Saku Mäkinen

ENHANCING SOFTWARE DEVELOPMENT THROUGH VALUE STREAM MAPPING - A CASE STUDY

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2024

ABSTRACT

Mäkinen, Saku
Enhancing Software Development Processes Through Value Stream
Mapping – A Case Study
Jyväskylä: University of Jyväskylä, 2024, 64 pp.
Information Systems, Master’s Thesis
Supervisor(s): Seppänen, Ville

This thesis investigates how Value Stream Mapping (VSM) can be used to identify waste, bottlenecks, and inefficiencies in software development processes. VSM, first created for the manufacturing sector, has been modified and applied to several industries, including software development, to improve and visualize process flow. Despite its growing popularity, its implementation in software development remains challenging due to the abstract nature of software processes. This study employs a qualitative case study methodology, involving VSM workshops with three software development teams within a global IT company. The research aims to understand the suitability of VSM for software development, its impact on process efficiency, and whether it creates more value than it takes resources to implement. The results show that VSM can offer a structured method for mapping out existing processes, pointing out inefficiencies, and fostering collaboration among teams when brainstorming improvement suggestions and putting them into practice. Key findings show that a lack of pre-planning, cross-functional dependencies, and resource constraints frequently result in significant bottlenecks throughout the development and testing phases. The research highlights the significance of integrating VSM with strong change management methodologies to address resistance and enable seamless process transformations. Using automated technologies to improve productivity and communication, including important stakeholders early in the process, and strengthening feedback loops between development and testing are some of the recommendations. This research improves the understanding of VSM's applicability in software development and offers practical advice to businesses trying to improve productivity and streamline workflows. The study concludes that while VSM can significantly enhance software development workflows in the company, its success relies heavily on the adoption of complementary strategies that address human and organizational dynamics.

Keywords: Value Stream Mapping, Software Development, Lean Management, Change Management, Agile Development, Software Process Optimization.

TIIVISTELMÄ

Mäkinen, Saku
Enhancing Software Development Processes Through Value Stream Mapping – A Case Study
Jyväskylä: Jyväskylän yliopisto, 2024, 64 s.
Tietojärjestelmätiede, pro gradu -tutkielma
Ohjaaja(t): Ville Seppänen

Tässä Pro gradu -tutkielmassa selvitetään, miten Value Stream Mappingia (VSM) voidaan käyttää työkaluna, jolla tunnistetaan turhaa työtä, pullonkauloja ja tehottomuutta ohjelmistokehitysprosesseissa. VSM kehitettiin ensin teollisuusympäristöön. Sitä on myöhemmin muokattu ja sovellettu useille uusille toimialoille, kuten ohjelmistokehityksen prosessien tehostamiseen ja visualisoimiseen. VSM:n kasvavasta suosiosta huolimatta sen käyttäminen ohjelmistokehityksessä on haastavaa ohjelmistoprosessien abstraktien rakenteiden vuoksi. Tässä tutkimuksessa käytetään laadullista tapaustutkimusmetodologiaa. Tutkimusaineiston keräämiseen käytettiin VSM-työpajoja, joihin osallistui yhteensä kolme ohjelmistokehitystiimiä globaalista tietoliikenneyrityksestä. Tutkimuksen tavoitteena oli ymmärtää VSM:n soveltuvuus ohjelmistokehitykseen ja VSM:n vaikutus prosessien tehokkuuteen. Tutkimuksella pyrittiin selvittämään luoko VSM lisäarvoa suhteessa sen implementointiin käytettäviin resursseihin. Tutkimustulokset osoittavat, että VSM on toimiva työkalu olemassa olevien prosessien kartoittamiseen, tehottomuuden havaitsemiseen ja tiimien välisen yhteistyön edistämiseen. Keskeisimmät tutkimustulokset osoittavat, että esisuunnittelun puute, eri toimintojen väliset riippuvuudet ja resurssirajoitteet johtavat usein pullonkauloihin tiimien prosesseissa. Tutkimuksessa havaittuja mahdollisia suosituksia ovat automatisoidun teknologian käyttäminen tuottavuuden ja viestinnän parantamiseksi, mukaan lukien tärkeiden sidosryhmien aktivoiminen prosessin varhaisessa vaiheessa, sekä kehitystyön ja testauksen välisten palautesilmukoiden vahvistaminen. Tämä tutkimus parantaa ymmärrystä VSM:n sovellettavuudesta ohjelmistokehityksessä ja tarjoaa käytännön neuvoja yrityksille, jotka pyrkivät parantamaan tuottavuutta ja virtaviivaistamaan prosessivirtojaan. Tutkimuksen johtopäätöksenä on, että vaikka VSM voi tehostaa yrityksen ohjelmistokehitysprosesseja, sen menestys riippuu vahvasti myös siitä, otetaanko käyttöön strategioita, jotka huomioivat ihmisten ja organisaation dynamiikan.

Asiasanat: Ketterät menetelmät, Ohjelmistokehitys, Ohjelmistokehityksen optimointi

FIGURES

TABLES

TABLE OF CONTEXTS

1 INTRODUCTION

Modern software development has gone through dramatic changes over the past decades. Initially dominated by linear and sequential methodologies such as the Waterfall model, the industry has increasingly shifted toward iterative, cyclical frameworks like agile, Scrum, and DevOps. These methodologies enhance integration, frequent increments, and adaptive planning, which are noteworthy approaches to addressing changes in requirements and enhancing the rate of market deployment.

Despite the adoption of modern software development methodologies, many projects continue to face challenges. These include frequent changes to project scope, communication breakdowns within and between teams, and ineffective resource management. All these might lead to delays and increased costs. These difficulties demonstrate the need for better methods of process management. As a result, businesses are using lean management techniques in increasing numbers. These methods were initially developed for the manufacturing sector but have subsequently been modified for use in a variety of industries, including software development. Value Stream Mapping (VSM), a method for visualizing and optimizing the flow of materials and information needed to produce a product or provide a service, is one of the key elements of lean management.

The goal of lean management is to reduce or eliminate waste while optimizing value for customers. Value-added and non-value-adding operations are identified by VSM, which helps businesses optimize their workflows and enhance productivity. Because VSM may enhance complex development workflows, there has been a lot of interest in implementing it in software development.

Software development processes usually involve several related phases, including testing, deployment, specification, coding, and maintenance. Organizations can use VSM to get a comprehensive understanding of their development processes, spot inefficiencies, and make data-driven choices that will boost output. This method is a useful tool for increasing overall development efficiency since it not only helps identify possible issue areas but also minimizes delays and maximizes the utilization of resources.

There also are some issues that organizations must face when they decide to implement VSM. While manufacturing processes are more open to straightforward mapping, software development is more abstract and is often an iterative process. Implementing VSM can be challenging, as it often requires significant changes to already established workflows, which can lead to resistance from teams. For VSM to be successfully adopted, it is essential to apply effective change management practices that facilitate a smooth transition and encourage team buy-in.

Furthermore, cautious organizational change management is necessary for the successful implementation of VSM within software teams. Theories of change management, including Bridges' Transition Model and Kotter's 8-step model, place a strong emphasis on leading teams through change, dealing with opposition, and developing a common goal. A change in the team's development methodology is frequently necessary for the successful deployment of VSM, which makes change management procedures essential to reaching the intended results.

The use of lean and agile approaches in software development has been the subject of earlier research, which has shown how these approaches can shorten lead times, foster better teamwork, and increase overall productivity. However, the use of VSM in software development is still an emerging area of research. Prior research reveals that while VSM can help detect bottlenecks and waste, its application in software settings is more challenging than in traditional manufacturing due to the abstract nature of software development processes (Khurum, Petersen & Gorschek, 2014; Tyagi Choudhary, Cai &Yang, 2015). Furthermore, it has been emphasized that the deployment of VSM must be accompanied by effective change management practices to guarantee long-term gains (Ali, Petersen & Schneider, 2016).

Despite these insights, a lack of thorough research has been done on the actual software development teams in which VSM is applied, as well as the difficulties that may occur in doing so. By performing an empirical study to determine the suitability of VSM in software development and analyse its influence on process efficiency, this research seeks to close this gap.

Using a qualitative case study methodology, the study conducts VSM workshops inside a global IT company. The empirical data is gathered using a combination of feedback surveys, and the main data is gathered from VSM mapping workshops from three distinct software development teams. The goal of the workshops is to map out each team's present development process, pinpoint any inefficiencies or bottlenecks, and suggest doable fixes. The information is examined to evaluate the difficulties encountered in putting VSM recommendations into practice as well as the efficacy of VSM in locating and resolving process inefficiencies. To assess the effect of VSM on team performance, quantitative metrics like cycle time, resource allocation, and velocity are employed in addition to the qualitative data. These objectives are reflected in the research questions this thesis attempts to answer:

- What are the main pain points of a given company’s software development and how they can be identified by Value Stream Mapping?

- What value does VSM bring to the company's teams and how can this value be realized in the current software development environment?

Following this introduction, Chapter 2 provides an overview of the theoretical background, covering software development methodologies, VSM principles, and change management theories. Chapter 3 details the research design, methodology, and data collection processes. The results from the empirical study are presented in Chapter 4, which includes case descriptions and analysis of the VSM workshops. Chapter 5 discusses the implications of the findings, while Chapter 6 concludes the thesis by summarising the key contributions, limitations, and suggestions for future research.

This thesis benefited from the use of artificial intelligence (AI) tools, specifically for language refinement and rephrasing support. These tools were utilized to enhance the clarity and readability of the text, ensuring the effective communication of the research findings and analyses presented herein.

2 THEORETICAL BACKGROUND

This narrative literature review looks at important ideas and frameworks that can help shape future research. It carefully examines and brings together these concepts to provide a solid foundation for what is to come. By building on previous studies, a comprehensive understanding is established to provide a solid foundation for subsequent investigations. Articles used are searched through Google Scholar and JYKDOK. The articles point out that being timely and reliable is key for credibility. Reliability of the articles is checked by looking at a JUFO classification of at least 1, along with how often an article is cited and the quality of those citations. These factors reflect each article's academic impact and acceptance within the scholarly community.

2.1 Software Development Processes

In this chapter, we are studying what is software development, and what are the most common practices and processes in software development. Also, we are going to look at the most common challenges that are faced during the software development processes.

2.1.1 Overview of Software Development

Software development is a discipline that has undergone many improvements through time, and it still does when a new strategy, way of work, or technology is introduced. It aims to keep up with the new advancements in technology and modern business requirements as efficiently as possible. The goal of software development is to reach the final software product. There are different ways to reach that goal and every company and person working on a software product is trying to find the best processes to build the final product.

The term “Software Engineering” dates back to 1968 and NATO where a study group that focused on addressing software issues and promoting its use in various aspects of life (Al-Sagga, Swahla & Abdelnabi, 2020). Like other engineering disciplines, software is created for specific purposes. IEEE (1990) describes software engineering as "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.". In a software development project, many steps need the approach of systematic, disciplined, and quantifiable ways to build the software product to produce the best software product possible. The end product should be as reliable as possible, easy to maintain, and meet the business requirements that are based on the same criteria (Braude and Bernstein, 2016). Software development is done by teams and needs strong cooperation between the teams and individuals (Giuffrida & Dittrich, 2015). Tasks are often distributed across several teams. These tasks are managed and prioritized based on specific criteria. Some tasks can be completed in parallel; others depend on the completion of preceding tasks. Coordination among these tasks, processes, and teams is essential to achieving the best software product possible at minimal cost and time (Giuffrida & Dittrich, 2015).

Various approaches have developed over time to handle the difficulties and complexity that come with developing software. Modern systems like agile, lean, and Scrum encourage flexibility, adaptability, and frequent collaboration while classic approaches like the Waterfall model emphasize thorough preparation and documentation. It is essential to comprehend different approaches to choose the best strategy for a particular project.

2.1.2 Traditional Software Development: The Waterfall Method

There are many traditional methods to develop software products. There are for example the waterfall approach, iterative and incremental approach, spiral approach, and evolutionary approach. They are often referred to as planned or heavyweight approaches which implement these approaches are particularly useful for developing large, complex software systems, as they help eliminate outdated, informal development practices and deliver high-quality software systematically, meeting user requirements within a predefined timeframe (Matharu, Mishra, Singh & Upadhayay, 2015). On the other hand, the traditional software development process has many downfalls due to its heavy size and other disadvantages such as:

1. Limited flexibility, because the project plan is completed before starting the project
2. Lack of customer involvement during the project
3. High-risk due to testing and validation being at the end of the process
4. Lengthy development cycle (Pargoangar, 2023).

Many traditional projects fail because of their size and changes during the project. Big projects fail due to their big size and long development time. Once the plan is finished and the development starts the business requirements might have changed because of the fast-changing business environment. For example, in the waterfall approach changes are difficult and costly to make during lengthy development. In the planning phase project management would be able to "see the future" and predict problems upfront to make the software product effective to the business (Jammalamadaka and Krishna, 2013). If the requirements change the work that has been done might be for nothing and development might have to be planned and start all over again. In many cases, the software product that is finished might be "old" because of the changing environment and long development time.

The Waterfall Method (Figure 1) is the most traditional way of software development, and it is linear and simple to understand. The Waterfall model is a static approach to systems development that progresses linearly and sequentially, completing one activity before moving on to the next one (Adenowo & Adenowo, 2013). Fowler (2018) thinks that the waterfall method stages can be broken up based on the following activities: requirement analysis, design, coding, and testing. Pressman (2005) identifies the steps of the waterfall method as: communication (project initiation and requirements gathering), planning (estimating, scheduling, and tracking), modeling (analysis and design), construction (coding and testing), and deployment (delivery, support, and feedback). Pfleeger and Atlee (2006) describe the model as including the following phases: requirement analysis, system design, program design, coding, unit and integration testing, system testing, acceptance testing, and operation and maintenance.

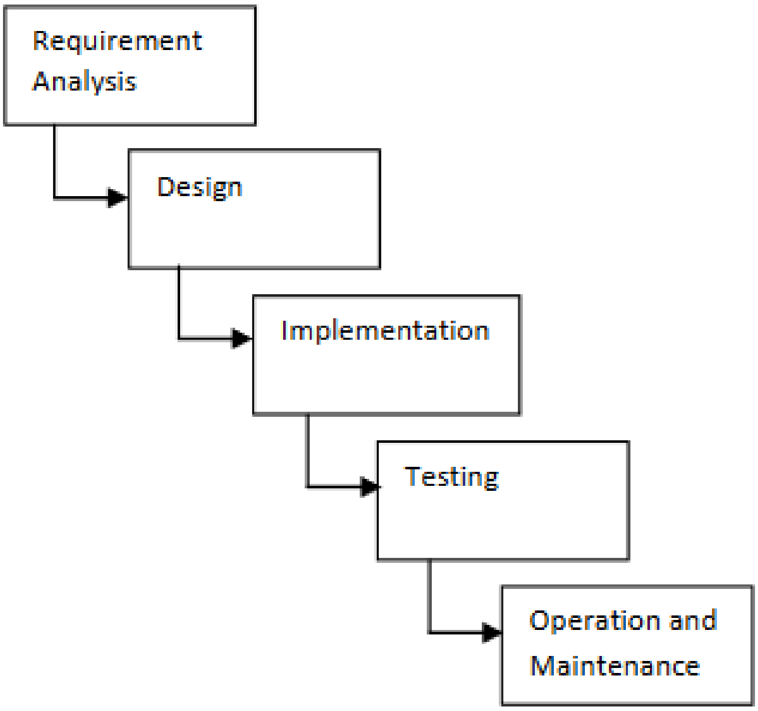
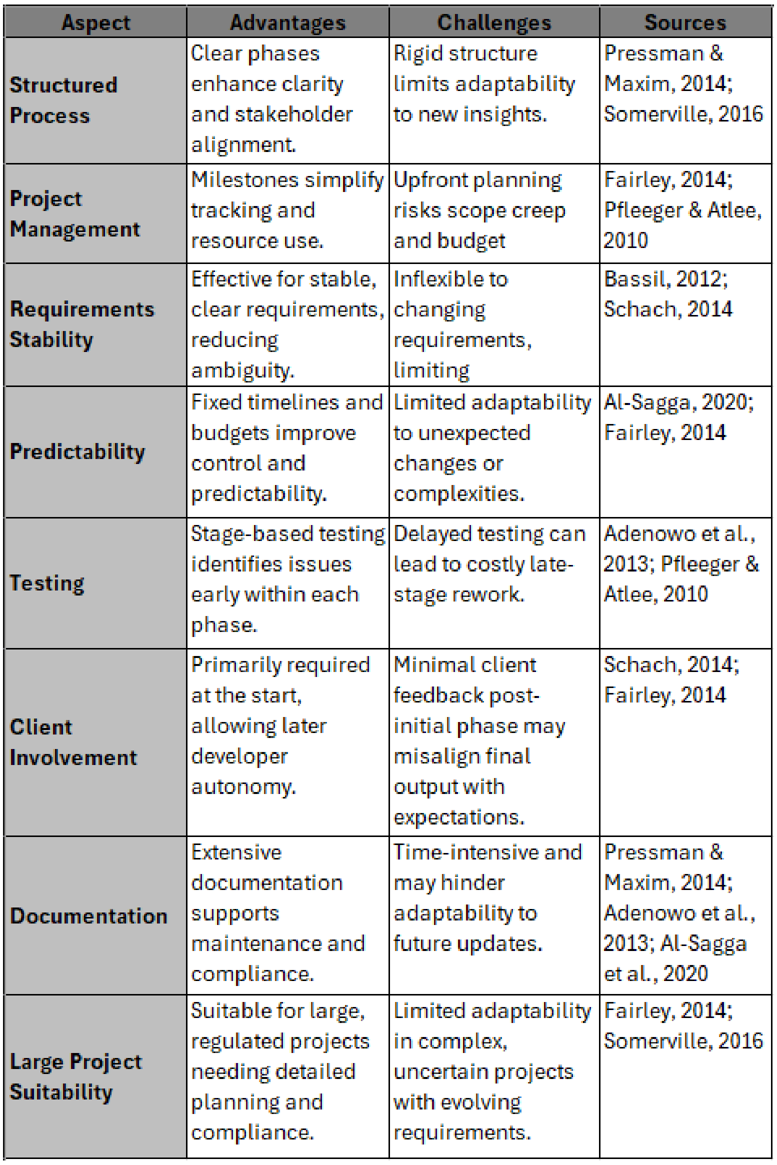


Figure 1 The phases of a Waterfall Model (Pfleger & Atlee, 2006)

According to Pressman and Maxim (2014), the Waterfall technique is a linear approach to software development that divides projects into successive phases that offer structure and predictability. This paradigm works well for projects with consistent needs because it provides transparency and control through well-defined milestones and thorough documentation (Somerville, 2016). However, because of its rigidity, it can be difficult to change to changing needs, and if problems are found late in the development process, the delayed testing phase may result in expensive adjustments (Pfleeger & Atlee, 2010). The Waterfall model's main strengths and drawbacks are listed in the following table, which also shows how it works well in organized settings and how it falls short in more dynamic settings highlights the situations in which it works best and any restrictions that might affect its applicability for certain projects. Table 1 provides a comprehensive overview of the advantages and disadvantages of the waterfall method.

