Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Petrus Holm

Exploring the Influence of Working Agreements on Software Development Cycle Times

Master's Thesis
Helsinki, December 30, 2022

Supervisor: PhD Arto Hellas
Advisor: MSc Ari-Pekka Koponen

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author: Petrus Holm

Title:
Exploring the Influence of Working Agreements on Software Development Cycle Times

Date: December 30, 2022 **Pages:** vii+51

Major: Computer Science **Code:** SCI3042

Supervisor: PhD Arto Hellas

Advisor: MSc Ari-Pekka Koponen

Introduction: Effective team collaboration is essential for the success of software development projects. One key aspect of collaboration is ways of working, defined as explicit and implicit rules and norms that guide team behavior. Swarmia, a Software as a Service productivity tool, offers a Working Agreements feature to help teams set and track their ways of working. There is little prior research on how teams utilize such tools and their impact on performance.

Methods: The study used a quantitative research design and analyzed data from a sample of approximately five hundred Swarmia client teams. The research aimed to understand the relationship between how teams use Working Agreements and whether the use of Working Agreements influenced software development cycle times. SQL and multiple linear regression were used for the analysis.

Results: In the studied sample, 64% of teams configured four or fewer Working Agreements, with the most common agreements being limits for pull request cycle and review times. Limiting the number of open pull requests increased cycle times while prohibiting main branch pushes and limiting review time decreased them. Each team member using Slack notifications reduced the pull request cycle time by 2.5 hours. Furthermore, using team-level Slack reports reduced the development times. Larger teams miss their targets more often but perform better in terms of pull request cycle time.

Conclusion: The study shows that ways of working made with digital tools can positively affect the behavior of software development teams. Multiple relationships were found between the Working Agreement configuration and cycle times. The findings can inform the development of productivity tools and strategies for improving team collaboration. Future research should focus on optimal Working Agreements for different teams, Slack notification optimization, and the impact of agreements that cannot be currently configured in these tools. More metrics and interviews would provide a better view of the teams' situation.

Keywords: software development, self-managing teams, working agreements, cycle time

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tieto-, tietoliikenne- ja informaatiotekniikan maisteriohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä: Petrus Holm

Työn nimi:
Yhteisten työskentelytapojen vaikutus ohjelmistokehityksen läpimenoaikoihin

Päiväys:	Joulukuu 30, 2022	**Sivumäärä:**	vii+51
Pääaine:	Tietotekniikka	**Koodi:**	SCI3042
Valvoja:	FT Arto Hellas		
Ohjaaja:	FM Ari-Pekka Koponen		

Johdanto: Tehokas tiimityöskentely on tärkeää ohjelmistokehitysprojektien menestykselle. Tiimityön keskiössä ovat yhteiset työskentelytavat, jotka määritellään tiimin käyttäytymistä ohjaaviksi eksplisiittisiksi ja implisiittisiksi säännöiksi ja normeiksi. Swarmia on SaaS-palvelu tuottavuuden parantamiseen, jonka Working Agreements -ominaisuus auttaa näiden työskentelytapojen sopimiseen ja seurantaan. Aiempaa tutkimusta vastaavien työkalujen käytöstä ja vaikutuksesta tiimien suorituskykyyn on vähäisesti.

Menetelmät: Tutkimus toteutettiin määrällisenä tutkimuksena, jossa analysoitiin noin viidensadan Swarmia-asiakastiimin dataa. Tavoitteena oli selvittää Working Agreements -ominaisuuden käytön suhdetta prosessien läpimenoaikoihin. Analyysiin käytettiin SQL:ää ja lineaarista regressiota.

Tulokset: Otoksen tiimeistä 64% määritteli neljä tai vähemmän Working Agreement -sääntöä. Yleisimmät säännöt olivat pull requestien läpimenoajan ja arviointiajan rajoittaminen. Pull requestien määrän rajoittaminen lisäsi läpimenoaikoja, kun taas päähaaraan puskemisen kieltäminen ja arviointiajan rajoittaminen vähensivät niitä. Jokainen Slack-ilmoituksia käyttävä tiimin jäsen vähensi pull requestien läpimenoaikaa 2,5 tunnilla. Lisäksi tiimikohtaiset Slack-raportit vähensivät kehitykseen käytettyä aikaa. Suuremmat tiimit pääsevät tavoitteisiin harvemmin, mutta suoriutuvat paremmin pull requestien läpimenoaikojen osalta.

Johtopäätökset: Tutkimus osoittaa, että digitaalisilla työkaluilla määritellyt työskentelytavat voivat vaikuttaa positiivisesti ohjelmistokehitystiimien käyttäytymiseen. Working Agreement -konfiguraatiolla ja läpimenoajoilla havaittiin useita yhteyksiä. Tulokset voivat ohjata tuottavuustyökalujen kehittämistä ja tiimityöskentelyn parantamisponnistuksia. Tulevaisuudessa tutkimuksen tulisi keskittyä erilaisten tiimien optimaalisiin sääntökokoonpanoihin, Slack-ilmoitusten optimointiin sekä niiden sääntöjen vaikutuksen tutkimukseen, joita ei tällä hetkellä voi määritellä näissä työkaluissa. Useampien mittarien ja haastattelujen käyttö antaisi paremman näkymän tiimien tilanteeseen.

Asiasanat:	ohjelmistokehitys, itseohjautuvat tiimit, työskentelytavat, prosessin läpimenoaika
Kieli:	Englanti

Acknowledgements

Kaikkea minkä aloittaa ei ole pakko saada valmiiksi.

Taistelin tämän ajatuksen kanssa syksyllä 2020. Työelämä meinasi viedä mennessään ja oli hieno tunne keskittyä yhteen asiaan kerralla. Pohdin, voisiko vielä maisteriopintoihin löytää vastaavan kiinnostuksen. Kun harrastuksesta tulee intohimo, löytää uudenlaisen motivaation niin opintoihin kuin työtehtäviin: hyppy tietotekniikan maisteriohjelmaan on eittämättä ollut elämän parhaita päätöksiä. Diplomityössäni pääsinkin yhdistämään molempien tieteenalojen oppeja.

Akateemisen kirjoitustyylin me-muoto tuntui aluksi hieman erikoiselta – yksinhän diplomitöitä tehdään. Huomasin kuitenkin, että Holmin tutkimusryhmä on ollut jo pitkään paljon enemmän kuin allekirjoittanut.

Kiitoksia valvojalleni Arto Hellakselle ehtymättömästä tuesta ja kannustavasta otteesta. Tapaamisissamme sain uutta intoa vääntöön sekä arvokkaita oppeja niin työhön kuin muuhunkin elämään.

Haluan kiittää ohjaajaani Ari-Pekka Koposta arvokkaista näkemyksistä ja puskemisesta kohti tavoitetta. Ratkaisut haastavimpiinkin ongelmiin olivat usein vain yhden Slack-viestin päässä. Kiitos myös muille swarmialaisille kaikesta avusta sekä yhteisistä hetkistä.

Kotona opitut Windows 3.1:n komentorivin alkeet ja kannustus läksyjen pariin ovat kantaneet alakoulusta tähän päivään. Kiitos perheelle tuesta niin opinnoissa kuin muissakin elämän haasteissa.

Kiitos laitoslordi emeritus Ilarille mähmintäpuheluista ja lyhyestä oppimäärästä lineaarimalleihin. Lopuksi, vähemmän akateemisesta tuesta vuosien varrella kiitos lukemattomille ystäville.

Joulukuu 30, 2022

Left to my own devices,
Petrus Holm

Abbreviations and Acronyms

CD	Continuous Delivery
CI	Continuous Integration
DORA	DevOps Research and Assessment
IEEE	Institute of Electrical and Electronics Engineers
IPCT	In Progress Cycle Time
IRCT	In Review Cycle Time
PR	Pull Request
PRCT	Pull Request Cycle Time
RMCT	Ready to Merge Cycle Time
SDLC	Software Development Life Cycle
WA	Working Agreement
WIP	Work In Progress

Contents

Chapter 1

Introduction

Technology companies are looking for ways to get the most out of their software development teams in the highly competitive market. Previously, the focus has been on the performance of individual contributors with the help of artificial metrics, such as lines of code [39].

A new wave of tools has emerged to answer the need for team-level productivity insights. These products are openly opinionated and aim to influence their client companies to use specific popular development methodologies, such as Continuous Integration. Companies building solutions in the field include LinearB [26], Haystack [15], Jellyfish [20], and Swarmia [48], among others.

Swarmia is a Finnish startup company that has developed its namesake developer productivity tool since 2019. The product vision is based on the idea that development teams use the tool for self-improvement. The application integrates with version control and project planning systems to provide data-based insights to the teams.

Working Agreements (WAs) is a key feature of Swarmia. With Working Agreements, teams configure ways of working into the application. Then, the application starts tracking them and informing the team members of their progress. For instance, a team can set limits to open issues or pull requests. The Working Agreement templates are mainly based on the concept of flow, first introduced in Toyota Production System [57] and later adopted in modern management frameworks (e.g. [7, 23]).

1.1 Scope and Problem Statement

In this thesis, we explore whether Working Agreements influence teams' pull request (PR) activity. For the purpose of this thesis, the terms productivity

and performance are mainly used to refer to teams' PR metrics.

The study is a quantitative research based on Swarmia client teams' data. For the study, data from a sample of teams in Swarmia's platform was collected for analysis. The data included codebase statistics, such as commit frequency and batch size. Additionally, Swarmia internal data on how teams configure Swarmia was used.

As a null hypothesis, we assume that using Working Agreements does not influence team productivity. This hypothesis is then challenged using two research questions:

RQ1 How do teams use Working Agreements?

RQ2 What is the effect of Working Agreements on software development cycle times?

For RQ1, we examine the usage data of Swarmia teams. The aim is to determine how popular the Working Agreements feature is and how the teams have configured it. The first research question is investigated with SQL aggregate queries.

We look into RQ2 from two aspects: Do Working Agreements overall influence teams' software development cycle times and if they do, are some configurations of working agreements more influential than others. For the second research question, an multiple linear regression model is utilized to look into the Working Agreement configuration's relationship with the teams' performance, specifically, the software development cycle times.

Chapter 2

Background

2.1 Software Engineering

The craft of software engineering emerged during the 1960s. Software crisis, the growing gap between expectations and results for software, called for a more systematic approach to software production. Authors suggested that software should be considered a branch of engineering, like hardware. NATO-organized conference in 1968 widely launched the new, engineering-inspired approach to software production. [32]

Even though the comparison to engineering was initially used to provoke thought [32], it has since become a part of the software production vocabulary. However, the concept of software engineering has remained a topic of dispute in academia. Boehm [3] is consistent with Webster's dictionary definitions for "software" and "engineering", introducing a formal definition for software engineering: applying science to create "useful to man" computer programs and documentation. He argues that the difference between arbitrary development and software engineering is that the latter ensures the result is useful for the end-users, satisfying the specification. On the other hand, Sommerville [44] establishes the gap between software development and software engineering: he defines software development as the actual development process where code is written, whereas software engineering is tied to the life cycle of the software – from planning to maintenance.

The engineering mindset and the lifecycle of the software are included in IEEE [19] definition on software engineering. The definition can be described as a combination Boehm's and Sommerville's definitions. For this thesis, we will be using the IEEE definition: "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software".

Software engineering can be divided into four main activities: software specification, development, validation, and evolution [44]. These steps can also be called a software process or a software development life cycle (SDLC). Commonly used SDLCs include the Waterfall model, Validation & Verification model (V model), and Agile methods [2]. Projects can benefit from different approaches: dimensions such as level of risk, budget, or project timeline vary between the SDLCs [1, 7]. The industry data shows a trend towards the iterative approaches [44].

2.1.1 Waterfall Model

The so-called Waterfall model is widely referred to as the most well-known SDLC. Waterfall model consists of 5 sequential process stages: requirements analysis, design, coding, testing, and maintenance [1]. As the name implies, a project following the Waterfall model is one-way: it moves only forwards, and the different stages do not overlap. Each stage produces a deliverable to be used as a basis for the upcoming stage [2].

In practice, projects rarely blindly follow the Waterfall model: upcoming stages are prepared parallel to the current one, and previous stages are revisited when the need arises [44]. This "iterative Waterfall" is in line with the original model from the research paper Waterfall is derived from: the author initially intended the method to be used in an iterative manner [40].

A significant shortcoming of the Waterfall model is that the software is delivered only once. On the other hand, it is good if the requirements are well-known beforehand and stay relatively static during the project. For example, a security system for a power plant would be a potential use case for the method.

2.1.2 Agile Methods

Agile methods is an umbrella term that includes methodologies like Extreme Programming [56], Lean Development, and Scrum [41]. The terminology is further complicated due to the imprecise usage of the term in the industry. The common properties of Agile methods include iterative development, focus on communication, and a critical stance toward intermediate deliverables like requirements [7]. Furthermore, Agile evangelists have agreed that Agile is more about a "state of mind" rather than a single process or methodology [7].

