Collaboration via Aligned Autonomy for Commercial Software Teams

by

Eirini Kalliamvakou
B.Sc., National Kapodestrian University of Athens, 2001
M.Sc., Athens University of Economics and Business, 2004

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

Collaboration via Aligned Autonomy for Commercial Software Teams

by

Eirini Kalliamvakou
B.Sc., National Kapodestrian University of Athens, 2001
M.Sc., Athens University of Economics and Business, 2004

Supervisory Committee

Dr. Daniel M. German, Supervisor
(Department of Computer Science, University or Victoria)

Dr. Margaret-Anne Storey, Departmental Member
(Department of Computer Science, University or Victoria)

Dr. Marian Petre, Outside Member
(Faculty of Mathematics, Computing and Technology Computing and Communications, The Open University)

Supervisory Committee

Dr. Daniel M. German, Supervisor
(Department of Computer Science, University or Victoria)

Dr. Margaret-Anne Storey, Departmental Member
(Department of Computer Science, University or Victoria)

Dr. Marian Petre, Outside Member
(Faculty of Mathematics, Computing and Technology Computing and Communications, The Open University)

ABSTRACT

Modern software organizations produce increasingly complex and sophisticated products that build on the effort of multiple individuals and teams. This reality highlights the critical importance of collaboration and the support of its various facets, which are still central concerns for software engineering research and practice. Software organizations also aim to motivate their developers and teams and help them be productive. Knowledge work research highlights the importance of autonomy in work design for satisfaction and happiness. The now pervasive adoption of agile methods and advocacy of self-organization have made autonomy and its challenging practical application a mainstream focus for software engineering research and practice.

Employee autonomy and effective collaboration are thus essential for software companies to motivate developers and help them deliver successful software products. Yet, essential as it might be for organizations to combine them, autonomy and collaboration seem conceptually and practically at odds with one another; is it possible for people or teams that are working together on something be autonomous? One can imagine teams finding it challenging to organize the development work of autonomous developers. Furthermore, on the organizational level it can be difficult to align autonomous agents towards a desirable company strategy. Finally, management

may need to be revisited as a function when individuals or teams have autonomy in their work.

Given the complex landscape that software teams are part of in today's modern organizations, we need to understand how they collaborate in the context of their environment. This dissertation builds on three substantial, diverse case studies based in industry, capturing various ways that several software organizations organize collaborative development work. In the first study I examined how 24 commercial software teams in different companies organize their development work through their use of GitHub. In the second study I probed how Atlassian scales the practices of its rapidly growing development teams and enacts a culture that keeps them aligned to the strategic goals. In the third study I explored the role of engineering managers at Microsoft and how they support software developers and teams to organize their own work and generate quality outcomes that meet organizational goals. The studies are primarily qualitative and I have used a variety of data collection methods including interviews, observations, documentation review, and surveys.

Tension between autonomy and collaboration surfaced in the studies and it became the central challenge I investigate in this dissertation. By understanding the meaning of autonomy for the studied organizations, the definition and characteristics of autonomy evolved and, upon synthesis of the findings, I argue that autonomy is not incompatible with collaboration but rather that the two concepts build on each other.

I articulate and propose a conceptual framework of *collaboration via aligned autonomy* for software companies in this dissertation. This represents a holistic view of organizations and includes four areas to consider when making autonomy the foundation of collaboration: team collaboration practices, scaling strategies, cultural values, and manager roles. The framework has implications for the study of collaborative software development by proposing to look beyond the combination of independence and coordination as the basis of collaboration. At the same time, the framework can guide commercial software teams and organizations on how to empower development teams, yet not compromise strategic vision.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

"My name is Sherlock Holmes. It is my business to know what other people do not."
Sherlock Holmes – The Adventure of the Blue Carbuncle

Getting through a doctoral program and writing a dissertation is a huge effort. There are many people to thank for helping me along the way. Hopefully I can — even in a small way — express how important their support and contribution was.

I'll start with my academic support system, which is another type of family. First off, Daniela Damian was my academic supervisor for a substantial part of my degree and the main force behind me coming to UVic and Canada in the first place. Dana supported me (both financially and academically) for a long time and played a crucial role in how I learned and grew as a researcher. She was always available to discuss and ready to challenge anything I put forward, and she opened a world of amazing opportunities for me. She also made me a member of her SEGAL lab, which gave me a tremendous sense of belonging. Although things didn't end up exactly how we planned, I owe her my gratitude for pushing me to become better and — for lack of a better word — *enduring* me.

Gratitude also goes to Daniel German for his great academic supervision. Daniel, without fail, acted as my sanity check the entire time I have been a PhD student. Simply put, if I could get an idea (spoken or written) past Daniel and his unforgiving logic, I could get it past pretty much anyone else too. Our discussions were always long (sorry Daniel!) and full of ideas, witty arguments, and food for thought. Daniel's support and continuous presence have been instrumental in how I developed my ideas, how I resonated about research and practice, and how I see my next professional steps. He was also — ironically, given his own research focus — the first one to encourage me to build my skills in qualitative research and make it my central focus. Hopefully I have helped him build more empathy for qualitative research in return, but I am deeply thankful to him for supporting me to engage in it regardless.

My two other committee members, Peggy Storey and Marian Petre, have been the best committee members any PhD student could ever hope for, and more. At times both of them have acted more as supervisors than committee members, offering thorough feedback and excellent advice. Peggy also opened her lab to me and included me in all the activities, presentations, and interactions that make the CHISEL lab the awesome place it is. Marian was geographically away but always quite close and our cups of coffee — physical or digital — were great support for me.

During my time at SEGAL I was lucky to engage with a lot of awesome students. Adrian Schroeter showed me the ropes around the lab and UVic and shared all his wisdom and experience with me when I first arrived. Eric and Alessia Knauss (and their awesome daughter Marie!) showed me around Victoria and beyond, and included me in all their fun activities to make sure I don't overwork. Especially with Alessia we shared the same office for 2 years, and had almost daily conversations about work and life. I had some amazing times with Jordan Ell and Braden Simpson, both in and out of the lab, and I still laugh when I think about some of our jokes and banter. I also met Germán Póo-Caamaño and his wife Tatyana because of their ties with the SEGAL lab and had great conversations (academic and not) and lunches with them at UVic. Thank you for all the good times Francis Harrison, Prashant Chhabra, Jyoti Sheoran, Aminah Yussuf, Guy Evans, and Lloyd Montgomery. I also met and worked with Kelly Blincoe, and I thank her for the great discussions and advice she had for me, and for being my roommate in Florence.

I thank all the CHISEL members, new and old, that I've had the pleasure to socialize with, for being so friendly and inclusive. Cassie Petrachenko is one of the most efficient and helpful people I have ever met and I can absolutely see why she is the gatekeeper. Special thanks goes to Alexey Zagalsky for letting me rant about pretty much anything, and for his insightful and helpful comments in response.

A big thank you also goes to my department chair, Ulrike Stege, who was extremely supportive and helped me maneuver sensitive situations. All the ladies in the 504 office — Wendy, Nancy, Erin, Jen, and Shanel — thank you for being so helpful and for making everything such a smooth sail the whole 5 years.

I feel I owe a lot to two people from my academic environment in the Athens University of Economics and Business, where I was before I came to UVic. Dr. Diomidis Spinellis was a huge academic influence on me and I thank him for all his support and advice along the way. I also had the pleasure to — in equal parts — work and hang out with Dr. Georgios Gousios; thanks for all the help and for taking the time to catch up at every conference!

During my PhD studies I had the fortune and privilege to go to Microsoft Research for a summer internship. Chris Bird — who I looked up to before we ever met — was a fantastic mentor, and gave me an amazing experience. I am grateful to him for giving me the autonomy to explore the many facets of the project we worked on and for all his support before, during, and after my internship. Tom Zimmermann was always available to answer questions and offer advice in his quiet but powerful

demeanor, and together with Chris made me feel like a valued peer the entire time. I am also thankful to Andy Begel and Rob DeLine for all their constructive feedback and lively discussions with me.

While academic support was crucial, I could not have pulled through without the support of my friends and family. Although I left Greece and moved 10 hours time-difference away, my friends were always available to chat and virtually hang out, even if we needed to schedule it. Thank you Stathis Lytras, Panos Tsaknias, Michail Dimakos, Eva Tsouana, Spyros Samothrakis for all our chats, laughs, and giggles. Thanos Tsouanas has been a significant presence in my life in various ways and for multiple reasons. I'm also proud to have my friends Nikolas Samaridis, Sotiris Skoumatzos, and Apostolos Kefalas by my side since kindergarten.

A slightly odd thank you goes to Arthur Conan Doyle. Anybody that knows me knows I am a huge fan of Sherlock Holmes, and I have read all the stories about him multiple times. Often during my PhD I opted for reading Doyle's stories yet again because they are so familiar and comforting, and never fail to put me in awe of the brilliance of Sherlock Holmes. I have used quotes at the beginning of all chapters in this dissertation as a small tribute to my hero of so many years.

There are no words good enough to describe the significance of my family in my life, and of course during my PhD years. My father has always been a role model for me and one of the people that I admire the most for his wit and character. My mother is the sweetest and most caring person I have ever met, full of unwavering support. The two of them were on Skype every single Saturday for our weekly catch-up; sometimes seeing me frustrated, usually seeing me happy and full of news, but always being there to *listen*. They have supported me more than they know, and I hope I've made them proud. My wonderful brother and his fantastic wife always had a virtual seat for me at their breakfast table, and were ready to chat.

Finally, I met my boyfriend, Graeme, in the last two years of my degree but his impact has been life-changing. It all started with him coaching me to give a presentation at ICSE 2015 that I still get compliments about. He introduced me to his amazing family who made me feel like a dear member right away, and filled some of the gap from my own family being far away; thank you Jan, Terry, Robert, Dolma, and little Dorje! Graeme was my conscience, confidant, friend, cheerleader, advocate, and so much more at a time that proved to be difficult and critical for me. He also was patient about my long hours and frequent rants. Thank you for all your support and for being the amazing person you are. This dissertation is dedicated to you.

DEDICATION

To Graeme, for changing *everything*.

Chapter 1

Introduction

"Come, Watson, come!" he cried. "The game is afoot. Not a word! Into your clothes and come!"

Sherlock Holmes – The Adventure of The Abbey Grange

Two important elements for creating success in software companies today are ***collaboration*** and ***autonomy***. Modern software companies are geared toward supporting teams to *collaborate*, to build complex and sophisticated products [64]. At the same time, providing knowledge workers with *autonomy* in their work makes them happier and more satisfied [124]. Effectively combining individuals' autonomy and teams' collaboration can help improve software quality, increase teams' productivity, and boost employee motivation and organizational engagement [20].

How can modern software companies realistically tap into the potential unleashed by combining ***collaboration*** with ***autonomy***? Such concerns are attracting attention in software engineering research [152], and are discussed by practitioners [1].

Some software development environments have successfully balanced autonomy with collaboration. Open source software development for example is known for building on the contributions of loosely-coupled volunteers to deliver high-quality software. In recent years organizations have been looking to replicate the success of open source software development by following similar development principles, a trend known as Inner Source [191]. Despite the progress, organizations are still looking to provide autonomy to their development teams in meaningful ways while they build software collaboratively.

Another example of infusing autonomy in the collaborative software development

[1] https://hbr.org/2017/02/how-spotify-balances-employee-autonomy-and-accountability

process is the case of agile software development methodologies, which are increasingly being adopted in software organizations [207]. The Agile Manifesto [13] directly advocates self-organizing teams while it highlights that not much guidance is needed for individuals besides providing them with the environment and support they need. However, agile software development methods have often been scrutinized for having limited applicability to large-scale projects. Recent software engineering research is proposing and exploring ways in which software companies can achieve organizational agility [75], and more studies are needed on how the proposed solutions may apply in practice.

Looking at collaborative development in modern software companies, there are some notable trends that reveal conflicting issues at play:

First, some modern development tools (such as GitHub or BitBucket) offer means to decentralize development work but it is unclear how they fulfill the requirements of building software collaboratively. Is it possible to work independently and collaboratively? ***What practices do commercial software teams use to balance collaboration with autonomy?***

Second, when a software organization grows it may be unclear if and how to adjust its development process to a larger scale. One can imagine that scaling autonomy beyond a few people or teams may result in misdirected work in the organization and jeopardize its strategic goals. ***Are there risks involved in scaling autonomy? What work practices do software teams and organizations use to address the risks?***

Finally, organizations need great managers to make teams successful but increasingly today we hear about a push toward self-management and self-directed individuals and teams. ***What are the attributes that are perceived to make great engineering managers in an environment that values autonomy?***

In this thesis I address these questions and propose a conceptual framework that enables teams to achieve collaboration via aligned autonomy, highlighting those areas that require consideration and providing actionable strategies for software teams and organizations to apply. While this is the outcome of synthesizing the findings of multiple case studies, the studies themselves did not start off as investigations of autonomy. Rather, the necessity and challenge of balancing autonomy and collaboration surfaced in the studies and with that my understanding and definition of the concept of autonomy evolved. As a result, the questions I address with the thesis emerge from the particular research questions of each study. By presenting the stud-

ies, questions, and findings I hope to guide the reader in following the interpretations and understanding the proposed framework and its parts.

The participating software teams and organizations in the studies use practices that are in line with agile methods. The discussion of collaboration and autonomy in this dissertation is thus scoped on agile teams and companies that seek organizational agility. Such a focus is timely and appropriate. First, agile methods are becoming pervasive in industry, as shown by a recent large-scale industry survey reporting that 94% of respondent organizations follow an agile approach [207]. Second, agile methods are often tailored in practice [73] and there are even frameworks proposed for their appropriation [30]. This creates a need for a deeper understanding of the interplay between the agile practices and other organizational factors. Finally, there are questions about if and how agile practices apply to large projects [56] and scale to entire organizations [75], and additional studies can provide further insight.

The components that make up this thesis and what they bring into the investigation are listed below:

Chapter 2: Background

This chapter includes a review of empirical studies of collaborative software engineering in commercial and open source software development settings, agile software development, as well as work in organizational psychology and behaviour that explain the benefits and need for software developers to be autonomous.

Chapter 3: Methodology

In this chapter I present the epistemological stance I have taken in developing this thesis, and my approach for drawing insights from the case studies I have conducted. I also provide an overview of methodological choices for the case studies, while the details for each study are presented in the corresponding chapter.

Chapter 4: Autonomy as independence — The GitHub study

The study presented in this chapter focuses on 24 commercial companies using GitHub as their collaborative development platform. Alongside the use of GitHub's features I collected the practices the teams used to collaborate. The case's findings show that teams, either collocated or distributed, want to be able to function in the same decentralized way. That includes independence during development, lightweight communication, transparency/visibility of status, and agile development; all these are supported by coordination points clearly marked by pull requests.

The chapter discusses how software teams can collaborate while having autonomy, and results in a series of **team collaboration practices** for the framework of

collaboration via aligned autonomy.

Chapter 5: Aligned Autonomy — The Atlassian study

The study presented in this chapter focuses on understanding the challenges that software companies face when following agile practices at a large scale, and the work practices used to overcome them. I report a study of teams at Atlassian, and how they balance agility and strategy under conditions of rapid growth. The findings show that alongside the practices that enable autonomy, teams follow practices that support alignment with organizational strategy, ensuring that the autonomous effort is channeled in the right direction. The combination of the two — autonomy and alignment — is an imprinted culture in the organization, and acts as its unwritten but agreed on process.

The chapter discusses the risks involved in scaling autonomy and the work practices that can help avoid or address the risks. The study results in **scaling strategies** and **cultural values** for the framework of *collaboration via aligned autonomy.*

Chapter 6: Autonomy as empowerment — The Microsoft study

The study presented in this chapter focuses on identifying and understanding the attributes of great engineering managers. I report a study of engineering managers at Microsoft, and their perceived role and attributes in a modern software company that seeks to empower its developers. The case study shows that enabling autonomy and ensuring alignment are considered part of great management and that managers are seen as coaching and motivating engineers, while also acting as a conduit between them and the organization. The chapter discusses the attributes that are perceived to make great engineering managers in an environment that values autonomy. The study results in a set of **manager roles** for the framework of *collaboration via aligned autonomy.*

Chapter 7: A framework of Collaboration via Aligned Autonomy

This chapter synthesizes the case studies' findings, discusses how they relate, and summarizes them in *a conceptual framework of collaboration via aligned autonomy.* I discuss the framework in light of existing work in several research domains, present implications for research and practice, and discuss the validation and limitations of the research.

Chapter 8: Conclusions and future direction

In this final chapter I summarize the main points of the thesis and the potential avenues it opens for future work.

The proposed conceptual framework aims to contribute to the knowledge about

collaborative software engineering and generates testable hypotheses about its effects, which could serve as a basis for further research. The work I present in the chapters offers insights that can help software companies and projects to empower developers and improve collaboration to generate better organizational outcomes. Software companies can also use the thesis results to formulate their own interventions.

Chapter 2

Background

"It is not so impossible, however, that a man should possess all knowledge which is likely to be useful to him in his work, and this, I have endeavoured in my case to do."
Sherlock Holmes — The Five Orange Pips

In this chapter I review empirical studies and literature that relates to the concepts of autonomy and collaboration, to create ground for understanding how the concepts are discussed in the dissertation. Literature is also introduced in the rest of the thesis chapters as needed, in line with interpretivist research (I elaborate more on this in Chapter 3).

Autonomy has been advocated in multiple bodies of the literature. For software engineering, it is most associated with agile software development methodologies, which are increasingly becoming a norm in software companies and beyond. I present theoretical and empirical work that explains why organizations (should) strive to have autonomous individuals and teams, ways that autonomy has been enacted, and the concerns about its implementation and scalability. The majority of the literature relates to software engineering, but I draw from a diverse set of research fields — after all, autonomy is not a new concept and comes in many guises. Throughout the chapter I link the literature to the questions that set the stage for the empirical work presented and discussed in the rest of the dissertation.

2.1 Toward a definition of Autonomy

Dictionaries define autonomy by means of self-governance and independence; Merriam-Webster expresses it as "the quality or state of being self-governing" but also as

"self-directing freedom and especially moral independence" [1]. The Oxford Dictionary defines autonomy as "the ability to make your own decisions without being controlled by anyone else" [2].

For the scope of the thesis, I am discussing employee autonomy, either individuals or teams. It is safe to assume that employees don't go about their day-to-day work having a dictionary definition of autonomy in mind. Instead they attach meaning to it (or may inherit it from the organization they work in) and enact it. We may hypothesize that autonomy is a familiar concept to all, and that the word evokes similar understanding for everyone. However, as Janz et al. [107] and Hoda et al. [102] point out, in the literature autonomy is interchangeably referred to as empowerment, self-direction, or self-management, depending on the discipline. This signals to me that the concept is not universally agreed on but a malleable one, and it should be interpreted in context.

In my search of the literature I suspected that concepts such as empowered teams, autonomous teams etc. may have different names and appear as different terminology, especially between disciplines. To test and address the issue I used multiple search terms on Google Scholar (e.g. "autonomous teams", "self-organized teams" etc.) to get an indication about possible different terms. At the same time I consulted with researchers and academics with knowledge of domains outside strictly software engineering. Starting with the journal articles by Janz et al. [107] and Hoda et al [102], which mentioned different terms that autonomous teams appear under in different literatures, I loosely followed a backward and forward snowballing technique, to gather more papers and build an understanding of the terminologies and the characteristics that different disciplines assign to the particular type of work team. I opted out of the systematic version of snowballing and literature review because the goal was understanding the concept, rather than mapping the knowledge.

Management literature sees autonomy as "the extent to which an individual or group has the freedom, independence, and discretion to determine what actions are required and how best to execute them" ([107], p. 47). In the context of work, this definition means that an autonomous entity is one with decision making-capacity, without the need to seek approval from others. Janz et. al [107], go a step further and include responsibility in autonomy. That is, autonomous entities do not only have the capacity to make their own decisions, but also the responsibility to carry them out.

[1] http://www.merriam-webster.com/dictionary/autonomy
[2] http://dictionary.cambridge.org/dictionary/english/autonomy

Organizational theory agrees with this, and sees self-organizing teams as responsible for scheduling, hiring and task assignment [160]. The inclusion of responsibility when talking about autonomy is an important one because the afforded freedom is usually what skeptics criticize about autonomy.

Even reviewed literature does not necessarily provide a single-sentence definition of autonomy. Instead a contextual characterization of autonomy emerges from the way it is captured and/or discussed in studies; I present a number of those in the rest of the chapter. In the dissertation I have taken the same approach; I have built my understanding of autonomy, its meaning and characteristics, and its implications for organizing collaborative software development work from each study's findings and context. Therefore, the definition and understanding of the concept of autonomy evolves throughout the thesis. I discuss the evolution in each study (Chapters 4, 5, and 6), and as part of the synthesis (Chapter 7).

2.2 Why Autonomy is desirable

The literature offers a variety of reasons why autonomy is desirable and why it works — in theory at least — in the favour of organizations and institutions that enact it. The concept of autonomy transcends professions so I am not limiting the work I discuss in this chapter to just software engineering organizations; the discourse and findings are similar across industries and in many cases were discussed in other domains first before software development. While some cases are discussed here as well, I am focusing more on the software engineering research area from section 2.3 onwards. Below I provide a high-level view of the discourse relating to employee autonomy, before going into details about the effects of autonomy in sections 2.2.2 and 2.2.3.

2.2.1 A bit of history

To discuss the advocacy in favour of autonomy, it is necessary to consider the evolution in management thinking. Originally, organizations followed top-heavy hierarchical and management structures to organize and carry out work. Pearce and Manz [166] reviewed the historical foundations of leadership and noted that in the 20th century management and leadership — termed "scientific management" — were focused on how to make workers as productive as possible by optimizing their tasks. There

was no autonomy or employee decision making to speak of in this setting; workers were there to provide manual labor, and their tasks were predesigned down to the last movement. Workers were thus considered interchangeable, since what they were contributing were essentially hand and feet movements that "scientific" managers had decided for them. Management and worker responsibilities were separated; managers identified precise work protocols, and workers followed the dictates [166], which set the tradition of command and control in organizational life.

Between the 1920s-1950s the field of management turned its attention to how workers think and feel about their work (see [42] for a detailed overview) aiming to motivate employees to identify with organizational goals [116], and improve performance [180]. Two milestones have been key to shifting the focus of management. The first milestone was the introduction of psychology and sociology theories in management, researching factors that affect employee needs [139] and engagement [95]. Second, the rise and increased mobility of knowledge workers turned attention toward the behavioural aspects of employees [113]. Peter Drucker [61] in 1959 led management thinkers to see the corporation as a social institution and workers as assets, challenging existing management principles as fit only for manual labour. Especially after the 1960s — that were considered the modern era of management — the focus is on employee work attitudes and motivation [141] and the recognition of people management as separate from work organization or project management [165].

These developments and the recognition of knowledge work as different enough from manual labor to warrant different organizational treatment, brought the empowerment of employees — by giving them autonomy in designing their work — in focus for both researchers and practitioners. As Crainer [42] and Pearce and Manz [166] note in their review of management history, global market pressures led organizations to look for better ways to compete; through a flexible workforce, reduction in organizational response time, and full use of organizational knowledge.

Software organizations are, of course, part of this landscape; strict, upfront planning doesn't work in rapidly changing conditions [66] and instead interventions are needed from all. Rigid hierarchies and top-down management structures can stifle creativity and innovation, however. Tore Dyba [66] likened organizations responding to market changes to jazz bands improvising, arguing that strict adherence to enforced processes can limit developers when creating their own understanding of the situation to which they have to respond. Takeuchi and Nonaka [197] advised that when it comes to new product development the higher levels of hierarchy's involvement should

be limited to providing guidance, money, and moral support. In self-organizing agile teams, informal and transient roles emerge [102] to provide guidance. Such conclusions, coupled with the notion that today's employees desire from work more than a paycheck [61], create questions about hierarchical structures, and set the stage to not only talk about but also try to find ways to enact self-management, shared leadership and autonomy.

Overall, increased employee autonomy has been linked to improvement in organizational outcomes. In some cases this is seen as a direct link to, for example, the productivity of individuals or teams. Mostly, however, the effect is second-order; employees that are autonomous are more motivated, more satisfied with their work, and report a higher quality of work life [107]. It is these psychological and behavioural factors that, in turn, have a positive effect on productivity, performance, and success [15]. I discuss these effects below.

2.2.2 Job satisfaction, motivation, and engagement

Autonomy is described as a strong motivator for employees in a variety of research domains, but has applicability in software engineering as well. In a systematic literature review, Beecham et al. [15] discussed motivation in software engineering and reported that motivation has the single largest impact on practitioner productivity and software quality management. Alas, the authors concluded that motivation continues to be problematic to manage, in part due to the resistance or slowness of software companies to make changes that contribute to a sense of positive motivation. The literature review from Beecham et al. lets two things come to the surface. First, autonomy and empowerment are definite motivators for software engineers; this is a strong signal about why software organizations should make a real effort to build on them. Second, autonomy is linked to organizational outcomes; autonomy affects how long it takes developers to deliver software, their retention by the organization, their productivity, their absenteeism, and project success.

Janz et al. [107] link autonomous teams to business process improvement by showing that autonomous teams have a positive impact on the quality of work life and ultimately the performance of workers. Tata [199] mentions that companies use autonomous teams with reported benefits of increased performance, higher quality products, less absenteeism and reduced turnover. Studies in the domain of psychology and organizational behaviour see autonomy as a job characteristic, as the extent to which

a job provides the employee with the discretion to choose how their work is done. According to job characteristics theory [93] more autonomy means more responsibility for the individuals or the teams, which puts them in a "critical psychological state" (p. 160) that translates to increased motivation and satisfaction, and results in better individual and team performance. Van Mierlo et al. [204] reported that individuals in teams with high levels of autonomy (i.e. the team works as a self-governing unit, separate from other groups) are less likely to report emotional exhaustion and are more keen to learn on the job. Pritchard and Karasich [1] conducted a study in multiple organizations, statistically analyzing factors in how they promote autonomy, and brought empirical evidence that organizational climate has a significant effect on organizational performance and employee satisfaction.

These findings are consistent with previous work discussing the merits of autonomy and bottom-up decision making in organizations to fight the lack of employee engagement — a plague for modern organizations as reported in studies by Gallup [3] researchers [95, 98]. Janz et al. [107], looking at autonomy as a business process improvement, brought evidence from multiple organizations that flattening organizational hierarchies, empowering bottom-up decision making, reducing checks and controls, and transforming managers to coaches led to increased job satisfaction and internal motivation — what the authors refer to as quality of work life. They also pointed out that these positive effects of autonomy have been advocated before them but that organizations are not necessarily adopting the guidelines.

Notice here that most work that discusses autonomy and empowerment reflects employee perceptions, i.e. how employees perceive the degree of autonomy they are afforded. This is important for a variety of reasons. First, work in organizational psychology has established that employee perceptions about work conditions affect organizational outcomes, such as customer loyalty and the financial performance of the organization [97]. Second, it is not uncommon for organizations to claim autonomy and empowerment but practice *symbolic* self-management [148]; assessing developers based on what they produce and having them compete for resources. Third, there are signs in management [68] and entrepreneurial literature [28] (and software engineering is catching on [16]) that the attitudes of employees and whether they identify with organizational values (or culture) has strong impact on their motivation and output.

In software engineering research, autonomy and self-management have been dis-

[3] http://www.gallup.com/home.aspx

cussed in the same breath with agile software development methodologies. I will talk more about agile software development methods in section 2.4.2, although they probably don't need much introduction nowadays; it suffices to say that they represent an approach to software development that focuses on flexibility through short development iterations that include all stages of the work from requirements engineering to testing and deploying. The agile methods came about with the promise of dynamic adaptation [155] for companies facing rapid change [185]. The goal behind the methods is — as the name reveals — to achieve *agility*; under conditions of agility teams have the capacity to efficiently and effectively respond to changes in user requirements or market conditions. Keeping up with change is a real problem for software (and other) organizations, with very low percentages being able to keep up and many companies experiencing financial loss [125]. To achieve agility, individuals and teams need autonomy [125] — discretion and independence granted to them, leading to decentralized decision making. In the agile rhetoric, autonomy increases the speed and effectiveness of problem solving because it brings decision-making authority to those who face and handle problems every day [123].

Autonomy is also discussed as part of a new way of developing products. Takeuchi and Nonaka [197], the leading figures in lean manufacturing principles — where, incidentally, agile methods have originated — identified six characteristics of the new product development approach. Among them, self-organizing project teams were described as critical to achieve speed and flexibility in product development. One of the critical points made in the work of Takeuchi and Nonaka is the importance of transparency and the free flow of information. This is in complete agreement with the literature on self-management (see [92] for an excellent review) where the major risk of autonomy is recognized as isolation of entities, and the antidote is the access to external information, and transparency. The empirical study I present in Chapter 5 focuses on the risk of isolation posed by scaling autonomy to a growing organization, and discusses the role of open communication and culture in mitigating the risk.

2.2.3 Knowledge worker productivity

One of the main realizations that is shaping how organizations operate and how employees have been managed in the past 60 years, is that the nature of work has changed, shifting from manual to intellectual labor. Both research and practice have been investigating the characteristics of knowledge work and workers to understand

their role and motivation, and drive their productivity. The differentiating factor of knowledge work is that the basic task of knowledge work is *thinking*; knowledge workers process non-routine problems that require non-linear and creative thinking [174]. More recently, the globalization of markets and corporations brought on an intense need for innovation in organizations and knowledge workers are regarded as important elements for organizations' survival and prosperity [146]. Globalization applied to the workforce too though, making knowledge workers mobile, and able to find work that fit their distinct sense of purpose and motivation [61]. The retention of knowledge workers is critical for organizations nowadays, making it important to understand their motivation and what can help them be productive.

Let's start with what *doesn't* motivate knowledge workers. Earlier the literature on what alienates knowledge workers finds that centralization and formalization have a negative influence on how engaged knowledge workers feel [159]. While Quntanilla and Wilpert reported that increasing autonomy for knowledge workers led them to find more meaning in their work [171], more recently Donnely questioned that claims of increased flexibility and autonomy for knowledge workers are true [60]. The results are still in line with the argument in favour of autonomy though; Donnelly found that knowledge workers (in this study, consultants) were not able to exercise greater control of their working arrangements because the narrow views of their employer on what constitutes professional behaviour restricted them. Thomas Davenport [54] and Frank Horwitz [103], both prominent figures in the literature about knowledge worker management, have repeatedly brought evidence that knowledge workers resist command-and-control styles of working and, instead, seek autonomy in their work. That means that hierarchical, top-down management structures need to be reconsidered in today's companies that mostly rely on knowledge workers.

There is agreement in the literature that the way to make knowledge workers productive relies on finding ways to motivate them. Because knowledge workers are in demand and — due to the global workplace — highly mobile, they are in a position to pick jobs they feel strongly about. Mládková points out that for work to be motivating for knowledge workers it needs to provide them with a sense of purpose and pride [146]; if these needs are not met, employees are not committed to the organizations they work in. Ehin [68] explains that knowledge workers need different management because generating knowledge is essentially a voluntary process. These findings agree with what Peter Drucker — the first to talk about knowledge workers — argued in 1959 is a shift from a universe of mechanical cause to one of purpose

and process, where people of knowledge and high skill can organize themselves for joint effort and performance. Unfortunately, Blacker [21] concludes that although the determinants for knowledge worker productivity are known, a way to integrate them still eludes organizations.

Software engineers belong to the category of knowledge workers. The ongoing research into developer productivity and motivation [144], and the concerns of practitioners on how to manage developers [4] indicate that Blacker's conclusions are still valid today and organizations are looking for ways to motivate knowledge workers to have them be productive.

2.3 Collaboration challenges in software engineering

The Cambridge Dictionary defines collaboration as "the situation of two or more people working together to create or achieve the same thing" [5]. For software teams in particular, the creation or achievement in the definition would be about producing software. While the need and potential benefits of introducing autonomy in an effective way are relevant to software engineering as in other knowledge work domains, collaborative software engineering has its own difficulties that making the combination of autonomy and collaboration and interesting challenge — one that is not yet resolved for research or practice. In this section I review well-known challenges of collaboration in software engineering teams, before moving to discuss autonomy in software engineering, in section 2.4 below.

In his PhD thesis in 2010, Jorge Aranda [6] focused on communication and coordination as the most critical challenges of software engineering. He argued in favour of a shared understanding of the foundation of coordination — plans, goals, and context — rather formalizing its process. His review of earlier work showed a disconnect between the principles of process engineering — planning, stability and repeatability — and those of agile software development which focus on flexibility and self-organization. This is important for two reasons. First, the widespread use of agile methodologies in software organizations today signals that the industry itself is showing disagreement or incompatibility with the ideals of process engineering. Second, a number of reports

[4] `http://developermedia.com/developer-personalities-audience-brief-report/`
[5] `http://dictionary.cambridge.org/dictionary/english/collaboration`

from industry published after Aranda's thesis ([161, 119, 205, 106] to name a few), show that even agile practices are not prescriptions, demonstrating the industry's desire for flexibility and ability to make the best choices *for them* rather than following process instructions.

Collaborative software development projects bring together the interdependent efforts of team members, coordinated toward a common goal [211]. Malone and Crowston [135] view collaboration as building on efficient coordination of direct or indirect actions needed to manage interdependencies. *Communication* and *coordination* are considered among the critical challenges in collaborative software engineering. Cataldo and Herbsleb [33] showed — using the concept of socio-technical congruence developed earlier by Cataldo et al. [34] — that coordination breakdowns are harmful to product development outcomes, but that they are not the only challenges. Maintaining *awareness* of interdependencies is also challenging; tools — such as Tesseract [183] or Palantir [181] — can provide alerts of potential coordination needs and recommend communication between interdependent developers [194], but it is not without overhead [32]. Especially in the last few years, developers use an even wider variety of communication channels to maintain awareness and manage knowledge. Storey et al. [192] recently reported the multitude of challenges that come bundled with the benefits of socially-aware communication channels; developers often are overwhelmed and find that information is fragmented due to the simultaneous use of multiple communication channels. Such an account shows that building and offering more tools is not the solution to awareness, and that integration of tools is an important aspect to manage information overload.

Conflicts occur even in the presence of awareness tools and teams spend effort to resolve them [182]. Efficient *task division* and scheduling can help to minimize, but not eliminate, coordination overhead and code conflicts [111]. Modularization of tasks is an accepted way to minimize interdependencies [215], but it is impossible to remove them altogether.

Such collaboration challenges are typical in software development, and they are accentuated when teams are distributed. Collaborative Development Environments (CDEs) integrate source code management tools and bug trackers with collaborative development features [122] and have been suggested as a solution to the communication and coordination challenges of distributed software projects in organizations [4]. Yet, these challenges persist and projects old [99] and new [177] report them. For the purposes of this thesis, I operationalize a team's collaboration through the practices

they use to handle **coordination** through **communication**, **awareness**, **task division**, and **conflict resolution**, facilitated by tools. I refer to these aspects as *the collaboration elements* — they are prominently featured in Chapter 4.

2.4 Autonomy in software engineering teams

As we saw, autonomy has been discussed in many domains and in various forms that are not software-engineering-specific. Yet, there is growing interest in it in software engineering research too nowadays. Below I discuss two prominent appearances of the concept of autonomy in the software engineering literature: open source software (OSS) development, and agile development.

2.4.1 Autonomy and open source software development

OSS development has been an intriguing research domain for well over a decade now. Studies of developer motivation and project organization were the first attempts of the research community to understand and explain the presence and evolution of OSS development, and to try to predict its future as a software development trend analyzing its viability as an economic and business model (for example [127]). Interest in the collaboration model that supports OSS development has also been considerable, given that OSS developers are highly distributed and mostly don't engage in face-to-face communication. Due to the pragmatic constraints they face, it has been considered surprising that OSS projects achieve smooth coordination and consistency in design and innovation.

Given the special characteristics of OSS development and the interest in how OSS developers coordinate their work, research has shown interest in the tools and practices that facilitate OSS development. Version control was initially a centralized mechanism for coordinating and synchronizing development [55]. Today, with the advent of decentralized version control systems like Git and code management platforms like GitHub, the degree of independence of the participants is higher. Studies have found that decentralized version control systems can change development processes [10] and influence developer behavior [168]. Especially GitHub — as the platform of choice for many large and well-known OSS projects — has received research attention regarding the models of development it has popularized [84], and the OSS ecosystem it's creating [206].

Ducheneaut [63] found that OSS developers coordinate their work almost exclusively through three information spaces: implementation, documentation and discussion. The versioning system provides the backend of the implementation space, keeping track of changes made to the project related files and corresponding versions. The internet is used as the documentation space with wikis, mailing list archives, most recently also peer-developed resources like Stack Overflow. Because of the distributed and informal nature of OSS projects, discussions between project members, associates and users are done and tracked in mailing lists and online forums, more recently shown to also take place in the issue repository [91].

The voluntary nature of OSS development means that developers need to be loosely-coupled; self-organization is a main characteristic of OSS development. As pointed out earlier, self-organization does not mean that developers are irresponsible or make arbitrary contributions. In the end, contributions are peer-reviewed and judged on merit; Yamauchi et al. [218] found that OSS programmers are biased toward action rather than coordination and follow a rational decision-making process where the technologically-superior options are always chosen.

Walt Scacchi [184] thoroughly discussed collaboration affordances that facilitate work in OSS projects based on insights from several empirical studies on the work practices followed by OSS teams. First, OSS projects nurture the motivations of developers, and give them the responsibility to organize and manage themselves to fulfill their motivations. OSS projects provide an environment of shared beliefs in freedom regarding software, speech and choice [69], trust and social accountability. Beliefs, values and norms form part of the developer's mental model [71] and serve as affordances that enable collaboration and teamwork, while the failure to enact these beliefs can drive developers apart [70]. Finally, artifacts that developers use to prescribe or question what is happening in an OSS project (called software informalisms) are also considered collaboration affordances, creating a healthy environment for discourse in the project.

Autonomy has been a premise and necessity for OSS projects due to their nature, and it is one of the motivating factors for contributors [120]. Software organizations too, however, have realized that software developers — as knowledge workers — are motivated by autonomy and the chance to self-manage. The birth of agile development methodologies in 2001 articulated some guidelines for software companies about why and how to organize development in a way that it empowers software developers to make decisions, as I discuss below.

2.4.2 Self-management in agile software teams

The concept of empowering development teams by providing them freedom to make decisions about their work content and schedules is discussed in the software engineering literature most often in relation to agile software development teams [8]. Although a comprehensive history or a comparison is not in scope for the dissertation, it is important to note that the concept of empowering software teams was not invented with the advent of agile methods; it has been mentioned before in connection with expert engineering teams [167], for example.

The agile perspective entered the world of software development in 2001 through the Agile Manifesto [13] and challenged the view of software development, from an activity focused on prediction, verifiability and control, to one focused on flexibility and responsiveness to change [149]. In the agile approach teams are self managed in that they decide how work is coordinated. This is usually contrasted to plan-driven approaches of coordinating work through a top-down style of management [155]. While a plan-driven approach has a clear separation of roles and sequence of actions between them, in the agile approach the team can decide on roles and change them as they see fit depending on their needs [102].

Even though the agile practices were first brought to light by software engineering practitioners, they are consistent with theories and principles found in organizational behaviour. Cao and Ramesh [31], reviewed agile practices in light of the Dynamic Capabilities Theory and Coordination Theory, to find that all common agile practices are congruent with the premises of the theories. While this is important to provide more credibility to agile practices from a theoretical perspective, the industry is not really looking for the theoretical foundations. The authors acknowledge that and propose that it is more useful to understand the conditions under which specific agile practices are likely to be effective. In their in-depth study of self-organizing roles in agile teams, Hoda et al. [102] found that roles emerged in agile teams in an informal and implicit way. The identification of roles revealed the needs of self-organizing teams, ranging from mentoring, to coordinating interactions with customers, to removing members that posed a threat to the teams' self-organization.

The self-organization of agile teams is seen as facilitating the responsiveness and management of changes [147]. Besides handling change, however, Moe et al. [147] found that autonomy has a positive effect on the team members, creating involvement, participation, motivation, and the desire for responsibility. The paper points out that

management is tricky where self-management is involved; on the one hand direction and checkpoints can make self-management effectice, but on the other hand rigidity can undermine autonomy. This trade-off shows a balance that is a challenge for organizations, how to balance autonomy with direction?

Self-management is not without problems, of course. In a different study, Moe et al. [148] identified barriers to self-managment due to conflicting goals and competitive behaviour. Drury et al. [62] examined how agile software teams make decisions, and found that conflicting priorities, not taking ownership of decision despite team autonomy, and unwillingness to commit to decisions were among the obstacles agile software teams face. These findings speak to the necessity for autonomy to have a company's full support, and the difficulty of *applying* self-management without it. As I describe below, the discourse in the agile literature seems to point to autonomy and agility being more than mere practices for teams and organizations to follow.

2.4.3 Autonomy and agility as mindsets

Reading on empirical studies of agile development in software companies makes it clear that *autonomous teams (agile or otherwise) are not leaderless or uncontrolled* [102]. Augustine et al. [8], reporting on industry experience, argued that self-managing teams have minor communication and coordination cost because internally they have optimal channels of communication. In their framework for managing agile teams they identify six factors: organic teams, guiding vision, simple rules, open access to information, lightweight management styles, and adaptive leadership. Augustine et al. ([8], p.86) place special importance on vision, stating that:

> *"Agile managers guide their teams by defining, disseminating, and sustaining a vision that influences the internal models of individual agents. The vision should be a simple statement of project purpose and communicated to all team members, which has been shown to have a powerful effect on individual behaviour".*

The description seems to elevate "being agile" to a mindset rather than a formal process, which agrees with discussions in the software engineering community pointing to a realization that "to be agile is a cultural thing" ([131], p. 203).

Notice that in the description above *a vision is not a directive or an order*. More and more literature finds that agile teams need to be involved in business strategy

more and have information about organizational goals. Van Waardenburg and Van Vliet [205] found that agile teams with no business involvement found it challenging to be successful. This suggests a needed change in business mindset, where information and knowledge is channeled to agile teams, and business alignment is part of their management. As a result, when agile teams plan their upcoming actions they have information about short and long-term goals and can decide on ways to achieve them [188].

Unfortunately, as much as Augustine et al. in 2005 stretched the importance of vision to guide autonomous action [8], years later Martini et al. [138] and others [205] still pointed out that agile teams lack strategic information and objectives, which hinder their awareness and appropriateness of decisions. They sadly concluded that (p. 45):

> *"Leaders' mindset in not open to listen and they are not able to recognize strengths and weaknesses. This hinders the development of improvements".*

Autonomy cannot be effective in providing agility if it is not embraced as a mindset and supported holistically in an organization [131]. In a recent systematic literature review, Dikert et al. [56] reviewed challenges and success factors of following agile practices in large organizations. Their findings — in the vast majority coming from practitioners' experience reports — indicate that companies still find it hard to implement agile and they resist the change. At the same time, the companies that have successfully embraced agile development cite management support for self-organization, and mindset and alignment as critical success factors. While the utility of autonomy is undisputed, however, it is still not incorporated holistically as the foundation of how work is done, and *companies are not sure how to do so.*

There is also an identified *need for management to operate in a way that establishes and communicates vision to agile teams*, while leaving them room to make decisions. These are the main issues I discuss through the empirical studies presented in Chapters 4, 5, and 6.

2.5 Autonomy is not the whole story

A synthesis of the literature I described in the previous sections points to autonomy being a desirable state for individuals and teams in software (and also other types of)

organizations today. It is not a panacea, however, and it comes with concerns and implications for organizations. Early on, theorists such as Cummings [49] proposed that self-regulating work groups should have "responsible autonomy" (p. 632) and an organic organizational context. This is not very different from more recent studies still advocating autonomy as a job characteristic for software developers and providing evidence that it is difficult to find a balance between flexibility and rigour in the organizational context [212].

Autonomy is even more difficult to implement effectively when it involves more and more people and teams, i.e. when it is implemented at scale. Not surprisingly, autonomy and agility at scale are current concerns for both research and practice. Reinventing software development work to incorporate more autonomy also creates questions about how management should function. I describe these concerns in the sections below.

2.5.1 Do autonomy and agility scale?

Agile software development methods were originally conceived for small, collocated development teams [22]. However, large organizations are also attracted to the benefits of agile development but find that the principles are a poor fit for them [163]. Whether and how agile software development — and its self-organizing principles — can be applied on a large scale has become of interest to both research and practice. A workshop at the International Conference on Agile Software Development (XP2013) in 2013 focused on large-scale agile development research challenges, in an effort to form a research agenda that guides researchers toward the areas that would be of use to practitioners. As experts voted, the top-ranking topic requiring more investigation was coordination of work between teams in large-scale agile development [57]. Knowing which agile practices scale and which not was another topic for more research; this related to issues about how knowledge sharing might be served better when a company grows.

Some reports from industrial experience indicate that autonomy and agility can work at scale. Kettunen and Laanti [112], drawing on the experiences in Nokia, identify enablers of agility at scale (resources, training, and empowerment). In their view, not all parts of an organization are set for agile operations, and that is affected more by human factors rather than the development process. For example, the authors highlight that if the hiring process in an organization is not one that attracts suitable

people, there can be problems with having software projects be agile. This points toward the culture and mindset behind agility, that needs to be nurtured and pre-existing for agile interventions to grow and be effective. Learning from their findings in multiple agile projects inside Nokia that follow agile principles in a fragmented way, Kettunen and Laanti argued for the need to view the organization holistically when talking about agility. Recent research [75] explores how a holistic approach to organizational agility might work, advocating the close connection of development to business goals (in what the authors coin as *BizDev*).

In a sense, the scaling of agility is facilitated by empowerment and decentralized decision making. By having teams own and make their own decisions handovers are reduced [119], and speed can remain high because it's not bogged down by communication and coordination overhead [138]. However, as Martini et al. point out [138] awareness of strategic information is critical. This indicates that the dissemination of information from executives and the ability to elicit and internalize bottom-up feedback are means to autonomy being effective and delivering the promised benefits; these learning are especially important for leadership and management. Companies like Spotify [6] and Atlassian [7] frequently share their experiences and opinions (see for example blog posts [8], [9]) on how agility can work at scale through a culture of empowerment and decentralization.

Earlier research has pointed out that typical agile team roles, such as the product owner, can scale, although with difficulty [161]. What I have presented in this section points to the need for typical organizational roles, such as managers and leaders, to behave differently so that they understand and embrace autonomy before they can show teams and individuals how to enact it. I discuss autonomy and agility at scale in Chapter 5 where I investigate the related challenges, and the culture that can help overcome them.

2.5.2 Management and autonomy

How does one manage autonomous entities? If individuals and teams assume a major part of the decision making for functions such as task definition and assignment, it indicates that the manager's role may be changing. In their study of management of

[6] https://www.spotify.com/ca-en/
[7] https://www.atlassian.com/
[8] https://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1/
[9] http://blogs.atlassian.com/2015/11/how-atlassian-builds-innovation-culture/

virtual teams, Liu et al. [133] pointed out a distinction between strategy formulation and strategy execution; the organization defines the strategy and the team implements it. This sounds like a clean line between what is management responsibility and individual/team responsibility. Yet, the authors go on to say that the alignment between the two areas of responsibility becomes critical and that is the new challenge for organizations that want to empower their employees.

From an organizational theory perspective, self-organizing teams are closely discussed with the concept of *minimum critical specification*: senior management needs to define *only* the critical factors that are needed to direct the team and place as few restrictions on the team as possible (Hoda et al. [102], quoting Morgan's book *Images of Organizations* [150]). Morgan, in his book [150], emphasized the need for what he called bounded or responsible autonomy, a term that comes from the work of sociologist Andrew Friedman. Friedman's work [79] investigated worker resistance and managerial counter-pressure as a process that moulds the organization, and analyzed models of responsible autonomy and direct control. In Friedman's view the *direct control* approach, with high degrees of restraint and direction on the actions of employees, comes from Taylor's scientific management and does not fit knowledge work. Alternatively, a strategy of "responsible autonomy" (p. 47) is needed, which aims to:

> *"...allow individual worker or groups of workers a wide measure of discretion over the direction of their work tasks, and the maintenance of managerial authority by getting workers to identify with the competitive aims of the enterprise so that they will act 'responsibly' with a minimum of supervision".* (p. 48)

Self-directed teams are not leaderless or uncontrolled; instead leadership is adaptive and light [37], focused on setting direction [5], aligning people [35], obtaining resources and motivating the teams [197]. Companies with autonomous teams tend to have flatter structures [157] and have managers act as leaders [5] that create and communicate a vision, and empower others to act on the vision. In the software engineering literature, agile managers are described as guides of their teams by defining, disseminating and sustaining a vision that influences the internal models of individual agents [8], a concept that sounds similar to the minimum critical specification.

For agile projects, a product owner is the person that sets the direction through managing customer requirements and creating a prioritized backlog [161]. Managers

of agile development teams do not themselves deliver software (as that is done by the team) but their role is that of giving the team a clear view of their goal and direction [5]. For software companies that do not build products at a client's request, the strategic direction is decided by executives that have goals they would like the organization to achieve. Increasingly, organizations are replacing or complementing financial organizational goals with goals of offering superior customer value and satisfaction [121], recognizing it as the source of competitive advantage [216].

These new realities seem to require managers of development teams to move away from producing technical output or making technical decisions, and step into a leadership role where they motivate and inspire their developers, while they coach them. I discuss the emergent attributes of engineering managers in a software organization that values and enacts autonomy, in Chapter 6.

Chapter 3

Methodology

"You know, a conjurer gets no credit when once he has explained his trick; and if I show you too much of my method of working, you will come to the conclusion that I am a very ordinary individual after all."

Sherlock Holmes – A Study in Scarlet

In this chapter I discuss the overarching methodological choices behind the studies presented in the next three chapters. Each case's methodology is described in detail in the corresponding chapter. Here, however, I give an overview of my research approach and explain my epistemological stance. My goal behind doing this is twofold. On the one hand, I separate the research approach – which is in spirit the same for all the studies I conducted – from the low-level details of data collection and analysis that are particular for each case. On the other hand, by explaining my research rationale I also frame how I wish the studies, evidence, and conclusions to be read. For example, readers who identify with the positivist epistemology – extremely common in computer science and software engineering – might disagree with (or even altogether disregard) the way I have collected and analyzed data, or the inferences I draw from analyzing it. The same readers would probably look closely for a formalized problem statement, hypotheses and testing thereof, or other elements of deductive reasoning. They may not do so, however, if they identify with approaches – long established in other domains – that aim to provide answers about different types of problems. Hopefully what I present in this chapter explains the nature of the research problem in this thesis, why the selected methods are a good fit to address it, and how to interpret the results.

3.1 Research approach

Creswell [43], in his book about research design, describes *research approaches* as plans that encompass formulating research questions and procedures for collecting, analyzing and reporting findings. The overall decision has to do with *which approach should be used to study a topic*; this means that the topic/problem/question takes precedence and all other decisions follow. Provided the researcher has at least a working version of the research topic they are investigating, according to Creswell there are three elements to a research approach: the *worldview assumptions* of the researcher, the *research design* relating to this worldview, and the specific *methods* to translate the approach into practice [43]. I elaborate on each of these elements below.

3.1.1 The research problem

This thesis addresses the problem of effectively combining autonomy and collaboration in modern software teams and organizations, and is scoped on the use of practices in line with agile methods. For reasons I presented in Chapters 1 and 2, the investigation of this topic is timely and significant for both the software engineering research field and practice. Software organizations are increasingly using agile methods [207], which advocate self-organization. The software industry is knowledge work-driven and, as such, can capitalize on autonomy's appeal for the motivation of knowledge workers. The practical, cultural, and managerial aspects of how to accomplish this, however, are underexplored and unclear. In addition, the software engineering field has long studied, modelled, and tried to support the collaboration of software teams. As a research field we are still looking for solutions to collaboration breakdowns though, and besides the shining examples of OSS development and some agile methodologies, the field hasn't tried to make autonomy the basis of collaboration. There lies the research topic I focus on in this thesis – which serves the role of Creswell's [43] research problem: *how can development teams in software organizations collaborate effectively when individuals and teams have autonomy in their development work?*

An additional issue — as I mentioned in Chapter 2 — is that autonomy itself is a malleable concept. Incorporating autonomy in how teams and organizations perform their work goes beyond following a dictionary definition; it becomes a matter of *what it means* in particular circumstances and people, and *how it is enacted*. Understanding such aspects, therefore, becomes a parallel goal while we explore autonomy's implications for collaborative software development work and how the two can be

combined. The research I present in the rest of the dissertation did not start with the explicit goal of studying autonomy. My primary interest was to investigate how several software teams and companies organize collaborative development work and how their use of tools, principles, and practices help them overcome various collaborative challenges. I gradually realized, however, that a number of choices the teams and organizations made related to incorporating autonomy in their work, and that there were implications from doing so on various levels. At the end the learning I have gained — and that I communicate through this dissertation — relates to the challenges and implications of combining autonomy with collaboration in software teams and organizations. That creates two different points of focus throughout the thesis; what I studied in the cases, and what I discuss in this dissertation. I elaborate more on this in 3.3.

