Hochschule
Kaiserslautern
University of
Applied Sciences

MASTER'S THESIS

Title:
An Agile Collaboration Framework for Workstream-based Software Development

Study Program:
Intelligent Enterprise Management, PO 2020

Submitted by:
Mathias Kemeter
Brennereiweg 2
76726 Germersheim
Matriculation ID: 886273

Supervisor & First Reviewer:
Prof. Dr. Christine Arend-Fuchs

Second Reviewer:
Prof. Dr. Michael Jacob

Submission Date:
17 July 2024

FACULTY OF BUSINESS STUDIES

Table of Contents

Table of Figures

1. Introduction

The number of software developers globally is expected to reach almost 29 million people by the end of 2024.[1] Due to modern version control systems and remote collaboration software, these knowledge workers can operate with a high degree of autonomy. Software development methodologies are used by teams in software companies to ensure, despite the individual autonomy, that the contribution of each developer is valuable to the sold product and, from a management perspective, results in an adequate return on investment.

Agile software development methodologies have recently become the predominant working model, as the associated process frameworks promise a good balance between maintaining the knowledge workers' autonomy and efficiently reaching commercial goals. As methodological requirements differ from team to team and company to company, businesses have a broad choice of agile frameworks to fulfill their requirements. Instead of implementing existing frameworks, teams and companies may choose to craft a custom process model for addressing their specific needs.

This thesis introduces an agile development framework that addresses the particular needs of software development teams within large organizations. These teams are often responsible for a wide variety of topics while working within tight margin expectations.

1.1 Initial Situation and Problem Statement

Agile software development teams in large organizations face challenges directly or indirectly caused by margin and efficiency expectations, which create tension with the desired agility, and existing agile frameworks in particular. Therefore, the focus of this document is on software development teams and organizations that, in addition to customer expectations of their products, are affected by the following conditions:

[1] (Statista, 2024)

- *Profitability expectations* drive the expansion of development teams due to reduced management layers and the diversification of development topics.
- A *competitive job market* increases attrition and makes it difficult to maintain stable teams with balanced staffing according to business priorities.
- *Employees in the New Work era*[2] demand a high degree of autonomy and expect companies to be people-centric.

This master's thesis discusses the problem of how agile software development teams can integrate and tackle the above-mentioned challenges into their development process while ensuring high work throughput and keeping developer satisfaction at a high level.

1.2 Objectives and Approach

Based on the principles of agile software development and using elements of existing agile frameworks, this thesis proposes an agile, lean development framework that addresses the problem statement presented in Section 1.1.

To ensure developer satisfaction and autonomy, this framework has been iteratively crafted by a 20-strong development team using previous experience and elements of Design Thinking.[3] The resulting framework addresses the following core objectives:

- Implement a *unified process* for large teams to improve management reporting and efficiency.
- Enable the team to be responsible for an *increasing variety of development topics*.
- Keep the *staffing balanced* across teams while avoiding disruptive outplacement (so-called "lift and shift") of developers.
- Make best use of available knowledge and resources by *leveraging synergies* between teams.

[2] see Section 2.4
[3] see (Dam & Siang, 2024)

- Increase *developer autonomy and satisfaction* according to New Work principles.

In addition, this thesis discusses the success of the new framework based on feedback from the development team after running the framework for more than six months, which will allow a realistic portrayal of the approach while fostering transparency regarding its limitations and inherent weaknesses.

1.3 Structure of the Thesis

Chapter 2 introduces the fundamentals of agile and lean methodologies in Section 2.1 before summarizing the relevant existing agile frameworks in Section 2.2. The chapter concludes with a critical perspective on these frameworks and a consideration of the principles of New Work in the context of agile software development.

Based on the fundamentals presented in Chapter 2, Chapter 3 proposes a framework for workstream-based software development. After highlighting the necessity for this proposal in Section 3.1, the structures and processes related to it are listed in Section 3.2, along with additional best practices and recommendations in Section 3.3. The chapter concludes with a summary of the core proposal.

In reverse chronological order, Chapter 4 presents the journey from the initial situation to the proposal outlined in Chapter 3. As the proposed framework has been conceptualized and prototyped simultaneously with an iterative approach, Chapter 4 covers the design phase of the framework in Section 4.1, the change management and people transition aspects in Section 4.2, the implementation support offered by collaboration technology in Section 4.3, and the evaluation of outcomes after running the proposed framework for more than six months in Section 4.4.

Chapter 5 discusses how the new development framework compares to and differentiates from existing frameworks. The second half of the chapter highlights the study's limitations and outlines the potential for future research and improvement.

Chapter 6 concludes this thesis by briefly summarizing its main outcomes and discussion points and transferring the major findings into a Business Model Canvas.

2. Fundamentals of Agile Frameworks

This chapter provides a fundamental overview of agile frameworks for software development. In the last two decades, software development teams have moved away from traditional waterfall methods to frameworks such as Scrum or Kanban, with a steep increase in adoption since 2009, as depicted in Figure 1. Based on a survey involving 601 development and IT professionals conducted by Hewlett Packard, agile methodologies in software development can be considered the "new norm."[4]

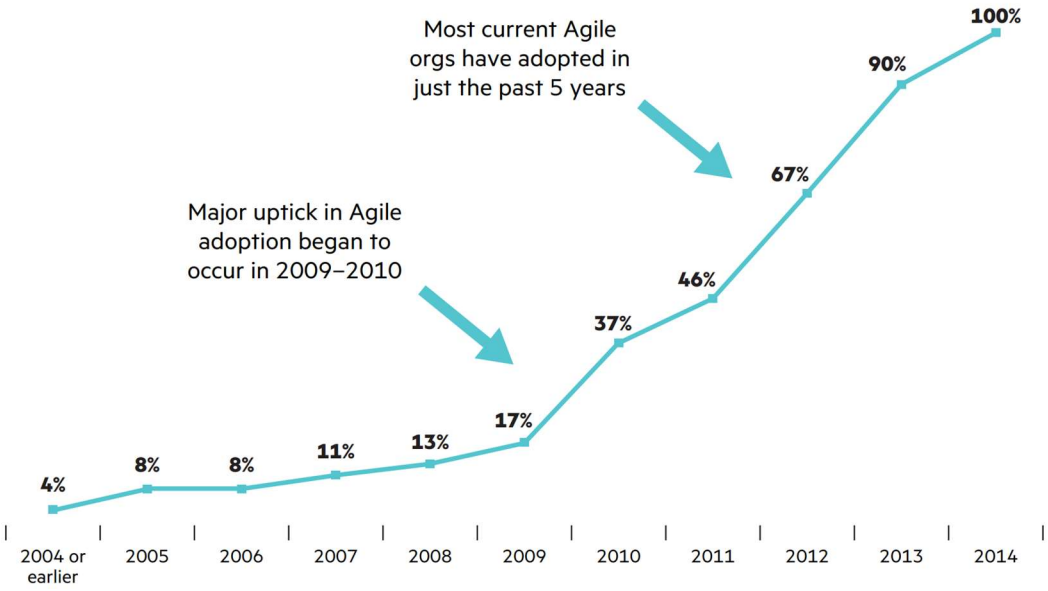

Figure 1: Agile adoption over time (n=475 firms with some adoption of agile methods in 2014)[5]

The main concepts of agile methods are presented in Sections 2.1 and 2.2. A brief practical evaluation of the discussed approaches, also from the perspective of the New Work movement, concludes the chapter in Section 2.3 and 2.4. Although the methods outlined here are widely adopted in the software development industry, most are based on observations and experience rather than scientific studies. This

[4] see (Hewlett Packard Enterprise, 2017, p. 1)
[5] taken from (Hewlett Packard Enterprise, 2017, p. 2)

point also applies to the most popular framework, Scrum, which is "founded on empiricism and lean thinking,"[6] as the official Scrum Guide states.

2.1 Introduction to Agile and Lean

Agile and *Lean*, as in *Agile Thinking* or *Lean Management*, are often referred to when discussing agile practices for software development. While there is no precise definition of the terms, some underlying principles are commonly implied when using them.

2.1.1 Principles of Lean Thinking

The core idea of *Lean Thinking* (or *Lean*) is continuously eliminating waste,[7] which, in the context of software development and beyond, implies activities that do not add value to the product. The concepts applied to software development today were based initially on lean manufacturing and the associated one-piece flow, which originated in Toyota's production system in the 1950s and 1960s.[8]

In 1996, Womack and Jones defined Lean Thinking based on five principles, which are often referred to in subsequent literature:

> "*Lean Thinking is the antidote to waste. There are (5) Lean Principles:*
>
> - ***Specify Value.*** *Value can be defined only by the ultimate customer. [...]*
> - ***Identify the Value Stream.*** *The Value Stream is all the actions needed to bring a product to the customer. […]*
> - ***Flow.*** *Make the value-creating steps flow. [...]*
> - ***Pull****. Let the customer pull the product from you. Sell one. Make one.*
> - ***Pursue Perfection.*** *There is no end to the process of reducing time, space, cost, and mistakes.*

[6] (Schwaber & Sutherland, 2020, p. 4)
[7] also known as "Kaizen"
[8] see (Liker, 2003)

> *Lean is doing more with less. Use the least amount of effort, energy, equipment, time, facility space, materials, and capital—while giving customers exactly what they want.*"[9]

Starting from its definition as a set of principles, it becomes evident that Lean Thinking is not a process that can be standardized and replicated.[10] Instead, it can be seen as a foundational mindset facilitated by certain *lean practices*.

From a software development perspective, continuous integration and continuous delivery (CI/CD) are preconditions to applying lean principles on an organizational level as they stabilize flow and help the organization move from large-batch development toward smaller increments, like a one-piece flow in manufacturing.[11]

Achieving the state of continuous delivery requires "a culture in which interactions between development, operations, and information security teams are generally a win-win."[12] This culture is considered a generative company culture according to sociologist Ron Westrum's three cultures model,[13] which is depicted in Figure 2.

Pathological (power-oriented)	Bureaucratic (rule-oriented)	Generative (performance-oriented)
Low cooperation	Modest cooperation	High cooperation
Messengers shot	Messengers neglected	Messengers trained
Responsibilities shirked	Narrow responsibilities	Risks are shared
Bridging discouraged	Bridging tolerated	Bridging encouraged
Failure leads to scapegoating	Failure leads to justice	Failure leads to inquiry
Novelty crushed	Novelty leads to problems	Novelty implemented

Figure 2: Westrum's three cultures model and associated characteristics for organizations[14]

In conclusion, the technical possibility of CI/CD and a generative organizational culture are catalysts for establishing the 5 Lean Thinking principles within a software company.

[9] (Womack & Jones, 2003)
[10] see (Ladas, 2008, p. 13)
[11] see (Humble, et al., 2020, p. 168)
[12] (Humble, et al., 2020, p. 168)
[13] see (Westrum, 2005, pp. 22-24)
[14] own representation based on (Humble, et al., 2020, p. 10)

2.1.2 Manifesto for Agile Software Development

Traditionally, software development processes followed the so-called Waterfall Methodology, or the closely related V-model. Figure 3 shows the approach of detailing software requirements during a dedicated verification phase. After concluding verification, the software is coded as specified and subsequently tested during the validation phase.

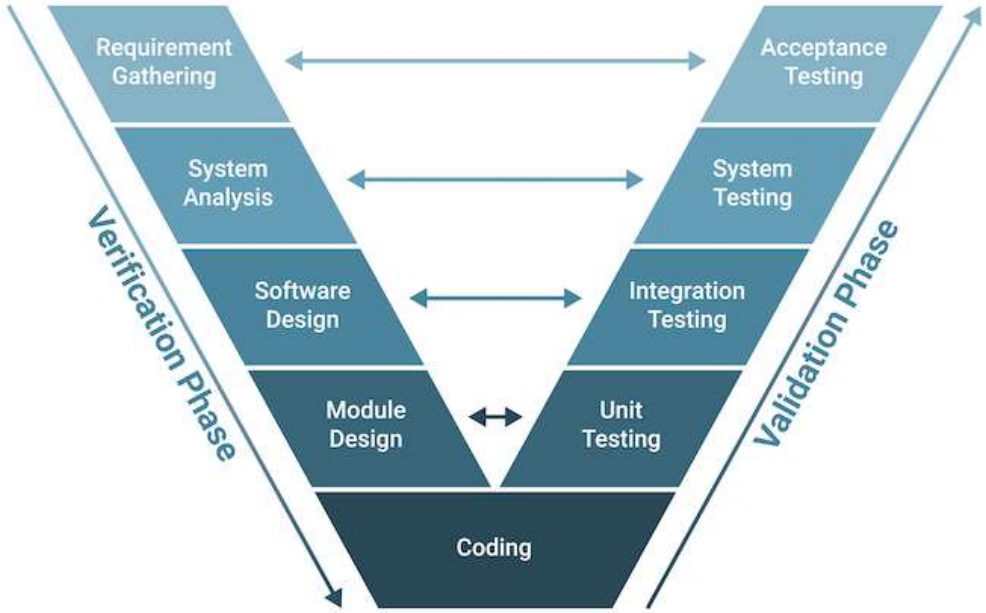

Figure 3: The traditional V-model of software development[15]

Each step in the traditional model needs to be finalized before linearly starting with the next step so that there is only a limited feedback loop and missing flexibility to react to changing requirements. Traditional models are still practiced in highly regulated domains, such as health care, as they are deemed beneficial for controlling risk and quality.

[15] taken from (Oppermann, 2023)

The major disadvantage of the traditional models is their lack of agility to react to changing conditions in a VUCA[16] environment and the closely related risk of delivering a product without relevance and value. This situation has led to an accelerated movement toward agile software development, a paradigm that embraces change and uncertainty in the software development process. Agile methods aim to deliver value to customers faster and more frequently while ensuring high quality and continuous improvement.

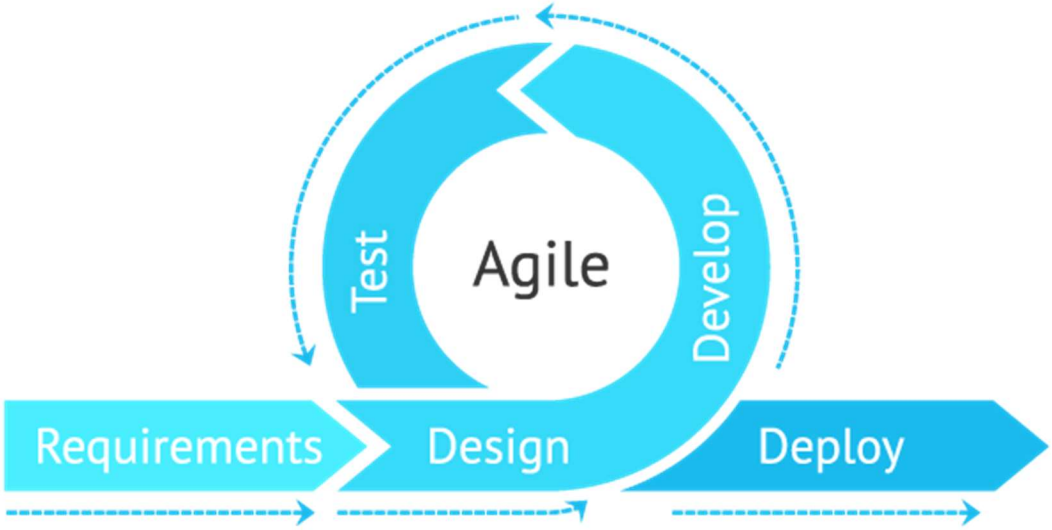

Figure 4: Iterative approach to agile software development[17]

After agile methodologies gained popularity in the 1990s, a group of agile practitioners agreed in 2001 on a manifesto, which today is the foundation for agile software development and related frameworks:

> "*We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
>
> - ***Individuals and interactions*** *over processes and tools*
> - ***Working software*** *over comprehensive documentation*
> - ***Customer collaboration*** *over contract negotiation*
> - ***Responding to change*** *over following a plan*

[16] VUCA = volatile, uncertain, complex, and ambiguous conditions
[17] taken from (Digital Template Market, 2020)

> *That is, while there is value in the items on the right, we value the items on the left more.*"[18]

There are many parallels between the Agile Manifesto and lean approaches, which explains their frequent co-existence in modern software development. The primary parallels include the following:

- Steady flow and small batch sizes are logical prerequisites for being able to respond to change.
- The target to create value with working software aligns with the lean principle of measuring value only based on the customer.
- Both paradigms put individuals at their core. While this is an explicit part of the Agile Manifesto, it is also implied by the required organizational culture for implementing Lean Thinking.

