grounded Theory [1]. The purpose of this paper is therefore to offer guidance, based on my experiences, for researchers considering a similar trajectory. While we should avoid overgeneralizing from a sample of one, others may benefit from adopting some ideas and avoiding possible risks.

Grounded Theory is such a simple method, yet so easily and often misapplied [1]. It is easy to describe but hard to understand. My Grounded Theory study appears typical: confusion followed by insight. A researcher starts a Grounded Theory study open to see where the research leads. Once unleashed, the method seems to have a mind of its own as it guides the researcher towards research treasure through uncharted territory. I've been teaching and performing Extreme Programming for over 11 years, yet the method delighted me by finding precious insights.

The structure of the paper follows the stages of a Grounded Theory study, Section II "Getting started," Section III "Constant Comparison," Section IV "Theory Construction," Section V "Theoretical Saturation, " Section VI "Managing The Data," Section VII "Publishing," Section VIII "Advantages and Disadvantages of Extended Participant Observation," and Section IX "Conclusion."

II. Getting Started

Following Glaser's advice of "just do it" [8], I learned the intricacies of Grounded Theory by diving into a study and reading about the method as I went. This section examines

[1]We have written this paper in the first person because it predominantly recounts the experience of the first author and because its narrative elements flow better this way.

early important decisions such as which variant to use, selecting a site, and gathering rich data.

A. Selecting a Method Variant

Grounded Theory has many variants with both practical and philosophical differences. Initially, I chose Constructivist Grounded Theory simply because it was the latest, a fine justification for selecting software libraries, but as I later discovered, not the best reason for selecting a sociological research method.

I continue to use Constructivist Grounded Theory for several reasons. Glaser's recommendations are not always practical (or acceptable in our community of practice); for instance, simply taking notes without recording/transcribing interviews and delaying the literature review until near the end of the study. Furthermore, a constructivist epistemology offers many advantages over Classic Grounded Theory's objectivist epistemology for software engineering contexts. Software development is a socio-technical endeavor. Many concepts software engineering researchers study are socially constructed (e.g. code, comments, tests, projects, schedules, teams). Moreover, software developers can be quite reflective and helpful in contributing to the construction of concepts. Constructivist Grounded Theory better acknowledges socially constructed concepts and participants' contribution to theorizing than many other Grounded Theory variants.

Recommendation: If you want just to get started, begin with Stol's comparison of the three variants [1], read a few exemplars (e.g. [3], [4]), and then read Charmaz's book [9]. If you want more depth about each variant, read Evan's article [10]. Note that Evan includes Glaser's pontificating criticisms of Constructivist Grounded Theory [11], but excludes Bryant's rebuttal [12].

My primary text was Charmaz [9], which I supplemented with Glaser's books [6], [13], [8] to build a deeper understanding of the method and the historical context of Constructivist Grounded Theory. I find Charmaz's writing more approachable and less polemical than Glaser's.

Lesson Learned: I initially read Glaser's books in the wrong sequence. His works are additive and build on each other.

Recommendation: Read Glaser's books in order of publication, oldest first.

B. Selecting a Research Site

Pivotal Labs is a division of Pivotal—a large American software company with 17 offices around the world. Pivotal Labs provides teams of agile developers, product managers, and interaction designers to other firms. Its mission is not only to deliver highly-crafted software products, but also to help transform clients' engineering cultures. To change the client's development process, Pivotal combines the client's software engineers with Pivotal's engineers at a Pivotal office where they can experience Extreme Programming [14] in an environment conducive to agile development. Pivotal Labs has followed Extreme Programming [14] since the late 1990s.