Characterizing a *way of working collaboratively* is, of course, a complex problem. One needs to understand in-depth **how** software development is carried out in teams and organizations. It is also necessary to **understand the context and rationale** behind the observed practices; only then can we distinguish between what can apply to other teams and organizations and what is rooted in the realities of only one, or a few. Finally, **how** individuals and teams collaborate in an organizational setting is influenced by the philosophy and habits of the organization (reflected in its culture), and manifests in how it manages itself. Studying the *how*s and *why*s behind a phenomenon requires designs and methods for contextual understanding of a plethora of perspectives [43]; these are mostly associated with qualitative research designs, which is what I have followed in my line of inquiry.

3.1.2 Worldview, design, and methods

Worldview

The philosophical ideas behind research may not always be made explicit [187] but they nonetheless have an influence on methodological choices. Creswell [43] believes providing such information helps explain why a particular research approach is selected, however, and suggests that researchers should be explicit about their worldview. Although Creswell uses the particular term, worldviews come by other names too; epistemology is another popular name [44], while they are also referred to as paradigms [130]. In any case, worldviews represent "a basic set of beliefs that guide action" ([88], p. 17); this means that although they are a philosophical orientation,

the researcher brings them into their study by using them as their belief about the world and the nature of research.

Discussing the different wordviews that exist is not in scope for this chapter or thesis; there is literature on the subject and usually any textbook on research design devotes at least one chapter to them (for example, [43, 44, 85]). It is in scope, however, to present my own worldview (following Creswell's advice), which has influenced the methodological choices behind the research described in this thesis.

I subscribe to the **constructivist** or social constructivist worldview (usually combined with interpretivism). Social constructivists believe that individuals develop subjective meanings of their experiences, through which they understand the world in which they live and work [44]. The meanings are varied and multiple, and hence researchers look for the complexity of the views and the context within which they have developed [43]. As a result, the goal of the research is to rely as much as possible on the participants' views of the situation being studied. The questions are broad in nature to allow participants to construct meaning; the more open-ended the questioning the better. It is important to point out that in such a view, researchers recognize that their own backgrounds shape their interpretation, and their intent is to make sense of the meanings others have about the world. Rather than starting with a theory (as is the norm in the positivist worldview), researchers generate or inductively develop patterns of meaning, or theories. Crotty [44] identified that for constructivists the basic generation of meaning is always social and arises out of interaction with a human community. In line with this, the process of qualitative research is largely inductive, where the researcher generates meaning from the data collected in the field.

Such a worldview is almost the norm in social sciences; fields like anthropology and psychology have made their discoveries through research that is congruent with this view. At the same time, this view is in stark contrast to the positivist view which is almost the norm in computer science. There is some level of dissonance here, in my view. The deterministic philosophy of positivism – where causes determine effects – is in line with the mathematical training of the software engineering community, but not necessarily the socially complex demands of some problems of software engineering practice. Is the worldview that is the foundation of the natural sciences the only fitting way to research software engineering? Or should we, when discussing software development teams, their communication, coordination, workflows etc., remember that we study *people* operating in social and organizational environments, and hence

employ methods and techniques that capture aspect of these environments? I discuss this further in section 3.2.

Research design

Given the worldview I described and the nature of the studies, it is not a surprise that my mode of enquiry has been mostly qualitative. Qualitative research designs have stemmed from anthropology, sociology and the humanities and are a good fit for studies that focus on people and their habits, culture, beliefs, or interaction with their environment. Of the different available qualitative research designs I have selected the **case study** design of enquiry. Case studies look at bounded systems, and allow researchers to develop an in-depth analysis of an activity, process, or group [43]; this is the result of collecting detailed information using a variety of data collection methods over a sustained period of time [221]. At the same time, I have borrowed elements from other qualitative research designs; I have organized and processed data borrowing from grounded theory's principles, and I have collected descriptions and configured them into stories borrowing from narrative research.

At different points in the case studies presented in Chapters 4, 5, and 6, I used methods of analyzing results that come from quantitative modes of enquiry. There were practical reasons for doing so, and I explain the specific details in the corresponding chapters. However, I do not consider my research approach to have been quantitative in nature, although it could be argued that it resembles a mixed methods design at times [85], combining qualitative and quantitative data. I do still consider my approach largely qualitative, driven by the type of research topic and the contextual nature of concepts. This is another demonstration that the choices between different designs and approaches do not always fall neatly in mutually exclusive categories [43].

Research methods

Research methods involve the forms of data collection, analysis, and interpretation for studies. Usually with qualitative methods researchers allow behaviours and perspectives to emerge, and the data collection and analysis methods reflect that. As I discuss in detail in each of the following three chapters, I collected data through **observations, interviews, questionnaires with mostly open-ended questions**, and **review of different types of documentation**. I analyzed the data by using

qualitative coding and processing techniques that grounded theory has developed for organizing unstructured data. On occasion, data collection and processing were happening in parallel, and the emerging findings informed the next steps in data collection.

Once data reached saturation, in my interpretations, I have looked for **themes and patterns** – typical in case studies and grounded theoretic work. In my analysis, I have been cautious to not just categorize and group similar concepts, but to note and understand perspectives that came from isolated accounts too. In an inductive manner, I have generated meaning from the data I collected, I have used the literature as an aid to understand and/or explain the emerging patterns, and I offer a theoretical construct – a framework – that can have wider reach than the individual cases I studied. In my synthesis of the case studies (Chapter 7) I have followed analogical inference, whereby noting the shared properties of two or more things, one can infer they also share some further property [136]. This process is very frequent in the humanities and is similar to case-based reasoning [3].

As Creswell points out [43], the worldviews, the designs, and the methods all contribute to a research approach that *tends* to be quantitative, qualitative, or mixed. I used this section to explain the worldview that has informed my choice of particular designs and methods, in line with the nature of my enquiry. Given the focus on people and the way they work, as well as my constructivist worldview, I have followed a (mostly) **qualitative approach**, through a design based on case studies that have allowed me to collect rich, contextual information about the participants, their perspectives, their practices, and their environment.

3.2 The spectrum between a quantitative and qualitative approach

In one way or another, textbooks and articles on research methodologies explain that choosing between a quantitative and a qualitative approach is not really a dichotomy (for example [12]). Researchers are urged to not entirely discard the one or the other approach but rather to consider, in a continuum between strictly quantitative and strictly qualitative approach, what better fits the research problem they are addressing. Most research in the software engineering discipline leans toward the

quantitative end of the continuum. Although there are traditional and legitimate reasons for doing so, one thing to consider is what may be dangers of only relying on designs and methods of a quantitative nature to draw ones conclusions. This, I feel, is even more important when we discuss the human aspects of software engineering and we study developers, teams, and organizations or communities.

One potential danger is that the phenomena under study may not lend themselves well to be studied in a laboratory setting or other controlled environment [6] (citing [67]). In my case, to understand the context and nuances of software development work it is critical to get information about the natural setting or that work as it undoubtedly is part of the decisions and practices development follows. Through the case study methodology and the interviews I conducted I was able to discuss and zoom in on concepts that would be hard or impossible to capture in a closed-ended questionnaire or a controlled experiment. I was also able to better understand interactions in and between development teams through observations.

A related danger is that, for matters that involve *people operating in social contexts*, methods such as quantitative data mining or lab experiments can give questionable results. This is not because the methods themselves are questionable, of course, but because the loss of context may hide factors that are potentially more important than the ones mined. I have raised this concern previously in another study [110]; there we identified a series of perils that researchers may face when mining GitHub because of the way the data is organized. The perils were not explicitly known but, nonetheless, presented a risk of making erroneous inferences if they were not accounted for. By contrasting mined information with information obtained through a questionnaire and interviews, we found that the interpretations from mined data alone did not always reflect the real purpose of projects or the degree to which they were collaborative. The conclusion was that while data mining techniques are helpful to uncover information about developers' activity and projects' evolution, to make inferences about collaboration we need qualitative data. The work in [110] and the knowledge I gained from it has influenced the particular methodological decisions I describe throughout this chapter.

A final danger is that strictly quantitative approaches scarcely (if at all) capture the *perceptions* of participants in a study. Perceptions are extremely important in the study of social or organizational events, activities, or other phenomena, and they increase the researcher's understanding of underlying conditions [86]. What is more, perceptions *are* the participants' reality; the beliefs the participants hold about their

environment and the practices around them end up affecting their behaviour [175], and the way they perform within the organization [104]. Software engineering has started taking into account the software developers' perceptions to further understand their productivity [144] and the role of their disposition in accepting new development methodologies [16]. This is an improvement over a strictly positivist view on software engineering, and in my studies I have used a variety of qualitative methods to elicit the perceptions of those involved in the software development process.

3.3 Overview of studies

Each study in the dissertation corresponds to a chapter, but I provide an overview here. While there is a sense that all research questions were formally defined and decided upon at the outset of the research, this has not been so. As is characteristic with qualitative research, the research design evolved as I gained knowledge from the results, and as important issues emerged that were previously unknown to me.

As I mentioned before, the research I present and reason about in this dissertation did not start off as an explicit study of autonomy. My focus was on exploring how teams and organizations incorporate various trends in how they organize software development work and what that means for the way they collaborate. Autonomy surfaced in various forms in the studies, and it revealed implications for how entities collaborate. For the narrative of the dissertation that means two things. One, that as I understood more about the meanings attached to autonomy from study to study the associations about it changed; I have signalled this in the titles of the corresponding chapters and I address it in the text. Two, that the questions the chapters address are abstractions of the research questions the studies investigate; I provide some initial information below, but leave the specifics to describe in the corresponding chapters. In short, *there is a distinction between what I studied and what I discuss in the thesis*. In the end I have synthesized the findings to generate learning about how autonomy and collaboration can be combined effectively, given they are both important to software organizations today.

3.3.1 Research questions and the link between chapters

I provide some context for each of the studies below, while leaving the elaboration on them for the respective chapters. Figure 3.1 is the visual representation of the

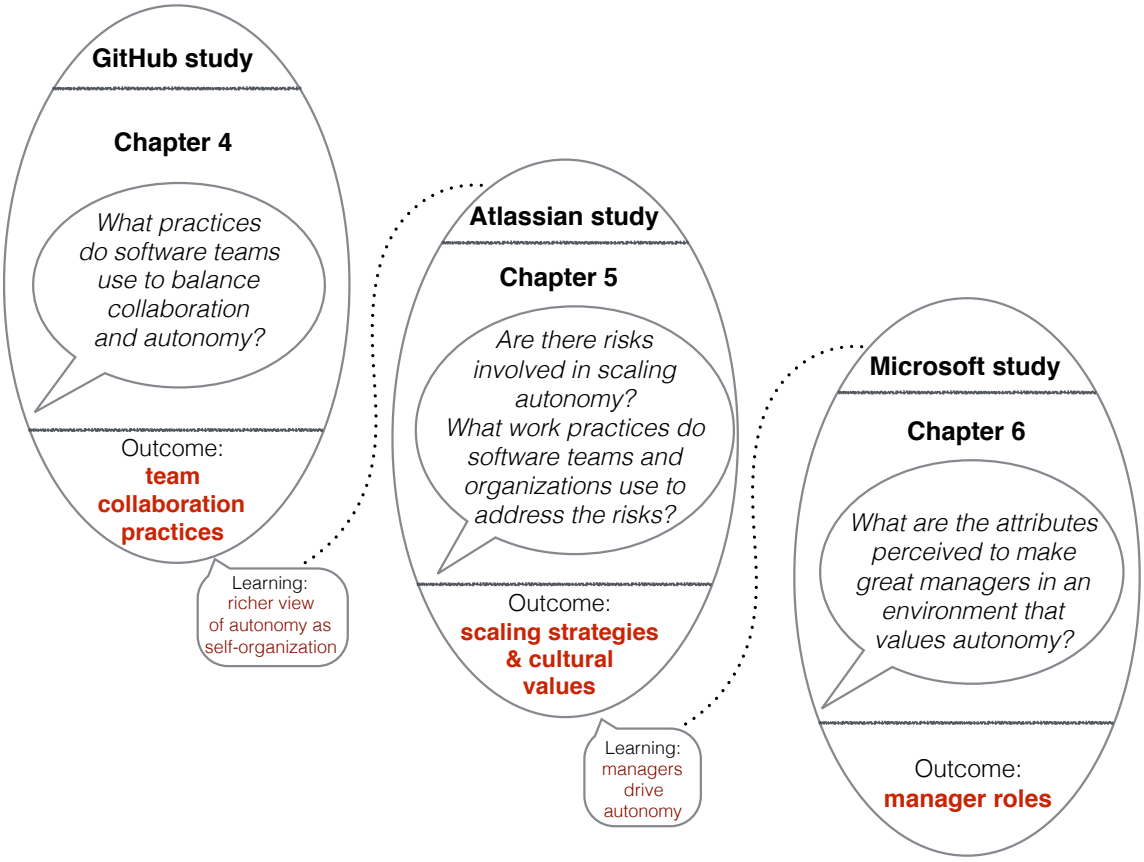

Figure 3.1: Visual representation of the three studies and their respective thesis chapters, and how they connect to each other. Each oval contains the name of the study, the relevant chapter and the question it addresses, and the outcome of the study as it contributes to the *collaboration via aligned autonomy* framework (discussed in Chapter 7). The speech bubbles show emerging lessons that informed the following studies. The dashed lines show links between chapters. Note: the particular questions that guided data collection for each study are presented in the corresponding chapter.

chapters and the links between them. Each chapter is represented by an oval, with the name of the study at the top, the question addressed in the chapter, and the main outcome of the study. The dashed lines represent links between the chapters. I explain Figure 3.1 throughout the following sections.

The GitHub study

The GitHub study started in the summer of 2013 and was conducted over 6 months, until the end of the year. One of the main drivers behind the study was GitHub's increasing popularity. GitHub is a code hosting service and code management tool, based on the version control system, Git. Git already gained favour with many large

and well-known open source projects (starting with the Linux kernel) before GitHub's launch, but was considered difficult to use and learn. GitHub built a easy-to-use platform around Git; users could have the benefits and powerful workflows behind Git, while using a simple and intuitive interface. GitHub's popularity gave open source projects exposure conducive to community building and attracting newcomers. GitHub also embedded social features in its design; allowing users to follow other users, watch repositories of interest, and get updates on ongoing activity. The term "Facebook for developers" that was informally associated with GitHub, speaks to its popularity, but also the social networking effect it created for software developers.

GitHub also popularized a contribution process known as fork and pull; developers copy (fork) the central repository and can work on it in isolation from the ongoing development work, and submit their changes by issuing a pull request. A pull request signals that there are submitted changes to the central repository which, after review, can be merged by a project owner or maintainer. GitHub and the fork and pull workflow make the individual developers independent while they work on their fork; this has clear utility for the usual practices of open source projects. Reports that companies are also adopting GitHub, however, made it unclear how pervasive it can be with commercial software teams that — admittedly — have different habits. The GitHub study set out to understand what practices commercial software teams use alongside GitHub's features to collaborate, and what the effect of using Github is on their collaboration.

In Figure 3.1 I have provided the high-level research question of the chapter; the question asks *what practices software teams use to balance collaboration and autonomy*. The study question in Chapter 4 is refined to capture all the elements at play: GitHub and its features (capturing autonomy as independence), practices, and the effect on collaboration (capturing the interplay) as the participants perceived it. In the GitHub study, autonomy as a concept has a narrow scope; it is equal to independence.

The study was based on the interviews of 30 participants, and the conclusions about commercial software teams were based on 24 interviews with representatives of an equal number of commercial software teams in organizations that were primarily small, and using agile methods. The outcome of the study was a series of **team collaboration practices** that teams used in conjunction with GitHub's features that allow independence; they cover all elements of collaboration, and the corresponding effects that the participants identified. After reflecting on the study's findings it became clear that autonomy is a richer concept than I originally thought, and resembled

more self-organization.

The Atlassian study

The Atlassian study started in the autumn of 2014 and was conducted over a year, until October of 2015. One of the main drivers behind the study was Atlassian's successful rapid growth. Atlassian is a successful company, building software products for collaborating teams, with a focus on software development teams. Their tool suite includes BitBucket, which is a competitor to GitHub, and offers similar functionality. The organization's use of agile methodologies under conditions of rapid growth made it an interesting case to study, especially because agile practices have been questioned about their fit with large-scale operations. The Atlassian study set out to understand the challenges of an agile software organization that grows rapidly, and the work practices it follows to address them.

The initial findings showed that development teams at Atlassian were using the same collaboration practices in the GitHub study, albeit with using a different tool. This links the GitHub study to the Atlassian study, shown by a dashed line between the two ovals in Figure 3.1.

Individuals within teams, and teams within the organization, are afforded high levels of autonomy at Atlassian; I elaborate on this in Chapter 5. This is not a surprise, as agile methodologies (and the entire movement) are associated with self-organization. Given the findings of the GitHub study and the strong influence of the agile mentality in Atlassian, in that study my conceptualization of autonomy was as self-organization. Early on, the interviews and observations revealed that while the practices were working well, there was general concern about how practices, that involve autonomy, scale. Hence, the Atlassian chapter is focused on understanding if there are *risks involved in scaling autonomy, and what work practices software organizations use to address them*. This is shown in Figure 3.1.

The study was based on on-site observations, interviews, and a review of various types of documentation over the course of a year. The findings showed that Atlassian tries to maintain and enhance autonomy as it grows by ensuring some degree of alignment of teams to strategy, to avoid inefficiencies. The way the organization balances between high levels of autonomy and alignment is a shared culture; the outcome of the study was a set of **scaling strategies**, and **cultural values** that facilitate the balance. An additional lesson out of the study was that people in

managerial roles are key to driving autonomy; by clearly communicating the vision of the organization, they help create a shared mindset around it that acts as the boundaries within which entities can be autonomous.

The Microsoft study

The Microsoft study started in the spring of 2016 and was conducted over 4 months, until August 2016. One of the main drivers behind the study was the growing push toward self-organizing software engineering teams (for example the work of Hoda et al. [102] discussing the roles involved). This raised questions about the implications for the engineering manager role and what aspects of that role are considered important. I already knew that managers play a part in the effective use of autonomy (the link to the Atlassian study in Figure 3.1). Microsoft has also gone through a cultural shift and has placed importance on empowerment and a growth mindset; the belief that anyone can develop their potential and that talents – as well as ideas – are not fixed, and failures (albeit, controlled) are learning opportunities. In the Microsoft study the conceptualization of autonomy grew into the notion of empowerment.

Given the push for self-organization and empowerment, the focus of the study was to understand *what are the attributes of great engineering managers*. Early on the practices and circumstances that were described to me about how teams work and what role managers play showed that there is indeed autonomy given to engineers and teams. Having established that autonomy is indeed enacted – and, also, showed up as an important attribute – I interpret these attributes as important for managers in environments where entities are autonomous.

The emphasis in the study was on gathering perceptions. There are various reasons for this which I present in detail in Chapter 6. The chapter, then, addresses the questions of *what are the attributes that are perceived to make great engineering managers in an environment that values autonomy*. The study was based on 37 interviews with developers and managers – middle and upper level. Based on the interview findings, I created a conceptual framework with functions and levels of interaction of great engineerings managers. I, then, used the conceptual framework to design and deploy a survey to a larger population inside Microsoft to assess how the findings generalize beyond the 37 interviewees. Based on the survey, I created a ranking of the attributes by importance, and probed for similarities and differences in the views between different demographic groups in the population.

The outcome of the study has been a framework of **engineering manager attributes that characterize the multiple roles managers play**, along with evidence of demographic differences in views, and examples and strategies used by managers in the context of software engineering specifically, in an environment of autonomy.

3.3.2 Synthesis

While each study in the thesis is self-contained in its design and findings, the framework and conclusions I present in Chapter 7 build on all three of them. This is done through evidence synthesis [58]. Evidence synthesis is frequently used in systematic literature reviews, in the form of meta-analysis [25]. One broad distinction of types of synthesis is between *integrative* and *interpretive* [58]. An integrative synthesis is primarily concerned with summarizing data and has been used more for pooling quantitative evidence for comparable phenomena. Interpretive synthesis uses induction and interpretation to bring concepts identified in different studies into the development of higher-level concepts. Extending previous work, Dixon-Woods et al. [58] described several techniques for performing evidence synthesis. One of the techniques, *meta-ethnography* has been developed specifically for synthesizing qualitative studies through three strategies: reciprocal translational analysis, refutational synthesis, and lines of argument synthesis. Broadly speaking, the three strategies form a discussion of the similarities and differences between themes and concepts between studies, to build a general interpretation grounded in the findings of separate studies.

In all three studies I conducted, the underlying theme and focus has been to *combine autonomy with collaboration*, making the studies fitting for evidence synthesis. I have already provided some links between the studies in my overview above, but I elaborate further in Chapter 7 on the similar themes between them. In an inductive manner, I then aggregate the findings of the different studies into a theoretical construct, in this case a framework. Finally, I discuss implications of the findings and how the study findings can apply beyond the particular studies [72], as well as what are the limits of the generalization.

3.4 Outcomes and lessons learned

This dissertation offers potentially useful findings for both theory and practice.

First, I propose a theoretical framework that combines *autonomy* and *collaboration* for commercial software teams and organizations. The framework is grounded on the empirical evidence from three main studies; in total, I have analyzed the practices of over 60 different teams in over 25 organizations. Through analysis and synthesis I have identified four areas – team collaboration practices, scaling strategies, cultural values, and manager roles – that require attention if autonomy and collaboration are to be combined effectively and used as a way of working. Each dimension is further broken down into specific approaches that play a role. I discuss theoretical and practical implications in Chapter 7.

Second, the recorded, contextualized cases, practices and examples I have gathered are data, but also an offering to the growing body of knowledge on collaborative software development practices in organizations. Several studies that investigate aspects of software engineering in organizations (for example, [191, 209]) urge for more studies of how software engineering is conducted in practice, as research is seen as lagging in this respect.

Finally, the participants in the studies discussed the practices they follow in handling the identified areas along side benefits, successes, challenges, and failures. While these associations are contextualized, and therefore cannot be entirely *prescriptive*, I have provided rich pictures that can be *descriptive* enough to allow the readers to recognize if something has potential to work for them. Especially smaller companies that do not have the resources or wealth of experience to try and err multiple times before they model their collaboration effectively, could use the findings from the studies and thesis as guidelines or, at least, food for thought.

3.5 Summary

In this chapter I have described my worldview and the overarching research design I have followed in defining the research problem, breaking it down into questions, and designing case studies to investigate each question. In the following three chapters I present each of the three case studies, with its own methodology and findings (Chapters 4, 5, and 6). Subsequently, in Chapter 7, I present a study synthesis that brings together the findings to form the framework of *collaboration via aligned autonomy*.

Chapter 4

Autonomy as independence — The GitHub study

"There should be no combination of events for which the wit of man cannot conceive an explanation."

Sherlock Holmes — The Valley of Fear

This chapter discusses practices that commercial software teams use to balance collaboration and autonomy. The discussion is based on an empirical study of commercial software teams using GitHub as their collaborative development platform.

GitHub is a popular online code hosting service built on top of git, a decentralized version control system (DVCS). It provides an open development environment and visibility of project activity through both notifications and a simple interface. Its transparency promotes increased awareness and reduced communication [52], giving GitHub the potential to mitigate challenges teams face in collaboration, such as coordination and communication breakdowns [33], lack of awareness [181], and code conflicts [182]. In fact, GitHub's motto is "collaboration without upfront coordination" [1]. At the time of this study GitHub was host to more than 20 million repositories and had over 9 million users, including some of the largest Open Source Software (OSS) projects, such as Ruby on Rails, Bootstrap, and Node.js.

[1] https://help.github.com/articles/using-pull-requests/

4.1 Why study the use of GitHub

The motivation for studying the use of GitHub in commercial software teams was based on a variety of interesting aspects of GitHub. First, it is intriguing that GitHub claims to mitigate coordination hurdles in today's largely distributed teams. Many a team have faced coordination breakdowns [99], while coordination is regarded as the biggest challenge in collaborative software engineering [6]. Yet, GitHub's implementation of "collaboration without upfront coordination" seems to work in practice, evident by its wide adoption. Through a surge of researcher interest in recent years, we have learnt about the use of GitHub as a hub for software development activity. Dabbish et al. showed that GitHub's transparency allows developers to learn about others' progress, creating a social coding environment [52]. Marlow et al. [137] discussed further that activity traces help developers form impressions about projects to contribute to and developers to collaborate with. Tsay et al. [201] corroborated the importance of GitHub's social aspects, reporting that characteristics of the discussion around pull requests are associated with the decision to accept them. GitHub also seems to have impact on development practices such as testing [168] and the choice of contribution model [84], through its technical features.

The second motivation for studying the use of GitHub is that the insights the software engineering research community has mostly come from OSS projects hosted on GitHub. That may not be surprising, as GitHub's workflow and development model seem to fit the OSS-like practices which are aligned with GitHub's motto of minimized coordination. Such practices include independence, autonomy in decisions about organizing work, and lightweight communication [90]. These practices are not usually found in hierarchical software organizations. Yet, reports on social media (for example [115, 140, 23]) point to a fast-growing number of commercial projects now using GitHub as a development environment, including large corporations such as Lockheed Martin, Microsoft, LivingSocial, VMware, and Walmart. How does GitHub support the collaborative development in these projects? As commercial organizations do not allow public access to their code base, and use private repositories or deploy GitHub internally, we — as researchers — do not have an answer to satisfy our curiosity.

At this point, the closest I was to thinking about autonomy was as a degree of independence, meaning that entities do not have to coordinate their actions with other entities.

4.2 Case study setting and methodology

The choice of development practices has implications for the collaboration process of a team, and even though certain decisions are technical in nature, they have organizational impact. For example, the use of Version Control Systems (VCS) targets the management of the codebase, but also provides the basis for the team's workflow by outlining how contributions are handled. I use the term *workflow* to refer to a development protocol, meaning a set of rules and steps that team members are to follow when managing, reviewing, and integrating changes to project artifacts.

GitHub builds on the concept of decentralization of development work. In a decentralized workflow, a project repository acts as the reference point. Collaborators, even without write-access to the repository, can fork it creating their own full copy of it (a "clone"), and work on their tasks independently. When finished, developers open a pull request to signal they have changes in their clone and, usually after inspection, those can be integrated by a project maintainer. A decentralized workflow also includes branching strategies for members with write-access to a shared repository to still separate their activities by purpose, submit their changes through pull requests, and merge frequently. As a result, GitHub's workflow is such that it allows for independence during development, separating the developers' workspaces and activities and allowing them to make their own decisions in isolation. This independent form of work that requires less coordination provides a new paradigm for commercial teams, but the mechanisms through which decentralized workflows frame collaboration have been largely unexplored.

Understanding collaboration practices in teams using GitHub to produce proprietary software in commercial organizations is equally interesting and important because practices of closed source projects do not readily seem tailored for the GitHub development environment of independence, visibility of work and self-assignment of tasks. Often developer participation in closed source projects is by invitation only — developers are assigned tasks and, in extreme cases, are only authorized to work on the code associated with their tasks. On these projects, developers are also typically not aware of code changes made outside their assigned code areas [208]. Yet, commercial proprietary projects are adopting GitHub increasingly.

4.2.1 Research goal and questions

The goal of the study presented in this chapter is to understand how proprietary software teams use GitHub, and how their use of GitHub supports collaboration. GitHub acts as a vehicle for independence in development work given its decentralized nature, and its emphasis on transparency of information and progress status. The particular way of organizing work and collaboratively developing software — using GitHub — has been widely adopted in OSS projects, where autonomy is a cornerstone of the collaborative model, and the two concepts have been co-existing successfully. Specifically the study's research question was:

RQ:*What practices do commercial software projects follow in conjunction with GitHub's features to collaborate? What is the effect of the combination of practices and features on their collaboration?*

4.2.2 Some useful concepts

Collaboration may sound like an abstract concept, and it is certainly difficult to pin down what "good collaboration" is. In the course of the study and analysis, I used five concepts to operationalize collaboration; coordination, communication, awareness, task division, and conflict resolution. Henceforth, these five key elements are referred to as *collaboration elements*. I synthesized the five elements after reviewing collaborative software engineering literature, as described in Chapter 2; they represent typical collaboration challenges in software development, and are accentuated for teams that are distributed. *Coordination* and *communication* often break down in large and distributed teams, and result in build failures [33] and longer resolution times [99]. Maintaining *awareness* of interdependencies and ongoing work is also challenging; tools can provide alerts of potential coordination needs and recommend communication between interdependent developers [181], but create overhead [32]. *Conflicts* occur even in the presence of awareness tools and teams spend effort to resolve them [182]. Efficient *task division* and scheduling can help to minimize, but not eliminate, coordination overhead and code conflicts [111].

In the study, practices and processes that address the *collaboration elements* are what makes up a team's collaboration, and "good collaboration" in my interpretation means that a team has some way (hopefully a good one) to address each element.

4.2.3 Data collection and analysis

I used a mixed-method approach in a qualitative study to examine collaborative practices of projects from commercial organizations using GitHub. Through an initial survey sent to 1,000 GitHub users, I collected responses from 240 participants. The survey polled participants about how they use GitHub to address each of the five *collaboration elements*, their reasons for adopting GitHub and the effect it has had on their development practices. From the pool of survey respondents that volunteered for interviews I selected those that indicated they participate in collaborative projects. I conducted in-depth, semi-structured interviews with the first 30 respondents on the list. After the first interviews I realized that several interviewees used GitHub primarily as part of their job. This provided a great opportunity to look into a type of projects that has not been investigated yet, although I did not get the chance to triangulate the interview results since I did not distinguish between commercial and OSS-related use in the survey. Out of the 30 interviews, 24 came from commercial projects and 6 from OSS projects.

The 24 interviewees from the commercial organizations are part of development teams that use GitHub without making their code publicly available. They do that through private repositories in GitHub's web-based interface or by utilizing GitHub's enterprise version which they host on their own servers. Throughout the study I refer to the 24 interviewed cases as "commercial projects" or "commercial organizations". In accordance with other studies (e.g. [89]), in the context of this study I use the term "commercial" to refer to *projects or organizations that produce proprietary software and do not make their software visible in public repositories or receive contributions by entities outside the commercial organization.*

To answer my research question, I analyzed the data from all 30 interviews. To distinguish the practices of the commercial projects I explicitly compared the information from the 24 commercial project interviews with those gathered through the other, 6 OSS project interviews.

Survey procedure

The project was approved by the University's Ethics Board, the certificates of approval are provided in Appendix A.

I acquired a list of recently active users through GitHub's public API. The primary goal of the survey was to recruit participants for interviews; it put me in touch and

gave me an initial feel of the audience. I kept the survey short and the questions straightforward to keep the respondents interested in proceeding with an interview. The survey consisted of open-ended questions, and is included in Appendix A.

I asked participants why they adopted GitHub ("What was the reason you decided to use GitHub?") and how it has affected their development process ("How has the use of GitHub affected your development process?"). I also included one closed-ended question which asked participants if they use GitHub primarily to collaborate with others or for solo projects. At the end of the survey, I asked participants whether they would volunteer for an interview.

I piloted the survey internally with members of my research group and externally with 100 GitHub users. I obtained 19 responses (19% response rate) and conducted trial interviews with four of these survey respondents. I used these pilot survey responses and interviews only to fine-tune the wording of the questions of the survey and build the interview script; their results are not included in the analysis. After this pilot phase, I sent a revised survey to 1000 GitHub users and received 240 responses (24% response rate).

Survey responses

I used thematic analysis techniques [40] to process the survey responses and understand the participants before proceeding with getting data from the interviews. I coded answers to open-ended questions into mutually exclusive categories. Given the simplicity of the questions and answers I did not feel it was necessary to include a second coder. However, a colleague acted as an external auditor and gave me feedback on the coding scheme I developed. Table 4.1 contains a summary of the coded survey responses. The label "N/A" corresponds to unanswered questions.

The majority of the respondents reported that they used GitHub primarily to collaborate. Regarding the reason the respondents adopted GitHub, I asked participants to provide their primary reason, and so received only one reason per participant. The most commonly cited reasons were the desire to contribute to OSS projects or share code, and prior experience with git.

The effect on development practices that participants cited most often stemming from the use of GitHub, was adopting a workflow of branching and pull requests. The participants also reported writing better code as a result of using GitHub, due to the self-consciousness that public visibility of their code creates. Adoption of

Table 4.1: Coded survey responses summary.

Survey item	**Survey responses**	**240**
GitHub use	Primarily for collaboration	148 (62%)
	Primarily for solo projects	90 (37.5%)
	N/A	2 (0.5%)
Reason for adopting GitHub	To contribute to OSS or share code	67 (28%)
	Because they used git already	52 (22%)
	To collaborate with others	35 (15%)
	To host projects and store files	26 (11%)
	GitHub's popularity	24 (10%)
	GitHub's interface / ease of use	24 (10%)
	GitHub was adopted at work	8 (3%)
	Other reasons	4 (2%)
GitHub's effect on develop-ment	Adopt branching workflow	53 (22%)
	Be conscious about writing better code	44 (18%)
	Adopt/learn best practices	43 (18%)
	Write more code / Contribute	41 (17%)
	Same effect as using any DVCS	31 (13%)
	N/A	28 (12%)

best practices was another effect; this included code reviews, frequent commits, and generally learning from looking at others' code. Respondents also noted writing more code and contributing more as a result of using GitHub. 13% of the respondents attributed any effects they saw in their development to using DVCS, not GitHub specifically.

Interview procedure

Eighty two survey respondents volunteered for follow-up interviews. I selected the first 30 volunteers that indicated they use GitHub primarily for collaboration and conducted one-hour long semi-structured interviews. All interviews were recorded and transcribed to facilitate iterative analysis.

To focus the interview questions, I started all interviews by asking the interviewee in what setting they primarily used GitHub (job-related or not). I then asked the interviewees' to describe their current job, their responsibilities, their team, and their typical workflow. I guided the interviewees to provide details about how they interact with other members of their team, the tools they use daily as part of their collaboration, and any challenges they face. In the remainder of the interview, the questions focused on the collaboration elements: *Communication, coordination, awareness, task*

division and conflict resolution. I asked how their team handled each element and how their use of GitHub affects each one. The questions I used for this part of the interview are shown in Table 4.3, while the full interview guide is included in Appendix A.

Similar to the survey responses, I used thematic analysis techniques [40] in processing interview data. I loosely grouped the data around the collaboration elements and performed open coding, in which I assigned hierarchical codes to interesting parts of the interviews. The code system developed iteratively, by coding, discussing codes with a colleague that acted as external auditor, combining and splitting codes, and writing memos about more abstract phenomena as they emerged. I then used axial coding [40] to iterate over the data I had collected, which allowed me to build the answer to the research question.

Table 4.2 summarizes some basic characteristics of the interviewees.

Table 4.2: Interviewees' and projects' descriptive characteristics.

	Interviewees	**30**
GitHub use	Primarily for job-related projects	24 (80%)
	Primarily for OSS contribution	6 (20%)
Job situation	Professional developers	25 (83%)
	Managers/CTOs (also do development)	4 (13%)
	Developers in internship	1 (4%)
	Job-related projects	**24**
Distribution	Distributed	16 (67%)
	Collocated	8 (33%)
	Median team size	**7**

Out of the 30 interviewees, 24 reported on *the use of GitHub in commercial projects*; 20 were professional developers and 4 were managers or CTOs that also participate in the development activities. The 6 participants that indicated using GitHub primarily for OSS projects also identified themselves as professional developers (5 out of 6) with the exception of 1 developer in internship. The majority of the commercial projects that interviewees were part of were geographically distributed (16 out of 24).

The *commercial project* interviewees are part of small development teams, the median team size in the group of interviewees was 7. Even in the cases that developers are working for large organizations running multiple projects with development teams having more than 10 members, they reported that a standard practice is to form subteams with a maximum of 4-5 developers working on any given project. When asked,

Table 4.3: Interview questions corresponding to collaboration elements and how GitHub supports them

Collaboration element	**Sub-area**	**Question**
Coordination	Definition	What does coordination mean to you? How would you define it?
	Example	Can you give an example of coordination in your team?
	Coordination Needs	What are the occasions that you find it essential to coordinate with your team?
	Issues/Solutions	Can you remember an issue with coordination? How did you resolve it? Do you have any conventions on coordination you follow in your team?
Task division	Criteria	How do you decide how to split the work between team members?
Awareness	Activity	How do you keep track of activity in your project?
	People/Artifacts	Do you prefer to track the activity of people or artifacts? Why?
	Maintaining Awareness	How do you stay aware of actions or changes in the artifacts?
	Challenges/Problems	Does maintaining awareness get too much? Is something missing?
Communication	Communication Needs	What are the occasions that you find it essential to communicate with your team?
	Challenges/problems	Does communication get too much? Is something missing?
Conflict resolution	Conflicts	Do you come across conflicts? How do you resolve them?
GitHub's support	Role	How does GitHub fit in with all those collaboration pieces?
	Benefits	How has GitHub helped your team collaborate?
	Usage evolution	Has your use of GitHub changed over time? How?
	Problems	Is there something missing? Does GitHub hinder anything?

these interviewees indicated that they are also active contributors to OSS projects, either by maintaining their own public repositories or by contributing code to others' repositories.

The participants described development practices that are in line with agile methodologies, although some highlighted that they adopt what practices are relevant for them and so refrain from calling their team or organization strictly agile.

4.3 How commercial software teams use GitHub to collaborate

In this section I present the findings that lead to answering the research question of the study, based on the data from the 24 interviews about commercial software projects. In the interviews I asked the participants explicitly to answer the questions based on the commercial software project they use GitHub for. In the discussion of the results, I compare the answers of commercial projects to answers coming from

the 6 OSS interviews.

The reported collaborative development practices from commercial projects focus on three underlying themes: enhancing *independent work* by using a branching workflow, reducing *communication and coordination* by using GitHub's visibility, and *self-organizing* for task assignment and conflict resolution.

4.3.1 Working together through independent work

GitHub offers two different development models (based on Git) resulting in different workflows. In the *fork and pull model* collaborators keep their own local copy (a "fork") of the repository and make changes there, issuing a pull request when done to inform the project maintainer about the new changes. Upon review the changes are pulled in the original repository. In the *shared repository model* team members have direct push access to the repository.

In the interviews, I asked participants to walk me through the workflow that their team is using. 23 out of the 24 interviewees reported that their team follows a workflow using pull requests. This pull-based workflow was reported in two flavours: a simple one (19 interviewees) and a complex one (4 interviewees), described below.

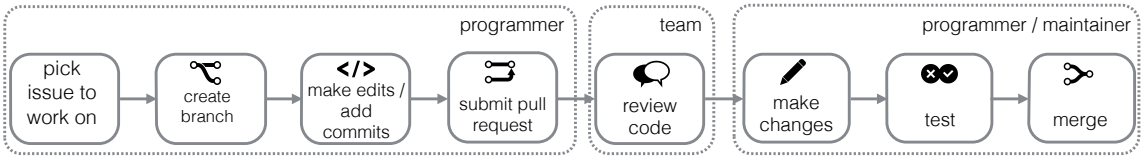

Figure 4.1: Branching, pull-based workflow used in 79% of interviewed cases. Another 17% used a more complex variation of this type of workflow.

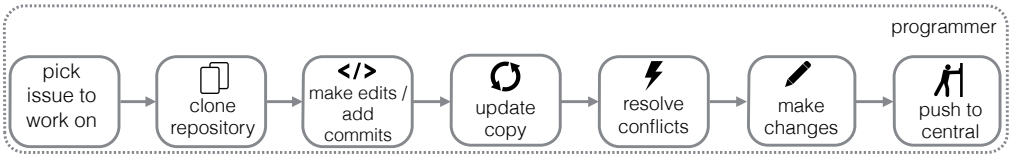

Figure 4.2: Centralized workflow reported by one commercial team in the study, using git commands instead of SVN.

Figure 4.1 shows the graphical representation of the workflow to which 19 out of the 24 (79%) interviewees converged. This workflow uses a central repository that belongs to the organization and then developers work on features on separate

branches within the repository. After the developer is finished with committing their changes to their local branch they submit a pull request, making their changes visible to the rest of the team for code review. Four interviewees reported using a more complex variation of the workflow in Figure 4.1 that uses branches for development, fixes and releases in addition to the feature branches, therefore helping with release management [2].

The remaining interviewee reported that his team is using a typical centralized workflow heavily influenced by SVN but using git commands (shown in Figure 4.2), as they had recently migrated and were not familiar with all of git's workflow capabilities.

The major difference between the workflows in Figures 4.1 and 4.2 is the use of pull requests.

> *"When I first started we all pushed to one branch, but the problem is you push and nobody knows what changes are going through and there is no chance for a review. Our current system is that you don't make any changes without submitting a pull request[...]and then one or more members will review it and you need one thumbs up from another team member to merge it, otherwise it can't go in. "* [P13 - professional developer in commercial project]

According to GitHub, the fork and pull model is popular with OSS projects for reducing coordination requirements. At the same time, the shared repository model — giving team members direct push access to the main repository — is a model considered suited to small teams and organizations [3]. However, surprisingly almost all the small teams and organizations in the study reported using a hybrid between the two models, a *branch and pull* workflow. This workflow allows having one repository for the project (as in the shared access model) but enable autonomous feature development within it through branches, and still use pull requests (as in the fork and pull model).

Pull requests were reported to have a dual utility: they signal coordination points, and they give the opportunity for code review. In Figure 4.1 I have highlighted the activities that fall under different roles, as the interviewees reported them from their practice, where individual work is intermittent with team work. A submitted pull

[2] all four cited the branching model at nvie: http://nvie.com/posts/a-successful-git-branching-model/

[3] https://help.github.com/articles/using-pull-requests/

request is the last action before the hand-off of the new contribution to the team to discuss and review and, therefore, a clear (and critical) point when coordination is needed. The result of the code review can indicate needed changes; in that case the programmer makes edits and adds commits and creates a new pull request that will go through code review. Upon reaching agreement either the contributor or the maintainer take the final actions before merging. In the branching workflow shown in Figure 4.1 there is explicit provision for the team to weigh in before new code is merged and that lowers the risk of skipping code review. That is in contrast to the centralized workflow the interviewees reported (Figure 4.2) where the developer works entirely in isolation before they push their changes and they show up in the central repository.

There is an even more important aspect to highlight here. Before submitting the pull request, *the developer is independent in their actions*; they can organize their development as they see fit within their branch and do not need to coordinate with others before they submit their changes. The *pull request then acts as a checkpoint for alignment of work*; by reviewing the submitted code, the team ensures that what the individual developer created independently respects and fits with the team's standards of quality, coding style etc.

Note that contrary to git being a DVCS supporting workflows that build on each developer having a complete copy of the repository and history as their individual development environment, this was not the way it was used in the commercial projects we interviewed.

4.3.2 Reduced communication and coordination needs

Table 4.4 shows the interviewees' preferred source of information to track the progress and status of the project. For the most part, GitHub's visible, automated output is preferred to avoid superfluous communication. Programmers reported getting the information they need through the issue list to see outstanding items, and the commits list to see ongoing activity. Notification emails were another option mentioned, although interviewees are conscious about information overload. IM was reserved for keeping interviewees aware of blockers; IM clients that integrate with GitHub (the most cited example was Campfire [4]) combine update messages with ongoing communication, used as needed without being intrusive. Interviewees commented

[4]https://campfirenow.com/

that updating the status of issues so that the newsfeed is current requires effort, but preferred this to having additional communication for awareness.

Table 4.4: Interviewees' preferred method of keeping track of activity.

	Awareness source	
Progress / Status	Issue list	6 (25%)
	Commit list	7 (29%)
	Notification e-mails	5 (21%)
	Chat client integrating with GitHub	6 (25%)

Interviewees recognized two needs for communication taking place on GitHub: questions for other team members when encountering problems, and discussions around specific coding items.

> *"As far as collaboration and communication, on GitHub most of the communication and collaboration has to do with one of two things: either concerns, or the aftereffects of a change."* [P6 - professional developer in commercial project]

Interviewees reported directing their questions to team members by using the @mention functionality on GitHub as their first line of communication (13 out of 24 interviewees — 54%). Typically, the use of the symbol @ in front of a username means that a message is directed to them specifically. However, if the user is not present in the communication channel and does not see the mention, they may not pay attention to the message. GitHub has implemented the @mention convention as a notification trigger, sending an email to a user that was mentioned relative to an artifact, drawing their attention and focusing communication as a result. Code-centric communication was reported to be successfully handled through GitHub comments: inline comments for code reviews, and issue, commit, and pull request comments for communication surrounding a specific artifact. For more elaborate conversations, however, the majority (20 out of 24 interviewees — 83%) reported moving to external tools such as an IM client, mailing lists, or having direct communication when applicable. The communication venues outside GitHub afforded the teams more text space and synchronicity to discuss ideas about the software, compared to discussing particular fixes through comments on GitHub. The switch between different communication tools for different purposes is a communication protocol that teams have grown into through continuous use, which helps team members know where they would find information or communication relevant to a particular subject.

All interviewees consider coordination essential for collaborative development. Their fear was that inadequate coordination can lead to duplicated work due to lack of awareness of others' activity, while too much coordination was seen as overhead. They found that a highly visible project status mitigates the danger of an awareness gap when development is happening independently, and replaces the need for communication and coordination. Also, a branching strategy dictates the steps of the process the team needs to follow and limits coordination needs to specific points, as was mentioned earlier with pull requests.

4.3.3 A touch of self-organization

Another practice commercial project interviewees reported was using self-assignment to divide tasks between them, based on their expertise and availability. This practice was reported by 16 out of the 24 interviewees (67%).

> *"Mostly we self-assign tasks over the sprint planning process, usually a person who has experience with the application will volunteer more so than someone who doesn't. Sometimes there is a task that is very very specific to an issue and one or two developers have worked on it in the past and they will take it up."* [P6 - professional developer in commercial project]

In the cases of self-assignment of tasks project managers and/or team leads still participate in the estimation and prioritization of tasks, to define them to be "bitesize" [P14 - professional developer in commercial project], but then developers pick which tasks they will work on. In 54% of the cases (13 of 24 interviewees) the team leads were the ones that followed the task assignment decisions the developers made to maintain awareness.

The reported benefit of self-assignment was that since team members know their expertise, if they self-organize to define and select tasks to work on, the team is collectively working with the optimum task division. Interviewees pick tasks from GitHub's issue list (only 6 interviewees — 25%) or the bug tracker, task management or project management tool they are using that integrates with GitHub but is an external tool (18 interviewees — 75%). The most used tools respectively were JIRA [5], Asana [6], and Pivotal Tracker [7]. The reason for not using GitHub's issue tracker was

[5] https://www.atlassian.com/software/jira
[6] https://asana.com/
[7] http://www.pivotaltracker.com/

that interviewees found it too minimal for their teams' needs.

> *"Pivotal gives us a lot of really great features and allows very fine grained control over the process of the ticket, every status of it along the lifecycle is kept track of. GitHub has more of an open-closed status."* [P34 - professional developer in commercial project]

The second area that involved self-organization was resolving conflicts. Interviewees use external continuous integration tools for tests and builds; these tools show where problems are, and then decisions are up to the team members on whether to solve the problem individually or collaboratively.

> *"We use a CI server and when it sees an update it pulls the code down, runs the test suite and if it's all green it will rebase that up to the master branch. If the test cases are failing we work together to try and see why it failed, or we send it back to whoever broke the build because you can't merge that code."* [P24 - professional developer in commercial project]

4.4 Discussion

After seeing how commercial development teams use GitHub in practice, there are several interesting aspects to discuss. In the sections below I summarize the practices we saw commercial projects use together with GitHub's enablers, and discuss their overlap with known practices from OSS development and how GitHub acts as a vehicle for their adoption in industry. Further than that, I discuss the interplay of autonomy and collaboration as supported by the study's findings, reflecting on the practices commercial projects use that enact autonomy in practice.

4.4.1 Collaborative practices and GitHub's features

At the beginning of the study I set a goal to investigate how GitHub — a DVCS with provisions for independence — serves collaboration through support for the collaboration elements. In the course of the study I recorded the practices of the commercial software teams and the corresponding supporting GitHub features. The answer to the study's research question is summarized in Table 4.5, where I map each practice that the interviewees reported as helping their smooth collaboration to the

corresponding GitHub feature that enables it. The third column shows the effect on the corresponding collaboration element as it was related with each practice by the interviewees.

Table 4.5: Practices reported by commercial projects, the corresponding GitHub enablers, and how they supported the collaboration elements in the study

Reported practice	**GitHub enabler**	**Collaboration element effect**
Independent development	Branching workflow (also mentioned as GitHub workflow)	Minimized *coordination* needs
Progress and status monitoring	Timeline, notifications, integration with chat client	*Awareness*
Code reviews	Pull requests with in-line comments	Highlighted *coordination* needs
Code-centric communication	Comments on commits, issues, and pull requests	Targeted *communication*
Self-organization	Public issue tracking	*Task Division*
Automatic testing and deployment	Service hooks in corresponding tools	*Conflict Resolution*

As the table demonstrates, GitHub fulfills all five collaboration elements, which is one explanation of its popularity for collaborative development. Based on what the interviewees reported, especially regarding *communication* and *coordination* — long standing challenges of collaboration — GitHub has a positive influence, by focusing communication on artifacts, and by minimizing and signalling the essential coordination points through its decentralized workflow. Developers are able to be independent for all other phases of the workflow, and well-coordinated at the phases they need to, avoiding communication and coordination overheads.

4.4.2 GitHub as a vehicle for adopting best, OSS-style, practices

Overlap between commercial and OSS practices

The practices that we learnt from the study that small commercial teams follow are not new. They have been known and used in OSS projects.

Commercial projects on GitHub use pull requests in the same way as OSS projects do: to isolate individual development and perform code review before merging. Pre-

vious GitHub studies have shown that pull requests are used in OSS projects roughly as much as shared access to the repository [84]; they provide an opportunity for code review when the pull request is received [84], and they work as a screening mechanism when open source projects receive contributions from new participants [137]. This was also confirmed by the OSS interviewees:

> *"Even if you are not sure if the other dev is capable of contributing good code, you can review pull requests and if the fifth pull request is good you give him/her commit bit."* [P3 - professional developer and OSS project founder]

> *"In the case of Fedora, if anyone is making a significant commit to a project, even if they have commit access because they are on the team, they still open a pull request and the code isn't merged in unless somebody gives it a +1. So, we use the pull requests to review code before we merge it in."* [P16 - professional developer in OSS project]

In this study, interviewees from commercial projects did not mention explicitly using pull requests as a screening mechanism; it is natural to assume this is because there is already established familiarity or trust between team members due to their existing experience working together.

Regarding coordination and communication needs, OSS interviewees reported using GitHub as a communication platform the same way as commercial projects do. The emphasis was on GitHub's transparency and having communication be code-centric, with pull requests acting as a coordination mechanism.

> *"Through github specifically there is not really a messaging system, so everything is down on a commit or a pull request or something like that. We don't really use github as a direct communication thing unless directly talking about bits of code and pull requests. We usually find that IRC or email works better for side notes. Like I said you can't send a PM on github."* [P16 - professional developer in OSS project]

Dabbish et al. [52] have reported similar findings for OSS projects on GitHub. They found that comments are used by OSS projects for direct feedback and code review, and that activity information flows across the site in the form of updates. These findings agree with what is already known for OSS projects in general [218].

Finally, self-organization is a long known practice in OSS projects [45]. Self-assignment of tasks has been pointed out as a difference between how OSS projects and commercial projects handle task-actor dependencies [47]. From the study's evidence we see that the practices may not be that different in some cases.

I summarize a comparison between the practices reported for commercial projects using GitHub and OSS projects in Table 4.6. To avoid repetition I only include the first column from Table 4.5. I give examples of where the same practices have been reported for OSS projects; from the study's own 6 OSS interviewees or from previous studies of OSS projects, using GitHub or not.

Table 4.6: Overlap between reported commercial project and OSS project practices. The entries starting with P represent study participants. The bracketed entries represent bibliographical references.

Reported practice	**Also in OSS projects**
Independent development	P3, P16, [84]
Progress and status monitoring	[52, 51, 137]
Code reviews	P16, [84, 176]
Code-centric communication	P16, [52, 90]
Self-organization	[45], [46, 47]
Automatic testing and deployment	[168]

Beyond the tool: using GitHub in place of process

For some software organizations (usually small in size) the primary focus is on making their daily operations of collaborative development more effective. That means looking for solutions and best practices for their development and collaboration. The evidence showed that GitHub is a viable option as a Collaborative Development Environment (CDE) in small software companies; they made use of the pull-based workflow to have code reviews, and GitHub's integration of tools and information, typically found in CDEs.

First, the adoption of a pull-based workflow for a company's proprietary software projects was the study's most surprising finding. So far OSS projects have used the fork and pull development model to have control over the quality of the contributed code through incremental code reviews and screening new contributions [202, 84]. GitHub offers an easy-to-use implementation of pull requests and their use as a code review mechanism through in-line comments to commercial projects too. Besides the fork and pull model, GitHub also supports a development model where all collaborators

have direct commit access to a main repository. For small teams inside companies, where development is restricted to a predefined group and members have established trust through prior collaboration, the shared repository development model seems more fitting (asserted also by GitHub [8]). However, the teams in the study used a hybrid between the two models: the team maintained one repository to which all collaborators had access but made the decision to use pull requests between branches.

This *branch and pull* variation has advantages over both the fork and pull and shared repository development models, in a company setting. The fork and pull model assumes multiple forks of the original repository. A team member or a project manager needs to view all the individual forks if they want to see ongoing activity, which makes it less traceable than using multiple branches that are visible in the main repository. At the other end of the spectrum, the simple shared repository model gives team members direct commit access. It still leaves traces of activity after commits have been made but less distinctive than a pull request. The teams in the study recognized better traceability of changes and the ability to do code reviews as the benefits of pull requests over the simple shared repository model; both benefits are especially important for development teams that continuously deploy and ship code.