Table 1 The waterfall method’s advantages and disadvantages



Aspect	Advantages	Challenges	Sources
Structured Process	Clear phases enhance clarity and stakeholder alignment.	Rigid structure limits adaptability to new insights.	Pressman & Maxim, 2014; Somerville, 2016
Project Management	Milestones simplify tracking and resource use.	Upfront planning risks scope creep and budget	Fairley, 2014; Pfleeger & Atlee, 2010
Requirements Stability	Effective for stable, clear requirements, reducing ambiguity.	Inflexible to changing requirements, limiting	Bassil, 2012; Schach, 2014
Predictability	Fixed timelines and budgets improve control and predictability.	Limited adaptability to unexpected changes or complexities.	Al-Sagga, 2020; Fairley, 2014
Testing	Stage-based testing identifies issues early within each phase.	Delayed testing can lead to costly late-stage rework.	Adenowo et al., 2013; Pfleeger & Atlee, 2010
Client Involvement	Primarily required at the start, allowing later developer autonomy.	Minimal client feedback post-initial phase may misalign final output with expectations.	Schach, 2014; Fairley, 2014
Documentation	Extensive documentation supports maintenance and compliance.	Time-intensive and may hinder adaptability to future updates.	Pressman & Maxim, 2014; Adenowo et al., 2013; Al-Sagga et al., 2020
Large Project Suitability	Suitable for large, regulated projects needing detailed planning and compliance.	Limited adaptability in complex, uncertain projects with evolving requirements.	Fairley, 2014; Somerville, 2016

Some things unite the waterfall methodology and the agile methods. The Software Development Life Cycle (SDLC) is used to develop and deliver high-quality, reliable, cost-effective software products within a specified timeframe in the software industry. This is also called the software development process model (Shylez, 2017). Figure 2 demonstrates the SLDC and its parts. The SLDC is valid in both traditional and agile ways of work. In agile the cycle goes around every iteration, while in traditional ways for example waterfall methodology the steps are done one at a time.

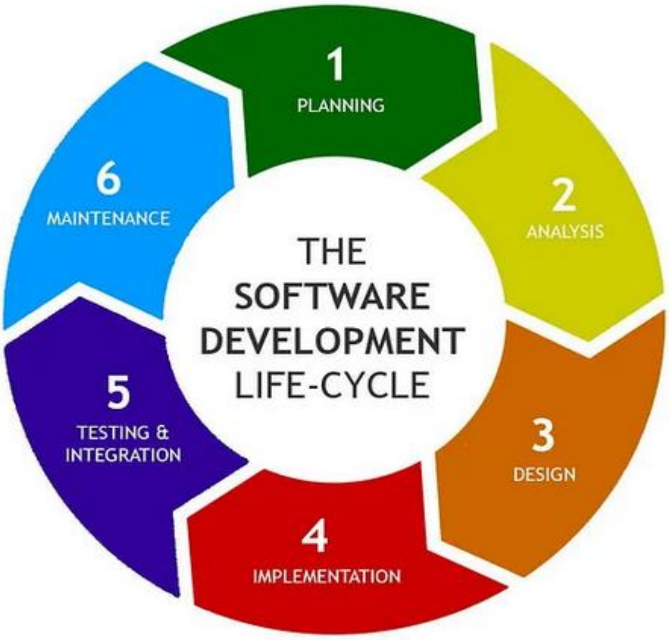


Figure 2. Diagram illustrating the Software Development Life Cycle (SDLC). Adapted from "Software Development Life Cycle (SDLC)," by A. Artjoms, 2024, Medium.

2.1.3 Agile Software Development: Lean, Scrum, and Scaled Agile

Agile encompasses a wide range of software development approaches and philosophies. It serves as a conceptual framework for software engineering, beginning with initial planning and progressing through iterative and incremental interactions until project completion (Al-Sagga, et al. 2020). The primary aim of agile is to minimize overhead in the software development process while facilitating adaptability to changes without compromising the process or necessitating excessive rework.

Although there are various methods within the agile philosophy, they are all united by the "Agile Software Development Manifesto" established by Beck et al. in 2001. This manifesto, introduced and endorsed by a group of 17 software engineering consultants, outlines a set of principles and values for software and system agility. The four values and twelve principles form the cornerstone for directing the software development process and defining the distinctive characteristics of any method associated with agility (Rodríguez, Mäntylä, Oivo, Lwakatare, Seppänen & Kuvaja, 2019). Fast changing environment does mean that the team must adjust its structure, relationships, and behaviour according to the situation. In each iteration, the team continues practices that were effective in the last iteration and modifies the practices that posed obstacles in their workflow (Al-Sagga, et al., 2020). These values and principles took the industry by storm and changed the way of work for software engineering. Due to agile's reduced costs and improved productivity, quality, and satisfaction, the agile paradigm has made a significant impact on the software development industry over the past 20 years and continues to do so (Ericson, Lyytinen & Siau, 2005). The main reason for the improvements in reduced costs, improved productivity and quality is due to the help of agile's flexible handling techniques and improved communication and coordination mechanisms (Rauf & Al-Ghafees, 2015).

Agility refers to the capability to adapt quickly to environment changes, new user requirements or delivery constraints. It is linked with nimbleness, suppleness, quickness, dexterity, liveliness, and alertness (Ericson, et al., 2005). The amount of agility a company possesses will determine its level of competitiveness (Al-Shagga et al., 2020). Each iteration in the agile framework includes planning, requirement analysis, design, coding, and testing (Choudhary and Rakesh, 2016). The main advantages of using agile ways of work are:

- Improved communication and coordination among team members.
- Quicker releases.
- Increased flexibility of design.
- A more reasonable and adaptable process (Choudhary and Rakesh, 2016).

One of the most dominant characteristics of agile ways of working is the ability to adapt to changes, as can be seen in Figure 3 (Pressman, 2005). As the surrounding environment is subject to changes, conventional software development processes incur nonlinear increasing costs as the project progresses in response to changes. In contrast, the incremental delivery aspect of agile processes reduces and flattens these costs, enabling changes at later stages without significant losses in cost and time (Pressman, 2005). This gives a great advantage in every spectrum of success measurements that are used in software development projects. Using an agile framework does not guarantee success but makes the failures more manageable in a way of minimizing costs and the loss of time.

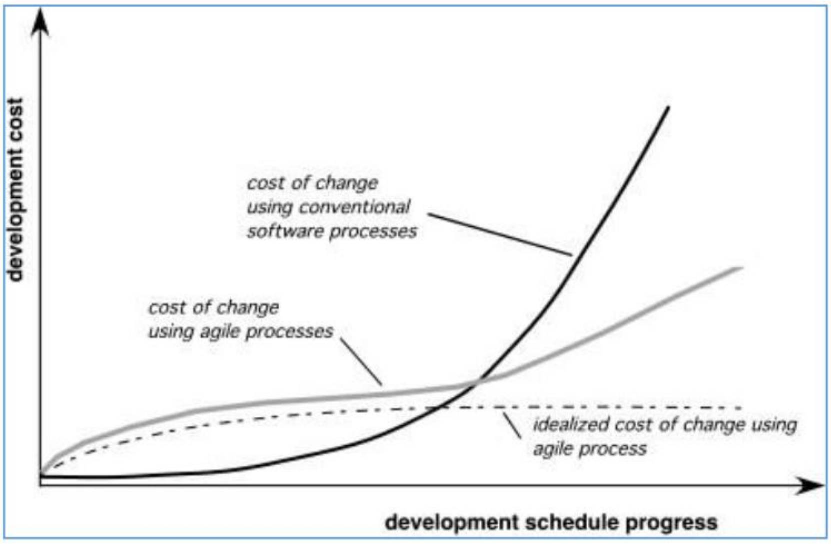


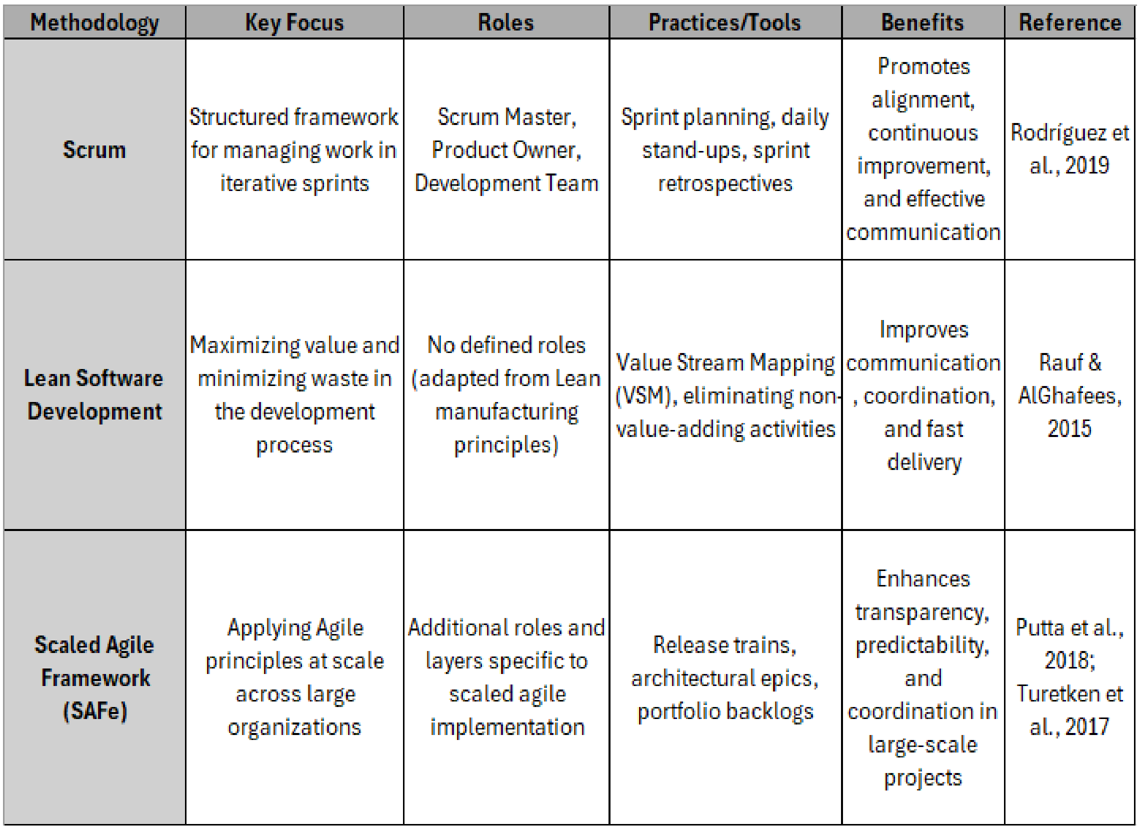
Figure 3 The development cost vs. change in the development process (Presmann, 2005).

Many agile methods support and take their base knowledge from the Agile Manifesto principles. Each agile method consists different mixture of practices which then construct a description of how the day-to-day work is done by the team member. Each method is different from the other usually from its set of terminology and practices (Elbanna and Sarker, 2016). The most common agile methods are Test-Driven Development (TDD), Feature Driven Development (FDD), Extreme Programming (XP), Scrum method, and Dynamic System Development Model (DSDM) to name of few (Al-Sagga, et al., 2020).

Over time each agile method has developed over the years and there are new methodologies have emerged taking the basis of their predecessors. Every methodology has its principles, iterations, life cycle, advantages and disadvantages. All agile software development methodologies construct the software through iterative and incremental processes.

Agile can be efficient also in large-scale projects. There are principles and guidelines for using agile methodologies in complex and large-scale software development such as Scaled Agile Framework (SAFe) (Putta, Paasivaara & Lassenius, 2018). SAFe improves company's transparency, alignment, productivity, predictability and time to market challenges. SAFe is a framework that tries to make the whole organization agile way (Brenner and Wunder, 2015). The SAFe approach tackles the issue of scalability by increasing the extent of certain agile practices and including novel ideas and methods (such as release trains, business and architectural epics, and portfolio backlogs) that mesh well with both basic and scaled agile techniques (Turetken, Stojanov & Trienekens, 2017). Table 2 demonstrates three popular agile methods Scrum, Lean Software Development and Scaled Agile Framework (SAFe).

Table 2. Three popular agile methods Scrum, Lean Software Development and Scaled Agile Framework (SAFe)



Methodology	Key Focus	Roles	Practices/Tools	Benefits	Reference
Scrum	Structured framework for managing work in iterative sprints	Scrum Master, Product Owner, Development Team	Sprint planning, daily stand-ups, sprint retrospectives	Promotes alignment, continuous improvement, and effective communication	Rodríguez et al., 2019
Lean Software Development	Maximizing value and minimizing waste in the development process	No defined roles (adapted from Lean manufacturing principles)	Value Stream Mapping (VSM), eliminating non-value-adding activities	Improves communication, coordination, and fast delivery	Rauf & AlGhafees, 2015
Scaled Agile Framework (SAFe)	Applying Agile principles at scale across large organizations	Additional roles and layers specific to scaled agile implementation	Release trains, architectural epics, portfolio backlogs	Enhances transparency, predictability, and coordination in large-scale projects	Putta et al., 2018; Turetken et al., 2017

There are many frameworks to choose from and it is not an easy task to choose the right one to use. Any company that wants to be effective has at least to study the opportunities of agile versus traditional way of software product development. It is important to consider that agile methods may not always be appropriate for every company, and traditional methods remain a viable alternative (Kurapati, Manyam & Petersen, 2012). The reason for that is for the company to succeed in transformation from a traditional software development method to an agile method needs transformation work and that could be a challenge to a company (Kurapati, et al., 2012). So, the effort is bigger than the gains. Many companies adopt a tailored approach to the agile framework to ensure it is the most efficient for their specific needs (Campanelli & Parreiras, 2015). The approaches for the agile method tailoring are based on method engineering and the main criteria for the method tailoring are internal environment and objectives (Campanelli & Parreiras, 2015).

Software development teams using agile methods work on the same tasks as in traditional software development methods but only in iterations (Figure 4). The first step that the company needs to do is to create agile teams from the existing development teams. When forming agile teams, it is important to carefully consider the skills and knowledge required for the project and its specific needs (Cho, 2010). The goal of team formation is to minimize the time needed to grasp the business logic, development tools, and programming languages. In addition to team members, agile teams benefit from having a leader who can envision the bigger picture and steer the team towards sound decisions, even though self-management is a key feature of agile, particularly in Scrum method (Cho, 2010).

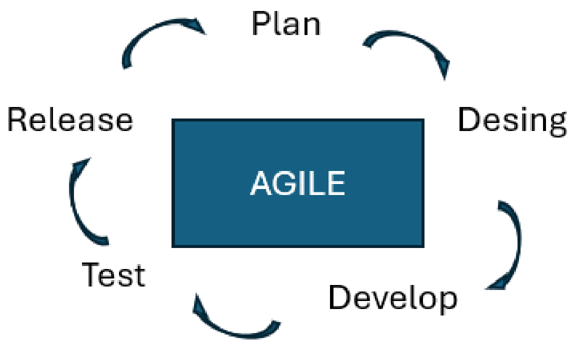
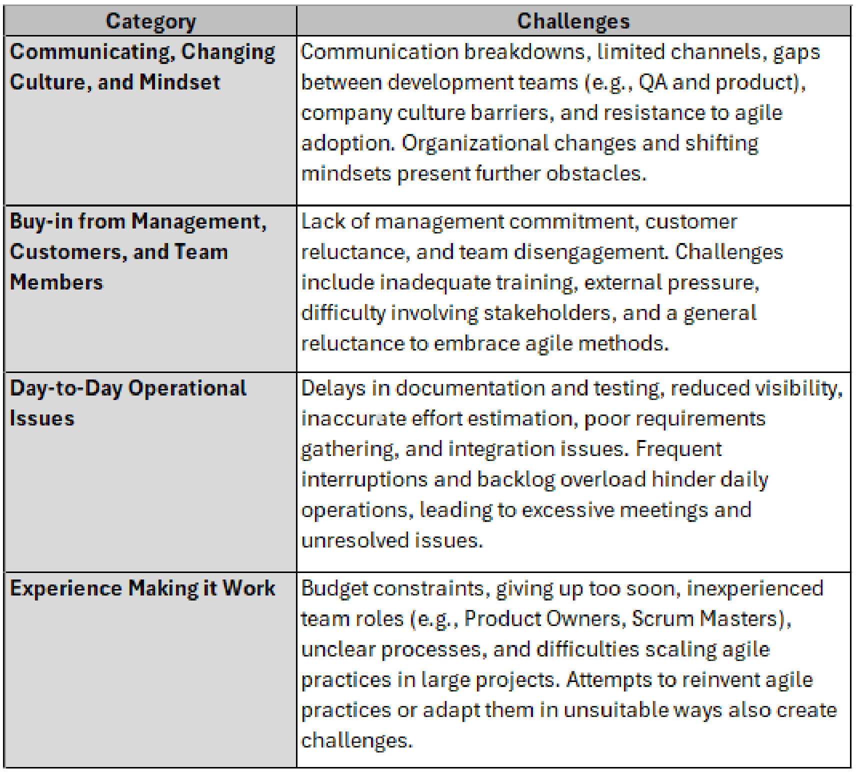


Figure 4. Agile Methodology Design (Matharu, et al., 2015)

According to Kettunen and Laanti (2008), software companies use agile methods where small teams work closely with customers to build high-quality software products with frequent iterations and feedback. agile methods are highly effective for small projects. It must be noted, that for complex software products and organizations, additional constraints must be met to realize the benefits of using agile methods. Nerur, Mahapatra and Mangalaraj (2005) reported the problems of migration challenges to agile methods. Despite these difficulties, companies that use agile for large-scale or distributed projects have shown benefits (Dingsøyr, Moe, Fægri & Seim, 2018; Eckstein, 2013). Agile is used in development projects by organizations such as Yahoo, Google, Microsoft, Siemens, CNBC, and AOL (Hajjdiab & Taleb, 2011).