In iterative development, new versions of a working product are delivered constantly: most productive teams deploy software to production even multiple times in a day [11]. This approach enables the end-users to use the

software from the early days of the development cycle. Development teams can receive constant feedback from users and iterate the software accordingly for the next releases [2].

Focus on communication encourages development teams to enable low-threshold, constant communication practices instead of documentation-heavy decision-making methods [7]. Agile methods introduce practices like daily stand-up meetings and pair programming to support this [13]. When Agile methods emerged, co-locating team members were also seen as the key to success for working communication [7]. With the possibilities presented by modern collaboration platforms like Slack and Zoom, low threshold communication has become possible for remote teams – working from the same physical location is no longer seen as crucial for Agile implementation.

Another definitive characteristic of Agile methods is the critical stance toward intermediate, non-critical deliverables. By reducing the resources spent on these artifacts, teams can focus their efforts on developing the product [7]. The self-managing nature of Agile teams further promotes the idea that they have the best context to prioritize and plan their work [1].

Even though Agile has an evergrowing status in the software development world, traditional methods like Waterfall are still needed. Large teams and complex projects are seen as environments where more structured processes should be preferred. On the other hand, teams are increasingly incorporating Agile properties into their traditional software processes and vice versa. [7]

2.1.3 DevOps

Agile has transformed the way software is developed. However, in the software lifecycle, the performance of testing, deployment, and post-deployment functions like fault recovery should be considered. DevOps brings development and IT operations closer together, bridging the gap between these silos and enabling responsiveness throughout the lifecycle of the software [16]. DevOps aims to shorten the software development lifecycle by introducing automated tools that ship code to production faster, boosting both productivity and quality of development [8]. According to Humble and Molesky [18], DevOps practices can be divided into four main aspects: automation, culture, sharing, and measurement.

The most distinctive automation practices introduced by DevOps are continuous integration (CI) and continuous delivery (CD). In continuous integration, the developers' changes are often submitted back to the main branch as soon as possible, usually preceded by a build task and automated unit tests run by a CI worker [44]. CI is often followed by CD, which is in charge of integration testing, end-to-end testing, and finally, deploying the approved

version into a staging environment [4]. The abbreviation "CD" is commonly also used to refer to continuous deployment.

In addition to the steps in continuous delivery, continuous deployment automates software deployment to production environments. In contrast, continuous delivery pipelines require the user's manual interaction to initialize deployments. In the industry, CD usually refers to continuous delivery, though the term is used in quite a liberal manner. Systems consisting of both CI and CD functions are called the CI/CD pipeline. In modern-day DevOps, a stable CI/CD pipeline is the backbone of the team's success: In addition to customer satisfaction, implementing these practices can have an organizational impact, such as improved work satisfaction [11].

In essence, DevOps is about ways of working. Sociotechnical factors must be considered to create a well-functioning DevOps function [16]. Implementing DevOps requires changes to the organization's work culture and a change of mindset from individual contributors [24]. For example, rotating responsibilities and increased information sharing are ways to enforce DevOps practices [24].

Measuring processes is a cornerstone of increasing efficiency. In DevOps, production environments are measured continuously, and these metrics are used as gauges of development productivity [24]. An increasing trend is also to measure the development processes directly: For example, statistics from version control systems are analyzed to extract information on the organization's health [11].

2.2 Developer Productivity

2.2.1 Defining Productivity and Performance

Productivity in industrial engineering is defined as the ratio of input and output [5, 51]. In manufacturing, input and output are relatively easily measured as the used resources and the produced goods. On the other hand, non-manufacturing businesses rely on more complex definitions: in knowledge work, person-hour is not considered a relevant metric of productivity [52]. The definition of output has also been under debate. In 1988, Chew [5] proposed that output should include metrics other than "the number of units," for example, quality, timeliness, and price of the product. Many relatively similar context definitions have also emerged in the literature [52].

The terms productivity and performance are sometimes used interchangeably. To further complicate the terminology, the authors define performance as a component of productivity and vice versa. Based on a vast amount of

literature research, Tangen [52] defines performance as an "umbrella term of excellence" consisting of productivity and other non-cost elements. In contrast, productivity is defined as a "physical phenomenon" – the ratio of input and output. As an example of the opposite approach, SPACE authors define performance as the "outcome of a system or process," measured by metrics like service health or customer satisfaction. Productivity in the SPACE framework is about the sum of its parts, with performance as one of them [12]. To sum things up, these terms are used to refer to varying phenomena in different situations.

In the examples before, productivity is reviewed from the company's perspective. Often it is beneficial to look into the metrics of a smaller entity: a department, a team, or an individual [12, 52]. These metrics can provide more accurate insights into the entity in question.

When measuring the productivity of teams or individual employees, it can be tempting to incentivize them based on the metrics. For example, software developers could get promotions or raise if they contribute more than average to the product. Often these ambitions can lead to unwanted results as employees optimize their work to meet the metric rather than focusing on the company's interests [5, 50]. For example, if the metric used was the number of pull requests, it would be tempting to break the work into tiny increments to pump up the numbers. Individual preferences further tricky the topic, as some employees by habit contribute more commits than others [33].

2.2.2 Productivity in Software Engineering

Determining the productivity of engineers is seen as crucial for the success of software development. Therefore, substantial research has been conducted in the past years [33]. No consensus has been found on measuring productivity accurately. Historically, source code-based metrics such as lines of code or amount of commits have been utilized [33]. Nowadays, these metrics are considered, at best, an incomplete representation of the true productivity [12].

Modern frameworks have explored the work of a software engineering team as a joint effort: as popular SDLCs emphasized teamwork, productivity should be assessed in the same context. The team is seen as the "basic work unit" [29], and the research has shifted towards how teams can improve together. Even though high-performing individuals are still seen as valuable assets, companies should focus on building and enabling high-performing teams [12].

Agile methods estimate productivity by estimating velocity – units of completed work in a time frame. Although velocity is considered a better in-

dicator of productivity than source code metrics, it has significant drawbacks. The topic is widely covered by Forsgren et al. [11] in their industry-shaping book *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. Most importantly, velocity is highly relative to the context and should not be used as an absolute metric. Secondly, velocity is an excellent example of a metric prone to be gamed by the employees, as described previously. [11]

2.2.3 Developer Productivity Frameworks

Due to the misconceptions about developer productivity, research has gravitated toward finding reliable ways to discuss, measure, and enhance developer productivity. DevOps Research and Assessment (DORA), an Alphabet Inc subsidiary, has pioneered the research of DevOps and developer productivity. Their book Accelerate [11] presents four key metrics to measure software delivery performance, often referred to as the DORA metrics. The metrics are as follows:

1. Lead Time
2. Deployment Frequency
3. Mean Time to Restore
4. Change Fail Percentage

Lead Time is a metric derived from Lean manufacturing to the software context. It refers to the time from a feature request to the change that fulfills it. Often in software, the changes are not directly requested by single customers but instead decided by the teams using a large amount of input from stakeholders. DORA metrics define Lead Time as the time from the first commit to code running in the production environment [11].

In this thesis, we use Lead Time to refer to the time from the first commit to a new branch to the moment it is merged to the main branch, as seen in figure 2.1. The industry term for this metric is Pull Request Cycle Time.

Batch size has been popularized by it is used as a key performance metric in the Toyota Production System [57]. DORA researchers note [11] that batch size is a sub-optimal metric in software development. Therefore, they chose Deployment Frequency as a suitable proxy to estimate the batch size. Deployment Frequency is the frequency of pushing a new version to the production environment.

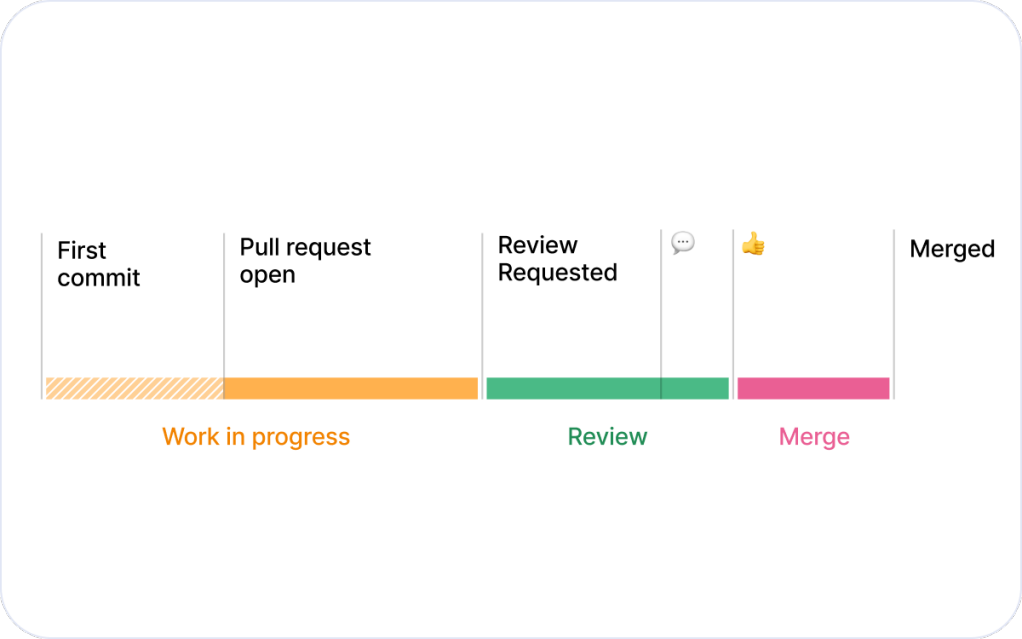


Figure 2.1: Pull Request lifecycle [49]

Lead Time and Deployment Frequency are metrics for software delivery tempo. Two additional metrics are used to determine if the increase in tempo has disadvantages in terms of software quality or system stability: Mean Time to Restore and Change Fail Percentage. Both of these measure situations where the production environment has failed. Forsgren et al. [11] articulate that failure is a common situation in rapidly changing software systems, and the absolute amount of failures is an obsolete metric. Instead, they propose two better suitable metrics: the time used to restore an application after an incident and the percentage of changes that lead to a production system failure.

In Accelerate, the results are clear: increasing software delivery tempo has no negative impact on stability or quality. Instead, teams that performed well in the first two metrics also had better results in the rest measures.

A more recent work from the DORA team is called the SPACE framework. The framework is named after its dimensions: satisfaction and well-being; performance; activity; communication and collaboration; and efficiency and flow. The framework extends DORA metrics by including, for example, perceived metrics. [12]

Forsgren et al. [12] argue that productivity measurement should not rely on a single metric but rather include at least three metrics from the SPACE

dimensions. Furthermore, the metrics should include quantitative, qualitative, and individual and team-level metrics. Also, performance measures on the organizational level are critical to getting a comprehensive image of productivity.

The SPACE publication proposes a set of example metrics to concretize the framework. These proposals are not to be used as is but rather serve as an inspiration for those developing productivity tracking systems. [12]

2.3 Software Engineering as Teamwork

Work is often organized around teams. To distinguish a team from other groups that work together, we use the definition by Katzenbach & Smith: "a team is a small number of people with complementary skills who are committed to a common purpose, set of performance goals, and approach for which they hold themselves accountable" [21]. The fundamental difference is the accountability on both individual and team levels: team members are primarily responsible for each other, their supervisors, or other stakeholders [21].

In software engineering, teams work together to design, develop and maintain products. Working as a team offers upsides such as increased employee satisfaction, innovation, and productivity [30]. In modern software development, the team is considered the basic work unit rather than the individual contributor [29]. This shift has motivated companies and executives to focus on building high-performing teams.

2.3.1 Self-managing Teams

Modern software development methods encourage, or even insist on, the use of self-managing teams [13, 30]. Other terms used to define the phenomena include self-organizing team, autonomous team, and empowered team. Even though most studies report that forming self-managing teams leads to positive outcomes, there have also been opposing conclusions. Factors like poor leadership or the broader nature of the situation can lead to the difficulty of implementing such a team [30].

To enable genuine self-management within a team, company leadership and other internal stakeholders must respect the team's independence and improvement efforts [29]. Additionally, teams are not usually designated with a leader: instead, they are expected to form processes for a shared leadership function. Even though shared leadership can lead to better functioning teams, it adds more complexity and requires better cohesion and communication [43].

To improve, self-managing teams have to change their behavior, either by self-managing or from an external stimulus. Teams that can self-improve can achieve more autonomy than their counterparts. Norms, standards that regulate team members' behavior, are of the essence when building a productive software development team. Norms can include technical and non-technical rules; for example, teams can commit to test-driven development or promote pair programming. The impact of norms for development teams is achieved through increased simplicity of standard processes: team members can trust that their colleagues do certain activities and refrain from the unwanted ones. [46]

Team norms can be formed in two ways: organically within the team or by others, or as part of the organization's guidelines or direct influence [53]. Agile teams tune their ways of working during sprint retrospective meetings. Teams can also host dedicated sessions to discuss their norms: many SDLCs encourage teams to review their methods regularly. To enforce the norms, teams use whiteboards, notifications, and other reminders to refresh their memory and keep each other accountable.