Second, GitHub serves as a CDE that integrates tools and information under the same interface. Leveraging the visibility of information, team members can have focused, artifact-based communication. More detailed conversations were handled through external tools, like IM clients that integrate with GitHub and show committing activity. Although interviewees didn't mention breakdowns, future work can look into whether the use of external tools has negative side effects for transparency, team knowledge management and discovery. The commercial teams I studied practiced self task-assignment; this is not a GitHub-specific capability but can still function because team members view all updates regarding who is working on what.

I should note that this is not the only recorded occasion of organizations adopting OSS-style practices for their proprietary software development. Literature shows that large organizations are interested in promoting open collaboration by opening project participation to all employees in the organization and use OSS-style collaborative development practices to do so, a trend termed Inner Source [191].

[8] https://help.github.com/articles/using-pull-requests/

4.4.3 The balance of autonomy and collaboration

The question driving this chapter had to do with *the practices that software teams use to balance collaboration and autonomy.* The study I described gave an account of how a tool (GitHub) and process (decentralized workflow) supported autonomy (in the form of independence) in development while effectively supporting collaboration. Beyond the specific mapping of practices to features and effects (Table 4.5) I would like to discuss the interplay of autonomy and collaboration at a higher level, relating to the role of transparency and visibility of information, and the approach of having processes become standardized organically. Through this discussion I am also placing the concept of having *collaboration via aligned autonomy* in the more general discourse, beyond strictly software engineering.

Transparency makes centralized information visible

Through the study we saw that commercial projects, through their use of GitHub, are replicating open source-style practices in their software development. GitHub's collaborative environment complements autonomy — which happens in isolation — with transparency and centralized information. Teams in the study also used the transparent, integrated environment of GitHub as a ground to practice self-organization; self task-assignment was supported by having one central, integrated space of activity and information, where developers can see updates about open and closed tasks and what other members are working on.

In the domain of management, Martine Haas's work [92] argues how the combination of autonomy with knowledge that is external to the team makes self-managing teams more effective. She describes the main risk with autonomy to be that teams or individuals may become isolated, form almost a cult with "not invented here" syndrome, and resist paths that — although not optimal for their narrow scope of tasks — are beneficial for the organization. In Haas's view having external knowledge addresses the risk, something that is echoed by other work agreeing that autonomy and empowerment need information sharing to function effectively [173]. GitHub's branching provides isolation as we saw in the study. The visibility of the information about ongoing activity and progress is the external knowledge Haas posits makes autonomy effective. I will explore this idea more in the following chapter when I discuss the risk of autonomy becoming isolating and practices to avoid it.

What the study findings suggest is that autonomy is not limited to the narrow-

scope I originally conceptualized for it, that of independence. The teams' task self-assignment practice indicates that autonomy was enacted as self-organization, which broadens its scope to include the freedom of organizing one's own work. With the solid ground of transparency and visible information, autonomy becomes effective and supports effective collaboration, as was indicated by the study's findings.

Agents of change toward a de facto tool and process

An additional finding in the study was that developers who were already using GitHub and work in commercial projects act as agents of change, recommending the use of GitHub in the workplace. While this may start with individual developers and teams, it can end up gaining ground within the organization in a subversive manner, making the workflow and process that seems to be organically adopted with GitHub (remember that independently the vast majority of the interviewees reported the same *branch and pull* workflow) gains ground. Previous work has shown that peer interaction, in the form of recommendations and observations, can lead to tool discovery [153] and adoption [217].

In the study I saw cases where the final decision to adopt GitHub came from the bottom up but also from the top down.

> *"The reason that we moved to GitHub was because I was using it personally and so at that point I approached the company and I said that we are using git and it'd be nice to have this functionality."* [P28 - professional developer in commercial project]

> *"I decided we should use GitHub and part of it is picking a tool that a lot of people are going to be familiar with."* [P21 - CTO in commercial software organization]

This form of cross-pollination is not just between teams in the organization, or individuals in teams, but also between the organization and the OSS community. GitHub has formed a community around its service, with large and prominent OSS projects. Due to the community's openness, common and best practices are visible to all. Even if companies are not using or participating in the GitHub social coding environment for their proprietary software development, they can still observe it. One of the OSS interviewees remarked:

> *"I think it's really interesting how all the projects collaborate on there and I think when companies move their code to GitHub they become aware of other projects that are on GitHub and they pay attention to how those projects do things and so everybody who host their project on GitHub eventually starts to use the tool in almost the same way and they realize the benefits of the social coding aspect with all the comments and the pull requests and stuff like that."* [P16 - professional developer in OSS project]

As the GitHub environment itself is heavily influenced by OSS development and collaboration practices, it spreads OSS-style processes to commercial projects too. Through the consistent use, both the tool and the process can become established *de facto* standards in an organization. This *de facto* process can be seen as a manifestation of *autonomy while accepting guidance*. This (on the surface curious-sounding) combination has been described in social psychology and the self-determination theory of motivation. Chirkov et al. [36] argued that autonomy and dependence are not opposing concepts, and they used it to describe persons that take actions of their own will (autonomy) and follow guidance at the same time (dependence); these persons willingly rely on others' direction, especially if they perceive them as supportive and responsive. I will further explore this idea in the following chapter to discuss the role of cultural values in an organization to drive this sort of willing dependence in an environment of autonomy. As it stands in the case of this chapter and study, having processes evolve into *de facto* standards sounds like a manifestation of autonomy with guidance, and it describes developers who have buy-in for decisions and follow suggestions, in line with what we know about knowledge work.

4.5 Threats to validity in the study

The study faces the threats to validity that qualitative studies face in general. A construct validity threat is the interviewees' understanding of the concepts of collaboration and coordination, especially given their code-centric perspective. I mitigated this risk by asking interviewees for examples and by providing explanations when needed, to ensure a mutual understanding of the constructs. A further construct validity threat comes from GitHub building on top of git and the unavoidable significant overlap in their functionality. The interviewees may not have made an explicit distinction; on my part I specifically focused the questions on GitHub and due to

that I believe that the results are valid despite the overlap.

In terms of internal validity, the participants in the survey and interviews were active but self-selected, and I can only rely on what they report given their time and motivation to participate in the study. Finally, in terms of external validity, I achieved saturation in the results of the participants I studied but the themes, principles, and practices I found may not generalize to everyone. The majority of the participants were professional, experienced developers which gives confidence that the insights are valuable and relevant.

4.6 Conclusion

In this chapter, I presented a study exploring the use of GitHub by software development teams in commercial organizations. The study provided evidence of commercial software teams following a model of *collaboration via aligned autonomy* with a workflow that combines autonomy with coordination points, builds on transparency and visible information to help team members align in their efforts, with positive influence on the collaboration process. This led to a discussion about the balance of autonomy and collaboration and how the two concepts can be combined in practice.

What we learned in this chapter is that, at least in the narrow scope of independence, autonomy appears to be compatible with collaboration. We saw that with the combination of tool, features, and practices in the study that provide autonomy, all collaboration elements are fulfilled.

The type of autonomy I described and the corresponding communication and coordination practices worked for the small teams and organizations in the study. It's not clear, however, if it is possible for autonomy to work at a larger scale and still serve collaboration in a adequate manner. Or does autonomy break at some point and the risk of isolation becomes dangerous?

I explore these aspects in the following chapter where I describe a study that shows *collaboration via aligned autonomy* happening at scale in a software organization experiencing fast growth, and the scaling issues that come with it.

Chapter 5

Aligned Autonomy — The Atlassian study

"There is nothing more deceptive than an obvious fact."
Sherlock Holmes — The Boscombe Valley Mystery

This chapter discusses potential risks when scaling autonomy, and work practices that can be used to address the risks. The discussion is based on an in-depth study of Atlassian and the organization's challenges and practices in handling rapid growth.

Atlassian is a well-known software company building tools to support teams — not limited to software development teams — that work (or wish to) in an agile way. At the same time, Atlassian uses its own products internally and agile practices that it tries to scale in two ways. One, the adoption of agile-minded practices goes beyond software development across the organization, including functions like Legal and Marketing. Two, Atlassian is growing in size and expanding in scope at a rapid rate which means that the work practices the organization follows need to accommodate a growing number or people and projects. That includes the self-organization that is part of the agile mindset. While originally agile methods were scoped to suit small collocated teams [213], nowadays there is consideration given to how agile principles may scale to fit large-scale software development [162] and support organizational agility [112]. The study of some of Atlassian's growth pains and how it scales its agile practices can help us understand more about these aspects.

5.1 Why study Atlassian

Several reasons led to selecting Atlassian as an environment to study.

First, it is an agile organization throughout, producing software for agile teams, and educating others in how to be agile[1]. Part of the strong culture of agility is supporting employees to self-organize, giving them autonomy in various ways, as I will present later.

Second, Atlassian practices open collaboration across all its departments and functions. Initiatives like the internal quarterly hackathon — ShipIt[2] — regularly give developers room to innovate and bring down barriers between teams, products, and departments. This quarterly event aside, however, Atlassian has all information be open internally and shared across the organization, with room for employees to contribute to projects that are outside the scope of their particular task; this is not a frequent practice one hears about in large companies.

Third, Atlassian is a successful organization, well-known for its fast growth and innovation. In Gartner's 2015 report on evaluating Application Development Lifecycle Management (ADLM) Atlassian was ranked a leader in the market (next to Microsoft and IBM), in both ability to execute and completeness of vision [80]. As a high-functioning organization, it seems reasonable to look at Atlassian to understand growth-related challenges and learn from how the organization balances growth with agile operations or — as I will analyze it — autonomy at enterprise level.

Finally, Atlassian has a strong relationship with its employees through its culture, evident by Atlassian being voted first and second best place to work in Australia and the US respectively multiple times — see the link for the results in August 2016 [3]. Atlassian's core values[4] carry strong messages through simple words. The Atlassian case gives a chance to study agility at scale, and discuss the interplay of growth, autonomy and culture.

5.2 Case study setting and methodology

I visited the San Francisco office in October 2014 to study the software teams' "growing pains" and the work practices they use to overcome them. Specifically, the study's

[1] https://www.atlassian.com/agile
[2] https://www.atlassian.com/company/shipit
[3] http://reviews.greatplacetowork.com/atlassian
[4] https://www.atlassian.com/company/values

research question was twofold:

RQ: *What are the challenges agile software organizations face while growing rapidly? What are the work practices followed to address the challenges?*

To this end, for two weeks I conducted workplace observations and interviews with various employees and members of management. From the start of the project and for one year I was also given an organizational email account, was granted access to internal systems, and received three editions of the company's internal employee satisfaction survey, including all response data.

5.2.1 Getting acquainted with the teams

I was embedded in the team that develops Bitbucket, Atlassian's code management tool and code hosting service. I invited team members to interviews in the first stand-up meeting I attended, while the Head of Engineering for Developer Tools gave me access to the team's calendar to select upcoming meetings I wanted to attend. I was also given a desk in the Bitbucket team's work area and I observed their interaction and practices, kept notes, and reviewed team discussions and documentation on the respective tools, HipChat and Confluence (detailed in the following section).

Atlassian has created a Marketplace around its tools, for users and tool designers to sell custom add-ons and extensions. I was introduced to the development leads for the Core and Purchasing teams. These used to be one Marketplace team but were recently split, and the leads introduced me to both teams. Finally, to broaden the potential reach of the project and conduct as many interviews and meeting observations as possible, I wrote a blog post introducing myself to the San Francisco office, giving details about the project and asking for participants. The blog post was circulated through the company extranet, and is provided in Appendix B.

5.2.2 Data collection and analysis

The project was approved by the University's Ethics Board, the certificate of approval is provided in Appendix B.

The data for this study was collected through a variety of methods. I conducted observations which were documented through systematic note keeping. I also conducted interviews which were recorded, with consent from the informant, and then transcribed. The meetings I attended were also recorded, with consent from the attendees, while I also kept notes throughout. Using the access to the internal systems

I'm often asked what Bitbucket licenses are in Greenzone and if there are any missing.	https://extranet.atlassian.com/display/DATA/questions/2450567811/how-do-bitbucket-licenses-get-into-hams-and-greenzone?src=browse
It seems to me that the Standing List Desk is really only a "Placebo Page" to make Atlassian newbies *think* they could get a standing desk (once they've been 'long-standing') So let's auction Natasha Prasad's desk tonight to raise funds for Room to Read! Vote Yes or No or add your own answer!Also see:Who do you think should get my standing desk when I leave?	https://extranet.atlassian.com/questions/2450569113/should-we-auction-natashas-standing-desk?src=browse
whilst trying to ssh dpabst.jira-dev.comi get the following error message:	https://extranet.atlassian.com/questions/2450570865/why-do-i-get-an-error-about-my-rsa-key-when-i-try-to-ssh-into-a-unicorn-instance?src=browse

Figure 5.1: Sample of log kept for accessed pages in the internal Q & A space

I performed documentation review; I kept notes and screenshots of pages that were important in showing how the teams work and how they plan and coordinate their actions. I also kept a spreadsheet of the pages I accessed in the internal Q & A community space, and the questions that were on them. A sample is provided in Figure 5.1

The notes served as an extended memory of the things I heard and observed during my two-week research visit. The interview transcripts were coded and analyzed in line with grounded theory techniques for analyzing unstructured data [40]. One thing that facilitated the coding process was the goal of recording challenges and practices to address them. This provided some existing structure to organize the coding. My previous study of software companies using GitHub focused some of my questions on how Atlassian development teams approach the collaboration elements (a detailed comparison is provided in Appendix B). These areas of practice mapped to interview questions, and then the coding process revealed the themes under each area. As I processed the interviews I kept a list with the reported work practices linked to the use of reported tools. Added to the list were the practices that I observed, and those that showed up through the review of the documentation.

For the practices I also coded the purpose and rationale that was discussed around them. A stable working schema formed before I finished processing the interviews, indicating saturation, although I continued to analyze all the remaining interviews. As the goal of collecting practices facilitated and simplified the coding process there didn't seem to be a need for a second coder. This ended up serving pragmatic issues as

well, as all data that I processed was confidential and required signing a non-disclosure agreement.

I was also conscious of emerging concerns in the interviews. It became evident quickly, for example, that interviewees were content with the way they were working. The higher up in the hierarchy the interviewee was however, the more they expressed either current limitations or uncertainty about the best way to scale practices and work habits. Another observable surprise was that culture was inadvertently mentioned by almost all interviewees; by the descriptions it seemed to underline everything that was happening in the company. Both aspects are featured in my analysis and reasoning about the case.

As an overview — and to provide an audit trail — Table 5.1 provides a list of all the reported work practices in the Atlassian case study. The practices are mapped to the corresponding data sources in which they were recorded, indicated by checkmarks. Note that the presence or absence of a checkmark is not an indication of strength, i.e. a practice with 4 checkmarks is not more important or more valid than one with 2 checkmarks. For example, the practice that builds are self-managed by teams did not surface in the interviews because I didn't ask about it, while it was present in the other data sources. In the rest of the chapter I will be discussing these practices and the context in which they emerged.

Table 5.1: List of all reported practices in the Atlassian case study. A check mark means that a practice was recorded in the corresponding data source; Interviews (marked with **I**), Documentation (**D**), Observations (**O**), and Satisfaction Survey (**S**). The presence or absence of a checkmark is not an indication to the practice's importance or validity.

Reported practice	**I**	**D**	**O**	**S**
Developers use branches for their work on a task or feature	✔	✔		
Developers select the task they are going to work on each sprint	✔	✔	✔	
Developers submit pull requests when they have changes ready	✔	✔		
Developers make changes on their branch	✔	✔		
End-to-end ownership of a team's workflow	✔	✔	✔	
Deployment to production is handled by the development team	✔	✔	✔	✔
Submitted pull requests need two reviewers	✔	✔		
The backlog is decided by the development team	✔	✔	✔	
Builds are self-managed by teams		✔	✔	✔
Development teams define their own scope and timeline	✔	✔	✔	
Teams include engineers, designers, devops, and support engineers	✔	✔	✔	
Access to repositories is open (internally in the organization)	✔	✔		
Micro-services architecture		✔		✔
Ubiquitous personas		✔	✔	
Offsites notes are visible		✔		
There are ways to provide feedback on decisions on products	✔	✔	✔	
Teams follow a pull-based workflow	✔	✔		
There are organizational roles to communicate product directions to teams	✔	✔	✔	
Progress and status of all teams is visible		✔		✔
Other teams' mission and progress is transparent	✔	✔		
Teams give regular demos of their work in progress	✔	✔	✔	
There are templates, and design and coding guidelines available		✔		
There are regular synchronization opportunities between teams	✔	✔		
Employees move around different teams and learn and learn first hand how they work	✔	✔		
Employees rotate roles and learn what are the responsibilities and pain points	✔	✔		
Teams share their successes and failures with other teams	✔	✔	✔	
There is cross-pollination in tools and processes between teams	✔	✔		
Decisions on key products directions are transparent	✔	✔		
Teams practice code review	✔	✔		✔
There are impromptu synchronization opportunities between teams	✔	✔		

5.3 The agile environment at Atlassian

I provide a detailed description of the product development process that Atlassian teams reported, in Appendix B. Table 5.2 shows an overview of the teams' product development process, broken into phases, with the corresponding purpose, the communication mechanisms, and the tools used by the teams.

The teams and product managers stay updated regarding the organization's current strategy — and hence, the high-level objectives — through the Vision, Targets, Focus, and Metrics (VTFM) document. The VTFM is written and communicated to the organization by the co-founders, and any employee can comment on it. The product development process starts when the VTFM objectives have been translated into a roadmap and goals for the development team. More information on the VTFM and the cascade from the founders to the product teams is provided in Appendix B.

Table 5.2: Overview of the product development process followed by the Atlassian teams in the study. The process is broken down to phases, and each is mapped to the purpose, as well as the communication mechanisms and tools the teams use.

Phase	**Purpose**	**Communication mechanisms**	**Tools used**
Ideation	Propose what the team should build next	team meetings	Confluence
Requirements building and Design	Decide on specifications, describe goals, estimation, planning, initial designs	team meetings	Confluence, JIRA
Implementation	Code development, redesign, testing, dogfooding, deployment	code-centric communication, comments on artifacts, demo meetings, design walls	BitBucket, JIRA, Crucible, Bamboo
Special weeks	Focus on the team and its operations, innovation, bug fixing, performance improvements	team meetings, comments on artifacts	Confluence, JIRA

5.4 Challenges in scaling an agile environment

During the interviews I asked the participants to reflect on challenges they come up against in the way they work. I also reviewed challenges mentioned in the internal documentation, the employee satisfaction survey, work management documents and posts on the internal blog. In the rest of Section 5.4 I review the reported challenges, and in Section 5.5 I present the work practices Atlassian uses to address them. I provide representative quotes where appropriate, indicating the participant's ID and organizational role.

Atlassian employees were vocal about challenges, in the spirit of one of Atlassian's core values of open and honest communication (expressed as the "Open company, no bullshit" value). Open communication is a claim often made by organizations, but case studies [92] show that even when open communication is encouraged politics are prevalent [156] and influence what people share and with whom [105].

Atlassian's case seems different. Frequently blog posts discuss aspects of the organization's life or techniques and processes that can be improved, and they approach the subject by providing rationale, recommending solutions and being open to questions and comments from the rest of the organization. The same is true about the internal satisfaction surveys, on which many employees provide constructive feedback besides rankings. I interpret this as an investment on the part of the employees to identify and solve problems instead of hushing them up, and in that sense Atlassian gives the impression of operating like a community as well as an organization.

The overarching challenge reported at Atlassian is finding ways to "grow up without slowing down", as multiple managers and executives expressed it. This is not a problem particular to Atlassian; finding ways to preserve and scale up whatever constitutes excellence in an organization while addressing the inevitable delays that come with increased activity is a common theme with growing organizations in various domains [196]. Two areas surfaced as attracting the most consideration; **maintaining efficiency while growing**, and **having effective autonomy at scale**. Atlassian's work practices are aimed to address the issues, and are evaluated and reconsidered regularly. Some practices become obsolete quickly with growth, and the organization goes through cycles of introspection and interventions, while working full speed toward attaining its organizational and business goals.

5.4.1 The challenge of maintaining efficiency while growing

Atlassian is experiencing evolution in several ways at the same time. First, the organization is *growing in size*; founded by 2 people in 2002, Atlassian now has around 2000 employees and almost doubled the number of employees between 2014 and 2016. Second, the organization is *expanding in offering*; starting with JIRA in 2003, Atlassian is currently offering a tool suite of more than 15 products, both as web services and on site installations. Finally, Atlassian is *extending its scope*; once only targeting software teams, Atlassian is now promoting its tools to teams like Marketing and Legal, as well as provides training and certification in agile methods

using its tool suite. The rapid growth and expansion is full of opportunities, but creates a challenge for the organization to remain nimble in its operations.

One such case is the increased volume of development activity which stretches the demand on infrastructure for testing, builds, and other deployment activities. These areas experience slow system performance, but also become bottlenecks due to handoff of responsibility between teams — the development team needs to hand off responsibility to the build team, for example — with the overhead of coordination [162]. The sentiment in the satisfaction surveys was that the resulting delays are frustrating to developers, and in their view prevent new features and improvements from reaching the customers as quickly as possible. Performance engineering (activities related to optimizing the performance of operations) was the area that engineers were most unhappy with in all three of the internal satisfaction surveys of March, June, and October 2014. In terms of ranking, Performance Engineering got negative Net Promoter Scores. In terms of comments, quotes such as the one below (from a respondent to the anonymous satisfaction survey) capture the general sentiment when explaining a low ranking:

> *"Build engineering is a problem (build instability, either due to tests or build environment). Also, waiting for sysadmins/IT to respond to requests."*

The comment shows challenges in systems performance and bottlenecks created by needing to include other IT teams in the middle of the workflow.

5.4.2 The challenge of autonomy at scale

Atlassian provides its teams the opportunity to self-organize. Development teams choose their own development process, build their own roadmaps, and come up with their own ideas for features to build. Even inside the development teams, individual developers make decisions on how to organize their work: they can choose their own tools, self-assign tasks, and follow their own timeline for development. With self-organization happening at a larger scale, however, some limitations of autonomy begin to show.

Autonomy may overlook dependencies

Atlassian product teams have the ability to decide what they release to their customers and when. For example, the Bitbucket team does not pre-commit to dates toward customers, and there is no pre-announcement that certain features will be released on certain dates. The team decides when a feature is ready to release to customers and it uses its autonomy to serve its high quality standards; if the feature is considered subpar it will not be released. However, while the team makes its own autonomous local decisions, other teams might be affected. Functions like marketing and legal need to produce work based on what the Bitbucket product will include when released and, therefore, last-minute changes have repercussions. The Head of Engineering described an incident when due to a last-minute decision to not release a certain feature for BitBucket, there were complications for the corresponding marketing campaign which ended up being cancelled, with the associated financial cost. He concluded:

> *"Not pre-committing to dates is a thing of beauty but can also create problems."* [P10 - Head of Engineering for Developer Tools]

The challenge Atlassian is facing with autonomy at scale seems to be *balancing between the autonomous decisions of the individual product teams and the underlying dependencies that other teams may have with them.*

The balance between a rigid or fluid process

Following formalized processes is often seen as potentially stifling creativity or innovation in teams [18], and that is usually one of the arguments in favour of self-organization. While functions that require a high degree of creativity need the freedom to explore, other operations can benefit from variance reduction to make them more efficient [19]. As the conditions for operational tasks — like continuous integration for example — become more predictable and repeatable, performance can improve. The Head of Risk Management explained this view:

> *"Process and flexibility are not necessarily opposites. Process is not evil, bad process is. Maybe if you have more process where you need less flexibility, you can afford more flexibility where you want to be innovative."* [P15 - Head of Risk Management]

The same participant explained that areas such as reporting or licensing are supporting and should interfere as little as possible with areas that need the room to be innovative and experimental, like product development and design. This comment is similar to the challenge I described earlier about deployment activities; those activities are also supporting development and Atlassian is looking for ways to automate or make them faster.

The challenge then becomes to *identify the right processes to automate or standardize* to avoid or compensate for the slowness that comes with the complexity and growth of operations.

5.5 Work practices at Atlassian

Throughout the study I collected work practices Atlassian uses. Alongside practices for software development, I recorded all work practices that the interviewees described as supplementing the software development ones; these are followed in general in the organization and relate to information sharing and communication habits. The practices are already listed in Table 5.1, and I describe them in detail in the following pages. The practices I have captured are by no means the sole practices used in the organization. A comprehensive list was out of scope for the particular study and thesis; the goal is to present the practices the organization uses that address the challenges, and discuss them in terms of handling autonomy at scale.

5.5.1 How Atlassian tries to maintain efficiency at scale

Maintaining development speed and efficiency under conditions of rapid growth is an important challenge for Atlassian and is discussed in this chapter. The particular solutions that have to do strictly with *system performance* are out of the scope of the thesis. I have grouped the work practices Atlassian follows to maintain efficiency at scale under three themes: providing autonomy, supporting independence, and establishing conventions.

Practices to provide autonomy

During implementation the individual developers have decision making freedom; the practice is that developers ***select tasks for themselves***. Developers on the BitBucket team described this practice taking the form of developers volunteering during

the planning meeting for existing issues on JIRA, or for currently discussed tasks still on Confluence. Unless someone brings up a different alternative the assignment is recorded on Confluence and JIRA. This task self-assignment practice is not mandatory for all teams; the BitBucket and Purchasing teams practice it while in the Core Marketplace team the development lead assigns the tasks to developers.

Next, by following a branch and pull workflow, ***developers make changes on their branch***. Using the branch as their personal workspace, developers are independent in organizing their development, including the tools they use to do so.

Two other practices that provide autonomy have to do with the team ***defining its own scope and timeline*** and ***deciding its backlog for the sprint***. The team is essentially working with two backlogs at any given time; the product backlog (represented by the team roadmap) and the sprint backlog. The development team uses multiple sources of ideas to create its roadmap for the next 6 months (more details in Appendix B). *The resulting roadmap — both its content and its timeline — is developed through the developers' ideas within the frame of the business strategy.* A program manager explained this kind of bounded freedom:

> *"Deadlines are internal, when we want to ship. Developers and product managers decide what is going to go into the pipeline.* VTFM *also sets mission, value and focus areas for the year and that will play into the priority. Sometimes it's not even a deadline it's more this is the date that we're going to aim for. Not the end of the world if we miss it."* [P12 - Program Manager]

For the sprint backlog, development teams have a free hand to prioritize and select features to implement, picked off the roadmap.

Practices to ensure independence

I mentioned above the use of individual branches as providing autonomy for each developer. However, ***developers also use branches for their work on a task or feature***, to isolate work on different tasks. As we saw in the previous chapter this practice is used in both open source and commercial settings [109]. The developers' independence is also serviced by ***submitting pull requests when they have changes ready***; that way they don't directly change the codebase with their work but make it visible to others — and allow for review — before it is merged.

Atlassian moved to continuous integration in the interest of speed, automating most of the deployment process. This decision eliminated Quality Assurance (QA) teams, making development teams ***responsible for their entire workflow***. As a result, a team can go from objectives and iteration planning, all the way to delivering the finished product to the customers without hand-offs to other teams. Teams ***handle deployment to production*** as well as **builds** as part of their development, a practice that makes them more independent and, potentially, faster. Beyond the practices and rationale that were described by developers and managers in my conversations with them, Atlassian has been vocal about their culture of "do-ocracy" [5] as a way to maintain development speed by decentralizing certain operations. The argument there — as has been described earlier in sources like the Mythical Man Month [26] — is that additional coordination, such as required because of responsibility hand-off, can result in delays; therefore, by making teams responsible and able to handle the workflow from idea to delivery themselves they only have to be concerned about their own performance and speed.

Atlassian believes that ***teams being self-sufficient in skills required to complete their work*** helps maintain speed. Product or development teams are broken down to ad hoc feature teams. The Head of Engineering for Developer Tools described that feature teams are made up of developers, a designer and a Development Operations engineer (DevOps). DevOps engineers are concerned with the performance of the systems the development team is using, such as its release pipeline. Both designers and DevOps engineers may be working with multiple feature teams at a time. By having access to all the skills required for its activities, the team doesn't rely on other teams or departments to complete its work, or seek approval to proceed. This, again, means that additional coordination with other teams and possible delays in the hand-off process, are avoided.

Finally, ***using a micro-services architecture*** instead of a monolithic one is an architectural practice that promotes de-coupling and independence. Monolithic application architectures are deployed as a whole; changes either need to stand by to be deployed in the next release, or the entire solution needs to be deployed. Developers cannot work independently then, since all development and redeployment efforts need to be coordinated. Continuous deployments and frequent updates are difficult, and the product can become slow to respond to market or competition shifts. A micro-services architecture on the other hand is developed as a set of smaller indepen-

[5] https://www.atlassian.com/atlascamp/2015/archives/developer-best-practices/coding-culture

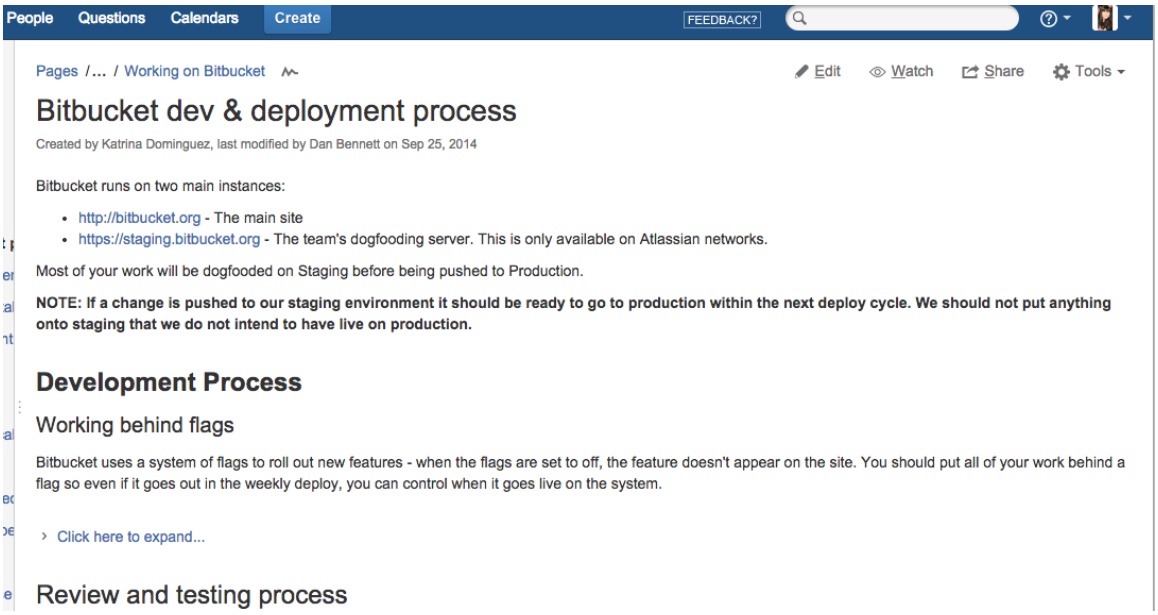

Figure 5.2: Screenshot from the BitBucket team's Confluence page, where they describe how they use a micro-services architecture through the use of flags to make feature teams independent.

dent services that can be deployed independently, thus de-coupling the corresponding teams handling them. This practice reinforces the teams' ability to handle its entire workflow without relying on other teams. The Bitbucket team describe their development process on their Confluence space (see screenshot above); the team uses branches to separate work in progress from work that is ready to be released in the next cycle. They use flags to control which new features go on the site when — unless a feature flag is set to "on" it will not go live on the site even if it is part of the deployment. This practice provides additional decoupling within the product team — individual developers and feature teams are now decoupled from others.

Practices that establish conventions

Given the autonomy and independence of developers and teams, Atlassian teams adopt practices to ensure that work and progress is visible, even if someone is part of a different development team or other part of the organization. Development teams follow the convention of ***two reviews for each pull request*** to verify that the quality of the work is such that it can be merged with the rest of the codebase. On the organizational level and across teams, ***development teams follow a pull-***

based workflow, which proves useful when one team needs a change or fix in another team's code. A member of the Purchasing team described how he was able to extend BitBucket's functionality without deferring to the BitBucket team:

> *'I had an issue with the Bitbucket site and I wanted to filter pull requests that were just by my team. Bitbucket provides a view that is a team filter but it shows all the teams that I am affiliated with. I added a feature to be able to see only my team."* [P11 - developer]

Actions like the one described above mean that individuals or teams don't need to request a change from another team and wait to have it prioritized, but can instead solve problem themselves and submit the change. However, teams complement this flexibility to make changes with ***practicing code peer review***, making progress easy to follow and ensuring quality.

5.5.2 How Atlassian tries to make autonomy work at scale

Information transparency is critical for autonomy, to provide context to decision making [92]. ***Access to all code repositories is open*** to all employees inside Atlassian. The same applies to all information in the organization — the only exceptions are passwords, and specific compensation details for each employee. The benefit from open access to sources of information is twofold: teams do not need to ask for information from each other (avoiding potentially facing delays in replying), and can remain aware of dependencies between their work and others'.

Atlassian uses a variety of synchronization mechanisms to support autonomous entities be aware of each other — especially as there is gradually more of them. These synchronizations mechanisms — featured in the practices I describe below — are used to overcome the challenge relating to overlooking dependencies. They seem to serve three purposes: achieve clarity of purpose, be aware of status, and teams learning from each other. In addition to the synchronization mechanisms there are also guidelines that are meant to ensure some degree of commonality in the end result, targeted mostly at design and coding styles.

Clarity of purpose through involvement in product decisions

Atlassian encourages its employees to follow product decisions and participate in the discussion around them, making the purpose of their work clear. Product teams carry

out customer interviews and off-site visits that are targeted toward getting feedback and understanding customer needs. The ***notes from the interviews and off-site visits are visible***; posted on Confluence, they usually attract a lot of comments from teams that are directly or indirectly catering to a relevant user type. This process frames the "who are we building this for" rationale clearly for product teams, used when deciding and planning their work. As one of the developers explained:

> *"I know who we are building this for, I'm one of them. And even if I wasn't there is enough clarity around here and in people's heads about it."* [P5 - developer]

High-level decisions and analysis of market conditions that explain the underlying rationale are also open to everyone, as the Head of Ecosystem revealed:

> *"One thing I opened lately is a* SWOT *[Strength Weaknesses Opportunities Threats, for marketing] analysis. We open everything to get ideas and questions."* [P2 - Head of Ecosystem]

Decisions on key product direction are transparent and employees can ***provide feedback on them***. Initially, the rationale and description of the direction are posted as blogs on Confluence by the co-founders, and broadcasted to the organization. The approach ensures that the rationale behind decisions is transparent, and opens opportunities for anyone to ask questions, provide comments, and provide constructive criticism. On the face of it the two practices sound the same; they are not. Having transparency of information is not the same as being able (and encouraged) to act on it and affect it. This is not to say that all the key product decisions at Atlassian are crowdsourced. The organization is taking concrete steps in exposing its business rationale and decisions. The exposure not only helps as a platform to provide feedback and answer questions, but also motivates people. The Head of Engineering for Developer Tools explained:

> *"If we are inviting people to contribute to the company's growth, motivation has to be intrinsic such that they believe the vision is correct and what they will do is the right approach, and the thing they leave behind is not going to die."* [P10 - Head of Engineering for Developer Tools]

A program manager echoed this:

> *"People are quite good at jumping on board with the company goals, understand why that happened and it's only possible because it's so open about why something is changing and what they want to focus on."* [P12 - Program Manager]

The motivational aspect of the clarity of purpose is seen as more important than how to coordinate the work. When asked how difficult it is to coordinate work, especially if it involves decisions that involve more than one product team, the Head of Ecosystem commented:

> *"Coordination follows after you get buy-in. The biggest problem is ensuring that everybody believes that what you're doing is the right thing to do. If you settle on that, coordination follows and we have mechanisms and tools for that, all our plans, documents, JIRA, Confluence etc. Also everyone is included and they have valuable feedback for us. So, here it happens a lot more organically because of that, people will have ideas, comments etc. People are also open with how things are going to impact them and how they are affected and they will openly say if there is something they cannot take on because it doesn't work with their plans."* [P2 - Head of Ecosystem]

Atlassian doesn't only rely on teams and their members to follow the available information. There are ***organizational roles*** in place with the responsibility of explaining the strategic intent that comes as input from the executive levels. Speaking about his leadership role, the Head of Ecosystem explained:

> *"I work with the executives on strategy, it's a process of creating a list of fifty things that you could do and decide on five things you need to do and their priority. Then I work with product managers to refine the strategy so that they explain it to the developers. Once you have that you can build roadmaps, and from the development teams' perspective this is where the work starts with their product manager. My job is to set the strategy for them and when it reaches roadmaps I go back to the innovation cycle."* [P2 - Head of Ecosystem]

As mentioned earlier, the product manager will translate the strategy into a roadmap together with the team, and then the development manager and team leads

will refine the selected features into tasks, which will be picked up by developers to work on. This cascade and translation of strategic intent to tactics and then implementation is done through series of meetings — I observed such a meeting of the Development Tools development management team, focused on strategic decisions for Hipchat and discussion on how they will work with the scope and goals of the development tools teams. At the same time, the notes from these meetings, as well as possible blog posts that discuss these ideas are visible (and can be commented on) on Confluence.

Monitoring dependencies through visible status

A cross-product perspective is a reality with two meanings at Atlassian. One, as products become integrated into a tool suite the respective product teams need to keep the integrations in mind. Two, as Atlassian grows and more people, teams, and products are added there is need to ensure that everyone shares the same mindset and have the same organizational points of reference. The practices that support synchronization for this purpose build on transparency.

During the implementation phase of product development ***the progress and status of any one team is visible*** to the rest. This theme applies across the organization to individuals, teams, levels of hierarchy etc. Progress and status are visible on teams' JIRA boards, their repositories, and their plans and documents on Confluence. This is especially helpful in facilitating cross-product work.

Beyond the progress of a team, however, its ***mission and direction are also transparent***. Following on Confluence what other teams aim to do in the next few weeks or months and how they make progress brings awareness to other teams. It also wards against duplicated effort, and allows teams to make decisions about their work by looking at the work others are producing. Teams keep up with each other's progress through viewing documentation and repositories, even if they do not explicitly communicate or meet for that reason. That said, there are regular check in points, for example ***teams give regular demos of their progress***. The feature teams will do a weekly demo for their product team, while once a month there is a similar demo meeting with teams that there are integrations with; for example, BitBucket has a monthly demo meeting with the Stash (now called BitBucket Server) product team.

Teams also have a chance to share their progress with everyone else in the all-staff

meeting. While I was visiting the San Francisco office, the all-staff meeting included a presentation about the Playbook, a project that the R&D division had been working on for the last few months, aimed to provide suggestions and good patterns for healthy teams. As the initiative had reached a level of maturity that made it presentable to others, it was demonstrated and explained to all the staff.

Teams learning from each other

Another synchronization mechanism is the interaction and sharing of experiences between teams to learn from each other. With autonomous decisions on how to organize work, especially as the company increases in size, it is easy for teams to miss how others use practices or tools, possibly missing pockets of excellence amidst the large number of teams and employees [196]. The organization wants to quickly assess the fit of the practices it is using, and exchanging lessons learned from different approaches acts as a crowdsourced way to do so.

At Atlassian, there are both ***scheduled and impromptu opportunities for synchronization between teams***. The word "synchronization" was used in the interviews and documentation; it means that teams have a shared understanding, or that they have coordinated. In practice, this is essentially any process through which teams that don't work together – or have dependencies – learn from each other. These range from regular meetings across the teams in Development tools, for example, to chance encounters over lunch. Teams or individual members often organize informal events, like brown bag sessions [6], which act as demos or workshops. They are a chance to hear more about what is produced by another team, or tools and practices that they have been experimenting with. These events are announced on Confluence and are open to anyone who wants to attend. Events like informal meetings, talks, workshops etc. are additional forums for teams to ***share their successes and failures*** besides blogging about them on Confluence. In the same vein, all notes from retrospectives are visible, editable and can receive comments on Confluence.

The synchronization points and the sharing of experiences have the effect of supporting ***cross-pollination in tools and processes between teams***, a practice that enables consistency across teams to happen in an organic way. As many interviewees mentioned, the teams' choices of process and environments are not enforced; that gives ground for experimenting and having variety.

[6] https://www.atlassian.com/devops/culture#!knowledge-sharing

> *"A lot depends on the team, Bitbucket has their own process, as low as they use Python, not Java. That's OK as long as there is a minimum set of information out of that workflow and minimum communication tools used that we can get information pushed up the stack at the same level of visibility. Takes a bit more work than a prescribed workflow but allows teams to work more organically. No recipe but guidelines."* [P12 - Program Manager]

Using their opportunities to share their experiences and knowledge they gained, teams promote or debunk beliefs and practices. For example, developers from teams using Kanban and Scrum would exchange views and experiment with something that they heard worked for another team.

Guidelines toward commonality

To frame the product development thinking, product teams work with personas (a concept from user-centric design [129], using fictional characters as proxies for users). Atlassian has been open about the way they develop and use personas [7]. In the day-to-day life around the office, the personas are printed and posted in various places throughout the office, including common areas like the kitchen and lounging areas. These ***ubiquitous personas*** act as a reminder of who the organization is building its products for and, therefore, the customer needs that the developers, designers etc. need to keep in mind when making decisions.

The development teams also use ***templates and design and coding guidelines***; these are meant as mechanisms to ensure that output is consistent in structure and appearance even if it is created by different developers and teams. As mentioned earlier, there are templates on Confluence for frequently used information artifacts like meeting minutes, retrospective notes, feature descriptions etc. One oversight that was mentioned during the interviews is that there are no naming conventions in place, making it hard to search for them. Development teams keep coding guidelines on their Confluence space, as a way to provide information for newcomers and members of other teams that want to contribute. Design guidelines are used to promote a consistent appearance and rationale behind product design. Some of the design guidelines are publicly available [8], since there may be developers or existing customers that

[7] `https://design.atlassian.com/how-we-design/personas/`
[8] https://design.atlassian.com/

want to develop add-ons for Atlassian tools and offer them through the Atlassian Marketplace.

5.6 Further challenges

Even with the provisions made by Atlassian's work practices, challenges still exist. In the interviews additional challenges were mentioned that correspond to exposed tradeoffs or limitations of current practices. I provide a summary below, while the detailed description and relevant quotes are provided in Appendix B.

The **information transparency at scale** that Atlassian follows creates issues of information overload. All information and work artifacts are by default visible across the organization and archived in Confluence. While this is great for visibility, the interviews identified information overload as a side-effect. The volume of information that is produced during the course of a single day is overwhelming and potentially distracting. Even with mitigation strategies like following certain types of information, participants commented that they still end up receiving a large number of notifications. At the same time — as it was repeatedly noted in interviews and the satisfaction surveys — it is difficult to track and navigate information on Confluence. Confluence's search function is not optimal and, given the amount of incoming information, makes it hard to find information again. The lack of uniform rules for archiving and curating documents and other communication artifacts across the organization also makes retrieval challenging. Better categorization and information tagging, with guides of recommended document naming conventions across the organization, could be solutions for easier navigation, until the search functionality is improved. The rapidly growing number of Atlassian employees that participate on Confluence is only expected to accentuate this challenge.

Another challenging aspect of information transparency in the large is balancing the growing number of voices and the quality of discussions. One, there is always the potential of heated debate (exacerbated by geographical distribution), which may create negative experiences and/or uncertainty about how to contribute ideas and information. Two, there is a question of boundaries around how long items should be open for discussion before decisions are made, or the weight that each voice carries. There seems to be room for introducing some desired patterns for communication to provide guidance around healthy debate and how long discussions should go on for. OSS projects use a "Code of Conduct" as a set of guidelines for smooth communica-

tion and interaction in their large and diverse communities; this practice could work in Atlasssian's case too, instilling conversation etiquette as part of the on-boarding experience for newcomers too. Finally, Atlassian is home to different generations of information users that interpret information sharing and communication etiquette differently. Setting examples and educating users on what is considered mindful and constructive language, content and level of participation is a gateway to ensuring smooth and effective communication, especially as Atlassian is considering opening up some of its communication channels to their partners and customers in the future.

Interviewees also highlighted the **continued state of flux** in the organization as a challenge.

On the one hand, Atlassian grows at a fast pace and new employees are on-boarded to the organization constantly. The organization subsequently changes its structure to accommodate and integrate them into its routine. On-boarding is an area that Atlassian pays a lot of attention to, and the organization is always looking for ways to spread the company culture to a constantly new crowd. A practice Atlassian has followed is to shuffle senior developers around so that they can help and guide new employees and/or new teams, and in that way indoctrinate the culture that is so important to the organization. On the other hand, this very practice of employees moving around results in frequent changes in roles and teams. Such shifts are mostly ad hoc and allow the organization to be flexible to adjust to market opportunities and challenges, and in a way its agile mentality. A challenge that was mentioned along side the pursuit of flexibility, however, was that that teams do not always have the stability to be as efficient as they could, or that they find it difficult to establish or maintain a team identity. Finally, participants mentioned that movement of people between teams may make ownership and responsibility for code or decisions unclear. One side-effect that was mentioned was that questions may be directed to the wrong person, who may have now moved to a different role. While people volunteer to still answer questions and direct people to the right source of information, it can prove time-consuming and a distraction. This challenge brought up the need to curate information, especially when it relates to ownership and responsibility.

5.7 Discussion

After reviewing the work practices used at Atlassian to handle agility and growth, we can see that the organization uses autonomy as one of its scaling techniques.

Autonomy in Atlassian's context was associated with the freedom to make decisions on how to organize one's work, without the need to coordinate with others or require approval, to avoid delays. However, autonomy was referenced alongside the need for a shared understanding of organizational objectives; in this section I discuss this combination and how the work practices at Atlassian fit with it. I first introduce additional literature to introduce the concept of Aligned Autonomy, originating in the organizational leadership literature. I subsequently review Atlassian's work practices as to how they fit with Aligned Autonomy; I first present my own interpretation and then the validation I received from Atlassian. Finally, I discuss how autonomy seems to be conceptualized differently when applied at scale, and the role of culture in making it effective.

5.7.1 The concepts of Directed Opportunism and Aligned Autonomy

Directed Opportunism appears in the book *The Art of Action* by Stephen Bungay [28]. The book posits that organizations fail in execution due to imperfect information and uncertainty that create organizational friction and gaps between plans, actions, and outcomes. Bungay [28] sees the problem originating in the nature of organizations — they are made up of multiple individuals but pursuing a collective goal. For Atlassian, the concern of potential organizational friction becomes relevant when considering how to scale its agility; the organization wants to maintain autonomy in its agile teams, but not to put its collective goals at risk.

What is a possible solution? Bungay [28] discusses that often companies that want to align the individual and the organization choose to enforce more control by providing detailed instructions. Bungay's conclusion is that specifying too much detail creates confusion when a situation demands something other than what instructions say, and as a result the adjustment to emergent conditions is slow. The proposed solution, modelled after an approach followed by Prussian Army military strategists in the nineteenth century to coordinate large battles, comes in the form of *directed opportunism*.

Directed opportunism proposes that alignment can be achieved by limiting the direction to defining and communicating the intent (i.e. what is the goal to be accomplished), and otherwise giving individuals freedom to adjust their actions in line with the intent. Bungay describes "backbriefing" as a technique of allowing each level

of an organizational hierarchy to define how they will achieve the intent of the next level up, and inform them. What is important is that the intent is incorporated in the strategy of the organization and, rather than being a plan, it is a framework of decision making. As Bungay puts it,

> *"[Strategy] is an original choice about direction, which enables subsequent choices about action."* (p. 97-98)

Organizational theories have a similar concept called *minimum critical specification*; senior management needs to define *only* the critical factors that are needed to direct the team and place as few restrictions on the team as possible (I discussed this, and related literature in Chapter 2). According to Bungay, instructions should contain all that people cannot determine for themselves to achieve a particular purpose, but only that. As long as the higher intention is made clear, the individual initiative can be relied on to adjust actions according to the situation. In that sense, Bungay concludes in one of the most striking quotes from his book,

> *"The more alignment you have, the more autonomy you can grant."* (p.65)

Bungay sees Alignment and Autonomy as two dimensions instead of ends of a spectrum. The combination of high levels of Alignment and high levels of Autonomy — a state of *Aligned Autonomy* — creates an environment where individuals and teams have both clarity about the intent and freedom about how to pursue it. While the principles behind this state are clear enough, they create implications for the organizational structure, the communication structure, and the culture of working. For example, in order to achieve high levels of Autonomy teams need to be de-coupled as much as possible to decentralize their actions and work in parallel. To achieve high levels of Alignment, transparency of setting goals and communicating them is key, creating a particular cultural environment of visibility, awareness, and shared goals in the organization.

It is important to note that the concept of *Aligned Autonomy* has attracted some attention from practitioners [9]. The principles behind it are followed by a few organizations that go through the same transformation as Atlassian; moving from a small startup to a mature startup to an enterprise, and deal with issues of scaling their agile practices. Although the discussions about the principles and the concept of *Aligned*

[9] `https://hbr.org/2017/02/how-spotify-balances-employee-autonomy-and-accountability`

Autonomy have mostly taken place in practitioner blogs [10], it is interesting to note the impact.

Below I discuss how the concepts are applied — not consciously — at Atlassian, and how the discovered practices fit with them.

5.7.2 Aligned Autonomy through Atlassian's work practices

Overall Directed Opportunism seems to be a model that is followed at Atlassian, even if it is not called that. I presented how the co-founders and other executives provide direction by clearly communicating the intent in their frequent communication with the entire company. I also described practices that provide freedom to teams to adjust their action in line with the intent. This is also true inside the individual teams, where developers have the freedom to decide and adjust their own action in line with the intent of the team level, reflected in the roadmap and backlog. Finally, I described how through various communication mechanisms both the intent and the achieved results are communicated and enable independent action. These seem to me to fulfill all the criteria of Directed Opportunism, as Bungay describes them in his book [28].

From Bungay's analysis it seems that if work practices provide Autonomy and Alignment (ideally high levels of those), an organization can follow the model of Directed Opportunism, with operations happening in parallel and independently without losing focus relating to the goals to be achieved. Atlassian is a high-functioning organization and seems to be navigating the challenging world of scaling agility successfully through its practices; analyzing how the practices aid Autonomy or support Alignment can show if and how the concepts are relevant in the particular setting, especially since the concept hasn't been used in the field of software engineering research.

In re-examining the recorded work practices I have attempted to categorize them as to how they support Autonomy and Alignment. In doing so, several necessary considerations surfaced.

- First, the Autonomy and Alignment concepts need definitions. For the purposes of this analysis, the working definition of Autonomy is "the freedom that individuals or teams have in doing their work without the need to coordinate with someone else". The working definition for Alignment is "the clarity that

[10] `http://blog.idonethis.com/cells-pods-squads-structure/`

individuals and teams have in knowing what the strategic goal is and how their work fits in that without the need to consult with someone else".

- Second, Autonomy and Alignment are distinct concepts but it is their combination that is the most relevant; that means that each practice needs to be discussed both in terms of Autonomy and Alignment, even if its primary purpose is to support the one rather than the other.

- Third, it is necessary to introduce *medium* levels for Autonomy and Alignment, as the low and hight ends of the spectrum may not be appropriate to capture work practices that attempt to combine both aspects but involve tradeoffs.

- Fourth, following from the consideration above, I have considered a practice to support a **high** level of Autonomy when it creates conditions that make an individual or team independent of others; **medium** level of Autonomy where the practice creates a need to coordinate with someone else; and **low** level of Autonomy when a practice makes an individual or a team dependent on someone else. Regarding Alignment, a **high** level corresponds to conditions that help define how an individual's or team's work and others' fit in with strategic decisions about product direction; **medium** level means that a practice creates the need for an individual or team to consult with someone else; **low** level of Alignment reflects practices that block an individual or a team from knowing how its work and others' work fit in with strategic decisions about products.

Table 5.3 includes all recorded practices that Atlassian uses to address the challenge of maintaining efficiency at scale. Table 5.4 includes all recorded practices that Atlassian uses to address the challenge of scaling autonomy. In each table the practices are grouped by purpose, and I have used icons to indicate the level that they are applied in; individual ( ), team ( ), organization ( ), or across-teams ( ). Finally I have related each practice to a level of autonomy and alignment; I provide a visual indicator (using for high, and for medium), and a short explanation for each entry.

I have summarized the information in table form to avoid rehashing the discussion of practices. To demonstrate, I provide two examples below, but the full explanation of how I related each practice to levels of autonomy and alignment is provided in Appendix B.

Illustrating examples

The practice that *submitted pull requests require two reviewers* creates a need to coordinate for the review and therefore results in a medium level of Autonomy for the team. It is also associated with a medium level of Alignment as developers need to consult with reviewers about how well their implementation fits with the goals the team is trying to accomplish; the goal has been set when creating the backlog, and has been products of the original high-level strategy.

The practice of *the team deciding the backlog* provides a high level of Autonomy for the team but is essentially bounded and informed by the intent and strategy communicated through meetings, blog posts and all-staff events. To that end, the practice supports a medium level of Alignment as it creates a need to consider these decisions when deciding backlog items.