Agile approaches offer flexibility, adaptability, and enhanced teamwork, but implementing them can be extremely difficult at the management, operational, and organizational levels. Effective implementation requires a profound culture transformation, which many firms find difficult to accomplish, in addition to a change in development procedures. Miller (2013) states that poor communication, opposition to change, and a lack of support from important stakeholders are typical obstacles. Adoption of agile might operationally result in daily difficulties such as backlog management, cross-functional team coordination, and task visibility. Successful integration might also be hampered by imprecise procedures and a lack of seasoned agile practitioners. more details are presented in Table 3.

Table 3 Agile method challenges according by Miller (2013)



Category	Challenges
Communicating, Changing Culture, and Mindset	Communication breakdowns, limited channels, gaps between development teams (e.g., QA and product), company culture barriers, and resistance to agile adoption. Organizational changes and shifting mindsets present further obstacles.
Buy-in from Management, Customers, and Team Members	Lack of management commitment, customer reluctance, and team disengagement. Challenges include inadequate training, external pressure, difficulty involving stakeholders, and a general reluctance to embrace agile methods.
Day-to-Day Operational Issues	Delays in documentation and testing, reduced visibility, inaccurate effort estimation, poor requirements gathering, and integration issues. Frequent interruptions and backlog overload hinder daily operations, leading to excessive meetings and unresolved issues.
Experience Making it Work	Budget constraints, giving up too soon, inexperienced team roles (e.g., Product Owners, Scrum Masters), unclear processes, and difficulties scaling agile practices in large projects. Attempts to reinvent agile practices or adapt them in unsuitable ways also create challenges.

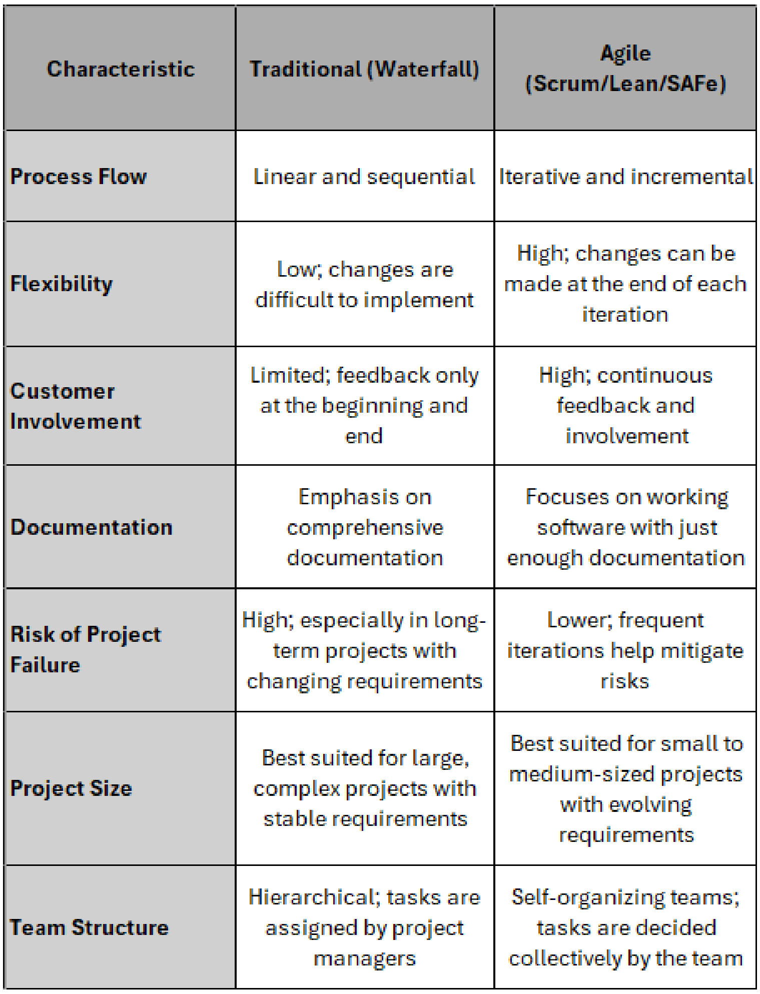
2.1.4 Comparison of Traditional and Agile Approaches

Agile and traditional approaches differ primarily in how flexible and adaptable they are to changes. Conventional methods, like the Waterfall technique, have a rigid, phase-by-phase schedule that restricts flexibility after development begins. Agile approaches, on the other hand, take advantage of iterative cycles, which enable regular revaluation and adaptation in response to stakeholder input. Agile and conventional software development methodologies differ from one another, as Table 4 illustrates.

Agile methods are highly adaptive due to continuous collaboration between developers and customers and scoping the ways of work in every iteration, making them effective especially for small to medium-sized projects. Agile focuses on short-term planning and specific functionalities, making it more adjustable and easier to modify compared to the extensive long-term planning which is required in traditional methods. Agile approaches put working software ahead of documentation, while traditional techniques place more emphasis on thorough documentation, which causes engineers to spend a lot of time on paperwork. Sure, there is documentation, but just that extensive in the agile ways of work. Due to their structured, sequential character, traditional techniques often call for larger teams and greater budgets, whereas agile approaches typically include smaller teams and essentially a lower budget from the start of the project. Every approach has pros and cons of its own and a position in the field of software development.

Even if traditional approaches are dependable and disciplined, they can become inflexible and unyielding to modifications, which makes them less appropriate for fast-paced work settings. Agile approaches are very effective for projects with quickly changing requirements since they are built to handle uncertainty and change (Campanelli & Parreiras, 2015).

Table 4. Comparison between traditional and agile methods



Characteristic	Traditional (Waterfall)	Agile (Scrum/Lean/SAFe)
Process Flow	Linear and sequential	Iterative and incremental
Flexibility	Low; changes are difficult to implement	High; changes can be made at the end of each iteration
Customer Involvement	Limited; feedback only at the beginning and end	High; continuous feedback and involvement
Documentation	Emphasis on comprehensive documentation	Focuses on working software with just enough documentation
Risk of Project Failure	High; especially in long-term projects with changing requirements	Lower; frequent iterations help mitigate risks
Project Size	Best suited for large, complex projects with stable requirements	Best suited for small to medium-sized projects with evolving requirements
Team Structure	Hierarchical; tasks are assigned by project managers	Self-organizing teams; tasks are decided collectively by the team

Each methodology — traditional and agile — has advantages and disadvantages of its own. The project's needs, organizational structure, and other factors all have a role in which strategy is selected. While the Waterfall approach may work well for projects with defined and consistent requirements, agile methodologies are more versatile and give teams the capacity to respond fast to changing client needs (Kurapati, Manyam & Petersen, 2012).

2.2 Value Stream Mapping in Software Development

In this chapter, Value Stream Mapping (VSM) will be examined, including its application in software development and the benefits it can bring to improving software development processes.

2.2.1 Introduction to Value Stream Mapping

VSM is a method or tool that uses practical ways to help people visualize and comprehend the information and material flow that a product goes through as it moves along the value stream. The value stream refers to the comprehensive array of activities—both value-adding and non-value-adding—essential for delivering a product or service from its initial conception to the final delivery to the client (Nowak, Pfaff & Karbach, 2017). Customer loop, production control, supplier loop, manufacturing loop, information flow, lead time data bar with critical path are the components of VSM that provide us with a comprehensive picture of the entire supply chain, from the needs of the customer to the supplier's delivery (Chen, Lixia & Bo Meng, 2010). VSM is used to explore and find wastes, inefficiencies, and non-value-added steps using a single, definable process out of a complete product development process (Tyagi, Choudhary, Cai & Yang, 2015). It can be used in a variety of processes, industries, and settings. Another description of VSM in software development is that it is a lean methodology used to visualize the current process flow, called current state map or CSM. VSM is also supposed to pinpoint activities in the process that either add value or do not and develop a collaborative plan for enhancing the process. This plan outlines the desired future state and is called a future state map (FSM) (Khurum, et al., 2014; Rother and Shook, 1999). VSM examines the movement of information as well as materials through the process. Akin to the examination of material flow in software development, artefact flow pertains to the flow of "work items," like requirements, use cases, or user stories, that are involved in the process. Software development requires recording verbal, written, formal, and informal communication in production processes where "information flow" mostly consists of scheduling data. Additionally, it is critical to recognize and evaluate the data, expertise, and abilities required to carry out the value-adding tasks during the software development process (Ali, Petersen & Schneider, 2016).

The primary method for transitioning from agile to true continuous software development is flow, which is a fundamental tenet of contemporary lean thinking (Fitzgerald & Stol, 2015). Throughout the whole development process, flow refers to the management of an ongoing stream of activities that create value (Petersen & Wohlin, 2011). Because it places more of an emphasis on controlling backlog than on project phases and milestones, this method varies from traditional project management (Power & Conboy, 2015). The flow concept, for instance, is helpful when thinking about continuous software engineering, which stresses movement continuously rather than concentrating on agile techniques specifically (Fitzgerald & Stol, 2015). Flow promotes teamwork, knowledge sharing, and the evaluation of costs, benefits, and technical metrics—all essential elements of DevOps (Bang et al., 2013).

When we refer to value-added (VA) activities, we are discussing activities that yield a higher return on investment than the initial cost. This return on investment can manifest in increased revenue, enhanced user satisfaction, or cost savings, such as reduced customer support expenses (Redish, 1995). Lean manufacturing is based on the fundamental premise of minimizing non-value-added processes and process lead times. Activities referred to as non-value-added are those that use resources but result in nothing at the end of the process (Maroofi & Dehghan, 2012). Lean manufacturing approaches' primary goals are to increase overall productivity, individual costs, equipment performance, and product quality (Joseph & Ronald, 2012).

Lead Time (LT) and Process Time (PT) are the main metrics of flow that VSM measures. When talking about LT it means the time that the actual task or work phase takes to do. This means only the work, not bureaucracy, top management approval or other bottlenecks. PT means the whole time that it takes from the start of the task to the end of the task in total, including all the possible bottlenecks added together.

VSM also measures the percentage of complete and its accuracy %C&A in short. It measures the quality for example developed code. If %C&A for a code developed task is 90% it means that the code developed was 90% working and the missing 10% needs fixing or must done again.

In the context of value stream analysis, a bottleneck refers to a specific stage or phase within the value stream where the flow of processes is significantly impeded, resulting in a marked slowdown or complete halt. One of the best ways of identifying bottlenecks in the value stream is value stream mapping (Staron & Meding, 2011). Bottlenecks can be identified by looking at PT and LT in the VSM.

Value Stream Maps are structured in workshops or exercise environments where the stakeholders are participating and committed to the cause. Exercises or workshops are straightforward and follow usually the same structure (Figure 5)

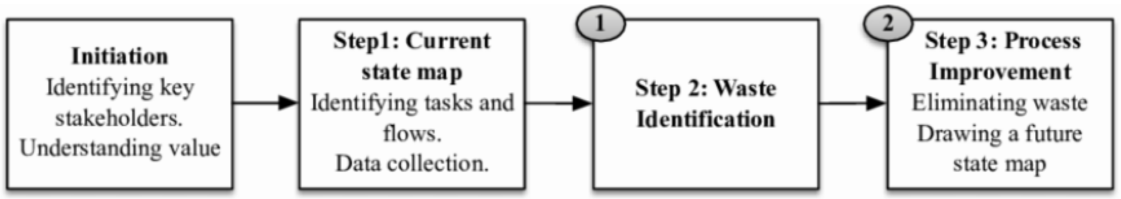

Initiation
Identifying key stakeholders.
Understanding value
Step1: Current state map
Identifying tasks and flows.
Data collection.
1
Step 2: Waste Identification
2
Step 3: Process Improvement
Eliminating waste
Drawing a future state map

Figure 5 VSM workshop structure. bin Ali, (2015)

First, all stakeholders are informed about the purpose of the exercise, the benefits of using Value Stream Mapping (VSM), and how it can positively impact their work. This ensures that everyone understands the value of the activity and is motivated to participate. Next, the stakeholders collaboratively build the CSM by identifying each phase of the process, determining who is involved at each stage, and recording key metrics such as PT, LT, and the Percentage of %C&A. Once the CSM is complete, the team uses the collected data to identify bottlenecks and inefficiencies in the workflow. After pinpointing these issues, the team engages in a brainstorming session to generate improvement ideas and solutions. These ideas are then used to create the FSM, which outlines the optimized version of the process.

2.2.2 Benefits of Value Stream Mapping

VSM is one of the most popular tools when it comes to making manufacturing almost anything more efficient. It is most used in lean manufacturing cases but can adapt to numerous lean processes like software development. VSM has numerous intangible benefits, it is said that the VSM framework will help the development teams to reduce the product development lead time by 50% (Tyagi, et al. 2015).

VSM is a useful tool to identify waste in the process and when it comes to VSM, traditionally everything that does not add value is waste (Khurum, et al. 2014). On the other hand, VSM has been criticised for not showing all kinds of waste as well as for its inability to represent various production routes (Dinis-Carvalho, et al., 2015).

Potential bottlenecks and inefficiencies can be found by mapping the current state of the process flow using VSM. Through process visualisation, VSM offers an integrated perspective of the development lifecycle, enabling teams to identify regions prone to delays or non-value-added tasks. By removing these obstacles, the process becomes more efficient since it runs more smoothly and has better flow overall (Thummala, 2004).

By fostering a shared and consistent understanding of the process and opportunities for improvement, VSM helps team members communicate and collaborate more effectively. Team members can grasp the workflow from various angles since they have shared insight throughout the entire process. The team can contribute to process improvement more successfully with VSM (Nasir, Minhas & Sweden, 2018).

VSM contributes to developing a roadmap for future improvements (Future State Map, or FSM) and establishes a baseline for the existing state of processes (Current State Map, or CSM). Teams can focus on continuous process optimization by removing non-value-added tasks and piece by piece improving the workflow thanks to this iterative approach. Thus, VSM encourages a culture of adaptation and ongoing development (Ali, Petersen & De França, 2015).

Through an emphasis on value creation from the customer's point of view, VSM makes sure that the team's work is in line with the organization's goals. According to Darwish, Haque and Shehab (2010), this alignment aids in allocating resources to the most valuable tasks and guarantees that process enhancements are both effective and relevant to the demands of the company and its clients.

With VSM, managers and teams could have a complete picture of the process and use the data and insights from the mapping exercise/workshop to make better decisions to enhance their processes. The visualization of important process data assists stakeholders in identifying possible improvement areas and prioritizing actions that will have the biggest effects on productivity (Jeong and Yoon, 2016).

As a lean tool, VSM fosters a culture of waste reduction and continuous improvement by establishing an atmosphere that continuously looks to improve the efficacy of processes. This kind of thinking is essential in dynamic, fast-paced industries like software development, where it is critical to continuously provide value and adjust to changes (Tankhiwale & Saraf, 2020). Implementing Value Stream Mapping (VSM) in software development has many benefits but requires considerable time and resources.

2.3 Change Management in Software Development

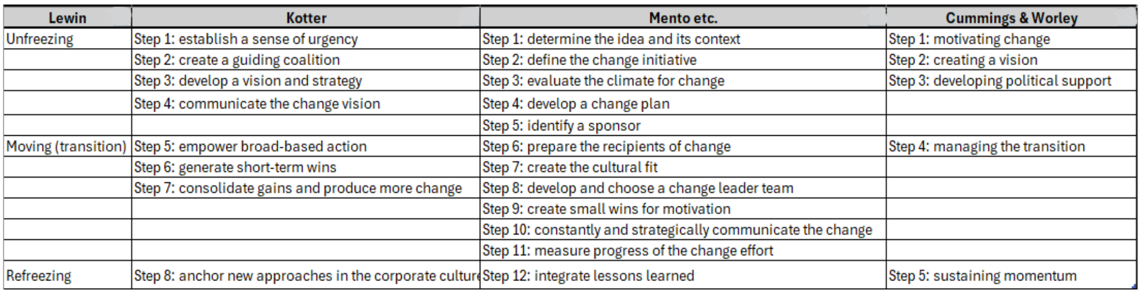
In this chapter, the subject of change management will be explored. Some of the most popular change management theories will be broadly reviewed, followed by an examination of their application within a software development context as a tool to implement changes that are brainstormed and identified in the VSM.

2.3.1 Overview of Change Management

Although there are various ways to define change, the two most important contributors to the emerging field of change management, John Kotter and William Bridges, provide the following definition. For handling the situational and emotional aspects of change, Kotter's eight-step approach (developing urgency, building a guiding team, creating a vision, communicating for buy-in, enabling action, achieving short-term victories, not letting up, and making it stick) provides strategies (Kotter, 2007). Bridges tackles the topic of change at a more specialized, individual level, suggesting that individuals involved in organizational change must modify their identities. According to Bridges (2009), transitions occur in three stages: beginnings, neutral zones, and endings. Organizations always aim to modify and adjust their operations to changing conditions in a business environment that is becoming more complicated and dynamic (Al-Haddad & Kotnour, 2015; Burnes, 2011). Organizations are required to allocate significant financial resources towards implementing diverse adaptations to accommodate the evolving business landscape.

As said, there are many different models when it comes to change management. Every model has its key components and phases. Table 5 presents four different models and their perspectives on how to make a successful change in the organization (Lewin, 1948; Kotter, 2007; Mento, Jones & Dirdorfer, 2002; Cummings & Worley, 2016).