2.3.2 Tools for Teamwork

Teams use a large number of tools to facilitate their work. Issue trackers like Jira, version control systems like Git, and chat tools like Slack play a central role in the day-to-day interaction of software development teams. Many team norms revolve around how team members are expected to use these applications. For example, the developers could commit to preferring public channels over direct messages in an instant messaging program.

As a large part of information-intensive work is done using IT systems, these tools have a remarkable efficiency improvement potential. Teams can aim to boost their performance with the help of software. For example, using a web-based spreadsheet editor can reduce the need to share new versions within the team manually.

In addition to systems designed to help teams work together, software solutions exist to improve team performance. Especially in software development, where the team's outputs are mainly lines of code, it is tempting to conduct team performance metrics from the codebase [28]. Authors argue that these tools are problematic, as they only focus on outputs, reduce quality out of the equation and provide a one-sided view of the team's health [11, 12, 33].

A new wave of solutions addressing team performance has emerged to tackle this dilemma. The tools are openly opinionated and aim to influence their client companies to use specific popular development methodologies, such as Continuous Integration. Examples of such tools include LinearB [26],

Haystack [15], Jellyfish [20], and Swarmia [48]. Even though these tools utilize version control as input for their insights, they aim to offer multi-sided, research-backed metrics for the teams.

2.4 Managing Change

Change management is a systematic approach to organizational change. The change itself is an evident element of all organizations: planned change, on the other hand, requires direct efforts from the company. A vast number of change management frameworks have been proposed, with no single solution for all situations found. [55]

2.4.1 Introducing Changes

It has been generally proven that to make users conform to change, the new changes should be in line with their team's norms [54]. Therefore, organizations should look into current norms and enforce changes in them while introducing broader organizational change. Without this step, things might change on the surface, but no actual change is achieved.

The stronger individuals identify with their team, the stronger the group opinion's effect is: the individuals tend to align with the group's view on the norms [54]. High cohesion is usually a positive phenomenon, but this situation can cause conflict with the organization's change initiatives. Instead of winning the individuals over one by one, it is also necessary to change the team's collective views.

Kotter [22] summarizes that by making change stick, people will not revert to their old ways. Often, the change initiatives are handled as additional projects to the actual work. In a modern, constantly changing work environment, it is therefore important to focus on stickiness instead of short-term wins. The change must be woven into the day-to-day methods, making it a natural part of how teams work. On the other hand, short-term wins, such as achieving weekly goals, are an important way to motivate people for long-term change. [22]

2.4.2 Individuals and Change

The role of individual contributors is significant in organizational change. In software development, the success of change can be concentrated into three significant factors: the engineers' knowledge of the change outcomes, their view on the need for change, and their contribution to the change process. [25]

To spread knowledge on the upcoming change, the organizations must first create a vision of the future after the change. With a shared vision, it is easier to move the whole organization into one common direction [22]. To achieve a shared view of the change, organizations must communicate it clearly, sharing knowledge throughout the organization.

To initiate the need for change within individuals, it is necessary to create a sense of urgency: the change has to happen now, not in the future [25]. Furthermore, the employees need to see the change as applicable and achievable [22]. In the end, employees must be ready to compromise in the moment of change, which requires internal motivation for it [36].

In self-managing teams, the lack of contribution is not a problem, as the team members make the decisions in the first place. When discussing change initiatives that emerge within the team, the contribution is in-built into the process. Of course, teams must ensure that all team members get a say in the matter. The change initiatives that start from an external stimulus, such as an organization-wide move to Agile, are the ones that require elaborate inclusion of the individual contributors.

2.4.3 Change in Software Development

Software development best practices are constantly changing, requiring engineers to stay adaptive throughout their careers. As people move between organizations, they bring in new ways of working. This creates pressure for the teams to be able to digest the new methods and push the incoming employees to align with the current ones.

Furthermore, software development is an industry with constantly changing requirements. This is one of the key reasons why methodologies like Agile have become so popular: they claim to help organizations cope with the vast amount of instability in the ever-changing business environment [14]. The improved internal processes are seen as a way to improve the product, for example, by reducing the errors in production [9].

In addition to self-management, Agile methods promote self-improvement. Practices such as retrospectives are designed to help teams create organizational change based on their own observations [10].

Chapter 3

Methodology

3.1 Research Context

This study was done for Swarmia, a company that offers engineering teams the insights and tools for self-improvement. Swarmia has a product called Swarmia, which is a Software as a Service platform. Teams utilize Swarmia to measure their performance and find potential improvement places in their development pipeline. Swarmia is designed to help teams that work in a self-managed manner.

To provide insight to the client teams, Swarmia integrates with issue trackers like Jira and version control systems like GitHub. The teams are then provided with insights accessible to all team members through the Swarmia web application. In addition, Swarmia can be configured to push updates to the team's Slack channel. A team of any size can use Swarmia: the teams include both teams working within software companies and in-house development teams of non-software companies.

Swarmia provides onboarding to help client teams get started with the platform. In these meetings, an initial WA configuration is done together with the team representatives.

3.1.1 Working Agreements

One of the features in Swarmia is called Working Agreements. Teams can configure up to eight agreements, generally described as team norms or collaboration guidelines. The agreements are selected from a predefined list of templates: each team can customize them to suit their use case. For example, a team could agree that they want to enforce linking pull requests to Jira issues. Swarmia would then track the set condition automatically and

inform the team about the status of the agreement.

template in Swarmia UI	machine-readable name
Limit pull requests in progress	wip_pull_requests
Reduce pull request cycle time	max_pull_request_age
Avoid pushing directly to the default branch	no_direct_pushes_to_main_branch
Avoid working alone	min_issue_contributors
Limit issues in progress	wip_issues
Reduce issue cycle time	max_issue_age
Reduce pull request review time	max_pull_request_review_time
Link pull requests to issues	pull_request_linking

Table 3.1: Working Agreement templates

Working Agreement templates in Swarmia are summarised in Table 3.1, along with their machine-readable names. The machine-readable names are primarily used to refer to the WAs in text, graphs and tables.

Teams discourage pushing changes to main branch without a pull request by using no_direct_pushes_to_main_branch. Work in progress (WIP) limits wip_pull_requests and wip_issues are used to set maximum number of open pull requests or issues, respectively. Duration limits for different processes can be configured with three WAs: max_pull_request_age for reducing time spent on a pull request overall, max_issue_age for limiting how long an issue is worked on, and max_pull_request_review_time to boost code reviews.

The minimum number of individual contributors on a single issue tracker ticket is tracked with min_issue_contributors. Using this WA teams encourage working as a team to avoid siloing. To bridge the gap between version control and issue tracker data, teams can configure pull_request_linking, which ensures that all PRs are accompanied with a related issue tracker identifier.

The Working Agreement view in Swarmia is shown in Figure 3.1. All team members can use this view to set and review WAs for their team. In the list left of page, current WAs and their status is shown. On the right a detail view of a WA, in this case max_pull_request_age, is shown. In this view the user can access the current configuration and statistics, as well as see the items not within the target.

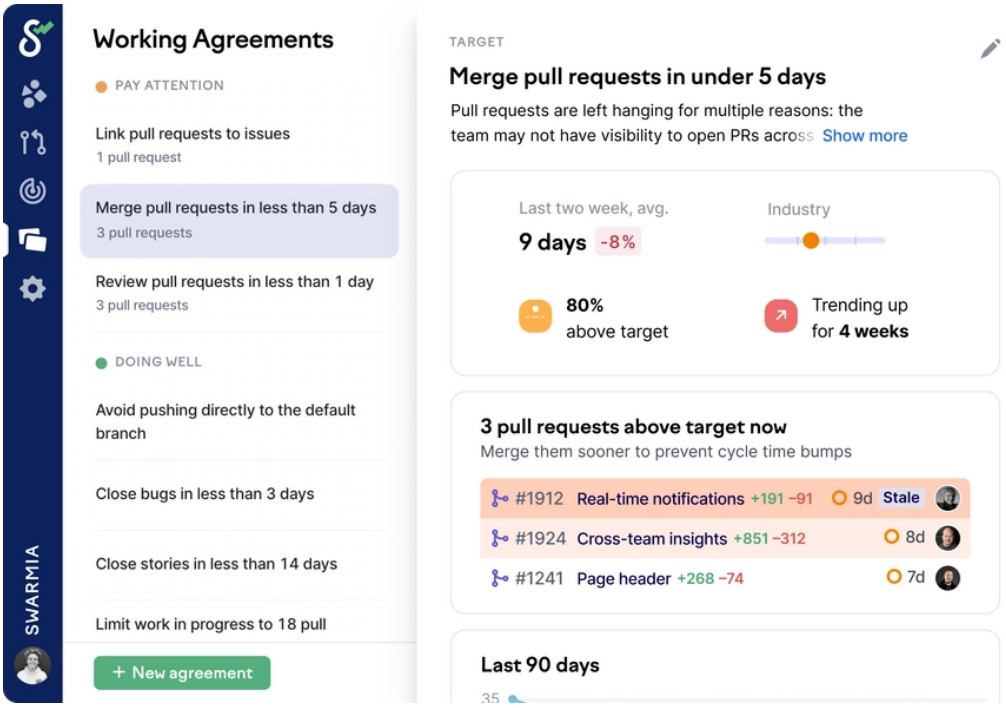


Figure 3.1: Working Agreements in Swarmia. On the left, a list of currently configured WAs. On the right, a detail view of a selected WA with WA-specific statistics.

3.2 Data

The data collected by Swarmia includes, but is not limited to, pull requests, issues, and failed deployments. Furthermore, Swarmia has read-only access to the source code, which is used to estimate the complexity of the change in question. Swarmia never stores client code but instead calculates the size of the change bundle and stores it in a separate database.

Processing of the data is done in Google's BigQuery data warehouse service. All data was processed according to the Swarmia privacy policies, including handling all the data with Swarmia managed devices. All aggregate data was removed at the end of the research.

Most of the information needed is already collected at Swarmia for business intelligence purposes but needs to be structured in a suitable format. The way raw data was combined into this format is shown in Figure 3.2. Additionally, for the needs of this study, weekly aggregates of team metrics were exported from the Swarmia application backend to BigQuery. This

parameter	description
wip_pull_requests	WA status
max_pull_request_age	WA status
no_direct_pushes_to_main_branch	WA status
min_issue_contributors	WA status
wip_issues	WA status
max_issue_age	WA status
max_pull_request_review_time	WA status
pull_request_linking	WA status
github_authors	Pull request authors
swarmia_users	MAU in Swarmia team
slack_users	Swarmia users with Slack notifications
daily_digest	team's Slack daily digest status
target_days	numeric target for a given WA
days_in_use	days since WA configuration
target_achieved	whether target was achieved
ct_days	normalized cycle time in days

Table 3.2: Variables used in the study

data set is called "cycle times" in Figure 3.2. These metrics were merged with basic team information, working agreement usage, and team member activity. Finally, team identifiers and timestamps were removed so the data points would be independent. A sample of the resulting data is displayed in Table 3.3. All variables used in the study are listed in Table 3.2.

wip_pull_requests	wip_issues	...	ct_days	slack_users	daily_digest
0	1	...	1.3	3	0
0	1	...	0.00031	1	0
1	0	...	-0.46	4	1
0	0	...	-0.058	5	1
0	1	...	-1.8	2	1

Table 3.3: Sample rows of linear model input data

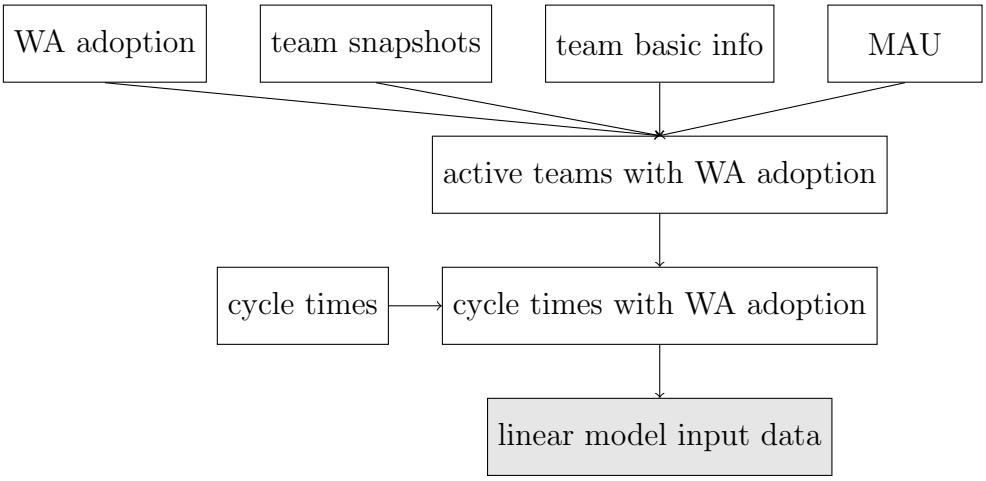


Figure 3.2: Data flow in BigQuery

3.3 Research Questions

The first research question of this study is how teams use Working Agreements: more specifically, which share of Swarmia teams use each agreement and how these teams have configured their Working Agreements setup.