Software development teams may work in a lean and agile manner simultaneously, and popular development frameworks such as Scrum or SAFe have been designed with both aspects in mind. This combination is sometimes referred to as *lean-agile mindset*.[19] However, it is worth noting that both paradigms have slightly different views on the same process: Lean focuses on doing more with less, which usually appeals to a management view, while Agile focuses on delivering customer satisfaction, which usually appeals to developers.

2.1.3 Manifesto for Software Craftsmanship

Software Craftsmanship is a movement that focuses more on the developer's view of the software development process und builds on the agile paradigm. In 2008, Robert C. Martin,[20] signatory of the Agile Manifesto, proposed adding a fifth value to the manifesto: *"Craftsmanship over Execution"*[21]

[18] (Beck, et al., 2001)
[19] (Scaled Agile, Inc., 2024)
[20] widely known as "Uncle Bob" in the software development community
[21] see (Bria, 2008)

Martin intended to highlight the simultaneous importance of delivering the expected outcome (eventually working code), adhering to good craftsmanship, and delivering the outcome with an appropriate inner quality. In an earlier and more dramatic version, he proposed the value of *"Craftsmanship over Crap"*, which made his intent even more explicit.[22]

Following Martin's and others' initiatives, a group of software professionals created the Manifesto for Software Craftsmanship, which currently has more than 35,000 signatories. The declaration includes the following:

> "*As aspiring Software Craftsmen we are raising the bar of professional software development by practicing it and helping others learn the craft. Through this work we have come to value:*
>
> - *Not only working software, but also* ***well-crafted software***
> - *Not only responding to change, but also* ***steadily adding value***
> - *Not only individuals and interactions, but also* ***a community of professionals***
> - *Not only customer collaboration, but also* ***productive partnerships***
>
> *That is, in pursuit of the items on the left we have found the items on the right to be indispensable.*"[23]

The Software Craftsmanship Manifesto has a clear connection to the Agile Manifesto while enhancing and challenging it. The direct comparison of values in Figure 5 makes the emphasis on quality and craftsmanship evident.

[22] see (Bria, 2008)
[23] (the undersigned, 2009). This statement may be freely copied in any form, but only in its entirety through this notice.

Software Craftsmanship	Agile
Well-crafted software	Working software
Steadily adding value	Responding to change
Community of professionals	Individuals and interactions
Productive partnership	Customer collaboration

Figure 5: Values of Software Craftmanship and Agile compared

Some of the practices of Software Craftsmanship are exclusive to this concept. Building on the comparison of software development as a craft rather than an engineering discipline, practices such as the *Craftsman Swaps* have been established among practitioners.

The Craftsman Swaps are inspired by the European late-medieval practice of Journeyman Tours,[24] where craftsman apprentices would work and travel across the nation to gain practical experience and develop new skills inspired by foreign places and work practices. The Software Craftsmanship movement transferred this concept to software development by establishing time-boxed employee exchanges between two companies: Two developers would change their jobs for a particular time to gain new experience and transfer knowledge and improvement opportunities to their original company.[25] The practical example of the companies *8th Light* and *Obtiva*, which have been early adopters of the practice, shows that given the right contractual boundary, the swaps are even possible in a competitive situation.[26]

Some elements of Software Craftmanship conflict with lean principles—for example, activities such as the removal of technical debt are encouraged by the values of Software Craftsmanship but could be considered wasteful from a lean perspective. However, the idea of Software Craftsmanship can be applied to the frameworks introduced in Section 2.2.[27]

[24] also denoted by the German term “Wanderjahre” or “Walz”
[25] see (Rauch, 2016)
[26] see (Wong, 2009)
[27] see (Lucena & Tizzei, 2016, p. 9)

2.2 Overview of Agile Frameworks

This section provides an overview of commonly used frameworks and methodologies, namely Scrum and Kanban, to introduce agile principles to software development teams. After presenting the basics of Scrum and Kanban, existing approaches for scaling agile methodologies on an organizational level are briefly described.

2.2.1 Scrum

The term *scrum* is borrowed from rugby football, where it describes a situation in which the players gather in a dense pack to restart the game after an incident.[28] The analogy with team sports highlights the ambition of the Scrum framework: Software development is team sports, and the team needs to align tightly and challenge each other to get the ball back into the game and achieve meaningful results.

Figure 6: Rugby players in a scrummage (also known as scrum)[29]

Looking at the *Scrum framework*[30] in a business context, it commonly gets described as “a lightweight framework that helps people, teams and organizations generate value through adaptive solutions for complex problems.”[31] The goal is to

[28] (World Rugby Limited, 2024)
[29] taken from (Quino Al, 2017)
[30] abbreviated: Scrum
[31] (Schwaber & Sutherland, 2020, p. 3)

"optimize predictability and to control risk"[32] within software development. Although Scrum is not restricted to the domain of software development, it is most frequently used in this context. The fundamentals of Scrum were initially described in a short document by Ken Schwaber and Jeff Sutherland,[33] which serves as a single source of truth for ample secondary literature on how to adopt the framework in real-world software development. Thus, Scrum is deemed easy to adopt but hard to master.

Figure 7 provides an overview of Scrum's most important roles, events, and artifacts, which will be explained in the following subsections.

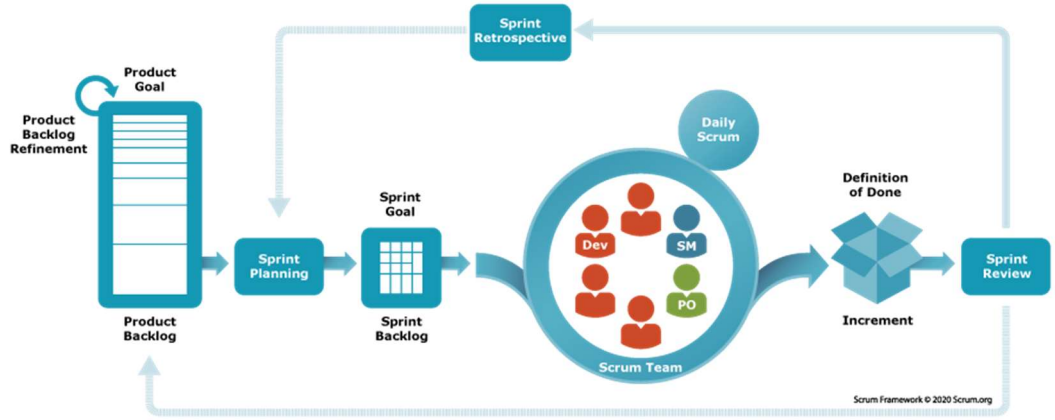

Figure 7: Scrum roles, events, and artifacts[34]

2.2.1.1 Roles

A *Scrum team* is a non-hierarchical team of empowered individuals. Ideally, the team should consist of ten people or less[35] and be split into three personas: Developers, Product Owner, and Scrum Master.

Developers are professionals and craftsmen, who create the product. They plan and coordinate sprint activities so that a defined increment is delivered at the end of the sprint according to previously defined quality and functional criteria.[36]

[32] (Schwaber & Sutherland, 2020, p. 3)
[33] (Schwaber & Sutherland, 2020)
[34] taken from (Scrum.org, 2024)
[35] see (Schwaber & Sutherland, 2020, p. 5)
[36] see (Schwaber & Sutherland, 2020, p. 5)

The *Product Owner* is accountable for delivering a product with maximum value to its customers and stakeholders. To achieve this, the Product Owner manages the product backlog so that the prioritization of product deliverables is in harmony with the product goal. The Product Owner ensures the team members understand the product backlog and its corresponding goal. This role is the single point of contact for stakeholders and is responsible for mapping their technical and non-technical requirements to the product backlog.[37]

The most significant role, the *Scrum Master*, is typically described as a servant leader who maximizes the team's effectiveness by removing obstacles that may prevent the team from reaching their goal. This role also ensures that every individual understands and follows the elements of Scrum.[38]

2.2.1.2 Events

Scrum is organized in cycles called *sprints*—another sports analogy. Sprints have a fixed duration of between one and four weeks, and the next Sprint starts immediately after the previous one has finished.[39]

Each sprint has a *sprint goal*, which the team (including the Product Owner) has agreed upon during the sprint planning. The *sprint planning* event is conducted before each sprint and is intended to define the work packages that need to be done to achieve the sprint goal.[40]

During the sprint, the team (excluding the Product Owner) will meet in *Daily Scrum* meetings, which should not exceed 15 minutes. The goal is to ensure the team is still working towards the sprint goal. The team may adjust planning or remove obstacles to ensure coherence with the sprint goal.[41]

At the closing of each sprint, and before planning the next sprint, the team conducts a sprint review and sprint retrospective. The *sprint review* is the Scrum event with the largest audience, as it may also include customers and stakeholders. The team

[37] see (Schwaber & Sutherland, 2020, pp. 5-6)
[38] see (Schwaber & Sutherland, 2020, pp. 6-7)
[39] see (Schwaber & Sutherland, 2020, p. 7)
[40] see (Schwaber & Sutherland, 2020, p. 8)
[41] see (Schwaber & Sutherland, 2020, p. 9)

presents the outcome of the recent sprint to this audience and gathers their feedback, which may be fed back to the backlog by the Product Owner. On the contrary, the *sprint retrospective* is the most sensitive of all Scrum events. The team uses this meeting to discuss what went well during the last sprint and where there is potential for improvement. This includes technical as well as individual and collaborative aspects.[42]

2.2.1.3 Artifacts

The Scrum Guide describes three artifacts that ensure transparency over the planned work and the delivered value of the development team: the product backlog, the spring backlog, and the increment. Each artifact is bound to a commitment, which sets the scale for measuring progress.

The *product backlog* is a sorted list of work items, also called *backlog items* or *user stories*, that are required to reach the *product goal*. The latter is a commitment agreed upon within the Scrum team. The product backlog is a living document that undergoes a continuous refinement process.[43]

During the sprint planning, the Scrum team agrees on a *sprint goal* and assembles the *sprint backlog* with the work items required to reach this goal. At this stage, each selected backlog item needs to be in an actionable state. It needs to contain a sufficient level of detail and knowledge to be delivered within the next sprint, and the team needs to agree on an effort estimation for the work to be done.[44]

As soon as a backlog item meets the *definition of done*, it is considered an *increment*. An increment must be usable—that is, it must represent a ready-to-consume feature rather than work effort within the sprint. The measurable output of a sprint is the sum of the delivered increments.[45]

[42] see (Schwaber & Sutherland, 2020, pp. 9-10)
[43] see (Schwaber & Sutherland, 2020, pp. 10-11)
[44] see (Schwaber & Sutherland, 2020, p. 11)
[45] see (Schwaber & Sutherland, 2020, pp. 11-12)

2.2.2 Kanban

The Kanban approach in software development originated in lean production systems such as the Toyota Production System (TPS).[46] While TPS utilizes a lean pull system[47] associated with the term kanban (small k), the denomination Kanban or Kanban Method (capital K) usually refers to the methodology and its adaption specific to software development processes.[48]

David J. Anderson defines Kanban as "the evolutionary change method that utilizes a kanban […] pull system, visualization, and other tools to catalyze the introduction of lean ideas into technology development and IT operations."[49] Starting from this definition, Kanban, at its core, is a lean pull system suitable for software development to achieve a continuous pace of work and be able to implement (process) changes with minimal resistance.[50]

There are several principles associated with Kanban. Most literature refers to these five principles for defining the core of the Kanban method:[51]

- Visualize work
- Limit work-in-progress
- Make policies explicit
- Measure and manage flow
- Identify improvement opportunities

Visualizing work helps to identify wasteful activities and to obtain transparency of the current process and its bottlenecks. Value stream maps (VSM) and Kanban boards are commonly used visualization tools. The VSM originated from lean manufacturing and is used to visualize the process stages with its average lead and processing times. Typical stages for software development are *specification*, *implementation*, *testing*, and others. As the primary operative visualization, the Kanban board makes the work that flows through the process visible alongside the

[46] see (Liker, 2003)
[47] see Subsection 2.1.1
[48] see (Anderson, 2010, p. 6)
[49] (Anderson, 2010, p. 6)
[50] see (Anderson, 2010, pp. 2-5)
[51] see (Boeg, 2012, p. 13), (Anderson, 2010, p. 16)

agreed principles, as shown in Figure 8. In line with the VSM, the Kanban board will typically show activity stages where actual work is performed, as well as buffer stages where work items wait without adding value or information to the product.[52]

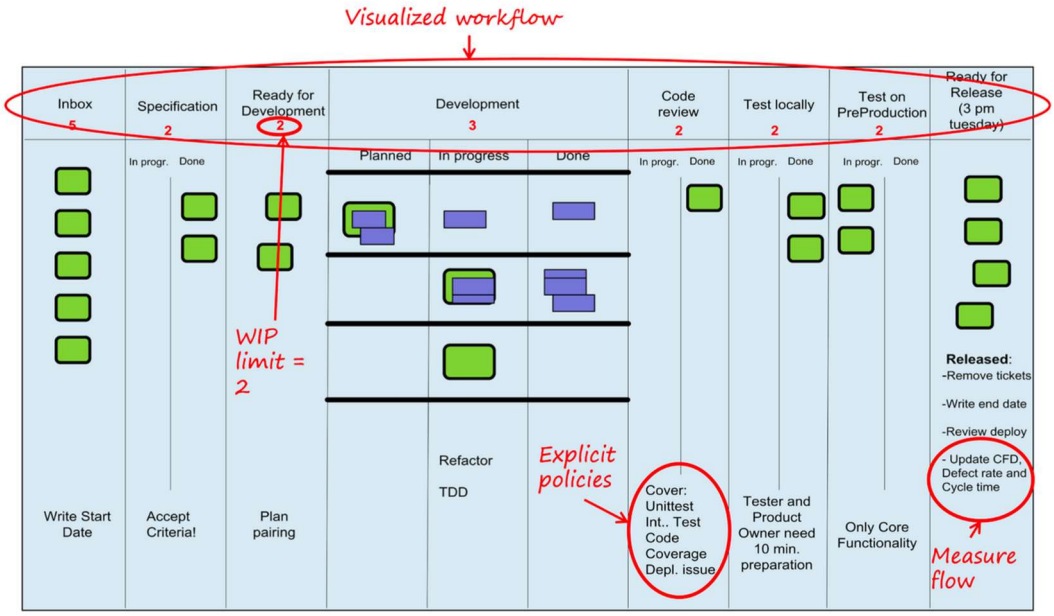

Figure 8: Visualization of work items and the main Kanban principles in a Kanban board[53]

Limiting the work-in-progress (WIP) is a tool that maximizes the throughput of a Kanban system according to Little's law.[54] WIP limits are defined for each process stage. For activity stages, the optimal WIP limit is dependent on the number of developers, whereas, for buffer stages, the WIP limits are defined independently. According to the practice of continuous improvement, the process starts with best-guess limits, which are adjusted over time as the team gathers experience. In addition to maximizing flow, WIP limits make bottlenecks transparent and enforce team discussions, as new work can only be pulled once the current work has been done.[55]

[52] see (Boeg, 2012, pp. 21-25)
[53] taken from (Boeg, 2012, p. 15)
[54] Cycle time = WIP/Throughput per Unit of Time
[55] see (Boeg, 2012, pp. 27-32)

Making policies explicit and agreeing that any change to a policy is a team decision is a cornerstone for proper quality assurance. Every team member commits to adhering to these governance rules. If it appears necessary to break a policy, this is understood as a trigger for a team discussion on improving existing policies based on recent experience. Policies need to be visualized and enforced to prevent the degeneration of the process.[56]

Measuring and managing the flow of work come as a pair, which is in line with the common business principle that one can only manage what gets measured. The goal of managing the flow of work is to increase the business value of work and decrease cycle times of work items by achieving a steady and sustainable flow, where tasks move through the process without wasteful delays and bottlenecks. In this context, being sustainable means creating and managing a stable and predictable flow. Jesper Boeg suggests considering these measurements to enable proper flow management:[57]

- A *cumulative flow diagram (CFD)* shows the current amount of work in the system and makes WIP, velocity, and other aspects visible.
- *Cycle time* measures the average time it takes a work item to move from start to delivery. In addition, the variation of cycle times shall also be determined to measure the predictability of the process.
- *Defect rate* tracks the number of open and newly created bugs to allow conclusions on the current state of quality.
- *Number of blocked items* identifies and emphasizes issues that prevent a steady and optimized flow of work.