Validation of interpretation

The analysis I presented above is somewhat subjective, although I provide the full explanation in Appendix B as an audit trail for the reader to follow my reasoning. As part of wrapping up the project with Atlassian, I had an exit interview with the program manager that had been overseeing the project. Although the program manager was aware of the goals for the project and the nature of the enquiry, I had not had an interview with him about the work practices followed at Atlassian. Before the exit interview I submitted a list of the practices that appear in Tables 5.3 and 5.4 to the program manager and requested that he:

- verifies, to the best of his knowledge, that these practices are followed at Atlassian
- assigns a level of Autonomy that he associates with each practice (the definition of the levels is the one I presented in 5.7.2)
- assigns a level of Alignment that he associates with each practice (the definition of the levels is the one I presented in 5.7.2)
- provides any additional information or clarification regarding the above.

One thing that the program manager highlighted was that due to the cross-pollination between Atlassian's teams some of the practices may vary between development teams. For example, while for most teams the practice is to assign two

Table 5.3: Practices to address the challenge of maintaining efficiency at scale, grouped by purpose and related to levels. Each practice is assigned levels of autonomy and alignment with a corresponding explanation.

Challenge: Maintain efficiency while growing						
Purpose	**Recorded practice**	**Level of practice**	**Autonomy**	**Explanation**	**Alignment**	**Explanation**
Provide autonomy	Developers select the task they are going to work on each sprint			Developers have freedom to select their tasks		Clarity about how each task fits with others (decisions made when building the backlog)
	Developers make changes on their branch			Isolation between different developers		Need to consult with others in code review
	Development teams define their own scope and timeline			Freedom from the need to coordinate with other teams		Need to keep in mind the strategic intent
	The backlog is decided by the development team			Team has freedom to decide features and tasks to work on		The backlog is informed by strategic decisions
Ensure independence	Developers use branches for their work on a task or feature			Isolates development that relates to different tasks		Developers can see how their work fits with the product (pre-decided when building the backlog)
	Developers submit pull requests when they have changes ready			Developers need to coordinate with team members for code review		Need for consultation about the fit between delivered feature and intended functionality (in code review)
	End-to-end ownership of a team's workflow			Independence from others for work completion		Clearly defined process for testing, building, and deployment
	Deployment to production is handled by the development team			Independence from others for work completion		Clearly defined process for testing, building, and deployment
	Builds are self-managed by teams			Independence from others for work completion		Clearly defined process for testing, building, and deployment
	Teams include engineers, designers, devops, and support engineers			Teams are self-sufficient in skills needed to complete work		Developers need to consult with the other roles in the team
	Micro-services architecture			Teams work on decentralized parts of the product		Architecture defines how the separate parts work together
Establish convention	Submitted pull requests need two reviewers			Some need for coordination to perform code review		Need to consult with code reviewers
	Teams follow a pull-based workflow			Developers can contribute fixes to other teams without the need for permission		Developers need to keep in mind the goal and objectives of the other team
	Teams practice code peer review			Developers need to coordinate review with other teams		Clarity about how another team's work fits with strategic intent

Table 5.4: Practices to address the challenge of scaling autonomy, grouped by purpose and related to levels. Each practice is assigned levels of autonomy and alignment with a corresponding explanation.

Challenge: Autonomy at scale						
Purpose	**Recorded practice**	**Level of practice**	**Autonomy**	**Explanation**	**Alignment**	**Explanation**
Clarity of purpose	Offsite notes are visible			Artifacts enable teams to act independently		Artifacts provide point of reference
	Decisions on key products directions are transparent			Some need for coordination to provide feedback		Establish shared understanding
	There are ways to provide feedback on decisions on products			Some need for coordination to provide feedback		Establish shared understanding
	There are organizational roles to communicate product directions to teams			Some need for coordination to arrange communication		Establish shared understanding
Monitoring dependencies	Access to repositories is open (internally in the organization)			No need to ask for permission		Access to information to understand strategic intent
	Progress and status of all teams are visible			Allows teams to work independently		All parties aware of how work fits with strategy
	Other teams' mission and progress are transparent			Allows teams to work independently		All parties aware of how work fits with strategy
	Teams give regular demos of their work in progress			Occasional need for coordination to arrange events		Transparency of actions toward objectives
Teams learning from each other	There are regular synchronization opportunities between teams			Occasional need for coordination to arrange events		Transparency of actions toward objectives
	There are impromptu synchronization opportunities between teams			Occasional need for coordination to arrange events		Transparency of actions toward objectives
	Teams share their successes and failures with other teams			Occasional need for coordination to arrange events		Transparency of actions toward objectives
	Employees move around different teams and learn first hand how they work			Need to coordinate to have people moved from other teams		Need to consult with existing members to understand how team's work fits with strategy
	Employees rotate roles and learn what are the responsibilities and pain points			Need to coordinate rotation		Need to consult with other members to understand how the role fits with strategy
	There is cross-pollination in tools and processes between teams			Occasional need for coordination to arrange events		Transparency of actions toward objectives
Guidelines toward commonality	Ubiquitous personas			Artifacts enable teams to act independently		Artifacts provide point of reference
	There are templates, and design and coding guidelines available			Artifacts enable teams to act independently		Artifacts provide point of reference

reviewers per pull request, other teams may assign fewer or more reviewers, or give provisional approval for pull requests. The program manager verified 27 of the 30 practices as followed by Atlassian's teams (3 practices were not verified because they may vary with teams). The differences that were noted between my interpreted categorization and program manager's external audit are shown in Table 5.5 below.

Reported practice	Autonomy/Alignment interpreted	Autonomy/Alignment validated	Explanation of disparity
The backlog is decided by the team			Medium autonomy because tasks need to be aligned with the product/company strategy
Micro-services architecture			Medium levels because of need to coordinate the transition to the architecture
There are ways to provide feedback on decisions on key product dimensions			Medium alignment because of the need to review and discuss the decisions

Table 5.5: Practices that were only partially validated during external audit

During the exit interview the program manager mentioned that the concepts of Autonomy and Alignment were not easy to relate to, and we proceeded to go through the practices so that he had a chance to explain and/or amend his answers. There were no changes made to the answers that the program manager gave, but he elaborated on his answers and the rationale of his responses. The small difficulty faced by the program manager in relating to the concepts right away, speaks to how the concept of Aligned Autonomy is not followed consciously at Atlassian and it is not the result of articulating the purpose to lead the organization via Directed Opportunism. Nevertheless, the empirical approach that Atlassian has taken in structuring its teams, giving them freedom, communicating strategy to them, and addressing the scaling challenges of agile practices, seems to agree with the concepts, evident by the categorization of the practices presented above. As a result, there is an indication that the concept of Directed Opportunism and the model of Aligned Autonomy are useful frameworks to guide the scaling efforts of agile practices in software companies. The practices that I have presented and the rationale behind them can act as useful guidelines to organizations that are facing similar challenges. While this is not an extensive and comprehensive validation of the categorization and other people could offer other views, the overlap between the researcher's interpretation and the practitioner's validation is considerable.

5.7.3 The interplay of autonomy and scale: Agility, growth, and cultural values

Current software engineering research recognizes that growing and large software organizations feel the pressure to quickly adjust to swift market changes [177], and that development speed needs to remain high to keep up [179]. To accommodate both speed and frequent change most modern software companies follow agile practices in their development [101]. Recent trends in software engineering call for scaling the agile concept to the enterprise level [112], however, offering advice on best practices to do so [126]. Following agile principles under conditions of fast growth is an identified challenge because companies are looking for speed *and* good inter-team communication that keeps everyone on the same page. In the view of major companies in the software industry "where most companies fail is doing both at the same time without hindering the development cycles" ([177], p. 984). It is this identified need, and potential failing point for many software companies, that requires a systematic approach.

The Atlassian study has interesting findings for both aspects of handling growth: how to scale agile practices to facilitate speed, and how to implement an agile mindset at an enterprise level.

One interesting point to discuss out of the Atlassian study is how to use agile practices to address the potential delays brought on by the increased number of people that participate in operations. Agile teams are largely self-organizing and have some freedom to make decisions and organize their work. The self-organization built into agility is one way Atlassian used to address scaling issues; by having teams self-organize they only ever need to deal with their own small scale than the collective large one. In this study (influenced from the findings described in Chapter 4) self-organization served as the conceptualization of autonomy. The Atlassian study showed that agility at scale becomes an issue of handling autonomy at scale.

Autonomy in Atlassian's case was discussed alongside alignment. We saw that the organization uses several scaling strategies that build on autonomy but there is also emphasis on articulated guidelines and synchronization opportunities between the autonomous entities. At the same time, Atlassian's organizational culture forms another mechanism of alignment. As internally public roadmaps, transparent decision making, and synchronization meetings help communicate the intent for the strategic outcomes, Atlassian's values play their part in articulating the intent for or-

ganizational behaviour. The organization's five official values describe employees that appreciate and practice transparency, team spirit and responsible autonomy; they are the guidelines of good organizational citizenship and set the criteria based on which to make autonomous decisions. While these are the values Atlassian specifically aspires to, on an abstract level they can transfer to other organizations too. Transparency, shared ownership, clarity of purpose, and aligned autonomy, were key concepts in the Atlassian case study, and play a part in making the scaling strategies work.

The interplay between work practices and values is important to keep in mind, not only for the existing Atlassians, but also to form hiring criteria. Hiring was described by executives as a key component of the organization's successful operation, where the focus is on cultural fit more so than on technical excellence. As was described by the interviewees, success in hiring people with good cultural fit makes it easier for the alignment mechanisms to work because individuals and teams are already in line with the mindset of the organization. Any software company looking to learn from Atlassian's example should be taking away the message that establishing what it believes in and stands for through common values is an essential foundation for the work practices or scaling strategies to work.

The Atlassian case also highlighted that agile work practices can scale across all functions of the organization, not restricted to software development. This concept goes as far back as Takeuchi and Nonaka's [198] discussion about agile methods in the product development research area. They advocated self-organizing project teams and subtle control (concepts that strongly resemble autonomy and alignment) and that agile methods should be systemwide, covering the organization's strategy, its product, and its development function. More recently this need for integration between business strategy and development has been described as *BizDev* [74], within the concept of Continuous software engineering [75].

Continuous software engineering (or Continuous Star [75]) is a holistic view of software development, as placed within and connected to the organizational environment. It pulls together several trends in software engineering aimed toward continuous delivery, assessment, and improvement. One of the key aspects of the concept of Continuous software engineering is the idea of *BizDev* — the close linking of business and software development in organizations — as a potential remedy to the usual disconnect between these functions.

The work practices at Atlassian seem to be congruent with the *BizDev* idea. Atlassian used organizational vision and strategy as an aligning factor, facilitated by the

culture of transparency in strategic decisions and public (within the organization) discussion about them with all employees. This approach created a shared understanding between employees that they carried over when making their autonomous decisions. As an example, the templates used for documenting technical specifications or feature descriptions included a field to explain how the particular item relates to the VTFM. In that sense the practices I described throughout the Atlassian study show approaches to implementing *BizDev* and could address long-recognized gaps and mismatches between development and other functions in the organization, such as marketing, legal, support etc.

5.8 Threats to validity in the study

It is possible that since one of the foci of the study was culture, participants in the interviews tried to paint a more favourable picture that what happens in the real day-to-day life in the organization. I mitigated the risk by using multiple sources of information and reviewing documentation for a year. In both my observations and in the documentation there were many indications that the culture and openness that was described were a real and significant part of the organizational life. What is more, all the members of the organization that I interviewed and interacted with during the project were also very open about their challenges and concerns. As such, I have confidence that the findings represent the reality of the organization.

While I spent two weeks on site — which could be considered a short time — I interviewed and observed a diverse set of roles in the San Francisco office. However, I did not visit offices in other locations and it is, therefore, likely that the findings are not representative of the entire organization. Yet, two of the interviewees had experience with the Sydney office; one of them was a visiting program manager from headquarters, and the other one was an employee on secondments for the past year. Their interviews showed that the practices that we talked about were followed in both locations. To further address the risk, I relied on triangulation whenever possible, combining evidence from interviews, observations, documentation, and the satisfaction survey. Finally, I conducted an interview that served as external audit with a program director in Sydney who was overseeing the project. As preparation for the interview I asked the program manager to fill out a survey and review the list of practices I compiled; he confirmed that all practices are used at Atlassian as a whole.

Regarding external validity, the findings and their analysis come from one organi-

zation, which poses questions about representativeness. Yet, the goal in the study was to identify contextual factors such as strategy and culture. The cultural concepts of Aligned Autonomy have been described by practitioners in other companies through company blogs, and there is overlap with the study findings.

5.9 Conclusion

The question driving this chapter had to do with *the potential risks of scaling autonomy and the practices that software teams and organizations use to address them.* The study I presented explored the application of agile development and work practices under conditions of rapid growth, and across an entire organization.

The study shows that a model of *collaboration via aligned autonomy* at the scale of the organization comes with certain challenges, and I reviewed a number of practices used to address them. Atlassian's overarching approach was to cultivate a culture of Aligned Autonomy to address the challenges; built on transparency, open communication, and involvement. With such culture, the teams and individuals share a common rationale and purpose that guides their autonomous actions.

While autonomy in this chapter was conceptualized as self-organization to include the freedom in organizing work, one thing that surfaced was that the freedom also comes with *responsibility* of alignment with organizational strategy. In that sense, autonomy seems to carry a meaning of self-management or empowerment; individuals and teams are given the freedom and are empowered to take ownership of what they decide to do. For the autonomous decisions to be effective, individuals and teams need to have clarity of purpose and knowledge of the strategic intent. In Atlassian's case the rationale and purpose were broadcasted and disseminated by blogs and open communication. On top of that, intent was communicated by managers acting as messengers between groups and hierarchical levels, although I did not explore this aspect in the study, which can be considered another limitation. I made managers the focus of the next case study I conducted at Microsoft, and later reflected on how the management insights from Microsoft related to the observations and lessons learned at Atlassian.

The role of the engineering manager is understated in software engineering, and the function of the manager as messenger and translator of strategic intent is not readily what we imagine they do. I explore the attributes and role of the engineering manager in depth in the next chapter, where I describe a study that shows what role

managers play in environments that value autonomy, and developers make some of the decisions that managers used to make.

Chapter 6

Autonomy as empowerment — The Microsoft study

"Any truth is better than indefinite doubt".
Sherlock Holmes – The Yellow Face

This chapter discusses the attributes that are perceived to make great software engineering managers in an environment that values autonomy. The discussion is based on an empirical study of developers and managers at Microsoft and how they view the role of the software engineering manager.

Case studies from diverse industries — reported both in consultancy reports [219] and academic articles [87] — show that great managers make a significant difference in the performance of teams and organizations [210]. Conversely, the wrong person in a manager role has detrimental effects on employee engagement, productivity, and the quality of produced results [14]. As software development today is done in teams, for all but the most trivial programs, managers are essential to organize the collaborative effort of creating good software and support the people that carry it out. Yet, despite the potential impact, the software engineering literature offers few papers on people management in software engineering teams.

As we saw in Chapter 2, often in highly hierarchical organizations managers make decisions on the nature and priority of work, and assign it to workers. Both theory and practice have looked outside that model however, and nowadays organizations are looking for ways to empower and inspire their knowledge workers – we saw evidence of that in the case described in Chapter 5. In such empowering environments, where employees (in our case, developers in particular) can make decisions on the na-

ture, priority and assignment of work, what is the manager's role? More specifically, *what are the attributes of great management* in an environment of empowerment and autonomy? I address this question in this chapter and the study I describe. The evidence I offer comes from an in-depth investigation of engineering management at Microsoft – a large, established, and successful software company.

6.1 Why study Microsoft

Three reasons led to the selection of Microsoft as a case study setting. First, Microsoft is a large software company, with multiple layers in its management structure. As a result, it is possible to gather information about management practices and concerns, and also get multiple perspectives. As I will discuss in the methodology of the case study, I had the opportunity to elicit feedback from developers, team leads, and two management levels, in my investigation of the role and attributes of engineering managers. That the organization is long established also means that there is wealth (and probably diversity) of experience in what has and hasn't worked in engineering management.

Second, Microsoft has gone through a shift in its organizational culture in the last few years. When the current CEO took over in 2014 he talked about a change in Microsoft's culture, one that should be based on a mindset of personal learning and growth [1]. The "growth mindset" is a concept taken from Carol Dweck's book [65]. In the book, the Stanford Professor of Psychology discusses the difference between two set of attitudes someone can hold; the traditional "fixed mindset" or her proposed "growth mindset". Under a fixed mindset, a person believes that talent and abilities are fixed and, therefore, spend their energy trying to avoid failure. On the other hand, under a growth mindset, a person believes that their talents can be developed through hard work and support from others, and therefore failure is seen as a learning opportunity. Based on Dweck's research, individuals with a growth mindset tend to achieve more and put more energy into their development. On the organizational level, whole companies can embrace a growth mindset as an approach to leadership, to empower their employees [2]; Microsoft is one of those companies [3]. That Microsoft embraces such a mindset is signal that it is looking to empower its employees and

[1] http://www.businessinsider.com/microsofts-ceo-email-2014-7

[2] https://hbr.org/2014/11/how-companies-can-profit-from-a-growth-mindset

[3] https://hbr.org/2016/10/how-microsoft-uses-a-growth-mindset-to-develop-leaders

that it enacts autonomy at least to some degree.

Finally, Microsoft is a successful software company. While skeptics might see that as meaning an "elite" organization with no generalizable findings to offer, I see it as a (rare) chance to learn from one of the best software organizations. The way Microsoft approaches engineering management may well be a contributing factor to its success and, as such, of critical importance to understand on both theoretical and practical fronts.

6.2 Case study setting and methodology

If someone wants to understand how to manage software engineers and teams thereof today, they will face two challenges. On the one hand, not much research has focused on managing software engineers. The software engineering literature equates management with managing the software project, not people. Recent software development approaches like *Lean* and *Agile* have nuggets of wisdom for how to manage people, but mostly look for processes that make efficient use of time and skills. We still don't know what to look for in a great engineering manager, and how to further develop their skills to support the software teams they manage.

On the other hand, one cannot – with confidence – simply import general management principles and lessons from other domains. Why?

- *Management gurus suggest that established management principles should be revisited.* In a recent Distinguished Guest Lecture, Harvard Business School visiting professor Gary Hamel noted that management has not kept up with profound global shifts, remaining unchanged in the last 60 years [4]. Other experts also question the applicability and relevance of known management principles [154].

- *Software engineering practitioners approach management with skepticism (and disdain)*; some argue that the creativity and high skill level of software engineers makes managing them a special challenge [172]. Google even found itself in the position of having to remove managers to convince its employees and leadership through data that managers are not only necessary, but impactful [5].

[4] `http://bit.ly/2j5gsy6`
[5] `http://on.inc.com/1s8yu4a`

- There is an underlying assumption that the same management practices would apply to all knowledge workers, but *this assumption has not been investigated or verified.*

There is a need to investigate engineering management and the elements that make it up, especially given shifts and trends that software engineering is experiencing currently, such as decentralization and autonomy. Such a study can shed light on how management practices transfer over to software engineering, put management strategies in context to be helpful in software engineering, and help convince the skeptics – be they engineers, managers, or academics. In the rest of the chapter I describe a study that addresses this need through an investigation specifically of software engineering management at Microsoft – a modern software company – enlisting the views of both managers and engineers.

6.2.1 Research goal and questions

The research goal behind this chapter is to understand how software engineering managers function and what can make them great in an environment that empowers developers and gives them autonomy in their work; given such understanding both theory and practice can capitalize on the positive impact of great managers on motivation and engagement [98].

In the study I am presenting I looked for attributes that *characterize* great software engineering managers, *why* and *how* these attributes are important, and how they are put to use. Although I knew that Microsoft's organizational culture puts emphasis on empowerment, I avoided framing the study particularly about autonomy to the participants. The goal was to not risk biasing the participants' perception of how autonomy is experienced and how important a part it plays in their work – I rather trusted that if autonomy is as central as the organization claims it is in their practice, it will be reflected in the data. Autonomy did organically surface as a critical factor of software engineering management, and alongside it I was able to probe and extract other important parts of the software engineering role.

6.2.2 Data collection and analysis

The project was approved by the University's Ethics Board, the certificate of approval is provided in Appendix C.

The research methodology comprised two high level phases. In the first, exploratory, phase, I interviewed 37 software engineers and engineering managers to identify important attributes of great software engineering managers. In the second, confirmatory, phase, I developed and deployed a survey to a larger population.

Interviews

I used interviews to identify the important attributes that make a great software engineering manager as well as understand why such attributes are critical and how they manifest in software engineering contexts.

Participant selection. I purposely sought to interview a diverse group to capture as many varying opinions and experiences as possible. To that end, I used a *stratified purposeful sampling approach* [164] to recruit interviewees. This selection strategy is a form of *Maximum Variation Sampling* [164] and is appropriate when "the goal is not to build a random and generalizable sample, but rather to try to represent a range of experiences related to what one is studying" (p. 52). To capture multiple perspectives, I interviewed *software engineers* (those being managed), and managers at multiple levels.

Software engineering manager (or simply, *engineering manager*) is the name of a particular role at Microsoft. These managers are responsible for delivering results through one or more teams of engineers; they assist the team with goal setting, handle hiring decisions, manage resources for the team(s), and are responsible for guiding the engineers' professional development and reviewing their performance. As part of communicating business direction to their team(s), engineering managers liaise with other teams and meet with upper management. Before major releases, engineering managers represent their team in cross-team discussions about project status, and decisions on the features that ship to customers.

Although the current study focuses on the engineering manager role, I elicited the perspectives of managers at multiple levels. These included *team leads* (often owning a feature with a small number of engineers reporting to them), *engineering managers*, and *upper level managers* (those that hire, advise, and review engineering managers). Since I found that responses were in alignment across the different management roles, I make no distinction in the remainder of the chapter; I simply divide those interviewed and surveyed into engineers and engineering managers.

For both engineers and managers, I selected participants along the dimensions

Table 6.1: Participants in the role and experience dimensions

	New	**Experienced**
Engineer	P1, P2, P3, P4, P5 (out of 50)	P6, P7, P9, P18, P20, P23, P26 (out of 40)
Team Lead	P27, P30 (out of 10)	P28, P31, P32 (out of 10)
Manager	P11, P12, P14, P17, P19, P21, P22, P25 (out of 40)	P8, P10, P13, P15, P16, P24 (out of 40)
Upper Manager		P29, P33, P34, P35, P36, P37 (out of 20)

of *experience* (new to the role – hired in the last 6 months – or long time in their current role – longer than 5 years), *number of employers* (has their entire career been at Microsoft or have they worked elsewhere), *gender*, *organizational level* (engineer, team lead, engineering manager, and upper level manager), and *product group* (e.g., Windows, Office, Azure).

I sent recruitment emails to a random sample ranging between 10 and 50 people, depending on the size of the stratum. For those that accepted, I sent a follow up email before the interview asking them to rank the top five most important attributes from a list of 16 manager attributes (I refer to those as *seed* attributes in the rest of the chapter). The list of seed attributes was compiled based on the annual company poll where Microsoft employees evaluate their manager (11 attributes) and my review of additional management literature [142, 94, 61, 195, 82] (5 attributes, one from each of the studies); I added attributes found in the literature that were not already included in the company poll or very similar to those. I also provided space for the participants to add other attributes that they felt were important.

Table 6.1 shows the role and experience of the 37 interviewees. In the parentheses I provide the number of participants that I sent invitations to from that stratum.

Interview protocol. All participants were asked – regardless of their level – to refer to and talk about the *engineering manager* role. I asked interviewees basic demographic questions; the information collected was the participants' number of years of professional experience, the number of different companies they worked in, and their current role at Microsoft. Next, we had an in-depth discussion of three of the attributes I selected from their top five seed and write-in attributes. I determined that a discussion of three attributes could pragmatically fit in the time I had with the participants to both collect all the information I needed, and not rush their answers.

This was confirmed as a fitting strategy after the first few interviews. The attributes were selected for discussion based on the ranking given by the participant; I chose the highest ranking attributes. When interviewees had provided write-in attributes that they felt were more important, I gave those priority in our discussion. Gradually, as I achieved saturation regarding some of the attributes, I intentionally picked for discussion attributes that I had less information about as long as the participants had highlighted them as important (i.e. were in the top five).

I asked why they felt each attribute was important for great managers to have. I also asked about strategies to gain or utilize the attribute (for managers) to ensure my understanding of the nature of the attributes, and to offer actionable insights from the study. I intentionally used the abstract term "great" without providing a definition so as not to bias interviewees and instead gain an understanding of what it meant to them. I accomplished this by employing a "War Story" elicitation procedure to explore concrete experiences from the interviewee related to the attribute [134]. I explained that I was interested in experiences they had any time during their development career, not limited to Microsoft, and also asked them not to use names or indicate whether various experiences, thoughts, etc. referred to their current or prior teams or managers. Interviews lasted around an hour and were recorded with the interviewee's permission.

Analysis. The interviews were transcribed; I then identified all attributes brought up during the interviews and performed an open card sort [189] to identify categories, and organized them into themes [143]. Each card represented an attribute that was described in an interview, either seed ones or those that emerged from the participants; 83 cards were sorted into 15 categories. The card sorting was performed with the help of a second researcher (employed at Microsoft), in two rounds. For each category, we examined the context for every card in that category and came up with a name and a short list of examples for the attribute. The interview transcriptions were then coded according to these categories.

Survey

I designed a survey based on interview results to validate, and see how the attributes generalize to a broader population.

Survey instrument. The survey instrument is provided online[6], and in Ap-

[6] `https://github.com/cabird/Engineering_Manager_Study`

pendix C. The survey's primary purpose was to *assess the importance of each of the identified attributes* from the interviews and determine if there were additional ones to add. The survey asked respondents to rate each attribute of engineering managers on a ten point scale from "not important" to "critical". The displayed order of the attributes was random for each respondent. I provided the name of each attribute with a short description of examples demonstrating it in practice (see 6.2 for the text of each). I also provided a write-in question for respondents to provide attributes that they felt were important that we did not include; I received 123 responses. I reviewed the responses manually and found that they were paraphrasing or giving concrete examples of one of the 15 attributes or identifying a sub-case of one of the attributes and did not generate new input.

As with the interviews, all participants were instructed to discuss about the engineering manager role. That means that engineers and team leads were discussing a role that is organizationally above them, the engineering managers were discussing their own role, and the upper level managers were discussing a role that is organizationally below them.

The attribute **being technical** was mentioned by all interviewees, but the use of the term needed to be unpacked to provide better understanding (I explain this further in section 6.5). I, therefore, probed the attribute more with a scenario based question, asking respondents to pick one of two candidates for a manager position, and justify their choice. The first candidate was described as excellent technically and competent socially while the second one as competent technically and excellent socially.

Through the survey I collected gender demographics, geographical location, role of the respondent, role of the person the respondent directly reports to, size of the team the respondent is in or manages, number of years the respondent has been in their current role, and number of years they have been at Microsoft in total. This allowed me to check for differences in opinions from various demographics (e.g., gender, role, etc.).

I used Kitchenham and Pfleeger's guidelines for personal opinion surveys in software engineering research [114]. I followed the practices suggested by Morrel-Samuels for workplace surveys [151] such as avoiding terms with strong associations, using response scales with numbers at regular intervals with words only at each end, and avoiding questions that require rankings. The survey was piloted [24] with 199 randomly selected developers, leads, and engineering managers. The pilot included an additional question at the end of the survey asking if anything was unclear, hard

to understand, or should be modified in any way. I received 26 responses (13% response rate) and based on feedback, clarified the wording of several questions. The full survey can be found in the study's supplemental materials [2].

The final survey was sent to 3,646 people in the software engineering discipline spread across the strata described. The survey was anonymous, as this increases response rates [203] and leads to candid responses [151]. I received 563 responses, leading to a 15% response rate. Online software engineering surveys usually report 14% to 20% response rates [170].

Analysis. I used descriptive statistics to rank attributes and regression modelling techniques to identify demographics that positively or negatively influence the importance of the manager attributes. The details will be discussed in Section 6.6.

All the empirical evidence I present and discuss in the rest of the chapter about great engineering managers reflects the perceptions of the participants about attributes, actions and strategies. Hence, when I mention "the great engineering manager" in the text, it should be interpreted as *what engineers and managers perceive as important for engineering managers that are seen as great*. I use the former for brevity, and to avoid repetition.

6.3 Conceptual framework for great engineering managers

The study identified 15 high level attributes of great software engineering managers; they are listed in Table 6.2 with a description for each and a quote capturing the participants' impression. I also organized the attributes into a conceptual framework, in Figure 6.1. The 83 attributes identified during the interviews and surveys were qualitatively analyzed and card sorted in the 15 attributes I present in this chapter. Of these final 15 attributes, 6 (40%) map back to seed attributes; 4 (27%) to the Microsoft Poll and 2 (13%) to the management literature. The remaining 9 attributes (60%) were write-in attributes provided by the participants.

The framework is organized along two dimensions, labeled on Figure 6.1: the levels of interaction, and the engineering manager's functions. The engineering manager seems to be interacting on two levels; with the individual engineer, and with the engineering team or other entities in the organization. This is symbolized at the top of each column in Figure 6.1. The vertical, dashed-lined grouping in the framework

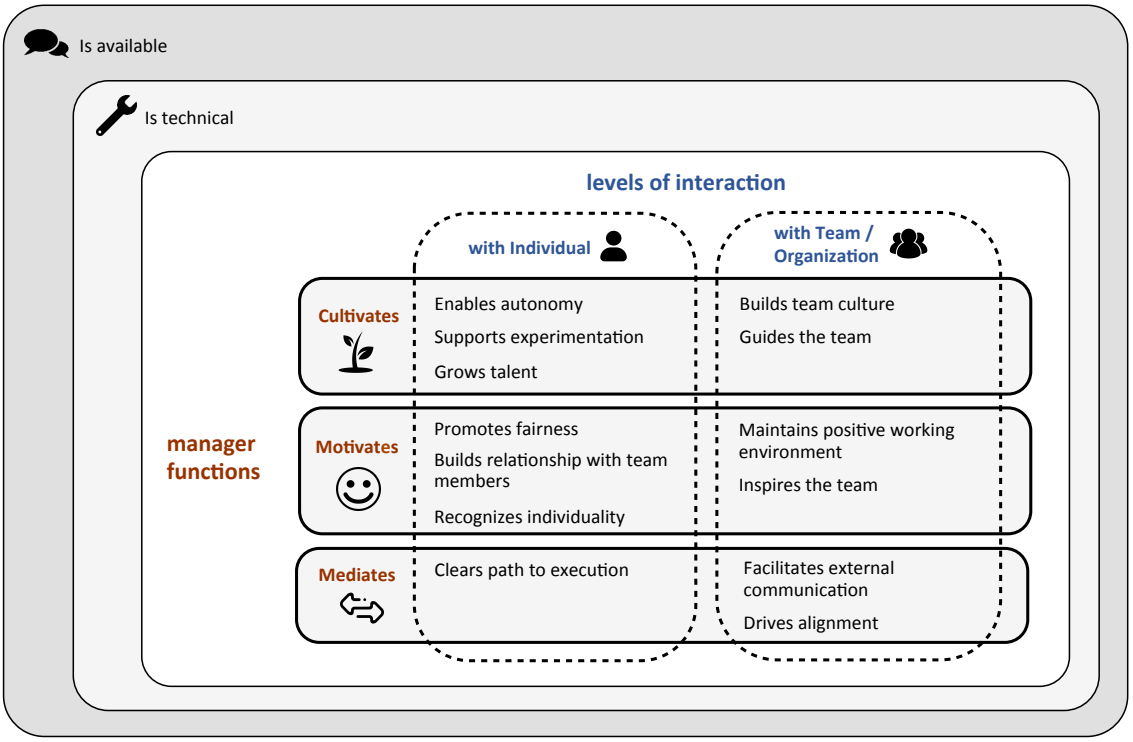

Figure 6.1: Conceptual framework for great software engineering managers

shows the relevant attributes for each interaction.

A second categorization of the attributes surfaced from the data, around three functions of the engineering manager: *cultivating engineering wisdom*, *motivating the engineers*, and *mediating communication*. In Figure 6.1 the functions are mentioned in short by the words *cultivates*, *motivates*, and *mediates*. These reflect the perceptions of the involved parties, not just the job description. The horizontal grouping with solid lines in Figure 6.1 shows the relevant attributes; I present all identified functions and attributes in detail in Section 6.4.

Throughout the chapter, I use icons to indicate the dimensions the attributes belong to. For the level of interaction, I distinguish whether the interaction is with the team (👪) or the individual (👤). For the manager function, I signal when the attribute relates to the manager cultivating engineering wisdom (🌱), motivating the engineers (☺), or mediating communication (⇦).

Two attributes – **being technical** and **being available** – are relevant in all interactions and functions. I note this because these attributes were often mentioned by participants as being pre-requisites to, or supportive of, the other attributes (*e.g.*, a manager would need to be technical to facilitate external communication with another

Table 6.2: Attributes of great software engineering managers, each with a short description and representative quote. The attributes in the table appear in descending order of importance based on the results of the quantitative analysis, described in 6.6.

Attribute and description	Representative quote
Maintains a positive working environment - to provide flexibility to balance work and personal life, energize the team through organizing events, celebrate team successes, and ensure good morale	*"Take them out to lunch, or have birthday cakes etc. Everyone needs to come in with energy to do their best."*
Grows talent - to provide opportunities for challenging work, suggest training for the engineer to gain industry-relevant skills, and provide actionable feedback to improve engineer performance	*"Your manager is your direct connection to your career; if they?re not giving you good projects you might as well not work here."*
Enables autonomy - to provide freedom on how engineers work, show trust and support for their decisions, and help engineers be independently responsible	*"I tell them where I would like to end up; the way there, they are the ones that know better how to get there."*
Promotes fairness - to show appreciation for the engineer's contributions, hold themselves accountable for the team's progress, and recognize value publicly while correcting the engineer privately	*"Think back to what you liked when you were 5 years old and what made you happy, what was the gold star. Cheerleading, people like it and they want it."*
Recognizes individuality - to understand each engineer's strengths and weaknesses, value diverse perspectives in the team, and fine tune the definition of success to each individual's talents and interests	*"I felt that my manager was interested in what I was doing, you feel like someone's in your corner and they are rooting for you."*
Inspires the team - to be viewed as a leader, to respond in situations individually rather than have general approaches, and demonstrate passion about their work, their team, and the company	*"In a battle no one follows a manager into war, everyone follows a leader, and it's about whether you are telling people what to do or if you are coaching them."*
Supports experimentation - to encourage the engineer to try out new things, and signal a safe environment for unsuccessful attempts	*"Discovering that something isn?t going to work is not a problem, it?s about evaluating what is the best solution."*
Clears path to execution - to shield the engineer from randomization, remove distractions and blockers, and help to resolve issues or conflicts	*"I had a manager, she kept the path clear for me to do my work, to go sit down and code for 10 hours. That was perfect."*
Drives alignment - to share information about higher-level context, explain the business intent for the product/service, create a mission with input from the team, and set clear goals for the trajectory	*"They have to know and believe in it, it's about what our mission is and why is that important, why does it matter."*
Builds team culture - to demonstrate the rules, norms, and habits of the team, and create ?what this team believes in? with input from the team	*"We have a culture of openness in the team; very open conversations about what works and what we should improve."*
Guides the team - to coach engineers on quality aspects (e.g scalability), provide guidance through appropriate questions to engineers struggling with their tasks, and help the engineer build independent decisions making skills	*"If the requirements change I don't want to redesign. I want to make sure they have thought about scalability and extensibility."*
Is available - to signal themselves as approachable, and devote time to the engineer when needed	*"I ask the manager if he has 5 minutes and he always says yes, and then 20 minutes later he is still there."*
Facilitates inter-team interaction - to act as a buffer with other teams and managers, negotiate what the team can provide when, and mediate their own team?s requests to other teams	*"I will make sure it bubbles up and I correspond with my peers to make sure that we get what we need, but you have to not micromanage."*
Is technical - to be knowledgeable about the system and technologies the engineer is working with, understanding the complexity of problems and solutions, and have input for design dilemmas	*"The manager keeps up with languages, platforms, development practices. Otherwise they will not have the respect of their team."*
Builds a relationship with team members - to take an interest in the employees' life outside work, and care about them as a person	*"Treat them as a friend, I think that is what I learnt, you have to build the relationship first, the work part is much easier."*

team about a dependency between modules). To indicate this, in the framework I show these two attributes encompassing all the rest.

I hasten to note that while I identified 15 attributes from coding the 83 that emerged from the interviews, they are not completely independent of each other and inter-relationships exist between them. For instance, the attribute **supports autonomy** may enable the attribute of **grows talent**. As this study is an initial exploratory foray into identifying and characterizing these attributes, I do not explore the subtle relationships between them.

In an effort to evaluate how well the identified attributes and the conceptual framework represent the ideas of participants, I conducted *member checks*, a technique used by researchers to evaluate the internal validity or "fittingness" of a study [220]. I reached out to those I had interviewed to elicit feedback on the attributes and conceptual framework. Of the 37 individuals interviewed, 31 were still employed at Microsoft and not on leave. Of these, 16 provided feedback (ten managers, three leads, and three engineers). All indicated that the attributes and framework accurately captured their views and experiences. Interestingly, one respondent indicated that it would be a high bar for a manager to embody all attributes while another indicated the attributes were required only for a "good manager" and that a "great manager" should exceed them. Many were enthusiastic about the results and asked to share preliminary drafts of this chapter with others.

6.4 The functions of great engineering managers

In this section I present qualitative evidence for the attributes. I use representative cases, examples, and quotes that came out of coding a discussion of each particular attribute with interviewees. The findings are thus contextualized, showing *why* attributes are considered important and *how* they are instantiated. I indicate whether a quote comes from an engineer (Ⓔ) or manager (Ⓜ), and the participant ID. The attributes are presented following the horizontal grouping in Figure 6.1: cultivates, motivates, and mediates.

The attribute **is available** has a simple description (see Table 6.2) and I have not included it in the detailed presentation of the attributes below. Yet, it was reported as important for helping engineering managers achieve the other attributes.

6.4.1 Cultivates engineering wisdom

Developers and managers described great engineering managers **enabling the autonomy** of engineers. The manager communicates and explains the desired outcome (see **Drives alignment** later), but the engineer is seen as the expert of implementation and owns decisions such as picking tasks and tools. One manager described this, linking it to higher speed in their team:

Ⓜ *"We inherited a pile of code, and discovered one piece was entirely busted. Should we fix it or should we rewrite it? The guidance was "get this thing to work 100% bulletproof". We started rewriting it from the ground up with TDD [Test Driven Development], the team made that decision. We got it figured out in an afternoon when it takes a person a week."* [P11]

The engineer needs context informing their decision-making in order to enact autonomy [178]. A common way of enacting autonomy, described in interviews, was involving the engineer in discussions usually owned by managers, e.g deadlines.

Ⓜ *"I leave almost every decision to the engineer if they feel strongly enough about it. Say a partner needs something by a particular deadline, even then I would give a lot of leeway. I would not say "we have to have this done by this time", I would present the rationale and ask how we can meet this, or if meeting this is even the right thing for us to do."* [P13]

The autonomy also manifests on the team level, with room for the engineers to collectively decide on coding conventions, and development process.

Engineers develop their skills as they try new things [66], while newcomers can familiarize themselves with a project's landscape [53]; the great engineering manager **supports experimentation**. Both new hires and senior developers favoured this attribute, for different reasons.New hires saw benefit in experimenting in that they build their skills. For senior developers, the benefit was that they have the chance to keep up with new technologies. However, experimentation should be safe and the engineering manager needs to signal that.

Ⓔ *"Nobody wants to report failures. Then it's less stressful to do what other people do, rather than try something new. It has to be somehow communicated that we will let you try out stuff with the assumption that if it doesn't work it's fine."* [P7]

Managers agreed experimentation with safe haven is important for getting good ideas on the table, but also try to be cautious of potentially introducing technical

debt.

Ⓜ *"We built someone's fun project into a product. Since we rely on that tool we have debt now to pay because the person did it as a hobby and wanted to make progress fast and not disturb any project work. We encourage that but if we don't do it with some quality gates, we inherit more debt later."* [P14]

The room for autonomy and experimentation supports another attribute of the great engineering manager, that of **growing talent**. The interviewees reported three main ways to enable talent growth.

First, by *providing opportunities for challenging work*; the great manager picks the scope and priorities for the team to have impact toward overall goals, and provide learning and experience.

Ⓔ *"You want to advance and grow and not necessarily do test automation for 10 years. I think that's where they need to look out for you and make sure you are working on something that will advance your skills."* [P2]

Second, by *providing timely and actionable feedback* to improve the engineer's performance; developers see such feedback as a signal that the manager is invested in their progress and career, and helps establish trust toward them. Managers commented that giving negative feedback is a tough process but that postponing giving negative feedback is a property of bad managers; the delay only leaves less time to the engineer to address their performance issues.

Third, managers described *providing learning opportunities* for the engineer to learn something on the job, instead of telling them how to approach it. One manager said:

Ⓜ *"I ask a new hire to take difficult tasks on. By failing we are going to find out where you need the help and then you will go to person X and ask them for help."* [P31]

While managers find that such a strategy works, they cautioned about people too introverted to collaborate with others or ask for help – especially if they have strong technical knowledge themselves.

The great engineering manager establishes and demonstrates the habits of the team, and what the team believes in; in this sense they **build the team culture**. The culture covers issues that have to do with the quality of work the team aspires to:

Ⓜ *"Messaging to the team "our goal is to ship a product with 0 known bugs" is a very aspirational goal, but it should be everybody's goal. You may not know what the bug*

is or how to fix it, that's a different thing, but there should be no concept in people's head that it's okay to ship with bugs." [P21]

The great engineering manager **guides the team** by coaching developers on quality aspects; implementation decisions are left to engineers, but the manager ensures that software quality aspects like reliability, scalability, availability, and extensibility are considered.

Similar to **enabling autonomy**, the technique that managers agree works better is to ask questions rather than give directives, and involve the team.

Ⓜ *"It's more about asking questions instead of providing direct answers, getting the team together for brainstorming to determine direction instead of making decisions for people. Even if you already know the outcome you need the team to get to, you can help them arrive to the same conclusion and guide the conversation that way."* [P27]

Developers and managers pointed out the importance for the developers to be convinced about the direction, connecting it to job satisfaction, and turnover in the team:

Ⓜ *"If the engineers don't feel they are doing something important it doesn't matter if it's going to get them a promotion or money because they will feel they are doing nothing with their lives, because they spend so much time at work. If they don't hear from my boss that something is important to do it's hard for them to buy into the idea of doing this, and that guarantees churn."* [P14]

6.4.2 Motivates the engineers

Developers and managers agreed that a great engineering manager **promotes fairness** in their interaction with the individual engineer. One perceived component of fairness was *actively showing appreciation for the engineers' work*:

Ⓔ *"They value your contributions, they give you feedback that what you have done is helpful, or brought success to the team or the organization. That is great motivation to engineers."* [P18]

Email – especially if upper management can be included in the communication to give more exposure to the work – and meetings were the commonly mentioned venues. Managers give opportunities to the individual engineers to present their work in team meetings where they can give positive reinforcement in front of the whole team.

Managers highlighted that showing appreciation for the engineers' contributions is important, following a maxim of "Praise publicly; correct privately" which has positive impact on the developers' trust and motivation. A great manager holds themselves accountable for mishappenings; it acts as a form of proof that helps developers be candid when having work challenges. A manager gave the following example:

Ⓜ *"Sometimes they forget tests for a scenario and there will be an email from management about it. I will take the responsibility, and then talk individually to the person to make sure we do a better job. They know they didn't get hit at that point and they trust you; you protected them rather than passing blame."* [P28]

An extension of demonstrating appreciation, is that the great engineering manager **builds a relationship with each team member**. Knowing about personal interests beyond work helps create common ground and empathy between the developer and the engineering manager; it contributes to motivating engineers, and helps build trust with the engineering manager. Managers especially insisted on how important this attribute is to great engineering managers, and how it is frequently overlooked. Managers mentioned a different approach to how they meet with engineers, based on this attribute.

Ⓜ *"Having 1-1 meetings in their office is better, it gives them home field advantage so they don't feel like they are going to the principal's office. We might talk about life and things at work, it goes back and forth. It makes them comfortable and allows them to open up about work things because we have camaraderie."* [P31]

The great engineering manager **recognizes the individuality of each engineer**; this attribute helps the manager tailor the work to the interests and skills of the engineer, and build a team with complementing skills. A developer described how this can impact productivity:

Ⓔ *"I had a manager asking me what I was interested in and giving me work related to that and I felt a lot more comfortable and happier. I had a manager try to mold me to their definition of what a good engineer does and I was probably working the hardest and yet my output was probably the least."* [P9]

The great engineering manager takes steps to **maintain a positive working environment** for the team. One example is the flexibility to achieve work-life balance; it signals a manager personally interested and invested in the well-being of the engineer beyond the professional level. A second example of maintaining a positive working

environment is energizing the team. One of the managers mentioned that the general feeling in the stand-up meetings shows the team's health.

Ⓜ *"We do daily scrum meetings that are about making fun of each other, making jokes at 10 in the morning. We can tell in the meeting a certain amount of health, and if people are happy they are generally good with their job. They will also talk to me if something is off."* [P31]

Finally, echoing the trait of showing appreciation on the individual level, managers pointed out that celebrating team successes is good for morale, and linked it to job satisfaction.

Ⓜ *"There is a lot of job satisfaction that comes from knowing that people care about what you're doing. Celebrations can also be about transparency, I share all the feedback that we get from higher up – good and not so good – people like to see a direct line between what they do on a daily basis to senior leadership"* [P35]

Engineers also found having a positive working environment important, sometimes even more so than the work they do:

Ⓔ *"I would compromise on the work or the product if the team has a positive working environment which I think the manager influences deeply."* [P9]

The great engineering manager **inspires the team**. This attribute was almost exclusively mentioned by managers; they felt that the engineering managers that are great are viewed as leaders, although in organizational culture, management and leadership are seen as different functions [83]. The simplest way to explain the difference between manager and leader was by drawing the line between enabling and giving a directive.

Ⓜ *"Managers issue decrees, whereas leaders encourage everyone to move to a certain direction. Are you telling them what to do in a way that they are on board with it, they get it, and they see it as a growth opportunity, not as a directive?"* [P33]

6.4.3 Mediates communication

The engineering team has dependencies and communication needs with other teams in the organization [17]. A great engineering manager mediates information flow between their team and other stakeholders.

First, the great engineering manager **clears the path to execution**. Engineering managers recognized that any type of interruption can be detrimental to the engineers' productivity:

Ⓜ *"The operational space for engineers is that flow moment where they are deep into writing code. The last thing they need is some random person – business person, or the manager – going "hey, have you got a second?", you just killed half their day in that moment. Giving them the space to focus is important, just be the person that tells other people to go away. Defend their time."* [P36]

The concept of "flow", originally introduced by Csikszentmihalyi [48] has been popularized and is now well-known and frequently cited in software engineering practice. In a state of flow, the software engineer is focused on their work and their performance is optimized [158]; unfortunately, the state of flow is fragile, and a developer whose concentration is broken needs significant time to recover. The concept of "flow" in this case is distinct from how "flow" is described in lean manufacturing and product development where it refers to a sequence of development actions, each clearly adding value to the creation of a product [75] (citing [214]).

The great engineering manager acts as a noise filter, protecting the developers' flow state and keeping the path to execution clear for them. A manager described this further during the member checking:

Ⓜ *"[...] as an Engineering Manager I often feel that my job is to shield the team from distractions, among other things, basically to get out of their way, keep the business out of their way, and let them execute against clearly communicated/understood/shared goals."* [P14]

One strategy mentioned by the interviewees was for the engineering manager to filter incoming requests to the developer, and discuss with them individually if and how to prioritize them. The engineering manager also handles requests or changes coming from upper management, in the service of flow. Developers and managers see the great engineering manager shielding the team from changing requirements ("randomization", in corporate lingo), and manage upwards to negotiate workload. Through insulation from randomization, the engineering manager helps the team maintain its collective flow and performance.

The great engineering manager also **facilitates external communication**; with other engineering teams, and with upper management.

Often, the engineering team has outgoing requests for other teams; the great engineering manager pushes to get what their team needs, especially if it is critical to work that is underway. In facilitating communication with external entities the manager is not bypassing the engineer or limiting their autonomy; the manager is instead using their status and connections to achieve results on behalf of their team. The other forms of external communication are performance reviews and other meetings with upper management, where the engineering manager represents the team.

The great engineering manager navigates the two attributes, **clearing path to execution** and **facilitating external communication** with upper management. On the one hand, to shield their team, the engineering manager may prioritize requests from upper management lower, or decline them. On the other hand, to advocate the teams' work, the engineering manager needs to showcase the team's impact on achieving organizational goals, which are closely related to upper management requests. To balance between the two situations data and prior success are persuasive factors.

Ⓜ *"For stopping randomization I bring soft data: names, efforts assigned to them, show we don't have capacity for more, just different priorities. Sometimes management may ask for things that are, regardless of our capacity, probably wrong, not timed right, or not scoped enough. Then I bring a deeper analysis of what the problem space is."* [P14]

The engineers' focus is on implementation details and delivery of features; there lies a risk that the goals they see are removed from the strategic vision of the organization because they are unaware of business drivers [205]. The great engineering manager actively **drives alignment** between individual output and the organizational scope. By sharing strategic information about goals, and clearly explaining the intent and desired outcome, the engineering manager ensures that effort is channeled in the right direction, and that engineers know enough to make informed decisions as part of their autonomy on implementation. An engineering manager explained:

Ⓜ *"Every team member are the experts of their area, they know the best things that can come out, but they can push in different directions. Those are their personal contributions and they feel ownership to them. My goal is to see how we can align these contributions in a way that they still feel like their own and they are now all pushing in the same direction, helping the organization move the business forward. "* [P13]

As with **guiding the team**, the great engineering manager *consults* with the team

about achieving the higher level goals and does not enforce a certain path.

Ⓜ *"I pose a question like "hey, here are some of the challenges our business is facing, how do you think you can help in these respects? Here are some seeds of lines of thinking, but help us come up with a strategy together, what are things we can do to get there.""* [P13]

6.5 Being technical

The attribute of **being technical** emerged consistently in the interviews, but in a rather unexpected way that warranted more careful examination.

By the respondents' account the engineering manager, as a rule, does not produce technical output, i.e. does not write code. Respondents were explicit about a *great* engineering manager not making technical decisions; rather the engineers do.

What, then, is the role of technical knowledge for an engineering manager? According to the interviewees, a technical engineering manager:

- Is respected by the team
- May be more vigilant about quality issues
- Would be a fair evaluator of the engineers' work
- Empathizes with engineers and **clears the path to execution**
- Would represent the team better, both to other teams and to upper management

I noticed all interviewees using the same, albeit abstract, lingo of the engineering manager **being technical**. To address the possibility that the interviewees meant different things concealed by the use of the same term, I contacted them and asked them to clarify what **being technical** meant for them; replies came back from 25 interviewees. The overlap in their responses signals they originally meant similar things; I have mind mapped their input in Figure 6.2. **Being technical** was described in terms of what the engineering manager should be in a position to comprehend, and how it helps.

The engineering manager needs to have enough technical knowledge to understand the engineers' work; tools and technologies they are working with, and the system they are building. Languages and frameworks were the most cited examples of technologies.

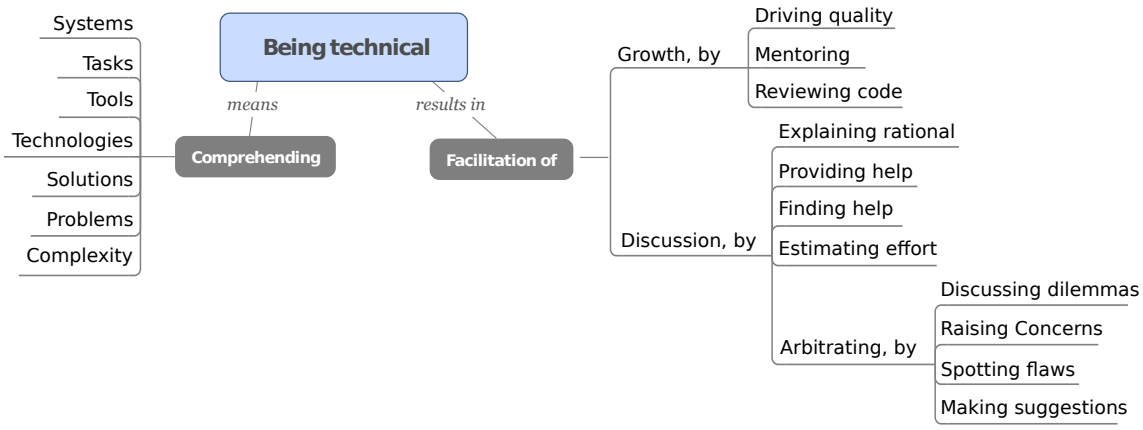

Figure 6.2: What falls under being technical, according to the participants' input

The engineering manager should also understand the tasks engineers work on, the complexity of problems they report, and the proposed solutions. With a nuanced view of the engineers' work, the engineering manager can facilitate their growth; they mentor and set the standards of quality through code reviews and feedback.

Comprehension also helps the engineering manager facilitate discussion of approaches to implementation, or what to build next. An engineering manager with enough technical knowledge can explain the rationale between alternatives, and ask the right questions about why one is preferable to another.