The second research question is what is the effect of Working Agreements on software development cycle times. The hypothesis based on previous research (e.g. [12, 46, 53]) is that as Working Agreements promote methods that accelerate software development, they should directly impact the team's performance. We will look into the Working Agreement composition in different teams and determine which agreements impact the teams' cycle times most.

3.4 Approach

To answer RQ1, data from Swarmia is used to determine how many of the teams have enabled which Working Agreements. Furthermore, we look into how many Working Agreements teams usually use and what goals they have configured for the WAs. For example, WIP limit agreements require teams to specify a threshold for open items. Lastly, team characteristics, such as team size, and their connection to the Working Agreements setup, are analyzed. Some Working Agreements can be configured multiple times for different issue types. The way teams configure these issue-related WAs is further investigated.

Data from all teams using the Working Agreements feature is collected and analyzed to help answer RQ2. Historical activity data is pulled from the teams' source control repositories from 1 July 2020 to 31 July 2022. The data is aggregated weekly for 24 months: some teams have been clients for longer than two years, and some for a shorter amount. All teams included in the study had at least two months of data. The variability of the data collection period should be acceptable, as averages are used, and the period length is not a variable in the analysis.

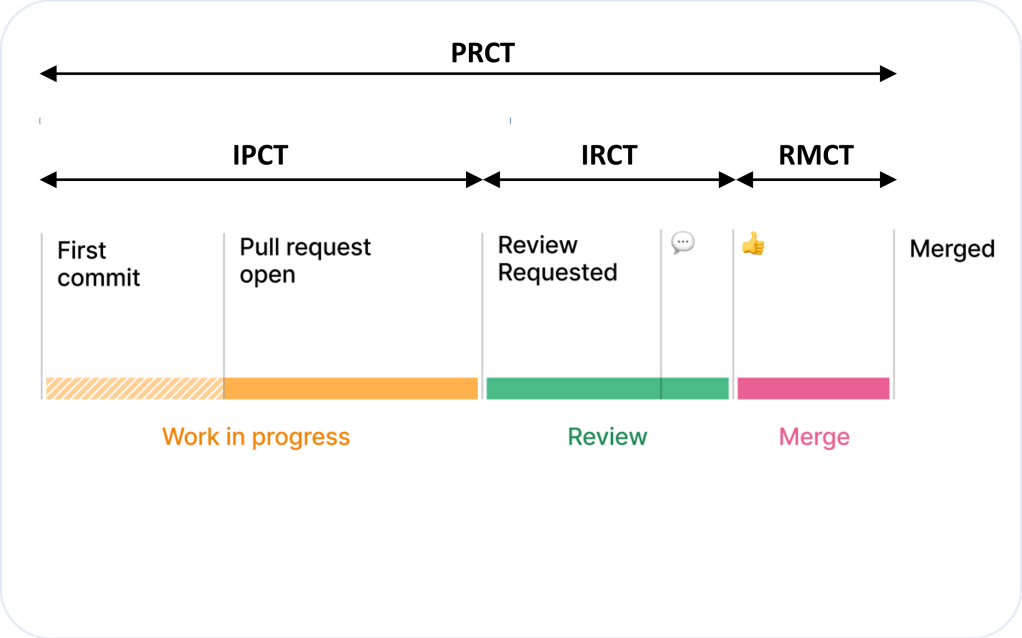


Figure 3.3: Cycle time definitions

The key metric used in the analysis is the Pull Request Cycle Time (PRCT). Additionally, we look into the sub-parts of PRCT: In Progress Cycle Time (IPCT), In Review Cycle Time (IRCT), and Ready to Merge Cycle Time (RMCT). The span of these metrics in relation to each other is shown in Figure 3.3.

Even though Swarmia exports data from issue trackers, these data sources were left out of the analysis based on the scope of the thesis. The analysis is done in BigQuery in line with Swarmia's privacy policy. In addition to a standard SQL query engine, BigQuery offers advanced data analysis capabilities. For this thesis, the inbuilt linear regression feature was utilized. Figure 3.2 presents the data flow before building the regression model.

We look at the relationship between the independent and dependent vari-

ables in linear regression. The dependent variable is the one that, as a hypothesis, depends on the values of the independent variables. Respectively, the independent variables are the configuration of Working Agreements, team member metrics, and Slack notification settings.

	A - wip_pull_requests	B – max_pull_request_age	C – no_direct_pushes_to_main	D – min_issue_contributors	E – wip_issues	F – max_issue_age	G – max_pull_request_review	H – pull_request_linking	ga – github_authors	sw – swarmia_users	sl – slack_users	dd – daily_digest
A	1.0	0.49	0.28	0.16	0.29	0.15	0.42	0.24	-0.09	-0.04	-0.03	0.2
B		1.0	0.23	0.18	0.32	0.31	0.64	0.39	-0.07	0.02	0.06	0.23
C			1.0	0.25	0.26	0.16	0.24	0.35	-0.1	0.0	-0.06	0.09
D				1.0	0.3	0.25	0.15	0.3	0.03	0.09	0.0	0.03
E					1.0	0.58	0.28	0.43	-0.09	0.01	-0.02	0.15
F						1.0	0.32	0.34	-0.03	0.08	0.05	0.19
G							1.0	0.3	-0.01	0.03	0.08	0.32
H								1.0	-0.09	0.0	-0.01	0.1
ga									1.0	0.79	0.78	-0.18
sw										1.0	0.84	-0.07
sl											1.0	-0.01
dd												1.0

Table 3.4: Correlation matrix. The color indicates the strength of correlation. The codes used in y-axis are defined on x-axis.

Pearson correlation coefficient for independent variables is calculated in Table 3.4 to ensure the data is compatible with linear regression analysis: two variables should not correlate for the model to work as expected. The most considerable correlation encountered is the of slack_users and swarmia_users with a value of 0.84. The pairs with high correlation deal with the same themes, for example, WAs related to pull requests. Even though some of the WAs correlate, the data can be used for the model with slight modifications: github_authors and swarmia_users were dropped to include only the team member metric, slack_users.

Linear regression can be formulated as follows:

$$Y = \beta_0 + \beta_i X_i + \beta_{i+1} X_{i+1} + \ldots + \beta_n X_n + \epsilon \tag{3.1}$$

where

Y is dependent variable

β_0 is the constant coefficient (intercept)

β_i is slope for X_i

X_i is independent variable

ϵ is error

The training data is passed to linear regression models in a form presented in Table 3.3. The data has gone through two significant modifications to suit linear regression. First, we calculated each team's average cycle times before enabling any Working Agreements. This average is then subtracted from each data point's cycle time to achieve a normalized data set: the aim is to diminish the differences between teams to achieve comparable cycle times. Secondly, WA activation dates are mapped to Boolean values for each row: the value is zero if WA has not been activated yet and one if it has.

At the beginning of the analysis, three team member-related metrics were included: github_authors, swarmia_users, and slack_users. The amount of Swarmia users did not have statistical significance and strongly correlated with github_authors with a correlation of 0.79. In the end, out of the three metrics, only slack_users was retained in the final model.

The targets set for agreements max_pull_request_age and max_pull_request_review_time were studied with dedicated models. These target models use a Boolean independent variable, calculated from $cycletime < target$. The dependent variables in these models are daily_digest, slack_users, days_in_use: the days the WA has been in use, and target_days: the set target for the given WA. A sample of data used in these models is shown in Table 3.5.

The threshold value α selected for the study diverts from the standard convention of $\alpha = 0.05$. Instead, results with a p-value of $p < 0.10$ are considered as having statistical significance against the null hypothesis. Furthermore, results with a p-value of $p < 0.05$ and $p < 0.01$ were given additional noteworthiness. These three categories of significance are denoted with asterisks in figures and tables. All results are reported with a the precision of three significant figures.

In the analysis, p-values are considered as one factor that contributes to the reliability of the findings. When interpreting the findings, we rely on the

daily_digest	slack_users	days_in_use	target_days	target_achieved
1	22	514	3.0	1
1	8	166	7.0	1
1	1	70	14.0	0
1	3	40	3.0	0
0	10	132	7.0	1

Table 3.5: Sample rows of target model input data

weights of the independent variables of the linear model for further evidence. We note that the results are exploratory and future research should explore similar phenomena in other contexts.

Chapter 4

Results

4.1 Use of Working Agreements

4.1.1 Selecting Working Agreements

To get an overview of how teams are using Working Agreements, we set out with the proportional popularity of WAs. As shown in Figure 4.1, the agreements max_pull_request_review_time and max_pull_request_age are used the most: by 341 and 315 teams, respectively. Another atypical WA is min_issue_contributors, with approximately 150 teams. The remaining Working Agreements are evenly popular: wip_pull_requests 219 teams; wip_issues 204 teams; no_direct_pushes_to_main_branch 229 teams; pull_request_linking 206 teams; and max_issue_age 225 teams.

The distribution of how many WAs each team has in use was calculated. The results are plotted in Figure 4.2. Overall, out of the teams included in this study that enabled at least one Working Agreement, the majority enabled only a few WAs. From the included sample, 64% of the teams configured four or fewer agreements. In addition, teams usually configure only one Working Agreement per template.

Even though eight Working Agreement templates exist, the number of WAs in use can exceed this: issue-related agreements can be configured multiple times for a single team. For example, max_issue_age can be configured distinctly for Epics, Stories, Tasks, and Bugs one to four times. Still, only a tiny share of teams, 4.3%, configured more than eight WAs.

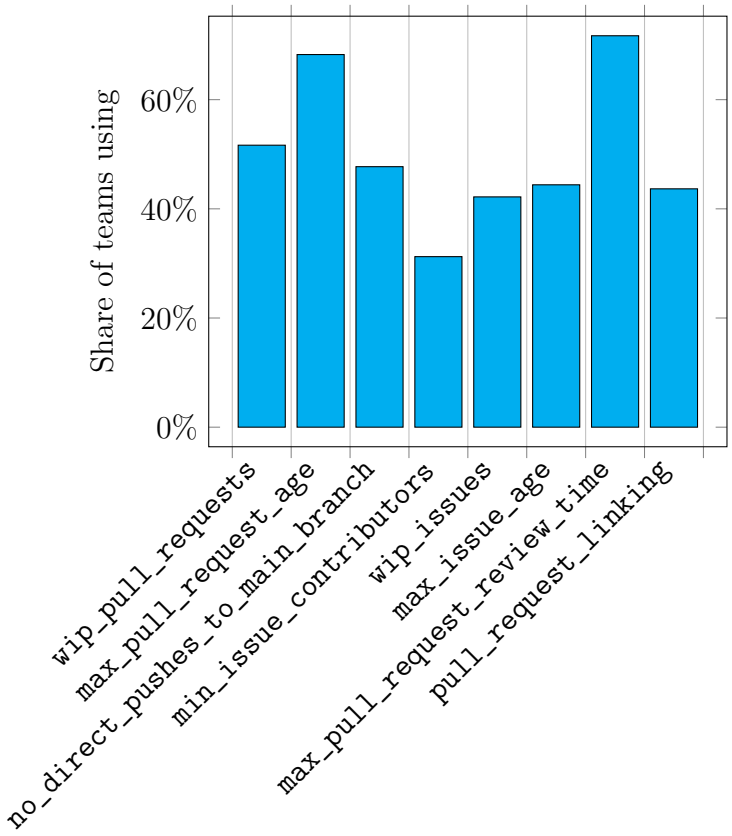


Figure 4.1: Working Agreement usage. Teams can configure multiple WAs.

4.1.2 Issues and Working Agreements

The usage of issue-related Working Agreements was analyzed. Figure 4.3 shows that for all three templates, it is rare for a team to configure more than one Working Agreement. In wip_issues, 120 teams configured it once, 30 teams twice, six teams three times, and five teams four times. For max_issue_age, the results are 150, 21, 3, and 2 teams; for min_issue_contributors, 115, 12, 1, and 1 teams.

The distribution of issue types in these three Working Agreements is shown in Table 4.1. Story is the most popular issue type, with 359 configured Working Agreements. A notable difference is min_issue_contributors, where Epic is often configured in 54% cases. This result implies that teams want to ensure co-creation on top-level issues, while age and the number of issues are more relevant for the smaller entities.