According to the principle of visualization, the measures taken need to be transparently visualized and shared with the team. Figure 9 shows examples for these diagrams.

[56] (Boeg, 2012, pp. 35-37)
[57] (Boeg, 2012, p. 44)

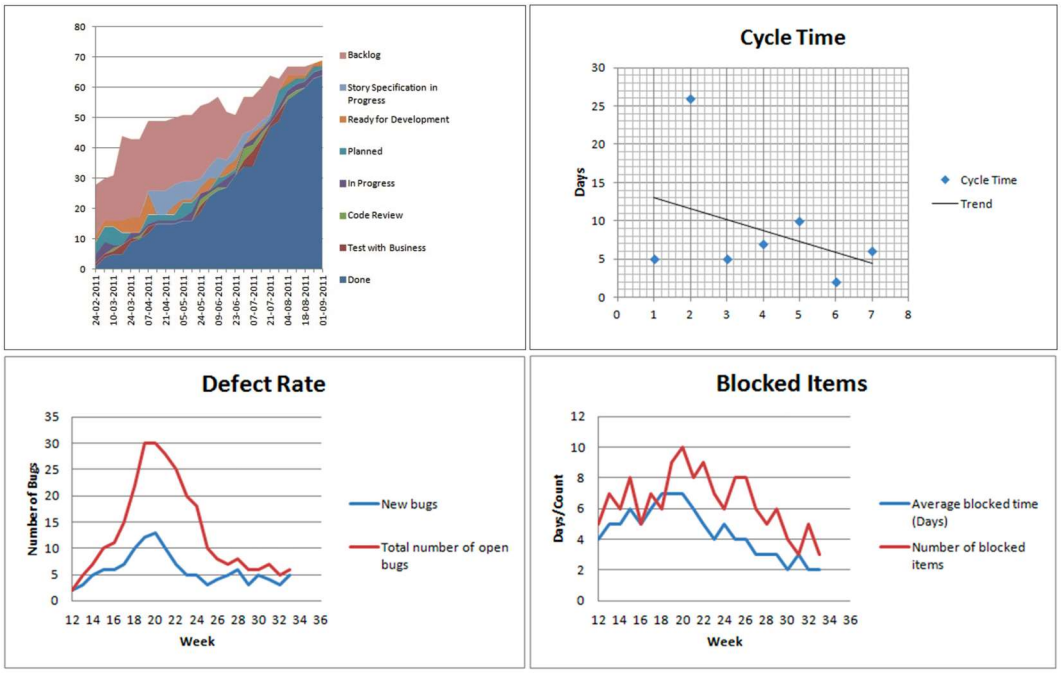

Figure 9: Visualizing measurements of a Kanban system (CFD, Cycle Time, Defect Rate, Blocked Items)[58]

Identifying improvement opportunities and acting on them is a vehicle that moves a Kanban system from its initial state, which is usually a combination of the current process and some best guesses, to an increased level of efficiency. As Kanban visualizes bottlenecks clearly and enforces taking explicit and informed decisions based on empirical data, improvement potentials can quickly be identified. Actual improvement proposals to the system can be quantified and proven by measuring their impact. Ad-hoc discussions paired with a regular cadence of retrospectives help as a catalyst for continuous improvement of the process.[59]

The approach of introducing a Kanban system by starting with the current process and gradually implementing the core principles as they have been described above illustrates why Kanban is considered a change method rather than a fixed process framework. In the words of Corey Ladas, “Kanban is not a process. Kanban is a practice that embodies principles.”[60]

[58] taken from (Boeg, 2012, p. 53)
[59] see (Boeg, 2012, pp. 79-80)
[60] (Ladas, 2008, p. 14)

2.2.3 Scrumban

The principles of Kanban, as introduced in Subsection 2.2.2, do not exclude the implementation of Scrum. Technically, this means a software development team can practice Kanban and Scrum simultaneously.[61] However, since Kanban is a method of change to optimize the flow of work, there will be a tendency to move away from the strict ceremonies of Scrum after a few iterations.[62] Corey Ladas, who coined the term *Scrumban*, pointed out that "Scrum can also be useful as a starting point for an experienced agile team to evolve into a leaner process."[63]

Considering Scrum as a foundation for applying Kanban principles suggests a logical progression towards a team practicing solely Kanban, not Scrum. Accordingly, Ladas characterized Scrumban as the transition from Scrum to Kanban.[64]

The outcome of this thought process was a "deployment strategy for continuous delivery"[65] that combines elements of Scrum and Kanban and breaks certain rules of the Scrum framework. Starting from Scrum, Ladas suggested sequentially applying the following process changes to reach an increased flow of work and transition to a pull-based system:[66]

1. Reduce the iteration length to a maximum of 2 weeks.
2. Set multi-tasking limits. This results in an effective WIP limit, which shall be made explicit and forces the team to complete existing work before new work gets pulled.
3. Introduce a late binding of tasks to owners and consequently decouple planning from assignment to avoid artificially created bottlenecks.
4. Introduce a "ready queue" for staging backlog items that are ready to be processed. The ready queue implicitly signals the prioritization of work items.

[61] see (Boeg, 2012, p. 17)
[62] see (Ladas, 2008, p. 100)
[63] (Ladas, 2008, p. 12)
[64] see (Ladas, 2008, p. 85)
[65] (Ladas, 2021)
[66] see (Ladas, 2008, pp. 88-97)

5. Break down the “in progress” state to a finer granularity of states and introduce additional inter-process buffers.
6. Use the CFD diagram and associated measures instead of the burndown chart to control the process and flow of work.
7. Do not estimate work items, but use the size of the ready queue as a trigger for planning activities. Planning based on the average historical cycle time is sufficient.
8. Define work standards for each process state, which may be improved according to the principle of continuous improvement.

The outcome of this strategy is a leaner process that harmonizes with the principles of Kanban and contradicts Scrum in a few aspects:

- Estimations are not practiced as they are considered a wasteful activity.
- The team is fully self-organized, and roles such as Product Owner and Scrum Master do not exist.
- Work items do not have artificial deadlines and may well exceed the duration of an iteration.

The iterative approach with review and retrospective meetings has been carried over from the Scrum framework, as well as the necessity to agree on a “definition of done” for work items. According to Ladas, “Scrumban demonstrates the relationship between continuous delivery and continuous improvement,”[67] which both remain the ultimate goal of the transition strategy.

2.2.4 Large-scale Agile Frameworks

Agile methodologies such as Scrum and Kanban typically target individual cross-functional and co-located development teams. The “team of ten” or the “two-pizza team”[68] has become a catchphrase in this context. Due to the popularity and proven success of agile software development, larger organizations are also seeking to introduce agile processes at scale to be better prepared for competitive and disruptive markets. This approach does not oppose the methodologies introduced

[67] (Ladas, 2021)
[68] see (Amazon Web Services, 2023, p. 28)

earlier in this chapter but opens new problem spaces, which are not yet well covered by the highlighted tools or best practices. For example, the Scrum framework describes only Product Owners, Scrum Masters, and the development team. There is no dedicated role, nor is there a necessity for management hierarchies, such as middle management or a team leader, which is a predominant concept for handling the complexity of large organizations. In addition, development teams in large organizations tend to have many cross-team dependencies, which does not reflect the often-assumed ideal of having a self-organized and self-independent team.[69]

There are several agile frameworks, primarily based on Scrum and Kanban, that address the scaling needs of large agile organizations. The details and complexities of these large-scale agile frameworks are beyond the scope of this thesis, but for the sake of completeness, the following paragraphs highlight the major frameworks and their core components.

The most widespread framework is the *Scaled Agile Framework (SAFe)*. It has been built for large organizations and comes with different variants catered to the individual needs of an organization. With its basis variant, SAFe introduces three planning layers: team, program, and portfolio planning. Each layer follows an individual agile process, which is integrated using the agile patterns of the SAFe framework.[70] SAFe combines teams into Agile Release Trains and introduces a variety of new roles, such as the Product Manager, System Architect, and more.[71] Figure 10 schematically depicts the full complexity and coverage of SAFe.

[69] see (Schell, 2019)
[70] see (Schell, 2019)
[71] see (Almeida & Espinheira, 2021, p. 19)

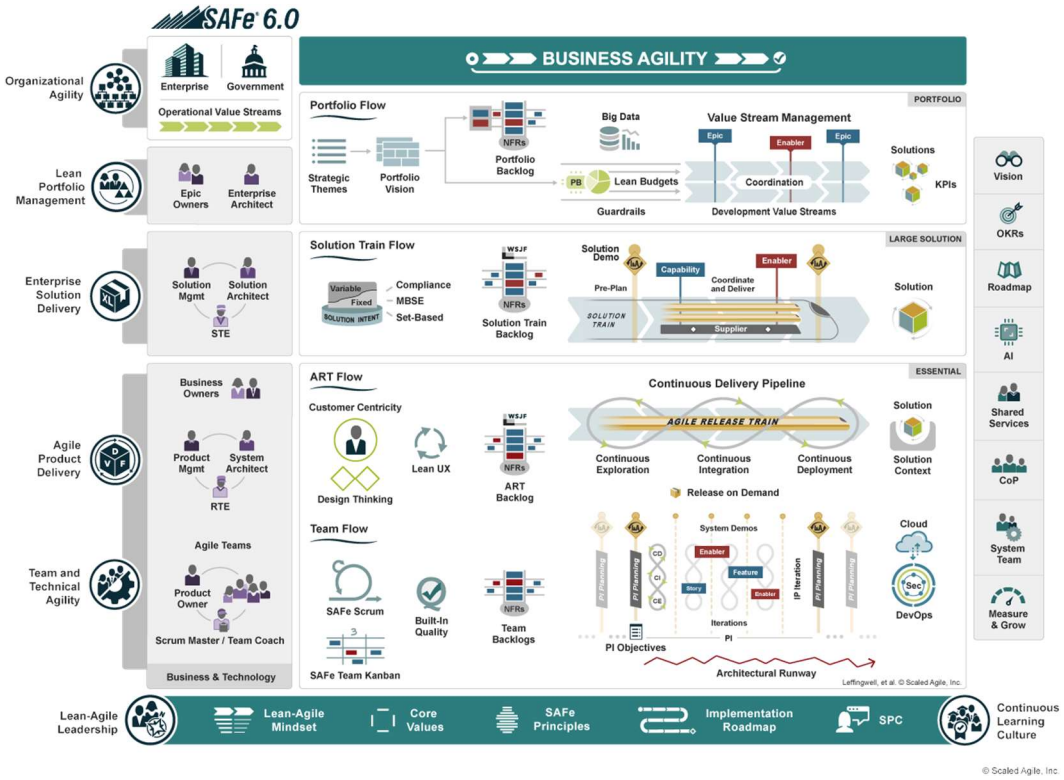

Figure 10: The full complexity of SAFe in a nutshell[72]

The *Large-Scale Scrum (LeSS)* framework is suitable for large projects rather than large enterprises. The framework is based on Scrum and limited to a maximum of eight teams with eight members working on the same backlog in synchronized sprints. It propagates cross-functional teams and eliminates traditional roles such as project manager or team leader. With *LeSS Huge*, a concept also exists for scaling to more than eight teams by introducing the role of an Area Product Owner.[73]

The *Nexus framework* acts on a similar scale to LeSS and proposes combining between three and nine Scrum teams working on the same product and backlog. Other than LeSS, Nexus introduces the Nexus integration team to function as a glue between the independent teams.[74]

[72] taken from (Leffingwell, 2023)
[73] see (Almeida & Espinheira, 2021, p. 18)
[74] see (Almeida & Espinheira, 2021, p. 18)

Disciplined Agile Delivery (DAD), *Scrum@Scale*, and *Spotify's Agile Scaling Model* are other less popular frameworks. For more details, Almeida and Espinheira conducted a comparative review of large-scale agile frameworks.[75] The authors point to a study in which seven out of thirteen global companies were found to prefer implementing a custom framework for scaling agile practices rather than adopting one of the above-mentioned frameworks. Many of these customized frameworks are evolutions of failed implementations of a standard framework.[76]

2.3 Criticism and Practicability

Agile approaches such as Scrum and Kanban assume certain preconditions to be effectively applicable. Practicing the Agile Manifesto, which prefers customer collaboration over contract negotiation and promotes adapting to change, often means the project scope is agile and volatile. The volatility in planning and scope can create tension with the project management triangle of cost, time, and scope. It can also lead to breaches of traditional customer contracts where these aspects have been agreed upon in advance. This example shows that the agile mindset should ideally be spread across the full value chain of product development, which includes developers, managers, customers, and more.

In their research on challenges when implementing large-scale agile frameworks, Conboy and Carroll call this aspect "misalignment between customer processes and frameworks." The study suggests that customers should be involved during the selection of a large-scale agile framework.[77] While this recommendation may be applicable for mid-sized companies with close customer collaboration, it may not be practicable for large-scale enterprises with many customers.

In practice, companies start implementing agile frameworks for only parts of their value chain, as implementing them on a broader scale is not possible due to the complexity of processes and the lack of control and influence over stakeholders, such as customers or even departments within the company itself. At the interface between agile practitioners and non-agile parts of the value chain, there is friction

[75] (Almeida & Espinheira, 2021)
[76] see (Conboy & Carroll, 2019, p. 44)
[77] see (Conboy & Carroll, 2019, p. 47)

between traditional outside requirements, agile principles, and the uncertainties that the agile mindset embraces. This typically leads to bounding conditions, which are unsuitable for implementing agile frameworks. Figure 11 provides an incomplete list of bounding conditions that may typically interfere with conditions that are ideal for implementing agile frameworks.

Agile Ideal	Enterprise Reality
Stable teams	Fluctuation and reorganizations
Co-located teams	Distributed teams
Cross-functional teams	Functional teams
Self-organized teams with motivated individuals	Individuals requiring guidance and guard rails
Teams of ten or less	Teams of twenty or more
Focused work on one product	Work on multiple products and components in parallel
Greenfield product development	Technical debt and high maintenance workload
Predictable work packages	Less-predictable aspects, such as research activities and maintenance load
Close customer collaboration	Little customer interaction

Figure 11: Ideal conditions for applying agile frameworks compared to perceived enterprise realities

The most adopted and, hence, most criticized agile framework is Scrum, which dedicates an own role, the Scrum Master, to addressing the friction points mentioned above. As Scrum is a framework rather than a process, the solution to obstacles tends to be the appropriate selection and usage of tools in the Scrum toolbox. Over time, this large solution space reduces the amount of guidance the framework provides and may increase the chance of eventually deviating from the Scrum approach. Nevertheless, Scrum Master is a certified role that incorporates

high investment in terms of training and from a monetary perspective. This personal upfront investment may lead to the intrinsic motivation of individuals to strictly teach and adhere to the commonly propagated Scrum rules.

Sutherland described this commonly practiced and rigid approach to Scrum as *Type A* Scrum with static isolated sprints. Possible evolutions are *Type B* or *Type C* Scrums with overlapping iterations and variable sprint lengths.[78] This evolutionary interpretation of Scrum is close to Ladas' approach to Scrumban. Ladas sees the rigid over-commitment to the Scrum artifacts as an error commonly committed by the community of practitioners.[79] Conboy and Carroll confirm this view by highlighting "overemphasis on 100% framework adherence over value" as one of the main challenges companies face when implementing large-scale agile frameworks.[80]

Practitioners of Kanban criticize Scrum for its extensive practice to estimating work packages. From a Kanban perspective, which only looks at the overall flow rather than individual work items, this is considered a wasteful activity.[81] Other central artifacts of Scrum are also questioned in literature:

- The static pattern of daily standup meetings shall be replaced by a strong focus on issue management and resolution of impediments.[82]
- Planning games cause operational overhead when used in larger teams responsible for multiple products.[83]
- Product planning, design, and requirements analysis are solely covered by the Product Owner role, which trivializes the complexity of this part of the process.[84]
- Assigning tasks to individuals during sprint planning creates an artificial bottleneck. Late-binding of tasks is practiced to increase flow.[85]

[78] see (Sutherland, 2005, pp. 13-14)
[79] see (Ladas, 2008, p. 11)
[80] see (Conboy & Carroll, 2019, p. 47)
[81] see (Anderson, 2010, p. 137) and (Ladas, 2008, p. 99)
[82] see (Anderson, 2010, pp. 90, 240)
[83] see (Anderson, 2010, p. 111)
[84] see (Ladas, 2008, p. 12)
[85] see (Ladas, 2008, p. 89)

The criticism within the agile community, as summarized above, targets the rigid approaches of Scrum in various ways and proposes more demand-based agile tools instead. When following this path and moving from Scrum to Kanban, the level of direct guidance and easy-to-implement process artifacts decreases, which reduces acceptance of the approach in more process-oriented enterprises. Simply put, selling a packaged and clearly defined product such as Scrum and its ceremonies to the management team is easier than selling the confidence that each obstacle can be overcome by continuously applying process changes and adapting to surrounding conditions.