Table 5. Change management steps according to Lewin, (1948), Kotter, (2007) Mento et al., (2002) and Cummings and Worley, (2016)



Lewin	Kotter	Mento etc.	Cummings & Worley
Unfreezing	Step 1: establish a sense of urgency	Step 1: determine the idea and its context	Step 1: motivating change
	Step 2: create a guiding coalition	Step 2: define the change initiative	Step 2: creating a vision
	Step 3: develop a vision and strategy	Step 3: evaluate the climate for change	Step 3: developing political support
	Step 4: communicate the change vision	Step 4: develop a change plan	
		Step 5: identify a sponsor	
Moving (transition)	Step 5: empower broad-based action	Step 6: prepare the recipients of change	Step 4: managing the transition
	Step 6: generate short-term wins	Step 7: create the cultural fit	
	Step 7: consolidate gains and produce more change	Step 8: develop and choose a change leader team	
		Step 9: create small wins for motivation	
		Step 10: constantly and strategically communicate the change	
		Step 11: measure progress of the change effort	
Refreezing	Step 8: anchor new approaches in the corporate cultur	Step 12: integrate lessons learned	Step 5: sustaining momentum

Several studies have highlighted that most organizational change initiatives fail, with an estimated failure rate of 60–70% (Barnes, 2011; Ashkenas, 2020; Jones, Firth & Hannibal, 2018). A high failure rate prompts ongoing worry and curiosity about the variables that can lower failure and boost organizational change success (Rafferty, Jimmieson, Armenakis, 2013). Numerous change management models have been created by researchers and consulting firms to enhance the success rate of change projects. Despite the plethora of available models, it remains crucial to fully identify these aspects and address the knowledge gaps related to successfully implementing organizational change management (Barnes, 2011). Because of this, using one or a small number of models is insufficient to cover all possible change scenarios (Burnes and Jackson, 2011). Additionally, certain aspects may be overlooked or left out, which increases the risk of failure if the model is not fit for the changing context (Erdogan, Anumba, Bouchlaghem & Nielsen, 2005). Consequently, including current models could result in a more thorough understanding of how to guarantee organizational transformation is effective as well as aid in the creation of a change management strategy (Errida & Lofti, 2021).

Success is a hard term to define, especially in change management. It always depends on the perspective if the change is successful or not. There are common things in many models that provide the best basis for a successful change in the organization. According to Errida and Lofti (2021), successful change requires the following active focus points:

- Clear and shared vision and strategy of change
- Change readiness and capacity for change
- Activities for managing change
- Managing resistance to change
- Effective communication
- Motivation of employees

- Engaging stakeholders
- Leadership and sponsorship
- Reinforcement and sustainability of change
- Monitoring and measurement.

2.3.2 Change Management in Software Development

There are many common things with regular change management and change management in software development. There are still some twists in software development and the industry surrounding it that must be noticed when practising change management in a software development environment.

There are different methods for different changes in software development. It can be challenging to identify comprehensive system requirements that accurately reflect current circumstances and adapt to changing needs, change management is a complex aspect of the requirements engineering process of software development (Jayatilleke & Lai, 2018). System requirements are always changing due to several factors, including changing consumer demands, shifting market dynamics, competition from around the world, and governmental laws. For these kinds of changes, there are Requirement Change Management (RCM). Successful RCM requires coordination and communication across stakeholders, as it is a process driven by collaboration (Niazi, El-Attar, Usman & Ikram, 2012; Kumar & Kumar, 2011). Insufficient RCM can result in high software costs, erratic needs, postponed timelines, and never-ending testing, all of which can damage the project's success and the company (Jayatilleke & Lai, 2018).

Global software development (GSD) is a software development paradigm wherein development activities are carried out by experts situated in many global locations to produce successful products for a company (Sinha, Senguta & Chandra, 2006). The global software industries are becoming more interested in using GSD to generate financial gains (Niazi, et al., 2012). Due to the much lower development costs, outsourcing development to supplier firms in low-cost nations has gained popularity. But for those who work on software projects produced at the same physical location, GSD presents several challenges that do not arise in collocated projects (Niazi, et al., 2012). Because development teams are dispersed across multiple geographic places, communication and coordination are negatively impacted by differences in ethnicity and time zone, which leaves development teams deficient in skills, abilities, and trust (Khan and Azeem, 2014).

RDM and GSD are more generally linked in software development but do not have much common ground or answers in the context of this study. More in the context of this study three of the most common problems are faced during change in the realm of software development.

Change resistance is one of the most common troublemakers in the change management scene. People are generally inclined to resist change and may at times engage in extreme measures to avoid it. Yet, research also shows that most people tend to oppose these kinds of changes because they are typically accompanied by more pressure, urgency, and danger than regular organizational operations (Ford, Ford & D'Amelio, 2008; Kotter, 1995). According to Ford (1996), "Even in circumstances that favour creative action, people will likely choose familiar behavioural options that are relatively more attractive based on their past success, relative ease, and certainty." To overcome change resistance depends highly on the situation and people working on the problem but most of the time the sense-making approach (Weick & Weick, 1995). The sense-making approach is predicated on the idea that people participate in sense-making processes to interpret the meaning of issues and events that surround change-oriented settings, such as creative performance, which are frequently ambiguous and fuzzy (Weick & Weick, 1995). The sense-making method also emphasizes that resistance issues are not planned; rather, they are created within a specific social environment and rely on signs and knowledge people gather about the degree of uncertainty and the risks they confront. For instance, these cues affect people's feelings of uneasiness and worry when things change (Jones, 2001).

Numerous studies show that employees' attitudes and behaviours are significantly influenced by the type of leadership they receive, particularly from empowering leaders (Zhang & Bartol, 2010; Srivastava, Batrol & Locke, 2006). Hon, Bloom and Grant (2014) suggest that overcoming employees' resistance to change may also be significantly influenced by the behaviours of leaders. Encouraging leaders to develop relationships with their subordinates that are built on trust and power sharing. They also communicate a compelling work vision to them, provide coaching, demonstrate concern, and help their followers become more self-reliant Zhang & Bartol, 2010). By encouraging decision-making, exhibiting their trust in subordinates, and giving followers the ability to be as flexible as circumstances demand, they foster a strong sense of power-sharing among their followers (Arnold, Arad, Rhoades & Drasgow, 2000). Also, based on current research, workers under the direction of empowered leaders are more likely to be inspired to question established practices and take calculated risks that come with being innovative (Zhang & Bartol, 2010).

2.4 Summary and Transition to the Empirical Study

The research's theoretical foundations emphasize how crucial software development frameworks and methodologies like agile, Scrum, and VSM—are to streamline development processes. Traditional linear approaches to software development, such as the Waterfall technique, have given way to more dynamic and iterative models that prioritise continuous delivery and adaptability to change. Through iterative cycles and ongoing stakeholder involvement, the adoption of agile and lean techniques has transformed software engineering by boosting efficiency, quality, and team cooperation.

Nevertheless, software development continues to encounter challenges, despite the pervasive adoption of these methodologies. Long development cycles, scope fragmentation, poor communication, and ineffective resource management are typical problems. These problems lead to lower-quality products, delays, and cost overruns. Lean management concepts, particularly VSM, provide an organized method for locating and getting rid of non-value-adding tasks to address these issues.

VSM started in the manufacturing industry and has since been modified for use in software development to optimize and visualize workflow processes. It assists businesses in identifying inefficiencies, mapping out current procedures (Current State Map), and developing improvements and opportunities with the team brainstorming session. It is not easy to adopt VSM in software teams because successful implementation necessitates excellent change management and a detailed understanding of intangible flows, such as information exchange. With VSM there is a possibility to find bottlenecks (where the process flow stops or slows down drastically) and eliminate them with the suggested improvement ideas. In the literature, VSM has proved to be an efficient tool to identify waste and find bottlenecks in the processes where it has been implemented.

The incorporation of VSM inside software development processes is made easier by change management theories like Bridges' Transition Model and Kotter's 8-step model. By addressing the human elements of change, these models help teams overcome opposition and develop a common vision. Change management, when used in tandem with VSM, guarantees that suggested enhancements are successfully adopted and maintained inside a company. VSM perk is that all the improvement ideas come from inside the team and there is no top management who dictates the changes. This motivates the team members to make the change when they are the ones who started them.

This study transitions from theory to practice by exploring how VSM can be utilized within software development teams to identify bottlenecks and inefficiencies, propose actionable improvements, and reveal if VSM is a tool that brings value to the company. The empirical part of this research presents a case study with three software development teams in a global IT corporation, where VSM workshops were conducted to assess the current state of development processes identify areas for enhancement and seek improvement from the team to their processes.

The empirical section will delve into the practical application of VSM workshops, detailing the findings from each team’s experience and workshop. It will go over the difficulties encountered during implementation, the suggested changes, and how VSM assisted in identifying pain points. This practical thesis tries to bridge the gap between theoretical principles and real-world applicability, illustrating how VSM may be used to achieve more efficient and successful software development processes.

3 METHODOLOGY

This chapter introduces the objectives and research questions of this study and the selected methodology to answer them. After the targets, research questions, and chosen methodology of this study are presented, they are followed by a discussion of the environment and the case company. The description of the data collection and analysis process comes next. Finally, a few ethical restrictions related to the study are covered and presented.

3.1 Objectives and Research Questions

Prior the research has identified what Value Stream Mapping is, how it is implemented and what are the possible positives to implementing it in one’s software development process. Everybody in the industry wants to be more efficient in their way of working and VSM has proved its efficiency in the industrial line and many agile frameworks as a way to identify waste and make the value flow more efficiently in the value stream. But the most interesting question is how it works in practical software development teams and what it provides that it can be used the best way possible to make the value stream more efficient. Also, are the results of VSM proposals easy to put into action, or do they require different strategies for managing change. This study aims to figure out how useful VSM is to software development teams and what kind of improvements it can provide and how they should be implemented.

This thesis strives to answer the following research questions:

1. What are the main pain points of a given company’s software development and how they can be identified by Value Stream Mapping?
2. What value does VSM bring to the company's teams and how can this value be realized in the current software development environment

The focus of this study is to study how useful VSM is as a tool to identify waste and how it brings value to a software development process and what kind of improvement ideas result from the VSM workshops. Every team is one of a kind, so it is important to study VSM with different teams and see if it can produce value for each and are differences between the teams. VSM could be a useful tool for a given company to identify waste and enhance its software development processes. VSM also requires resources that could be costly if VSM does not produce added value.

3.2 Selected Methodology

Research methods in the field of information systems (IS) can be classified into two main streams: qualitative and quantitative. Quantitative research aims to generalize or validate specific theories and typically works with numerical data. Conversely, qualitative research focuses on understanding human perceptions and experiences within a specific social or cultural context (Myers and Avison, 2002; Stake, 2010). Human-technology interaction and organizational difficulties have replaced technology-focused themes in the IS area, making qualitative research approaches increasingly important (Myers 1997). The sample sizes used in qualitative methods are smaller than those in quantitative approaches. Different data collection techniques are also used. In qualitative research, the most common techniques for gathering data are observations, questionnaires, document analysis, and interviews (Stake, 2010; Myers and Avison, 2002).

The need for this study was initiated from practical needs and curiosity from practitioners who were eager to develop the way the company's teams were working and keep track of what they were doing and how. Recent SAFe framework implementation introduced VSM as a tool to identify waste and seek for improvements within the teams their self. A few Value Stream Maps were done in the recent year but not between one's team, only at the management level. So, it is interesting to go to the team level and see if VSM is a tool for a practical tool to identify waste in software development and what kind of added value it can bring to the processes when implemented. Considering the exploratory nature of this research and the objectives of this study there was a need to collect practical data from the teams in the concept of workshops. A qualitative case study approach was chosen based on the exploratory nature of the research as well as the business's practical requirements. The purpose of this approach is to gain an in-depth understanding of the software development processes and the potential benefits of VSM. When examining a phenomenon in the context of real-life experiences and when current literature is absent on the subject, qualitative case studies are especially appropriate (Yin, 2009). This aligns with the objectives of this study, which seeks to explore the practical application of VSM at the team level in a software development context.

Since the workshops in this study involved close collaboration with participants to analyse and improve their processes, they are consistent with the concepts of action research. Action research facilitates ongoing communication between participants and the researcher, which promotes iterative process improvement and learning (Avison, Baskerville & Myers, 1999).

The research's strategic decision to use workshops as its main data-gathering strategy was in line with standard practices for implementing VSM. Usually presented as an interactive exercise or in a workshop format, VSM enables participants to interact directly with the methodology. As a result, the workshops provide an organic setting for gathering the required information.

It was concluded that interviews would not substantially increase the study's usefulness while being taken into consideration as an extra data-gathering approach to obtain further insights into the VSM process. Rather than using formal interviews, feedback forms successfully performed the role of collecting participant reflections and ideas.

3.3 Case Description

The empirical data for this study was collected within the company by conducting workshops for three different teams with a few extra questions added in each workshop. Although it had several goals, the main goals were to determine if VSM is a practical tool that can identify waste in a software development environment and what kind of improvements it can provide to make the company’s teams more efficient. Also, within the workshops and after there was a feedback query that gave a free word to workshop participants to impress their opinion about VSM and is something that should be done occasionally.

The case company under study is a substantial global IT corporation with an extensive history in the industry. Over the years, the company has changed from being a telecommunications company to a software company, and it has maintained an open approach to using modern, agile processes in various areas of the business. The case company employs people globally, with offices and most teams being distributed across countries.

Representatives from the company were contacted in the spring 2024 of to introduce the idea of researching how the company could leverage VSM and improve its team’s efficiency. At first, there were bi-weekly meetings with three managers and a SAFe consultant to discuss the topic and modify the research subject in a way that serves both, the company and the researcher. When the scope was set, and agreement was taken place there was a decision to arrange three VSM workshops of three different teams from three different product lines to see the VSM in all areas of business. The industry and the area for all three teams are the same, but the products and the maturity of the teams differ from each other. The motivation for the company to participate in this research was to test VSM as a tool to be used in the future to identify waste and make the team processes more efficient.

This means that the workshop's main idea and structure remained the same for the whole research, but the content and data would vary between the teams. In total 37 team members participated in the workshops the biggest workshop by participants was 18 people at a time and the smallest group were 9 people. The main point of the workshops was that the team were physically available in the conference room for the whole workshop, and this was done successfully. Only 5 people declined the workshop invitation due to not being able to be physically in the workshop. The teams chosen to attend the workshops agreed with the company representative and the researcher. For both, it was important to gather data from a wide range of teams as possible and it was successful.

Workshops were arranged with each team’s management (Scrum Master and Product Owner). Every workshop had a time slot varying from 2,5 hours to 4 hours and the workshop itself was facilitated by the researcher in the summer of 2024. Every team leader was eager to help and interested in the idea of VSM workshops. They wanted to participate in the research and had a positive attitude towards the subject and the researcher. The fact that neither the company nor the researcher offered any incentives suggests that the motivation for participating in this study was most likely a sincere interest in the subject.

3.4 Data Collection

Workshops were the most usable and the best way to data collection for the VSM. VSM is almost always held in a workshop scenario when done in teams. It is the most convenient way to get the team together and keep the team members focused for the whole time needed. The workshop provided the VSM-Maps as one form of data including each step of the process, who is responsible for each step and the times and accuracy of each step with the proposed improvement ideas. Data was also collected for each workshop end using a few questions regarding the improvement ideas and how likely they could be implemented in the eyes of each team member. Furthermore, a feedback survey administered via Google Forms inquired whether the participants found the VSM workshop to be beneficial and whether they would be interested in attending a follow-up workshop to evaluate their progress. Also, in the forms were questions about the workshop facilitator's performance and a free word for the organizer. The researcher also took notes by hand in each workshop. In the actual workshop, everything was done by hand for practical reasons, so Value Stream Maps were done by using Post-it notes and done on a wall or a window.

Every workshop had the same setup and structure for each workshop. Only spent time for the workshops differed. At first, there was a short motivation for the participants and some explanation for the theory of VSM. Workshop times set some limitations for the results of the workshops, so there was no future state map, but the teams built current state maps and brainstormed improvement ideas during the workshops.

The optimal time for each workshop was 4 hours but the workshops lasted from 3 to 4 hours depending on each team's schedule and priorities. Everything got done was planned and overall workshops were efficiently organized, and the data were successfully collected.

3.5 Data Analysis

All workshops produced the same data. The only difference was workshop 1 which had 16 people to participate, so there were two Value Stream Maps produced. Because everything produced in the workshops was done by hand there was a task to virtualize everything, and it was done by the researcher using an app called Klaxoon where the Value Stream Maps were virtualized and the data from each session end was collected. Also, every Value Stream Map was documented in photo form to be careful.

Every improvement idea was recorded and talked through the teams' management and top management. During the workshops, team members were able to vote for the best improvement ideas and every team management chose the top three improvements and was ready to start implementing voted improvements to their processes.

Feedback forms were sent to each participant at the end of every workshop and the form gave an understanding of the overall attitude towards VSM workshops and how useful they were. Forms calculated the data automatically from each workshop.

3.6 Research Ethics

The way that the teams were selected might have resulted in a few limitations that could influence this study. Top management recommended these three teams to participate in the workshops and the team's management chose to participate. So, rather than taking part in the study because they were genuinely interested in the subject, some participants may have felt pressured or aggressively persuaded. Probably, some individuals answered differently because they felt pressured to attend the workshops, which may have had an impact on their motivation. This seems highly unlikely since the workshops were team exercises, and they chose to participate to make their teamwork more efficient.

The fact that the subject may occasionally be delicate is another problem. Workshops were held face-to-face, and the team got through the whole process. Sometimes there is a possibility to hurt each other feelings, so it is possible that every participant did not want to say their opinions at the cost of hurting someone's feelings. Also, the team management was in the workshop, so that could influence free speech.

Data is analysed manually, and no computer-aided software is not used. It could have improved the accuracy and reliability of the data. According to Friese (2019), using that form of qualitative research software throughout the analysis phase helps the researcher uncover insights that he might not have otherwise discovered and enhances the entire research process.

4 RESULTS

The results of the conducted qualitative study are introduced in this chapter. First, the background of the team is presented at the workshop, and then the current situation and the current state map are discussed. After that, the pain points and bottlenecks are pinpointed, followed by a review of the improvement ideas and opportunities. The results of each workshop are presented separately, and in the end, the workshops are studied as a whole, and a comparative analysis of the findings is conducted.

4.1 Workshop 1

4.1.1 Team Background

Team 1 has been built the most recently and has been working together for about two years. They started their process from scratch because the project is kind of a new leap for the company. Some of the team members were taken from the other projects inside the company and some work as an external workforce. They have done only one real deployment to the customer, so the process is fresh, and the process can move and adjust in a fast phase if necessary. The team age average is the lowest and it includes 18 team members. Team members work in two different sites. In the actual workshop, there were 17 participants, so the team was split in two and they made 2 Value Stream Maps. Both maps had approximately the same data, so they were processed as one. All improvement proposals and process enhancement opportunities were considered by both VSMs.

4.1.2 Current State VSM

The entire software development process, from the first customer request to the last phase of customer feedback and analysis, is described in the Value Stream Map from Workshop. The process involves various stages including Request Analysis, Prioritization, Planning, Sprint Planning, Development and

Documentation, Testing, Releasing to Production, and finally, Customer Feedback.