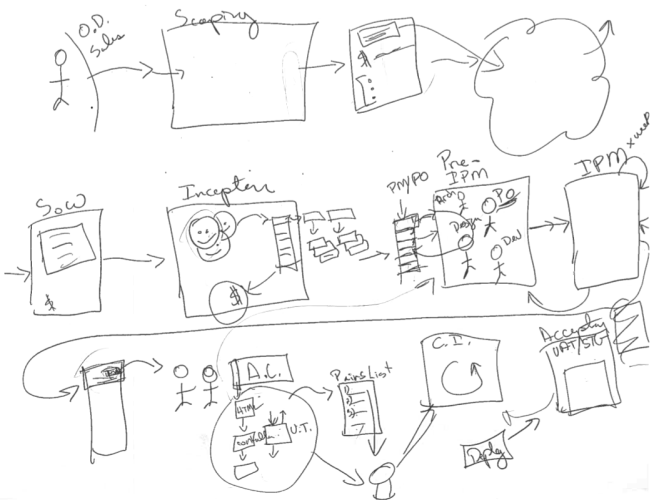

Fig. 1: Interview 6: Pivotal's Software Development process

We selected Pivotal because: 1) it is successful; 2) it is interesting in its continued use and evolution of extreme programming; 3) it is accessible and cooperative with research. Both Classic and Constructivist Grounded Theory advocate picking an interesting site to see "What's going on here?"

Lessons learned: Selecting a consultancy provided exposure to many different projects, both greenfield and brownfield. Pivotal was open to research, provided we did not reveal client information. Pivotal Labs' business goals aligned with gaining insights about the way of working. Clients own the code, whereas Pivotal teaches its way of working. Client-Pivotal dynamics revealed tensions in adopting agile software development. However, the consultancy's focus on billing 40 hours per person per week limited research activities (e.g. data collection, data analysis, and attending conferences) to personal time.

Recommendation: Since Grounded Theory starts with the question "what is going on here?," start with an organization that excels at what you hope to research. Consultancies can work well.

C. Gathering Rich Data

Initially, the two main data sources were interviews with Pivotal employees and notes from participant observation. I began with no solid expectations about the number of interviews or amount of participant observation that I would conduct. The interviewees consisted of interaction designers, product managers, and software engineers who had experience with Pivotal's software development process from five different Pivotal offices. At first, I selected interviewees opportunistically; for example, when a product manager visited from another office, I requested an interview. When I visited the Los Angeles area, I scheduled interviews at the Santa Monica office.

Lesson learned: Opportunistic interviewing sometimes generated an analysis backlog.

Recommendation: Prioritize analysis over data collection.

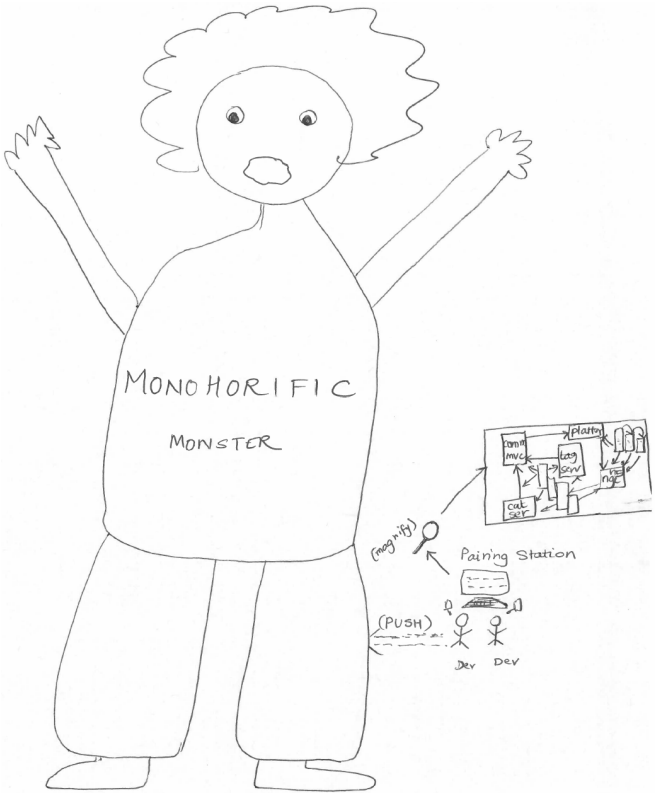

Fig. 2: Interview 31: Software Engineer's drawing for "How do you feel or how do you think about the code?"'

I relied on "intensive interviews," which are "open-ended yet directed, shaped yet emergent, and paced yet unrestricted" [9], to abandon assumptions and better understand the interviewee's perspective.

Lesson Learned: Creating an interview guide helped me generate open-ended questions. Transcribing and coding my interviews enabled me to hear when I asked leading questions.