In addition to the controversies within the agile community and the point of view of management teams, the broader community of software developers addresses more shortcomings of the approaches highlighted in Section 2.2. The Manifesto for Software Craftsmanship, introduced in Subsection 2.1.3, criticizes the Agile Manifesto for its delivery-oriented principles. According to the movement, agile principles are a catalyst for delivering features for the sake of delivery and flow while cutting corners on internal product quality.

Event	Duration	Overall hours
Sprint	4 weeks	160 hours
Sprint Planning	8 hours	8 hours
Daily Scrum	15 minutes per day	5 hours
Sprint Review	4 hours	4 hours
Sprint Retrospective	3 hours	3 hours
Backlog Refinement	5% - 10% of sprint	12 hours

Figure 12: Duration of re-occurring Scrum events for a 1-month sprint[86]

Developers tend to consider the ceremonies of Scrum as meeting overhead, which effectively reduces focused working time and, subsequently, productivity. Figure

[86] see (Schwaber & Sutherland, 2020) and (Sutherland, 2010, p. 27)

12 shows the meeting durations considered by the Scrum Guide and Sutherland's Scrum Handbook for a one-month sprint.

While there is no clear recommendation for meeting durations in the Scrum Guide regarding other sprint lengths, it is generally assumed that meeting times are linearly reduced for shorter sprint durations. Figure 13 shows the relative distribution of meeting and non-meeting times in a typical Scrum setup based on the numbers in Figure 12. It can be concluded that 20% of the available working time is usually spent on meetings.

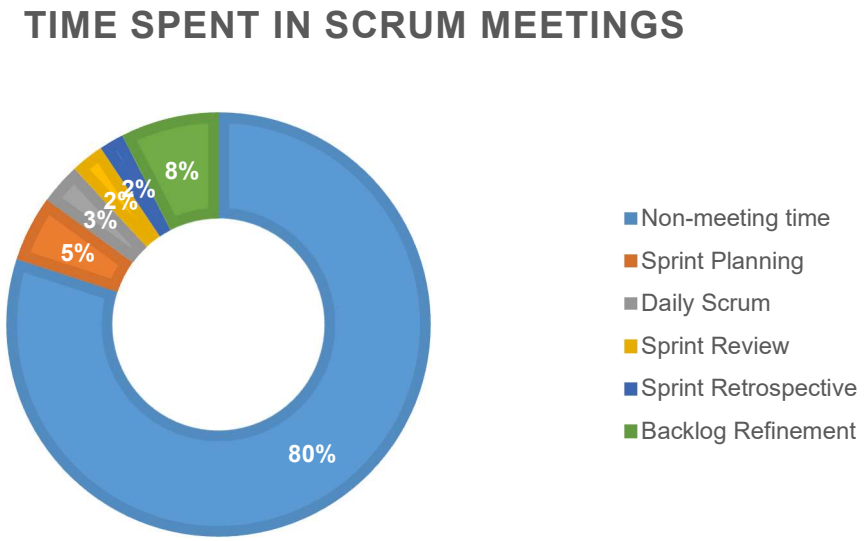

Figure 13: Relative distribution of meeting and non-meeting times in a typical Scrum[87]

The focused working time will typically be significantly less than 80% of the available working time due to context switches before and after events and further non-Scrum events like employee meetings.

In summary, implementing an agile software development process involves various challenges and shortcomings:

- It is desired, but may not be practicable, to implement Agile across the entire value chain.

[87] own representation based on assumptions in Figure 12

- Project and quality management may conflict with agile development teams.
- Agile embraces uncertainty, while contract and risk management does not.
- Ideal preconditions for implementing an agile development process are hard to achieve.
- The adoption rate is higher for approaches with simple and exhaustive implementation guidance, which often results in over-commitment to certain framework elements.
- Applying agile principles may result in working but inferior software.
- Process orchestration and participation reduce the bandwidth of developers.

As mentioned in the introduction to Chapter 2, agile principles are based on experience rather than scientific research. Additionally, there are few robust studies on the effectiveness of the methodologies. In their meta-analysis on outcomes of agile project management, Koch et al. found only 35 published studies, of which only a fraction showed a causal relationship between methodology and outcome.[88] In this context, Schermuly and Meifert state, "if Agile were a vaccine, we would not even have reached phase 1 of testing yet."[89]

2.4 New Work in the Context of Agile

Adjacent to lean and agile practices, the *New Work movement* has become popular in the last decade. This movement originates in Frithjof Bergmann's concept of New Work.[90] While Bergmann's ideas are socially utopian, Markus Väth took them further and formulated five principles of New Work, known as the *New Work Charter*.[91] Väth's less utopian but still humanistic understanding of New Work finds significant adoption in the industry. Companies that participated in the New Work Barometer 2022 were likely to identify the movement with Väth's New Work Charter or the closely related theme of psychological empowerment of employees rather

[88] see (Koch, et al., 2023, pp. 688-689)
[89] (Schermuly & Meifert, 2022a, p. 30), translated from German
[90] see (Bergmann, 2004)
[91] see (Väth, et al., 2019)

than Bergmann's initial idea.[92] This section gives a brief overview of the concept, restricting the focus to Väth's interpretation of New Work and how it relates to agile methodologies.

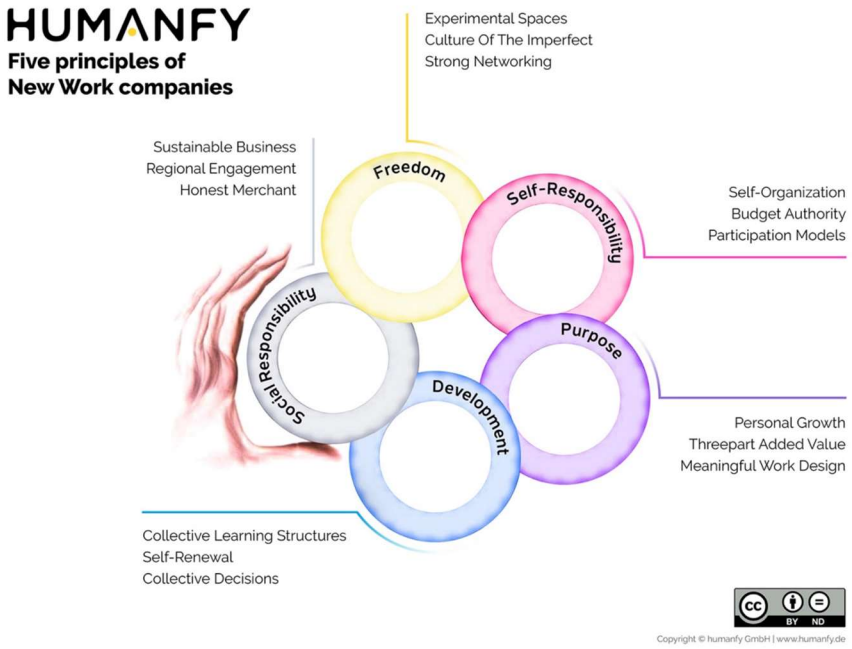

Figure 14: The five principles of the New Work Charter[93]

The New Work Charter aims to provide "answers to questions about future leadership models and organizational structures, process and product dynamics, new values, and complex economic contexts—economically and humanly."[94] It proposes five principles to be reflected by companies and employees:

1. **Freedom**: Granting individuals the freedom to act on new ideas and topics.
2. **Self-responsibility**: Simplifying the decision-making process by empowering employees and managers and setting the right boundary conditions with financial participation.
3. **Purpose**: Creating a common organizational purpose and using employees according to their strengths.

[92] see (Schermuly & Meifert, 2022b, p. 6)
[93] taken from (Väth, et al., 2019)
[94] (Väth, et al., 2019)

4. **Development**: Treating the organization as an organism to promote internal learning structures, organizational self-renewal, and collective decision-making.
5. **Social Responsibility**: Companies and their employees are embedded in the wider social, political, and ecological environment. To maintain a sustainable business, it is essential to engage with this environment and value its well-being.

The people-centric and self-managed approach of these principles is reflected in the agile and lean movements introduced in Section 2.1. Companies may adopt agile frameworks and processes to cope with constant change. Agile can help address this challenge from a product perspective, but New Work offers a more comprehensive approach by integrating agile methods as part of the New Work tooling.

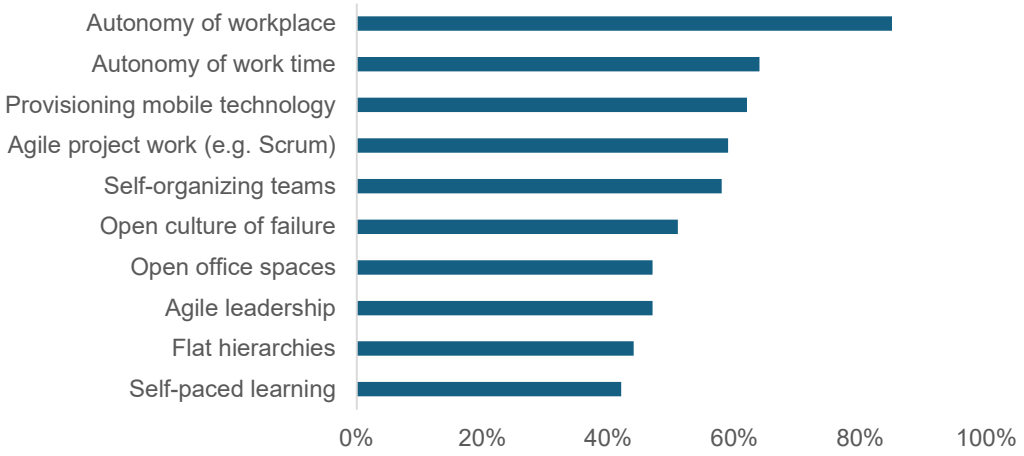

Figure 15: Percentual adoption of New Work measures in companies in 2022[95]

After autonomy of workplace/time and provisioning of mobile technology, agile project work is the fourth most implemented New Work tool amongst companies that participated in the New Work Barometer 2022.[96] According to the authors of the New Work Barometer, companies miss out on the measure's potential when implementing Agile solely for increasing the performance and innovation power of teams. They propose to additionally aim at using agile approaches for improving

[95] own representation based on (Schermuly & Meifert, 2022b, p. 11)
[96] see (Schermuly & Meifert, 2022b, p. 11)

efficiency and humanization, which is expected to result in a higher satisfaction rate with agile frameworks.[97]

While the literature on agile methodologies usually does not reference the New Work movement, Agile has become a vital part of the New Work toolbox and is referenced in literature accordingly. When implementing Agile in the context of New Work, employees may even fail to tell the concepts apart.[98]

[97] see (Schermuly & Meifert, 2022a, p. 28)
[98] see (Schermuly & Meifert, 2022a, p. 30)

3. Proposal for a Workstream-based Development Framework

This chapter proposes a novel Workstream-based Development Framework (WDF), based on existing agile frameworks such as Scrum or Kanban, while addressing some of their limitations. The framework incorporates the principles of New Work by giving developers an entrepreneurial role in the development process and allowing for flexibility and adaptation in response to changing conditions.

3.1 Motivation

The most popular agile framework, Scrum, and its central element of sprints, as introduced in Subsection 2.2.1, relies on a team sports metaphor: In most cases, team sports is about delivering performance, competing, and eventually winning a contest. While this picture resonated well with management principles two decades ago, and does so today in some cases, the competitive framing does not resonate well with Generation Y or Generation Z employees in the era of New Work, who value aspects such as a supportive work environment and work-life balance higher than former generations.[99]

Additionally, the operating models of software companies have changed from a license-based on-premises model towards cloud software, which needs to be operated and continually enhanced and maintained. Being a winning company in this market environment is not well encapsulated by a sports analogy—it is less important to win individual games (i.e., license deals) than to operate successful software and maintain customer satisfaction with a resilient and sustainable organizational setup. This is where the analogy of Software Craftsmanship, with its focus on quality, ownership, and collective responsibility, becomes more relevant.

This master's thesis proposes an agile development framework based on elements of Scrum and Kanban and pursuing a collective approach to software development dependent on Software Craftsmanship and New Work principles. As traditional craftsmen are technical professionals and, in many cases, small business owners

[99] see (Waworuntu, et al., 2022, p. 286)

with an entrepreneurial spirit simultaneously, the Software Craftsmanship perspective inevitably harmonizes with the New Work principle of self-responsibility based on economic value and financial contribution, as described in Section 2.4.

The main goal of the framework introduced in this thesis is to create a sense of collective ownership and contribution in a team of software developers, enabling them to focus on the long-term success of the jointly owned product. This way, the team gets closer to a *collective* in its original sense: “an organization or business that is owned and controlled by the people who work in it.”[100] Eventually, the development team will become a vibrant, dynamic collective committed to creating a high-quality, versatile, and user-centered product. Figure 16 shows the analogy of traditional craftsmen being collectively responsible for the outcome of their work.

Figure 16: Vibrant scene of craftsmen collectively building a barn[101]

Shaping and prototyping the framework introduced in this thesis, started by observing the interactions and outcomes of a team of ten software developers at the business software company SAP in 2019. The team initially focused on developing one specific database engine within a larger development organization, and their internal development process was somewhat aligned with Scrum.

[100] (Cambridge Dictionary, 2024)
[101] taken from (Fath, 2018)

The working mode at the time could be seen as a diluted version of Scrum, something occasionally called by the derogatory name ScrumBut.[102] It is possible that the team unknowingly took some iterations of Ladas' Scrumban approach[103] and moved from their initial Scrum to a more agile and less wasteful process. One prominent example is the absence of estimations, which the team has claimed are meaningless due to the high amount of less-predictable research work that needed to be done in this environment.

In 2020, during the COVID crisis, which involved economic challenges that caused budget and hiring constraints, the team was merged with two other teams. This situation eventually led to 20 developers under one formal team leader, who has been responsible for four database engines. Because they were unable to re-hire on attrition, the staffing of the individual teams became unbalanced, putting the development process under stress. This situation caused a significant slowdown and loss of focus for major development projects targeting technical debt and also customer-facing features. While trying to avoid disruptive redeployment of developers, the team of 20 sought a more resilient working model that helps rejuvenate velocity by enabling a transition from historical team silos to shared responsibility for the four database engines across the 20 developers.

3.2 Overview of Structures and Processes

This section describes the practices established as part of WDF. At the framework's core, the development team and respective planning layers are modeled as a matrix with technical differentiators in one dimension and planning themes in the other. A planning theme, here called a workstream, may span more than one technical component, and technical components may be involved in more than one planning theme.

Figure 17 shows an example of ten developers working in a matrix setup with three technical components and three workstreams. In this example, workstream A involves the technical components A and B, workstream B only involves component

[102] see (Scrum.org, 2024)
[103] see Subsection 2.2.3

A, and workstream C is again a cross-topic between components B and C. Each workstream is staffed with experts in the respective technical components, whereas one individual acts as the lead for the workstream.