Engineers often encounter design dilemmas [50], and discuss them with the engineering manager, who is expected to act as an arbiter. In an iterative process the engineering manager makes suggestions and helps the engineer navigate their way out of the dilemma by spotting flaws and raising concerns. Overall then, **being technical** means that the engineering manager *has enough knowledge to hold informed discussions that will help the engineer make decisions.* Developers particularly mentioned that having enough knowledge cannot be faked, and that they can tell even from a very brief conversation if someone has enough technical understanding.

🄴 *"It is very easy for the engineer to understand if the other person is technical or not. For example I may talk about something that I am working on and then I will say what I will do and how long it will take. When you discuss there will be some technical details, and they will be unable to say if you need some help or parts from other teams."* [P18]

There is an underlying tension at this point. Engineers at Microsoft (and most

companies in the tech industry[7]) are usually promoted to managerial posts because of their technical excellence; this explains why they find "letting go" challenging as one manager of managers explained.

Ⓜ *"The biggest piece is giving up the need to jump in and do. New managers see this all the time; it may take the engineer 3 days to do something and the manager one. They have to step back and let the other person do it in 3 for the long term success of the team, otherwise that's not people management. That's counter-intuitive to someone who was promoted the first time because they were the best technical person on the team. You think as a software company as you move up the most senior person should be the most technical person; that probably is true from an intellect perspective but that's not how they spend their time"* [P29]

Engineering managers who kept on producing technical output when they became managers described situations of exhaustion and burnout; they also recognized that when they continue to do technical work there are less opportunities for the engineers to grow. This highlighted that while a level of technical knowledge is required for a engineering manager – enough to cover the items in Figure 6.2 – technical *excellence* is **not** the most critical factor for greatness. This point was further explored in the validation survey.

I explored this point further in the survey with the scenario based question described in the methodology. The majority of respondents (75%) indicated they would hire someone with average technical skills and excellent social skills (*social manager* henceforth for brevity), over the reverse (*technical manager*), as an engineering manager. When elaborating on their choice in their response, the reasons they gave are congruent with what was explained to me in the interviews, especially under the "tends to the motivational aspects" grouping. For example, the following quote from a survey participant's response to the hiring question captures the general sentiment of the participants that chose the "social" manager:

💬 *"Even though he is not technically 100% great, this is something which he/she can learn fast. Inspiring others is not something you can learn overnight and this skill is precious."*

I provide additional details from the survey about the **being technical** attribute in the following section.

So far I have explained all the attributes found in the qualitative investigation and

[7] `http://bit.ly/1bxQ3ni`

why they are important, and I have described strategies to apply them in practice. I also wanted quantitative support across a broader population, which led me to deploy the survey described in 6.2.2, and the results of which I present in the rest of this section.

6.6 Quantitative support through survey results

Here I provide survey results about the attributes' relative importance, and the view of attributes across demographic groups. I only display statistically significant results ($p < 0.05$) in the tables and figures; in the text I provide coefficients in parentheses.

Figure 6.3 shows the distributions of the importance score for each of the attributes, in descending order of mean score. For each attribute, the top portion shows a violin plot for data from engineers and the bottom from managers. The thicker horizontal bar for both top and bottom indicate the interquartile range and the small vertical line indicates the mean.

A Principal Component Analysis showed that while there is non-trivial personal tendency to responses (personal tendencies appear to account for 33% of the variance), the responses for the attribute ratings were largely independent [108]. 13 of the 15 principal components are required in order to capture 95% of the variance in the responses and (except for the first, which captured personal tendency) each component had exactly one dominant attribute that had a weight of over 0.5. This provides quantitative evidence that the attributes capture ideas with little overlap and they are largely independent. All attributes are rated quite high, with even the lowest, **builds relationship with team members** receiving an average rating of 7.47. The information in Figure 6.3 corresponds to the views of all types of respondents in the engineering discipline. Ratings were fairly consistent as well, as the interquartile range for eleven attributes was only two units and for the others, three units.

The highest rated attribute for great engineering managers is **maintains a positive working environment** (mean of 9.05), which was part of the motivational aspect of managers. **Grows talent** (8.98) and **enables autonomy** (8.91) follow in rating, demonstrating the importance of the engineering manager helping engineers develop their talents and allowing them to organize their work.

Being technical (7.84) ranked as 14th of the 15 identified attributes for great engineering managers; agreeing with input from interviews that technical knowledge is important, but not the most important attribute.

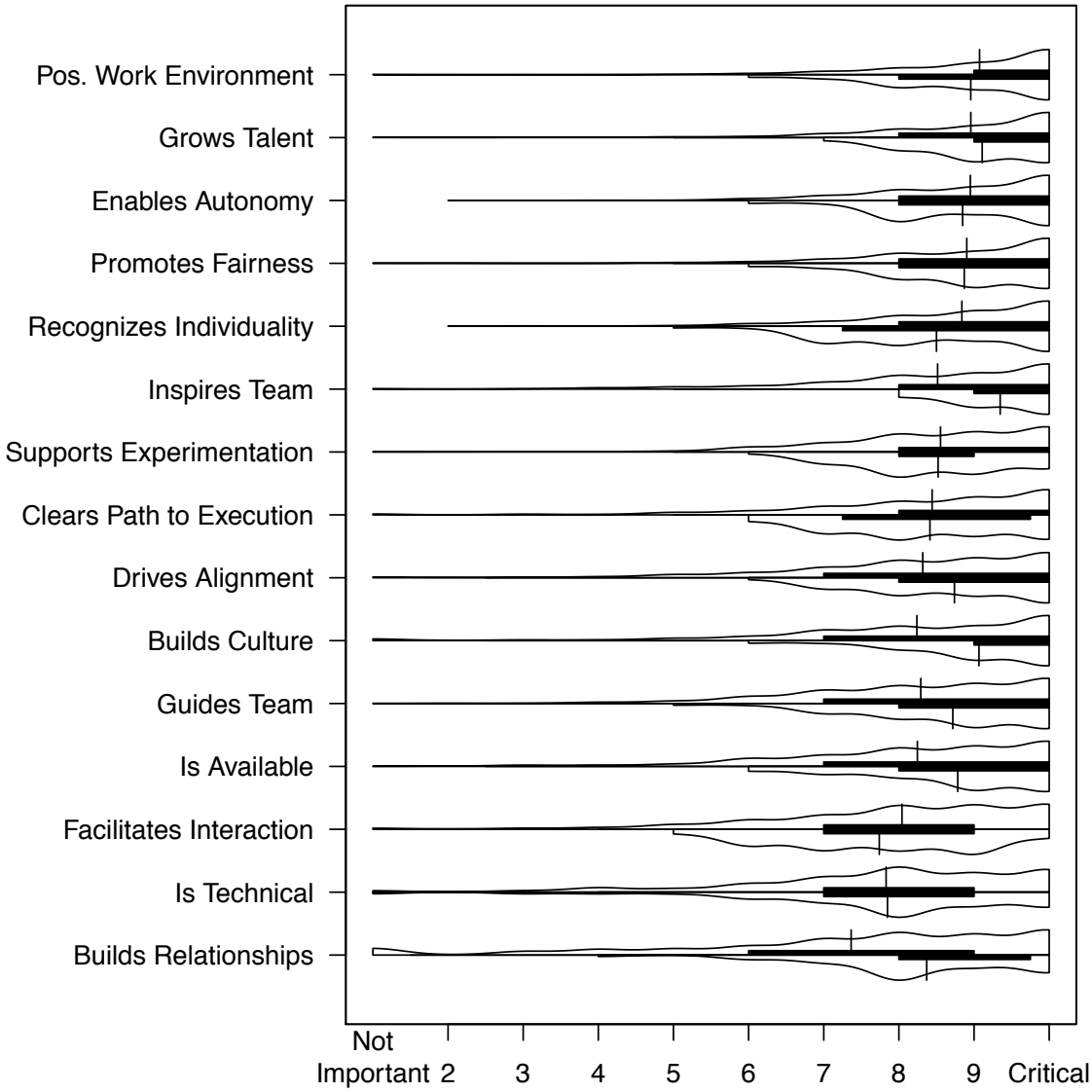

Figure 6.3: Violin plots of the distributions of importance given to each attribute. For each attribute, the top portion shows data from engineers and the bottom from engineering managers. The thicker horizontal bar indicates the interquartile range and the vertical line indicates the mean.

I related the respondents' hiring choice of a social manager (vs. a technical manager) with two logistic regression models. The first model related the hiring decision to demographics. The results show that engineering managers (+1.17) are more strongly in favour of hiring a social manager as compared to engineers. The second model related the hiring decision to the attribute ratings. Here, 5 attributes had significant coefficients. Positively linked to the choice of a social manager were **inspires the team** (+0.18), **maintains a positive working environment** (+0.24), **builds team culture** (+0.15), and **being available** (+0.17); that is, respondents with higher ratings for these attributes were more likely to hire a social manager. On the other hand, and not surprisingly, respondents who place more importance on the attribute **being technical** are less likely to hire a social manager (-0.71).

The results of the interviews and survey taken together point to the following conclusion: *great engineering managers need a sufficient level of technical knowledge – not excellence – while people management attributes are essential.* I further discuss this finding and its implications in 6.7.3.

I looked for demographic differences in the importance rating for the attributes. For each attribute I built a linear regression model using the demographics (gender, location, role, group size, experience) as independent variables and the importance rating of the attribute as the dependent variable. Table 6.3 summarizes the results and shows only the statistically significant coefficients, grouped by demographic. Each coefficient indicates the change in importance for an attribute relative to an artificial baseline (intercept) constructed as the majority class for each categorical demographic (e.g. gender, region, etc.) and a value of zero for numerical values; the regression model intercept corresponds to a male engineer, working in the U.S. in a team of zero people with no years of experience at Microsoft. In interpreting the magnitude of the change, recall that since the interquartile range of responses for most attributes was just 2 units[8], a 1 unit increase is a non-trivial change. The change indicated for engineering manager group size and years at Microsoft is the change when the demographic increases by one. For example, engineers in a team of 100 people value **enables autonomy** 0.90 units less than those in a team of 10 people.

A finding that stood out was that female participants rated **being technical** as more important (+0.84), relative to male participants. Coupled with other attributes that female participants rated more important for engineering managers – **grows talent** (+0.46) and **guides the team** (+0.53) – the findings indicate that female participants

[8]See the thicker horizontal bars in Figure 3.

Table 6.3: Change in ratings of attribute by demographic. Each demographic's rating for an attribute is compared to the ratings for that attribute for the majority class in that demographic category. For instance, females are compared to males and India is compared to the U.S. Each difference shown is statistically significant. Coefficients for *Mgr Group Size* is the change per additional person in the group being managed and for *Years at Microsoft* is the change per additional year at Microsoft.
Positive coefficients indicate higher importance. Negative coefficients indicate lower importance.

Demographic	**Attribute**	**Change**
Role:	builds a relationship with team members ↑	1.21
Managers (90)	inspires the team ↑	0.98
compared to	builds team culture ↑	0.97
Engineers (465)	is available ↑	0.60
	grows talent ↑	0.54
Gender:	is technical ↑	0.83
Female (59)	inspires the team ↑	0.80
compared to	facilitates external communication ↑	0.62
Male (494)	guides the team ↑	0.53
	grows talent ↑	0.46
Region: China (32)	is technical ↑	1.02
compared	recognizes individuality ↑	0.70
to U.S. (410)	guides the team ↑	0.65
Region: India (56)	builds team culture ↑	0.99
compared	builds a relationship with team members ↑	0.85
to U.S. (410)	grows talent ↓	−0.55
Region: Europe (50)	drives alignment ↓	−0.53
compared	recognizes individuality ↓	−0.50
to U.S. (410)		
Mgr Group Size	clears path to execution ↓	−0.01
(in people)	enables autonomy ↓	−0.01
Years at Microsoft	grows talent ↓	−0.03

seek to learn from technically stronger managers.

One of the goals in the study was to investigate how engineers and engineering managers compare in their views of what attributes are important for engineering management. While there is alignment in views, the statistical analysis shows differences between the two groups; engineering managers consider certain attributes more important than engineers do. Referring to Table 6.3, the attributes with the largest coefficients are related to motivational aspects, they are the attributes of **building a relationship with team members** (+1.21), **inspiring the team** (+0.98), and **building team culture** (+0.97). Engineering managers thus appear to rank attributes that create a feeling of being part of a team higher than the engineers do.

Growing talent is another attribute that engineering managers seem to value as more important, compared to engineers (+0.54). However, **grows talent** ranked second (out of the 15 attributes) in importance for either group, with managers seeing it as slightly more important.

6.7 Discussion

I presented the study and the evidence about the attributes of great managers that matter in the eyes of both managers and developers. Below I discuss the findings in light of a recent study at Google about the behaviours of great managers, as well as higher-level issues surrounding the findings and their implications for research and practice.

6.7.1 Comparison to Google's study of managers

The findings in the study I presented are in line with the 8 behaviours that Google identified as key to its managers [82]. In the course of a year, Google coded and analyzed interviews (both with current employees and employees leaving the organization), quarterly performance reviews, feedback surveys, and data on team performance to understand manager behaviours and their impact. While the wording of the findings may differ between the two studies, the spirit of the behaviours/attributes is similar. In my study I have independently identified the 8 behaviours in the Google study, as attributes of great managers; I provide the mapping between the two studies in Table 6.4.

Furthermore, *the study at Microsoft has identified additional attributes*; for ex-

Table 6.4: Comparison of findings with Google's study of managers

Manager behaviour in Google study	**Description (taken from the Google study report in [81])**	**Manager attribute in our study**
Is a good coach	Caters to the team members' skill set and personality with guidance and feedback, and pushes them to grow while still making them feel supported	Grows talent & Recognizes individuality
Empowers team and does not micromanage	Trusts the team to manage their work as they see fit, while still be available for questions	Enables autonomy
Expresses interest/concern for team members' success and well-being	Is caring, ensures the team members achieve their goals while ensuring everyone on the team feels personally as if they are valuable	Builds a relationship with team members
Is productive and results-oriented	Is relentless in removing obstacles for the team	Clears path to execution
Is a good communicator	Encourages open dialogue that permits team members to share issues and concerns	Builds team culture (partial fit)
Helps with career development	Shows that career development is not just promotion, but also growth	Grows talent
Has a clear vision/strategy for the team	Takes time to collaboratively create a vision and share and act on it	Drives alignment
Has important technical skills that help him/her advise the team	Has deep knowledge of the infrastructure, willing to get to the bottom of a problem	Is technical

ample, the participants placed importance on how the manager **inspires the team**, how they **support experimentation**, and how they **facilitate external communication**. As Google's study drew on insights about managers in several departments in the company — *not specifically software engineering* — the attributes I have identified reflect what is perceived as most relevant to the particular discipline. The study also provided a ranking of the attributes, by perceived importance. Further research should include the study of more organizations to identify additional attributes and check the applicability of the ones I have offered here.

6.7.2 The reimagined role of the engineering manager

The presentation and analysis of the identified attributes and the conceptual framework in Figure 6.1 allow us to see the role of the engineering manager. I see the role as *reimagined* to fit environments that give decision making power to developers. In that sense, we saw that the manager is **not** the one that makes the technical decisions or produces the technical output; there was consensus that the engineering manager should operate as an engineer as little as possible, if at all. The manager is also **not** the person defining and assigning work; the developers own these decisions and the responsibility to see them through. The role of the engineering manager surfaced to

be about indirectly helping the engineers and their teams to function as effectively as possible. As the Agile Manifesto highlights, that relates to the role of the manager, not much guidance is needed for individuals besides providing them with the environment and support they need [13].

The study's findings agree with the conclusion in [128] that productivity is not the only criterion for excellence and that *how* engineering is conducted is important. The engineering manager seems to introduce the developer to the other aspects of engineering that matter – for example by maintaining an environment that allows the engineer to experiment safely, or to ask for help from others. Also, with attributes such as **enables autonomy**, **supports experimentation**, **guides the team**, and through arbitrating decisions supported by **being technical**, the engineering manager cultivates effective decision making behaviour to engineers, getting them from good to great. This is especially important and fitting with an environment where autonomy means that developers make decisions – the great engineering manager teaches the developers how to make effective decisions.

The manager also contributes to making autonomy work at scale by **driving alignment**. By sharing information, explaining the strategic intent, eliciting feedback and getting the developers on board, the engineering manager essentially sets the boundaries within which developers can make autonomous decisions *that will benefit the organization*. As we saw in Chapter 5 and the literature, having clarity about the organizational goals and a clear link between the individual work and the strategy helps developers make more effective decisions about their work.

6.7.3 The interplay of management and autonomy

The manager's impact on software development

If they are not producing technical output and they are not making technical decisions, how do engineering managers affect software development?

Interviewees drew their own links between attributes and outcomes; they described the manager's impact on productivity through motivation, software quality through facilitating the team's technical work, and helping developers grow their skills.

As part of the validation survey, I asked participants how an engineering manager can impact the software developed by their team, in a positive or negative way. The 255 collected responses were coded and aggregated – participants reported a manager's actions or behaviours having an impact on motivation, software quality,

Table 6.5: The top 3 coded responses to the question "How can an engineering manager positively or negatively affect the produced software?", according to frequency of mention. For the productivity-related factors there was no reported negative impact.

Motivation factors			
Positive impact	**Maps to attribute**	**Negative impact**	**Maps to attribute**
empowering (19)	enables autonomy	micromanaging (27)	enables autonomy
shielding (15)	clears path to execution	unclear with direction (5)	drives alignment
focusing team on impact (8)	drives alignment	opinionated (3)	is technical
Software quality factors			
Positive impact	**Maps to attribute**	**Negative impact**	**Maps to attribute**
create good working atmosphere (27)	maintains positive working environment	not listening (19)	enables autonomy
connect team to org mission (14)	drives alignment	playing favourites (8)	cultivates fairness
experimentation (12)	supports experimentation	create the software themselves (5)	is technical
Productivity factors			
Positive impact	**Maps to attribute**	**Negative impact**	**Maps to attribute**
coaching for teamwork (10)	guides the team	none mentioned	not applicable
trust the team members (8)	enables autonomy		
help engineers improve (7)	grows talent		

and productivity. The codes were not mutually exclusive – i.e. a participant may have included a number of ways they see the manager impacting the software, and each of them was recorded. The result was 67 obtained codes, with saturation reached before half of the responses were coded. After two rounds of coding and the subsequent aggregation of codes I also mapped the codes to attributes in the framework from Figure 6.1 if applicable – it turned out that all the ways a manager can have impact mapped to the framework, which provides even more confidence about the coverage of the framework. For brevity, I am presenting the top 3 responses for positive and negative impact in Table 6.5. The number in parentheses represents the number of mentions of that behaviour or action.

The most frequently mentioned way a manager can positively impact the produced software was to "create a good working atmosphere" – this was linked to an impact on software quality. The response maps to **maintains a positive working environment**, which was the highest ranking attribute by both engineers and managers. The most frequently mentioned way a manager can have a negative impact is by micromanaging, which was seen as damaging to motivation. The most frequently mentioned positive impact on motivation was empowerment – micromanagement's mirror image – which maps to the **enables autonomy** attribute.

Participants also found that a manager that connects the team's mission to the organizational mission and focuses the team on making contributions with impact for the organization (both relating to the **drives alignment** attribute), positively impacts motivation and software quality; on the contrary a manager that is unclear about the direction the team should be taking has a negative effect on motivation. Finally, the participants found that a manager that coaches the team to work together has a positive impact on the team's productivity.

When discussing the interplay of management and autonomy then, based on the above, enabling autonomy and empowering the team members is one of the most recognized ways a manager can positively impact software development. It appears also that the autonomy is discussed alongside alignment in a similar way as in Chapter 5 – the capacity to use the organizational strategy as a framework for a team's goals and tasks. The manager can positively impact the team's software and the motivation of its members by clearly explaining strategic intent and focusing the team's efforts to having impact – the developers can then make decisions on which is the best way to accomplish that. In Atlassian's case (Chapter 5) alignment was facilitated by blog posts and other transparent information as well as program managers, while in Microsoft's case the engineering manager is the role that helps development teams see the connection between their work and the organizational strategy.

The responses also make it clear that the manager has a second-order effect on the software, rather than a direct one. They do, however, blur the lines of the effect – from examining the responses, the lines between impact on motivation and impact on software quality are unclear. For example, creating a good working atmosphere was mentioned as impacting software quality. Yet, it is hard to imagine that the work environment has a direct impact on software quality. It is more probable that a good working atmosphere is more conducive to productivity or motivation (for example through shielding or **clearing the path to execution**), which ultimately leads to better quality software to be produced. In the same vein, the manager's ability to connect a team's mission to the organizational mission (i.e. to explain the connection and make sure it is reflected in the team's work) can positively affect motivation (by getting buy in from developers) and productivity (by focusing resources on the right things), before it shows up as an improvement in software quality.

The suggested impact requires more investigation; I have provided some initial evidence of how the manager impacts the produced software in an important – although indirect – manner. This is enough to generate hypotheses at the moment,

which should be put to the test. Further work should focus on discovering the steps and links through which the impact materializes, and whether (and how) these can be operationalized and measured. These suggested links can inform research seeking to model the software development activity (e.g the Intermediate COCOMO model [145]), or empirical studies studying developer and software team productivity (such as [169, 41, 144]).

Management research in other domains can suggest further links to investigate. To demonstrate, the attributes of **guiding the team** and **enabling autonomy** are reminiscent of certain constructs of psychological empowerment [190]. Psychological empowerment is a form of increased intrinsic task motivation, and is influenced by *meaning* (the purpose an employee sees in their work) and *self-determination* (an employee having a choice in actions that relate to their work). Indeed, the interviewees in the case study I presented described autonomy from the manager and getting to do challenging work as empowering and motivating. Psychological empowerment correlates to innovative behaviour [190] and organizational commitment [9]; such correlations can guide further work to measure management impact.

The attributes of **builds a relationship with engineers**, and **maintains a positive working environment** share similarities in spirit with *employee perceived fit* with the organization, and may be affected by it. *Employee perceived fit* has been found to correlate to organizational commitment, job satisfaction, and turnover intention [29]. Finally, the attribute of **promotes fairness** is similar to the *perception of procedural justice*, known to correlate with job satisfaction and, in turn, with performance [27]. Such similarities demonstrate how prior work in other domains can – in conjunction with the findings of this study – lead to testable hypotheses for further empirical studies.

Different things matter

By reviewing the differences in perceptions between engineers and managers, we saw that managers rate higher attributes that relate to the team (**inspires the team** and **builds team culture**), or make engineers feel part of a team (**builds relationship with team members**). One explanation could be that the difference in perceptions is the result of the evolution in the managers' views due to their role; they may see an influence of these attributes on outcomes. Another possible explanation is that, as engineering managers have undergone management training, their different views may

be an artifact of materials or resources they became acquainted with. Future research can investigate these aspects further.

One of the unexpected findings in the study was the role of technical excellence for a great engineering manager; if it is not the defining characteristic, what is? Developers and managers highlighted people management skills as the critical element of the great engineering managers; however, these are not skills they have prior knowledge or training in, and is actually not what got them promoted to the managerial position in the first place:

Ⓜ *"It's a big career choice to choose management or technical because as much as people think you can do both, you can't. Most engineers tend to be introverts and that's how they got into technology, to just automatically assume they are going to be great people persons by giving them a new title is not true generally. It takes a lot of work to learn some of those behaviors that are not natural as a technical person."* [P29]

This view is consistent with discussions among software engineering practitioners about how engineers find the transition to managerial roles difficult [9]. One of the identified issues is that the necessary people management skills are not what got engineers promoted in the first place. There seems to be a concept of *diminishing returns of technical excellence* for engineering managers; while some level of technical knowledge (enough to cover the elements in Figure 6.2) is needed to understand and facilitate engineering work, after a point the focus on technical matters can jeopardize the people management aspect.

Engineering managers are, then, required to limit thinking and acting as an engineer, and instead focus on the people management aspect of their role, and exercise their social skills. This finding runs almost opposite to usual reward and promotion systems in large organizations, a paradox often satirically referred to as "the Peter principle" [10]. That the engineering manager does not need to be the most technical person on the team raises questions about how to select engineering managers, and our findings provide food for thought about which traits to look for.

There is an isolated case giving a different view on the role of technical competence. A study by Artz et al. [7] from economics reported that a boss's technical competence is the single strongest predictor of employee well-being. The study was based on

[9] `http://bit.ly/1bxQ3ni`

[10] `https://hbr.org/2014/12/overcoming-the-peter-principle`

35,000 randomly selected employees from various workplaces. Although the study was published in 2016, the data it used comes from National Longitudinal Surveys of Youth[11] in the past; two in Britain (in 1990 and 2000) and one in the United States (covering the time 1979-1988). It is, however, unclear which professions are included in the survey or what part of the population corresponds to software engineers.

Following from the implication about selecting engineering managers, is also an implication about training them. The input I provided could be used to augment existing training programs or design support activities and customize them to engineering managers. I highlighted that the people management aspect is for the most part foreign to engineers, yet critical for engineering managers; interviewees identified needing guidance in communication and generally "soft" skills. Engineering managers pointed out that mentoring would be especially helpful as early as possible before they take up the new role; this was seen as a way to clarify the practical aspect of what the role entails, and avoid pitfalls experienced by others.

6.8 Threats to validity in the study

The phrase "great software engineering manager" means different things to different people. While I intentionally left it undefined during the investigation in an effort to rely on participants' own opinions and values, there is clearly no universal definition and thus the results aggregate the views of people with diverse experiences.

While I interviewed many people, the goal was *not* to capture a representative random sample but to collect different perspectives. As such, I chose a stratified sampling approach in an effort to capture a wide range of responses from a diverse group. I then used these to inform the survey which was deployed broadly enough to provide representativeness and a sample large enough for statistical significance.

Because one of the primary instruments was a survey, I was concerned that the right questions were included and presented the right way [78]. To address construct validity [132], the survey questions were informed primarily from analysis of the interviews with software engineers and software engineering managers. I also deployed a pilot survey which led to feedback that allowed me to fine tune the questions in the final survey.

With regard to external validity [186], the analysis comes wholly from one software

[11] `https://www.nlsinfo.org/content/cohorts/nlsy79`

organization. This makes it unlikely that the results are completely representative of the views of software managers and engineers in general. However, because Microsoft employs tens of thousands of software engineers, works on diverse products in many domains, and uses many tools and processes, I believe that the approach of randomly sampling improves generalizability. In an effort to increase external validity, I have made the survey instrument available [12] so that others can deploy it in different organizations and contexts.

6.9 Conclusion

In this chapter I presented a study about the attributes that developers and managers agree make great managers, in an environment that advocates and practices autonomy for its employees. Through the study we learned that in a setting of *collaboration via aligned autonomy*, the role of the manager is reimagined into one of cultivating engineering wisdom, motivating, and mediating external interactions. In this study autonomy took the form of empowerment — the freedom to organize one's work came with the responsibility to use the business strategy as the guiding principle. The engineering manager facilitates the empowerment by supporting engineers to use their judgement and keeping them aware of the organizational objectives.

We saw that the manager was discussed as a facilitator, a communicator, a motivator, a mentor, and a conduit between stakeholders – some of these qualities have traditionally been associated with leaders rather than managers. The results seem to indicate that since some of the earlier tasks of the manager – like task assignment – now are owned by developers, the reimagined manager assumes more of a leadership role. In this sense, the critical criterion is not technical mastery; it's people management skills that are critical and will end up impacting the productivity and motivation of the team and – ultimately – the software it produces. Engineers that move on to become engineering managers need to be aware of these requirements. The engineering manager's role seems to be a transformation more than a transition and that calls for a shift in mindset. The attributes I presented in this chapter can help managers know what to focus on.

The next chapter is dedicated to synthesizing the findings so far. I will discuss the findings across the three case studies presented in Chapters 4, 5, and 6, and map

[12] `https://github.com/cabird/Engineering_Manager_Study`

the similarities and differences between them. The synthesis of all the findings will be represented in the framework of *collaboration via aligned autonomy* for commercial software teams, as it surfaced from the empirical studies.

Chapter 7

A framework of *Collaboration via Aligned Autonomy*

"Nothing clears up a case as much as stating it to another person."
Sherlock Holmes – Silver Blaze

In the previous three chapters I presented the case studies that make up the empirical evidence behind the thesis. The studies were presented each with their own methodological details, their contained findings, and a discussion that generated insights about the meaning and use of autonomy. In this chapter I am using the findings and insights of all three studies in an inductive manner. First, I provide a definition for autonomy by reviewing how it was conceptualized and enacted in the studies. If Autonomy is to be used as the basis of collaboration, however, it seems that it needs to be *aligned* with a company's goals and I provide a definition for Aligned Autonomy as well. Second, I describe and apply interpretive evidence synthesis to put together a theoretical construct that can have applicability beyond the three studies I have conducted. The synthesis results in a framework of *collaboration via aligned autonomy* which I present and discuss in terms of implications for theory and practice, as well as limitations.

7.1 The faces of autonomy

I used the studies I described in Chapters 4, 5, and 6 to discuss ways that autonomy is compatible with collaboration and practices that support the combination of the two. A characterization of autonomy emerged alongside the investigation of how it

is enacted while teams collaborate. The different meanings autonomy took in the studies allow us to provide a definition for it, although it remains a contextualized concept. By extending Autonomy to be aligned it appears that software teams and organizations can effectively use to drive their collaborative efforts.

In a narrow scope autonomy was conceptualized as *independence*, to mean that the work of individuals or teams is de-coupled so that the need for them to coordinate is minimized or eliminated. Such was the case with teams in the GitHub study using the version control system and a branch-and-pull workflow to isolate themselves during development and coordinate when incoming pull requests signalled it was necessary. There seemed to be more to autonomy than the removal of dependency, however, or the clear signalling of coordination needs. Autonomy included the freedom to select the tasks to work on, and so it seemed to be better conceptualized as the more elaborate concept of *self-organization*, to include decision making freedom.

In the Atlassian case teams were given ample freedom to make their own decisions in organizing their work. They were still de-coupled from others as much as possible but complete independence was not the goal. What the organization tried to accomplish through its work practices was to combine the autonomy (self-organization in this case) of various entities with alignment to the organizational strategy. The reasoning was that by having the objectives of the organization in mind, the autonomous effort is channeled in strategically the right direction. Through a shared culture of transparency and involvement, Atlassian followed a model of *Aligned Autonomy*, matching the autonomy provided to employees with the responsibility to achieve strategic goals.

In the Microsoft case the themes of autonomy and alignment surfaced again, this time as facilitated by the engineering manager. As part of their role, engineering managers cultivate engineering wisdom and help developers learn how to use the freedom they are afforded. At the same time they act as the developers' lifeline to the organization, communicating and explaining the strategic goals. By placing the individual or team work within the organization's objectives the autonomy enabled by the engineering manager acts as *empowerment* for developers to be responsible and successful.

As I mentioned in Chapter 2 autonomy comes by different names in the literature. My experience was similar in the studies I conducted where autonomy took the form of independence, self-organization, Aligned Autonomy, and empowerment, with different characteristics being more emphasized in each case. Inspired by the definition of

autonomy by Janz et al. [107], and the emergent meaning of autonomy in the studies, the most fitting definition of autonomy seems to be:

> **Autonomy** is the freedom and responsibility of an entity to determine the actions required to fulfill an intended purpose, and how best to execute them.

As an extension of Autonomy's definition, the most fitting definition of Aligned Autonomy seems to be:

> **Aligned Autonomy** is the freedom and responsibility of entities to determine the actions required to fulfill a common goal, and how best to execute them.

7.2 How synthesis works

As I mentioned in Chapter 3, the findings of qualitative studies can be synthesized inductively, through analogical inference [136]. Although there are fine distinctions between analogical and inductive inference on low-level details, those are out of scope for the thesis to discuss, and the two models of thinking are regarded as largely similar [200]. Thornton describes both analogical and inductive inference as trying to generate relatively general knowledge from relatively specific knowledge.

An approach to generating insights from the findings of multiple studies, that builds on inference, is evidence synthesis [25]. *Interpretive* evidence synthesis is used in systematic literature reviews to synthesize the findings of primary studies [58], usually through meta-ethnography. Meta-ethnography is an interpretive approach originally developed for combining the findings of ethnographic research conducted in the field of education. This synthesis method has the potential to provide a higher level of analysis, and some authors suggest that the strength of this approach lies in its attempt to preserve the interpretive properties of primary data [59]. In this chapter I use interpretive evidence synthesis through meta-ethnography to link the findings within and across the case studies. According to Dixon-Woods et al. [58] meta-ethnography uses three strategies:

- Reciprocal Translational Analysis (RTA), whereas there are identified links and similarities between the key concepts of the cases, thus translating them across cases

- Refutational Analysis (RA), whereas there are identified differences between the key concepts in the studies that are discussed
- Lines of Argument Synthesis (LAS), whereas a picture of the whole is built from studies of its parts.

In the remainder of the chapter I am using the above strategies on the key concepts of the studies, and I build a conceptual framework of *collaboration via aligned autonomy* as the result.

7.3 Key concepts in each study

Below I summarize the key concepts from each case study in table format. These concepts are high-level, abstracted from the case studies. I refer to the three studies as "the GitHub study", "the Atlassian study", and "the Microsoft study".

7.3.1 The GitHub study

The study described in Chapter 4, focused on how commercial software teams use GitHub. It uncovered the ***interplay of autonomy and collaboration*** and found that a tool that supports decentralized development — such as GitHub — can still support all the elements of collaboration in software teams. Table 7.1 includes all the key concepts in the study: the concept of collaboration, and its operationalization through the elements that make it up.

7.3.2 The Atlassian study

The study described in Chapter 5, focused on the ***work practices that can help an organization scale autonomy beyond a few people and teams, while still effectively pursue its organizational goals***. It uncovered the interplay of autonomy and scale, and found that a culture of shared ownership and involvement — such as Aligned Autonomy — can address inefficiencies, and prevent autonomy from becoming ineffective. Table 7.2 includes all the key concepts in the study. To avoid repetition, I have not included the concepts from Chapter 4, although they reappeared when I was studying Atlassian.

Table 7.1: The key concepts from the case study of how commercial software teams use GitHub, featured in Chapter 4.

Concept	Description	Desired state
Collaboration	practices and processes a team uses to produce a collective outcome; they address the collaboration elements (see below)	smooth collaboration: achieved objectives; adhered to goals and plans; avoiding overhead and rework
Coordination signals	actions to keep the team members synchronized in their efforts	lean coordination: balance between investing too much effort on achieving synchronization, and not having enough joint effort
Code-centric communication	the provision and exchange of information to aid the collaborative effort	targeted communication: lightweight communication attached to artifacts, supporting coordination, without creating overhead
Awareness through transparency	a state of having and using information about interdependencies, and the status of ongoing work	balanced awareness: effective monitoring of status, progress, and interdependencies, without experiencing information overload
Self-organization	the collective definition of tasks and voluntary self-assignment by team members, to achieve the shared goal	optimized division through self-assignment of tasks
Testing and deployment automation	identifying and fixing bugs in code or interfaces	supported conflict resolution: automated testing and deployment as part of the development workflow

Reflecting on the key concepts in the Atlassian study, it seems that they fall under two categories, *scaling strategies*, and *cultural values*. I elaborate on this when I describe the synthesis in Section 7.4.

7.3.3 The Microsoft study

The study described in Chapter 6, focused on ***what the role and attributes of engineering managers shape up to be in environments where development teams are empowered to determine their own work***. It uncovered the managerial implications of the presence of autonomy, and found that a manager in this context cultivates wisdom, motivates the team, and mediates its presence in the organization. Table 7.3 includes all the key concepts in the study. As previously, I have not included the concepts that I have already listed in other cases, even though they reappeared. I am discussing the similarities between cases in Section 7.4.1.

7.4 Interpretive evidence synthesis through meta-ethnography

Throughout the chapters I have discussed the key concepts summarized in Tables 7.1, 7.2 and 7.3. I also presented specific practices and strategies that the participants are

Table 7.2: The key concepts from the case study of how Atlassian scales autonomy through culture, featured in Chapter 5.

Concept	Description	Desired state
Scale	the growth and expansion of the organization in terms of the number of employees and the size and/or number of its products	apply the same principles to a higher number of people or on more complex operations without causing inefficiencies
Agility	the organization's ability to quickly and effectively respond to changes in the market, or its customers; on development terms, the use of agile development practices that emphasize developers' autonomy	have lightweight practices and a flat structure that allow the development team and/or organization to quickly change course when needed
Autonomy	the freedom and responsibility of an entity to determine the actions required for a certain task or goal, and how to best execute them	individuals and teams have the freedom to make decisions on how to organize their work to achieve goals
Independence	the lack of dependencies between entities and, hence, the lack of need for them to coordinate their actions	separation of concerns between individuals and teams [note: the organization realizes that independence can lead to isolation, and defaults to autonomy instead]
Decentralized decision making	the empowerment of entities to make decisions without the need to obtain approval from a central authority	apply decentralized decision making when possible, leading to flatter organizational structure
Clarity of purpose	the communication of organizational goals as well as the rationale and intent behind them; entities are given the opportunity to ask questions to obtain clarity	broadcasting of organizational goals and discussion on them; cascade and "translation" of goals to each subsequent organizational level
Transparency	access to information by everyone in the organization	open access to repositories, communication channels, documentation, internal systems (barring security restrictions)
Shared learning	the exchange of experiences, insights, and lessons learned between individuals and teams	impromptu and scheduled opportunities to share knowledge and experience, ideally though forums that can be archived and accessible
Shared ownership	the distribution of responsibility of various organizational operations to all rather than few	responsibilities are distributed in such a way that handoffs — and the potential resulting delays — are avoided whenever possible
Shared responsibility	the sense of having personal interest in the success of a process, mirrored in the tasks that one takes on	all employees feel that the achievement of organizational goals is everyone's main driver and are all involved and invested in succeeding
Culture	a set of values that the organization believes in and practices day-to-day	the values drive the desired behaviour in the daily life of the organization
Aligned Autonomy	the combination of high levels of freedom to the individuals and teams to make decisions (Autonomy) toward achieving organizational goals (Alignment)	clarity of purpose sets the organizational goals as the boundaries within which entities can be autonomous
Involvement	the ability of individuals and teams to provide their opinion and feedback on strategic decisions and organizational goals	variety of ways for employees to comment on strategic direction, ask questions, propose ideas
Articulated guidelines	explicitly expressed guidelines about communication, or non-creative processes (e.g. reporting), that can benefit from clarity and general acceptance	documentation for coding standards, specifications design guidelines, code of conduct, templates

using regarding the concepts; for brevity, I have not included them in the tables. The steps of synthesis which I describe in 7.4.1 and 7.4.2 (in line with the meta-ethnography method) are meant to allow the researcher to reflect on the findings of the studies *in tandem*. In that sense, the researcher can identify conceptual links within and across cases that were not apparent when strictly presenting the findings of each research question, or each case in isolation.

7.4.1 Reciprocal translational analysis

Reciprocal translational analysis discusses the studies' key concepts with three aims. One, after extracting the key concepts it takes a critical view of how they relate to the study. Two, it identifies similarities to build the common ground between

Table 7.3: The key concepts from the case study of what Microsoft managers and developers perceive are the attributes of engineering managers, in an environment where developers can work autonomously, featured in Chapter 6.

Concept	Description	Desired state
Cultivating engineering wisdom	explaining the criteria for making engineering decisions; give ownership of decisions to engineers; help develop the engineers' skills	the engineering manager creates conditions for on-the-job learning for developers and gives them growth opportunities
Motivating the team	creating a working atmosphere that energizes the team	individuals and teams feel motivated and engaged in their work
Mediating external interaction	acting as an information filter between the team and other organizational entities	balance between minimizing distractions for the team, and showing its impact to upper management
Being technical	having technical understanding of the problems, solutions, systems, and technologies the team is working with	the engineering manager can clearly explain the desired outcomes, engage the team to form a plan, and leave the implementation decisions to them

cases. Three, when applicable it "translates" concepts between studies. Due to the contextual nature of case studies, the key concepts or aspects of them may be given different names across studies, but hold the same meaning. The researcher is able to make these connections once they have analyzed all the studies, and can see the big picture. In this section I identify links between concepts, common themes and concepts across cases, and connect concepts that were discussed in a similar way.

One reflection is based on the concepts *within* the Atlassian study. In the Atlassian case, the focus was on identifying work practices that helped the organization scale autonomy beyond a few teams or departments, while still achieving its strategic goals. I recorded 30 practices used in the organization to that effect, and discussed how each enabled both Autonomy and Alignment. In Chapter 5 I discussed the cultural mindset at Atlassian and the work practices are intertwined and play a role in how growth is handled effectively. Upon reflection of the key concepts in the Atlassian case it seems that they fall under two categories, *scaling strategies* and *cultural values*. The work practices I recorded were part of the organization's plans to scale, but were also in line with the organization's mindset and beliefs. For example, providing teams with the independence to own their own deployment activities was a scaling strategy that was congruent with the organizational belief that responsibility is shared. This is in the spirit of what I described as part of the definition of autonomy: freedom and responsibility. After reflection, then, it appears that ***all the scaling strategies Atlassian employs are situated within its cultural values and align with***

them. This is represented in the framework of *collaboration via aligned autonomy* when I present and discuss it later in the chapter.

The rest of the reflections in the process of synthesis apply *across* cases. In both the GitHub and Atlassian studies ***development teams reported using similar practices, although using different tools***. All development teams in the two studies (with the exception of a single team in the GitHub study) used a workflow based on pull requests, to decentralize development; either through GitHub or BitBucket. Teams in the Microsoft study mentioned they use workflows that incorporate branches, while some (but not all) use pull requests. As part of the workflow teams in all three studies reported performing code reviews, although the tools differ or some of the rules; for example, some of the teams in the GitHub study want one reviewer per pull request, while the teams at Atlassian require two. Teams in the GitHub and Atlasian studies opt for awareness based on passive monitoring of progress and status; they used different tools to do so, but they built on the transparency of the development process and the information traces it produces (timelines, notifications etc.). Regarding communication, the teams in both the studies preferred code-centric communication in the form of comments on artifacts. When communication through comments is not enough (e.g. teams want to discuss goals rather than implementation) teams reported a variety of communication tools, that support both synchronous and asynchronous communication, and support integration with the tools used for code or issue management. In both studies automated testing and deployment are part of the workflow (usually the last step), making individuals and teams autonomous by sharing the ownership of these functions.

The teams in all three studies ***self-organize when it comes to task division***. Different tools may be used in different settings for teams to discuss, divide and enter their tasks, but the process of the team coming up with tasks collectively, and the developers self-assigning tasks over which they are given autonomy, was the dominant reported practice. The sense of autonomy and ownership over one's work (either an individual's or a team's) was discussed as having various benefits, in the perception and experience of the participants. Reported benefits were optimized task division, high motivation and engagement, and improved productivity. In the perception of the study participants (and supported by findings in software engineering [15], system design [107], and various business units [95]), the above benefits are hypothesized to contribute to higher quality in the end result.

Admittedly, the code management, issue management, and communication tools

that are employed by the development teams in the studies offer overlapping functionality. Yet, the almost identical process followed by the development teams makes us wonder if — in this case — process may matter more than tools. In other words, the results seem to indicate that *putting in place the desired collaborative development practices precedes the selection and appropriation of the development tools*. In that regard, the studies identified that **when teams are looking to enable both autonomy and collaboration in their development, they opt for a decentralized workflow that ensures both isolation and coordination through the use of signals**. What helps in this direction — and was a common element across studies — is that development teams have the opportunity to choose their own development practices and tools, either from the beginning, or in a subversive manner. In the GitHub study this manifested as developers that were *agents of change* advocating the use of GitHub, while in the Atlassian study it showed up as *cross-pollination* between teams in the practices and tools they use.

Both the Atlassian and the Microsoft studies discussed ***alignment as a major responsibility of management***. The clear communication of strategic intent and the buy in from the developer side (i.e. that developers are convinced a course of action is the right one) were described as critical factors for achieving goals. In both cases participants saw a bottom-up approach to setting strategy as a vehicle to achieving clarity of purpose and alignment with strategy. This bottom-up approach went through community-building in Atlassian's case (e.g. co-founders and executives writing blog posts about the strategic rationale, and anyone asking questions, posting comments, or giving feedback). In Microsoft's case the clarity of purpose was mostly discussed within the boundaries of teams, where developers can give their opinion on decisions, but are also asked to explicitly link their individual goals to specific organizational/strategic goals; such an exercise makes them more aware of what the strategic goals are and help achieve alignment such that the individual developer's output has impact for the organization. The clarity of purpose helps the team exercise its autonomy in defining its scope and timeline; this came up in both the Atlassian and Microsoft cases as the development team collectively deciding on its sprint backlog. In both cases there were organizational roles with the explicit responsibility to clearly communicate the strategic intent to their teams; for Atlassian it was program managers, for Microsoft it was engineering managers.

Alignment was not discussed solely in the context of connecting teams to strategy, but also creating connections between teams. As mentioned, teams had the auton-

omy to choose their own practices and tools, but there is also the chance of cross-pollination. Beyond exchanging experiences with trying different processes, however, ***teams have opportunities to synchronize with other teams to support the alignment***. This was mentioned both in the Atlassian and the Microsoft studies, although it was practiced in different ways. At Atlassian, teams organize events where they showcase their prototypes to other teams, and write blog posts about their progress. At Microsoft, the synchronization between teams is organized and facilitated by upper management.

The participants in both the Atlassian and the Microsoft studies talked about the ***efficiency with which development teams can achieve their goals, and how to protect team productivity by removing distractions***. In the Atlassian case this was discussed as "maintaining development speed at scale", recognizing that with the addition of more people coordination around activities can become cumbersome and slow down progress. Atlassian's strategy to remove potential delays was to flatten its hierarchy and distribute the ownership of those parts of the workflow that were previously handled through dedicated teams that, at scale, can prove to be bottlenecks. For example, builds, continuous integration, and deployment, are owned by each of the development teams; previously teams had to hand off their work to build and integration teams. Through this practice, Atlassian teams were given the autonomy to keep their own release schedule. Maintaining development speed at scale seems similar to what participants in the Microsoft study described as the manager's attribute to "clear the path to execution", by removing distractions and providing any resources the team needs.

A single developer or team may own their workflow, but may feel unable to extend their autonomy to saying no to incoming requests — especially when coming from upper management — or push for something they may still need from another team. In Microsoft's case the manager "facilitates inter-team communication", by filtering incoming and outgoing requests for their engineering team and exercising their role of "mediating interactions". Atlassian has opted to facilitate inter-team interaction or requests through decentralization. By providing open access to all project repositories and all information, Atlassian empowers its teams to take initiative and make the changes they can themselves; participants mentioned that if they need something fixed or changed in another product, they can simply make the change themselves and submit a pull request to the other product team.

Through the Atlassian and the Microsoft studies, we saw that ***for autonomy to***

work as a foundation of collaboration there is need for an appropriate mindset. Developers need to be in position not just to make decisions, but to make *the right decisions*; to that effect, they build on information transparency, and the clarity of purpose that drives alignment. The GitHub study, although didn't explicitly investigate this concept, corroborates that it is not strictly the use of a tool that brings the decentralization in decision making, or that makes such decision making effective. Atlassian mentors its employees in an experiential way, through its very open and transparent environment that sets the tone for what the desired behaviour is. The Atlassian study showed that cultural values promote the mindset that is appropriate for *collaboration via aligned autonomy*; this affects how the organization hires new talent, and what attributes they are looking for to consider someone a good fit. On the other hand, the Microsoft study showed that "cultivating wisdom" and mentoring are important managerial responsibilities (both were perceived as attributes that make great engineering managers), and they promote the necessary mindset.

Finally, all three cases exhibited decentralization of decision making. This bottom-up, decision making freedom was perceived to lead to the following benefits:

- **less need for running decisions by a central authority** (which usually causes delays). This was achieved through the decentralized workflow in the GitHub study; the end-to-end ownership of testing, builds, and deployment in the Atlassian study; and the facilitation provided by the engineering manager in the Microsoft study.

- **higher motivation for developers through empowerment**. This was achieved through self-organization in the GitHub study; a culture of ownership and involvement in the Atlassian study; and the creation of a positive working environment in the team in the Microsoft study.

- **increased capacity for creativity and innovation**. This was enabled by the transparency of information and achieved through support for experimentation in all three studies.

- **optimization of operational decisions**. This was achieved through self-organization in task assignment in the GitHub study; open access to repositories and information in the Atlassian study; and the cultivation of engineering wisdom by the engineering manager in the Microsoft study.

A summary of the synthesized findings is provided in Table 7.4

Table 7.4: The synthesized findings, summarized in table form

Summary of synthesized findings
An organization's scaling strategies are situated within its cultural values and align with them.
Development teams reported using similar practices, although using different tools.
Development teams self-organize when it comes to task division.
Alignment is a major responsibility of management.
Teams have opportunities to synchronize with other teams to support the alignment.
Common concern: Efficiency with which development teams can achieve their goals, and how to protect team productivity by removing distractions.
For autonomy to work as a foundation of collaboration there is need for an appropriate mindset.
Decentralization of decision making is a common form of granting autonomy.
Autonomy takes the form of independence, for small teams/organizations.

7.4.2 Refutational analysis

The goal of refutational analysis to identify the differences between studies and, if applicable, contradictions between them.

Autonomy was discussed in slightly different ways in the studies and an understanding of the distinction between the terms and their relationship is necessary. In the GitHub study autonomy was essentially equated to independence; by isolating development through the use of a pull-based workflow and feature branches, developers (and their tasks) had a degree of independence that, in turn, made them locally autonomous. In the Atlassian study autonomy was discussed as self-organization and in the Microsoft study as empowerment; in both cases autonomy represented freedom in decision making, meaning that individuals and/or teams have ownership over various aspects of their work. For this not to result in misdirected effort, autonomy was described as needing to be combined with strategies that provide alignment.

My interpretation is that ***independence for the teams in the GitHub study, is autonomy for the teams in the other two studies***. This may be a direct result of the difference in the size of the organizations between the Github study and the other two. It seems that when the organization is small, teams can afford to make themselves as independent as possible, without negative effects of isolation and misdirected effort. Probably because the organizations are small and are still using information transparency it is easy to maintain the necessary awareness that wards against the negative effects of organizational silos. For larger organizations and more complex products, however, it is almost impossible to remove dependencies between

teams, especially on the level of business goals. For example, given that the Atlassian products are meant to work together as part of a product suite, and are branded and promoted in a homogenous way by the organization, they share the same business and strategic goals, even if they don't share a codebase (although they still share interfaces and integrations). The same applies to Microsoft's products; even if one could set aside code dependencies, the products (and their teams) share strategic goals that cannot be realized unless the teams are in sync. In such a context, autonomy is still desirable since it provides the ground for creativity and efficiency, but it requires a high degree of alignment at the same time. While the GitHub study showed that autonomy can work at the local level without much need for alignment, the Atlassian and Microsoft studies put a lot of emphasis on the importance of alignment for autonomy to work effectively for the organization. **The findings indicate that alignment shows up as a need as an organization becomes larger and needs to scale its operations and activities**. In the scenario of larger organizations, one needs the teams to be aware and feel dependent on others on the strategy front, and factor that in their autonomous decisions. Hopefully, the realization of this distinction can help clarify what the optimal state is for small, medium, and large organizations, and move them toward establishing the right processes that tailor *collaboration via aligned autonomy* for them.

While alignment was a common concern in the Atlassian study and the Microsoft study, each organization approached it differently. Atlassian built on open communication, transparency, blogs, ad-hoc discussions and feedback on decisions etc. to create a shared understanding about the goals of the organization and allow individuals and teams to see where their work fits with the intended outcome. At Microsoft, the engineering manager was the main actor for driving alignment. By discussing with developers, either individually or in the team, providing and explaining business insights, the engineering manager is essentially a lifeline between the development team and the organizational strategy, actively linking the goals of the two to keep them aligned.

The GitHub study showed that the use of transparency can help lower communication and coordination needs for development teams. However, the Atlassian study identified that transparency and open communication at scale have limitations. These two findings highlight a **tradeoff between open communication and information overload**, in line with recent work that has identified the same challenge in the use of social and communication channels [193]. It seems that as an organization be-

comes larger, the *curation* of information becomes essential to maintaining important knowledge without its volume becoming overwhelming.

A final distinction between the cases is the challenge that developers mentioned in the GitHub study; working with non-technical users. In that case, developers reported that non-technical users were reluctant to use GitHub as they perceived it as complicated and too sophisticated for what they wanted to accomplish. The developers found that they needed to compromise and use additional tools that the non-technical users were already familiar with; these usually were task management tools. This challenge did not surface in the Atlassian or Microsoft studies. This could be because both Atlassian and Microsoft offer other tools as part of their suite that mitigate the challenge that the teams in the GitHub study reported. Especially Atlassian targets non-technical teams as a market segment, and their tools have characteristics that account for this use case. Recently, Atlassian acquired Trello [1], which was one of the task management tools that the teams in the GitHub study reported using when collaborating with non-technical people.

7.5 A framework of *Collaboration via Aligned Autonomy*

In this section I combine the key concepts of the studies in the form of a conceptual framework. The construction of the framework builds on the synthesis I presented in the previous section and serves the purpose of the *lines of argument synthesis*. Here I explain what the framework consists of, what the placement of the items on the framework means, and how it should be interpreted.