There are numerous team members and jobs involved in each step. For example, in the initial phases of request analysis and prioritizing, Product Owners (POs), Business Analysts (BAs), and Sales staff are crucial. The QA/test team and the development teams oversee the later phases, which include testing and development. Notably, a few cross-functional positions are essential in the design and sprint planning phases, such as the Scrum Master and System Architect. Each stage is measured in terms of PT and LT, which enables a detailed assessment of where inefficiencies and delays occur.

The overall PT for all phases is 48.3 days, whereas the LT is 111.1 days. This indicates a large lag between the start and finish of the tasks. This extended LT points to the existence of inefficiencies and bottlenecks, especially during phase handoffs and in phases when reliance on cross-functional teams is essential.

The workflow itself spans several distinct phases:

- Request Analysis and Prioritization, conducted by business-facing roles such as POs, BAs, and Sales.
- Development and Documentation, which heavily involves the development team, along with supporting input from system architects and designers.
- Testing and Deployment, where test and security teams ensure the final product meets quality standards before release.

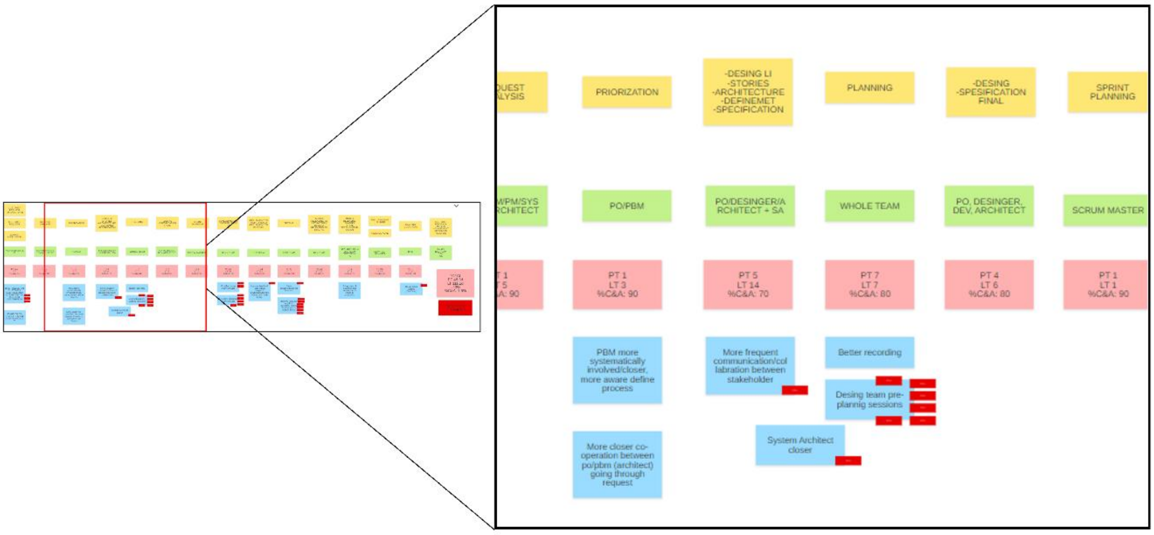


Figure 6 CSM from the Team 1

The thorough process mapping and important indicators like PT and LT offer insightful information about how the software development lifecycle is currently operating. This data is the starting point for locating bottlenecks, examining inefficiencies, and suggesting areas for improvement.

4.1.3 Pain Points and Bottlenecks

The analysis of the VSM reveals several critical pain points and bottlenecks that impede process flow and contribute to extended lead times. Numerous things, from a lack of resources to poor team communication, might result in these bottlenecks. There were several bottlenecks that VSM discovered.

The first bottleneck found is in the Development and Documentation part of the process. As seen in the Value Stream Map this phase represents the most significant bottleneck with the LT of 30 days and the PT being 15 days. There are a few of the most common reasons for these bottlenecks to form. One is resource limitations where the development team appears to be operating under significant load, which leads to high work-in-progress (WIP) levels. There might be multiple concurrent tasks which make the load heavier. This overload leads to longer wait times for tasks to be initiated and completed, as developers are required to juggle competing priorities. The imbalance between team capacity and workload is a primary contributor to extended lead times. There also might be some Cross-Team Dependencies. The Development phase often depends on timely inputs from System Architects and POs. Particularly for design specifications and architectural guidance. Developer idle time might come from delays in obtaining this feedback, which adds to the longer lead time. This implies that there is a need to enhance the coordination and exchange of information between the technical and business teams.

Planning the feature was also a bit difficult according to the LT and PT. This suggests that it might be difficult to involve all the stakeholders in the planning phase, so the feature could be planned and refined as accurately as possible. The involvement of key stakeholders, such as System Architects, Product Owners, and Designers is critical because their input in resolving architectural questions and finalising task details might be the reason for time wasted or not. Delays in obtaining their approval or participation could contribute to the long lead times observed.

Another bottleneck was found in the Testing phase. The PT was 5 days, and the LT was 10 days which indicates that there is 50 percent waste on the phase. The large discrepancy between LT and PT indicates substantial delays. This might be because of the cooperation between the developers and testers. If there are some communication difficulties, there is a big chance that time is wasted on unimportant tasks.

The %C&A statistic assesses the accuracy and completeness of the tasks that are transferred between stages. The percentage C%A varies from 70% to 90% between phases, suggesting a comparatively high percentage of tasks that need to be redone or better clarified before they can be processed further. This might be a signal from the previously said problems in the Value Stream as insufficient specification of the features done or inconsistent feedback mechanisms and loops between the team members like feedback from testers to developers regarding defects or issues may not be timely or thorough, leading to rework and cycle time expansion.

4.1.4 Improvement Opportunities

The workshop produced several improvement opportunities from the brainstorming session as seen in the VSM and the blue boxes. The brainstorming session produced in total 29 (15+14) improvement ideas and all of them were feasible and could be implemented. Every participant had three votes for their use, and they could use them as they liked. The idea was that the workshop provides the top 3 improvements which are then implemented to the process flow. In this case, because there were 2 VSMs, after the vote there were 6 improvement ideas that team members selected three best to be implemented to their process flow.

The first improvement idea was to include designers in the pre-planning session before the sprint planning. This would improve the readiness of tasks before the formal sprint planning process. Adding the designers to the pre-sprint planning sessions would ensure that backlog items are adequately refined and that any ambiguities are resolved. This could reduce the LT for Sprint Planning, allowing the team to start the sprint without delays.

The second improvement is to identify enablers earlier in the progress. The failure to identify technological enablers promptly, such as reliance on external teams or infrastructure, causes many delays. Developer and tester idle time might be reduced by putting in place a systematic process for finding these enablers early in the development cycle, which will ensure that the essential inputs or resources are available when needed.

The third improvement Is to improve feedback loops and documentation. The low %C&A metrics show that there are insufficient feedback mechanisms within teams, especially between development and testing. Putting in place higher-quality automated feedback methods, including automated defect reporting and continuous testing technologies, could improve communication and guarantee that problems are resolved quickly. Enhancing feature documentation would also lessen the need for clarification and revision, facilitating a more seamless transition between phases.

To be noted in the second group brought Artificial Intelligence (AI) to the conversation as well when discussing improvements. Given company has piloted their on-premises AI and it shows a lot of promise. AI does not come and get utilized fast, but it shows a lot of promise and might boost the efficiency of many processes if it works and gets used correctly.

4.1.5 Conclusion of the Workshop 1

A thorough examination of the VSM as it stands now identifies several inefficiencies in the software development lifecycle. The biggest bottlenecks arise in the Development and Documentation and Planning stages, mostly because of insufficient pre-planning, technical debt, and resource constraints. The low %C&A indicators also point to the need for improved task refinement and communication throughout the value stream.

The proposed improvements, including increasing the involvement of conducting pre-sprint planning sessions with the designers, identifying enablers earlier, and improving feedback loops, are designed to address these issues that emerged during the VSM workshop. Implementing these changes is expected to reduce lead times, increase process efficiency, and improve the overall quality and accuracy of task handoffs.

4.2 Workshop 2

4.2.1 Team Background

Team 2 has been working together for about 5 years with small changes in the line-up. It works for the flagship product of the company and plays a critical role in that. They are also in the middle of the organisational SAFe change, and it has been for the last year. SAFe Release Train has been introduced as a way of working for the team and is still a new thing for the team and its members. Most of the team members are internal and a few external workers. The team size is 12 team members. Team members work officially in one site but are rarely together due to the hybrid work and a few members are located in another city and doing their work fully remote.

4.2.2 Current State VSM

The second workshop outlines a refined software development process that spans from the initial customer request to the final stage of solution testing and customer feedback. The phases include Customer Request, UI Concept, Feature Analysis, Tech Meet, Implementation, Feature Testing, and finally, Releasing and Solution Testing.

Various team members are involved across these stages, including Product Owners (PO), Business Analysts (BA), the Architecture (ARC) and Development teams, UX/UI Designers, and Technical Leads. Each phase is accompanied by PT and LT metrics, which offer a detailed look into where inefficiencies may lie in the current process.

There is a discrepancy between the amount of time a task is worked on and the total amount of time it takes to complete in the pipeline; for instance, the total PT for all phases in this VSM is 51.3 days, but the LT is 93.2 days. This disparity draws attention to ineffective handoffs and teamwork, as well as potential process bottlenecks that impede the flow.

The workflow spans several key phases:

- UI Concept and Feature Analysis, led by cross-functional teams including Product Owners (PO), Business Managers (PBM), and Architects.
- Development and Testing, where Development teams, UX/UI Designers, and Technical Leads play a pivotal role in implementing and validating features.
- Releasing and Solution Testing, managed by Product Owners and Solution Testing teams, ensure that the solution meets both technical and business requirements before final release.

Each phase is measured for %C&A. The overall %C&A for this workflow is 0,002% signifying that a considerable portion of tasks require clarification or rework before completion and there lies a problem.

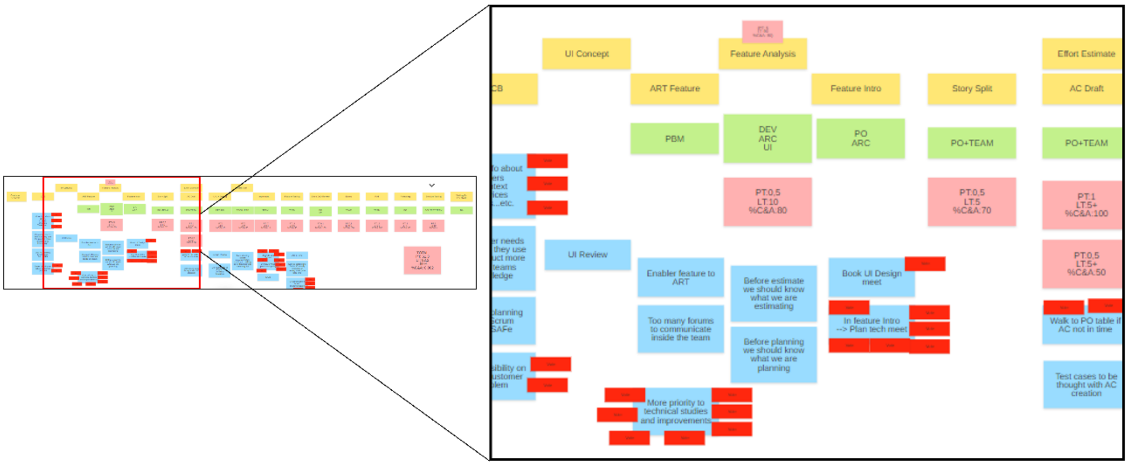


Figure 7 CSM from the Team 2

4.2.3 Pain Points and Bottlenecks

A more thorough examination of the second VSM workshop was able to identify several significant pain points and bottlenecks that reduce process efficiency as a way of work and for their process as a team. The team was able to specify the bottlenecks during the workshop in a few specific phases in the process.

The development phase presents a notable bottleneck with an LT of 14 days and a PT of 3 days for the tech meet. Considering the comparatively short PT, it seems that the time required to properly implement and test a feature is unnecessarily long. This suggests that during the phases of implementation and testing, there may be a delay in feedback and communication. This bottleneck may also be caused by resource constraints, as the development team may need to manage several projects at once.

Effort estimates and the AC draft are another bottleneck in the process. PT is 1,5 days and LT is 10+ days because the team said it sometimes takes longer and sometimes 10 days. Still, if it takes the 10days there is a significant amount of time wasted in this phase of the process. This suggests that cross-team communication, particularly between development and technical leadership, could be improved to streamline the technical requirements and feature specifications.

The %C&A in several phases remains low and that is a significant problem in the process. This usually suggests that there may have been a lack of clarity in the original task specifications, which resulted in rework and lengthier lead times. Increased accuracy of tasks transferred between phases and a decrease in these disparities could be achieved through improved documentation and communication.

It should be noted that the SAFe's way of working is relatively new to the team, so the process has not had the time to mature and develop among the team. This is something that only time and iterations will fix, so just doing more increments with the team will probably enhance the process when the team have the skill to work within the framework's limits.

4.2.4 Improvement Opportunities

During the brainstorming session in Workshop 2, several improvement opportunities were identified and categorized into feasible solutions. The session yielded 23 potential improvements, from which the top three were voted for implementation.

The first improvement opportunity that was found in the workshop was including Technical Leads sooner in the planning and estimate stages is the first improvement. One obstacle that was identified was the time it took to define technical requirements, especially during the Tech Meet. Teams can lessen the possibility of miscommunications and implementation inefficiencies by including Technical Leads in conversations during the Feature Analysis and UI Concept stages. This will facilitate smoother transitions between planning and development. So, the tech meet should be done earlier in the process than it is done now.

The second improvement is to freeze the UI when the implementation is underway. The team has a problem where the UI picture changes when developers are already doing the task. This leads to misunderstandings and work going to waste. Now, there should be one picture of the UI design attached to Jira tickets, so there is no room for mistakes, and nobody could not change the design without telling anybody. This also shows that there are some problems with communication in the team.

The third improvement was to take time and prioritize technical studies and improvements. The team was a bit worried that when always building new features and stories there is no time to upgrade and enhance the environment where the build things go. This grows the technical debt and at some point, it will catch the team back if not noticed.

4.2.5 Conclusion of the Workshop 2

The VSM from the second session identifies several ongoing problems, particularly regarding task clarity and communication between phases. Cross-functional cooperation frequently breaks down during the development and testing phases, which is when bottlenecks are concentrated. The brainstorming session revealed several areas for improvement. These include task definition, early technical engagement, and worry about technical debt. These changes are intended to relieve these bottlenecks and optimize the software development process.

The team anticipates that by putting these adjustments into practice, lead times will be shortened, and task accuracy will rise, improving overall process efficiency. Maintaining a constant focus on automation and cooperation will also assist in reducing future bottlenecks and enhance the way work moves through the value stream.

4.3 Workshop 3

4.3.1 Team Background

Team 3 has been working together for the last 10 years. It works in an older product, and it is the most used and worked-on product of the three. The team is part of a release train and works incrementally. Teams have 10 members, and most of them are internals who work in the site itself every day and rarely work remotely.

4.3.2 Current State VSM

The third VSM outlines a comprehensive process flow from Customer Request to Delivery and Customer Support. This workflow includes several key stages: Screening, Concepting, Design, Development, Testing, Documentation, Integration, and Approval, with a final handover to customer support.

During various stages, important parties like System Feature Teams, Product Owner, Architects, and Business Analysts are involved. This VSM stands out for having long lead and process times, which indicates that there are a lot of areas where it might be improved, like cutting down on idle time, enhancing communication, and expediting task handoffs.

The total PT in this VSM is 111 days, while the LT stands at a significant 579 days. The %C&A is 16.3%, indicating major inefficiencies and frequent rework across phases.

This workflow covers the following stages:

- Screening and concepting, where Product Owners, Architects, and Software teams evaluate customer needs and define the project.
- Development and Testing, where the feature team designs, builds, and tests the solution in collaboration with the UI/UX team.
- System Testing, Integration, and Release, involving cross-team collaboration between developers, testers, and system feature teams to validate and release the product.

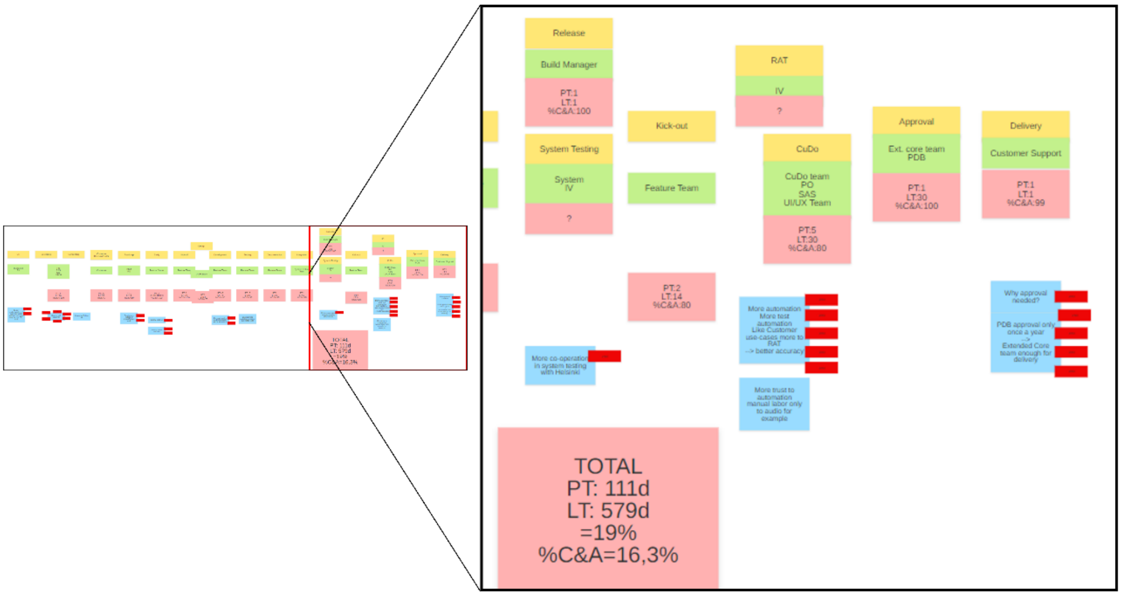


Figure 8 CSM from the Team 3

4.3.3 Pain Points and Bottlenecks

This VSM showed several bottlenecks and inefficiencies, especially concerning long lead times and a low percentage of %C&A, which indicated considerable delays and a high frequency of rework.