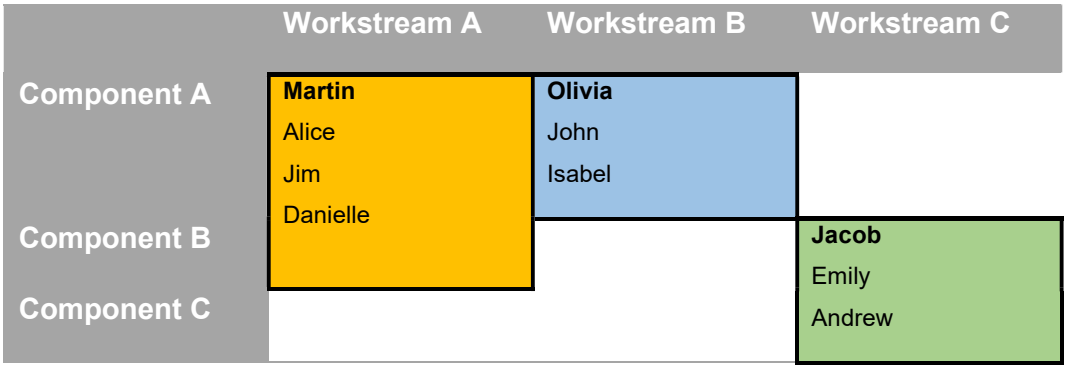

Figure 17: Example of ten developers working in a matrix of three technical components and three workstreams

3.2.1 Definition of Planning and Work Units

Organizing the team as a matrix of technical components and planning themes adds to the complexity of the agile planning process, as both dimensions need to be considered appropriately throughout the process. Before going into the details of the planning and execution process and its different layers, some central terms need to be defined:

- **Technical component**
  A technical component is a part of a larger software project that, due to its size and complexity, requires developers to be experienced with the component's source code to maintain and enhance it. A technical component is typically not a complete software product but contributes to it as a technical backbone.
- **Team**
  The software development team is a team of developers responsible for a small number of technical components. Each developer may specialize in one or more of these technical components. All team members have a common vision, share the same development process, and appreciate transparency on the overall activities within the team.

- **Workstream**
  A workstream is a large project or theme developed within the team. It may involve more than one technical component, and the development duration is between six months and several years. A workstream is ideally self-contained and can be planned and executed independently from other projects and initiatives within the team.
- **User story**
  The development of encapsulated functionalities or features that contribute to a workstream is called a user story. The definition of a user story is similar to the concept commonly used in Scrum. However, other than Scrum, there is no general recommendation for a user story to fit within the duration of an iteration. A single user story can spread over several iterations but will never exceed the lifetime of a workstream.
- **Development task**
  Each user story consists of multiple development tasks created during planning and execution. Tasks capture developers' actual work in the team and may be very technical. Each code contribution and developer activity is linked to a task.

Workstreams, user stories, and development tasks define a hierarchy: A *team* runs multiple *workstreams* spanning one or more *technical components*. Each workstream is organized into *user stories*, which are iteratively broken down into *development tasks*.

3.2.2 Layers of Planning and Execution

Planning and execution within WDF are organized along the hierarchy of layers introduced in Subsection 3.2.1. Depending on the respective layer of planning, different team members participate in the planning process to avoid unnecessary meetings and planning overhead for those not deeply affiliated with the respective planning topic. While Subsection 3.2.4 presents details of meeting audience, duration, and cadence, this subsection introduces the planning layers on an abstract level.

From a high-level perspective, the software development team structures its work into larger workstreams. The team collaboratively decides which workstreams need to be started, paused, or stopped. It should also be clear what the overarching goal of the workstream is, how it connects to the team's vision, and what criteria will be consulted to verify whether this goal has been reached. While the goal of the workstream remains stable over its lifecycle, the criteria for measuring goal achievement are part of a change process and may improve and become more concise over time. During workstream planning, the team jointly decides on goal refinements and required workstream staffing. Team members assign themselves to workstreams based on expertise, preference, and demand.

The next level of planning occurs within the respective workstreams. As described in Subsection 3.2.1, user stories are the main planning objects on the workstream level. Developers within a workstream must decide what work packages need to be prioritized and achieved to meet the workstream's overarching goal. Additionally, there needs to be clarity on future staffing requirements and the skills required to complete those work packages.

The concept of user stories is similar to its concept within Scrum. For each user story, the planning participants need to agree on a *definition of done* and nominate a responsible developer to divide the user story into individual development tasks and finally approve its completion. Figure 18 summarizes the primary goals of each planning level.

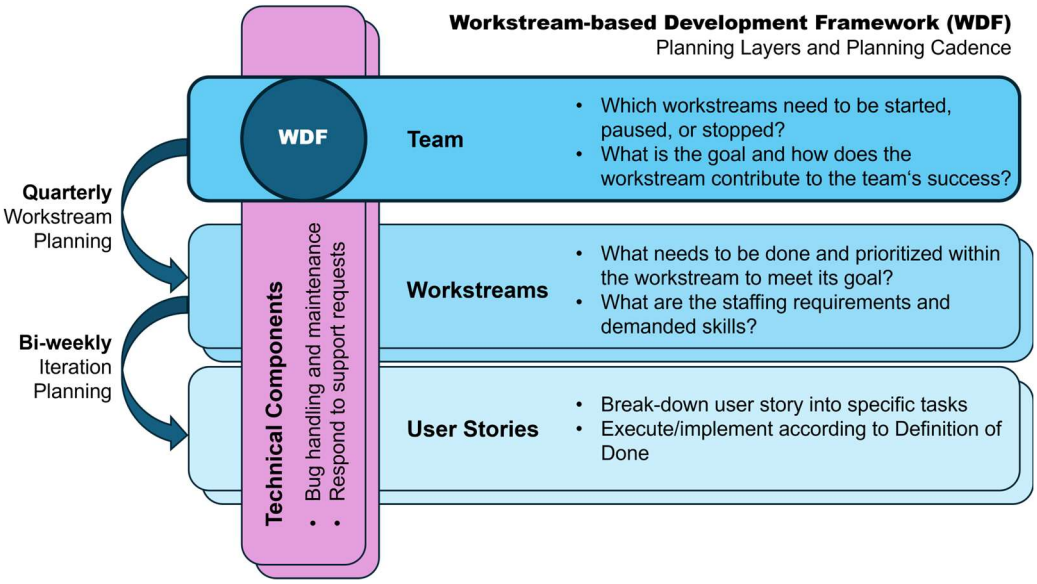

Figure 18: Planning layers and planning cadence of WDF

At the execution level, individual developers work based on tasks that are part of a user story. In addition to tasks originating in workstreams, developers contribute to maintenance tasks such as bug handling within the scope of their assigned technical component. This approach effectively reduces the bandwidth to work within the scope of a workstream since these component duties are independent of feature development. The developers within a technical component will conduct separate lightweight planning to orchestrate the mostly reactive maintenance activities.

3.2.3 Roles and Responsibilities

Since the team works collaboratively towards the same vision and tries to make decisions by agreement, there is no strictly given hierarchy within the team. However, based on a consensus decision, team members can take on specific roles within the framework. By assigning roles and responsibilities to accountable individuals, less critical decisions can be made more efficiently, and only more significant and valuable discussions may be shared with a wider audience within the team.

Having visible roles also promotes less experienced colleagues' growth and opens the possibility of widening their responsibilities within a controlled scope and for a

defined duration. Roles are not assigned exclusively, meaning one developer may have several roles simultaneously.

For WDF, one or more individuals must be assigned to the following roles based on team consensus.

3.2.3.1 Workstream Lead

Each workstream is led by a developer, who oversees the activities within the workstream and acts as a single point of contact for stakeholders outside the respective workstream. The Workstream Lead is assigned by a joint team decision when the workstream gets initially defined during the quarterly workstream planning. Taking this role involves a long-term commitment: Unlike other workstream members who can switch workstreams based on demand, the Workstream Lead should stay consistent throughout the workstream lifecycle.

3.2.3.2 User Story Owner

There is always one developer responsible for delivering a user story. Being the assigned owner of a user story does not necessarily mean being involved in its development tasks. The main responsibility of the User Story Owner is to break down the work package into individual development tasks and refine them regularly. Furthermore, the User Story Owner is the only person who can approve the completion of a user story based on its *definition of done* and is the single point of contact for this user story.

3.2.3.3 Task Assignee

Developers within a workstream may take any task created by a User Story Owner of the same workstream and assign themselves to complete it. Typically, the assignment happens in coordination with the other workstream members during their daily cadence.

3.2.3.4 Component Architect

Each technical component has an assigned chief architect, who must be consulted for major technical decisions made within a workstream that touch upon the respective technical component. The Component Architect may decide to consult

with other developers assigned to this technical component before making a final decision. Furthermore, the Component Architect orchestrates any maintenance activities within the component.

3.2.3.5 Collaboration Catalyst

Like the Scrum Master described in Subsection 2.2.1.1, the Collaboration Catalyst supports the team to stay consistent with the development framework. The Collaboration Catalyst does not need to be a single person, but the role may be shared between a small number of developers. Like the Workstream Lead, this role stays consistent over a longer period, as frequent changes may cause tensions in the team and result in a reduced flow of work through the process.

3.2.3.6 Team Lead

WDF does not make the Team Lead a mandatory role as it is based on collective decision-making, self-organized teams, and collective ownership. However, this framework specifically addresses the needs of teams within large development organizations. This environment is common in enterprise companies, where teams are embedded into a given hierarchy. Consequently, the role of the Team Lead or Development Manager is already manifested and should be leveraged to the best of its ability during the development process. There are at least two ways the Team Lead can effectively contribute to the team's success, given their role in the wider organization:

- *Bridge to the corporate environment*: Agile cells are often embedded in less agile environments, which leads to the tensions described in Section 2.3. The Team Lead is in an ideal position to safeguard the team from external requests that interfere with their development process. On the other hand, the Team Lead is able to translate the goals and progress of the team into an externally comprehensible message.
- *Escalation instance*: If, in certain situations, the team fails in its process of collective decision-making, the Collaboration Catalyst shall be the first escalation instance. In the rare event of deadlock situations, the Collaboration

Catalyst may decide to involve the Team Lead to include the corporate perspective in the decision.

The Team Lead is considered a part of the development team and can optionally have the additional developer role.

3.2.4 Meeting Structure, Cadence, and Duration

To jointly plan and execute tasks on the layers introduced in Subsection 3.2.2, a dedicated meeting structure facilitates collaboration and knowledge-sharing on the various levels. The general meeting structure of WDF is comparable to the structure given by the Scrum framework in Subsection 2.2.1.2. However, WDF adds the aspect of quarterly workstream planning and weekly component circles while reducing the duration of other events known from Scrum. Figure 19 gives an overview of meetings with their respective cadence and duration.

	Cadence	Audience	Duration	Moderator
Workstream Planning	Quarterly	Team	1.5h	Collaboration Catalyst(s)
Workstream Iteration Wrap-up	Bi-weekly	Workstream	0.5h	Workstream Lead
Iteration Review	Bi-weekly	Anyone	0.5h	Collaboration Catalyst(s)
Iteration Retro + Outlook	Bi-weekly	Team	1h	Collaboration Catalyst(s)
Workstream Circle	3-4 times per week	Workstream	0.25h	Workstream Lead
Component Circle	Weekly	Component	1h	Component Architect

Figure 19: Meeting cadence and duration of WDF

Each meeting has a specific purpose and may or may not require individual preparation by team members. It is more important to keep the purpose of the event in

mind than to follow a fixed agenda. The team may individually and optionally agree on an agenda if this helps ensure the essential talking points have been touched. These meetings, as listed in Figure 19, need to be considered as part of the process:

3.2.4.1 Workstream Planning

Every quarter, the team decides jointly which workstreams to start, pause, and stop. Stakeholders or team-internal initiatives may trigger those changes to the workstream setup. Each Workstream Lead is asked to prepare an overview of the workstream's agenda for the next quarter, which helps the team understand the state of each ongoing workstream. If the team decides to pause a workstream, it is not discussed during the iteration. In particular, the team does not conduct workstream circles and iteration wrap-ups for paused workstreams.

3.2.4.2 Workstream Iteration Wrap-up

The workstream iteration wrap-up is the preparation meeting for the iteration review and outlook. The goal is to discuss the progress of the last iteration and the planning for the upcoming iteration within the small group of developers currently assigned to the workstream. This approach keeps detailed discussions out of the review and outlook meetings with a larger audience. Ideally, during the iteration review, the Workstream Lead presents the results of the wrap-up meeting and clarifies open questions. Consequently, having a thorough wrap-up meeting within the workstream enables shorter meetings with a larger audience.

3.2.4.3 Iteration Review

Any stakeholder interested in the team's progress may be invited to this meeting, including adjacent and dependent areas such as documentation or solution management teams. Each Workstream Lead or a dedicated substitute gives an overview of the progress made during the last iteration. This overview is not limited to delivered features but may include technical details or work in progress to increase the transparency of activities across the team.

3.2.4.4 Iteration Retrospective + Outlook

The team-internal retrospective and planning are combined into one event. While the retrospective part of the meeting does not deviate from what has been described in Subsection 2.2.1.2, the emphasis of iteration planning is shifted towards an *outlook*. This naming indicates that the actual planning happens within the workstream iteration wrap-up, and the results are communicated to the broader team during the outlook. However, at this stage, legitimate discussions or objections from the broader team are still welcome and may require follow-up with certain developers. Workstream Leads may communicate the necessity of changing staffing due to missing skills or resource demand. The team jointly discusses the staffing situation, and individuals may switch their workstreams based on a collective decision.

3.2.4.5 Workstream Circle

As development will happen collectively, developers within a workstream are expected to collaborate and talk frequently without waiting for dedicated meeting slots. Some individuals may even choose to practice pair programming to foster collaboration and knowledge sharing. This approach makes fixed daily meetings for synchronization less important. However, for more inexperienced developers in a remote or hybrid work environment, the concept of daily meetings, as used in Scrum, still adds value to the collaboration process. Depending on preference, the workstream may opt for daily or bi-daily "*circle*" meetings. As suggested in Section 2.3, these meetings do not follow a static pattern but focus on issue management and resolving impediments. The format offers a safe circle for seeking support from fellow developers.

3.2.4.6 Component Circle

Similar to the daily workstream circle, there is a weekly component circle that facilitates collaboration between the developers assigned to the same component. A weekly cadence with a longer meeting duration caters to the less interaction-critical maintenance activities within a component better than a daily cadence, which may result in meeting overhead. Within the component circle, ongoing activities such

as the assignment and resolution of bugs can be discussed. The forum can also discuss technical requests that previously arose within the workstreams.

3.2.4.7 Exemplary Bi-weekly Meeting Schedule

It is strongly suggested by WDF to parallelize the meeting events introduced in Subsection 3.2.4 to a maximum. Holding all daily workstream circles simultaneously ensures that developers are unable to join the circle of a workstream they are not currently affiliated with, although experienced developers with broad interests may be tempted to do so. Furthermore, the parallel occurrence of meetings makes it easy for developers to switch workstreams at the end of an iteration, as they do not have to change their daily schedule and habits.

A situation where the members of workstreams or components wish to change their meeting schedule can quickly put the overall team schedule at risk, as competing meeting situations, such as those between the daily workstream circle and the component circle, may impede developers entering the workstream in the future. The Collaboration Catalyst needs to prevent those situations without exception.

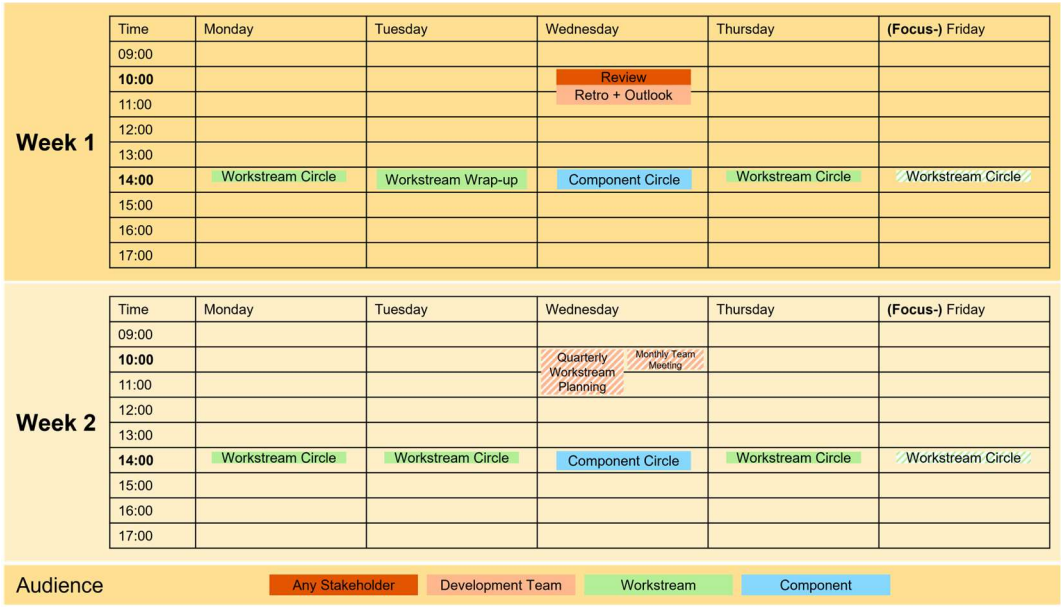

Figure 20: Exemplary bi-weekly meeting schedule for WDF

An exemplary meeting schedule with an iteration length of two weeks is depicted in Figure 20. The schedule uses parallelization where possible and foresees a daily event at 14:00 and additional meetings on Wednesdays at 10:00. To avoid meeting overhead, it makes use of the fact that workstream circles are not mandatory to be conducted daily. This gives the flexibility to optionally introduce a meeting-free day—in this case, denoted as “Focus-Friday”.