7.5.1 Framework elements: *Areas* and *Approaches*

The framework of *collaboration via aligned autonomy* is meant to represent what an organization will need to consider, in order to combine collaboration and autonomy effectively. Through the synthesis I have abstracted the key concepts of the cases (listed in Tables 7.1, 7.2, and 7.3) into four *areas*, described below. They are: ***team collaboration practices***, ***scaling strategies***, ***cultural values***, and ***manager roles***. Each of the *areas* contains the necessary *approaches*, based on the empirical

[1] `http://blog.trello.com/trello-atlassian`

evidence.

Team collaboration practices

The **team collaboration practices** were first discussed in Chapter 4. The team collaboration practices were supported by GitHub features, but also extended to the use of supplementary tools, and practices that covered all *collaboration elements*. Regarding communication, the approach was for teams to use **code-centric communication**, which took the form of comments on issues, commits, and pull requests. Other forms of communication included chat clients that integrated with GitHub to allow for displaying notifications. Regarding coordination, the teams' approach was to use pull requests as **coordination signals**. This was an affordance of the pull-based workflow; an incoming pull request triggered the coordination of team members to perform a code review and merge changes to the codebase. Regarding awareness, the approach was to achieve **awareness through transparency**. The traces of activity and progress on GitHub were transparently displayed to the team in various forms: notifications, timeline, and labels. Regarding task division, the teams took the approach of **self-organization**: developers would assign tasks to themselves based on their expertise and availability, after (in consultation with team leads or managers) there was an agreed-on list of issues on GitHub, or other forms of tasks in an integrated task management tool. Finally, regarding conflict resolution, most of the teams opted for **testing and deployment automation** through continuous builds and integration. This was made possible through service hooks to continuous integration tools. The ***team collaboration practices*** area and its included approaches are summarized in Table 7.5 below.

Table 7.5: The ***team collaboration practices*** area of the framework of *collaboration via aligned autonomy*, and its approaches

Team collaboration practices
Code-centric communication
Coordination signals
Awareness through transparency
Self-organization
Testing and deployment automation

Scaling strategies

The **scaling strategies** were discussed in Chapter 5. As part of its **scaling strategies**, Atlassian follows an *approach* of providing **independence**. This manifested as the removal of handoffs of responsibility (e.g. builds and deployment activities). To support the autonomy and independence in a meaningful way, the organization moved toward approaches such as **articulated guidelines** and **shared learning**, so that autonomous teams and individuals still have the opportunity to act in an aligned way, for the organization to achieve its direction through the autonomous efforts of its constituents. The ***scaling strategies*** area and its included approaches are summarized in Table 7.6 below.

As mentioned before, the scaling strategies followed by Atlassian were in the context of its more general cultural values.

Table 7.6: The ***scaling strategies*** area of the framework of *collaboration via aligned autonomy*, and its approaches

Scaling strategies
Independence
Articulated guidelines
Shared learning

Cultural values

The **cultural values** were discussed in Chapter 5. One of the main values was that of **transparency**. The organization has distilled this in the memorable organizational value of "open company, no bullshit", and it is practiced as open communication, and open collaboration throughout the organization. Transparency, thus, extends to communication channels, business strategy rationale, and repositories of information and/or code. Closely associated with transparency is the cultural value of **involvement**. The organizational value that best captures involvement at Atlassian is "be the change you seek"; employees are encouraged and empowered (especially through the transparency and open access) to make autonomous decisions, take initiative, and provide feedback. The cultural value of **shared responsibility** extends involvement to a sense of having a personal interest in the success of the process, mirrored in the tasks and initiatives someone takes on. **Shared ownership** corresponds to a sense of alignment: each person's or team's decisions empower them to be responsible for the

success of the organization as a whole. The ***cultural values*** area and its included approaches are summarized in Table 7.7 below.

Table 7.7: The ***cultural values*** area of the framework of *collaboration via aligned autonomy*, and its approaches

Cultural values
Transparency
Involvement
Shared responsibility
Shared ownership

Manager roles

The **manager roles** were discussed in Chapter 6. The roles (and the corresponding attributes) characterize great managers, in environments were developers and teams are provided with autonomy. The approach of **cultivating engineering wisdom** is one where the engineering manager teaches developers and teams how to exercise their autonomy and how to develop the appropriate decision making skills; this was done by giving them room to experiment and grow their talents, and by building a team culture. Through the approach of **motivating the team** showed the social side of the engineering manager; specific attributes related to promoting fairness and building a relationship with team members, and maintaining a positive environment for everyone in the team. By **mediating external interactions** the engineering manager acts as a conduit between the engineers and the organization: they clear the path to execution for individual developers, but also keep the whole team aligned (in purpose and activities) with what is the desired direction for the organization. Finally, the engineering manager needs to **be technical** enough to provide guidance and feedback, but not too much so that they don't undermine the individuals' or the teams' autonomy by making the engineering decisions. The ***manager roles*** area and its included approaches are summarized in Table 7.8 below.

7.5.2 Framework diagram

A visual representation of the *collaboration via aligned autonomy* framework is shown in Figure 7.1. The four *areas* are represented by different colors: red for **team collaboration practices**, green for **scaling practices**, blue for **cultural values**,

Table 7.8: The ***manager roles*** area of the framework of *collaboration via aligned autonomy*, and its approaches

Manager roles
Cultivate engineering wisdom
Motivate the team
Mediate external interactions
Be technical

and yellow for **manager roles**. To not overcrowd the framework I have not included the *approaches*, but the *areas* are labeled and their contents have been summarized in Tables 7.5, 7.6, 7.7 and 7.8.

The four *areas* overlap, and there is a layer of *core concepts* that are central to all areas, shown in the middle part of the framework. This surfaced in the process of combining the four *areas* and taking a retrospective view. By this I mean that while some of the findings surfaced in a specific study, they also allow retrospective interpretations in other studies too. In the following section I focus on discussing the overlap and the three *core concepts*.

7.5.3 Retrospective thinking: *Overlap* and *Core concepts*

Given the overall theme in the research in this dissertation, it is not surprising to find that, in retrospect, insights that appear later have relevance for earlier insights. In this section, I am focusing my retrospective thinking on three central concepts of the *collaboration via aligned autonomy* framework, illustrated by the three red, centrally-placed items in Figure 7.1.

Decentralized decision making is a team collaboration practice that we saw in the GitHub study, and it functioned in concert with the workflow, and all other practices surrounding the collaboration elements. It remains unknown if there is a cause-effect relationship, or what its direction is, between having *decentralized decision making* and choosing the particular approaches in the **team collaboration practices** area. Is it that a team desires *decentralized decision making* and that leads to particular selection of tools and practices? Or are the chosen practices what leads to a *decentralized form of decision making* for the team? The Atlassian case corroborated that the practices are not GitHub-specific, but they are *at least* compatible with what the tool affords.

While *decentralized decision making* surfaced as a team collaboration practice,

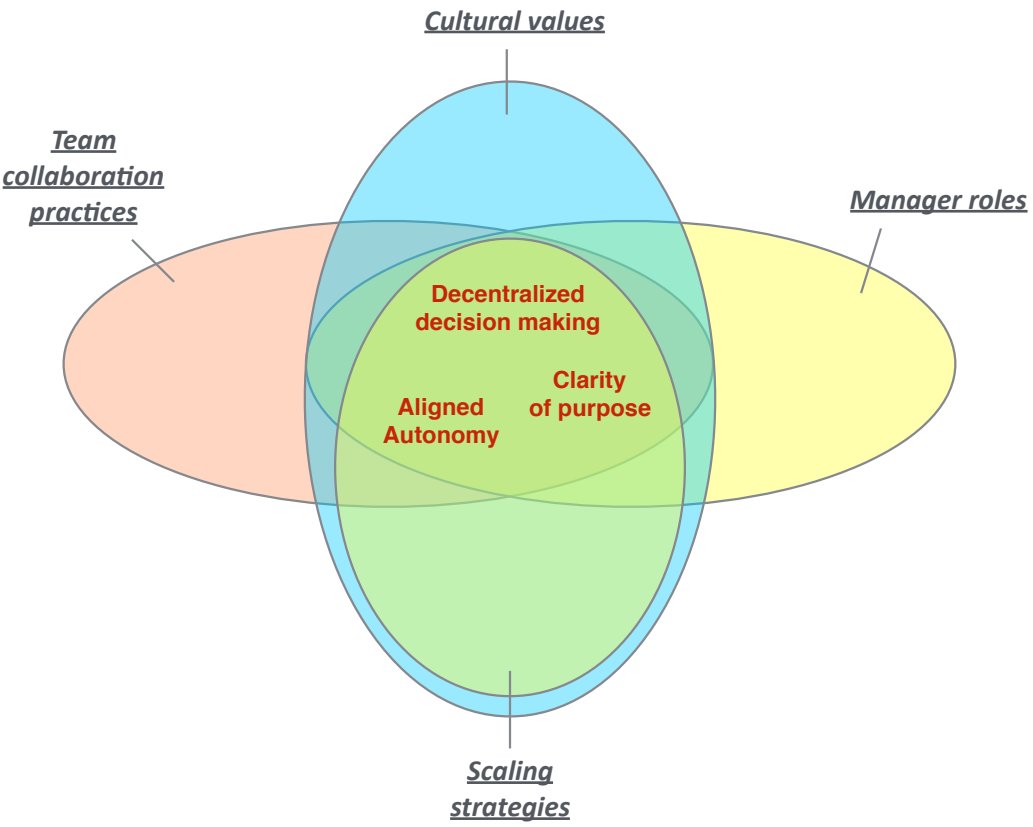

Figure 7.1: The proposed framework of *collaboration via aligned autonomy*.

it was also an overarching scaling strategy for Atlassian, to counteract some of the inefficiencies of growth. *Decentralized decision making* was part of Atlassian's culture, instantiated by approaches such as *shared ownership*, and *independence*. Finally, we saw that *decentralized decision making* was facilitated by the manager's role of *cultivating engineering wisdom* through enabling engineers to act autonomously and involving them in decision making.

Aligned Autonomy is both a cultural value and a scaling strategy that surfaced in the Atlassian study. It is also facilitated by some of the roles of the engineering manager, and more specifically by the combination of the attributes of *enabling autonomy* and *driving alignment*, that surfaced as part of the *cultivating engineering wisdom* and *mediating external interactions* roles respectively. In retrospect, this is also something that was present in the GitHub study, although it was perhaps hidden. The teams in the GitHub study were using GitHub and their team collaboration practices to remain both autonomous and aligned. Autonomy was afforded as inde-

pendence (I brought this up in Section 7.4.1) through developers operating inside individual branches, while alignment was made possible through code reviews and the passive monitoring of GitHub's transparent display of information.

Clarity of purpose is a cultural value and a scaling strategy that emerged in the Atlassian study. As Bungay tells us [28], *clarity of purpose* is a necessary element for *aligned autonomy* and *decentralized decision making* to work effectively: the autonomous entities need to have a common understanding of the boundaries within which they are autonomous, and what is the desired direction to channel their effort toward. This is something that was highlighted in the discussion of the *driving alignment* attribute in the Microsoft study, when an engineering manager *mediates external interactions*. Accounts from both engineers and managers mentioned that *driving alignment* hinges on knowing what the purpose and intent is, and the opportunity for the developers to be involved in the articulation of the purpose. In retrospect, Atlassian's open discussion and feedback of decisions enabled alignment by clarifying the purpose. In the GitHub study, clarity of purpose can be seen as an underlying team collaboration practice that is supported by the clarity of *code-centric communication* (e.g. the team discussing through issue comments and gradually achieving common ground) and *awareness through transparency* (e.g. discussions being persistent and act as shared memory).

Given that the three *core concepts — Decentralized decision making*, *Aligned Autonomy*, and *Clarity of purpose* — underline all three cases, they take centre stage in the framework. The importance of the core concepts is also highlighted by how they support all the *areas* of the framework. The findings from the studies indicate that **teams and organizations that wish to achieve *collaboration via aligned autonomy*, should look to establish *Decentralized decision making*, operate in a state of *Aligned Autonomy*, and build on *Clarity of Purpose* to direct their efforts.**

A further point to be made in retrospect relates to the role of transparency, in *collaboration via aligned autonomy*. In the GitHub study, teams used GitHub's transparent display of all project, artifact, and activity information to maintain their awareness, while working independently. This way of using transparency has been identified in earlier studies of GitHub; for example, Dabbish et al. [52], made similar observations for open source projects. For the commercial teams I investigated, transparency also served as a way to minimize potential communication and coordination overhead, by making it easy to find information without the need to ask for it.

Indeed, participants called issue lists, commit lists and notifications their "first line of defence" against communication overhead.

The central role of transparency in the Atlassian study was made clear by various means. First, transparency was a cultural value that manifested as the practice of open communication, the ability to openly comment on every artifact and decision, and the *by default* public (within the organization) nature of all generated content. **Transparency supported autonomy, by ensuring that people have the information they need to inform their decisions**. It also **ensured alignment by allowing people to see — and be part of — the big picture of organizational strategy, and see how their work fits within it**. Finally, transparency helped to solicit *meaningful* feedback; by having all the information, the individuals' and teams' involvement has better chances of being well-scoped and useful.

In the Microsoft study, transparency was still important, although it manifested in a different way. Microsoft does not employ the same open communication mechanisms or the radical transparency of information and content that Atlassian does. Yet, as part of *driving alignment* and *enabling autonomy* (attributes within the roles of engineering managers described in Chapter 6) developers and managers said that managers share information with their team openly about the strategy of the organization and what it means for the team's work. "Openly" in this case meant that managers would share discussions they had with upper management and strategic decisions that were in the making, and would involve developers in what should be the shape of the strategy. Even though the information may not be displayed to other teams or in central, open-access repositories (as done at Atlassian), it was still shared within the team in a transparent way to facilitate decision making.

It appears, then, that *Transparency* forms some of the ground *collaboration via aligned autonomy* stands on, and supports the other *areas*. I, therefore, have included *Transparency* as a fourth core concept in the framework, since it seems to transcend **cultural values**. In this sense, **to achieve *collaboration via aligned autonomy* teams and organizations need to employ transparency of operational and strategic information, as part of their team collaboration practices, scaling strategies, cultural values, and manager roles**.

Figure 7.2 shows the finalized version of the proposed framework. This reflects the retrospection (***Transparency*** has been moved to the centre of the framework, with the other *core concepts*), and includes all the *approaches* that make up the four *areas*.

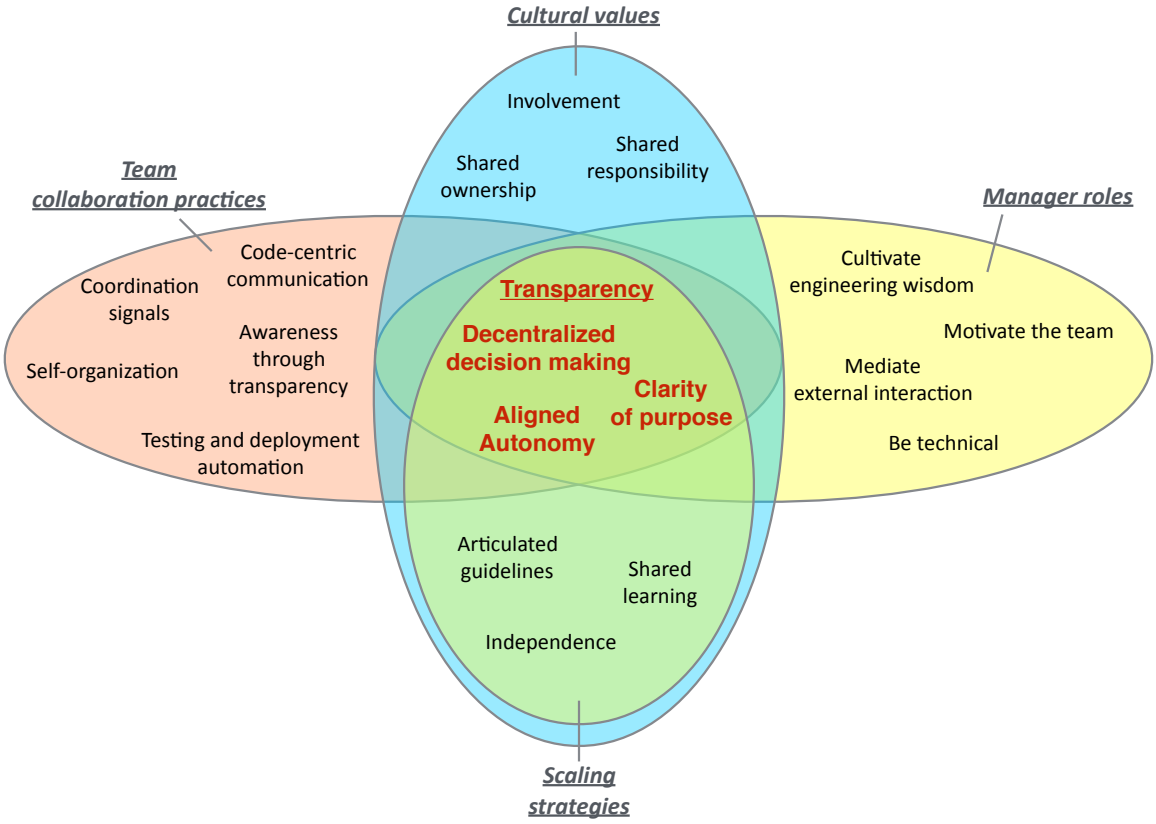

Figure 7.2: The final version of the proposed framework of *collaboration via aligned autonomy*. ***Transparency*** appeared as a core concept after the retrospection, and so is underlined in the diagram.

The *collaboration via aligned autonomy* framework highlights the *areas* that a modern software organization can influence to combine collaboration with autonomy, and how to *approach* them. It represents a view that is holistic, and takes into account the execution, strategic and managerial levels of an organization. Software companies often try to improve individual areas: they may adopt new collaboration practices, try new tools, or a different management style. Through the *collaboration via aligned autonomy* framework **I propose that organizations should consider all four *areas* instead of optimizing for one, and strive to infuse the four *core concepts* into their practice as much as possible**. I discuss the implications of the framework in the following section.

7.6 Implications of the framework

There are a number of implications that stem from the work in this thesis, for both theory and practice. I describe them below, relating them to existing work, where applicable.

Implications for theory

The work I presented in this thesis can ***add to several different bodies of the literature in software engineering research***.

First, the contents of the GitHub study add to the literature that investigates the *adoption of* OSS*-style practices in commercial software teams and organizations*, a need identified previously [191]. Chapter 4 presented evidence of a variety of practices that, supported by GitHub's features, resemble practices that have been identified in earlier work investigating the use of GitHub's social features [52] and technical features [84] in open source software projects. Secondly, the contents of the Atlassian study can add to the literature investigating the *adoption of agile practices in industry, and how agile practices scale*. Although there has been a growing number of publications focusing on the use of agile development methods in industry, there is an identified need for more case studies [106], and a focus on the ways agile practices can be adjusted to be of benefit to large organizations [57] or projects [38]. Finally, the contents of the Microsoft study can add to the literature investigating *developer productivity and motivation*. While there is no documented gap yet in this research area, I argue that given the evidence from other domains about the impact managers have on productivity and motivation, software engineering research should consider the role of engineering managers and how they affect, even if in a second-order manner, software engineering outcomes. The findings of Chapter 6 bring evidence that engineering managers can have a positive impact on what an engineering team produces and how effectively.

Throughout the dissertation I have built on concepts that have originated or are discussed in other domains, besides software engineering. Neither *collaboration* nor *autonomy* are concepts that are particular to software engineering, and both have been extensively discussed in other disciplines. That software engineering deals with producing technical work, means that there is need for *appropriation* of the concepts and principles other domains have found relevant, but it does not make software engineering completely unique. The way I have discussed the literature and empirical

evidence in this thesis, can bring awareness of concepts such as *empowerement*, *responsible autonomy*, *strategic fit*, and how relevant they are in the context of software engineering. With the growth of the software engineering industry, software development is becoming a central activity in many organizations' operations. For example, it is difficult to categorize companies like Amazon as retailers or software companies anymore. Practitioners have recently started to view software engineering as part of the organization (as opposed to a supporting activity), and realize the benefits of a holistic approach that combines engineering, business, and culture. While that is true of practice, however, it is only mirrored in software engineering research in areas such as Human-Computer Interaction or Computer-Supported Cooperative Work (which are usually considered to be on the fringe of software engineering). By framing the current thesis in an inter-disciplinary way my aim has been to ***raise awareness of the relevance of organizational factors within the context of software development practices***.

The main outcome of the thesis is the *collaboration via aligned autonomy* conceptual framework. The framework has implications for theory in that it proposes a view that extends a currently popular one, that builds on a combination of independence and coordination. Work on collaborative software development in recent years has followed the premise in Conway's law [39] that an organization's structure and the structure of the system it builds mirror each other. This has had tremendous impact on research looking to model the collaborative software engineering activity. Studies have focused on how to apply Conway's law to modularize software and minimize dependencies between individuals and/or teams [100], or how to define and distribute tasks between entities [118]. Yet, dependencies can only be minimized so much before there are risks of isolation that can be detrimental to the success of a software project [188], or an organization [205].

The *collaboration via aligned autonomy* framework extends this approach, while still respecting the principle of the software system/architecture and the organizational structure mirroring each other. In *collaboration via aligned autonomy* the modular teams and organizations are autonomous, but also aligned; this is accomplished by practices that address how teams collaborate, but also how they remain aware of the organizational direction through cultural values, and with help from the manager. The focus then is not on eliminating dependency, but on making teams aware of how they build on each other to enable them to influence the collective outcome. Although some earlier work argues that developers don't see their work in that

perspective [118], there is evidence that suggests otherwise. Literature on knowledge work and psychology, indicates that developers want to be *involved* and gain a sense of purpose [146] that they bring into their work as high motivation. The findings I presented in this thesis corroborate and build on this view. Independence is still part of the framework and it surfaced as especially important in small organizations, perhaps because scale has not interfered with how aware the developers are of the desired outcomes on both the product development and the business side. Therefore, this thesis proposes ***a framework for collaborative software development that adds areas to be considered when investigating software engineering activities in teams and organizations***.

Finally, the recorded cases and practices that I presented add to the knowledge repository of how software engineering is performed in practice. Previous work calls for more studies of software practices in industry [191], occasionally pointing out that software engineering research is lagging behind practice in coming up with recommendations [73]. Therefore, the thesis ***adds to the body of cases of engineering practices in industry***, highlighting contextual factors behind them and linking them to theory.

Implications for practice

The framework of *collaboration via aligned autonomy* has potential implications for practitioners. While the framework is meant to be used in its entirety and as a holistic approach, it ***can guide interventions that teams and organizations may wish to make incrementally in their development practices, scaling strategies, organizational culture, or managerial roles***. Recent research advocates for holistic approaches for continuous software engineering [75], and in that context the *collaboration via aligned autonomy* framework shows some ways to facilitate *BizDev* and integrate software development activities with business goals. The framework, complemented by the practices reported throughout the dissertation, can work on different levels of abstraction. On a high-level, the framework (as shown in Figure 7.2) highlights approaches that relate to the four colour-coded *areas*; the specific implementations of the approach can be left to the particular team or organization. For example, *shared learning* can be achieved through a variety of practices, not necessarily restricted to arranging brown-bag sessions, which was one of the practices I reported. On a low-level of abstraction, the case studies contain a variety

of practices that teams and organizations can consider to adopt. By shifting gears between the levels of abstraction, I have aimed to not be too prescriptive in the practices but to be descriptive enough in the principles that teams and organizations can design their own interventions, appropriating the approaches to fit their particular characteristics. Practitioners are looking for guidance on what practices to follow regarding collaboration, and the framework I propose provides food for thought.

A related implication is that this thesis — through the framework — proposes that software organizations should ***strive to have as much overlap as possible between the four areas***. The claim may sound bold but the aim behind it is to bring to the practitioners' attention that there may be more factors to consider than what may seem immediately relevant. Most often software companies tend to optimize locally: they adopt best development practices, or change their management style, or try a particular scaling practice. Based on the framework and the evidence in the thesis, it seems that interventions toward *collaboration via aligned autonomy* require attention to all four *areas*: team collaboration practices, scaling strategies, cultural values, and manager roles. For example, if a company is making decisions about how their development teams collaborate, they will have to consider how the decisions align with cultural values, scaling strategies, and manager roles.

Out of the four areas, software companies usually focus on teams and their managers, and then build on scaling strategies. It is often not obvious for a software organization that culture is also a primary concern, and should be included in how collaboration is set up, not as an afterthought. Throughout the thesis ***culture surfaced as a major component of collaboration via aligned autonomy, highlighting that it should be a concern early on***. A resulting recommendation, therefore, would be for organizations to create a culture that fosters autonomy and alignment; by forging the culture early on the latter two elements can "fall into place". The Atlassian case emphasized that by showing how the cultural values of the organization were set at the launch of the company and still drive its growth and work practices. At Microsoft the cultural values were not emphasized as early on. Yet, with the latest change in leadership the organization has shifted its culture and has now made it a major component of its organizational life.

The findings in Chapter 4 have implications for *how teams can select tools to use in their collaborative development, and how to complement them with other practices.* The variety of practices covers coding activities, but also communication and awareness. For example, teams using wikis as a communication tool may want to consider

integrations with their coding tools that will allow the seamless display of information on the status of work. Or, teams that are currently using GitHub (or planning to) may want to adopt practices of code-centric communication through that platform, to minimize coordination overhead.

The findings in Chapter 5 have implications for *organizational structure and culture*, as well as *hiring practices*. Regarding structure, building teams in a way they have permeable boundaries to support open collaboration was one of the approaches highlighted in the findings; this can be achieved in a variety of ways beyond the ones I described. Organizations that are looking to scale their practices of autonomy may want to consider adding feedback loops wherever possible. Also, efficiency at scale may be served well by removing layers in the hierarchy or handoffs of responsibility in the workflow. Regarding hiring practices, it seems that they should ***aim for cultural fit as well as technical skills***. It is not uncommon for software companies to mistake engineering culture (e.g. agile) for organizational culture and, hence, focus their hiring process solely on assessing the candidates' engineering skills. However, a growth in the number of employees means the influx of new cultural fragments, and unless fit is assessed, the nurturing of culture may be jeopardized (this was a concern of interviewees in the Atlassian study) and result in misalignment.

The findings in Chapter 6 have implications for *selecting and training engineering managers*. The finding that a great manager is seen as tending to social and motivational elements in the engineering teams that have autonomy highlights what skills organizations should be looking for or cultivating in their managers. At the same time, the findings can be used to inform managers themselves — both new and existing — about the attributes that matter to developers, to help them achieve results.

The findings of the studies taken together have the potential to ***introduce best collaboration practices in engineering teams, empower developers, and help managers generate better outcomes though their teams***. As practitioners are looking for guidance on these matters [2] [3] [4], this thesis potentially makes a timely and significant contribution by rigorously establishing what is relevant for software teams, organizations, and managers, that wish to combine collaboration and autonomy effectively.

[2] `https://hbr.org/2016/10/leaders-need-different-skills-to-thrive-in-tech`
[3] `https://hbr.org/2016/05/how-your-leadership-has-to-change-as-your-startup-scales`
[4] `https://www.atlassian.com/blog/git/the-essence-of-branch-based-workflows`

The implications I have described in this section highlight what are the potential contributions of the thesis. I elaborate on how these implications create opportunities for further research in Chapter 8.

7.7 Research validation

Strauss and Corbin [40], recognize three criteria to validate the quality and credibility of qualitative studies: *fit, usefulness*, and *sensitivity*. *Fit* represents whether results resonate with the audience for whom the research was intended. *Usefulness* represents whether the findings offer new insights about the investigated phenomenon. Finally, *sensitivity* represents whether data and findings were derived from research questions, or research questions were posed inspired by the analysis of collected data. Below I describe the strategies (suggested by Creswell [43]) I used to apply the three validation criteria.

Triangulation. Triangulation is meant to assess a study's *sensitivity*. I used multiple sources of data in each of the case studies to confirm the results and build a coherent outcome. Whenever possible data were triangulated during the analysis to assure the accuracy of the conclusions. In this sense, interviews were used to confirm survey results and vice versa, observations were used to confirm data collected through documentation review, and so on. This allowed me to identify minor discrepancies, but also to build context around the findings and create a better understanding.

Member checking. Member checking is meant to assess a study's *fit* and *usefulness*. I contacted the study participants, *within pragmatic boundaries*, to ask for their feedback, and ensure that my interpretations of the data make sense from their perspective. To that end I sent follow up emails to the interviewees from the Microsoft study, and sent a report to the interviewees from the Atlassian study. For the GitHub study I incrementally shared study findings with participants (after saturation) at the end of the interview. The member checking activities supported the findings by confirming that they accurately represent the reality of the participants, and that they are useful to them. On a few occasions the member checking interviews resulted in minor changes and adjustments; for example, I changed the names of two of the manager attributes in the Microsoft to improve clarity.

Dissemination to confirm accuracy and usefulness. This activity is closely related to the member checking strategy, and is meant to assess the *fit* and *usefulness* of a study. In each of the three case studies I shared my findings with the

participants in some written form (beyond member checking), either by preparing custom reports (Atlassian study) or sharing publication drafts (Microsoft study), or blog posts (GitHub study). The reports were always shared with the participants, asking for them to bring up any issues or inaccuracies, and give an indication of whether the results are helpful. The feedback I received indicated that the study findings resonated with the participants. In some cases the reactions were quite enthusiastic; for example for the Microsoft study the participants were requesting to share the results with their teams and were looking forward to using the insights from the study. In other cases the reaction was more reserved; for example for the Atlassian study, while I received feedback that the results and insights are accurate, the participants felt that they had already clarity about the way they worked and the associated benefits. Still, the participants indicated that other teams and organizations could find the results and insights useful in their work, which the organization supports.

Rich, thick descriptions. Providing rich and thick descriptions is meant to support the *sensitivity* and *usefulness* of a study. By providing detailed descriptions of the setting, participants, findings, assumptions etc., the researcher allows the reader to transfer information to their own settings, and determine whether the findings can be transferred because of shared characteristics. Throughout the dissertation I have provided thick descriptions of the participants, their organizations, and the research setting. I have also elaborated on the findings, and the process through which I synthesized them. Especially in Chapter 7 I discussed in detail all the similarities and differences between the cases, as well as limitations of applicability. In the description of the framework I have remained abstract enough to allow transferability of the concepts (explaining through reciprocal translation analysis what are shared characteristics), while in the individual case studies I have presented specific practices and strategies that could be used in a prescriptive manner.

External auditor. The use of an external auditor is meant to assess the *fit*, *usefulness* and *sensitivity* of the study. External auditors must not be involved in the study themselves but be able to review the research procedure and the findings. For the GitHub study, a colleague of mine was used as an external auditor for the survey and interview process as well as the qualitative coding. In the Atlassian study I had an exit interview and survey with the program manager from Atlassian that oversaw the research project. In the Microsoft study I consulted with other researchers about the research questions and aspects of the survey and interview processes. In order to have the results reviewed by external auditors I also submitted papers to referred

international conferences and journals. Reviewer comments provided the ground for making improvements in the presentation of the findings, and the acceptance of a manuscript for publication signalled that the study met the quality and validation criteria. When publication was not possible, I discussed the study methodology and findings with members of my committee that were not involved in the study, and with other researchers in the field that would provide feedback after I presented to them the relevant details of the study.

7.8 Limitations of applicability

The findings in this thesis and the resulting framework have limitations as to their applicability, due to caveats of analogical inference. I described earlier how analogical inference works; by identifying shared properties of cases one can provide support for the conclusion that other similarities exist. In that sense, the applicability of the findings I described is bounded by some of the similarities in the characteristics of the organizations I studied.

The teams and organizations I studied follow agile software methodologies. The teams in the Atlassian study explicitly followed agile methodologies, and the teams in the GitHub study subscribed to the agile approach and described practices in line with it. At Microsoft, we know from earlier work [16] that many teams opt for agile development, although it is not the only development methodology used. Given that the findings come from teams and organizations that use agile practices, it is possible that *collaboration via aligned autonomy* applies differently in environments that use alternative methodologies (e.g. Waterfall), if at all. However, agile methods are becoming pervasive in industry, as shown by a recent large-scale industry survey reporting that 94% of respondent organizations follow an agile approach [207], which brings more confidence about the applicability of the *collaboration via aligned autonomy* framework.

All the studied teams and organizations specialize in building application software, and for the most part provide their software as a web service. This is somewhat different from exclusively building embedded software, or being contracted to build software for individual clients. Atlassian also does not customize its products for its customers. These companies are, therefore, concerned with the end user and the market, but the features of the software are not bespoke. The nature of these companies' software development then gives room for autonomy,

and developers can offer input on what are the features that should be included in the products and how they should be built. This is both in the spirit of agile practices but also a more general model of collaboration in these organizations. Many software companies are commissioned to build software according to precise requirements and deadlines set by clients, which may limit the applicability of the concepts in the thesis.

The studied companies build software products for non-critical operations. Given the product offering of the studied companies, there is little to no need for compliance to certain regulatory or legal frameworks or other conformity to externally defined criteria. Although lately there are steps to introduce and sustain different flavours of agile practices even in regulated environments [76], the lack of external constraints (besides market conditions) as in the cases in this thesis leaves room to morph the strategy with input from multiple sources and affords autonomy to teams. As such, the practices and framework I discussed may need adaptation if they are to be applied in software development that needs to conform to certain criteria in a regulated environment.

The studied companies' products are not built on core functionality upon which everything develops. Atlassian and the teams in the GitHub study, offer a spread of products and services, which they choose to pursue as business opportunities. There is integration between the different products but no underlying core functionality that restricts how spinoff products are going to be built, as for example would be the case of products powered by a certain operating system or a database infrastructure. The main drivers for decisions, then, are market conditions and competitors and while those may create side effects for dependencies of scope, they pose fewer restrictions on technical dependencies. The *collaboration via aligned autonomy* framework and its related practices may need to be adapted in cases where core infrastructure dictates some of the product decisions. However, although Microsoft does have most of its products building on the Windows operating system, many of the concepts relating to autonomy and alignment were still relevant.

The frameworks and practices presented are relative to the specific case studies. All the observations and resulting theorization I presented come from the particular cases. While I have explained the potential for transferability to other organizations, future work should focus on investigating how the framework applies in environments with different characteristics than presented here. However, the principles behind some of the cultural and managerial concepts relate to employee empowerment and motivation, and take into account the nature of knowledge work,

while respecting the principles of agile software development, and are therefore likely to be applicable to more cases than the specific ones.

Chapter 8

Conclusions and future direction

"We balance probabilities and choose the most likely. It's the scientific use of the imagination."

Sherlock Holmes – The Hound of the Baskervilles

This chapter is meant to conclude the thesis, and discuss avenues for further research. As with any research project, there are always more activities to be done that can increase the value of results, or improve the study's reliability. I discuss some of these activities below as future research, after I summarize the results and thesis argument.

8.1 Summary of the results

Below I provide a list of the results I presented throughout the thesis:

- Through the GitHub study (Chapter 4) I identified five ***team collaboration practices*** that development teams use in conjunction with GitHub's decentralized workflow, and which cover all collaboration elements.
- Through the Atlassian study (Chapter 5) I identified three ***scaling strategies*** within an environment built on four ***cultural values***. The ***scaling strategies*** addressed concerns of how to maintain efficiency at scale, by scaling autonomy.
- Through the Atlassian study (Chapter 5) I also validated that the findings from the GitHub study generalize beyond the use of GitHub, since the teams at Atlassian were following the same ***team collaboration practices*** while using BitBucket.

- Through the Microsoft study (Chapter 6) I identified three ***manager roles*** that emerge in environments where development teams are empowered to determine their own work. The roles were further characterized through fifteen identified attributes for great engineering managers.

- Through a review of the empirical studies I ***characterized autonomy and provided a definition for it***, informed by its contextualized meaning for collaborative software development in the studies, and the literature.

- By using interpretive evidence synthesis through meta-ethnography I constructed ***a conceptual framework of collaboration via aligned autonomy***. The main components of the framework are ***team collaboration practices***, ***scaling strategies***, ***cultural values***, and ***manager roles***, each characterized through approaches. The framework builds on four core concepts: *Transparency*, *Decentralized decision making*, *Aligned Autonomy*, and *Clarity of purpose*.

- Throughout the thesis I presented several practices that software teams and organizations follow for collaboration, that have been abstracted in the areas and approaches in the framework.

8.2 Summary of the thesis argument

In this thesis I proposed a framework for software teams and organizations to organize their work based on the concepts of *collaboration* and *autonomy*. I argued that collaboration and autonomy are both essential in modern software development, but they can be problematic. It is also not always clear *how* software teams or companies can combine the two concepts successfully, in the context of what they want to achieve. I claimed that it is possible and useful to bring a more holistic conceptualization of software development work, and I proposed the framework of *collaboration via aligned autonomy* that ties elements of team practices, organizational culture, and management functions together. In this way, software development work is placed in the context that it is taking place, with multiple views accounted for: the operational role of teams, the strategic role of management, and the cultural and visionary role of the organization. I explained that such a holistic view, scoped on an agile approach, gives development teams a wider perspective than simply having knowledge

of implementation work, and empowers them to feel ownership, contribute important feedback, and participate *meaningfully* in decision making.

I discussed how this theoretical approach brings several implications for the study and practice of software development. The framework of *collaboration via aligned autonomy* implies that maximizing independence between individuals or teams and striving to keep them well coordinated, may be short-sighted at scale and could result in silos that jeopardize organizational goals. This is supported by the different, contextualized meanings that are attached to the concept of autonomy in software teams and organizations. Instead, providing autonomy to individuals and teams, but also displaying information transparently, and involving them in decision making can create an *orchestrated ensemble of autonomous entities* that decide their place in the puzzle, and collaborate toward the desired outcome. It further implies that autonomy does not equal misdirection, and that there are provisions that can ensure it works for the benefit of the organization, without the need to isolate individuals and teams to make them efficient. Finally, it carries an implication for those that conduct research of collaborative software development: it shows that studies should account for more than what traditionally falls under "software development", and include activities that may not be strictly about the development of software but, in a second-order manner, have an influence on it.

The conceptual framework is supported by interdisciplinary theoretical and empirical work, and by the empirical data I obtained from the case studies I presented. Even so, the way the framework has been developed and articulated still leaves open questions, and opportunities for further research. I elaborate on these in Section 8.3.

8.3 Future direction

In this section I present areas where future work can focus to validate, and extend the research in this thesis.

8.3.1 Validate current work

While the findings I presented in the thesis come from the study of multiple software organizations and teams, the software industry as a whole is both large, and diverse. In Chapter 7 I discussed the common characteristics of the case studies, and the limitations of applicability, due to software organization types that I didn't have

the chance to study. While I have been explicit about the limitations of the current framework, I plan to **validate the framework further by studying other types of software companies**. Obvious examples are startup companies, software organizations that operate within regulated environments, and non-agile organizations. By studying such cases I will be able to evaluate the transferability of the framework, and what adjustments should be made to make it applicable for more organization types.

At the same time, the validity of the framework can benefit from **conducting more case studies**, and basing the results on a larger number of companies. Empirical studies that are based on a single or a few cases are historically supported by evidence as contributing to scientific discovery [77], while intense observation has delivered insights in the social sciences ([117], p. 95). For the development of an empirical body of knowledge as championed by Basili [11], both single cases and large samples are essential. I plan — and encourage others — to replicate the studies I described in the dissertation in more organizations, and to this end I provide details of the methodology I followed, and access to the instruments I used in the relevant appendices, and throughout the detailed presentation in the chapters.

8.3.2 Extend current work

I suggest that the bulk of future work should be focused on the extension of the current framework. Two first steps are to **operationalize the areas and approaches I have identified** in the framework, and **link them to organizational outcomes**. Given that we have *evidence* that these areas and approaches contribute to the achievement of *collaboration via aligned autonomy*, and *indications* that they affect productivity, software quality, and motivation, operationalization will allow an assessment of how and to what degree each of the areas and approaches contribute to these outcomes. Such a quantitative expression can help organizations decide how to prioritize their interventions based on the impact they can expect on their bottom line.

A **longitudinal study** would be highly beneficial to identify additional *areas* in the framework and alternative *approaches* to achieving *collaboration via aligned autonomy*. The potential additional areas and approaches could help create a taxonomy, showing which areas require attention depending on the maturity stage of the organization, or changes in its organizational structure and operations. As we saw

throughout the thesis, several contextual factors are relevant to the framework (e.g. culture, growth rate) and these can potentially change over time, and/or cause other contextual factors to change. A longitudinal view of one or more organizations can reveal which aspects of the framework remain constant and which should be added or extended.

Although the domain of this thesis was software organizations, its main arguments are in principle applicable to other groups of people performing knowledge work. While team collaboration practices and some of the scaling strategies may be specific to software engineering, I plan to **perform a comparison of manager attributes and cultural values with teams in other domains** to evaluate whether the arguments of the framework hold across them.

Finally, based on the framework, one can **develop and evaluate an instrument to measure *collaboration via aligned autonomy* readiness** in teams or organizations. This dissertation discussed the areas and approaches at levels of abstraction where we can see if the approaches are present but not to assign an index to them, or compare between organizations or points in time. An instrument could provide the ground for self-assessment, and guidelines for teams and organizations wishing to improve some of their outcomes through a more methodical achievement of *collaboration via aligned autonomy*. Collecting feedback from practitioners that use such an instrument and learning how it helped them improve, can lead to more evidence-based recommendations and facilitation through artifacts and instruments.

8.4 Summary

In this dissertation I presented three substantial case studies based in industry that provide the foundation for a framework to understand the interplay of collaboration and autonomy. Through the synthesis of the findings I described a holistic approach that can guide teams and organizations to reflect on and improve their practice. The framework I presented combines practices on the team and organizational level and covers collaboration practices, scaling strategies, cultural values, and manager roles. I also provided a rich and contextualized discussion of the meaning of autonomy in modern software companies, and the subtleties of its relationship to other concepts. The evidence, findings, discussion, and framework can enrich research on collaborative software development by adding and refining factors to consider, and can support practice by guiding organizational decisions about how to enact autonomy at different

layers of the collaboration process.

Appendix A

GitHub study: data collection instruments

A.1 Ethics Approval

University of Victoria | Office of Research Services | Human Research Ethics Board
Administrative Services Building Rm B202 PO Box 1700 STN CSC Victoria BC V8W 2Y2 Canada
T 250-472-4545 | F 250-721-8960 | uvic.ca/research | ethics@uvic.ca

Certificate of Renewed Approval

PRINCIPAL INVESTIGATOR:	**Eirini Kalliamvakou**	**ETHICS PROTOCOL NUMBER:** Minimal Risk Review - Delegated	**13-176**
UVic STATUS:	**Ph.D. Student**	ORIGINAL APPROVAL DATE:	31-May-13
UVic DEPARTMENT:	**COSI**	RENEWED ON:	28-Apr-17
SUPERVISOR:	**Dr. Daniel M. German**	APPROVAL EXPIRY DATE:	30-May-18

PROJECT TITLE: **Co-CoDe: Coordination, Collaboration, and Development processes in free/open source software projects - A qualitative study**

RESEARCH TEAM MEMBER Daniela Damian, Co-supervisor/Professor (UVic), Daniel M. German, Co-supervisor/Professor (UVic)

DECLARED PROJECT FUNDING: **None**

CONDITIONS OF APPROVAL

This Certificate of Approval is valid for the above term provided there is no change in the protocol.

Modifications
To make any changes to the approved research procedures in your study, please submit a "Request for Modification" form. You must receive ethics approval before proceeding with your modified protocol.

Renewals
Your ethics approval must be current for the period during which you are recruiting participants or collecting data. To renew your protocol, please submit a "Request for Renewal" form before the expiry date on your certificate. You will be sent an emailed reminder prompting you to renew your protocol about six weeks before your expiry date.

Project Closures
When you have completed all data collection activities and will have no further contact with participants, please notify the Human Research Ethics Board by submitting a "Notice of Project Completion" form.

Certification

This certifies that the UVic Human Research Ethics Board has examined this research protocol and concluded that, in all respects, the proposed research meets the appropriate standards of ethics as outlined by the University of Victoria Research Regulations Involving Human Participants.

Dr. Rachael Scarth
Associate Vice-President Research Operations

Certificate Issued On: 28-Apr-17

13-176 Kalliamvakou, Eirini

University of Victoria
Office of Research Services | Human Research Ethics Board
Michael Williams Building Rm B202 PO Box 1700 STN CSC Victoria BC V8W 2Y2 Canada
T 250-472-4545 | F 250-721-8960 | ethics@uvic.ca | uvic.ca/research |

Certificate of Renewed Approval

PRINCIPAL INVESTIGATOR:	**Eirini Kalliamvakou**	**ETHICS PROTOCOL NUMBER:** Minimal Risk - Chair/Vice-chair	**13-178**
UVic STATUS:	**Ph.D. Student**	ORIGINAL APPROVAL DATE:	15-May-13
UVic DEPARTMENT:	**COSI**	RENEWED ON:	07-Jul-17
SUPERVISOR:	**Dr. Daniel M. German**	APPROVAL EXPIRY DATE:	14-May-17

PROJECT TITLE: **Co-CoDe: Coordination, Collaboration, and Development process in free/open source software projects**

RESEARCH TEAM MEMBER Co-supervisors (UVic): Daniel German, Daniela Damian

DECLARED PROJECT FUNDING: **None**

CONDITIONS OF APPROVAL

This Certificate of Approval is valid for the above term provided there is no change in the protocol.

Modifications
To make any changes to the approved research procedures in your study, please submit a "Request for Modification" form. You must receive ethics approval before proceeding with your modified protocol.

Renewals
Your ethics approval must be current for the period during which you are recruiting participants or collecting data. To renew your protocol, please submit a "Request for Renewal" form before the expiry date on your certificate. You will be sent an emailed reminder prompting you to renew your protocol about six weeks before your expiry date.

Project Closures
When you have completed all data collection activities and will have no further contact with participants, please notify the Human Research Ethics Board by submitting a "Notice of Project Completion" form.

Certification

This certifies that the UVic Human Research Ethics Board has examined this research protocol and concluded that, in all respects, the proposed research meets the appropriate standards of ethics as outlined by the University of Victoria Research Regulations Involving Human Participants.

Dr. Rachael Scarth
Associate Vice-President Research Operations

Certificate Issued On: 07-Jul-17

13-178 Kalliamvakou, Eirini

A.2 Initial survey

Title: How do you use GitHub?

<Greeting, introduction, overview of the project aim, link to consent form >

Questions:

1. Your GitHub username:
2. What is the primary reason that you decided to use GitHub?
3. Do yo use GitHub primarily to collaborate with others? (choose below)
   - No, I primarily use it for my own projects
   - Yes, I primarily use it to work on projects with others
4. What is your favourite way that GitHub helps you work with others?
5. How do you manage dependencies between yours and others' work on GitHub? Which processes/tools work for you?
6. Which GitHub tools or features do yo use yo keep track of activity? Activity might refer to developers or repositories.
7. How has the use of GitHub affected your development style?
8. Do you have a cool story to share about GitHub? Something that happened to you or you were involved with?

Please enter your email address if you're up for a short Skype / Hangout call so we can learn more about your response! We'll contact you to make an appointment at your convenience.

A.3 Interview guide

- ***Introduction***

  Hello <NAME>! Thanks for taking the time to do this interview with me! How are you?

  - First: is it OK to record the interview? We want to analyze interviews later for common themes.

- We are Software Engineering researchers at the University of Victoria. We want to find out how people use GitHub, especially with regard to how people work together. We have no commercial interest, this is a purely academic project. We will handle your data confidentially and will anonymize everything you say.

- Is that OK with you? Do you have any questions?

- ***Job situation***

  - What is your job situation at the moment?

  - What are your main tasks? (if needed, prompt with "development, quality assurance, managing..?")

  - What kind of company do you work for, how large is it?

  - Do you work in a team? If so, how is it distributed in locations?

  - What country are you based in?

  - Which technologies do you use?

- ***You and GitHub***

  - What do you use GitHub for? What re your most important use cases?

  - Do you use GitHub for work or outside work?

- ***<If GitHub is used for work>***

  - ***Team profile***

    - Are you in a team?

    - How big is the team?

    - How is it geographically distributed?

    - Is the rest of the team also using GitHub?

    - Can you think about your most important project? If you can't decide, just choose one of the more important ones. What is it?

    - What do you do there, what?s your role?

    - How often do you work on it?

    - With how many people do you collaborate there?

    - Are there any other roles in that project team? Which ones?

    - Are your repositories private?

- ***Collaboration style***

  - How would you describe the collaboration in your team?

  - Can you give an example of collaboration in your team?

  - What tools/technologies do you use for collaborating?

  - Are they adequate? Is there something missing?

  - Do you face problems? Can you give an example?

- ***Workflow***

  - Can you walk me through your workflow step by step? Starting from a list of tasks that the team needs to perform, what are the steps you follow until task completion?

  - What tools do you use in each step?

- ***Coordination needs***

  - What does coordination mean to you? How would you define it?

  - OK, thanks! For us, it is "actions to manage dependencies between people and/or tasks"

  - Keeping that in mind, can you remember the last time you had to do something like that? Cn you give an example of coordination in your team?

  - What are the occasions that you find it essential to coordinate with your team?

  - Can you remember a recent issue with coordination, i.e., where you had to manage dependencies, make a group decision or resolve conflicts between you and others? Not necessarily code-related dependencies, maybe they were organizational?

  - How did you resolve it?

  - What was the root cause for the problem, do you think?

  - Do you have any conventions on coordination that you try to follow and diffuse in your project? Why? How exactly?

- ***Task division***

  - How do you decide how to split the work between team members?

- ***Awareness***

  - How do you keep track of activity in your projects?

  - Do you prefer to track the activity of people or artifacts? Why?

- How does that help you?

- How do you stay aware of people?s actions?

- How do you stay aware of the state of / changes in / activity in artifacts?

- Does it get too much? Is anything missing? Do you face challenges?

– ***Communication***

- What are the occasions that you find it essential to communicate with your team?

- Does it get too much? Is anything missing? Do you face challenges?

– ***Conflicts***

- Do you come across conflicts?

- How do you resolve conflicts in your team?

– ***GitHub's role***

- How does GitHub fit in with all those collaboration pieces?

- Which of these things do you address through GitHub and which through external tools?

- How has GitHub helped your team collaborate?

- Has your use of GitHub changed over time? How?

- Is there something missing? Does GitHub hinder anything?

- ***Summary***

  - Overall, is coordination important to you?

  - Why is it so important, what do you think?

  - Does GitHub make anything of that easier or harder?

  - What are problems you see with coordination right now, in your work, in your personal situation?

- <If repositories were private>, ***Interaction on GitHub***

  - Can you remember a puu request that required coordination? Can you tell me what happened?

  - Do you remember a pull request that got rejected? What were the reasons?

  - Do you sometimes have new people become part of your project? E.g. with merge or commit rights.

- How do they learn your team?s practices? How do they learn to coordinate with you?

- Where did you learn about the coordination practices you currently use?

- Have you learned coordination strategies from other users on GitHub or somewhere else?

- ***<GitHub used outside work>***
  - ***Do you work with others?***
    - Do you receive pull requests from people outside your project?
    - Do you submit pull requests to other projects?
    - Which happens more often?
  - ***What is your collaboration style?***
    - Can you give an example of collaboration in your team?
    - How does Github support you with that? Which features do you mostly use when collaborating?
    - How do you decide how to split the work?
    - How do you keep track of activity?
    - Do you track the activity of people or artifacts?
    - How does that help you?
    - How do you stay aware of people?s actions?
    - How do you stay aware of the state of / changes in / activity in artifacts?
    - How does communication happen? On which occasions? Can you give an example?
    - How do you resolve conflicts? How does GitHub allow you to do that?
    - Do you face any challenges?
  - ***Would you recommend using GitHub for work?***
    - Why? Why not?
  - ***How would you compare coordination on GitHub with coordination in other tools?***
    - Do you see any differences?
    - Do you see any problems in one that are not present in the other?

- Do you see any advantages in one that are not present in the other?
- What do you think are the reasons for the differences?

Appendix B

Atlassian study: supplementary information

B.1 Ethics Approval

University of Victoria
Office of Research Services | Human Research Ethics Board
Michael Williams Building Rm B202 PO Box 1700 STN CSC Victoria BC V8W 2Y2 Canada
T 250-472-4545 | F 250-721-8960 | ethics@uvic.ca | uvic.ca/research |

Certificate of Renewed Approval

PRINCIPAL INVESTIGATOR:	**Eirini Kalliamvakou**	**ETHICS PROTOCOL NUMBER:** Minimal Risk Review - Delegated	**14-338**
UVic STATUS:	**Ph.D. Student**	ORIGINAL APPROVAL DATE:	02-Oct-14
UVic DEPARTMENT:	**COSI**	RENEWED ON:	30-Aug-17
SUPERVISOR:	**Dr. Daniel German**	APPROVAL EXPIRY DATE:	01-Oct-18

PROJECT TITLE: **Building Healthy and Efficient Teams in Software Organizations**

RESEARCH TEAM MEMBER Members: Daniel German (UVic), Daniela Damian (UVic), Lloyd Montgomery (UVic)

DECLARED PROJECT FUNDING: **None**

CONDITIONS OF APPROVAL

This Certificate of Approval is valid for the above term provided there is no change in the protocol.

Modifications
To make any changes to the approved research procedures in your study, please submit a "Request for Modification" form. You must receive ethics approval before proceeding with your modified protocol.

Renewals
Your ethics approval must be current for the period during which you are recruiting participants or collecting data. To renew your protocol, please submit a "Request for Renewal" form before the expiry date on your certificate. You will be sent an emailed reminder prompting you to renew your protocol about six weeks before your expiry date.