It appears that tasks are left unfinished for prolonged periods throughout the testing phase, which has a PT of 15 days and an extended LT of 45 days. With testing and subsequent integration delayed, the low %C&A of 80% suggests problems with task definition and handoffs between the feature teams and the system feature teams.

With a 95% %C&A, the Integration phase has a lead time of 60 days. Most jobs are accurate, however, proceeding to the final testing and release phases is significantly delayed due to the lengthy waits between feature handoffs. The creation of automated tests by developers has been identified to increase efficiency and decrease bottlenecks.

The Documentation phase stands out with a Process Time of 2 days but a Lead Time of 14 days, reflecting delays in formalizing documentation handoffs and approvals. Moreover, the approval phase shows an LT of 30 days, which might be attributed to a slow approval mechanism, potentially exacerbated by the annual PDB review process.

4.3.4 Improvement Opportunities

During Workshop 3's brainstorming session, several issues that needed improvement were noted. The main goals were to decrease the enormous lead times, boost task accuracy, and enhance team communication.

The first bottleneck identified was the long lead time between the concept and decision/order phases. At this point, better task prioritizing and a deeper comprehension of client demands may help to minimize rework and define expectations. Later in the process, idle time may be decreased by establishing more precise parameters for customer follow-up and engagement at the outset. The team would like to know and meet the actual customer and end users of the product. It would bring clarity to the progress and give more purpose to the team. It gives the team motivation to see what they have done and how the product helps the customer/end user.

The second bottleneck is that integration and testing still pose challenges, especially since manual testing is slower and requires more resources compared to automated testing. Promoting increased automated testing and involving developers in the process of writing automated tests directly could reduce a significant amount of idle time and expedite the verification process. Additionally, by increasing work accuracy through automation, the percentage %C&A would rise at this point. The team would want to put time and effort into the automated testing to enhance the accuracy and time in this phase.

Third, the lengthy LT observed in the Approval phase could be mitigated by implementing a more frequent, continuous approval process. Currently, the once-a-year PDB review significantly delays approval for tasks, which could be streamlined by establishing shorter feedback loops and more frequent approvals.

4.3.5 Conclusion of the Workshop 3

The third VSM highlights several noteworthy bottlenecks and problems, the most prominent of which are the lengthy lead times and low %C&A rates at various stages. The phases of Concepting, Development, and Testing are especially vulnerable to delays because of imprecise task specifications, lagging feedback loops, and manual procedures.

It is anticipated that the enhancements made during the brainstorming session—namely, promoting automation in testing, expediting the approval process, and augmenting client engagement at the outset—will shorten lead times and increase task correctness. The team hopes to improve the productivity of the software development process and facilitate more seamless transitions between stages by putting these changes into practice.

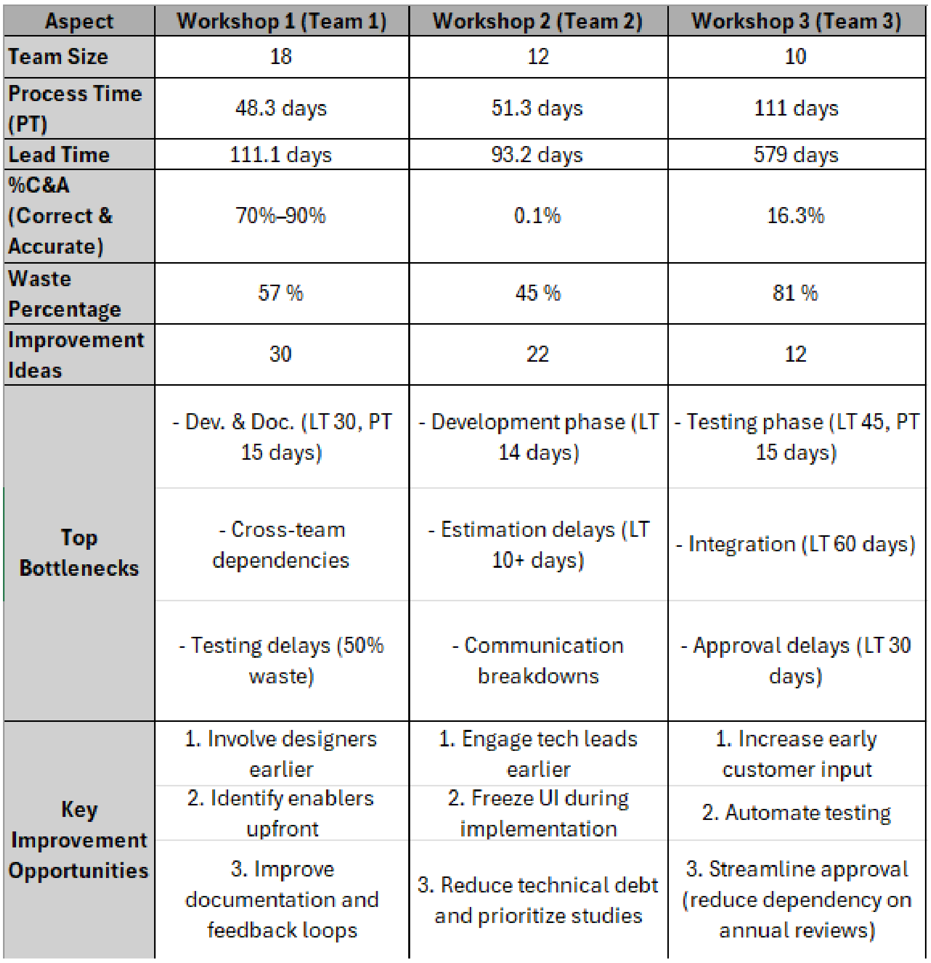
4.4 Comparative Analysis of Workshop Findings

The three workshops' outcomes show different trends and common issues that the software development teams face. Although every team has a different workflow and set of problems, there are clear common bottlenecks and variations in process performance. To give a better understanding of inefficiencies and areas for development, this section compares the workshop results.

4.4.1 Workshop Summary

Table 6 provides an in-depth comparison of the main findings from the three Value Stream Mapping workshops. Each team actively engaged in the identification of bottlenecks and inefficiencies within their respective processes, resulting in the generation of numerous improvement ideas among team members. Workshops challenged their participants to think about their processes as a team and break the silos inside the teams. During brainstorming sessions, teams explored and discussed ways to streamline their team's workflow, reduce waste, and improve overall productivity. The top three improvement opportunities were voted for each team and workshop to get prioritized for implementation to enhance their processes. Table 6 below highlights metrics like team size, process and lead times, %C&A, and the primary waste percentages identified in each workshop, alongside key improvement opportunities.

Table 6. Summary of the workshops



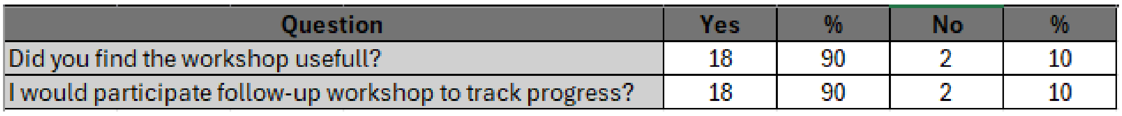
Aspect	Workshop 1 (Team 1)	Workshop 2 (Team 2)	Workshop 3 (Team 3)
Team Size	18	12	10
Process Time (PT)	48.3 days	51.3 days	111 days
Lead Time	111.1 days	93.2 days	579 days
%C&A (Correct & Accurate)	70%–90%	0.1%	16.3%
Waste Percentage	57 %	45 %	81 %
Improvement Ideas	30	22	12
Top Bottlenecks	- Dev. & Doc. (LT 30, PT 15 days)	- Development phase (LT 14 days)	- Testing phase (LT 45, PT 15 days)
	- Cross-team dependencies	- Estimation delays (LT 10+ days)	- Integration (LT 60 days)
	- Testing delays (50% waste)	- Communication breakdowns	- Approval delays (LT 30 days)
Key Improvement Opportunities	1. Involve designers earlier	1. Engage tech leads earlier	1. Increase early customer input
	2. Identify enablers upfront	2. Freeze UI during implementation	2. Automate testing
	3. Improve documentation and feedback loops	3. Reduce technical debt and prioritize studies	3. Streamline approval (reduce dependency on annual reviews)

4.4.2 Overall Value of VSM

There is a value produced by VSM to the teams. There was a feedback query to the workshop participants, and it got an overall 20 responses. As seen in Table 7 90% of the participants found the VSM workshop useful and would participate in a follow-up workshop to track the progress of the changes implemented.

The two "No" answers were reasoned as follows, "Everything is okay how it is" and the other one was "In software projects, such a simplistic approach to the value chain is not giving the best input". They thought that the way that VSM was implanted or used was not the best way possible. For the second negative comment, there might be a reason, and it was the restricted time slot. Normally

Table 7. Feedback survey results



Question	Yes	%	No	%
Did you find the workshop usefull?	18	90	2	10
I would participate follow-up workshop to track progress?	18	90	2	10

VSM workshops take about 2 days to facilitate and there was no time to do the future map which would have helped to illustrate the purpose of the VSM to the participants. When only doing the current state map there is a chance that it looks like from a team member's perspective that there is no plan to implement the changes that were brainstormed, and the workshops only produce ideas that are just forgotten.

There were a lot of positive comments in the feedback. Many comments related to the overall view of each participant and how they see the whole process. One key takeaway from the feedback is the ability of VSM to break down silos and create a holistic view of the process. VSM gave participants a more comprehensive grasp of the whole value flow, which was helpful in businesses where teams often concentrate exclusively on their assigned responsibilities. Addressing bottlenecks and inefficiencies that impact cross-functional teams requires this new viewpoint.

Positive feedback from participants highlighted the importance of this increased visibility, expressing how it improved their understanding of their role as well as the difficulties and interdependencies in the process. This change in perspective may result in more cooperative solutions, which would lower friction and boost the effectiveness of the software development process.

The workshop was also seen as an opportunity to stop and think about what the teams were doing. When working on their daily tasks, the team could rarely discuss the challenges and things they do as a group and everybody present. Many of the comments suggested that this type of workshop should be done once a year to track the progress and see if they can make changes to the progress.

Ultimately, the insight gained from the VSM workshops improves team roles individually and fosters a collaborative environment that aims to increase productivity and effectiveness in the software development process. Companies can better manage the complexity of their workflows and pave the road for creative solutions and long-term performance improvements by adopting this new viewpoint. VSM brought value and insights to every team in the research regardless of the team size, maturity or product. It also works as a team-building exercise and can break silos inside the teams.

5 DISCUSSION

This chapter addresses the research questions, presents a summary of the results that were previously introduced, compares them to earlier studies, discusses the findings, and makes recommendations for both theory and practice.

5.1 Interpretation of Findings

The results from the VSM workshops provided significant insights into the company's software development processes. The primary issues found by VSM largely correspond with the body of research already available on bottlenecks in knowledge-based work settings (Ali, et al., 2016). Systemic inefficiencies in software development are frequently brought about by team dependencies and incomplete task handovers, which prolong lead times in stages like development and testing (Fitzgerald & Stol, 2015).

Throughout the sessions, one of the main conclusions has been the difference between PT and LT. This gap shows that even while some actions may execute relatively quickly (PT), delays brought on by resource availability, permissions, and feedback loops result in a substantially longer waiting period (LT). Theoretical concerns about waste and inefficiency in lean processes are strengthened by this discovery. These kinds of delays are seen as "waste" in the software environment, according to Petersen and Wohlin (2011), and they can be avoided with improved coordination and more open lines of communication.

Several bottlenecks were identified during the workshops, particularly in the development, testing, and handoff phases between teams. Long lead times, non-optimal communication, and resource bottlenecks were typical issues. VSM was a useful tool in visualizing these problems, particularly during the work handover and cross-functional collaboration phases. For instance, in Workshop 1, there were discernible delays in the Development and Documentation phase because of a shortage of resources and a delay in the handoffs between the development and design teams. These findings are consistent with earlier research

(Petersen & Wohlin, 2011), which highlights the usefulness of VSM in locating inefficiencies in intricate, knowledge-based processes like software development.

The effectiveness of VSM was demonstrated by how it helped teams identify inefficiencies collectively, develop focused improvement plans, and enhance process transparency. For instance, in Workshop 2, improvements such as freezing UI designs during development were identified to reduce confusion and rework. Comparably, Workshop 3 stressed how critical it is to enhance customer communication and better match development activities with end-user requirements. The study also shows how, as Workshop 2 illustrates, VSM can improve sprint planning and backlog prioritization to further improve agile and SAFe practices. This is in line with research from Power and Conboy (2015), who contend that VSM in agile environments provides an organized method for iteratively enhancing the value stream.

The workshops also emphasized the value of involving key stakeholders early on, especially in cross-functional collaboration throughout the planning and refinement phases, such as technical leaders and product owners. This result is in line with the agile and lean literature, which emphasizes the significance of dismantling organizational silos and making sure team members are aligned early on (Bang et al., 2013). By drawing attention to areas where cooperation broke down and outlining interdependence, VSM acted as a diagnostic tool that helped everyone comprehend the workflow.

A unique concern raised during Workshop 2 was the issue of technical debt. The team stated that upper management was pressuring them to concentrate only on creating new features while the environment in which these features were implemented had become outdated and bug-ridden, making the work more difficult and time-consuming. Even though this worry had nothing to do with the value stream itself, it became apparent that it was an important problem that required addressing. This was ranked by the team as one of the top three enhancements to be made, highlighting the necessity of taking your time while updating and maintaining the environment. This may only be the other side of the coin. Team 2 perceives that the pursuit of business objectives took precedence over environmental maintenance efforts, possibly indicating a tendency to deflect responsibility by seeking external factors to account for any issues. If maintenance work is critical, it should be prioritized. Business goals should not take precedence over critical maintenance work. There may be a situation where the team assumes that developing a new feature is the top management's priority and feels unable to speak up. Anyway, this demonstrates how operational issues that are not immediately related to the value stream can be brought to light through VSM workshops.

Finally, VSM encourages teamwork by providing a forum for all team members to share their concerns and participate in the workshop's decision-making process. One of VSM's advantages is its inclusivity, which guarantees that all viewpoints are considered when proposing process enhancements. Even if there are a few more vocal people in the workshop, everyone gets the opportunity to add their suggestions for improvement and vote for the most important reforms in the process. Of course, the loudest personalities may take the general discussion in a certain direction, but everyone always gets a chance to present their opinions and suggestions thanks to the structure of the workshop.

5.2 Value of VSM in Software Development

The workshops proved that VSM is a useful tool for identifying software development inefficiencies. In software, job handoffs and information flow are the main points of emphasis, as opposed to traditional manufacturing, where VSM concentrates on the physical flow of materials. The way that VSM aided in locating bottlenecks in cross-team communication, delayed feedback loops, and erroneous job handovers demonstrated its capacity to map these intangible elements. This result is consistent with research suggesting that VSM should be modified for knowledge work by emphasizing feedback cycles and communication channels more (Fitzgerald & Stol, 2015). The workshops showed that teams can better understand their workflow and pinpoint areas where they can increase efficiency, especially in cutting down on waste and delays, thanks to the visibility that VSM provides.

The workshops also made clear, though, that teamwork and a willingness to question established procedures are critical to VSM's success in software development. A small percentage of participants voiced doubts in the feedback regarding the usefulness of VSM, especially in more complicated software settings. This reflects concerns raised in earlier research (Dinis-Carvalho et al., 2015), where it was noted that the abstraction needed for VSM in non-physical workflows occasionally oversimplified the complexities of software development.

In the feedback from workshop 2, there was a concern that the VSM does not suit their process as well as it is supposed to be. This means that it might be a bit too simple to measure things as accurately as possible. In agile, lean and SAFe framework thins move iteration to iteration which brings the challenge to try to simplify the iterative process as one value stream. Because VSM was originally invented in a manufacturing environment where things go only one stream forward it might work better with a waterfall framework VSM could be more efficient but in agile the tool gives enough information and data to discover bottlenecks in inefficiencies.

5.3 Implications for Theory and Practice

The research broadens understanding of lean principles in software development from a theoretical standpoint. It supports the idea that, with the right adjustments, manufacturing-specific tools, such as VSM, can be converted for use in software development. This builds on earlier studies on lean's application in agile frameworks by demonstrating how VSM can act as a link between general agile concepts and targeted process improvement projects.

Validating the importance of feedback loops and cross-functional cooperation as the main causes of software development bottlenecks is a significant theoretical contribution. According to the study's findings, one of the main causes of inefficiency is feedback delays. This idea is supported by agile literature but isn't as frequently discussed in lean theory (Power & Conboy, 2015).

The report provides useful advice on how to include VSM into routine software development tasks for practitioners. The workshops show that regular VSM sessions, either as part of sprint reviews or backlog refinement sessions, yield the best results. Teams may keep an eye on their processes and spot new bottlenecks quickly by regularly mapping and updating the value stream.

This study also suggests increasing automation in areas that are bottlenecks, such as deployment and testing. The research on continuous delivery and integration already in existence supports the workshops' conclusion that manual chores are a key cause of delays (Ali, et al., 2016). Teams can decrease manual involvement, minimize lead times, and improve process accuracy by investing in automation.

In the end, the study highlights how important it is to foster a continuous improvement culture. Beyond just identifying inefficiencies, VSM supports teams in taking responsibility for process enhancements and engaging in dialogues aimed at resolving issues. This aligns with the broad goals of agile and lean methodologies, which give priority to continuous feedback and incremental improvements (Fitzgerald & Stol, 2015).

6 CONCLUSION

This thesis set out to explore how Value Stream Mapping (VSM) could be applied to enhance software development processes. With an emphasis on using VSM as a tool to discover inefficiencies, bottlenecks, and areas for improvement within software development workflows, the research was carried out through a series of workshops within a case company. The study set out to address two main queries:

- What are the main pain points of the company's software development processes, and how can they be identified by Value Stream Mapping?
- What value does VSM bring to the company and its teams, and how can this value be realized in the current software development environment?