On the other hand, Bug-related WAs are the least configured overall, with only 36 WAs, a stark contrast to Story. For max_issue_age, Bug is the second-

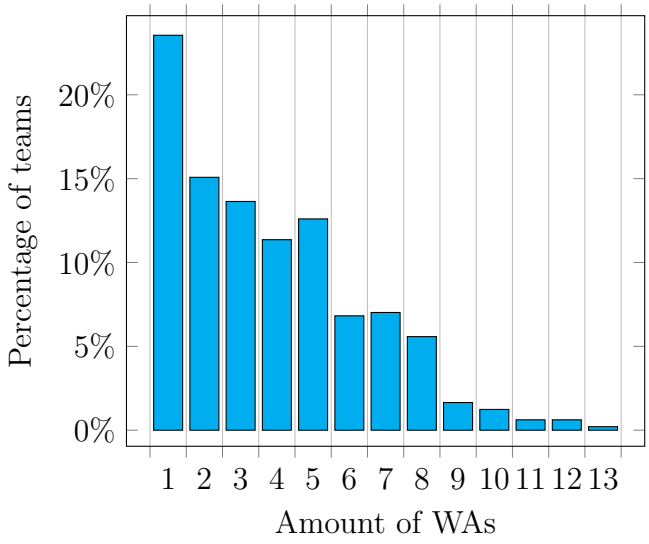
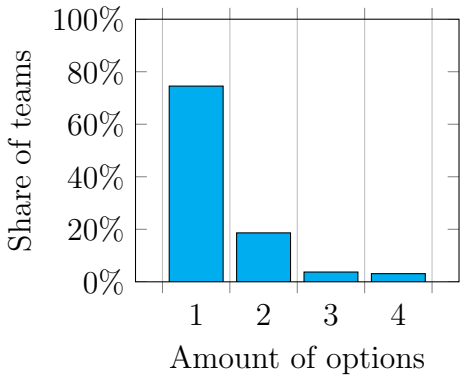


Figure 4.2: Teams' WA usage distribution

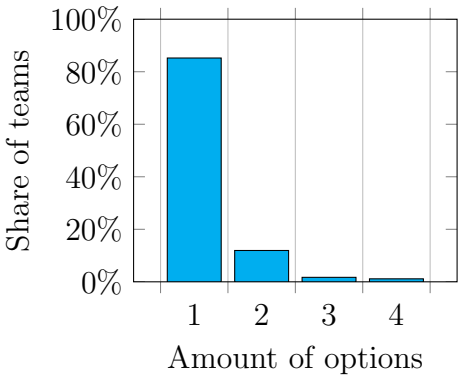
WA	Story	Bug	Epic	Task
max_issue_age	155	29	12	33
min_issue_contributors	52	1	75	10
wip_issues	152	6	29	32
Grand total	**359**	**36**	**116**	**75**

Table 4.1: Issue type distribution

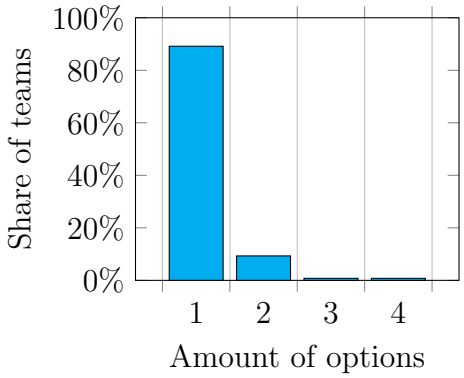
most-popular issue type, even though Story also dominates in this category. Teams aim to reduce the number of long-term bugs in their systems using this Working Agreement.



(a) wip_issues



(b) max_issue_age



(c) min_issue_contributors

Figure 4.3: Issue related WA usage

4.2 Working Agreement Setup Effect on Cycle Times

The primary method used to answer RQ2 is the multiple linear regression model. The aim was to determine how Working Agreement configurations and other independent factors, such as team size, affect PRCT. The results are presented in Table 4.2. The values in the weight column represent the influence of the variable in days: negative values, therefore, imply a decrease in PRCT, while positive values imply an increase. Additionally, the same

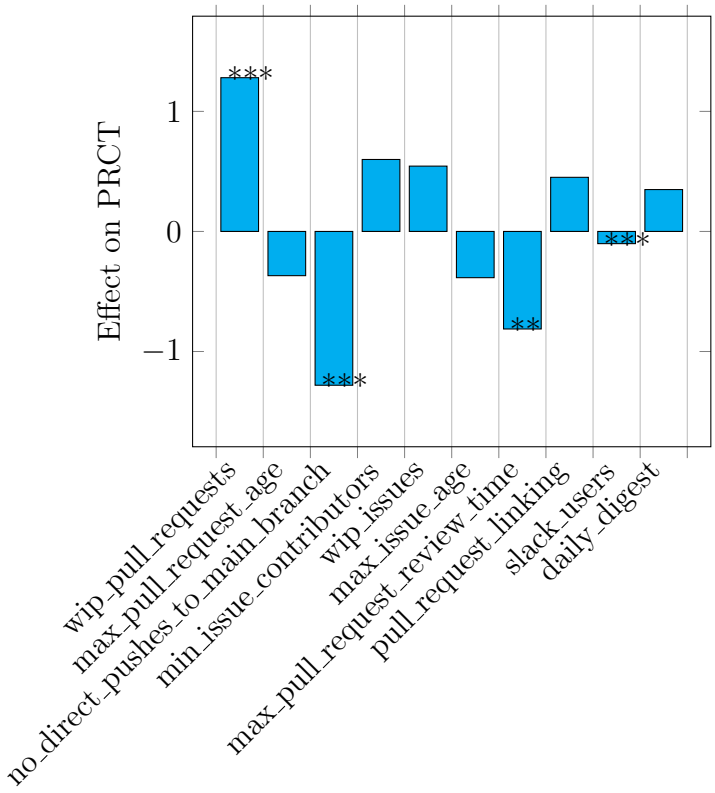


Figure 4.4: Linear regression results for PRCT, one asterisk $*$ denotes $p < 0.10$, two asterisks denotes $p < 0.05$, and three asterisks denote $p < 0.01$.

information is visualized in Figure 4.4, with the asterisks indicating the p-value of the perceived result. When interpreting the results, one should pay attention to the intercepts of each model, as they provide the baseline to which the weights should be compared.

To summarize, regarding pull request -related WAs, we observed that wip_pull_requests has a strong prolonging effect on PRCT ($w = 1.28, p < 0.01$). On the other hand, max_pull_request_review_time demonstrates a moderate decreasing impact on PRCT with ($w = -0.812, p < 0.05$). Although we do not observe a statistical significance in the following variables, the results are as follows: max_pull_request_age ($w = -0.368, p > 0.10$) and pull_request_linking, a unique WA with a connection to both pull requests and issues, ($w = 0.451, p > 0.10$).

When considering issue-related WAs, we observed that no issue-related WA influences PRCT. Although we do not observe a statistically significant result, the weight for the variable min_issue_contributors is ($w = 0.599, p > 0.10$). Similarly, wip_issues has ($w = 0.544, p > 0.10$) and max_issue_age

variable	weight	p-value
wip_pull_requests	1.28^{***}	0.000402
max_pull_request_age	-0.368	0.318
no_direct_pushes_to_main_branch	-1.28^{***}	0.000166
min_issue_contributors	0.599	0.128
wip_issues	0.544	0.17
max_issue_age	-0.385	0.299
max_pull_request_review_time	-0.812^{**}	0.0238
pull_request_linking	0.451	0.197
slack_users	-0.102^{***}	0.00208
daily_digest	0.349	0.224
INTERCEPT	1.38^{***}	$4.84 \cdot 10^{-6}$

Table 4.2: Linear regression results for PRCT, one asterisk $*$ denotes $p < 0.10$, two asterisks denotes $p < 0.05$, and three asterisks denote $p < 0.01$.

($w = -0.385, p > 0.10$). Lastly, a WA not directly connected to either PRs or issues, no_direct_pushes_to_main_branch has a major impact on PRCT with weight ($w = -1.28, p < 0.01$).

The linear model also included a few independent variables outside the WA setup. slack_users has weight ($w = -0.196, p < 0.01$). While WA-related variables have Boolean values, user-related metrics do not: they are measured in absolute team members. Therefore, while the slack_users weight is relatively small, it can have a larger impact on the PRCT than a single WA configuration.

In Progress Cycle Time (IPCT) results are presented in Table 4.3 and Figure 4.5, structured similarly to the PRCT counterparts. Regarding PR-related Working Agreements, the sign and magnitude of the weights are pretty similar as in the PRCT model: wip_pull_requests ($w = 1.55, p < 0.01$) and max_pull_request_review_time ($w = -1.51, p < 0.01$) are both prime examples of this. Again, there are also WAs without statistical significance: max_pull_request_age ($w = -0.235, p > 0.10$) and pull_request_linking ($w = -0.253, p > 0.10$).

When it comes to the issue-related WAs, there was one WA with influence on IPCT: max_issue_age ($w = 1.07, p < 0.01$). Another notable difference in contrast to PRCT results is that in the IPCT model, slack_users does not have statistical significance ($w = -0.0773, p > 0.10$), while daily_digest does

variable	weight	p-value
wip_pull_requests	1.55^{***}	0.0000879
max_pull_request_age	-0.235	0.535
no_direct_pushes_to_main_branch	-1.01^{***}	0.00301
min_issue_contributors	0.682	0.11
wip_issues	0.699	0.0937
max_issue_age	1.07^{***}	0.00597
max_pull_request_review_time	-1.51^{***}	0.000166
pull_request_linking	-0.253	0.502
slack_users	-0.0773	0.0915
daily_digest	-0.595^{**}	0.0481
INTERCEPT	0.570^{*}	0.0355

Table 4.3: Linear regression results for IPCT, one asterisk $*$ denotes $p < 0.10$, two asterisks denotes $p < 0.05$, and three asterisks denote $p < 0.01$.

$(w = -0.595, p < 0.05)$.

The linear models in which In Review Cycle Time (IRCT) and Ready to Merge Cycle Time (RMCT) were used as the dependent variable did not provide statistically significant results.

4.3 Working Agreement Target Effect on Cycle Times

The previously presented models looked into the WA configuration only on top-level: whether an agreement is used. Most Working Agreements require specific goal-setting, such as defining the maximum amount of open pull requests. Two additional models were crafted to gain more insight into the effect of these targets on the team's performance. In these models, the dependent variable was Boolean datatype: whether the team met the goal. The dependent variable values were calculated using the logical statement $cycletime < target$.

These target models aim to measure the relationship between two WAs and their corresponding metrics: max_pull_request_age and PRCT, as well as max_pull_request_review_time and IRCT. The WAs were selected for further

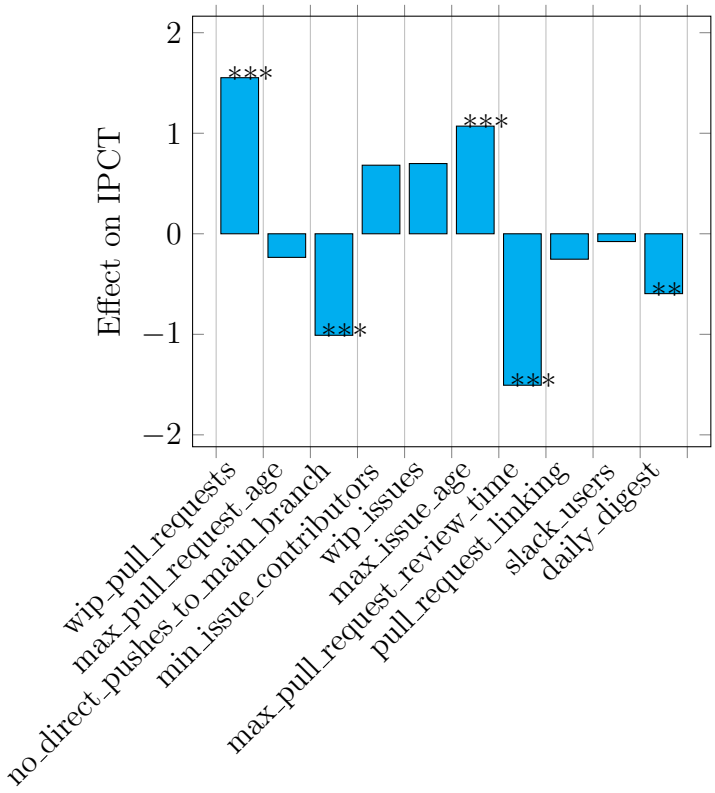


Figure 4.5: Linear regression results for IPCT

analysis as they are PR-related and have well-defined metrics to which data was readily available.

For independent variables, we used previously introduced daily_digest and slack_users, as well as two new ones: days_in_use to denote how many days have passed since the activation of the WA and target_days, the configured goal in full days. As the PRCT is by average significantly longer than IRCT, this also affects target_days: for max_pull_request_age, the recommended configuration is 7 to 14 days, but for max_pull_request_review_time one to two days. When interpreting the model results, it is essential to keep this difference in mind.