Recommendation: Use an interview guide at most three times between revisions. This reinforces constant comparison and theoretical sampling. When you learn how to formulate open-ended questions dynamically during an interview, replace the interview guide with one open-ended question.

I asked early interviewees to draw their "view of Pivotal's software development process." Figure 1 shows one response. When analysis shifted interviewing onto new topics, I would start with a new drawing question. For example, asking participants, "please draw your feelings about the code" often resulted in conversations about code ownership. Figure 2 shows an example.

Lesson Learned: Asking participants to draw helped understand their perspective and provide a natural starting place for follow-up questions without forcing topics or perspectives.

Recommendation: Try initiating interviews with a drawing question.

Meanwhile, I worked as a software engineer on eight sequential projects lasting two-years and five-months (see [4] for details). I wrote extensive field notes on individual and collective actions, what participants found interesting or problematic, and emerging anecdotes and observations; for example:

"Monday was the first time we started getting data from the client's servers. We've been implementing for several months without knowing whether our system would work with real backend systems. In order to make progress, we created mocks for each of the client's systems based upon documentation. Now we need to modify our code base to match the reality of the systems' implementation."

Lesson Learned: Collecting and analyzing field notes from participant observation is not easy. Just as Charmaz discovered that taking breaks from observation to write down her thoughts can be difficult [9], I found it very challenging to record observations during intensive activities (e.g. pair-programming).

Recommendation: Limit participant observation to 20-30 hours per week. Write detailed observations after work, but use post-it notes or an unobtrusive notebook for capturing insights during intense activities. This was more culturally acceptable than typing on a laptop. Explain your actions to participants. For example, saying "I want to reflect on what you said" or "what you said was really insightful" makes the participant feel valued rather than ignored. In short breaks, try recording verbal notes; some phones will convert spoken words into text.

Participant observation provided a rich data set for emergent ideas. For example, one day while pair programming, I realized that I did not know who was on my team because of significant team churn. This realization led me to convert task allocation data into a chart showing the start and stop dates for each team member, which raised the question, *how was it possible to be successful [15] with the client with so many people rotating through the project?* Investigating this research question led to the Theory of Sustainable Software Development [3].

Lesson Learned: Participant observation often provided research insights not available from interviews or documents.

Recommendation: Supplement interviews and document analysis with participant observation or direct observation (watching without interacting).

III. Constant Comparison

I used line-by-line coding [9] to identify nuanced interactions in the data and avoid jumping to conclusions. Whenever insight occurred, I would write it down as a memo. My advisor reviewed the initial codes while reading the transcripts and listening to the audio recordings. We discussed the codes and the coding process during weekly research collaboration meetings. To avoid missing insights from these discussions [13], we often recorded and transcribed the discussion into Grounded Theory memos. As data was collected and coded, I stored initial codes in a spreadsheet and I used constant comparison to generate focused codes.

Lesson Learned: I tend to continue too long with initial coding.

Recommendation: Use constant comparison to drive research forward.

Concept: The underlying, meaning, uniformity and/or pattern within a set of descriptive incidents.

Category: A type of concept. Usually used for a higher level of abstraction.

Property: A type of concept that is a conceptual characteristic of a category, thus at a lesser level of abstraction than a category. A property is a concept of a concept.

Coding: Conceptualizing data by constant comparison of incident with incident, and incident with concept to emerge more categories and their properties.

Fig. 3: Glaser's Definitions [16]

A. Understanding Constant Comparison

Glaser's seminal book assumes the reader is a sociologist, making it inaccessible to many software engineering researchers and other non-sociologists. The term *constant comparison* is especially challenging. Struggling to understand it—not just remember the definition, but deeply understand how it works—I read three of Glaser's books [6], [8], [13] and attended one of his seminars in Mill Valley, CA. The workshop helped me understand Grounded Theory terms that were not clear from the books. I later discovered clearer definitions [16] (see Figure 3), which Glaser wrote in response to Strauss and Corbin's [17] criticisms.

Lesson Learned: I needed to read four of Glaser's books and listen to him speak to properly understand constant comparison—Grounded Theory's main data analysis method. It involves continually comparing codes to codes, codes to categories, codes within a category, and comparing categories to categories not only to generate and refine emerging theory but also to decide what data to collect next.