Project Closures
When you have completed all data collection activities and will have no further contact with participants, please notify the Human Research Ethics Board by submitting a "Notice of Project Completion" form.

Certification

This certifies that the UVic Human Research Ethics Board has examined this research protocol and concluded that, in all respects, the proposed research meets the appropriate standards of ethics as outlined by the University of Victoria Research Regulations Involving Human Participants.

Dr. Rachael Scarth
Associate Vice-President Research Operations

Certificate Issued On: 30-Aug-17

14-338 Kalliamvakou, Eirini

B.2 Blog post addressing employees at Atlassian

What is the magic of the Atlassian teams (and can it work better)?

Hello Atlassians,

My name is Eirini and I ask you to take 5 minutes to read this post on an upcoming research project taking place in Atlassian.

Probably the first thing that comes to someone when they hear the name “Atlassian” is team collaboration. It’s the goal of Atlassian’s products, but also the foundation of the company’s innovation and creative thinking. Judging by Atlassian’s success, then your teams are doing a great job of transforming individual expertise into collective excellence. Your collaboration evolves and sustains a company with a great reputation and an impressive track record in products, customers, and tool adoption.

Collaboration is a notoriously tough nut to crack though. Often it doesn’t work exactly how you would expect it to (not knowing why), and sometimes it works surprisingly smoothly (again not knowing why). When teams can’t collaborate well even the most mundane task becomes a disaster, never mind the talented people or the clear plan. When teams work well together they overcome obstacles and deliver something they are proud of. So, what’s the recipe?

Short answer is that there isn’t one (yet), which is why when you find teams that collaborate and perform well you want to study the magic. That’s my passion. I see all major successes today being team-driven and for my doctoral work I chose to look at how teams collaborate well and what it takes to support them, which aligns with what Atlassian is about. In my previous studies I examined open source-like development practices in environments that support open collaboration, such as GitHub.

Part of how teams work best relates to how they perceive themselves; how satisfied they are with what and how they’re doing, and what’s getting in their way. Performance metrics and surveys only tell part of the story; nothing beats the insight from observing teams at work, and talking to people. I invite you to help in a study that will look to:

- **learn about what works great**. You, your team, and your company will know what you don’t need to change (which is as useful as knowing what to change). Plus you get to be part of real scientific research and maybe teach others by your example.
- **zoom in on things that could work better for you**. Something that may have been bugging you and your team and keeping you from “being the best you can be”.

All the insight will serve to put together an actionable framework in the end. Meaning, your own recipe.

I’ll be happy to talk with everyone, thank you for your time!

B.3 Product development process & tool use

The following is a description of the general approach to product development that the teams at Atlassian follow.

Throughout Atlassian, teams use the very tools they develop and offer to their customers. This practice, colloquially known as dogfooding [96], has multiple functions for the teams. First, it allows them to **test certain features within the Atlassian environment and get feedback** from people that closely resemble the customers, before releasing to the actual customers. Second, since the developers are also consumers, the practice **gives ideas to the teams for features they should be developing next**. Third, the **developers identify with the product beyond the scope of their job**. As the head of Engineering put it, referring to the the Bitbucket product:

⬭ *"People are very passionate about making it good and they take pride in representing themselves as the Bitbucket team. Every time we change anything we get a reaction from the community through social media, and even for just that fact they want to make it good."*

The product teams use the stack of Atlassian tools; they mix the tools with other mechanisms (such as meetings) and match tools to different purposes. Below, I describe the product planning and development process, as was described by the Head of Engineering and the developers of the Bitbucket team, and discuss it along side the tools that are usually used at each stage.

The Ideation phase

In the *Ideation* phase, the product team decides on the next things it is going to build. As reported in the interview with the Head of Engineering, input for this stage comes from multiple sources, as listed below:

1. **Ideas that come internally from developers**, who are also consumers for the products
2. **Business goals set organizationally for Bitbucket**. The rationale is that BitBucket is the main tool of the developer tool suite and it introduces customers to the rest of the Atlassian tools; how can it be built in a way that it helps other products too?

3. **Ideas for improvement coming from the support team talking to customers**
4. **Customer interviews**, performed by the product management team. The description of how the customers' teams work, provides insight on how to fit the tools to the teams' workflow
5. **Tracking competitive products** and what they offer
6. **Educated guesses that come with experience**; ideas turn into features shipped to customers, and are evaluated for 2-3 months in terms of the engagement they generate.

The organization's strategic goals about products are broadly set in a document describing the Vision, Targets, Focus, and Metrics (VTFM), which is written and communicated to the organization by the co-founders through Confluence. The VTFM describes where the organizations sees itself a year ahead, and what it wishes to accomplish. As the broad goals cascade via the leadership (CEO, vice presidents, Head of Engineering, Ecosystem etc), and management, they are described more concisely. Once the goals are clear enough to be considered together with the other sources of information listed above, they culminate in a product roadmap for the development team, with short and long-term objectives, up to 6 months. The roadmap produces big streams of works (epics, in the agile terminology). During this phase, the main communication mechanism is meetings, and the primary tool used is Confluence.

From the observation of the planning meetings, it was clear that Confluence is the tool that acts as the record and the reference point for the team. Confluence is a wiki that serves as a knowledge management tool, allowing teams to organize and discuss content that has to do with their work. Confluence has a high degree of versatility for teams, because they can (and do) use it in different ways and for different purposes depending on the stage of the work they are in and the needs they have at particular times. Confluence supports the creation of spaces, which are essentially sections within the wiki. Any individual, team, department or other entity in the organization can form a space that will be dedicated to holding information about a particular topic. For example, the Bitbucket team has its own Confluence space where they have built a knowledge base for all the information that is of interest to the Bitbucket team, or that current and future members need to know about. Spaces are made up of pages; there are templates for pages that need to have a specific form, but pages are also fully customizable. During meetings, the product teams create meeting minutes

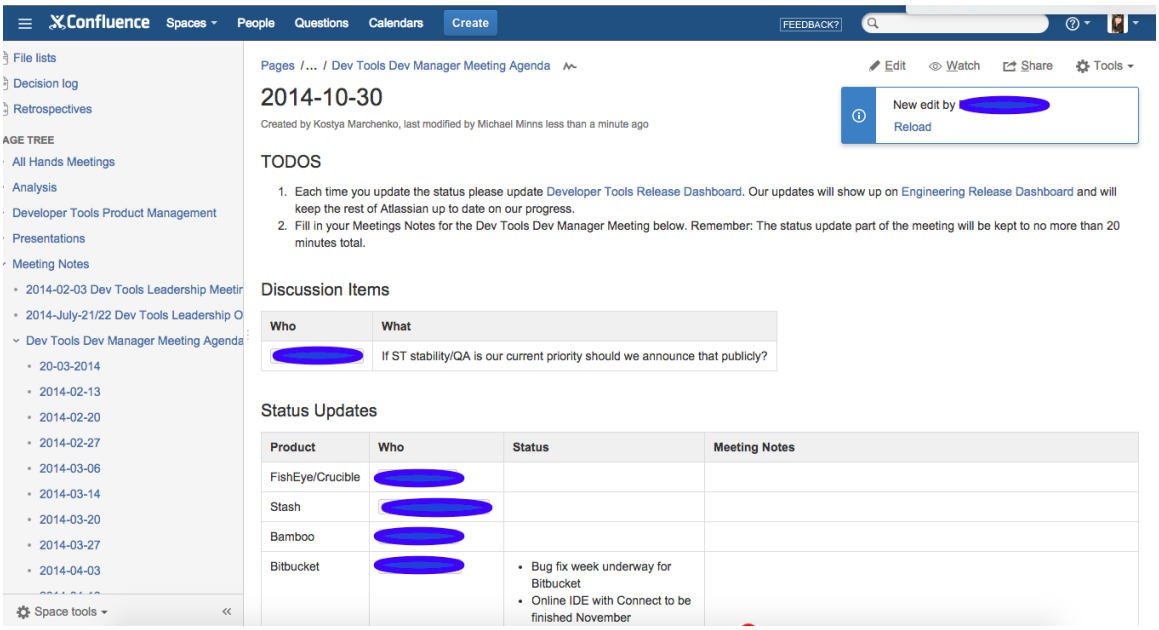

Figure B.1: Screenshot of a Confluence page used for a meeting agenda. Names of participants have been removed.

on Confluence, noting the date, goals, attendees, discussion items and any decisions that were made through the meeting (example shown in Figure B.3). By default, all meeting minutes are publicly visible in the Atlassian extranet, and open to comments and edits.

The Requirements building & Design phase

Despite the name, this is one phase involving two activities that inform each other in iterations. In the *Requirements building & Design* phase the team gains more clarity and reaches consensus about how the things it will build should work. The epics in the roadmap are turned into specifications by product managers. These specifications are equivalent to the more traditional requirements, and are written in a succinct manner. As with all other documents, specifications are publicly visible within the organization. They receive on average 30-40 comments, a fact that was mentioned in the interviews and was confirmed by the review of documentation. Employees that are not part of the product team can comment and give opinions. There is a template page in Confluence for documenting specifications. All product teams are outcome-driven, and the goal is always to satisfy customers. The specifications, therefore, are

aimed toward describing what will be the value for the customer if the corresponding piece of work was to be completed, and if this should be a goal for the team. The team decides which goals to focus on next, based on the value to the customer and the effort for the team, within the scope of the high-level strategic goals. The specification template has a specific area that describes how a particular feature links to strategy, so that the team can take that into account when deciding on the next features to build. Assuming that a goal is picked up for the next iteration, designers create User Interaction (UI) and User Experience (UX) designs to define what the experience will be for the customers. All specifications are documented in Confluence.

Once the epics are defined, feature teams are created ad hoc and take ownership of the work. The feature team is made up of 1-3 developers and a designer, and there is a team lead that makes sure that the specifications are in the right shape and are in charge of making sure that the specification description and the upcoming estimations are in agreement. Specifications are broken into tasks (user stories, in agile terminology) and sub-tasks, are documented in Confluence (examples shown in Figure B.3), and are brought into the meetings. In the weekly planning meeting, the team discusses the priority and effort estimates for upcoming tasks. Effort estimation and scheduling are consensus-driven and are done by the developers. Different teams may follow their own practices regarding effort estimation. For example, the Bitbucket team follows the technique of "t-shirt size estimation", where they define a task as being small, medium, large or extra-large. The Head of Engineering explained that this is because the team decided that estimation through assigning story points to tasks or requirements was time-consuming without giving them much more benefit than the t-shirt sizing approach, which was considerably faster. The Core Ecosystem team favoured planning poker as their method of estimation for tasks. There is a planning meeting at the beginning of eacj week, and a demo meeting at the end of the week, where designers show their designs and get feedback from the developers.

User stories that represent tasks are opened as tickets on JIRA, which supports issue tracking, bug tracking, and project management functions. JIRA holds the necessary information to plan the sprint, and track how tasks progress, allowing the developers that work on the tasks to move them from one stage of work to the next, changing their status. JIRA supports both Scrum and Kanban boards, but is customizable if teams are using more customized workflows. JIRA also integrates with the Bitbucket tool and so provides traceability between the tasks and the code contributions that correspond to them. The Bitbucket and Marketplace teams are

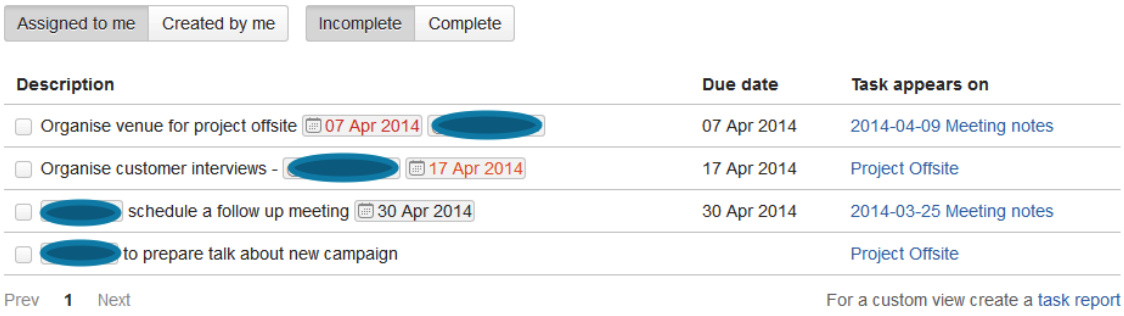

Figure B.2: Screenshot of a task list on a Confluence page. Tasks can be automatically generated out of a Confluence meeting agenda, and link to JIRA tickets. Names of participants have been removed.

using JIRA for their tasks, and the specifications link to JIRA from the description on Confluence. This is the practice of all 3 teams, although they are following different agile methodologies (Scrum versus Kanban). Once there is a list of tasks in place, the team decides the work distribution in their planning meeting. In the Bitbucket team and the Purchasing team the developers select tasks for themselves, while in the Core Marketplace team, the development team lead assigns the tasks to the developers. Once there is an agreed distribution of work and everyone know what they are going to be working on for the week, the team enters the next phase.

The Implementation phase — An iteration

The following phase is *Implementation*. This includes code development, redesign, testing, validation of prototypes (what is referred to as dogfooding), and finally deployment. Once the feature team have the list of tasks they will be working on for the next sprint, and developers have picked or been assigned their tasks, they begin their individual part of the implementation, which is code development. The development workflow is described in more detail in the next section, but is such that it makes use of a branching protocol and the use of pull requests and incremental code reviews. All along the workflow, the developers collaborate and share prototypes with other members. This happens either during the regular demo meetings, or informal meetings and serendipitous interaction. For example, I observed that it was not uncommon for members of the Bitbucket team to talk with the designer while looking at the design wall with printed mockups of the prototype they put together. Next to design walls

Figure B.3: Photo of a design wall. There are printed screenshots of an upcoming infographic that was to be released publicly on the Atlassian website (published version at `https://www.atlassian.com/time-wasting-at-work-infographic`). The sticky notes on the screenshots correspond to comments about the contents of a screenshot, left by other employees passing by.

around the office there are sticky notes, pens and highlighters that anyone can use to leave comments. I observed that this was a popular practice; employees walking by a design wall would stop and look and leave a brief comment on a sticky note (example shown in Figure B.3). On one occasion, the same employee that left a comments passed by the design wall later and saw that someone had commented on their comment and used another sticky note to responds, creating an informal discussion thread through sticky notes.

Once the developers have gone through enough cycles of sharing their progress and getting feedback, they submit their work for peer review. The code management is handled through Bitbucket and the tasks are moved from one status to another

on JIRA. Developers write and maintain tests along the workflow, and they have to provide test notes with their code. Once the submitted code has successfully gone through peer review, the new features are submitted to dogfooding, which means that they are made available internally to Atlassian employees for one week to two or three months before they are made available to the customers. After the dogfooding phase, and any resulting changes, the new features are deployed to production and then released to customers. The tools that are used in these phase, on top of using JIRA and Bitbucket, are Atlassian offerings for software teams; code reviews are handled through Crucible, and continuous integration, deployment and release management are handled through Bamboo. Once an epic is completed, the feature team dissolves and its members can now form different feature teams as needed.

Communication is a key element for the product teams (and the organization as a whole), and supporting communication is a common feature in all Atlassian products, and therefore the suite of tools that the product teams use for their work. All the artifacts on Confluence, JIRA, and Bitbucket can be commented on and create a basis for discussion and feedback. In parallel to all the activities in this (and indeed all other too) phase of product development, teams also communicate closely through Hipchat, the enterprise chat tool used at Atlassian. Hipchat has multiple functions as a communication tool. For product teams, it serves to handle all the communication happening in the team and form a record of discussion and decisions about features. All product teams have their own chat rooms inside Hipchat, although the chatrooms are open to anyone in the organization. Any employee can, therefore, look at the discussion that is happening inside a product team and participate. Product teams can also choose to have multiple chat rooms, used for different purposes. For example, the Bitbucket team has a standup channel, and a channel that tracks deployments and releases, among the channels it keeps besides the main communication channel. Hipchat also integrates with the other Atlassian development tools and so delivers notifications about submitted pull requests, deployments, tests, and releases (example shown in Figure B.3). That way the team remains aware of how work is progressing in various stages without needing to monitor the separate tools.

When the new features are released to the customers, there are supporting activities that take place, handled by other teams. For example, technical writers write blogs about the new features which are posted on Atlassian's public blog. The marketing team handles the promotion of the new release and features and there is mention of the updated product in social media. As the features are taken up by

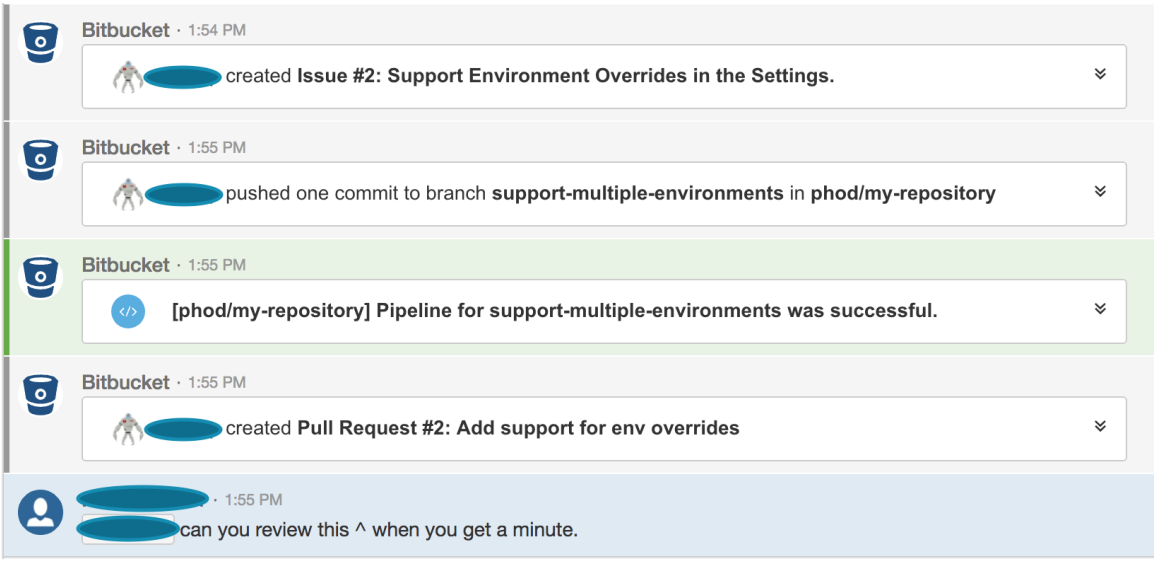

Figure B.4: Example of notifications from Bitbucket showing up in Hipchat.

the customers, the product teams would like to evaluate the features to see how they served the customer needs, and where there is room for improvement. The product teams use a metrics-informed usage report that shows them how often and in what way the features are used by the customer. The Head of Engineering explained that there is still way to go to make full use of the data-driven decisions for features, but it is used more and more.

On a monthly basis, teams hold a retrospective, during which they review their process and progress of the last month and propose things they should start following and things that didn't work for the team and should, therefore, no longer be part of their process.

The Special weeks

Interleaved with the product development process, there is a protocol that the team has come up with that shifts the attention from the product to the team and its operations. Out of every 10 weeks there is a "special week" that takes the form of one of the below:

- **Innovation week**: the developers can work on whatever they work, as long as it is loosely related to Atlassian products. These may be ideas that the developers have put forward to the team but have not been prioritized by the

other developers, and they need to result into something that is shipped.

- **Performance week**: a week that is dedicated to looking at the systems that the team and the product is using and trying to find ways to improve their performance. The systems' performance drops gradually, so this is a practice that is seen as good housekeeping on the team's part.
- *Bug fix week*: a week devoted to resolving as many bugs as possible. Bugs come from reports from the customers of from the team itself or other teams in the organization. Unless the bugs are critical and/or they prevent the product from working, they are deferred to a later stage for resolution. The Bug fix week is the team's chance to catch up, and ensures that the less desirable task of fixing bugs is a shared one.

It is important to note that during these special weeks, the product development activities stop, in favour of what is the chosen focus for that week. This is something that is taken into account during the planning of the epics etc and the expected progress and time line.

B.4 Collaborative development practices

B.4.1 Code development workflow

The workflow that is described below is the process that corresponds to the code development in the *Implementation* phase. At Atlassian, teams make their own decisions on the workflow they will follow and that means that there is some diversity in the practices. I asked the teams we interacted with to describe their workflow to me, starting with a list of tasks that the team has to perform, and ending with deployment to production.

The three main teams that I observed (Bitbucket, and two Marketplace teams — Core and Purchasing) use a *branch-and-pull workflow*. Stories in JIRA represent tasks for developers and are picked up or assigned during planning meetings. Both Bitbucket and the Purchasing team have developers volunteer for tasks, while the Core Marketplace team follows a process where the project manager and developers discuss stories, estimate story points, and agree on the stories a developer will work on for that week. Teams also use JIRA as an agile project management tool. Entries in JIRA represent issues that can be tasks, bugs or other types or tickets. However,

JIRA works as a team's dashboard, through which they can organize their sprint. As team members work on tasks, they update the status, which supports visibility of the progress of work. JIRA supports the Scrum and Kanban methodologies, but is also customizable to be used with variations.

After task selection, developers work on their separate feature branch, committing their code there. This usually takes the form of multiple commits, with comments attached to them. Once a developer is ready with their code for the task, they submit a pull request through Bitbucket. All three teams follow the protocol that the developer submitting the pull request nominates 2 reviewers for their code. While reviewers are typically team members, there is no restriction to nominating someone from another team to review a different team's pull request. In addition, the pull request is visible and anyone can participate in the code review. Generally the author selects reviewers that have expertise in the area of the contribution. The pull request needs two "thumbs up" (two approvals) to be accepted for inclusion in the codebase.

The review of the pull request involves discussion, which usually happens in either comments attached to the pull request, or on Hipchat. Rarely an author and a reviewer need to sit down together so that the author can explain parts of the code they have written for the reviewer to gain better understanding. Part of the code review on pull requests is a list of changes requested by the reviewer. The author needs to also supply test notes so that the reviewer can run the functionality on a live system to check that everything is working. Occasionally a reviewer may give a conditional approval, contingent on the author making a particular change. Once the pull request is accepted it is merged to the main branch and it transitioned to a staging status, also reflected by a change in status for the task/story in JIRA.

Most teams at Atlassia use continuous builds and integration and in most cases 30-60 minutes after merging there is a successful build that can be deployed to production. The automatic deployment system will then close all issues associated with the pull request and its relevant tasks, delete the feature branches, and handle other parts of the release. When asked if everyone on the team is keeping consistent with this workflow, a project manager replied

⬭ *"On the whole we are consistent, but there are exceptions. I can think of one developer that always leaves a story open until it is ready for review. So he may be working on it for days but it looks open all this time until he sends a pull request."*

The workflow described above is identical to the *branch & pull* workflow that was

reported by teams using GitHub as their collaborative development platform.

B.4.2 Collaboration elements

The teams that I reviewed at Atlassian use Bitbucket as their collaboration development platform and they submit pull requests using it. Part of the communication happening in the team during development is through comments on the pull request and/or the JIRA tickets that correspond to the task the pull request is associated with. This form of communication is code-centric and attached to the artifact, in a similar fashion to what I observed in the way teams use GitHub. Developers also track the progress of tasks and the project through dashboards pulling information from JIRA, displaying the change in status for all tasks.

Teams at Atlassian use Hipchat as the hub of their *communication*. Hipchat was also mentioned by all the Atlassian employees that I interacted with and observed as the main communication tool. Although at the beginning of my study (and visit) I used email to coordinate with Atlassian employees to ask questions or arrange interviews, I quickly switched to using Hipchat as it was recommended by many employees and it was mentioned as their preferred method of being contacted. Inside Atlassian, Hipchat has replaced email as a communication tool; interviewees reported that it provides an unobtrusive way for employees to contact others, by mentioning them or messaging them directly, while the rate of receiving notifications is customizable to reflect the level of awareness that a user wants. Most employees have installed the Hipchat application on their smartphones and keep in touch with messages when they are away from the office.

Communication is a big part of the teams' life. Product teams have their own communication channels or rooms inside Hipchat, with separate rooms serving different purposes. This practice allows for a separation of the different types of discussions the team has, and keeping multiple, but not convoluted records of decisions made during discussion. The Bitbucket team, for example, has been using a separate Hipchat room for its daily stand-up meeting, while later on during the study Hipchat integrated with a stand-up bot to make it easy for teams to report and retrieve statuses for stand-up meetings. With this practice, all the information concerning the stand-up meetings and the progress or problems reported in them is recorded and archived for awareness, while team members that are not in the office or are remote members can still connect with the rest of the team and share their information. Teams also

use Hipchat rooms that are meant for social purposes only, where members discuss off-work topics. Social rooms are not team specific and usually focus on a theme, and are open to anyone in the organization that wants to join.

Hipchat integrates with a number of other tools, and sends automated messages to the teams' rooms, updating them about the submission of pull requests, the outcomes of tests and staging, as well as deployments. In this way, even if the team is discussing something else or members are passively present in a chat room, they can still be informed through notifications. The developers maintain *awareness* through the automated messages on Hipchat, but also the dashboards that show the progress of tasks.

The highly visible project status helps developers stay coordinated as well. The need for *coordination* was something that all the participants agreed on, recognizing that it is important to find the balance between too little and too much for coordination. The participants found that the *branch & pull* workflow highlights the points when coordination is needed, that is the submission of a pull request. Communication through comments, and additional discussion on Hipchat complement the signalling of points that require the team to coordinate. Since the team's protocol is such that expects the author of a pull request to make all the necessary changes for its approval, *conflict resolution* is a responsibility of the author. In that sense, the author has ownership of the task from the time that it is assigned to them, up until the time it is deployed to production.

As mentioned before, the *task division* process is not uniform between all teams at Atlassian. Some teams (like the Bitbucket team and the Marketplace Purchasing team) estimate tasks and issues via consensus and have the developers pick the tasks that they will work on for the next week. The Core Purchasing team follows a process of the manager and the developers discussing the nature of the tasks the team needs to perform, but the tasks are then assigned by the manager.

B.4.3 Comparing practices across platforms

Atlassian is an interesting case as its development teams use OSS-style development practices (making it a good Inner Source example). Atlassian also practices open collaboration across the entire organization with all communication being accessible and transparent, together will all information sources and repositories. Below I elaborate on the similarities I noticed between the practices used in commercial teams using

GitHub and those using Bitbucket as their collaborative development platform. The overlap between the practices that were reported by the teams using Bitbucket and those I recorded from the teams that were using GitHub is large. This is important, as it lends merit to the argument that the **workflow and accompanying practices that we see are not tool-specific, but they represent a de-centralized and de-coupled way of working, *supported by the tools***, that teams in several organizations favour. The practices from both cases are summarized in Table B.1.

Table B.1: Practices reported by development teams at Atlassian, the corresponding Bitbucket enablers, and how they support the collaboration elements. I have included the GitHub enablers from the corresponding study to highlight the overlap.

Reported practice	**Bitbucket enabler**	**GitHub enabler**	**Collaboration element effect**
Independent development	Branch & pull workflow	Branching workflow (also mentioned as GitHub workflow)	Minimized *coordination* needs
Progress and status monitoring	Timeline, notifications, integration with HipChat	Timeline, notifications, integration with chat client	*Awareness*
Code reviews	Pull requests with in-line comments	Pull requests with in-line comments	Highlighted *coordination* needs
Code-centric communication	Comments on commits, issues, and pull requests	Comments on commits, issues, and pull requests	Targeted *communication*
Self-organization	JIRA issue tracking	Public issue tracking	*Task Division*
Automatic testing and deployment	Service hooks to Bamboo	Service hooks in corresponding tools	*Conflict Resolution*

In both cases — teams using GitHub, teams using Bitbucket — I used the same enquiry protocol, although I also accommodated positive digression in the conversations with participants. The longest duration of the investigation at Atlassian allowed me to gather more data and understand more processes than strictly the development and code management ones. However, in my investigation of the way teams in both cases use the collaborative development platforms the two studies are comparable.

The OSS-style practices that were reported to me in the investigation of commercial software teams using GitHub, were also followed by the commercial teams using Bitbucket inside Atlassian. This fact validates the OSS-style practices I noticed being used by commercial teams, since it appears they are not restricted to a specific tool or organization. It is worth noting that even though Atlassian used a different platform, the features that Bitbucket provides are largely similar to the ones offered through GitHub. Both platforms build on the pull-based development model and offer the same branching workflows. The functionality around pull requests and issues is also identical, as well as how notifications are handled both through email messages and

as integrations with other communication tools. Both GitHub and Bitbucket offer the commenting functionality on every artifact, be it a pull request, an issue or a commit. In-line comments are available in both tools, so that developers can have code-centric discussions. During the two studies, there was no noticeable influence of the technology on the collaborative development practices. This is not to say that the practices are tool-agnostic; it rather points out that its the concept of decentralizing work that is getting adopted, alongside with the tool.

The investigation of Atlassian validates the presence of OSS-style collaborative development practices in commercial software teams as were recorded in the GitHub study, and extends their relevance beyond the specific tool and size of organization.

B.5 Further challenges identified in interviews

Even with the provisions made by Atlassian's work practices, challenges still exist. Two challenges that stood out from the interviews and reviewing Confluence have to do with the side-effects of open communication at scale, and the the side-effects of keeping a flat and flexible organizational structure. The description below highlights that there are trade-offs between challenges and practices; some practices are put in place to mitigate certain challenges, but subsequently expose limitations or create challenges of their own.

B.5.1 Challenge of information transparency at scale

1. **Communication openness creates information overload** All information and work artifacts are by default visible across the organization and archived in Confluence, resulting in high volumes of information produced during the course of a single day.

The practice of open communication showcases a tradeoff between practices and challenges. Transparency of all information creates a buzzing environment at Atlassian, containing all forms of information: meeting minutes, enterprise strategy decisions, roadmaps and plans, personal posts and opinion pieces, to mention a few. This creates

instant visibility of all information that is produced and circulated in the organization, and is a practice that not only is a powerful cultural value in the organization, but also serves as the basis for autonomy to be effective.

The downside is the danger of information overload. The interviews identified this as a definite problem. Firstly, **the volume of information that is produced during the course of a single day is overwhelming**. As one of the interviewees said, referring to Confluence:

💬 *"The conventional wisdom is: Don't even go there, you'll get sucked in"* Following the information can prove a major distraction and the mitigation strategy for everyone is to follow only certain types of information or colleagues to keep up to date. Confluence offers a mentioning function; certain people, teams or locations can be mentioned in an article and subsequently notified through email. In addition, there are tags that users can apply to artifacts, making them more identifiable as to their relevance. Even with filtering however, product managers and executives end up receiving a large number of notifications because of the frequent mentions and generic tags. Information overload was the major challenge that the interviews showed related to the total visibility of information.

The second reason that the accumulating information becomes problematic is that **it is difficult to track and navigate information on Confluence**. This was a repeated note in the interviews, and confirmed through the internal satisfaction survey; the search function on Confluence is far from optimal and, coupled with the amount of incoming information, makes it hard to find information again. Teams and individuals maintain their own spaces within Confluence where they archive the information that is relevant to them for easy navigation. However, there are no uniform rules for archiving and curating documents and other communication artifacts across the organization, making retrieval challenging. The challenge shows that there is need for better categorization and tagging of information, as well as a guide of recommended naming conventions across the organization for files and other documents, to make information easier to navigate, until there is a better search functionality in place.

The rapidly growing number of Atlassian employees that participate on Confluence is only expected to accentuate this challenge.

2. **Information openness can cause friction** Information sharing in the open poses a challenge to balance between discussion and decision; comments and

questions deserve consideration but with scale it becomes complicated and slow.

Having a large (and growing) number of people in an open communication environment has the potential of heated debate. While this is as natural as it is healthy to an extent, the more people involved in a discussion the easier it is to encounter problems. Several interviewees brought up the same incident of a post on Confluence discussing early ideas on product branding, which received severe criticism and resulted in negative sentiment. The general view was that it was not the open discussion that created the problem, but that the idea was in too early of a stage to invite constructive discussion. In a open information and communication environment premature criticism of ideas can leave a bad taste to the original poster (and others) as well as uncertainty about how to contribute ideas and information. Geographical distribution and the fact that two ends of a heated debate may not have met each other can amplify this challenge, making arguments risky. The Head of Risk Management commented:

💬 *"Right now it is up to the individual to know when something is posted as a request for feedback or for your information. We can still be open but put some more thought into our comments"*

Participating in an open communication environment that makes all information visible also puts a question of boundaries. The interviews showed that although there is strong belief that every comment and question deserves an answer, there are no clear boundaries around the time to discuss and the time to make a decision on an issue or the weight that each voice carries. As a result, addressing every comment on a page that has to do with a decision can be a distraction and not meet everyone's expectations of having an open discussion on the issue. There seems to be room for introducing some desired patterns for communication that could give some guidance around how long discussions should go for etc.

As the company grows, the number of people that have impact on a certain decision remains almost stable, while the number of people that can participate in a discussion — with the possibility for heated debate — increases, creating a risk of inefficient conversations. While operating on decency and common sense etiquette is sufficient at present, articulating guidelines promoting healthy debate could help with anticipating possible future problems. OSS projects use a "Code of Conduct" as a set

of guidelines for smooth communication and interaction in their large and diverse communities; this practice could work in Atlasssian's case too, instilling conversation etiquette as part of the on-boarding experience for newcomers too.

Finally, Atlassian is home to different generations of information users that interpret information sharing and communication etiquette differently. Setting examples and educating users on what is considered mindful and constructive language, content and level of participation is a gateway to ensuring smooth and effective communication, especially as Atlassian is considering opening up some of its communication channels to their partners and customers in the future.

B.5.2 The challenge of a flexible organizational structure

1. **Rapid growth brings in new hires regularly** As Atlassian grows and expands, it onboards new members contantly.

New employees contribute their skills toward Atlassian's growth, and the organization subsequently changes its structure to accommodate and integrate them into its routines. Even though Atlassian constantly looks to improve its onborading activities, how to spread the culture to a constantly new crowd remains a challenge. A practice Atlassian has followed is to shuffle senior developers around so that they can help and guide new employees and/or new teams. Culture is important to Atlassian and is generally regarded as the company's winning factor. As one of the interviewees wondered:

💬 *"Everyone is brand new all the time, how do we indoctrinate this culture that existed with the older members that are now outnumbered 10 to 1 by new people with their own ideas?"*

2. **Cross-functional work requires employees to move around** Integrations between Atlassian's products create a need for product teams to work together, resulting in a change in roles and teams.

With Atlassian's scope expanding, changes in the organizational structure reflect changing needs. Some occasions call for people to move from one team to another, move to a different hierarchical role, or for teams to work independently or together. Shifts like these are mostly ad hoc and allow the organization to be flexible to adjust

to market opportunities and challenges, and in a way its agile mentality. A challenge that was mentioned along side the pursuit of flexibility was that of teams not enjoying enough stability to be as efficient as they could be:

💬 *"When you change all the time you never really get into the groove, I'm looking forward to have teams be teams for an extended time"*

A team's identity may also be challenging to establish and maintain. Occasionally managers may also find themselves needing to argue and explain the potential negative impact of moving people to other positions. A development team lead said:

💬 *"I don't have a problem losing people, but my concern is to do it in a way that it doesn't cripple the team"*

3. **Ownership and decision making become unclear** Decision making responsibility for tasks or parts of a product, becomes challenging to trace as people move between teams.

With adding new people, and having existing people change roles or teams, who has say in decision making may become unclear. In addition, multiple developers touch a system; everyone feels ownership, while it may become obscure who should participate in discussions or who is responsible for what, as the following quote demonstrates:

💬 *"People come to me because they think I'm the go-to for that. Even if that's no longer the case I don't want to send them away. I will volunteer to locate information for them, although it turns into a distraction often enough"* While people volunteer to answer questions and direct people to the right person, or even find out who that is for them, it can prove time-consuming and a distraction. This challenge brought up the need to curate information, especially when it relates to ownership and responsibility.

B.6 Analysis of how Atlassian's work practices fit with Aligned Autonomy

Let us start with the practices that relate to the level of the individual. Table B.2 below acts as a reminder of the practices and shows the categorization which I discuss now.

The *use of branches for specific tasks or features* isolates the development that relates to different tasks. In that sense it provides a high level of Autonomy to indi-

Reported practice	**Level of Autonomy**	**Level of Alignment**
Developers use branches for their work on a task or feature	high	high
Developers select the task they are going to work on each sprint	high	high
Developers submit pull requests when they have changes ready	medium	medium
Developers make changes on their branch	high	medium

Table B.2: Practices on the level of the **individual** and their related levels of Autonomy and Alignment

vidual development. At the same time the practice provides a high level of Alignment since the developer can see how her work fits in with the product, as this has been pre-decided when building the backlog, and do not create a need to consult with others. The practice of *task self-assignment* provides a high level of Autonomy in task division, and a high level of Alignment since again, via the decisions about the backlog, the fit of one task with the others and the objectives for the product team is clear. The *submission of pull requests* creates some need for the individual developer to coordinate with team members for the code review that comes right after; as a result it relates to a medium level of Autonomy. Regarding Alignment, the practice is also related to a medium level, since it requires team members to consult with others about the fit between the delivered feature and the intended functionality, which happens in that code review. Finally *keeping the changes to the branches*, in a similar way to the practice of using feature-focused branches, isolates development between different developers and, hence, results in a high level of Autonomy. It is still required that the developer then consults with others through the code review process to assess the fit with the intended functionality and, therefore, the practice corresponds to a medium level of Alignment.

Table B.3 shows the practices at the team level and their related levels of Autonomy and Alignment.

The *end-to-end ownership of the workflow* provides the team with a high level of Autonomy, since the practice makes the team independent of others in completing work from task definition to deployment to production. At the same time the practice provides support for high level of Alignment because the organization has a clearly

Reported practice	**Level of Autonomy**	**Level of Alignment**
End-to-end ownership of a team's workflow	high	high
Deployment to production is handled by the team	high	high
Submitted pull requests need two reviewers	medium	medium
The backlog is decided by the team	high	medium
Builds are self-managed by teams	high	high
Teams define their own scope and timeline	high	medium
Teams include engineers, designers, devops and support engineers	high	medium

Table B.3: Practices on the level of the **team** and their related levels of Autonomy and Alignment

defined a documented process for teams to follow in order to test, build and deploy their work. For the same reasons, the fact that the process is such that *deployment to production is handled by the team* provides support for both high levels of Autonomy and Alignment. Similar to deployment, *builds are also self-managed by teams* and, therefore, support high levels of Autonomy and Alignment. Another practice that makes teams self-sufficient and, therefore, provides high levels of Autonomy is that of *the team including engineers, designers, devops and support engineers*. In terms of Alignment, the practice means that the developers will need to consult with others to ensure how their part fits with product decisions, as they stem from providing support to customers for example. It, therefore, provides support for medium level of Alignment, with the potential to support it fully, the more seamless the flow of information is between the roles.

We saw that *submitted pull requests require two reviewers* before they are approved for merging to the main codebase. This is a practice that creates some need for coordination (to carry out the review) and therefore results in a medium level of Autonomy for the team member and for the team. It is also associated with a medium level of Alignment as it creates the need for the developer to consult with someone (the reviewers) about how well their implementation fits with the bigger picture of the goal the team is trying to accomplish; the goal has been set before, when creating the backlog, and has been refined all the way from the original high-level strategy. In fact, the practice of having *the team deciding the backlog* provides a high level of

Autonomy for the team on the one hand, but is essentially bounded and informed by the intent and the decisions about strategy made on higher levels and communicated through briefing techniques, like meetings, blog posts and all-staff events. To that end, the practice supports a medium level of Alignment as it creates a need to consult with those decisions and consider them when deciding backlog items. Along the same lines, the *teams define their own scope and timeline*, which gives them a high level of Autonomy on the team level by not having to coordinate with other teams to make decisions. They do, however, need to consult with the communicated intent in order to define their scope within the boundaries of the strategy, and so the practice supports a medium level of Alignment.

Table B.4 shows the practices at the level of the organization, and their related levels of Autonomy and Alignment.

Reported practice	**Level of Autonomy**	**Level of Alignment**
Access to repositories is open	high	high
Micro-services architecture	high	high
Ubiquitous personas	high	high
Offsites notes are visible	high	high
There are ways to provide feedback on decisions on products	medium	high
Teams follow a pull-based workflow	high	medium
There are organizational roles to communicate product direction decisions to teams	medium	high

Table B.4: Practices on the level of the **organization** and their related levels of Autonomy and Alignment

The practice of *open access on repositories* is a practice that relates to high levels of both Autonomy and Alignment, because it provides freedom about the actions that teams can take. Teams can access and follow the information that they find is relevant to them, no more or no less. The fact that they don't have to ask for permission or approval means independence, while the easy access to information that is critical for teams to understand the big picture (such as rationale about business decisions) provides the ground for alignment. This practice works closely with the practice of providing ways to *provide feedback on decisions on products*, through the various

mechanisms I have already listed earlier. Providing feedback creates some need for coordination and, so, corresponds to medium levels of Autonomy. It is a technique that supports a high level of Alignment, however, by making the feedback and explanation of rationale visible so that, collectively, the organization moves toward the same mindset about what the goals are. The practice of *having organizational roles to communicate product decisions to the teams* moves in the same direction; it is a technique that provides high level of Alignment, although it creates need for coordination and bounds Autonomy.

The *use of a micro-services architecture* for some of the Atlassian products supports the Autonomy of the product and feature teams since they work on decentralized parts of the product. Such an architecture, by design, defines and shows how the separate parts work together and, so, supports a high level of Alignment. The *ubiquitous presence of personas* and the *transparency of the offsites notes* enable the teams to act independently, while still giving them the same point of reference and, therefore, correspond to high levels of both Autonomy and Alignment.

Finally, as mentioned earlier for the team level, teams follow a pull-based workfow. This makes whole teams independent from each other, but able to contribute fixes etc without needing approval or permission (high level of Autonomy). Alignment is supported at a medium level due to the need to consult (at least through reading the documentation) with the team about their goals and objectives to know how a particular fix contributes. What is more, a developer that is external to the team and contributes need to consider how their submitted work supports their product's fit with the higher level goal.

Table B.5 shows the practices that relate to work that is organized or communicated across teams, and their related levels of Autonomy and Alignment.

The transparency of *the progress and status or all teams' work* and *their mission*, are practices that support high levels of Alignment by making all parties aware of how their own work fits with the strategic objectives and goals; an example is that when describing features teams have to document how these link back to the VTFM. At the same time, given the transparency of the information, teams can work independently of one another while having the same point of reference, and hence the practices also support a high level of Autonomy. High levels of both Autonomy and Alignment are also supported by the practice of producing and using *templates and guidelines*. These ensure that although teams are working separately they follow the same instructions that frame their independent actions into a unified outcome.

Reported practice	**Level of Autonomy**	**Level of Alignment**
Progress and status of all teams' work is visible	high	high
Other teams' mission and progress is transparent	high	high
Teams give regular demos of their work in progress	medium	high
There are templates, design and coding guidelines available	high	high
There are regular synchronization opportunities between teams	medium	high
There are impromptu synchronization opportunities between teams	medium	high
Employees move around different teams and learn first hand how they work	medium	medium
Employees rotate roles and learn what are the responsibilities and pain points	medium	medium
Teams share their successes and failures with other teams	medium	high
There is cross-pollination in tools and processes between teams	medium	high
Decisions on key product directions are transparent	medium	high
Teams practice code peer review	medium	high

Table B.5: Practices relating to **working across teams** and their related levels of Autonomy and Alignment

We saw that teams have the opportunity and are encouraged to *share with the rest of the organization both their successes and failures*. This practice is supported by a number of mechanisms that range from various meetings and events, both for *regular and impromptu synchronization* between teams and *demos of ongoing work* that supports everyone maintaining awareness and seeing the link between objectives and tasks. *Cross-pollination in tools and processes* is an important practice that is further supported by the awareness and synchronization between all teams. These practices support a high level of Alignment by making the actions toward fulfilling objectives of other teams transparent, and informing decisions. At the same time, they provide a medium level of Autonomy because they occasionally create the need for coordination to arrange events and to take action on lessons learned through them. High Align-

ment is also supported by the fact that *the decisions on key product directions are transparent* and usually explained at length through a variety of mechanisms, helping teams see how their tasks will fit into the strategic goals. The practice supports a medium level of Autonomy since it creates some coordination needs for the team and other organizational roles (such as the program or product manager) to translate the key product direction to actionable objectives and tasks for the team.

Another form of cross-pollination happens organically when *employees are seconded to different teams* and when they *rotate roles within their team*. Both these practices support medium levels of Autonomy and Alignment; they create a need for teams to coordinate to have people from other teams placed with them, and the new members need to consult with the established members of their new team to know and understand the fit between the team's work and the higher level objectives. They are both useful mechanisms, however, for creating empathy between teams and between roles, as the pain points become clear and the less desirable tasks are shared.

Finally, as mentioned before, the development teams practice code peer review. When looking at this practice from the perspective of developers that want to contribute to a different team that the one they belong to, this practice creates a coordination need between the external developer and the team he is contributing to both because the external contributor will need to coordinate their action with the team's objectives and because there will be need to coordinate the peer review itself. At the same time this practice helps make clear how another team's work fits under the strategic objective, and hence supports a high level of Alignment.

B.7 Member checking survey

The member checking survey can be accessed at:
`https://www.surveymonkey.com/r/3QJPTR3`

Appendix C

Microsoft study: data collection instruments

C.1 Ethics Approval

University of Victoria
Office of Research Services | Human Research Ethics Board
Michael Williams Building Rm B202 PO Box 1700 STN CSC Victoria BC V8W 2Y2 Canada
T 250-472-4545 | F 250-721-8960 | ethics@uvic.ca | uvic.ca/research |

Certificate of Renewed Approval

PRINCIPAL INVESTIGATOR:	**Eirini Kalliamvakou**	**ETHICS PROTOCOL NUMBER:** Minimal Risk Review - Delegated	**14-338**
UVic STATUS:	**Ph.D. Student**	ORIGINAL APPROVAL DATE:	02-Oct-14
UVic DEPARTMENT:	**COSI**	RENEWED ON:	30-Aug-17
SUPERVISOR:	**Dr. Daniel German**	APPROVAL EXPIRY DATE:	01-Oct-18

PROJECT TITLE: **Building Healthy and Efficient Teams in Software Organizations**

RESEARCH TEAM MEMBER Members: Daniel German (UVic), Daniela Damian (UVic), Lloyd Montgomery (UVic)

DECLARED PROJECT FUNDING: **None**

CONDITIONS OF APPROVAL

This Certificate of Approval is valid for the above term provided there is no change in the protocol.

Modifications
To make any changes to the approved research procedures in your study, please submit a "Request for Modification" form. You must receive ethics approval before proceeding with your modified protocol.

Renewals
Your ethics approval must be current for the period during which you are recruiting participants or collecting data. To renew your protocol, please submit a "Request for Renewal" form before the expiry date on your certificate. You will be sent an emailed reminder prompting you to renew your protocol about six weeks before your expiry date.

Project Closures
When you have completed all data collection activities and will have no further contact with participants, please notify the Human Research Ethics Board by submitting a "Notice of Project Completion" form.

Certification

This certifies that the UVic Human Research Ethics Board has examined this research protocol and concluded that, in all respects, the proposed research meets the appropriate standards of ethics as outlined by the University of Victoria Research Regulations Involving Human Participants.

Dr. Rachael Scarth
Associate Vice-President Research Operations

Certificate Issued On: 30-Aug-17

14-338 Kalliamvakou, Eirini

C.2 Interview guides

C.2.1 Interview guide for developers

- ***Introduction***

  - Who we are and what the project is about

  - Reiterate non-affiliation with HR, confidentiality and that we are not interested in identifying individuals

  - Ask for permission to record

- ***Context***

  - How many years of professional experience? In what roles?

  - How many different companies have you worked in?

  - What do you do for Microsoft?

- ***Attributes and their importance***

  - In the short survey you pointed out X attribute as being important for an engineering manager to have. Can you tell us more about experiences or instances that led you to consider X characteristic important?

  - Which of these attributes do you think a good manager cannot lack?

  - Have your views changed at all during your career? In what way?

  - *<If the developer is a mature employee>* Have you had a change of heart? Have you seen your views about what you think makes a great software engineering manager change during your career? What were some of the experiences that made you change your mind?

- ***Link to retention***

  - Has there been a time you have felt strongly "I really want to stay in this team"? What has played a role?

  - Has there been a time you felt strongly about recommending to someone else to join a team? What played a role?

  - Has there been a time that you felt strongly "I want to leave this team"? What played a role?

  - Are there any of these attributes that a great manager cannot lack? If the manager didn't have attribute X what would you think about how good they are, how would it affect what you think about the team you are in?

- ***Link to the engineering of software***

  - Have there been any instances that your manager did something that helped

you get your job done? What was it?

- What are your expectations about how your manager should support you in getting your job done?

- Do you receive mentoring from your manager? In what form?

- How often do you have conversations about technical aspects of your work with your manager? What are your expectations about that?

- ***Wrap-up***

  - Is there anything else that you think we should be asking?

  - Is there anything that you would like to add?

  - Do you have any questions for us?

  - Would you like to be notified about the results of this project?

C.2.2 Interview guide for leads

- ***Introduction***

  - Who we are and what the project is about

  - Reiterate non-affiliation with HR, confidentiality and that we are not interested in identifying individuals

  - Ask for permission to record

- ***Context***

  - How many years of professional experience? In what roles?

  - How many different companies have you worked in?

  - What do you do for Microsoft?

- ***Attributes and their importance***

  - In the short survey you pointed out X attribute as being important for an engineering manager to have. Can you tell us more about experiences or instances that led you to consider X characteristic important?

  Responsibilities

  - How are your responsibilities as lead different than an IC's? What changed in your day-to-day work?

  - How are your responsibilities as lead different than a manager's? How have you divided your concerns?

  - What changed in your interaction with your manager when you became a lead?

  - What changed in your relationship with the team when you became a lead?

  - What is the distribution of time you devote to doing technical work versus

managing people?
- What kind of technical work do you do? How is it different from what an IC does?
- What aspect of leading the team do you take on versus the manager?
- What aspects of managing people do you handle?
- Do you mentor engineers? In what way?
- What was the most recent instance that you helped an engineer get their work done?
- How do you see the lead's and the manager's role in the engineers' core work (engineering software)?

Team health

- How do you gauge if the team is healthy? What are the elements of team health to you?
- Do you have any mechanisms or techniques to measure those? How?

Team health

- How did you become a lead? Did you prepare in some way? Did you get mentoring?
- Did you get any training on how to manage a team?
- What were your expectations going in? Anything that didn't work out that way?
- In trying to do well in this role, did you try out things that didn't work out? What were they?
- In trying to do well in this role, did you have to pick up any skills or knowledge that you didn't have before? Can you give examples?
- What advice would you give to a new lead?
- Has your experience as a lead changed your views on what attributes are important for a great software engineering manager?

Link to retention

- Is retention a lead's concern?
- Have you had experience with engineers leaving a team? What is usually the impact?
- How do you try to avoid having a situation where someone leaves the team?
- How do you recover from it? Are there any resilience mechanisms that you build in the team to recover faster from an exit?

Challenges

- Do you have any challenges in your role as lead that data science could help you with?

- ***Wrap-up***
  - Is there anything else that you think we should be asking?
  - Is there anything that you would like to add?
  - Do you have any questions for us?
  - Would you like to be notified about the results of this project?

C.2.3 Interview guide for engineering managers

- ***Introduction***
  - Who we are and what the project is about
  - Reiterate non-affiliation with HR, confidentiality and that we are not interested in identifying individuals
  - Ask for permission to record
- ***Context***
  - How many years of professional experience? In what roles?
  - How many different companies have you worked in?
  - What do you do for Microsoft?
- ***Attributes and their importance***
  - In the short survey you pointed out X attribute as being important for an engineering manager to have. Can you tell us more about experiences or instances that led you to consider X characteristic important?
- ***Freedom to the team***
  - What are decisions that you leave up to the team? Can you give us examples?
  - Where does the team get its direction from? What are your mechanisms for communicating that to them?
  - Is it important for the dev team to have buy in for what they have to do? How do you obtain that?
  - Can there be too much freedom? What happens if the team wants to go in a different direction?
  - Have there been tensions on that point? How was it resolved?
  - Do you find that you have to manage upwards? How do you do that?
  - How do you link team mission to organizational mission?
  - How do you link individual goals to organizational goals?

- ***Positive working relationship/woking environment***

  - Has there been a case that a developer came to you with an idea that in your experience would not work? How did you handle that?

  - How do you provide flexibility for work/life balance? What are experiences that showed you that this is important?

  - Do you approach engineers proactively to see how they are doing or are you letting them come to you when they need something or are having problems?

  - How do you coach engineers? What is a recent experience that you provided coaching to one of your team members?

- ***Switch to managerial role***

  - How were you selected for the managerial role?

  - How did you prepare for the managerial role? What training did you receive?

  - In your quest to fit into the managerial role, where there things you tried out that didn't work? Can you give examples?

  - Was the lead role helpful in preparing you for a managerial role?

  - What were things you had to learn or skills you had to acquire to perform the managerial role?

  - Have your views changes during your managerial experience about what you consider important attributes for a great manager to have?

  - Do you think that a good engineer makes a good engineering manager? How are the skills and competencies needed for the two different?