Furthermore, as the dependent variable in these models is not continuous, the results must be interpreted differently. We set the threshold for "is False" to $Y < 0.50$ and "is True" to $0.50 \leq Y$, where Y denotes the dependent variable. In contrast to the previous models, a negative weight implies that Y moves towards 0: the target is not met. Similarly, a positive weight means that an increase in the independent variable results in the increase of Y: the target is met more often.

variable	weight	p-value
daily_digest	0.00536	0.761
slack_users	-0.00551^{***}	0.00262
days_in_use	-0.000234^{***}	0.000188
target_days	0.0144^{***}	$2.90 \cdot 10^{-7}$
INTERCEPT	0.768^{***}	≈ 0

Table 4.4: Target model results for PRCT, one asterisk $*$ denotes $p < 0.10$, two asterisks denotes $p < 0.05$, and three asterisks denote $p < 0.01$.

variable	weight	p-value
daily_digest	0.0392	0.121
slack_users	-0.00463^{*}	0.0921
days_in_use	-0.000128^{*}	0.0935
target_days	0.0978^{***}	$2.27 \cdot 10^{-11}$
INTERCEPT	0.435^{***}	$1.91 \cdot 10^{-12}$

Table 4.5: Target model results for IRCT, one asterisk $*$ denotes $p < 0.10$, two asterisks denotes $p < 0.05$, and three asterisks denote $p < 0.01$.

Regarding the PRCT target model, three out of four independent variables proved to influence whether the target was achieved. The variable target_days ($w = 0.0144, p < 0.01$) has by far the most significant weight. Next up is slack_users ($w = -0.00551, p < 0.01$), with a substantial negative weight: in the previous PRCT model, slack_users had a positive on cycle times. Similarly, days_in_use ($w = -0.000234, p < 0.01$) implies that the longer the Working Agreement is used, the less it affects whether the teams meet their targets: although, as the weight is relatively small, the negative impact of time is atomic. Daily Digest ($w = 0.00536, p > 0.10$) did not have a connection to Y in this model.

The results for the IRCT target model are generally similar, even though the p-values are lower for most variables. For this model, target_days ($w = 0.0978, p < 0.01$) has a significant weight, especially considering that the intercept weight is 0.435. Again, slack_users ($w = -0.00463, p < 0.10$) and days_in_use ($w = -0.000128, p < 0.10$) have a small negative effect. As in the PRCT target model, usage of Daily Digest ($w = 0.0392, p > 0.10$) did not help teams achieve their goals on review times.

Chapter 5

Discussion

5.1 Selecting Working Agreements

As with any goal, Working Agreements should be achievable. Unrealistic targets tend to be ignored by the team members and can decrease motivation [12]. That said, it is valuable that teams configure Working Agreements on their own. Not all WAs are suitable for all teams: for example, branching strategy needs to be considered during the WA setup.

Because Working Agreement templates are defined by Swarmia, the client teams have limited capacity to adjust or fine-tune WAs for their needs. Certain presumptions made might make the WAs less valuable or even unusable for some teams: to get the most value out of Swarmia, teams should work with widely accepted CI/CD practices.

At the beginning of a customership, Swarmia helps clients to get the most out of the product. Swarmia employees showcase product features in these meetings and configure initial settings with client representatives. The intention is to get the team started swiftly and help them discover the most relevant features of the product.

An unsuccessful initial setup of Swarmia may cause unwanted effects. For example, a client CTO could be eager to do the initial setup for all teams at once without input from their leads or team members. As stated in Accelerate [11], this is a bad habit: teams should choose their tools and setup. In the onboarding sessions, Swarmia can guide the client representatives with the best practices for WA setup by involving the team leads in the onboarding process. Successful onboarding can form a sound basis for the setup sessions the teams should have on their own.

It remains unclear how teams change their configurations over time. The working methods or changes made to them was not studied. For example,

some teams create WIP-labeled pull requests to inform their teammates of their status or to discuss the upcoming changes. Such a habit leads to a large amount of open PRs and the team should adjust their WIP Working Agreement accordingly. To keep the Working Agreements achievable, teams should review and adjust them regularly. In this study, the models always used the most recent configuration, not the initial ones. This way, the effect of the potential adjustments was included. Intuitively, with periodical iterations to the agreements, teams should find the optimal setup. Unfortunately, the number of adjustments or their frequency remains unknown in this study.

Working Agreements are not intended to be used as primary performance metrics but as a tool that helps teams achieve their objectives. Therefore, it is a safe assumption that teams aim to improve their productivity by enabling WAs. The two most popular WAs, max_pull_request_review_time and max_pull_request_age, aim to reduce the pull request cycle time. It seems teams using Swarmia are keen on increasing their PR throughput. Additionally, many practices that WAs introduce are already used in the teams. By using Working Agreements, teams get up-to-date information on their track record and ensure that the rules are followed in the future.

Most teams enable only a couple of Working Agreements: 64% of teams have configured four or fewer agreements. Teams might test the feature with a couple of agreements and then decide to go all-in if the feature proves helpful. Furthermore, teams usually configure only one Working Agreement per template. A potential reason is related to feature discovery: the possibility of creating multiple agreements might need to be clarified for new teams. The fact that Story is the most popular issue type suggests that most Swarmia teams are acclimated to using Swarmia for Story tracking while Epics, Bugs, and Tasks are tracked via other tools like GitHub – or not at all.

5.2 Effect of Working Agreements

Next, we will analyze the independent variables with statistical significance in the PRCT and IPCT models and the potential underlying reasons for the perceived results. Then, the more specific targets the teams set for the agreements are analyzed.

5.2.1 Effect of Working Agreement Setup

Limiting the number of open PRs with wip_pull_requests does increase PRCT with a weight of 1.28 and notably high confidence. This means that the PR age would increase by 1.3 days. Especially interesting is that it also increases

IPCT, the actual development time, by one and a half days. By introducing this working agreement, teams try to limit pull requests on their desk. As the agreement does not directly control the amount of open PRs, it should not form a bottleneck. In other words, as the agreement is not forced, the effect on developer behavior is indirect: nothing prevents them from creating more PRs than the set maximum in the WA configuration.

To get a deeper insight into the pull request WIP limit, we reflect on the results using Little's law [6]. The law can be formulated $L = \lambda W$, where L is the number of items in the system, λ is the arrival rate, and W is the average time items spend in the system. In this context, we can denote $W = T_{IRCT}$, the average time it takes to review a pull request. As now $W = L/\lambda$, an increase in W would mean that either L has increased or λ has decreased. Based on Little's law, setting a WIP working agreement for pull requests should decrease L and therefore decrease IRCT, given that the arrival rate does not also decrease in the process.

The results for wip_pull_requests conflict with Little's law: no relationship between IRCT and working agreement setup was found. On the other hand, trying to limit the number of PRs in the system (L) caused an increase in PRCT and IPCT. As no relationship between IRCT or RMCT and WA setup was found in the models, it can be assumed that the increase in PRCT resulted from increased IPCT: the developers work on pull requests longer. It seems that the PR limit can result in a larger batch size; the number of changes in a PR increases, which prolongs the development time, IPCT.

While literature suggests that WIP limits are suitable for team productivity [37], in some teams, their overall processes do not support using them. A potential solution would be to intervene earlier before the limit is exceeded. The solution could be implemented with more proactive notifications or in-built limits that disable creating new PRs before the old ones are reviewed and merged. Similar limits are already used in, e.g., GitHub: for example, a team can set a requirement for peer approval before a PR is merged.

By discouraging pushing to the main branch, team members are expected to work through pull requests instead of blindly merging their changes to the main branch. The results imply a strong decreasing effect on both PRCT and IPCT, with the respective weights $w = -1.28$ and $w = -1.01$. Practically speaking, this would shorten PR age by 1.3 days and development time by a day. These results could be caused by creating a superficial working habit: the teams implement a system where pull requests are created but approved without a proper review. In other words, PRs are made for each change to comply with the working agreement and potential branch protection rules. These for-the-sake-of-it PRs pull down the PRCT: without the WA, PRs probably were made only for the more extensive changes. Even though the

rule creates more friction between development and merging to main, reviewing all changes is considered a good practice and should increase code quality. Furthermore, even the rubber-stamped PRs are net-positive for the change history. To get a better touch of this working agreement's potential, we would have to use more relevant metrics, such as the change failure rate.

Pull request review times are considered a common bottleneck in software development [27]. By enabling max_pull_request_review_time, teams try to encourage themselves to focus on closing open PRs faster. Often review times can prolong due to the inactivity of the author or the reviewer. The result, a decrease of 0.81 days in PRCT, proves that the WA reduced review times by almost a day. An interesting result is that the WA also decreased IPCT by -1.51 days: this indirect effect could be caused by the teams' need to decrease the batch size, which should lead to quicker PR reviews. These smaller batches spend by average 1.5 days less time in progress.

Furthermore, max_pull_request_review_time is the only WA the personal Slack notifications address: each team member can opt-in to receive direct messages for PRs pending their contribution. The team daily digest also lists open PRs on top, again bringing them to the attention of developers. According to Maddila et al. [27], actively notifying developers of open PRs shortens PR review time.

The rest of the WAs proved no results in these models that could be trusted due to the lack of statistical significance. Not all Working Agreements aim to reduce cycle times directly, and consequently, there will not be a direct connection. An interesting phenomenon is that PRCT does not have a relationship with any of the issue-related WAs: even though it is a PR-related metric, one would intuitively expect there to be a connection with, for example, wip_issues and PRCT.

5.2.2 Effect of Working Agreement Targets

Finally, the effect of Working Agreement targets is analyzed. The target models for PRCT and IRCT yielded significant results. In these models, the independent variable was modeled as a Boolean metric, as we did not want to give additional weight to the subtraction of target and result: it only matters whether the team achieved their goals. For example, if the team sets their target of PRCT to seven days, both cycle times of five or six days can be seen as evenly valuable results.

The target for the WA, denoted with target_days, has a strong positive effect on the goal achievement. In other words, by setting more loose goals, the teams reach these goals more often. The result is intuitive: with more time to spend, it is easier to be on time. On the other hand, the weight

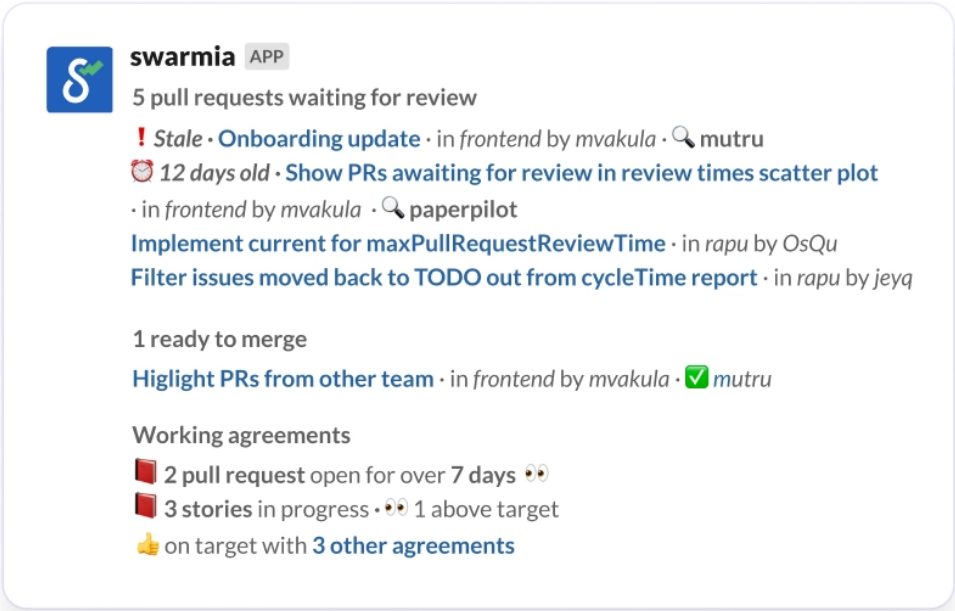


Figure 5.1: Daily Digest, a status report posted by the Slack integration

of the effect provides deeper insight: for PRCT $w = 0.0144$ and for IRCT $w = 0.0978$. Therefore, a ten-day limit for PRCT would affect 0.144 on whether the goal is met. Again, considering that the independent variable has Boolean values, it can be stated that the effect is minor. Furthermore, a typical PRCT limit is measured in weeks and an IRCT limit in days, so even a ten days increase would be considered significant.

This result is in line with Parkinson's law: a task tends to take the time allocated for it [34]. Although, a smaller target can motivate teams to work in smaller batches so that the task at hand might change: for example, a Jira issue might be split into three PRs instead of one, while the overall complexity and time spent might be the same. Unfortunately, the PR contents were not analyzed, so the agreement's effect on batch size remains unknown.