I routinely compared new codes to existing codes to refine codes and eventually generate categories. I periodically audited each category for cohesion by comparing its codes. When this comparison became complex, I printed codes on index cards and then arranged and rearranged until cohesive categories emerged. I wrote memos to capture the analysis of codes, examinations of theoretical plausibility, and insights.

Recommendation: To deeply understand constant comparison, iterate between reading about grounded theory (or attending seminars) and analyzing your data. Accept that it takes study, practice, and reflection to master.

B. Pulling in New Literature and Data

As the *code ownership* category emerged, I looked to literature on ownership in general. Understanding the psychology of ownership [18] allowed me to connect team code ownership to underlying emotional needs and explained how emotions affect the transition from individual code ownership to team code ownership.

When *removing waste* emerged as a topic, I examined Lean Software Development to understand how the Poppendiecks used the Toyota Production System waste taxonomy for software development [19]. Contrasting the emergent waste taxonomy with the Lean Software Development's top-down taxonomy helped me better understand and clarify its structure and elements.

Recommendation: Examine literature outside the studied domain.

To understand software development waste [4], I began collecting and analyzing data from retros. (A retrospection meeting, or retro, is a weekly meeting to collectively reflect on the work done during the week, i.e., a safe place where any team member can discuss any issue.)

Lesson Learned: Fortunately, I had been recording retro issues for over a year, so there was no waiting to gather the needed data.

Recommendation: Record easily captured information, just in case. Take lots of photos.

C. Dealing with Confusion

During the analysis process, confusion can emerge. I found that sensemaking of comparing codes, naming concepts, defining terms, and identifying theoretical structure could be straightforward or anfractuous. Glaser said that "confusion is the royal road to emergence," [20].

My process for solving this tension looked like this:

1) Be confused
2) Try an idea (e.g. a name, a definition, a relationship between categories)
3) Iterate on Step 2
4) Experience an "ah ha" moment
5) Verify with data

Lesson Learned: Confusion appears to be a typical, critical step for the Grounded Theory researcher.

Recommendation: Be patient; embrace the confusion.

D. Focusing on a Manageable Topic

For Glaser, a key matter is "what is the participant's main concern?" However, a study of this duration may produce several distinct core categories, necessitating ongoing scope adjustments. When my initial findings on course correcting seemed unwieldy, my co-advisor suggested that I focus on one concrete category at first.

Lesson Learned: Ambitious Grounded Theory studies can illuminate several interesting phenomena. For a novice, initially concentrating on one area helps drive constant comparison and theoretical sampling. Later, you can return to the other phenomena.

Recommendation: When first learning Grounded Theory, focus on one emerging phenomenon of interest.

IV. Theory Construction

The iterative process of constant comparison and memo-writing that generates a grounded theory may be straightforward (as it was with our software development waste taxonomy) or confusing (as it was for the theory of sustainable software development). Understanding the relationship between properties may require significant analysis. For example,

I grappled with the nature of the relationship between code ownership and teams thriving despite excessive churn. Does ownership encompass both ideas? No. Does surviving churn cover both ideas? Maybe. Are both concepts manifestations of the same more abstract phenomenon? No. After reviewing Glaser's 18 theoretical coding families, I reconceptualized surviving team churn as "sustainable software development" with team code ownership as one of its properties.

We then started teasing apart the relationships between all the subcategories. Several felt different from each other. In time, we realized that some were practices, some were policies, and some were underlying principles.

Refining the practices into clear, logical groups was difficult. We identified six practices that supported sustainable software development by supporting team code ownership. My advisor proposed a separation based on her reading of the interview transcripts, but it did not resonate with the participant observation data. This spurred me into reconsidering the relationships between categories based on different specific questions. The question that eventually worked was "which of these practices directly affect the code?" Three of them did: TDD/BDD, Continuous Refactoring, and Live on Master. Then I asked, "how are these other practices related?" Continuous Pair Programming, Overlapping Pair Rotation, and Knowledge Pollination all concern knowledge silos. I recalled one interviewee discussing caretaking the code like a gardener and another expressing concern over knowledge silos forming on his team. I then labeled each group with *in vivo* codes (phrases used by participants), leading to the "Caretaking the Code Practices" and "Removing Knowledge Silos Practices" categories.