6.1 Summary of Key Findings

Key conclusions from the workshops included the confirmation that VSM is an effective method for identifying and resolving software development inefficiencies. Key conclusions include:

1. Identifying inefficiencies: The workshops made clear that there were large differences between LT and process time PT. Even though some tasks might be completed rapidly, waiting times were greatly increased by resource availability, permissions, and communication delays. This is consistent with lean theory, which sees delays of this kind as waste that may be reduced by better cooperation and communication.
2. Visualization of Bottlenecks: Throughout the software development lifecycle, bottlenecks can be effectively visualized with the help of VSM. Long lead times, non-optimal communication and resource bottlenecks were common in the development, testing, and handoff stages, where notable problems were found during the workshops. Workshop 1

brought to light, for example, observable delays in the Development and Documentation phase, which were caused by a lack of resources and ineffective handoffs between the development and design teams. VSM can show the inefficiencies simply and understandably.

3. Cross-Functional Collaboration: During the workshops, the importance of involving key stakeholders early in the planning and refinement stages, such as technical leaders and product owners, was emphasized. Early involvement is essential for improving cross-functional cooperation and guaranteeing team member alignment, which helps break down organizational silos that could block progress.
4. Team Engagement and Inclusivity: VSM's structured approach promotes the value of teamwork by providing everyone with a platform to express their ideas and participate in decision-making. This inclusivity is a vital component of VSM, ensuring that different viewpoints are considered when determining how to improve the process. In the workshops, all participants had the opportunity to voice their opinions on potential improvements and present their arguments due to the phase where everybody's improvement ideas were discussed.
5. Managing Technical Debt: During the workshops, the topic of technical debt was brought up, which was an important concern to notice. Team members stated that there was often more managerial pressure to create new features than to maintain and improve the current development environment. This might be a one-sided view, but it still needs to be noted. This worry emphasizes how important it is to strike a balance between continuing maintenance and fresh development. This shows that VSM can bring team members together and raise some concerns that are not directly attached to the progress.

6.2 Theoretical Contributions

This research enhances the larger understanding of lean software development approaches. It proves that knowledge work may effectively adapt manufacturing-specific technologies like VSM, bridging the gap between basic agile concepts and focused process improvements. The study expands on the conversation within lean theory while supporting agile literature by validating the significance of feedback loops and cross-functional cooperation as crucial elements in alleviating software development bottlenecks. This thesis contributes to the theoretical discussion on the suitability of VSM for software contexts by emphasising these dimensions.

6.3 Practical Recommendations

From a practical perspective, this thesis offers several actionable recommendations for practitioners. VSM must be regularly integrated into software development processes. Ideally, it would be good to be used in an increment review or as often in a sprint review but once in a 4 iteration is a good time to make changes and track the progress. If done too often, the changes might not be visible enough and it might lower the motivation of the people who are making the changes. Practically it also would affect the actual work being done if done too often.

The results indicate that there is an urgent need to boost automation in areas that have been identified as bottlenecks, especially in deployment and testing. Investing in automation will reduce human error and streamline procedures, resulting in quicker delivery and higher-quality products. For example, one of the factors that could beneficially impact the processes is AI. The company should prioritize efforts to fully realize the potential value of AI.

Effective use of VSM requires fostering a culture that values ongoing development. Organizations can establish a culture of continuous learning and adaptation by empowering teams to assume accountability for their processes and participate in collaborative discussions aimed at addressing challenges. This means to emphasize the teams and how they work. Top management can set the big targets and show the way of the work, but there should be more "decentralized command" where the team makes the plans and executes them. That is how there is more ownership in the team, and they know the capacity and the environment to work as efficiently as possible.

Along with developing new features, organizations should give priority to maintaining and upgrading existing software environments. By balancing these efforts, technical debt will not necessarily build-up, resulting in longer-term development processes that are more streamlined and effective.

6.4 Limitations and Future Research

Even though this study provides insightful information, it is important to recognize its limitations regarding time and money. The research was done inside a single company, so it is possible, that the conclusions cannot be applied to different situations or companies. Deeper investigations may expand on this work by utilizing VSM in various companies and organizations, investigating its wider suitability and efficacy in a range of software development contexts to have more comprehensive results.

This thesis focused on shorter-period improvements identified during the VSM workshops and how the VSM can help enhance software processes due to limited time and resources. Future research should study the long-term impacts of VSM on process efficiency, identifying waste, team productivity, and return on investment (ROI) of implementing VSM. This could be done for example by tracking the progress of each team with a follow-up workshop and studying whether are they able to make their processes more efficient through VSM. This research would take time and resources to execute.

Additionally, researching how VSM might be adapted to solve the difficulties of modern software development—particularly in the contexts of DevOps, agile scaling frameworks, and microservices architecture—would be a great path for future exploration. As said, there are many ways to execute agile and every company has its own. The scope of the study could be broader and include many other teams from other companies as well.

6.5 Final Thoughts

In conclusion, this thesis highlights the potential of Value Stream Mapping as a lean and transformative tool to use in software development. Through the visualization of processes and collaborative identification of inefficiencies, VSM facilitates a culture of continuous improvement and improves communication among all organizational levels and most importantly within the teams. The insights gained from this research not only advance theoretical understanding but also provide practical guidance and guidelines to optimize and enhance software development practices and processes.

More effective processes, improved customer responsiveness, and a greater competitive edge in an increasingly complicated software market can result from the successful implementation of VSM. The journey towards optimizing software development processes through VSM requires commitment, adaptability, and a willingness to learn, but the rewards—improved productivity, quality, and team engagement—are well worth the effort and might lead to great results.

REFERENCES

Artjoms, A. (2024, July 18). Software development life cycle (SDLC). *Medium*. https://medium.com/@artjoms/software-development-life-cycle-sdlc-6155dbfe3cbc (11.10.2024)

Adenowo, A. A., and Adenowo, B. A. (2013). Software engineering methodologies: a review of the waterfall model and object-oriented approach. *International Journal of Scientific and Engineering Research*, *4*(7), 427-434.

Ali, N. B., Petersen, K., and De França, B. B. N. (2015). Evaluation of simulation-assisted value stream mapping for software product development: Two industrial cases. *Information and software technology*, *68*, 45-61.

Ali, N. B., Petersen, K., and Schneider, K. (2016). FLOW-assisted value stream mapping in the early phases of large-scale software development. *Journal of Systems and Software*, *111*, 213-227.

Al-Haddad, S., and Kotnour, T. (2015). Integrating the organizational change literature: a model for successful change. *Journal of organizational change management*, *28*(2), 234-262.

Al-Saqqa, S., Sawalha, S., and AbdelNabi, H. (2020). Agile software development: Methodologies and trends. *International Journal of Interactive Mobile Technologies*, *14*(11).

Amazon Web Services. (n.d.). Introduction to DevOps value stream mapping. *AWS Prescriptive Guidance*. Retrieved October 11, 2024, from https://docs.aws.amazon.com/prescriptive-guidance/latest/strategy-devops-value-stream-mapping/introduction.html

Arnold, J. A., Arad, S., Rhoades, J. A., and Drasgow, F. (2000). The empowering leadership questionnaire: The construction and validation of a new scale for measuring leader behaviors. *Journal of organizational behavior*, *21*(3), 249-269.

Ashkenas, R. (2013). Change management needs to change. *Harvard Business Review*, *3*, 20-23.

Avison, D., Baskerville, R., and Myers, M. (2001). Controlling action research projects. *Information technology and people*, *14*(1), 28-45.

Bang, S. K., Chung, S., Choh, Y., and Dupuis, M. (2013, October). A grounded theory analysis of modern web applications: knowledge, skills, and abilities for DevOps. In *Proceedings of the 2nd annual conference on Research in information technology* (pp. 61-62).

Bassil, Y. (2012). A simulation model for the waterfall software development life cycle. *arXiv preprint arXiv:1205.6904*.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... and Thomas, D. (2001). The agile manifesto.

bin Ali, N. (2015). *Operationalization of lean thinking through value stream mapping with simulation and FLOW* (Doctoral dissertation, Blekinge Tekniska Högskola).

Bridges, W. (2009). *Managing transitions: Making the most of change*. Da Capo Press.

Braude, E. J., and Bernstein, M. E. (2016). *Software engineering: modern approaches*. Waveland Press.

Brenner, R., and Wunder, S. (2015, April). Scaled Agile Framework: Presentation and real world example. In *2015 ieee eighth international conference on software testing, verification and validation workshops (icstw)* (pp. 1-2). IEEE.

Burnes, B. (2011). Introduction: Why does change fail, and what can we do about it?. *Journal of change management*, *11*(4), 445-450

Burnes, B., and Jackson, P. (2011). Success and failure in organizational change: An exploration of the role of values. *Journal of change management*, *11*(2), 133-162.

Campbell, R. J. (2008). Change management in health care. *The health care manager*, *27*(1), 23-39.

Campanelli, A. S., and Parreiras, F. S. (2015). Agile methods tailoring–A systematic literature review. *Journal of Systems and Software*, *110*, 85-100.

Chen, J. C., and Cox, R. A. (2012). Value stream management for lean office—A case study.

Chen, L., and Meng, B. (2010). The application of value stream mapping based lean production system. *International journal of business and management*, *5*(6), 203.

Cho, J. J. (2010). An exploratory study on issues and challenges of agile software development with Scrum. *All Graduate theses and dissertations*, 599.

Darwish, M., Haque, B., Shehab, E., and Al-Ashaab, A. (2010, September). Value stream mapping and analysis of product development (engineering) processes. In *Proceedings of The 8th International Conference on Manufacturing Research (ICMR 2010), University, Durham, UK* (pp. 14-16).

Dingsøyr, T., Moe, N. B., Fægri, T. E., and Seim, E. A. (2018). Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation. *Empirical Software Engineering*, *23*, 490-520.

Dinis-Carvalho, J., Moreira, F., Bragança, S., Costa, E., Alves, A., and Sousa, R. (2015). Waste identification diagrams. *Production Planning and Control*, *26*(3), 235-247.

Eckstein, J. (2013). *Agile software development in the large: Diving into the deep*. Pearson Education.

Elbanna, A., and Sarker, S. (2015). The risks of agile software development: learning from adopters. *IEEE Software*, *33*(5), 72-79.

Erdogan, B., Anumba, C., Bouchlaghem, D., and Nielsen, Y. (2005, September). Change management in construction: the current context. In *21st Annual ARCOM Conference* (pp. 1085-1095). SOAS, University of London. Association of Researchers in Construction Management.

Erickson, J., Lyytinen, K., and Siau, K. (2005). Agile modeling, agile software development, and extreme programming: the state of research. *Journal of Database Management (JDM)*, *16*(4), 88-100.

Fairley, R. E. (2011). *Managing and leading software projects*. John Wiley & Sons.

Fitzgerald & Stol, 2015, B., and Stol, K. J. (2017). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, *123*, 176-189.

Ford, C. M. (1996). A theory of individual creative action in multiple social domains. *Academy of Management review*, *21*(4), 1112-1142.

Ford, J. D., Ford, L. W., and D

Amelio, A. (2008). Resistance to change: The rest of the story. *Academy of management Review*, *33*(2), 362-377.

Fowler, M. (2018). *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional.

Giuffrida, R., and Dittrich, Y. (2015). A conceptual framework to study the role of communication through social software for coordination in globally-distributed software teams. *Information and Software Technology*, *63*, 11-30.

IEEE Standard Glossary of Software Engineering Terminology," in *IEEE Std 610.12-1990* , vol., no., pp.1-84, 31 Dec. 1990, doi: 10.1109/IEEESTD.1990.101064.

Jammalamadaka, K., and Krishna, V. R. (2013). Agile software development and challenges. *International Journal of Research in Engineering and Technology*, *2*(8), 125-129.

Jayatilleke, S., and Lai, R. (2018). A systematic review of requirements change management. *Information and Software Technology*, *93*, 163-185.

Jeong, B. K., and Yoon, T. E. (2016). Improving IT process management through value stream mapping approach: A case study. *JISTEM-Journal of Information Systems and Technology Management*, *13*(3), 389-404.

Jones, G. R. (2001). Organizational theory: Text and cases

Jones, J., Firth, J., Hannibal, C., and Ogunseyin, M. (2021). Factors contributing to organizational change success or failure: a qualitative meta-analysis of

200 reflective case studies. In *Research Anthology on Digital Transformation, Organizational Change, and the Impact of Remote Work* (pp. 1427-1450). IGI Global.

Kettunen, P., and Laanti, M. (2008). Combining agile software projects and large-scale organizational agility. *Software Process: Improvement and Practice*, *13*(2), 183-193.

Khan, S. U., and Azeem, M. I. (2014). Intercultural challenges in offshore software development outsourcing relationships: an exploratory study using a systematic literature review. *IET software*, *8*(4), 161-173.

Khurum, M., Petersen, K., and Gorschek, T. (2014). Extending value stream mapping through waste definition beyond customer perspective. *Journal of Software: Evolution and Process*, *26*(12), 1074-1105.

Kotter, J. P. (2007). Leading change: Why transformation efforts fail. In Museum management and marketing (pp. 20-29). Routledge.

Kumar, S. A., and Kumar, T. A. (2011). Study the impact of requirements management characteristics in global software development projects: an ontology based approach. *International Journal of Software Engineering and Applications*, *2*(4), 107.

Kurapati, N., Manyam, V. S. C., and Petersen, K. (2012). Agile software development practice adoption survey. In *Agile Processes in Software Engineering and Extreme Programming: 13th International Conference, XP 2012, Malmö, Sweden, May 21-25, 2012. Proceedings 13* (pp. 16-30). Springer Berlin Heidelberg.

Lewin, K. (1948). Field theory.

Matharu, G. S., Mishra, A., Singh, H., and Upadhyay, P. (2015). Empirical study of agile software development methodologies: A comparative analysis. *ACM SIGSOFT Software Engineering Notes*, *40*(1), 1-6.

Mento, A., Jones, R., and Dirndorfer, W. (2002). A change management process: Grounded in both theory and practice. *Journal of change management*, *3*(1), 45-59.

Maroofi, F., and Dehghan, S. (2012). Performing lean manufacturing system in small and medium enterprises. *International Journal of Academic Research in Accounting, Finance and Management Sciences*, *2*(3), 156-163.

Miller, G. J. (2013). Agile problems, challenges, and failures. Project Management Institute.

Myers, M. and Avison, D. (2002). An introduction to qualitative research in information systems. In Myers, M. D., and Avison, D. Introducing Qualitative Methods: Qualitative research in information systems (pp. 2-12). London: SAGE Publications.

Myers, M. D. (1997). Qualitative Research in Information Systems. MIS

Quarterly, 21(2).

Nasir, N., Minhas, N. M., and Sweden, S. E. R. L. (2018). Implementing Value Stream Mapping in a Scrum-based project-An Experience Report. In *QuASoQ@ APSEC* (pp. 44-51).

Nerur, S., Mahapatra, R., and Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM*, *48*(5), 72-78.

Niazi, M., El-Attar, M., Usman, M., and Ikram, N. (2012, May). GlobReq: A framework for improving requirements engineering in global software development projects: Preliminary results. In *16th International Conference on Evaluation and Assessment in Software Engineering (EASE 2012)* (pp. 166-170). IET.

Nowak, M., Pfaff, H., & Karbach, U. (2017). Does Value Stream Mapping affect the structure, process, and outcome quality in care facilities? A systematic review. *Systematic reviews*, *6*, 1-11.

Pargaonkar, S. (2023). A Comprehensive Research Analysis of Software Development Life Cycle (SDLC) Agile and Waterfall Model Advantages, Disadvantages, and Application Suitability in Software Quality Engineering. *International Journal of Scientific and Research Publications (IJSRP)*, *13*(08), 345-358.

Petersen, K., and Wohlin, C. (2011). Measuring the flow in lean software development. *Software: Practice and experience*, *41*(9), 975-996.

Pfleeger, S.L. and Atlee, J.M. (2006). *Software Engineering: Theory and Practice*, 3rd Edition. US: Prentice Hall.

Pfleeger, S. L., & Atlee, J. M. (2010). *Software engineering: Theory and practice (4th ed.)*. Prentice-Hall.

Power, K., and Conboy, K. (2015, May). A metric-based approach to managing architecture-related impediments in product development flow: an industry case study from Cisco. In *2015 IEEE/ACM 2nd International Workshop on Software Architecture and Metrics* (pp. 15-21). IEEE.

Pressman, R. S. (2005). *Software engineering: a practitioner's approach*. Palgrave macmillan.

Rafferty, A. E., Jimmieson, N. L., and Armenakis, A. A. (2013). Change readiness: A multilevel review. *Journal of management*, *39*(1), 110-135.

Rasheed, A., Zafar, B., Shehryar, T., Aslam, N. A., Sajid, M., Ali, N., ... and Khalid, S. (2021). Requirement engineering challenges in agile software development. *Mathematical Problems in Engineering*, *2021*(1), 6696695.

Rauf, A., and AlGhafees, M. (2015, August). Gap analysis between state of practice and state of art practices in agile software development. In *2015 agile conference* (pp. 102-106). IEEE.

Rodríguez, P., Mäntylä, M., Oivo, M., Lwakatare, L. E., Seppänen, P., and Kuvaja, P. (2019). Advances in using agile and lean processes for software development. In *Advances in computers* (Vol. 113, pp. 135-224). Elsevier.

Rother, M., and Shook, J. (2003). *Learning to see: value stream mapping to add value and eliminate muda*. Lean enterprise institute.

Schach, S. R. (2007). *Object-oriented and classical software engineering* (Vol. 6). New York: McGraw-Hill.

Shylesh, S. (2017, April). A study of software development life cycle process models. In *National Conference on Reinventing Opportunities in Management, IT, and Social Sciences* (pp. 534-541).

Sinha, V., Sengupta, B., and Chandra, S. (2006). Enabling collaboration in distributed requirements management. *IEEE software*, *23*(5), 52-61.

Somerville, I. (2016). *Software engineering (10th ed.).* Pearson Education.

Srivastava, A., Bartol, K. M., and Locke, E. A. (2006). Empowering leadership in management teams: Effects on knowledge sharing, efficacy, and performance. *Academy of management journal*, *49*(6), 1239-1251.

Stake, R. E. (2010). Qualitative research: Studying how things work. Guilford Press

Staron, M., and Meding, W. (2011, June). Monitoring bottlenecks in agile and lean software development projects–a method and its industrial use. In *International Conference on Product Focused Software Process Improvement* (pp. 3-16). Berlin, Heidelberg: Springer Berlin Heidelberg.

Tankhiwale, S., and Saraf, S. (2020). Value stream mapping (vsm) led approach for waste and time to market reduction in software product development process. *Telecom Business Review*, *13*(1), 27.

Thummala, G. S. (2004). Value stream mapping for software development process.