On the contrary, a decreasing effect on the independent variable can be seen with days_in_use, the number of days since WA activation. The weights are for PRCT $w = -0.000234$ and for IRCT $w = -0.000128$. Even though the effect is arguably tiny, it does accumulate over time: the teams are expected to use the product for years, so the weights can become even thousandfold.

5.3 Studying Teams and Individuals

SPACE divides productivity metrics into three levels: individual, team, and system [12]. Of these, this study focuses on the team level while also including some aspects from the individual viewpoint. Next, we will review the findings related to three metrics that deal with teams and individuals: slack_users, github_authors, and daily_digest.

The number of weekly PR authors, which we assume correlates strongly with team size, did not have a relationship to the cycle times and was, in the end, removed from the models due to an excessively high correlation with slack_users.

In previous work, authors have argued that smaller software development teams boost individual performance and operate with smaller total effort [31, 47]. Although larger team size and changes in team composition are often considered performance degrading factors, the results indicate that team size and cycle times have no direct connection.

Regarding the team-level results, we must highlight that we use metrics that not all teams aim to improve while others actively do. As metrics influence behavior [12, 45], the teams that measure cycle times daily might perform differently from those that do not.

5.3.1 Team Metrics and Pull Request Cycle Time

As for the PRCT model, slack_users has a weight of -2.45 hours per user. For a five-person team, this means that if each team member enables Slack notifications, PRCT decreases by 12 hours. The Slack messages nudge the individuals to review their peers' pull requests, reducing the gap between review request and change approval, a common gatekeeper for merging the change.

On the other hand, the team-wide notification Daily Digest does not influence PRCT. Teams integrate the Daily Digest into their current communication channel or create a dedicated channel. As with any high-volume medium, these channels can be flooded with content, causing the notifications to be missed or ignored by the team members.

The direct message continues to be an efficient way to reach individual contributors. Companies implementing direct message notifications should be careful to refrain from spamming developers needlessly [27]. Based on these results, Swarmia has created a valuable personal notification experience.

5.3.2 Team Metrics and In Progress Cycle Time

For the IPCT model, the situation is the opposite: slack_users does not have statistical significance, but daily_digest does, with $w = -0.595$. The result implies that team-level notifications decrease a single PR's development time by 0.6 days, while personal notifications have no effect. The lack of relationship between slack_users and IPCT is as expected: the direct message notifications only deal with the steps after opening a PR when IPCT has ended. The Daily Digest instead includes information about ongoing development tasks: the issues in progress are fetched from the issue tracker and summarized in this report.

The reasons behind this effect could be associated with peer pressure. Posting a public report daily to the Slack channel, the Daily Digest surfaces each issue's status for the team. Although teams can go through the same topics manually in the daily standup, the Daily Digest makes this update more structured and based on actual issue tracker data. The practice should create a positive sense of urgency for the individual contributors, and increase the development speed. Additionally, the team members can notice bottlenecks earlier and support their peers.

As slack_users correlates strongly with team size, it is interesting to notice how it has a decreasing effect on PRCT but not on IPCT. To generalize, the number of team members shortens the average pull request lifetime without affecting the actual development time. The gains are therefore found in the post-development steps: code review and merge. It seems that larger teams can find time to review each other's code with less lag and merge the changes faster after a green light from the reviewer.

Prior empirical research has shown that increasing team size does not linearly increase development cost [35] and that teams with less than nine members tend to be more productive overall [38]. Furthermore, Heričko et al. argue that for each project, there exists an optimal team size [17]. The results found here can provide direction for future research to determine in which stages the larger team size can be beneficial.

5.3.3 Targets and Team Metrics

The team metrics were used as dependent variables in the target models. Again, these weights should be interpreted with the Boolean dependent variable in mind, not as days as in the previous results. The usage of Daily Digest did not have a statistical correlation for either of the targets researched. This result is in line with the previous findings: Daily Digest does not influence PRCT or IRCT directly but affects IPCT. These results further highlight

the lack of statistical connection between Daily Digest and the targets the teams have set for PRCT and IRCT with the associated WAs, even though the Daily Digest especially emphasizes showcasing how teams are doing with their targets.

On the other hand, the effect of slack_users can be seen in these models, which is surprisingly contradictory with the previous results: the number of Slack notification users decreases whether the goals were met. The results are for PRCT $w = -0.00551$ and IRCT $w = -0.00463$, with a slightly negative impact on both. Especially the results regarding PRCT targets raise questions, as we have seen a positive effect with the PRCT itself and slack_users but not with the PRCT-related target.

Based on the results, larger teams perform better in terms of pull request cycle time but miss their targets related to it more often: they set too ambitious targets while performing quite well in absolute metrics. The added complexity of more team members might cause this, as it makes estimation harder.

5.4 Directions for Future Work

In Accelerate, the authors argue [11] that teams should be able to choose their methods and tools. Even though the study only handles monthly active users, it is hard to know their commitment to Swarmia: did they collectively decide to use such a tool and if they did, were the Working Agreements selected as a team or by a manager. The results could improve organization and team onboarding for various SaaS tools.

Moreover, the teams' overall motivation for improvement remains to be discovered. The perceived need for change within the team plays a crucial role in the success of organizational change [25]. The teams' background, focusing on the individual contributors' views on change, could provide more depth to the analysis. Additionally, the effect of the organization characteristics remains to be discovered: for example, companies with different technology stacks, amount of technical dept, or type of product might benefit from a different WA setup.

One of the key findings, the effect of slack_users, would deserve further analysis. Currently, personal notifications are measured on the top level: whether they are used or not. Variations of the notification could be researched to understand better which notifications have an impact. Future studies could include comparing competing tools, such as GitHub's Slack integration [42], Microsoft's Nudge Service [27], and Swarmia. Furthermore, the length of the effect would need further validation – is the boost achieved

by using Slack notifications persistent.

The most exciting direction of further research is tracking the ways of working that, at the time of writing, can not be configured with Swarmia. These norms can be implemented very differently in teams and are more challenging to track objectively than the WAs in Swarmia. However, they admittedly make up most of the teams' ways of working. For example, potential themes for these so-called non-technical Working Agreements could be meeting practices or communication guidelines.

5.5 Limitations of Work

Regarding internal validity, there are a few relevant remarks. Most notably, this thesis uses cycle times as the primary metric. Even though reducing cycle times is a goal many teams try to achieve, there are also potential downsides, such as reduced code quality. Frameworks like SPACE and DORA metrics highlight the need for multiple team productivity metrics to find an optimal balance: a single activity metric cannot reflect all relevant aspects. For example, the change failure rate, which Swarmia already calculates for the customer teams, would bring more depth into the analysis. Furthermore, self-reported measures such as whether an engineer felt productive today, would be a needed counterweight for activity metrics.

Linear regression models were used, even though $cycletime > 0$ always: the dependent variable would be truly continuous in an ideal situation. Furthermore, the factors used as independent variables cannot be considered to constitute all the potential factors that may influence PRCT. Factors such as team members' experience or whether teams operate remotely could affect team productivity but were not included in this study. When using a linear model, it is worth noticing that the relationship between independent and dependent variables will not be linear indefinitely. For example, even though adding another Slack user reduces PRCT by 2.45 hours, it does not mean that adding 1000 new users reduces the time by thousands of hours.

As this is a quantitative study, we rely solely on data available on Swarmia business intelligence tools. Conducting user interviews or surveys would unlock many avenues currently only asserted speculatively. For example, studying perceived performance is more labor-intensive but can shed light on otherwise hard-to-measure dimensions. How people feel they are making progress is a proxy of productivity [12]. A positive attribute of the study is that it was conducted using historical data: the teams were unaware that their usage logs would be used as thesis material.

Regarding external validity, there are multiple weak points in the study

setting. Although we set out to investigate how the Swarmia sub-feature affects team behavior, the aim is to be able to generalize the findings to any other ways of working system. Using Swarmia's Working Agreements is presumably more effective than non-technical implementations, as it actively keeps track of the agreements and nudges team members to stay on track. Regarding generalizability, the results achieved in this study are not directly derivative of other team norm systems. Furthermore, a tool like Swarmia attracts certain companies more than others, causing selection bias for the study. If these organizations are willing to invest in such a tool, they might have more motivation to improve than the companies that decide against using Swarmia or similar tools.

The two-year data collection period overlaps significantly with the coronavirus pandemic, which more or less affected any technology company. Changes like layoffs, remote work, or pivots cause significant effects on team dynamics. These probable effects must be considered with the available data. Even though Swarmia's client base is international and consists of all kinds of companies, the effect of a pandemic can not be ruled out completely.

Chapter 6

Conclusion

In this thesis we explored the use of Working Agreements and whether it influenced software development cycle times. The study was conducted using data from the Swarmia platform, which is a SaaS product teams use to improve their performance. Swarmia feature called Working Agreements and its relation to team performance was analyzed. The analyses were quantitative, where the way teams configure Working Agreements and the configuration's relationship with the team's software development cycle times were researched. The methods used included SQL aggregates and multiple linear regression models.

To summarize, the research questions and their answers are as follows.

For RQ1, "How do teams use Working Agreements?", we observed that a majority of teams configured four or fewer Working Agreements. The most popular WAs are limits for PR cycle time and PR review time. Regarding issue types, teams tend to gravitate towards Working Agreements that deal with stories.

For RQ2, "What is the effect of Working Agreements on software development cycle times?" we observed that PRCT and IPCT are shown to increase when teams limit the number of open pull requests. On the other hand, discouraging direct pushes to the main branch and limiting pull request review times decreased these metrics. Regarding team characteristics, more team members using Slack notifications reduced the PRCT, and using Daily Digest, a team-level daily report, reduced IPCT. Furthermore, larger teams were able to reduce their review times more often.

The positive effect of longer targets on whether teams achieved them was observed to be minimal. This finding is supported by Parkinson's law [34]: all allocated time tends to be spent on a task, or the task adjusts to fit the time. On the other hand, some of the results failed to reflect the findings from prior

research. Based on Little's law [6], limiting open PRs should decrease the PR review time, but no statistical correlation was found. Teams with more slack_users missed their targets more often but performed well on absolute cycle times. Larger teams, therefore, set too ambitious targets. The time the WA has been in use had a decreasing effect on whether teams met their targets, although much smaller than anticipated.

The top-level practical implication of the study is that certain working agreements do change behavior. Moreover, the result indicates that change requires more than metrics and the importance of setting meaningful targets. Multiple connections between a newly introduced metric and a changed behavior were reported. Most notably, the value of individual Slack notifications, team-level daily reports, and the PR review time limit was observed.

As a part of future work, a more thorough background profile for teams would provide a more complete view of the teams' situation, allowing, for example, research on what kind of WAs work for different teams. The value of DM notifications has been shown by Maddila et al. [27] and supported by this study. It would be interesting to see a more detailed analysis of the most efficient notifications. Furthermore, explicit ways of working that currently can not be configured in Swarmia are more challenging to track objectively but could provide valuable insights.

Bibliography

[1] A. Alshamrani and A. Bahattab. A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model. *International Journal of Computer Science Issues*, 12(1):7, 2015.

[2] S. Balaji. Waterfall vs V-model vs Agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management*, 2(1):5, 2012.

[3] B. W. Boehm. Software engineering - as it is. In *Proceedings of the 4th international conference on Software Engineering*, ICSE '79, pages 11–21. IEEE Press, 1979.

[4] L. Chen. Continuous Delivery: Huge Benefits, but Challenges Too. *IEEE Software*, 32(2):50–54, Mar. 2015. ISSN 1937-4194. doi: 10.1109/MS.2015.27.

[5] W. Chew. No-Nonsense Guide to Measuring Productivity. *Harvard Business Review*, 66(1), Jan. 1988. ISSN 0017-8012.

[6] D. Chhajed, T. J. Lowe, and F. S. Hillier, editors. *Building Intuition: Insights From Basic Operations Management Models and Principles*, volume 115 of *International Series in Operations Research & Management Science*. Springer US, Boston, MA, 2008. ISBN 978-0-387-73698-3 978-0-387-73699-0. doi: 10.1007/978-0-387-73699-0. URL `http://link.springer.com/10.1007/978-0-387-73699-0`.

[7] D. Cohen, M. Lindvall, and P. Costa. An Introduction to Agile Methods. In *Advances in Computers*, volume 62, pages 1–66. Elsevier, 2004. ISBN 978-0-12-012162-5. doi: 10.1016/S0065-2458(03)62001-2. URL `https://linkinghub.elsevier.com/retrieve/pii/S0065245803620012`.

[8] C. A. Cois, J. Yankel, and A. Connell. Modern DevOps: Optimizing software development through effective system interactions. In *2014*

IEEE International Professional Communication Conference (IPCC), pages 1–7, Oct. 2014. doi: 10.1109/IPCC.2014.7020388. ISSN: 2158-1002.