Lesson Learned: Theoretical structure may take time to emerge.

Recommendation: Review the 18 theoretical coding families in *Theoretical Sensitivity* [13]. However, do not adopt a coding family too readily—the coding family must earn its way into the theory like everything else.

Meanwhile, prior research treated code ownership as a team or organizational policy, whereas our participants understood code ownership as an individual's feelings toward the code, further complicating matters. My memos reflect the progression of searching for the best term: "code ownership" to "communal code ownership" to "collective code possession" and eventually to "team code ownership".

Lesson Learned: Choosing the right names for emerging ideas is important and challenging.

Recommendation: When struggling with names, try *in vivo* codes or terms used in existing literature. Try using variants in memos and conversations and see if any of them work. It is normal to feel conceptually uncomfortable during a Grounded Theory study. When in doubt, simply continue data collection and constant comparison.

V. Theoretical Saturation

Once a theory emerged, the next question was "what more data do I need to collect?" Starting this research study, I thought I understood theoretical saturation. Eventually, I realized that I did not and understood that the widely held definition of theoretical sampling as "the phase of qualitative data analysis in which the researcher has continued sampling and analyzing data until no new data appear and all concepts in the theory are well-developed" [21] is problematic.

Sampling until no new data emerges is subtly different than sampling to elaborate the emerging theory.

Grounded Theory is not about repeating the same questions until they stop producing new information. The Grounded Theorist alters the questions based on the emerging data and asks new questions to help elaborate and corroborate the relationships between codes, between codes and categories, and between categories in the emerging theory. The process stops once the researcher is satisfied that the theory's concepts and relationships are mature and the theory feels whole.

More data collection will often still reveal new concepts. The researcher may decide against including or further exploring these concepts for several reasons:

1) The concept is not directly relevant to the current theory (out of scope).
2) The concept has not earned its way into the theory. Potentially, a new Grounded Theory study at different site would produce more incidents revealing that concept.
3) The concept represents a poor depth-to-parsimony tradeoff; that is, adding the concept would increase the theory's complexity without a reasonable corresponding increase in explanatory power.

In other words, the idea of no new data appearing is rooted in interview-focused, sociological research where interviewees just stop saying anything new. In participant-observation, software engineering research, we often have much more data than we can manually analyze, so "no new data" never happens. Researchers must use personal judgment and experience to recognize a saturated theory and halt data collection.

At one point, I asked my advisor and co-advisor, "what questions should I ask my participants to achieve theoretical saturation?" This was unfair in retrospect. As the primary researcher, I am the one most qualified to know which relationships between the categories are not robust. I shifted from asking "what questions will get me theoretical saturation?" to asking myself, "what do I not know about this theory?", "what is confusing?", and "how do each of these categories relate?"

I finally understood Theoretical Saturation by reading and rereading Charmaz and Glaser, experiencing saturation firsthand and discussing these challenges with my advisor and co-advisor.

Lesson Learned: Theoretical sampling is collecting additional data to develop full and robust categories, identify the relationships between categories, and elaborate the main category's properties.

Recommendation: For saturation, focus on exploring and corroborating the emerging theory. Do not expect new data to dwindle.

VI. Managing the Data

During the study, I shifted from electronic to paper aids for constant comparison, as follows.

1) I initially did all constant comparison in Google spreadsheets.
2) I started hand-writing cards for hard-to-understand or complex comparisons and physically sorting them around me in a circle (e.g. needing to make sense of multiple perspectives about a backlog).
3) I printed the codes directly related to team code ownership onto index cards.
4) I printed all the retrospective notes onto index cards.

When looking at rows in a spreadsheet, I found them blurring together and easy for me to gloss over. When holding a card, I felt compelled to consider it before putting it down. In a spreadsheet, I subconsciously felt that adjacent rows were related even when they were not. With physical cards, I found that picking up the next card helped me see it as something new. However, maintaining *both* an electronic and physical copy was time-consuming and felt wasteful.