- ***Retention***

  - Is retention important? Does it come up as a concern for engineering teams in your experience? How do you deal with it?

  - What is different about having junior versus senior engineers on your team? What do you handle differently?

- ***Link to software engineering***

  - Has there been an instance you can remember that you did something that helped an engineer get their work done?

  - Do you provide mentoring? What kind?

  - Do you have technical discussions with the engineers? How often?

- ***Challenges***

  - What are the most important challenges that you have in your role as manager?

- ***Impressions***

- Which of these attributes do you think a good manager cannot lack?

- Gave you had a change of heart? Have you seen your views about what you think makes a great manager change during your career? What were some of the experiences that made you change your mind?

- Are there qualities that you see in other engineering managers that you wish you had?

- ***Wrap-up***

  - Is there anything else that you think we should be asking?
  - Is there anything that you would like to add?
  - Do you have any questions for us?
  - Would you like to be notified about the results of this project?

C.2.4 Interview guide for upper-level managers

- ***Introduction***

  - Who we are and what the project is about

  - Reiterate non-affiliation with HR, confidentiality and that we are not interested in identifying individuals

  - Ask for permission to record

- ***Context***

  - How many years of professional experience? In what roles?
  - How many different companies have you worked in?
  - What do you do for Microsoft?

- ***Attributes and their importance***

  - In the short survey you pointed out X attribute as being important for an engineering manager to have. Can you tell us more about experiences or instances that led you to consider X characteristic important?

  - Which of these attributes do you think a great manager cannot lack?

  - Have you seen your views about what you think makes a great software engineering manager change during your career? What were some of the experiences that made you change your mind?

- ***Recruiting an engineering manager***

  - When there is an engineering manager position to fill, where do you look for candidates?

  - Think about the last time you were looking to recruit an engineering manager,

what were the criteria you had in mind?

- What attributes are you looking for in developers that show you they are a good fit to become leads or managers? Why? How are these attributes helpful in managing a software engineering team?

- Once you have recruited a manager, what areas do you provide coaching on?

- How do you spot that someone needs support or help? How do you help them?

- ***Switch to managerial role***

  - How have you seen engineering managers transition in their role?

  - How do they prepare for the switch from an engineering role to a managerial one?

  - As they are figuring out their new role, have you seen people deal with challenges?

  - Was there a time that you advised someone when they made the switch from an engineering role to one with managerial responsibilities? What advice did you have for them?

  - Do you think that managing engineers is different than managing people in other fields of knowledge work? How?

- ***Link to retention***

  - Is retention important? Does it come up as a concern for engineering teams in your experience? How do you handle it?

  - Have you seen the attributes of managers be associated with retention in teams? In what way?

  - What have you seen in your experience are ways to help with retention in engineering teams?

  - Have you had experiences that showed you mechanisms to build in a team to make it more resilient to people exiting? What are they?

- ***Wrap-up***

  - Is there anything else that you think we should be asking?

  - Is there anything that you would like to add?

  - Do you have any questions for us?

  - Would you like to be notified about the results of this project?

C.3 Survey

The full survey, and the regression models can be found at `https://github.com/cabird/Engineering_Manager_Study`. For convenience, I am including the survey below:

Survey - The attributes of great software engineering managers

---

Your answers help us.

Thank you for agreeing to participate in this survey.

Microsoft Research is interested in uncovering the attributes of great software engineering managers. Our aim is to improve how managers attract, retain and grow talent in their engineering teams. Responses to this survey are private and confidential. Please click here to review the privacy statement. The survey takes about 10 minutes to complete.

This survey is anonymous - no personal information will be collected. Aggregated information may be shared with research collaborators outside of Microsoft and used in publications. We selected you as part of a sample of Microsoft employees based on your job role. If you have any questions about this research project, please contact t-eikall@microsoft.com

After completing the survey, you can enter a raffle for one of two $50 Amazon gift cards (official rules of the sweepstakes). Instructions for entering the raffle will be provided once you submit your response.

Thank you,
Eirini Kalliamvakou (T-EIKALL) and Christian Bird (CBIRD)

Demographic information

1) Which one of the following describes you best? (required)*

- You are an Individual Contributor
- You are an Engineering Lead

- ○ You are an Engineering Manager
- ○ Other - Write in: _____

Logic: Hidden unless: Question "Which one of the following describes you best? (required)" #1 is one of the following answers ("You are an Individual Contributor")

2) What role do you directly report to?

- ○ Engineering Lead
- ○ Engineering Manager
- ○ Other: _____

Logic: Hidden unless: Question "Which one of the following describes you best? (required)" #1 is one of the following answers ("You are an Engineering Lead","You are an Engineering Manager","Other - Write in")

3) What roles report to you? (select all that apply)

- ☐ Individual Contributors
- ☐ Engineering Leads
- ☐ Engineering Managers

Validation: Must be numeric

Logic: Hidden unless: Question "Which one of the following describes you best? (required)" #1 is one of the following answers ("You are an Engineering Lead","You are an Engineering Manager","Other - Write in")

4) How many people directly report to you (non-vendors)?

_____

Validation: Must be numeric

Logic: Hidden unless: Question "Which one of the following describes you best? (required)" #1 is one of the following answers ("You are an Engineering Manager","Other - Write in")

5) How many people in total report to you, directly and indirectly?

Validation: Must be numeric

Logic: Hidden unless: (Question "Which one of the following describes you best? (required)" #1 is one of the following answers ("You are an Individual Contributor") AND Question "What role do you directly report to?" #2 is one of the following answers ("Engineering Lead"))

6) How many people report to your Engineering Lead?

Validation: Must be numeric

Logic: Hidden unless: (Question "Which one of the following describes you best? (required)" #1 is one of the following answers ("You are an Individual Contributor") AND Question "What role do you directly report to?" #2 is one of the following answers ("Engineering Lead"))

7) How many people in total report to your skip level manager, directly *and* indirectly?

If you don't know exactly, please give a rough estimate.

Validation: Must be numeric

Logic: Hidden by default

How many people report directly to your skip level manager?

If you don't know exactly, please provide a rough estimate.

Validation: Must be numeric

Logic: Hidden unless: ((Question "Which one of the following describes you best? (required)" #1 is one of the following answers ("You are an Individual Contributor") AND Question "What role do you directly report to?" #2 is one of the following answers ("Engineering Manager")) OR Question "Which one of the following describes you best? (required)" #1 is one of the following answers ("You are an Engineering Lead"))

8) How many people in total report to your Engineering Manager, directly *and* indirectly?

If you don't know exactly, please provide a rough estimate.

Validation: Must be numeric

Logic: Hidden by default

How many people report to your Engineering Manager directly?

If you don't know exactly, please provide a rough estimate.

Validation: Must be numeric

Logic: Hidden unless: Question "Which one of the following describes you best? (required)" #1 is one of the following answers ("You are an Engineering Lead")

9) How many years in total have you been an Engineering Lead?

Validation: Must be numeric

Logic: Hidden unless: Question "Which one of the following describes you best? (required)" #1 is one of the following answers ("You are an Engineering Manager")

10) How many years in total have you been an Engineering Manager?

Validation: Must be numeric

Logic: Hidden unless: Question "Which one of the following describes you best? (required)" #1 is one of the following answers ("You are an Individual Contributor")

11) How many years in total have you been a professional software engineer?

Validation: Must be numeric

12) How many years have you worked at Microsoft? (required)*

13) In what location do you work? (required)*

- North America: USA - WA (Puget Sound regions: Redmond, Bellevue, Seattle, Sammamish, etc.)
- North America: USA - Silicon Valley, CA
- North America: USA - Other
- North America: Canada, Mexico
- Central America and South America
- Europe
- Asia: China
- Asia: India
- Asia: Middle East
- Asia: Other
- Australia, New Zealand, Oceania
- Africa
- Other:

Validation: Must be numeric

14) How old are you? (enter in years)

15) What is your gender identity? (required)*

- Female
- Male
- Other - Write in:
- I prefer not to say

The following questions ask specifically about the role of the Engineering Manager.

16) Below are 15 attributes that interviews with developers and managers indicated make great engineering managers -- but, of course, opinions vary. What do you think, based on your experience?
Under each attribute you can find examples of the engineering manager demonstrating that attribute.

Please rate how important each attribute is for distinguishing a great engineering manager from a good one, ranging from "critical" to "not important".

A great engineering manager...*

	Critical									Not important
mediates inter-team interaction Acting as a buffer with other teams/disciplines/managers, negotiating what her team can provide when, and mediating her own team's requests to other teams.										

grows talent Providing opportunities for challenging work, suggesting training for the engineer to gain industry relevant skills, and providing actionable feedback to improve engineer performance.	○	○	○	○	○	○	○	○	○	○
is available Signalling themselves as approachable, and devoting time to the engineer when needed.	○	○	○	○	○	○	○	○	○	○
guides the team Coaching engineers on quality aspects (e.g. scalability), providing guidance through appropriate questions to engineers struggling with tasks, and helping the engineer build judgement.	○	○	○	○	○	○	○	○	○	○
protects developer flow Shielding the engineer from randomization, removing distractions and blockers, and helping to resolve issues or conflicts.	○	○	○	○	○	○	○	○	○	○
inspires the team Viewed as a leader, responding to situations individually rather than having general approaches, and demonstrating passion about their work, the team, and the company.	○	○	○	○	○	○	○	○	○	○
supports experimentation	○	○	○	○	○	○	○	○	○	○

Encouraging the engineer to try out new things, and signalling a safe environment for unsuccessful attempts.										
drives alignment Sharing information about higher level context, explaining the business intent for the product/service, creating a mission with input from the team, and setting clear goals for the trajectory.										
enables autonomy Providing freedom on how engineers do their work, showing trust and support for their decisions, and helping engineers be independently responsible.										
cultivates fairness Showing appreciation for the engineer's contributions, holding themselves accountable for the team's progress, and recognizing value publicly while correcting the engineer privately.										
builds a relationship with team members Taking an interest in the employees' life outside work, discussing their general interests, and caring about them as a person.										

maintains a positive working environment Providing flexibility to balance work and personal life, energizing the team through organizing events, celebrating team successes, and ensuring good morale.										
recognizes individuality Understanding each engineer's strengths and weaknesses and finding fitting tasks for them, valuing diverse perspectives in the team, and fine tuning the definition of success to each individual's talents and interests.										
builds team culture Demonstrating the rules, norms, and habits of the team, creating "what this team believes in" with input from members, and facilitating an environment of shared success and responsibility.										
is technical Being knowledgeable about the system and technologies the engineer is working with, understanding the complexity or problems and solutions, and having input for design dilemmas.										

17) If you think there are attributes of great engineering managers which we missed above, please enter them below.

18) Imagine you are responsible for recruiting an engineering manager for team X, and you are down to your final two candidates:

1) Person A, a brilliant engineer with an impressive track record. They have expressed interest in furthering their career by becoming an Engineering Manager. Person A has been responsible for small feature teams as an Engineering Lead in the past. The teams they have managed so far have performed great. The feedback from Person A's directs has been reserved at best, highlighting that team members have felt awkward around them and that their social skills need improvement.

2) Person B, an enthusiastic Engineering Lead with a history of becoming a linchpin for all the teams they have worked with. They have brought to your attention lots of articles on leading and growing teams, and best people management practices. The teams they have managed so far have performed great. The feedback from their directs has been that they consider Person B an inspiring figure, but have had little input from them on technical aspects of the team's work.

Which of the two candidates would you choose?

- Person A, because:
- Person B, because:

19) Please rate your level of agreement with the statements about engineering managers below:

	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
If I heard of a team in another company with a great engineering manager I would consider moving there	○	○	○	○	○
An engineering manager that is technical is respected by their team	○	○	○	○	○
The engineering manager influences the success of the team	○	○	○	○	○
The engineering manager should be the person that makes the technical decisions in the team	○	○	○	○	○
The engineering manager should understand engineering,	○	○	○	○	○

but not practice it					
The most important goal for an engineering manager should be to drive execution excellence					
The engineering manager should be the mentor for the engineers					
Delegation is a way for an engineering manager to avoid performing certain tasks					
The most important goal for an engineering manager should be talent retention in their team					
The engineering manager should guide the team by providing solutions to coding problems					

The primary concerns of engineering managers should be business-related	○	○	○	○	○
The engineering manager should have insight on discussions about alternative options for design or implementation	○	○	○	○	○

20) Please rate your level of agreement with the statements about engineering managers below:

	Strongly agree	**Agree**	**Neutral**	**Disagree**	**Strongly disagree**
Software engineers are more engaged if the engineering manager cultivates a fun atmosphere in the team	○	○	○	○	○
If I heard of a team with a great engineering manager I	○	○	○	○	○

would consider moving there					
The engineering manager should provide rationale for thinking about problems, rather than provide solutions					
An engineering manager that is not technical is not respected by the team					
The most important goal for an engineering manager should be explaining the merit of a team's mission to its members					
The engineering manager influences the quality of the code					

The primary concerns of engineering managers should not be technical					
Delegation is a signal that the engineering manager trusts team members					
The engineering manager should be data-driven in all their discussions					
The engineering manager should be the shield from randomization					
The engineering manager should guide the team by asking questions					
The most important goal for an engineering manager should be making					

engineers happy					

21) Please rate your level of agreement with the statements about engineering managers below:

	Strongly agree	**Agree**	**Neutral**	**Disagree**	**Strongly disagree**
The engineering manager should be the person that software engineers go to for technical guidance	○	○	○	○	○
The most important goal for an engineering manager should be to generate business impact	○	○	○	○	○
The engineering manager should be the team's best advocate	○	○	○	○	○
The engineering manager	○	○	○	○	○

should coach engineers in the team to be self-sufficient					
The most important goal for an engineering manager should be to clearly explain the desired outcomes for the team	○	○	○	○	○
I would leave the company if I didn't like my engineering manager	○	○	○	○	○
The primary concern of engineering managers should be growing a healthy team	○	○	○	○	○
The engineering manager should enable, rather than direct	○	○	○	○	○

Delegation is a way for the engineering manager to empower team members by giving them ownership over what they do					
The engineering manager should contribute code to the project regularly					
The engineering manager should hold back their technical opinion unless asked					
The engineering manager should clearly state their technical opinion to the team					

22) Please rate your level of agreement with the statements about engineering managers below:

	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
The engineering manager should be the person that software engineers go to for coding help	○	○	○	○	○
Software engineers are more productive when their engineering manager takes an interest in their life outside work	○	○	○	○	○
The engineering manager should leave implementation decisions to the engineering team	○	○	○	○	○
The most important goal for an engineering manager should be to grow engineers	○	○	○	○	○
The engineering	○	○	○	○	○

manager should play a critical role in the motivation of the team					
Empathy for people is an important trait for an engineering manager					
The engineering manager should coach engineers in the team to share effort, successes and failures					
I would leave my team if I didn't like my engineering manager					
The engineering manager influences the success of the product					
The most important goal for an engineering manager should be to teach decision making to engineers					

The engineering manager influences the productivity of the team					

Wrap-up

23) Please rank the following factors that affect retention in an engineering team, from most important to least important.

The level of pay

The relationship with the engineering manager

The relationship between team members

How interesting the work is to engineers

The opportunity for career advancement

The working environment

24) How can a great software engineering manager have a positive or negative impact on the software his/her team produces?

After clicking "Submit" you will see instructions on how to enter the raffle for the gift cards. Good luck!

Thank You!

Thank you for taking our survey. Your response is very important to us.

As another way of saying thanks, we're raffling off two $50 Amazon.com Gift Certificates (official rules of the sweepstakes).
Click here to enter the raffle by email

Bibliography

[1] The effects of organizational climate on managerial job performance and job satisfaction. *Organizational Behavior and Human Performance*, 9(1):126 – 146, 1973.

[2] Artifacts for the Engineering Manager Study. `https://github.com/cabird/Engineering_Manager_Study`, 2017.

[3] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications*, 7(1):39–59, 1994.

[4] F. Abbattista, F. Calefato, D. Gendarmi, and F. Lanubile. Incorporating social software into distributed agile development environments. In *Automated Software Engineering - Workshops, 2008. ASE Workshops 2008. 23rd IEEE/ACM International Conference on*, pages 46–51, Sept 2008.

[5] Lougie Anderson, Glen B. Alleman, Kent Beck, Joe Blotner, Ward Cunningham, Mary Poppendieck, and Rebecca Wirfs-Brock. Agile management - an oxymoron?: Who needs managers anyway? In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '03, pages 275–277, New York, NY, USA, 2003. ACM.

[6] Jorge Aranda. *A theory of shared understanding for software organizations*. PhD thesis, University of Toronto, 2010.

[7] Benjamin M Artz, Amanda H Goodall, and Andrew J Oswald. Boss competence and worker well-being. *ILR Review*, page 0019793916650451, 2016.

[8] Sanjiv Augustine, Bob Payne, Fred Sencindiver, and Susan Woodcock. Agile project management: Steering from the edges. *Commun. ACM*, 48(12):85–89, December 2005.

[9] Bruce J Avolio, Weichun Zhu, William Koh, and Puja Bhatia. Transformational leadership and organizational commitment: Mediating role of psychological empowerment and moderating role of structural distance. *Journal of organizational behavior*, 25(8):951–968, 2004.

[10] Earl T. Barr, Christian Bird, Peter C. Rigby, Abram Hindle, Daniel M. German, and Premkumar Devanbu. *Cohesive and Isolated Development with Branches*, volume 7212 of *Lecture Notes in Computer Science*, pages 316–331. Springer Berlin Heidelberg, 2012.

[11] Victor R Basili, Richard W Selby, and David H Hutchens. Experimentation in software engineering. *IEEE Transactions on software engineering*, (7):733–743, 1986.

[12] Janet B Bavelas. Quantitative versus qualitative. *Social approaches to communication*, pages 49–62, 1995.

[13] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. *Manifesto for Agile Software Development*. Online under http://agilemanifesto.org/; retrieved February 25th 2016), 2001.

[14] R Beck and J Harter. Why great managers are so rare. `http://www.gallup.com/businessjournal/167975/why-great-managers-rare.aspx`, March 2014.

[15] Sarah Beecham, Nathan Baddoo, Tracy Hall, Hugh Robinson, and Helen Sharp. Motivation in software engineering: A systematic literature review. *Information and software technology*, 50(9):860–878, 2008.

[16] Andrew Begel and Nachiappan Nagappan. Usage and perceptions of agile software development in an industrial context: An exploratory study. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 255–264. IEEE, 2007.

[17] Andrew Begel, Nachiappan Nagappan, Christopher Poile, and Lucas Layman. Coordination in large-scale software teams. In *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, CHASE '09, pages 1–7, Washington, DC, USA, 2009. IEEE Computer Society.

[18] Mary J Benner and Michael Tushman. Process management and technological innovation: A longitudinal study of the photography and paint industries. *Administrative science quarterly*, 47(4):676–707, 2002.

[19] Mary J Benner and Michael L Tushman. Exploitation, exploration, and process management: The productivity dilemma revisited. *Academy of management review*, 28(2):238–256, 2003.

[20] Komal Khalid Bhatti and Tahir Masood Qureshi. Impact of employee participation on job satisfaction, employee commitment and employee productivity. *International Review of Business Research Papers*, 3(2):54–68, 2007.

[21] Frank Blackler. Knowledge, knowledge work and organizations: An overview and interpretation. *Organization studies*, 16(6):1021–1046, 1995.

[22] Barry Boehm and Richard Turner. Management challenges to implementing agile processes in traditional development organizations. *IEEE software*, 22(5):30–39, 2005.

[23] Will Bourne. 2 reasons to keep an eye on github. `http://www.inc.com/magazine/201303/will-bourne/2-reasons-to-keep-an-eye-on-github.html`, 2013.

[24] Ian Brace. *Questionnaire design: How to plan, structure and write survey material for effective market research*. Kogan Page Publishers, 2008.

[25] Nicky Britten, Rona Campbell, Catherine Pope, Jenny Donovan, Myfanwy Morgan, and Roisin Pill. Using meta ethnography to synthesise qualitative research: a worked example. *Journal of health services research & policy*, 7(4):209–215, 2002.

[26] Frederick P. Brooks, Jr. *The mythical man-month (anniversary ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[27] Steven P Brown and Robert A Peterson. Antecedents and consequences of salesperson job satisfaction: Meta-analysis and assessment of causal effects. *Journal of marketing research*, 30(1):63, 1993.

[28] Stephen Bungay. *The Art of Action: How Leaders Close the Gaps between Plans, Actions, and Results*. Nicholas Brealey Publishing, 2011.

[29] Daniel M Cable and Timothy A Judge. Person–organization fit, job choice decisions, and organizational entry. *Organizational behavior and human decision processes*, 67(3):294–311, 1996.

[30] Lan Cao, Kannan Mohan, Peng Xu, and Balasubramaniam Ramesh. A framework for adapting agile development methodologies. *European Journal of Information Systems*, 18(4):332–343, 2009.

[31] Lan Cao and Balasubramaniam Ramesh. Agile software development: Ad hoc practices or sound principles? *IT professional*, 9(2):41–47, 2007.

[32] Marcelo Cataldo and Kate Ehrlich. The impact of communication structure on new product development outcomes. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 3081–3090, New York, NY, USA, 2012. ACM.

[33] Marcelo Cataldo and James D. Herbsleb. Coordination breakdowns and their impact on development productivity and software failures. *IEEE Trans. Software Eng.*, 39(3):343–360, 2013.

[34] Marcelo Cataldo, James D. Herbsleb, and Kathleen M. Carley. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *Proc. of the 2nd ACM-IEEE Int. symposium on Empirical software engineering and measurement*, pages 2–11. ACM, 2008.

[35] Thomas Chau and Frank Maurer. *Logic versus Approximation: Essays Dedicated to Michael M. Richter on the Occasion of his 65th Birthday*, chapter Knowledge Sharing in Agile Software Teams, pages 173–183. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[36] Valery Chirkov, Richard M Ryan, Youngmee Kim, and Ulas Kaplan. Differentiating autonomy from individualism and independence: a self-determination theory perspective on internalization of cultural orientations and well-being. *Journal of personality and social psychology*, 84(1):97, 2003.

[37] Alistair Cockburn and Jim Highsmith. Agile software development: The people factor. *IEEE Computer*, 34(11):131–133, 2001.

[38] Kieran Conboy, Sharon Coyle, Xiaofeng Wang, and Minna Pikkarainen. People over process: Key challenges in agile development. *IEEE Software*, 28(4):48, 2011.

[39] M.E. Conway. How do committees invent. *Datamation*, 14(4):28–31, 1968.

[40] J.M. Corbin and A.L. Strauss. *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Sage Publications, 3rd edition, 2008.

[41] T. R. Coupe and M. N. Onodu. An empirical evaluation of the impact of case on developer productivity and software quality. *Journal of Information Technology*, 11(2):173–181, 1996.

[42] Stuart Crainer. *The management century: A critical review of 20th century thought and practice*. Jossey-Bass, 2000.

[43] John W Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, 2013.

[44] Michael Crotty. *The foundations of social research: Meaning and perspective in the research process*. Sage, 1998.

[45] Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. Free/libre open-source software development: What we know and what we do not know. *ACM Comput. Surv.*, 44(2):7:1–7:35, March 2008.

[46] Kevin Crowston, Kangning Wei, Qing Li, U. Yeliz Eseryel, and James Howison. Coordination of free/libre open source software development. In David E. Avison and Dennis F. Galletta, editors, *ICIS*. Association for Information Systems, 2005.

[47] Kevin Crowston, Kangning Wei, Qing Li, U. Yeliz Eseryel, and James Howison. Self-organization of teams in free/libre open source software development. *Information and Software Technology Journal: Special issue on Understanding the Social Side of Software Engineering, Qualitative Software Engineering Research*, 49:564–575, 2007.

[48] Mihaly Csikszentmihalyi. *Flow: The Psychology of Optimal Experience*. Harper Perennial, New York, NY, March 1991.

[49] Thomas G Cummings. Self-regulating work groups: A socio-technical synthesis. *Academy of management Review*, 3(3):625–634, 1978.

[50] Bill Curtis, Herb Krasner, and Neil Iscoe. A field study of the software design process for large systems. *Commun. ACM*, 31(11):1268–1287, November 1988.

[51] Laura Dabbish, Colleen Stuart, Jason Tsay, and James Herbsleb. Leveraging transparency. *IEEE Software*, 30(1):37–43, 2013.

[52] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, CSCW '12, pages 1277–1286. ACM, 2012.

[53] Barthélémy Dagenais, Harold Ossher, Rachel KE Bellamy, Martin P Robillard, and Jacqueline P De Vries. Moving into a new software project landscape. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 275–284. ACM, 2010.

[54] Thomas H. Davenport, Robert J. Thomas, and Susan Cantrell. The mysterious art and science of Knowledge-Worker performance. *MIT Sloan management review*, 44(1):23–30, 2002.

[55] Chris DiBona and Sam Ockman. *Open sources: Voices from the open source revolution.* " O'Reilly Media, Inc.", 1999.

[56] Kim Dikert, Maria Paasivaara, and Casper Lassenius. Challenges and success factors for large-scale agile transformations: A systematic literature review. *Journal of Systems and Software*, 2016.

[57] Torgeir Dingsøyr and Nils Brede Moe. Research challenges in large-scale agile software development. *SIGSOFT Softw. Eng. Notes*, 38(5):38–39, August 2013.

[58] Mary Dixon-Woods, Shona Agarwal, David Jones, Bridget Young, and Alex Sutton. Synthesising qualitative and quantitative evidence: a review of possible methods. *Journal of health services research & policy*, 10(1):45–53, 2005.

[59] Mary Dixon-Woods, Rachel L Shaw, Shona Agarwal, and Jonathan A Smith. The problem of appraising qualitative research. *Quality and Safety in Health Care*, 13(3):223–225, 2004.

[60] Rory Donnelly. How ?free? is the free worker? an investigation into the working arrangements available to knowledge workers. *Personnel Review*, 35(1):78–97, 2006.

[61] Peter Ferdinand Drucker. *Management challenges for the 21st century.* Routledge, 2007.

[62] Meghann Drury, Kieran Conboy, and Ken Power. Obstacles to decision making in agile software development teams. *Journal of Systems and Software*, 85(6):1239–1254, 2012.

[63] Nicolas Ducheneaut. Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work (CSCW)*, 14(4):323–368, 2005.

[64] Charles Duhigg. What google learned from its quest to build the perfect team. *The New York Times Magazine*, 2016.

[65] Carol Dweck. *Mindset: The new psychology of success.* Random House, 2006.

[66] Tore Dyba. Improvisation in small software organizations. *IEEE Software*, 17(5):82–87, 2000.

[67] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer, 2008.

[68] Charles Ehin. Un-managing knowledge workers. *Journal of Intellectual Capital*, 9(3):337–350, 2008.

[69] Margaret S Elliott and Walt Scacchi. Free software development: cooperation and conflict in. *Free/open source software development*, 152, 2005.

[70] Margaret S Elliott and Walt Scacchi. Mobilization of software developers: the free software movement. *Information Technology & People*, 21(1):4–33, 2008.

[71] J. Alberto Espinosa, Robert E. Kraut, Kogod School Of Business, and Theme Organization. Shared mental models, familiarity and coordination; a multi-method study of distributed software teams. In *Intern. Conf. Information Systems*, pages 425–433, 2002.

[72] Carole A Estabrooks, Peggy Anne Field, and Janice M Morse. Aggregating qualitative findings: an approach to theory development. *Qualitative Health Research*, 4(4):503–511, 1994.

[73] Brian Fitzgerald, Gerard Hartnett, and Kieran Conboy. Customising agile methods to software practices at intel shannon. *European Journal of Information Systems*, 15(2):200–213, 2006.

[74] Brian Fitzgerald and Klaas-Jan Stol. Continuous software engineering and beyond: trends and challenges. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, pages 1–9. ACM, 2014.

[75] Brian Fitzgerald and Klaas-Jan Stol. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123:176–189, 2017.

[76] Brian Fitzgerald, Klaas-Jan Stol, Ryan O'Sullivan, and Donal O'Brien. Scaling agile methods to regulated environments: An industry case study. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 863–872, Piscataway, NJ, USA, 2013. IEEE Press.

[77] Bent Flyvbjerg. Five misunderstandings about case-study research. *Qualitative inquiry*, 12(2):219–245, 2006.

[78] Floyd J Fowler Jr. *Survey research methods*. Sage publications, 2013.

[79] Andy Friedman. Responsible autonomy versus direct control over the labour process. *Capital Class*, 1(1):43–57, 1977.

[80] Gartner. Magic quadrant for application development life cycle management. `http://www.gartner.com/technology/reprints.do?id=1-2A61Y68&ct=150218&st=sb`, February 2015.

[81] David Garvin, Alison Berkley Wagonfeld, and Liz Kind. Google's Project Oxygen: Do Managers Matter? Technical report, Harvard Business School Case 313-110, 01 2013.

[82] David A. Garvin. How google sold its engineers on management. *Harvard business review*, 91(12):74–82, December 2013.

[83] Gerard George, Randall G Sleeth, and Mark A Siders. Organizing culture: Leader roles, behaviors, and reinforcement mechanisms. *Journal of Business and Psychology*, 13(4):545–560, 1999.

[84] Georgios Gousios, Martin Pinzger, and Arie van Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 345–355, New York, NY, USA, 2014. ACM.

[85] David E Gray. *Doing research in the real world*. Sage, 2013.

[86] Gina C. Green, Alan R. Hevner, and Rosann Webb Collins. The impacts of quality and productivity perceptions on the use of software process improvement innovations. *Information and Software Technology*, 47(8):543 – 553, 2005.

[87] Chris Groscurth. Great managers can fix broken performance management systems. `http://www.gallup.com/businessjournal/183770/great-managers-fix-broken-performance-management-systems.aspx`, June 2015.

[88] EG Guba. The alternative paradigm dialog. in. eg guba (ed.) the paradigm dialogue (p. 17-27), 1990.

[89] Vijay K. Gurbani, Anita Garvert, and James D. Herbsleb. A case study of open source tools and practices in a commercial setting. *SIGSOFT Softw. Eng. Notes*, 30(4):1–6, May 2005.

[90] Carl Gutwin, Reagan Penner, and Kevin Schneider. Group awareness in distributed software development. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, CSCW '04, pages 72–81, New York, NY, USA, 2004. ACM.

[91] Anja Guzzi, Alberto Bacchelli, Michele Lanza, Martin Pinzger, and Arie van Deursen. Communication in open source software development mailing lists. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 277–286, Piscataway, NJ, USA, 2013. IEEE Press.

[92] Martine R Haas. The double-edged swords of autonomy and external knowledge: Analyzing team effectiveness in a multinational organization. *Academy of Management Journal*, 53(5):989–1008, 2010.

[93] J. Richard Hackman and Greg R. Oldham. Development of the Job Diagnostic Survey. *Journal of Applied Psychology*, 60(2):159–170, April 1975.

[94] J Richard Hackman and Greg R Oldham. Development of the job diagnostic survey. *Journal of Applied psychology*, 60(2):159, 1975.

[95] James K. Harer, Frank L. Schmidt, and Theodore L. Hayes. Business-unit-level relationship between employee satisfaction, employee engagement, and business outcomes: A meta-analysis. *Journal of Applied Psychology*, 87(2):268–279, 2002.

[96] W. Harrison. Eating your own dog food. *Software, IEEE*, 23(3):5–7, May 2006.

[97] James K Harter, Frank L Schmidt, James W Asplund, Emily A Killham, and Sangeeta Agrawal. Causal impact of employee work perceptions on the bottom line of organizations. *Perspectives on Psychological Science*, 5(4):378–389, 2010.

[98] James K Harter, Frank L Schmidt, and Corey LM Keyes. Well-being in the workplace and its relationship to business outcomes: A review of the gallup studies. *Flourishing: Positive psychology and the life well-lived*, 2:205–224, 2003.

[99] James D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *FOSE '07: 2007 Future of Software Engineering*, pages 188–198, Washington, DC, USA, 2007. IEEE Computer Society.

[100] James D. Herbsleb and Rebecca E. Grinter. Splitting the organization and integrating the code: Conway's law revisited. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 85–95, New York, NY, USA, 1999. ACM.

[101] Jim Highsmith and Alistair Cockburn. Agile software development: The business of innovation. *IEEE Computer*, 34(9):120–122, 2001.

[102] Rashina Hoda, James Noble, and Stuart Marshall. Self-organizing roles on agile software development teams. *IEEE Transactions on Software Engineering*, 39(3):422–444, 2013.

[103] Frank M Horwitz, Chan Teng Heng, and Hesan Ahmed Quazi. Finders, keepers? attracting, motivating and retaining knowledge workers. *Human resource management journal*, 13(4):23–44, 2003.

[104] Andrea Howell, Andrea Kirk-Brown, and Brian K Cooper. Does congruence between espoused and enacted organizational values predict affective commitment in australian organizations? *The International Journal of Human Resource Management*, 23(4):731–747, 2012.

[105] Chris Huxham and Siv Vangen. Ambiguity, complexity and dynamics in the membership of collaboration. *Human Relations*, 53(6):771–806, 2000.

[106] Irum Inayat, Siti Salwah Salim, Sabrina Marczak, Maya Daneva, and Shahaboddin Shamshirband. A systematic literature review on agile requirements engineering practices and challenges. *Computers in human behavior*, 51:915–929, 2015.

[107] Brian D. Janz, James C. Wetherbe, Gordon B. Davis, and Raymond A. Noe. Reengineering the systems development process: The link between autonomous teams and business process outcomes. *J. of Management Information Systems*, 14(1):41–68, 1997.

[108] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[109] Eirini Kalliamvakou, Daniela Damian, Kelly Blincoe, Leif Singer, and Daniel M German. Open source-style collaborative development practices in commercial projects using github. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 574–585. IEEE Press, 2015.

[110] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. An in-depth study of the promises and perils of mining github. *Empirical Software Engineering*, 21(5):2035–2071, 2016.

[111] Bakhtiar Khan Kasi and Anita Sarma. Cassandra: Proactive conflict minimization through optimized task scheduling. In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 732–741. IEEE, 2013.

[112] Petri Kettunen and Maarit Laanti. Combining agile software projects and large-scale organizational agility. *Software Process: Improvement and Practice*, 13(2):183–193, 2008.

[113] Walter Kiechel. The management century. *Harvard business review*, 90(11):62–75, 2012.

[114] Barbara A Kitchenham and Shari L Pfleeger. Personal opinion surveys. In *Guide to Advanced Empirical Software Engineering*, pages 63–92. Springer, 2008.

[115] Eric Knorr. Github's new ceo: We're serious about the enterprise. `http://www.infoworld.com/t/application-development/githubs-new-ceo-were-serious-about-the-enterprise-249524`, 2014.

[116] AMY L. KRISTOF. Person-organization fit: An integrative review of its conceptualizations, measurement, and implications. *Personnel Psychology*, 49(1):1–49, 1996.

[117] Adam Kuper. *The social science encyclopedia*. Routledge, 2013.

[118] Irwin Kwan, Marcelo Cataldo, and Daniela Damian. Conway's law revisited: The evidence for a task-based perspective. *IEEE software*, 29(1):90–93, 2012.

[119] Lina Lagerberg, Tor Skude, Pär Emanuelsson, Kristian Sandahl, and Daniel Ståhl. The impact of agile principles and practices on large-scale software development projects: A multiple-case study of two projects at ericsson. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 348–356. IEEE, 2013.

[120] Karim Lakhani and Robert G Wolf. Why hackers do what they do: Understanding motivation and effort in free/open source software projects. 2003.

[121] Shun Yin Lam, Venkatesh Shankar, M. Krishna Erramilli, and Bvsan Murthy. Customer value, satisfaction, loyalty, and switching costs: An illustration from a business-to-business service context. *Journal of the Academy of Marketing Science*, 32(3):293–311, 2004.

[122] F. Lanubile, C. Ebert, R. Prikladnicki, and A. Vizcaino. Collaboration tools for global software engineering. *Software, IEEE*, 27(2):52–55, 2010.

[123] Craig Larman. *Agile and iterative development: a manager's guide*. Addison-Wesley Professional, 2004.

[124] Heather K Spence Laschinger, Joan E Finegan, Judith Shamian, and Piotr Wilk. A longitudinal analysis of the impact of workplace empowerment on work satisfaction. *Journal of Organizational Behavior*, 25(4):527–545, 2004.

[125] Gwanhoo Lee and Weidong Xia. Toward agile: an integrated analysis of quantitative and qualitative field data on software development agility. *Mis Quarterly*, 34(1):87–114, 2010.

[126] Dean Leffingwell. *Scaling software agility: best practices for large enterprises*. Pearson Education, 2007.

[127] Josh Lerner and Jean Tirole. Some simple economics of open source. *The journal of industrial economics*, 50(2):197–234, 2002.

[128] Paul Luo Li, Andrew J. Ko, and Jiamin Zhu. What makes a great software engineer? In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 700–710, Piscataway, NJ, USA, 2015. IEEE Press.

[129] William Lidwell, Kritina Holden, and Jill Butler. *Universal Principles of Design*. Rockport Publishers, October 2003.

[130] Yvonna S Lincoln, Susan A Lynham, and Egon G Guba. Paradigmatic controversies, contradictions, and emerging confluences, revisited. *The Sage handbook of qualitative research*, 4:97–128, 2011.

[131] Mikael Lindvall, Vic Basili, Barry Boehm, Patricia Costa, Kathleen Dangle, Forrest Shull, Roseanne Tesoriero, Laurie Williams, and Marvin Zelkowitz. Empirical findings in agile methods. In *Conference on Extreme Programming and Agile Methods*, pages 197–207. Springer, 2002.

[132] Mark S Litwin. *How to measure survey reliability and validity*, volume 7. Sage Publications, 1995.

[133] Xiaojing Liu, Richard J Magjuka, and Seung-hee Lee. An examination of the relationship among structure, trust, and conflict management styles in virtual teams. *Performance improvement quarterly*, 21(1):77–93, 2008.

[134] Wayne G. Lutters and Carolyn Seaman. The value of war stories in debunking the myths of documentation in software maintenance. *Information and Software Technology*, 49(6):576–587, 2007.

[135] Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Comput. Surv.*, 26(1):87–119, 1994.

[136] Arthur B Markman. Constraints on analogical inference. *Cognitive science*, 21(4):373–418, 1997.

[137] Jennifer Marlow, Laura Dabbish, and Jim Herbsleb. Impression formation in online peer production: Activity traces and personal profiles in github. In *Proceedings of CSCW '13*, pages 117–128, New York, NY, USA, 2013. ACM.

[138] Antonio Martini, Lars Pareto, and Jan Bosch. Communication factors for speed and reuse in large-scale agile software development. In *Proceedings of the 17th international software product line conference*, pages 42–51. ACM, 2013.

[139] Abraham Maslow. A Theory of Human Motivation. *Psychological Review*, 50:370–396, 1943.

[140] Jeremy McAnally. This week on github: In good company. `http://www.linux-mag.com/id/7348/`, 2009.

[141] Douglas McGregor. *The Human Side of the Enterprise*. McGraw-Hill, New York, 1960.

[142] Sanjay Menon. Employee empowerment: An integrative psychological approach. *Applied Psychology*, 50(1):153–180, 2001.

[143] Sharan Merriam. *Qualitative research: A guide to design and implementation*. John Wiley & Sons, 2009.

[144] André N. Meyer, Thomas Fritz, Gail C. Murphy, and Thomas Zimmermann. Software developers' perceptions of productivity. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 19–29, New York, NY, USA, 2014. ACM.

[145] Yukio Miyazaki and Kuniaki Mori. Cocomo evaluation and tailoring. In *Proceedings of the 8th International Conference on Software Engineering*, ICSE '85, pages 292–299, Los Alamitos, CA, USA, 1985. IEEE Computer Society Press.

[146] Ludmila Mládková. Knowledge workers and the principle of 3s (self-management, self-organization, self-control). *Procedia-Social and Behavioral Sciences*, 181:178–184, 2015.

[147] Nils Brede Moe, Torgeir Dingsøyr, and Tore Dybå. Understanding self-organizing teams in agile software development. In *19th Australian Conference on Software Engineering (aswec 2008)*, pages 76–85. IEEE, 2008.

[148] Nils Brede Moe, Torgeir Dingsøyr, and Tore Dybå. Overcoming barriers to self-management in software teams. *IEEE software*, 26(6):20–26, 2009.

[149] Nils Brede Moe, Torgeir Dingsyr, and Tore Dyb. A teamwork model for understanding an agile team: A case study of a scrum project. *Information and Software Technology*, 52(5):480 – 491, 2010.

[150] Gareth Morgan. *Images of Organization*. Sage Publications Inc., United States of America, updated ed. edition, 2006.

[151] Palmer Morrel-Samuels. Getting the truth into workplace surveys. *Harvard business review*, 80(2):111–118, 2002.

[152] Sebastian C Müller and Thomas Fritz. Stuck and frustrated or in flow and happy: Sensing developers' emotions and progress. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 1, pages 688–699. IEEE, 2015.

[153] Emerson Murphy-Hill and Gail C. Murphy. Peer interaction effectively, yet infrequently, enables programmers to discover new tools. In *Proceedings of the ACM 2011 Conference on Computer Cupported Cooperative Work (CSCW)*, CSCW '11, pages 405–414, New York, NY, USA, 2011. ACM.

[154] Alan Murray. *The Wall Street Journal Essential Guide to Management: Lasting Lessons from the Best Leadership Minds of Our Time*. HarperBusiness, New York, USA, 1 edition, August 2010.

[155] Sridhar Nerur and VenuGopal Balijepally. Theoretical reflections on agile development methodologies. *Commun. ACM*, 50(3):79–83, March 2007.

[156] A. Neus and P. Scherf. Opening minds: Cultural change with the introduction of open-source collaboration methods. *IBM Systems Journal*, 44(2):215–225, 2005.

[157] Michael S. Scott Norton. The 1990s research program: Implications for management and the emerging organization. *Decision Support Systems*, 12(4-5):251 – 256, 1994.

[158] Paul A. O'Keefe and Lisa Linnenbrink-Garcia. The role of interest in optimizing performance and self-regulation. *Journal of Experimental Social Psychology*, 53:70–78, 2014.

[159] Dennis W Organ and Charles N Greene. The effects of formalization on professional involvement: A compensatory process approach. *Administrative Science Quarterly*, pages 237–252, 1981.

[160] Jack D Orsburn. *Self-directed work teams: The new American challenge*. Irwin Professional Pub, 1996.

[161] M. Paasivaara, V. T. Heikkil, and C. Lassenius. Experiences in scaling the product owner role in large-scale globally distributed scrum. In *Global Software Engineering (ICGSE), 2012 IEEE Seventh International Conference on*, pages 174–178, Aug 2012.

[162] M. Paasivaara and C. Lassenius. Scaling scrum in a large distributed project. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*, pages 363–367, Sept 2011.

[163] Maria Paasivaara and Casper Lassenius. Scaling scrum in a large globally distributed organization: A case study. In *Global Software Engineering (ICGSE), 2016 IEEE 11th International Conference on*, pages 74–83. IEEE, 2016.

[164] Michael Quinn Patton. *Qualitative evaluation and research methods*. SAGE Publications, inc, 3rd edition, 2001.

[165] A.K. Paul and R.N. Anantharaman. Impact of people management practices on organizational performance: analysis of a causal model. *International Journal of Human Resource Management*, 14(7), 2003.

[166] Craig L Pearce and Charles C Manz. The new silver bullets of leadership: The importance of self-and shared leadership in knowledge work. 2005.

[167] Marian Petre. How expert engineering teams use disciplines of innovation. *Design Studies*, 25(5):477–493, 2004.

[168] Raphael Pham, Leif Singer, Olga Liskin, Fernando Figueira Filho, and Kurt Schneider. Creating a shared understanding of testing culture on a social coding site. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 112–121, 2013.

[169] R. Premraj, M. Shepperd, B. Kitchenham, and P. Forselius. An empirical analysis of software productivity over time. In *11th IEEE International Software Metrics Symposium (METRICS'05)*, pages 10 pp.–37, Sept 2005.

[170] Teade Punter, Marcus Ciolkowski, Bernd Freimut, and Isabel John. Conducting on-line surveys in software engineering. In *Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on*, pages 80–88. IEEE, 2003.

[171] S Antonio Ruiz Quintanilla and Bernhard Wilpert. Are work meanings changing? *The European Work and Organizational Psychologist*, 1(2-3):91–109, 1991.

[172] J. Hank Rainwater. *Herding Cats:A Primer for Programmers Who Lead Programmers*. Apress, 2002.

[173] W Alan Randolph. Navigating the journey to empowerment. *Organizational dynamics*, 23(4):19–32, 1995.

[174] Wolfgang Reinhardt, Benedikt Schmidt, Peter Sloep, and Hendrik Drachsler. Knowledge worker roles and actions?results of two empirical studies. *Knowledge and Process Management*, 18(3):150–174, 2011.

[175] CAHRS ResearchLink. Perception is reality: How employees perceive what motivates hr practices affects their engagement, behavior and performance. 2011.

[176] Peter C. Rigby, Daniel M. German, and Margaret-Anne Storey. Software peer review practices: A case study of the apache server. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 541–550, New York, NY, USA, 2008. ACM.

[177] Julia Rubin and Martin Rinard. The Challenges of Staying Together While Moving Fast: An Exploratory Study. In *ICSE '16: Proc. of the 38th Int. Conf. on Software Engineering*, May 2016. To appear.

[178] Gnther Ruhe. Software engineering decision support – a new paradigm for learning software organizations. In Scott Henninger and Frank Maurer, editors, *LSO*, volume 2640 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 2002.

[179] Kevin T. Ryan. Software processes for a changing world. In *Proceedings of the 2014 International Conference on Software and System Process*, ICSSP 2014, pages 8–9, New York, NY, USA, 2014. ACM.

[180] Richard M. Ryan and Edward L. Deci. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American Psychologist*, pages 68–78, 2000.

[181] A. Sarma, D. F. Redmiles, and A. van der Hoek. Palantir: Early detection of development conflicts arising from parallel code changes. *IEEE Transactions on Software Engineering*, 38(4):889–908, 2012.

[182] Anita Sarma, Gerald Bortis, and Andre van der Hoek. Towards supporting awareness of indirect conflicts across software configuration management workspaces. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, ASE '07, pages 94–103, New York, NY, USA, 2007. ACM.

[183] Anita Sarma, Larry Maccherone, Patrick Wagstrom, and James Herbsleb. Tesseract: Interactive visual exploration of socio-technical relationships in software development. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 23–33, USA, 2009. IEEE Computer Society.

[184] Walt Scacchi. Collaboration practices and affordances in free/open source software development. In *Collaborative software engineering*, pages 307–327. Springer, 2010.

[185] Roy Schmidt, Kalle Lyytinen, and Paul Cule Mark Keil. Identifying software project risks: An international delphi study. *Journal of management information systems*, 17(4):5–36, 2001.

[186] Janet Siegmund, Norbert Siegmund, and Sven Apel. Views on internal and external validity in empirical software engineering. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 1, pages 9–19. IEEE, 2015.

[187] Brent D Slife and Richard N Williams. *What's behind the research?: Discovering hidden assumptions in the behavioral sciences*. Sage, 1995.

[188] Hubert Smits. 5 levels of agile planning: From enterprise product vision to team stand-up. *Rally Software Development Corporation Whitepaper*, 2006.

[189] Donna Spencer. *Card sorting: Designing usable categories*. Rosenfeld Media, 2009.

[190] Gretchen M Spreitzer. Psychological empowerment in the workplace: Dimensions, measurement, and validation. *Academy of management Journal*, 38(5):1442–1465, 1995.

[191] Klaas-Jan Stol, Paris Avgeriou, Muhammad Ali Babar, Yan Lucas, and Brian Fitzgerald. Key factors for adopting inner source. *ACM Trans. Softw. Eng. Methodol.*, 23(2):18:1–18:35, April 2014.

[192] Margaret-Anne Storey, Leif Singer, Fernando Figueira Filho, Alexey Zagalsky, and Daniel M. German. How social and communication channels shape and challenge a participatory culture in software development. *IEEE Transactions on Software Engineering*, 2016.

[193] Margaret-Anne Storey, Alexey Zagalsky, Fernando Figueira Filho, Leif Singer, and Daniel M German. How social and communication channels shape and challenge a participatory culture in software development. *IEEE Transactions on Software Engineering*, 43(2):185–204, 2017.

[194] Margaret-Anne D. Storey, Davor Čubranić, and Daniel M. German. On the use of visualization to support awareness of human activities in software development: a survey and a framework. In *Proceedings of the 2005 ACM symposium on Software visualization*, SoftVis '05, pages 193–202, 2005.

[195] Margie Sutherland and Wilhelm Jordaan. Factors affecting the retention of knowledge workers. *Journal of Human Resource Management*, 2004.

[196] Robert I Sutton and Huggy Rao. *Scaling up excellence: Getting to more without settling for less*. Crown Business, 2014.

[197] Hirotaka Takeuchi and Ikujiro Nonaka. The new new product development game. *Harvard business review*, 64(1):137–146, 1986.

[198] Hirotaka Takeuchi and Ikujiro Nonaka. The new new product development game. *Harvard Business Review*, 64(1):137–146, 1986.

[199] Jasmine Tata. Autonomous work teams: an examination of cultural and structural constraints. *Work Study*, 49(5):187–193, 2000.

[200] Christopher Thornton. Analogical inference as generalised inductive inference. *Analogical and Inductive Inference*, pages 254–263, 1989.

[201] Jason Tsay, Laura Dabbish, and James Harbsleb. Let's Talk About It: Evaluating Contributions through Discussion in GitHub. In *FSE '14: Proc. of the 22nd Int. Symp. on Foundations of Software Engineering*, November 2014. To appear.

[202] Jason Tsay, Laura Dabbish, and James Herbsleb. Influence of social and technical factors for evaluating contribution in github. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 356–366, New York, NY, USA, 2014. ACM.

[203] Pradeep K Tyagi. The effects of appeals, anonymity, and feedback on mail survey response patterns from salespeople. *Journal of the Academy of Marketing Science*, 17(3):235–241, 1989.

[204] H. Van Mierlo, C. G. Rutte, J. K. Vermunt, M. A. J. Kompier, and J. A. C. M. Doorewaard. A multi-level mediation model of the relationships between team autonomy, individual task design and psychological well-being. *Journal of Occupational and Organizational Psychology*, 80(4):647–664, 2007.

[205] Guus Van Waardenburg and Hans Van Vliet. When agile meets the enterprise. *Information and software technology*, 55(12):2154–2171, 2013.

[206] Bogdan Vasilescu, Kelly Blincoe, Qi Xuan, Casey Casalnuovo, Daniela Damian, Premkumar Devanbu, and Vladimir Filkov. The sky is not the limit: Multitasking on GitHub projects. In *International Conference on Software Engineering*, ICSE, pages 994–1005. ACM, 2016.

[207] VersionOne. 11th annual state of agile survey: The state of agile development. 2016.

[208] Padmal Vitharana, Julie King, and Helena Chapman. Impact of internal open source development on reuse: Participatory reuse in action. *J. Manage. Inf. Syst.*, 27(2):277–304, October 2010.

[209] Kevin Vlaanderen, Slinger Jansen, Sjaak Brinkkemper, and Erik Jaspers. The agile requirements refinery: Applying scrum principles to software product management. *Inf. Softw. Technol.*, 53(1):58–70, January 2011.

[210] Fred O. Walumbwa and Chad A. Hartnell. Understanding transformational leadership?employee performance links: The role of relational identification and self-efficacy. *Journal of Occupational and Organizational Psychology*, 84(1):153–172, 2011.

[211] Jim Whitehead. Collaboration in software engineering: A roadmap. *Future of Software Engineering*, 0:214–225, 2007.

[212] Manuel Wiesche and Helmut Krcmar. The relationship of personality models and development tasks in software engineering. In *Proceedings of the 52nd ACM conference on Computers and people research*, pages 149–161. ACM, 2014.

[213] Laurie Williams and Alistair Cockburn. Guest editors' introduction: Agile software development: It's about feedback and change. *Computer*, 36(6):39–43, 2003.

[214] James P Womack and Daniel T Jones. *Lean Thinking: Banish Waste and Create Wealth in Your Corporation*. Free Press, 2003.

[215] Sunny Wong, Yuanfang Cai, Giuseppe Valetto, Georgi Simeonov, and Kanwarpreet Sethi. Design rule hierarchies and parallelism in software development tasks. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, pages 197–208, Washington, DC, USA, 2009. IEEE Computer Society.

[216] Robert B. Woodruff. Customer value: The next source for competitive advantage. *Journal of the Academy of Marketing Science*, 25(2):139–153, 1997.

[217] Shundan Xiao, Jim Witschey, and Emerson Murphy-Hill. Social influences on secure development tool adoption: Why security tools spread. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 1095–1106, New York, NY, USA, 2014. ACM.

[218] Yutaka Yamauchi, Makoto Yokozawa, Takeshi Shinohara, and Toru Ishida. Collaboration with lean media: How open-source software succeeds. CSCW '00, pages 329–338, NY, USA, 2000. ACM.

[219] Jixia Yang, Zhi-Xue Zhang, and Anne S. Tsui. Middle manager leadership and frontline employee performance: Bypass, cascading, and moderating effects. *Journal of Management Studies*, 47(4):654–678, 2010.

[220] Dvora Yanow and Peregrine Schwartz-Shea. *Interpretation and method: Empirical research methods and the interpretive turn*. Routledge, 2015.

[221] Robert K Yin. *Case study research: Design and methods*. Sage publications, 2013.