[9] G. Cugola and C. Ghezzi. Software processes: a retrospective and a path to the future. *Software Process: Improvement and Practice*, 4(3):101–123, Sept. 1998. ISSN 1077-4866, 1099-1670. doi: 10.1002/(SICI)1099-1670(199809)4:3⟨101::AID-SPIP103⟩3.0.CO;2-K. URL `https://onlinelibrary.wiley.com/doi/10.1002/(SICI)1099-1670(199809)4:3<101::AID-SPIP103>3.0.CO;2-K`.

[10] E. Derby and D. Larsen. *Agile retrospectives: making good teams great*. Pragmatic Bookshelf, Raleigh, NC, 2006. ISBN 978-0-9776166-4-0. OCLC: ocm71756468.

[11] N. Forsgren, J. Humble, and G. Kim. *Accelerate: the science behind DevOps: building and scaling high performing technology organizations*. IT Revolution, Portland, Oregon, first edition, 2018. ISBN 978-1-942788-33-1.

[12] N. Forsgren, M.-A. Storey, C. Maddila, T. Zimmermann, B. Houck, and J. Butler. The SPACE of Developer Productivity. *ACM Queue*, 1 (January-February 2021), Feb. 2021.

[13] M. Fowler and J. Highsmith. The Agile Manifesto. *Software Development*, 9(8):28–35, 2001.

[14] A. M. M. Hamed and H. Abushama. Popular agile approaches in software development: Review and analysis. In *2013 INTERNATIONAL CONFERENCE ON COMPUTING, ELECTRICAL AND ELECTRONIC ENGINEERING (ICCEEE)*, pages 160–166, Khartoum, Sudan, Aug. 2013. IEEE. ISBN 978-1-4673-6232-0 978-1-4673-6231-3. doi: 10.1109/ICCEEE.2013.6633925. URL `http://ieeexplore.ieee.org/document/6633925/`.

[15] Haystack. Haystack - Analytics for Engineering Leaders, 2022. URL `https://www.usehaystack.io/`. Accessed: 17.12.2022.

[16] A. Hemon-Hildgen, B. Lyonnet, F. Rowe, and B. Fitzgerald. From Agile to DevOps: Smart Skills and Collaborations. *Information Systems Frontiers*, 22, Aug. 2020. doi: 10.1007/s10796-019-09905-1.

[17] M. Heričko, A. Živkovič, and I. Rozman. An approach to optimizing software development team size. *Information Processing Letters*, 108(3):

101–106, Oct. 2008. ISSN 00200190. doi: 10.1016/j.ipl.2008.04.014. URL https://linkinghub.elsevier.com/retrieve/pii/S0020019008001130.

[18] J. Humble and J. Molesky. Why Enterprises Must Adopt Devops to Enable Continuous Delivery. *Cutter IT Journal*, 24(8):7, 2011.

[19] IEEE. *IEEE Std 610.12-1990*. IEEE, 1990. ISBN 978-0-7381-0391-4. OCLC: 1322558124.

[20] Jellyfish. Align Engineering Work with Business Priorities, 2022. URL https://jellyfish.co/. Accessed: 17.12.2022.

[21] J. Katzenbach and D. Smith. The Discipline of Teams. *Harvard Business Review*, Apr. 1993. URL https://www.ucipfg.com/Repositorio/GSPM/Cursos/PFC_GSPM_03/Lectura%202.pdf.

[22] J. P. Kotter. Leading change. *Harvard Business Review*, 2 (1):1–10, 1995. URL https://www.sobtell.com/images/questions/1495988227-20160927003643why_transformation_efforts_fail.pdf.

[23] J. Krafcik. Triumph of a Lean Production System. *Sloan Management Review*, 30(Fall 1988):41–52, 1988. URL https://www.lean.org/downloads/MITSloan.pdf.

[24] C. Lassenius, T. Dingsøyr, and M. Paasivaara, editors. *Agile Processes in Software Engineering and Extreme Programming: 16th International Conference, XP 2015, Helsinki, Finland, May 25-29, 2015, Proceedings*, volume 212 of *Lecture Notes in Business Information Processing*. Springer International Publishing, Cham, 2015. ISBN 978-3-319-18611-5 978-3-319-18612-2. doi: 10.1007/978-3-319-18612-2. URL http://link.springer.com/10.1007/978-3-319-18612-2.

[25] P. Lenberg, L. G. Wallgren Tengberg, and R. Feldt. An initial analysis of software engineers' attitudes towards organizational change. *Empirical Software Engineering*, 22(4):2179–2205, Aug. 2017. ISSN 1382-3256, 1573-7616. doi: 10.1007/s10664-016-9482-0. URL http://link.springer.com/10.1007/s10664-016-9482-0.

[26] LinearB. Developer Workflow Optimization | LinearB, 2022. URL https://linearb.io/. Accessed: 17.12.2022.

[27] C. Maddila, S. S. Upadrasta, C. Bansal, N. Nagappan, G. Gousios, and A. van Deursen. Nudge: Accelerating Overdue Pull Requests Towards Completion. *ACM Transactions on Software Engineering*

and Methodology, page 3544791, June 2022. ISSN 1049-331X, 1557-7392. doi: 10.1145/3544791. URL `http://arxiv.org/abs/2011.12468`. arXiv:2011.12468 [cs].

[28] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering*, 21(5):2146–2189, Oct. 2016. ISSN 1382-3256, 1573-7616. doi: 10.1007/s10664-015-9381-9. URL `http://link.springer.com/10.1007/s10664-015-9381-9`.

[29] N. Moe, T. Dingsøyr, and T. Dybå. Overcoming Barriers to Self-Management in Software Teams. *IEEE Software*, 26:20–26, Jan. 2010. doi: 10.1109/MS.2009.182.

[30] N. B. Moe, T. Dingsøyr, and T. Dybå. A teamwork model for understanding an agile team: A case study of a Scrum project. *Information and Software Technology*, 52(5):480–491, May 2010. ISSN 09505849. doi: 10.1016/j.infsof.2009.11.004. URL `https://linkinghub.elsevier.com/retrieve/pii/S0950584909002043`.

[31] A. Mundra, S. Misra, and C. A. Dhawale. Practical Scrum-Scrum Team: Way to Produce Successful and Quality Software. In *2013 13th International Conference on Computational Science and Its Applications*, pages 119–123, June 2013. doi: 10.1109/ICCSA.2013.25.

[32] Naur, Peter and Randell, Brian. Software engineering. In *Software engineering: Report of a conference sponsored by the NATO science committee, Garmisch, Germany, 7th-11th October 1968*, page 136, 1969.

[33] E. Oliveira, E. Fernandes, I. Steinmacher, M. Cristo, T. Conte, and A. Garcia. Code and commit metrics of developer productivity: a study on team leaders perceptions. *Empirical Software Engineering*, 25(4):2519–2549, July 2020. ISSN 1382-3256, 1573-7616. doi: 10.1007/s10664-020-09820-z. URL `https://link.springer.com/10.1007/s10664-020-09820-z`.

[34] Parkinson, Cyril. Parkinson's Law. *The Economist*, Nov. 1955. ISSN 0013-0613. URL `https://www.economist.com/news/1955/11/19/parkinsons-law`.

[35] P. C. Pendharkar and J. A. Rodger. The relationship between software development team size and software development cost. *Communications of the ACM*, 52(1):141–144, Jan. 2009. ISSN 0001-0782, 1557-7317.

doi: 10.1145/1435417.1435449. URL https://dl.acm.org/doi/10.1145/1435417.1435449.

[36] N. Rehman, A. Mahmood, M. Ibtasam, S. A. Murtaza, N. Iqbal, and E. Molnár. The Psychology of Resistance to Change: The Antidotal Effect of Organizational Justice, Support and Leader-Member Exchange. *Frontiers in Psychology*, 12:678952, Aug. 2021. ISSN 1664-1078. doi: 10.3389/fpsyg.2021.678952. URL https://www.frontiersin.org/articles/10.3389/fpsyg.2021.678952/full.

[37] D. G. Reinertsen. *The principles of product development flow: second generation lean product development*. Celeritas Publishing, Redondo Beach, California, 2009. ISBN 978-1-935401-00-1.

[38] D. Rodríguez, M. Sicilia, E. García, and R. Harrison. Empirical findings on team size and productivity in software development. *Journal of Systems and Software*, 85(3):562–570, Mar. 2012. ISSN 01641212. doi: 10.1016/j.jss.2011.09.009. URL https://linkinghub.elsevier.com/retrieve/pii/S0164121211002366.

[39] J. Rosenberg. Some misconceptions about lines of code. In *Proceedings Fourth International Software Metrics Symposium*, pages 137–142, Nov. 1997. doi: 10.1109/METRIC.1997.637174.

[40] W. W. Royce. Managing the development of large software systems. In *Proceedings of the 9th international conference on Software Engineering*, pages 1–9, Aug. 1970.

[41] Scrum.org. The Home of Scrum, 2022. URL https://www.scrum.org/index. Accessed: 17.12.2022.

[42] Slack Technologies. integrations/slack: Bring your code to the conversations you care about with the GitHub and Slack integration, Nov. 2022. URL https://github.com/integrations/slack.

[43] S. T. Solansky. Leadership Style and Team Processes in Self-Managed Teams. *Journal of Leadership & Organizational Studies*, 14(4):332–341, May 2008. ISSN 1548-0518. doi: 10.1177/1548051808315549. URL https://doi.org/10.1177/1548051808315549. Publisher: SAGE Publications Inc.

[44] I. Sommerville. *Software engineering*. Always learning. Pearson, Boston Munich, 10. ed., global ed edition, 2016. ISBN 978-1-292-09613-1.

[45] M.-A. Storey, B. Houck, and T. Zimmermann. How Developers and Managers Define and Trade Productivity for Quality, Apr. 2022. URL http://arxiv.org/abs/2111.04302. arXiv:2111.04302 [cs].

[46] V. Stray, T. E. Fægri, and N. B. Moe. Exploring Norms in Agile Software Teams. In *Product-Focused Software Process Improvement*, volume 10027, pages 458–467. Springer International Publishing, Cham, 2016. ISBN 978-3-319-49093-9 978-3-319-49094-6. doi: 10.1007/978-3-319-49094-6_31. URL http://link.springer.com/10.1007/978-3-319-49094-6_31. Series Title: Lecture Notes in Computer Science.

[47] J. Sutherland, N. Harrison, and J. Riddle. Teams That Finish Early Accelerate Faster: A Pattern Language for High Performing Scrum Teams. In *2014 47th Hawaii International Conference on System Sciences*, pages 4722–4728, Waikoloa, HI, Jan. 2014. IEEE. ISBN 978-1-4799-2504-9. doi: 10.1109/HICSS.2014.580. URL http://ieeexplore.ieee.org/document/6759182/.

[48] Swarmia. Gain visibility. Remove blockers. Ship 10x faster. | Swarmia, 2022. URL https://www.swarmia.com/. Accessed: 17.12.2022.

[49] Swarmia. Reducing pull request cycle time, 2022. URL https://help.swarmia.com/pull-request-cycle-time-in-swarmia. Accessed: 24.8.2022.

[50] C. Symons. Software Industry Performance: What You Measure Is What You Get. *IEEE Software*, 27(6):66–72, Nov. 2010. ISSN 0740-7459. doi: 10.1109/MS.2009.162. URL http://ieeexplore.ieee.org/document/5235133/.

[51] C. Syverson. What Determines Productivity? *Journal of Economic Literature*, 49(2):326–365, June 2011. ISSN 0022-0515. doi: 10.1257/jel.49.2.326. URL https://pubs.aeaweb.org/doi/10.1257/jel.49.2.326.

[52] S. Tangen. Demystifying productivity and performance. *International Journal of Productivity and Performance Management*, 54(1):34–46, Jan. 2005. ISSN 1741-0401. doi: 10.1108/17410400510571437. URL https://www.emerald.com/insight/content/doi/10.1108/17410400510571437/full/html.

[53] A. Teh, E. Baniassad, D. van Rooy, and C. Boughton. Social Psychology and Software Teams: Establishing Task-Effective Group Norms. *IEEE*

Software, 29(4):53–58, July 2012. ISSN 1937-4194. doi: 10.1109/MS.2011.157.

[54] D. J. Terry, M. A. Hogg, and B. M. McKimmie. Attitude-behaviour relations: The role of in-group norms and mode of behavioural decision-making. *British Journal of Social Psychology*, 39(3):337–361, Sept. 2000. ISSN 01446665. doi: 10.1348/014466600164534. URL `http://doi.wiley.com/10.1348/014466600164534`.

[55] R. Todnem. Organisational Change Management: A Critical Review. *Journal of Change Management*, 5(4):369–380, Dec. 2005. doi: 10.1080/14697010500359250.

[56] Wells, Don. Extreme Programming: A Gentle Introduction., Oct. 2018. URL `http://www.extremeprogramming.org/`. Accessed: 17.12.2022.

[57] T. Ōno. *Toyota production system: beyond large-scale production*. Productivity Press, Cambridge, Mass, 1988. ISBN 978-0-915299-14-0.