The advantages of digital aids include:

- Available wherever you have a computer
- Easy to share with co-authors
- Easy to backup and duplicate
- Simple to turn into physical cards

The advantages of physical aids include:

- Easy to rearrange
- Easy to see big picture
- Easy to focus on one area
- Easy to see outliers
- Easy to annotate with thoughts
- Encourages short capturing of ideas
- Time-consuming to turn into electronic storage

Lesson Learned: Sometimes using physical media can help with constant comparison.

Recommendation: There are many good ways to analyze data, and researchers need to find the techniques that work for them, but if you are getting lost in spreadsheets or NVivo, physical cards are worth trying.

VII. Publishing

I expected and experienced three main challenges with publishing this research. To the extent software engineering reviewers are familiar with Grounded Theory at all, they are accustomed to much shorter, interview-centric studies. Some of our reviewers did not understand analyses on data types (e.g. retrospective items) not usually used in Grounded Theory, or understand that such a long study produces multiple core categories, which cannot be addressed all in one paper. Meanwhile, some reviewers appear suspicious of qualitative research in general. To overcome their concerns, I distributed the methodology and findings very carefully across several interconnected papers.

Lesson Learned: It is extremely difficult to convey a large, participant-observation Grounded Theory study in a series of short papers, such that its methodological rigor is established and its findings are clear.

Recommendations: Use existing guidance for presenting grounded theory to a software engineering audience [1]. Rather than a long methods section, distribute ample methodological detail throughout the paper where appropriate. Explicitly address differences between interviews and participant observation. Clearly explain that you have conducted a *long* study with several core categories and that this paper only talks about this one category because space precludes treating all categories simultaneously. You can even cite our papers as precedent. Use a table or figure with examples to show the chain of evidence from raw data to categories. As reviewers, remember that the size of a competent grounded theory analysis precludes visualization even in a long journal article.

VIII. Advantages and Disadvantages of Extended Participant Observation

Extended participant observation allows for a deep understanding of participants' points of view. During this study, I have done almost 4,800 hours of pair programming. After thousands of conversations, I know what participants agree and disagree on.

Many software engineering researchers began in the software development field because they enjoy writing software. Writing code with a development team exposes the researcher to the latest software development techniques employed in industry. A researcher who has experienced pair programming for several weeks may have a richer experience to draw upon than a researcher who has no pair programming experiences.

Lesson Learned: Extended participant observation improves the researcher's technical skills, understanding of development processes, and empathy for software developers.

Recommendation: Recognizing that it does not work for everyone, consider participant observation (or direct observation). Avoid over-reliance on interviews.

During extended participant observation, a promotion may present a dilemma. At one point, I was encouraged to apply for a promotion to Associate Director, a sales role with no time for coding. Because working directly with the teams was crucial for my research, getting this position would have derailed the study.

Furthermore, the research cycle gets complicated. When is the right time to present research findings to the participants? Would presenting research findings alter the system under study? I resolved these tensions by not withholding my opinions while pair programming, making decisions that were best for clients, my company, and my team. For example, I suggested adding retros to a team that was missing them, and for a struggling team, I introduced stress relieving measures and team building exercises.

The academic cycle of writing papers, submitting, resolving reviewer comments, and presenting the research is lengthy. Since Grounded Theory produces theories that are grounded in the observed context, we realized that sharing my published insights would not adversely affect my research.

Lesson Learned: Sharing my published research with the organization confirmed resonance. Giving my academic talks at Pivotal offices became a way for me to give back.

Recommendation: After achieving theoretical saturation, share findings with the organization.

Moreover, business goals do not always align with the research goals. The research may be at the mercies of the business. Business concerns dictated my rotation from team to team. Fortunately, most of my rotations enhanced my research, but I could be moved from a really interesting project to one with little research potential. At one point, I was even transferred to a different business unit.

Lesson Learned: Business decisions can have profound impacts on research.

Recommendations: Either work with a company where all projects are relevant to the research area or negotiate a deal where the researcher has control of which projects they work on.

Glaser observed that Grounded Theory is easy to interrupt and pick up again. "Built into the method is the ability to put it down at will and pick it up later with virtually no need to backtrack or unduly review where the researcher was before the break in pace. The study is always ready to go forward on the current, next step. Thus there is no need to sacrifice the requirements of and need for family, friends and recreation for the research. The research in progress is always there waiting to move forward when the researcher can return to it" [8].