In weeks where new iterations begin, the meeting-heavier Wednesday is used to conduct iteration review, retrospective, and outlook. These events are preceded by the workstream iteration wrap-up on Tuesday afternoon. On Wednesdays, the component circle replaces the usual workstream circle. When using the concept of meeting-free days, the workstream members still meet at least three times per week, which is tolerated by the framework as per Subsection 3.2.4.5.

Every second week leaves spare time on Wednesdays, which may be shared among meetings with a lower cadence, including the quarterly workstream planning, as well as meetings that are not part of WDF, such as a general monthly team meeting.

3.2.5 Guardrails for Focused Delivery

Certain guardrails must be applied to avoid dilution of the development process over time. These are a set of rules that ensure that the team remains focused on delivering toward the goal of the workstreams instead of investing in less valuable work. This subsection summarizes the main areas of risk and the rules for mitigation.

3.2.5.1 Collective Decisions Require Full Transparency

It needs to be ensured that the team is empowered to make collective decisions. Otherwise, the team risks important decisions being taken within the authority of workstreams or user stories without involving the broader team and respective Component Architects. As synchronization between workstreams and the broader team happens at the transition between iterations, the following should be considered:

- **Always use an iteration length of two weeks**
  Workstream-based development works best with an iteration length of two weeks. Longer periods will decrease the level of team synchronization and transparency. Shorter periods will cause a high meeting density as regular framework meetings, like iteration review, will happen in the same week as preparation-intensive and less frequent meetings, such as workstream planning.
- **Prepare meeting content and take its scope seriously**
  To reduce meeting overhead, meeting durations within the framework have been set as short as possible. It is assumed that preparation for the respective meeting happens within the workstreams or on an individual level. If meetings are not prepared or the scope is not taken seriously, the team risks skipping relevant discussion points and decreasing transparency on activities.
- **Establish a culture of curiosity**
  A handful of enthusiastic developers can make the difference between an efficient framework implementation and a diluted process. When establishing a culture of curiosity and collective ownership, individuals will start challenging the output and planning of workstreams. Discussions at the team level are a great vehicle to get external input for the workstreams and increase transparency on details amongst the team.
- **Make process changes transparent**
  Ongoing change is part of the agile mindset. Despite the guardrails defined here, there may be a need to adapt the process in certain aspects. It must be avoided to make adjustments to the process without notice and assume joint agreement. Instead, each process change needs explicit collective consent from the team.

3.2.5.2 Getting Things Done by Focusing on Major Workstreams

Frequent context switches and multitasking are known to reduce individuals' and teams' productivity due to the associated mixing and switching costs.[104] Weinberg tried to quantify the impact of task switching for software development teams and found that each simultaneously started project adds 20% of switching costs to the team's effort. So, a developer working on two projects would have an availability of 40% per project and 20% switching costs. While other studies show that Weinberg's heuristic overestimates the effect, they still confirm the increased switching costs per parallel project.[105]

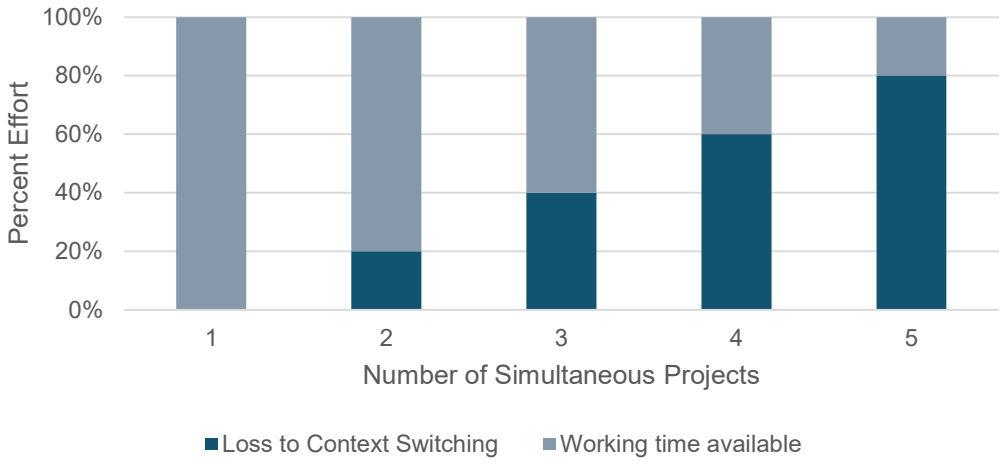

Figure 21: The cost of task switching for software developers

Contrary to the popular call to "*do more with less,*" WDF propagates the initiative to "*do less with more,*" which means working on less topics with more developers per topic. This approach requires a conscious prioritization and selection of topics ensured by appropriate guardrails:

- **A minimum of three developers per workstream**
  Having at least three developers per workstream helps the team focus on the most important topics and get things done instead of massively parallelizing topics based on stakeholder demand. Additionally, this guardrail

[104] see (American Psychological Association, 2006)
[105] see (Tregubov, et al., 2017, p. 138)

ensures proper knowledge transfer within the team, subsequently increasing resilience of the overall team setup. The team is less dependent on singular developers with deep expertise in certain areas.

- **Each developer is assigned to one workstream per iteration**
  To avoid a situation where developers take on tasks that do not contribute to the team's priorities, each developer is assigned to exactly one workstream. Switching workstreams during an iteration or taking on tasks from other workstreams is not allowed. Consequently, when running idle, each team member is forced to select a follow-up task within the scope of the current workstream. This guardrail is ideally enforced by a parallelized meeting structure as proposed in Subsection 3.2.4.7.
- **The Workstream Lead is stable**
  The Workstream Lead has a long-term commitment to this role and should not switch roles during the lifecycle of the workstream without a compelling reason, such as employee attrition.
- **Make pausing workstreams a good habit**
  The guardrails described in this subsection appear restrictive and difficult to reach for teams that adopt this framework. It ultimately reduces the number of parallel topics within a team to a minimum. The core idea is to close topics before starting with new developments. This makes it necessary to consciously pause or even stop workstreams, which may cause difficult conversations with stakeholders. As per Subsection 3.2.3.6, the Team Lead is in a good position to discuss the team's decision with stakeholders. The team should concentrate on making workstream management a good habit, especially putting workstreams on hold.

3.2.5.3 Refine Work Packages Constantly

Workstreams, user stories, and development tasks need constant refinement. This means adding new work packages, stopping deprecated work, or changing existing definitions. Depending on the planning level, which requires refinement, the following should be considered:

- **Quarterly refinement of workstreams**
  Changes to workstreams are discussed during the workstream planning meeting with the team, as these changes typically have a substantial impact. A new workstream should have a duration of at least six months while engaging at least three developers. If a topic does not fulfill this criterion, there is a high chance that it should be considered a user story that is part of a larger workstream.
- **Bi-weekly refinement of user stories**
  User stories are refined on a workstream level within the bi-weekly wrap-up sessions. Depending on the impact of the change, the Workstream Lead may decide to involve Component Architects or the whole team by facilitating a discussion during the bi-weekly outlook meeting. User stories belong to workstreams and may have any duration between a couple of days and up to six months.
- **Daily refinement of development tasks**
  Tasks, which reflect actual development work, are refined daily. As the refinement occurs within the boundaries and scope of the user story, these changes typically do not require a discussion. The User Story Owner may make ad-hoc changes to the tasks and, as a minimum, must communicate or discuss this with the affected developers during the daily workstream circle meeting. The input of one or more Component Architects may be required.

3.3 Best Practices and Recommendations

WDF, as described in Section 3.2, leaves various degrees of freedom in its implementation despite the guardrails applied with Subsection 3.2.5. As the framework is backed by an agile mindset, teams will choose to fill any gaps with agile elements and tools known from the frameworks introduced in Section 2.2. This section covers the most essential best practices to consider during the implementation and execution of the framework.

3.3.1 Integration of Feedback Loops

As the team runs their software components collectively and self-independently, they are expected to accept and incorporate relevant team-external feedback at any time and in their own interest. Depending on the expected impact, iteration and workstream planning may be used to discuss the implications of following up on the feedback.

There is also the iteration review in the framework, a formal event for gathering feedback from stakeholders. The team may decide to explicitly invite stakeholders to instances of the meeting series if their feedback is required.

Given the technical and detailed nature of iteration reviews, it is worthwhile considering incorporating a practice observed in Scrum implementations: lightweight, informational *Show & Tell Sessions* held less frequently (quarterly or bi-quarterly) and aimed at engaging stakeholders less familiar with the intricacies of specific software components. In these sessions, the team selectively promotes features with an end-user impact and concentrates on demos and practical examples rather than technical implementation details.

3.3.2 Team-internal Craftsman Swaps

The Software Craftsmanship movement identified craftsman swaps as a valuable tool to foster innovation and widen the expertise of individual developers. In an enterprise environment with corporate silos, it can even be challenging to change topic clusters within an organization or team in a non-disruptive manner. To balance staffing between teams, managers may reassign certain developers based on demand. However, this "*lift and shift*" approach typically involves disruptions for the affected developer and the sending and receiving teams. It may even be perceived as a small-scale reorganization.

The framework introduced in Section 3.2 defines boundaries to allow developers to voluntarily change their scope of work while setting the right incentives with the mindset of shared ownership. While it is not mandatory for developers to change their scope of work regularly, the team culture should encourage it. Some individuals may choose to stick to topics they feel most comfortable with, while others will experience the team-internal swaps as an opportunity to gather experience and

build expertise. Certain skill sets, such as low-level performance experts, may be able to change workstreams from iteration to iteration while consistently keeping their individual focus and contributing performance-related knowledge.

In summary, the flexibility to change topic clusters within a team in a non-disruptive way is not an option but a cornerstone of WDF. As described in Section 4.2, teams may choose to make little use of this powerful tool when initially implementing the framework but will increase and normalize its use over time.

3.3.3 Product Focus and Continuous Flow

It was implied in Section 3.1 that teams may lose focus when their process is under pressure due to external factors. Pressure typically leads to a high amount of parallelization, which comes with the mixing and switch costs described in Subsection 3.2.5.2. This situation causes a slowdown of flow, which worsens when looking at the behavior of individuals: Developers, like any human, may tend to procrastinate under pressure.[106] Hence, they may select rewarding but less important work packages over difficult and less rewarding topics that need prioritization. Agile methodologies, if not implemented correctly, can function as a catalyst for reprioritization on an individual level. Individuals may hide procrastination by parallelizing work and reporting progress on multiple work packages.

To avoid situations where developers are prone to procrastination, WDF forces everyone to focus on a given workstream. This enables the small group of workstream members to be closely involved with any development activities and to detect work with little value-add as early as possible. The restriction that subsequent or parallel development tasks can only be chosen from the scope of the workstream makes it harder to procrastinate on the overall workstream goal.

Developers may still choose to parallelize less-rewarding work packages with work packages where they expect a motivation-boosting short-term win. However, the framework ensures that even this short-win contributes to progressing the overall workstream. This way, a total slowdown of the workstream is unlikely to happen if

[106] see (Diepstraten, 2022, p. 20)

it has not been transparently and explicitly put on hold following a joint team decision.

Shaffer and Kazerouni suggest that introducing project milestones decreases procrastination within a project and increases the quality and speed of outcomes.[107] The personas that can effectively introduce milestones within this development framework are Workstream Leads and User Story Owners. The proper breakdown of work into user stories and development tasks is a core responsibility for managing not only velocity and flow, but also team culture.

3.4 Summary of Proposal

WDF, as introduced in this thesis, enables a development team to become a vibrant, dynamic collective committed to creating a high-quality, versatile, and user-centered product. Unlike other agile frameworks, it specifically addresses a large corporate organization's need for resilient teams with balanced staffing according to business priorities.

WDF organizes the development process of a software development team into two dimensions: workstreams, which capture major development themes, and technical components, which organize maintenance activities and consult workstream-based developments. The framework proposes a parallelized meeting structure to manage this complexity while at the same time reducing meeting overhead.

The division of major themes into workstreams, together with the restriction to staff each workstream with at least three developers, fosters focused work on prioritized themes. Team-internal craftsman swaps with reduced switching costs enable the team to flexibly move skilled developers to workstreams where their work contributes most to the team's success.

The team continuously refines work packages on the different planning layers and gathers feedback with iteration reviews and Show & Tell sessions, which are known from implementations of Scrum.

[107] see (Shaffer & Kazerouni, 2021, p. 6)

4. Practical Implementation and Evaluation

As mentioned in Section 3.1, Workstream-based Development Framework (WDF) has been initiated and prototyped by a software development team at the business software company SAP. This chapter summarizes the experience gathered while conceptualizing and implementing the new development framework. The chapter concludes with an analysis of the advantages, downsides, and challenges identified after using the framework for several months.

The relevant team is part of the development organization behind SAP's database offering. SAP HANA is an exceptionally large development project with more than 36 million lines of code and more than 3,000 authors contributing over 1.4 million commits since the year 2000. The source code is split into approximately 200 technical components, which are owned by the respective development teams.[108] Figure 22 compares lines of code between SAP HANA and popular large open-source software projects.

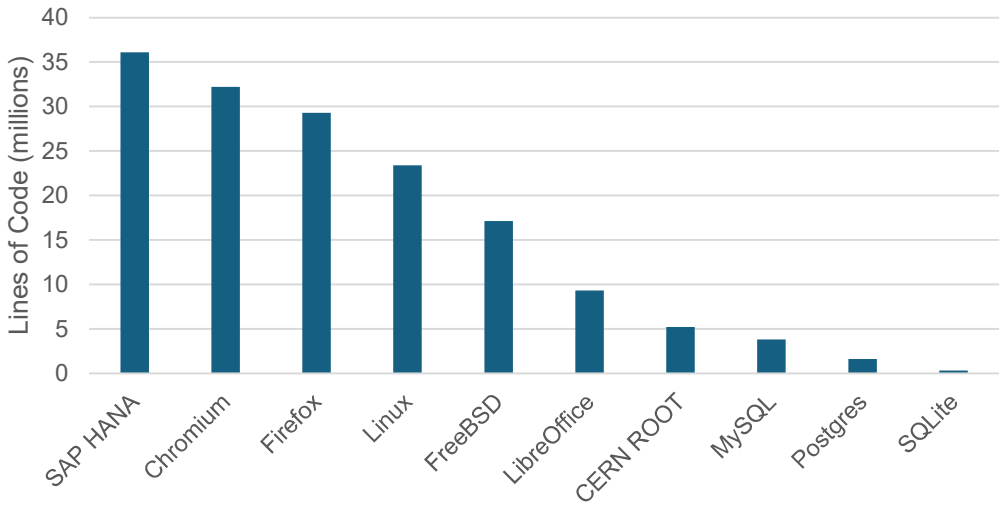

Figure 22: Comparing lines of code (millions) between SAP HANA and large open-source software projects as of May 2022[109]

According to Westrum's three cultures model, as introduced in Subsection 2.1.1, the team and its embedding organization may be allocated towards the generative

[108] see (Bach, et al., 2022, p. 196)
[109] own representation based on (Bach, et al., 2022, p. 196)

spectrum with a bureaucratic notion. Historically, the team has been merged from three teams with three different managers (subsequently referred to as "*microteams*"), where each team owned one or two components within the database product, into one team under one formal manager (subsequently referred to as "*the team*"). Each of the three microteams ran an agile process with different iteration lengths and made individual use of the tools introduced in Section 2.2. The conceptualization of a new collaboration framework was triggered by two aspects:

- **Manager perspective**
  The different iteration cycles of the individual teams increase complexity when reporting progress to upstream management: The team is expected to deliver on joint projects, which results in splitting these projects into silos within the team with some associated overhead costs due to the asynchronous development cycles. Furthermore, certain skills and roles are effectively redundant since it is unclear whether each microteam needs a separate quality, performance, or process expert.
- **Developer perspective**
  The increasing variety of topics and management expectations of delivery led to each developer parallelizing several topics, which made focused teamwork challenging. Some individuals experience the opportunity to collaborate with other microteams as a personal gain.