Turetken, O., Stojanov, I., and Trienekens, J. J. (2017). Assessing the adoption level of scaled agile development: a maturity model for Scaled Agile Framework. *Journal of Software: Evolution and process*, *29*(6), e1796.

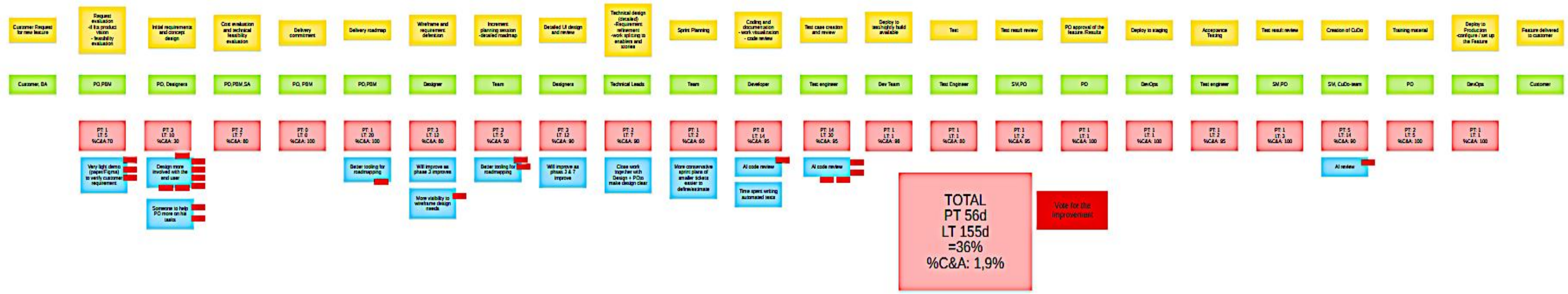
Tyagi, S., Choudhary, A., Cai, X., and Yang, K. (2015). Value stream mapping to reduce the lead-time of a product development process. *International journal of production economics*, *160*, 202-212.

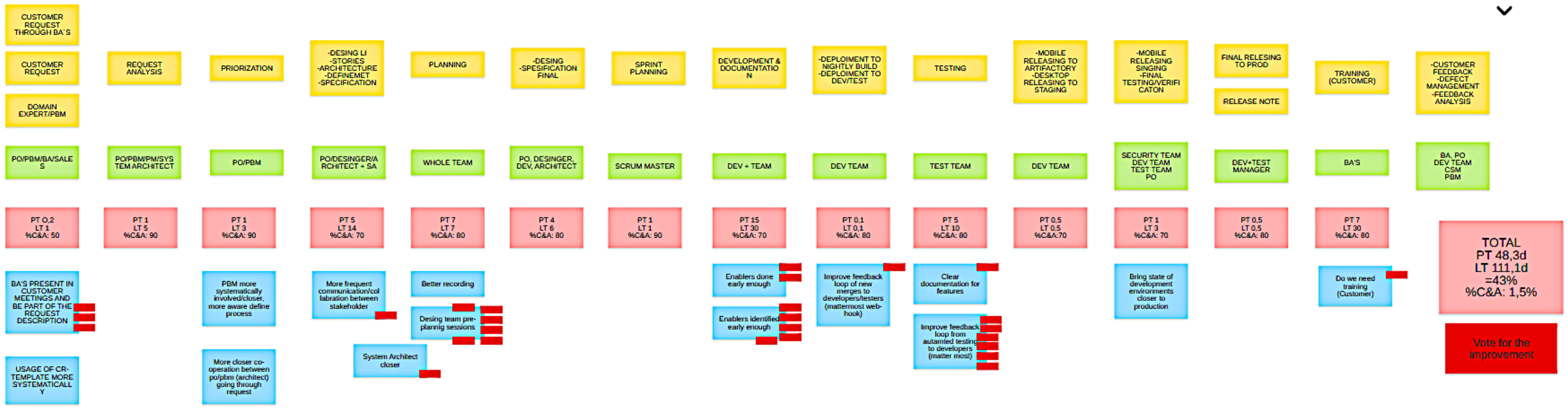
Weick, K. E., and Weick, K. E. (1995). *Sensemaking in organizations* (Vol. 3, pp. 1-231). Thousand Oaks, CA: Sage publications.

Yin, R. K. (2009). *Case study research: Design and methods* (Vol. 5). sage.

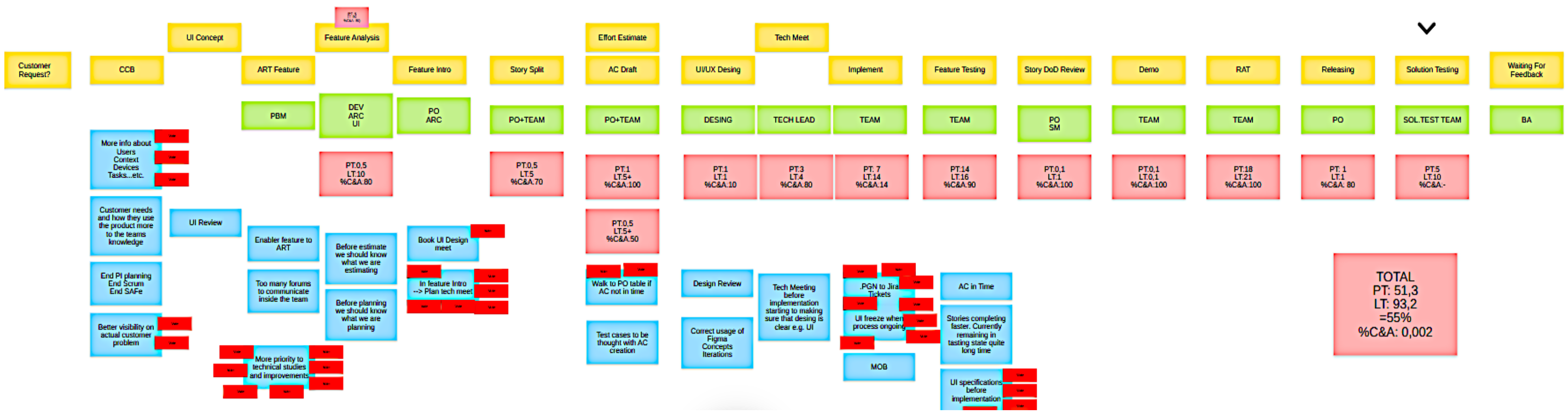
Zhang, X., and Bartol, K. M. (2010). Linking empowering leadership and employee creativity: The influence of psychological empowerment, intrinsic motivation, and creative process engagement. *Academy of management journal*, *53*(1), 107-128.

APPENDIX ONE TEAM 1 VALUE STREAM MAPS

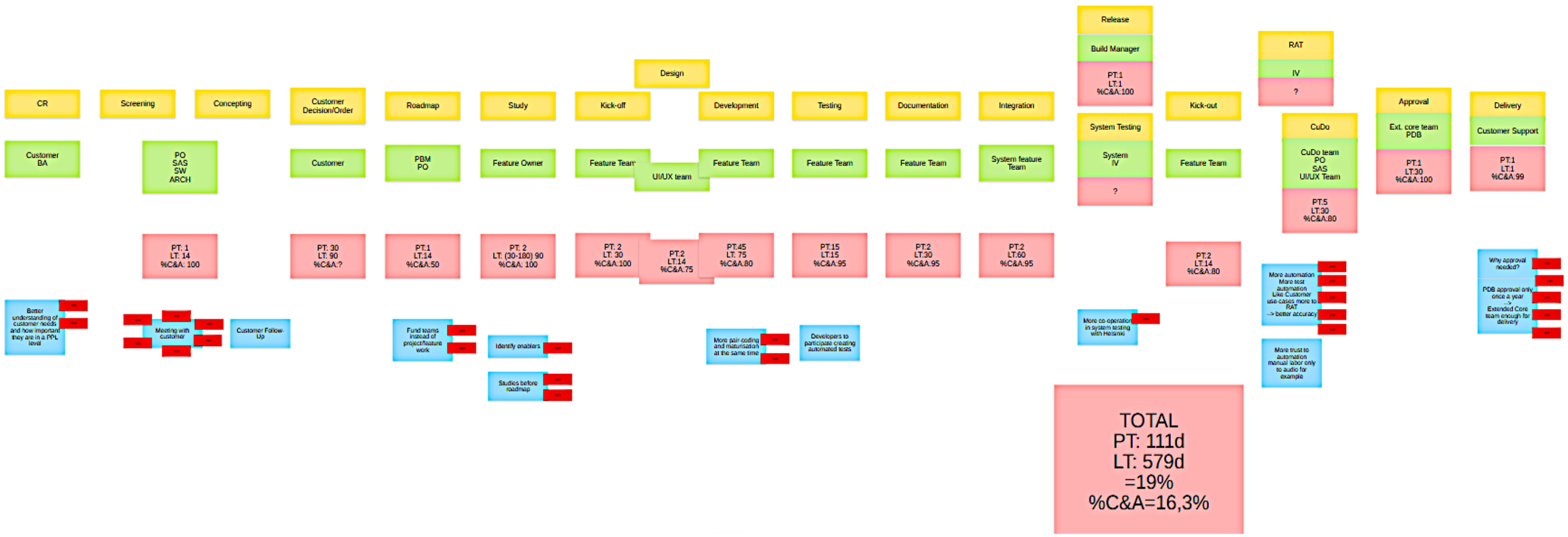

Customer Request for new feature
Customer, BA
Request evaluation -if its product vision - feasibility evaluation
PO,PBM
PT: 1 LT: 5 %C&A:70
Very light demo (paper/Figma) to verify customer requirement
Initial requirements and concept design
PO, Designers
PT: 3 LT: 10 %C&A: 30
Design more involved with the end user
Someone to help PO more on his tasks
Cost evaluation and technical feasibility evaluation
PO,PBM,SA
PT: 2 LT: 7 %C&A: 80
Delivery commitment
PO, PBM
PT: 0 LT: 0 %C&A: 100
Delivery roadmap
PO,PBM
PT: 1 LT: 20 %C&A: 100
Better tooling for roadmapping
Wireframe and requirement definition
Designer
PT: 3 LT: 12 %C&A: 80
Will improve as phase 3 improves
More visibility to wireframe design needs
Increment planning session -detailed roadmap
Team
PT: 3 LT: 5 %C&A: 50
Better tooling for roadmapping
Detailed UI design and review
Designers
PT: 3 LT: 12 %C&A: 90
Will improve as phase 3 & 7 improve
Technical design (detailed) -Requirement refinement -work splicing to enablers and stories
Technical Leads
PT: 2 LT: 7 %C&A: 90
Close work together with Design + POto make design clear
Sprint Planning
Team
PT: 1 LT: 2 %C&A: 60
More conservative sprint plans of smaller tickets easier to define/estimate
Coding and documentation - work visualization - code review
Developer
PT: 8 LT: 14 %C&A: 95
AI code review
Time spent writing automated tests
Test case creation and review
Test engineer
PT: 14 LT: 30 %C&A: 95
AI code review
Deploy to test/nightly build available
Dev Team
PT: 1 LT: 1 %C&A: 98
Test
Test Engineer
PT: 1 LT: 1 %C&A: 80
Test result review
SW,PO
PT: 1 LT: 2 %C&A: 95
PO approval of the feature /Results
PO
PT: 1 LT: 1 %C&A: 100
Deploy to staging
DevOps
PT: 1 LT: 1 %C&A: 100
Acceptance Testing
Test engineer
PT: 1 LT: 2 %C&A: 95
Test result review
SM,PO
PT: 1 LT: 3 %C&A: 100
Creation of CuDo
SW, CuDo-team
PT: 5 LT: 14 %C&A: 90
AI review
Training material
PO
PT: 2 LT: 5 %C&A: 100
Deploy to Production -configure / set up the Feature
DevOps
PT: 1 LT: 1 %C&A: 100
Feature delivered to customer
Customer
TOTAL PT 56d LT 155d =36% %C&A: 1,9%
Vote for the improvement


CUSTOMER REQUEST THROUGH BA´S
CUSTOMER REQUEST
DOMAIN EXPERT/PBM
PO/PBM/BA/SALES
PT O,2 LT 1 %C&A: 50
BA'S PRESENT IN CUSTOMER MEETINGS AND BE PART OF THE REQUEST DESCRIPTION
USAGE OF CR-TEMPLATE MORE SYSTEMATICALLY
REQUEST ANALYSIS
PO/PBM/PM/SYSTEM ARCHITECT
PT 1 LT 5 %C&A: 90
PRIORIZATION
PO/PBM
PT 1 LT 3 %C&A: 90
PBM more systematically involved/closer, more aware define process
More closer co-operation between po/pbm (architect) going through request
-DESING LI -STORIES -ARCHITECTURE -DEFINEMET -SPECIFICATION
PO/DESINGER/ARCHITECT + SA
PT 5 LT 14 %C&A: 70
More frequent communication/collabration between stakeholder
System Architect closer
PLANNING
WHOLE TEAM
PT 7 LT 7 %C&A: 80
Better recording
Desing team pre-plannig sessions
-DESING -SPESIFICATION FINAL
PO, DESINGER, DEV, ARCHITECT
PT 4 LT 6 %C&A: 80
SPRINT PLANNING
SCRUM MASTER
PT 1 LT 1 %C&A: 90
DEVELOPMENT & DOCUMENTATION
DEV + TEAM
PT 15 LT 30 %C&A: 70
Enablers done early enough
Enablers identified early enough
-DEPLOIMENT TO NIGHTLY BUILD -DEPLOIMENT TO DEV/TEST
DEV TEAM
PT 0,1 LT 0,1 %C&A: 80
Improve feedback loop of new merges to developers/testers (mattermost web-hook)
TESTING
TEST TEAM
PT 5 LT 10 %C&A: 80
Clear documentation for features
Improve feedback loop from autamted testing to developers (matter most)
-MOBILE RELEASING TO ARTIFACTORY -DESKTOP RELEASING TO STAGING
DEV TEAM
PT 0,5 LT 0,5 %C&A:70
-MOBILE RELEASING SINGING -FINAL TESTING/VERIFICATON
SECURITY TEAM DEV TEAM TEST TEAM PO
PT 1 LT 3 %C&A: 70
Bring state of development environments closer to production
FINAL RELESING TO PROD
RELEASE NOTE
DEV+TEST MANAGER
PT 0,5 LT 0,5 %C&A: 80
TRAINING (CUSTOMER)
BA'S
PT 7 LT 30 %C&A: 80
Do we need training (Customer)
-CUSTOMER FEEDBACK -DEFECT MANAGEMENT -FEEDBACK ANALYSIS
BA, PO DEV TEAM CSM PBM
TOTAL PT 48,3d LT 111,1d =43% %C&A: 1,5%
Vote for the improvement

APPENDIX TWO TEAM 2 VALUE STREAM MAP


Customer Request?
CCB
UI Concept
ART Feature
Feature Analysis
Feature Intro
Story Split
Effort Estimate
AC Draft
UI/UX Desing
Tech Meet
Implement
Feature Testing
Story DoD Review
Demo
RAT
Releasing
Solution Testing
Waiting For Feedback
PBM
DEV ARC UI
PO ARC
PO+TEAM
PO+TEAM
DESING
TECH LEAD
TEAM
TEAM
PO SM
TEAM
TEAM
PO
SOL.TEST TEAM
BA
PT.0,5 LT.10 %C&A.80
PT.0,5 LT.5 %C&A.70
PT.1 LT.5+ %C&A.100
PT.0,5 LT.5+ %C&A.50
PT.1 LT.1 %C&A.10
PT.3 LT.4 %C&A.80
PT. 7 LT.14 %C&A.14
PT.14 LT.16 %C&A.90
PT.0,1 LT.1 %C&A.100
PT.0,1 LT.0,1 %C&A.100
PT.18 LT.21 %C&A.100
PT. 1 LT.1 %C&A. 80
PT.5 LT.10 %C&A.-
More info about Users Context Devices Tasks...etc.
Customer needs and how they use the product more to the teams knowledge
End PI planning End Scrum End SAFe
Better visibility on actual customer problem
UI Review
Enabler feature to ART
Too many forums to communicate inside the team
More priority to technical studies and improvements
Before estimate we should know what we are estimating
Before planning we should know what we are planning
Book UI Design meet
In feature Intro --> Plan tech meet
Walk to PO table if AC not in time
Test cases to be thought with AC creation
Design Review
Correct usage of Figma Concepts Iterations
Tech Meeting before implementation starting to making sure that desing is clear e.g. UI
.PGN to Jira Tickets
UI freeze when process ongoing
MOB
AC in Time
Stories completing faster. Currently remaining in tasting state quite long time
UI specifications before implementation
TOTAL
PT: 51,3
LT: 93,2
=55%
%C&A: 0,002

APPENDIX THREE TEAM 3 VALUE STREAM MAP


CR
Customer BA
Better understanding of customer needs and how important they are in a PPL level
Screening
PO SAS SW ARCH
PT: 1 LT: 14 %C&A: 100
Meeting with customer
Concepting
Customer Follow-Up
Customer Decision/Order
Customer
PT: 30 LT: 90 %C&A:?
Roadmap
PBM PO
PT:1 LT:14 %C&A:50
Fund teams instead of project/feature work
Study
Feature Owner
PT: 2 LT: (30-180) 90 %C&A: 100
Identify enablers
Studies before roadmap
Kick-off
Feature Team
PT: 2 LT: 30 %C&A:100
Design
UI/UX team
PT:2 LT:14 %C&A:75
Development
Feature Team
PT:45 LT: 75 %C&A:80
More pair-coding and maturisation at the same time
Testing
Feature Team
PT:15 LT:15 %C&A:95
Developers to participate creating automated tests
Documentation
Feature Team
PT:2 LT:30 %C&A:95
Integration
System feature Team
PT:2 LT:60 %C&A:95
Release
Build Manager
PT:1 LT:1 %C&A:100
System Testing
System IV
?
More co-operation in system testing with Helsinki
Kick-out
Feature Team
PT:2 LT:14 %C&A:80
RAT
IV
?
CuDo
CuDo team PO SAS UI/UX Team
PT:5 LT:30 %C&A:80
More automation More test automation Like Customer use-cases more to RAT --> better accuracy
More trust to automation manual labor only to audio for example
Approval
Ext. core team PDB
PT:1 LT:30 %C&A:100
Delivery
Customer Support
PT:1 LT:1 %C&A:99
Why approval needed?
PDB approval only once a year --> Extended Core team enough for delivery
TOTAL
PT: 111d
LT: 579d
=19%
%C&A=16,3%