Lesson Learned: While Grounded Theory is easy to start and stop, balancing a 40-hour work week, research, and family is challenging. I kept Saturdays free, often using it for rest.

Recommendation: Build in rest time into your schedule.

As a developer, I have access to the stories in the backlog, daily conversations with my pair, team discussions, daily standups, weekly planning meetings, weekly retrospection meetings, and the code itself. There is a lot of potential data to analyze—much more than just interviews. The retrospection dataset alone had 663 items. The volume of data can be overwhelming. I resonated with the saying, "a potential problem with ethnographic studies is seeing data everywhere and nowhere, gathering everything and nothing" [9].

Recommendation: Use the ideas emerging from constant comparison to guide what kinds of data to collect.

IX. Conclusion

This paper attempts to use my experience running an ambitious grounded theory study to illustrate some challenges and recommend some practices that I found helpful. Some of the key challenges are as follows:

Grounded Theory is more complicated than it first appears. I had to not only experience the method firsthand, but also read and reread several seminal texts to really understand it. Questions asking participants to draw a picture helped me focus on interviewees' perspectives and avoid forcing topics. Participant observation revealed many insights that interviews could not. Prioritizing analysis over data collection help me avoid getting overwhelmed. My datasets were too unwieldy to analyze digitally; using physical index cards helped tremendously. Theoretical sampling is difficult to recognize and widely misunderstood. It is more about elaborating the emerging theory than running out of data. Extensive methodological detail and explicitly confronting the unusual aspects of the study helped assuage reviewer concerns.

Glaser says that a Grounded Theory study can create a rich trove of data. Indeed, I am still collecting and analyzing data and there is much more here worth studying. If you are conducting a Grounded Theory study of your own, and especially if you are struggling, feel free to contact me.

Acknowledgement

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

References

[1] 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*, ser. ICSE, 2016.

[2] T. Sedano, P. Ralph, and C. Péraire, "Practice and perception of team code ownership," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE, 2016.

[3] ——, "Sustainable software development through overlapping pair rotation," in *Proceedings of the International Symposium on Empirical Software Engineering and Measurement International Conference on Software Engineering*, ser. ESEM, 2016.

[4] ——, "Software development waste," in *Proceedings of the 2017 International Conference on Software Engineering*, ser. ICSE '17, 2017.

[5] W. M. Trochim. (2006) Research methods knowledge base, 2nd edition. [Online]. Available: http://www.socialresearchmethods.net/kb/

[6] B. G. Glaser and A. L. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Sociology Press, 1968.

[7] B. Brown, *Daring greatly: How the courage to be vulnerable transforms the way we live, love, parent, and lead*. Penguin, 2012.

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

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

[10] G. L. Evans, "A novice researchers first walk through the maze of grounded theory," *Grounded Theory Review*, vol. 12, no. 1, 2013.

[11] B. G. Glaser, "Constructivist grounded theory?" *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research*, vol. 3, no. 3, 2002.

[12] A. Bryant, "A constructive/ist response to Glaser's "constructivist grounded theory?"," *Historical Social Research / Historische Sozialforschung. Supplement*, no. 19, 2007.

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

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

[15] P. Ralph and P. Kelly, "The dimensions of software engineering success," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014.

[16] B. G. Glaser, *Basics of grounded theory analysis: emergence vs forcing*. Sociology Press, 1992.

[17] A. Strauss and J. Corbin, *Basics of qualitative research*. Newbury Park, CA: Sage, 1988.

[18] J. L. Pierce, T. Kostova, and K. T. Dirks, "Toward a theory of psychological ownership in organizations," *Academy of Management Review*, vol. 26, no. 2, 2001.

[19] M. Poppendieck and T. Poppendieck, *Implementing Lean Software Development: From Concept to Cash*. Addison-Wesley Professional, 2006.

[20] B. Glaser, "Notes from the grounded theory seminar held at mill valley," November 2015.

[21] J. M. Morse, "Theoretical saturation," *The SAGE Encyclopedia of Social Science Research Methods*, vol. 3, 2017.