These triggers created a suitable environment for implementing a new framework that specifically addressed the challenges this team and others in the software industry have been facing.

4.1 Iterative Conceptualization

The conceptualization of the new development framework was based on the results of an employee survey, where the development team expressed a desire to parallelize fewer development topics and, instead, team up on certain focus topics to enable team-internal collaboration and knowledge transfer.

The team decided to use elements of Design Thinking, in particular the *Define* and *Ideate* phases,[110] to develop a problem description and approaches to mitigate the identified issues.

Figure 23: Team ideation on shortcomings of the historical development process (actual screenshot)

The unstructured result of the team's ideation session on the shortcomings of their current way of working is shown in Figure 23. The open feedback can be clustered into the following topics, which represent mainly the developers' perspective:

- Better balance of workload between components
- Clear roadmap and clear prioritization across components

[110] see (Dam & Siang, 2024)

- Focus according to defined priorities
- Ability to re-prioritize if needed
- Approach topics as a team
- End-to-end customer feedback
- Improved knowledge management
- More staffing

Based on those results, the team and their manager agreed that a new development framework should address these five main challenges:

- **Focus on topics as one team**
  Getting things done by being able to focus on topics as a joint team instead of creating overhead through parallelization and frequent context switches.
- **Continuously deliver features**
  Keeping the (visible) flow by continuously delivering roadmap features instead of solely investing in less visible but still valuable engineering tasks.
- **Handle maintenance and operational tasks**
  Maintaining the base of the product by continuously fighting bugs and technical debt while still being able to ship features to customers.
- **Balance workload**
  Being resilient to changing surrounding conditions by balancing the workload between the technical components as required per priority.
- **Drive innovation**
  Achieving sustainable long-term success by driving innovation and investing in research activities that ensure a cutting edge in the future market environment.

The necessity to focus on major development topics as a team of multiple developers necessitated the compartmentalization of work into discrete projects. Subsequently, those projects were called *workstreams* to emphasize the expectation of continuous flow within the project. The desire to act transparently as one team resulted in the planning hierarchy introduced in Subsection 3.2.2, which enables major parts of the team to have transparency and co-determination on a digestible level.

After those main elements of the framework had been defined, the team leveraged the Design Thinking concept of prototyping new ideas early and continuously refining the process.[111] The guardrails described in Subsection 3.2.5 were iteratively prototyped and integrated into the refined process to avoid degeneration and motivate individual developers to fully adopt the new framework instead of rebranding historical work patterns to match the new roles, events, and artifacts.

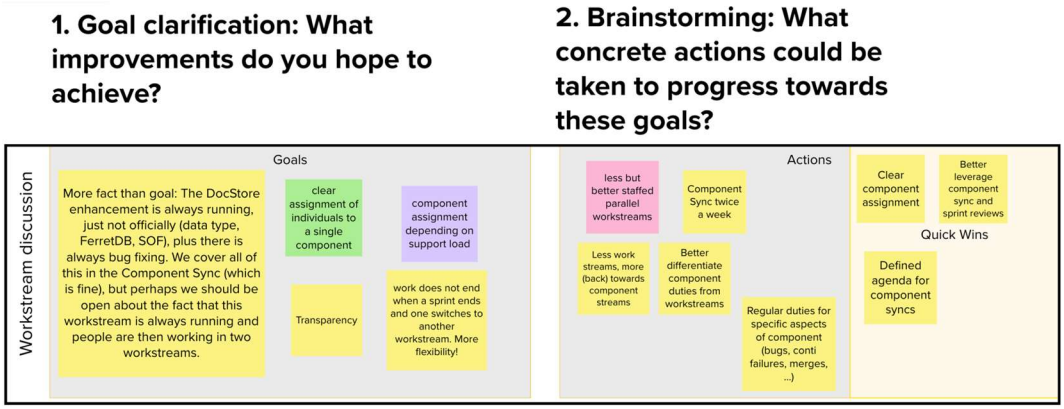

Figure 24: Result of whiteboarding iteration for process improvement after initial framework adoption (actual screenshot)

Figure 24 shows the output of a process improvement iteration. With this specific iteration, the team decided to better leverage and structure meeting artifacts such as iteration review and workstream circle.[112] It also decided to pause workstreams more consciously to avoid overhead due to parallelization.[113]

4.2 Change Management and People Transition

In an ideal setup, the team, as well as each individual within the team, would have participated equally and in their own interest in the conceptualization process outlined in Section 4.1. In practice, the team at SAP experienced at least three categories of participation for individual team members:

[111] see (Dam & Siang, 2024)
[112] see Subsection 3.2.5.1
[113] see Subsection 3.2.5.2

- **Constructive participation**

  Team members who choose to participate constructively have an intrinsic interest in conceptualizing a working model that benefits the team and themselves. The interest may be sparked by a general interest in agile methodologies or by the sheer awareness that they will be impacted by the outcome of the discussion.

- **Regressive participation**

  Team members who choose to participate regressively would like to avoid change as it involves effort and bears a certain risk, which the individual is not willing to take at this time. This view seems in many cases to be caused not by a strong belief in the status quo but by the wish to avoid disruption.

- **Nominal participation**

  Team members who choose to limit their participation to a nominal contribution invest as little as possible into the discussion as they do not acknowledge the value and impact of its outcome. Talking about the process of *how* work is done instead of actually *doing* work appears to them to be a wasteful activity. Figure 25 humorously depicts this participation type.

Figure 25: Humorous internet meme of workers prioritizing actual work over process change activities[114]

[114] unknown source

For the team at SAP, only a very few members chose to participate regressively. The majority either participated constructively or nominally in the discussion. Counterintuitively, from a change management perspective, the group participating regressively is less challenging than the group only participating nominally in the process. Regressive participation may still add value to the conceptualization phase as these individuals express strong opinions, which should be incorporated into the decision process. As they make their point of view transparent, it is easier to address their needs and desires while transitioning to a new framework.

The underlying needs of individuals who choose to participate nominally are less clear. While transitioning the team into the new framework, there is a chance that they will suffer from the decisions that have been made without their contribution. Regular pulse checks on the satisfaction level of the team and its members help assess how well these individuals are captured by the transition process. The iterative conceptualization approach, as explained in Section 4.1, makes it easy for nominally participating individuals to switch to active participation in the decision process at any point in time.

As mentioned in Section 4.1, the team at SAP also prototyped their improvement ideas during the conceptualization. Drafting the initial workstream-based model resulted in an iterative conceptualization that went hand-in-hand with a gradual implementation of enhancements by manifesting successfully prototyped concepts into the development process. The gradual implementation of WDF followed a three-step approach:

1. **Establish a joint agile framework**

   As the teams started with diverse processes with different iteration lengths, the first step was to streamline those processes and align each microteam with an iteration length of two weeks and the general meeting structure, as outlined in Subsection 3.2.4. Former Scrum Masters, who had been constructive participants in the conceptualization, shared the role of the Collaboration Catalyst.

   Formally, every microteam was now called a “workstream”. As the previous microteams still acted as a unit, this transition did not cause any major disruption to individuals but prepared the overall team to reinforce its level of

collaboration. Also, the assignment to technical components was made based on the previous team setup, which had already been organized along these components.

From an organizational perspective, the previous teams' historically separate task-tracking systems (several Jira instances) were merged into one cross-team task-tracking system (one Jira instance).

2. **Spin-off workstreams**

   Once the joint framework has been established and settled in, the team begins spinning off the first workstreams, which no longer correspond to the previous microteams. Good candidates for a seamless spin-off are very large projects within the microteams or cross-component projects involving more than one microteam. In practice, the team at SAP identified three candidate topics that transitioned into workstreams. The workstreams, defined in Step 1, continued if there was sufficient remaining staffing (at least three developers) after moving developers to the new workstreams.

3. **Become productive**

   This last step is reached when the team becomes independent from the previous microteams established according to component responsibilities. Each team member now has transparency on the overall team's priorities and acts in their interest. Workstreams are started, stopped, and paused interactively based on a joint team decision, and team members participate in those workstreams based on demand.

The duration of each transition step depends on the team's satisfaction rate and the necessity of re-adjusting certain screws in the process. Entering the next stage of transition must be an explicit decision made by the team. The gradual transition from microteams to one team acting within WDF is summarized in Figure 26.

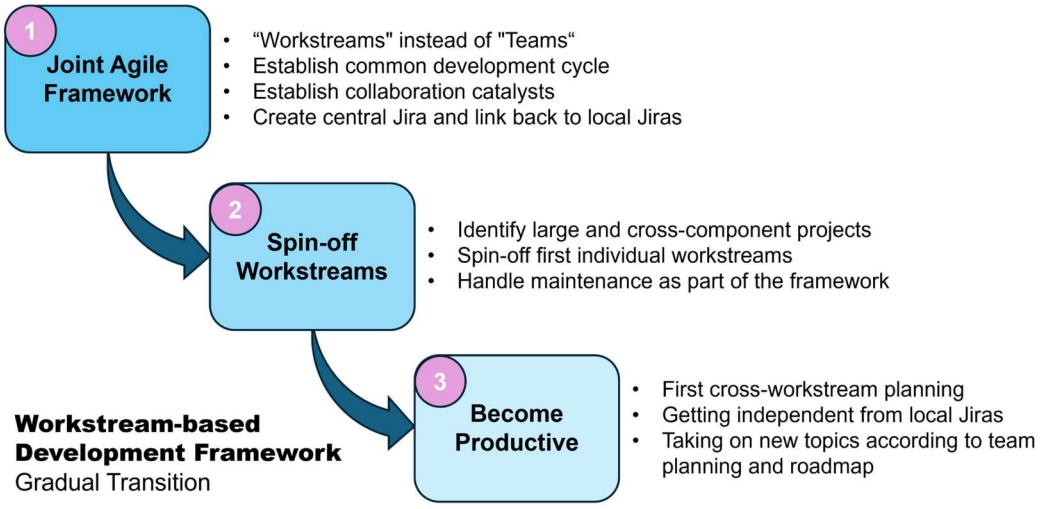

Figure 26: Three steps to gradually transition from Scrum to WDF

Based on the experience made, spinning off workstreams that do not correspond to the historical team setup, is the most disruptive step in the transition. Each team member should have regular check-ins with their manager or the Collaboration Catalyst to prevent a regressive spirit in the team at this stage. Even beyond the transition phase, the goal should be to facilitate the right dose of disruption when introducing new workstreams. New workstreams present a natural opportunity for a change in team dynamics, and recent studies indicate that certain challenges and tensions within the team may increase its resilience and success.[115]

4.3 Use of Collaboration Technology

Since the rise of New Work and the beginning of the COVID pandemic as a catalyst, hybrid working has become the predominant working model in information technology companies and other businesses. Teams no longer permanently work in collocation, but individuals leverage the given autonomy of workspace and work time, as depicted in Figure 15. This change in work culture makes it imperative to support employees and teams with digital and asynchronous communication and collaboration technology in addition to traditional, yet still relevant, face-to-face communication. This section outlines selected digital technology and tooling that

[115] see (Grass, et al., 2021, p. 8)

has proven effective in supporting the conceptualization and, eventually, the running of WDF in hybrid or remote working environments.

4.3.1 Direct Communication

In recent years, Slack and Microsoft Teams have become popular collaboration platforms, replacing solutions such as enterprise instant messaging, desk phones, videotelephony, enterprise wikis, and forums. Microsoft Teams is especially widespread in large enterprises due to its tight integration with the Microsoft Office suite and SharePoint.[116]

In addition to enabling direct communication between individuals via chat, audio, or video, Microsoft Teams organizes group discussions in *Teams* and *Channels*. Each team consists of several channels that are either public to each team member or private.

Collaboration platforms may easily create information-flooding and distraction for team members if they have not been properly introduced. To avoid such situations, agile teams should explicitly discuss and consent to usage practices for the collaboration platform. Some teams may prefer to use emojis and animated GIFs to add a personal note to written team discussions, whereas other teams may want to prohibit their usage to maintain a professional and more focused environment.

The reference team at SAP has chosen to implement these lightweight practices:

- **Use of threads**
  Threads add a contextual dimension to team chats. Instead of only having messages sorted by the time dimension, each message can be converted to a contextual thread by directly replying to it. Threads enable multiple parallel written discussions within the same team channel.
- **Concise use of mentions**
  As multiple parallel team conversations will create an amount of information that is hard for individuals to digest, no team member shall be expected to keep up with each thread. If it is required that a certain individual or group

[116] refer to (European Commission, 2024) for a more controversial view on the integration

reads a message, the author needs to explicitly tag this person or group. Tagging larger groups without good reason is unwelcome behavior.

- **Leverage OneDrive integration**
  Microsoft Teams automatically offers file storage backed by Microsoft SharePoint and integrated into Microsoft OneDrive for Business. Each channel has its own storage, which the team may leverage as shared persistence for documents. This integration replaces traditional shared folders on servers and increases the ease of collaboratively working on documents.
- **Actively manage channel visibility**
  Full transparency across team activities may lead to many visible topic channels within a team. Each team member needs to consciously decide on the relevant channels and should take the time to hide irrelevant channels from their overview list. This helps focus on important information while keeping less relevant information available on demand.
- **Prefer messages over email**
  The availability of collaboration platforms like Microsoft Teams, in conjunction with established communication channels like email, might hinder the user adoption of the former because the continued presence of email as a fallback option reduces the perceived necessity of transitioning to the new platform. The team explicitly agreed to favor direct or channel messages over email and to restrict the use of emails to team-external communication.

With its integration into Microsoft Outlook and its calendar component in particular, Microsoft Teams is also used to schedule remote or hybrid meetings. The reference team chose to set up one channel per technical component and per workstream, as well as one organizational channel for the entire team. Meetings would be scheduled within the respective channels, depending on the required audience. Recordings, notes, and chat messages that are part of the meeting would automatically be archived within the channel. Over time, this leads to a searchable team knowledge base that is transparent and accessible to each member.

4.3.2 Whiteboarding

Especially for open team discussions, as it frequently happens during the conceptualization phase, and for discussions on continuous improvements, a digital whiteboarding solution is required to capture the ideas and motions created by the team. Various solutions are available, with Mural, Miro, and Microsoft Whiteboard being three of the more popular ones. An exemplary whiteboard in the software Mural by vendor Tactivos can be seen in Figure 27.

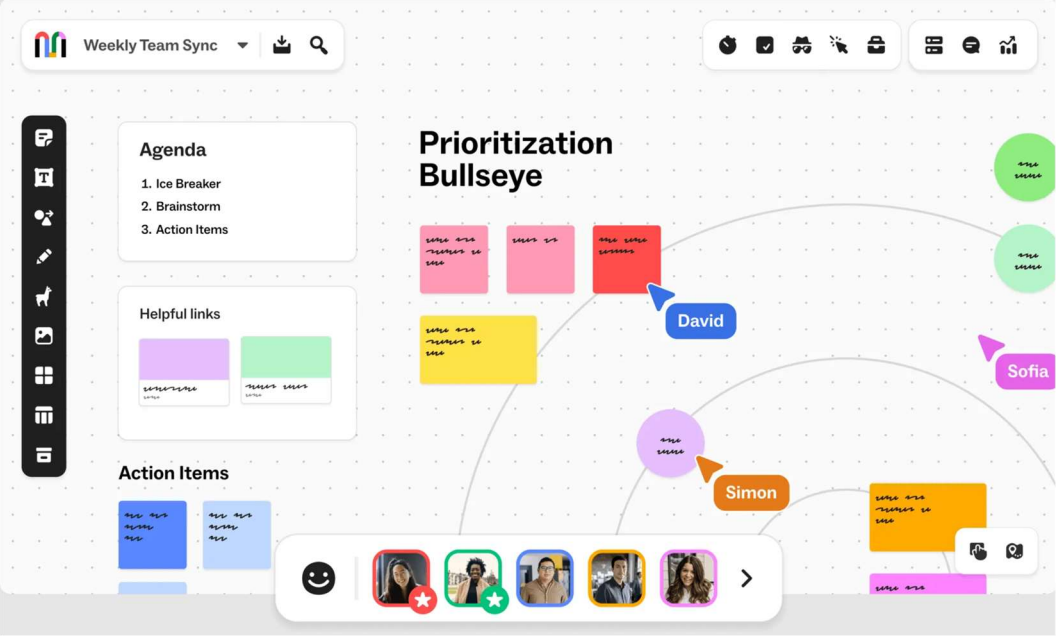

Figure 27: Digital whiteboards like Mural facilitate vibrant virtual team discussions[117]

Digital whiteboards are a one-to-one substitute for physical whiteboards and sticky notes in collocated work environments. The vendors and respective user communities typically provide a variety of preconfigured templates for different meeting types. Agile development templates such as whiteboards for retrospective meetings can be re-used for WDF.

Using preconfigured whiteboards requires thorough preparation to learn how to use the template or set up the preconfiguration by hand. While this is appropriate for formal meetings with a clear agenda, the potential of digital whiteboards should

[117] taken from (Tactivos, Inc., 2024)

also be leveraged for ad-hoc discussions. Ideally, opening a digital whiteboard during a team discussion should be as seamless as picking up a pen and starting to draw in a physical meeting. From a technical perspective, the barrier of opening a blank whiteboard can be lowered by setting up a folder (i.e., “Room” in software Mural) with full access for each team member. The link to this folder will be listed as part of the knowledge base within the primary collaboration platform.

Exemplary outputs of whiteboarding sessions during the conceptualization of WDF can be seen in Figure 23, Figure 24, and Figure 30.

4.3.3 Task Management

Whether working remotely or in collocation, many teams choose to use a digital project and task management system, with Jira by vendor Atlassian being a prominent option. As Jira supports agile project management within the Kanban methodology, it is also suitable for WDF.

For WDF, it is recommended to set up a hierarchy of items divided into *epics*, *user stories*, and *tasks*. In analogy to the hierarchy introduced in Subsection 3.2.2, epics correspond to workstreams, and each epic holds several user stories, which again contain several tasks. Technical components, as a second planning dimension, are covered by the default field *component*. Using those default fields enables the team to use the default Kanban reporting, as seen in Figure 9, without setting up tracking indicators manually. Furthermore, the team benefits from the default planning perspective, which is split into epics and, subsequently, workstreams. With this planning perspective, the backlog can be viewed and prioritized across workstreams and for each individual workstream.

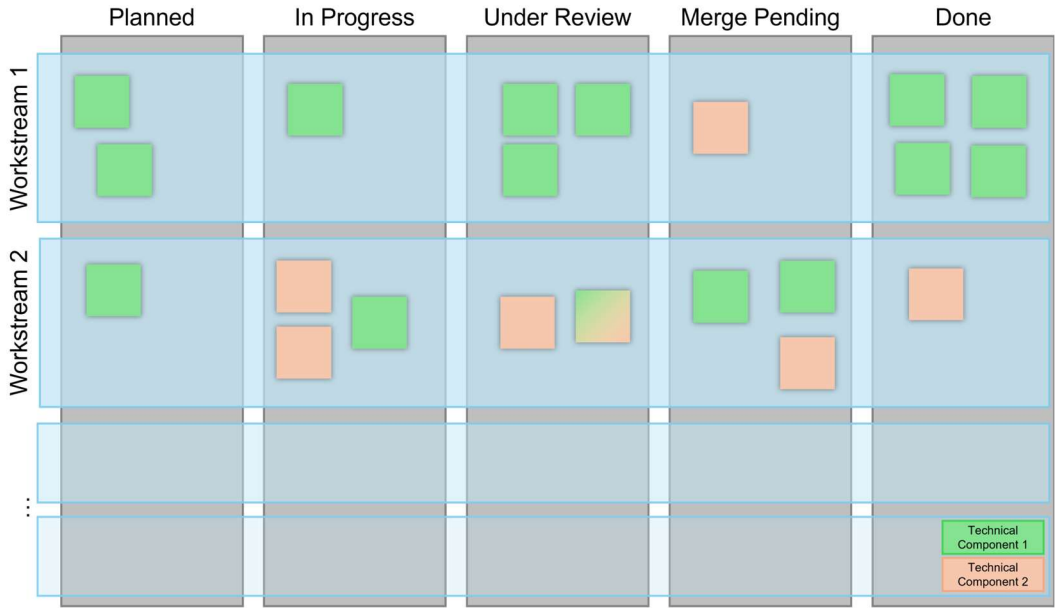

Figure 28: Kanban board incorporating process steps, workstreams, and technical components

The built-in Kanban board should be used to track daily progress on task level. The individual process steps should be defined based on the team's requirements. The reference team agreed to use horizontal "swimlanes" to differentiate between workstreams and to create columns for each of their relevant development steps:

- **Planned**: Planned tasks that are ready to be picked up by developers during the iteration
- **In progress**: Tasks that are currently assigned and under active development.
- **Under review**: Functionally finished tasks that undergo the peer review process.
- **Merge pending**: Coding that has been submitted to the repository and is undergoing the organization's continuous integration process.
- **Done**: Finished tasks that will be delivered with the next release.

The status *planned* flags the transition of an item between the general backlog and the iteration backlog. Figure 29 shows the planning perspective, where items can be pulled from the general backlog into the *planned* status. The respective items

subsequently appear in the *planned* column of the Kanban board, as shown in Figure 28.

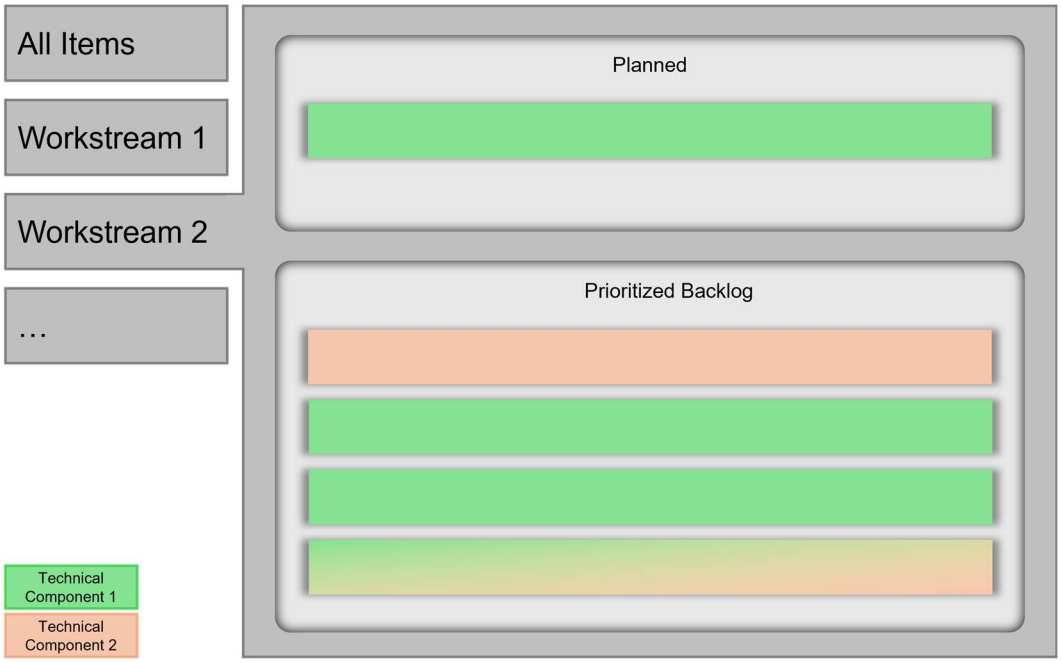

Figure 29: Schematic representation of the default planning perspective in Jira

4.3.4 Scheduling

For scheduling regular meetings and establishing the meeting structure introduced in Subsection 3.2.4, it is beneficial to create a shared team calendar and a shared team inbox for external communication via email. One of the primary email technologies in corporate environments is Microsoft Outlook, which allows the setup of shared mailboxes with a shared calendar by default. Giving all team members access to the shared account enables central and transparent planning of meetings. Only the relevant audience is invited to each meeting, but everyone within the team has on-demand access to get an overview of current team activity.

The exemplary structure of a shared team calendar can be seen in Figure 20. In this example, workstream-related meetings, such as the daily circle, are sent out without a direct link to the meeting itself. Instead, the workstream members use their communication channel in Microsoft Teams to join the meeting. This approach avoids organizational friction and overhead for adjusting recipients of meeting requests when team members change their workstreams between iterations.

As a best practice and to enable effective scheduling within large groups, the team may agree to also make their personal calendars, except for private appointments, accessible to the group. This process adds transparency and helps resolve meeting conflicts before they appear.

4.4 Analysis of Outcomes

Like many other corporate companies, SAP conducts regular employee surveys. The anonymous results of those surveys are given to the teams for internal discussions on measures that need to be taken. Roughly six months after the reference team of twenty developers adopted WDF, the team has been asked two questions as part of the internal discussion of the regular survey results:

1. Mainly thinking of your experiences within the team, is there something that made you particularly proud, happy, or engaged in the last months?
2. Mainly thinking of your experiences within the team, is there something that particularly pulled you down or held you back in the last months?

These questions were not specifically targeted toward the development process but were meant to capture the team situation in general. For this reason, most of the answers touched on general topics. Figure 30 shows answers filtered according to their relevance for assessing the development framework. Based on this anonymous feedback, this section summarizes the main successes and challenges the team experienced within this six-month period.

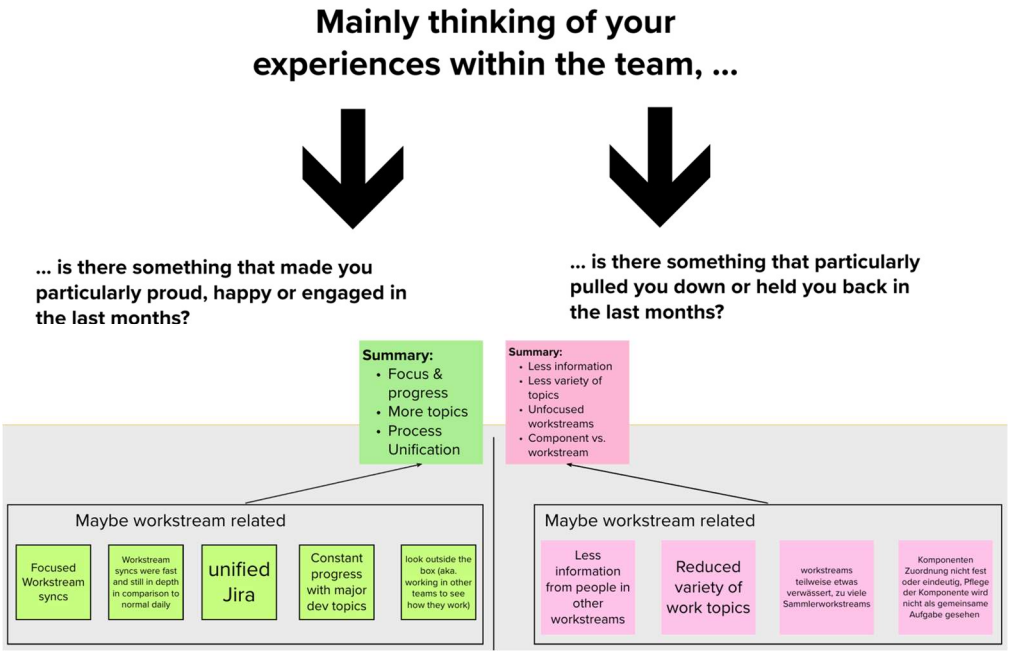

Figure 30: Results of the whiteboarding session on an employee survey filtered for comments that may refer to WDF (actual screenshot)

4.4.1 Successes During Implementation

The whiteboarding results in Figure 30 indicate that the team could realize some benefits of the new development framework after running it for six months. Some of the main successes that the team members highlighted are:

- **Process unification**

  The team experienced benefits immediately after taking the first transitional step of establishing a joint development framework, as outlined in Section 4.2. The process unification and shared Jira instance led to less administrative overhead for maintaining the team's tasks and communicating progress to team-external stakeholders. From an operational perspective, the shared task management and planning process presented the opportunity to reduce redundancy and centralize team-wide tasks. Examples are refactoring and quality initiatives that originate from the organization's central quality team and affect each development team. Historically, one individual per microteam needed to assess the impact on the jointly owned codebase and implement the resulting code changes. In the new setup, the process and shared ownership allows for centrally assigning one developer to take

care of the initiative's effects for the entire team, covering all technical components.

Another aspect to consider is the positive impact of process unification on the onboarding of new employees or temporary staff like students or interns: The transparency on the overall team goals and progress made it possible to centralize onboarding activities while the pool of developers, who have the capacity and motivation to mentor new employees, increased and was not restricted to the respective microteams. The overall onboarding experience and efficiency increased, and temporary staff in particular could easily be routed to the most urgent workstream instead of acting in silos with unbalanced staffing.

- **Increased focus and faster progress**

  Developers enjoyed the increased focus on development topics and the reduced distraction due to context switches. It was also appreciated that teaming up on topics with at least three developers led to more constructive and in-depth discussions, which effectively increased the quality of the output. By focusing on the major workstreams and limiting the scope of individual developers to their assigned workstreams, the team was able to close off two large unfavorable projects with a previous lifetime of two and five years within the first six to nine months after implementation of the framework.

- **Broader variety of topics**

  The ability to participate in workstreams beyond their historical scope of work was positively received by several team members. After the implementation of the framework, some team members permanently changed their scope of work based on staffing requirements and personal interest, while others made use of more frequent team-internal craftsman swaps by changing workstreams at the end of an iteration. This situation led to an overall increased knowledge exchange, consequently making the team more resilient to changing requirements.

In addition to the sentiments expressed, as seen in Figure 30, the team was able to drive the development of two additional technical components. Due to the concept of workstreams, the onboarding of new topics was a non-disruptive experience, as these components were initiated as a workstream staffed with developers with the right skillsets and motivated by contributing to the overall team's success and visibility within the wider organization.

4.4.2 Challenges During Implementation

While the introduction of WDF brought many benefits to the team, it also posed challenges during and after the implementation phase. Some of these challenges from a team member's perspective are listed below:

- **Less variety of topics**
  Some team members reported dissatisfaction that the variety of topics they worked on was reduced after adopting WDF. This view, which must be judged against the background of the team member's initial situation, does not necessarily contradict the opposing statement in Subsection 4.4.1. While some team members enjoy the opportunity to engage with workstreams far beyond their historical scope of work, others may feel restricted by being unable to take on additional work outside the scope of their workstream during the course of an iteration. What appears to be a downside for them, helps the overall team stay focused on large, unfavorable topics and avoid procrastination effects, as described in Subsection 3.3.3. A countermeasure for increasing the motivation of affected team members is to clearly communicate the value of their work to the success of the team and to motivate them to use retrospective meetings for discussing their individual situation and finding a solution within the sovereignty of the team.
- **Less information outside of own workstream**
  In Subsection 3.2.5.1, it was emphasized that one of the goals of WDF is to provide a high level of transparency regarding ongoing topics amongst the team. At the same time, with WDF, the team transitions from three microteams to five or six workstream units, which results in some people not

participating in certain topic discussions as much as they used to. While this enables quick decision-making and keeps detailed discussions efficient, which is appreciated by others, it is vital that regular review meetings are leveraged to summarize the most important discussions and findings and provide the right level of transparency to the team. The recommendations given in Subsection 3.2.5.1 are an iterative improvement that is a direct outcome of the feedback given as part of the employee survey.

- **Unfocused workstreams**
  Workstreams, as defined in Subsection 3.2.1, are not necessarily projects with a fixed timeline and definition of done. A workstream may have a duration between six months and multiple years. Towards the end of the lifetime of a workstream, it may still require work to be done, but with fewer resources involved. If there is not enough work to occupy at least three developers, it ideally needs to be paused until enough work piles up, or, as a second option, it needs to be merged with other workstreams facing similar challenges. Finding a common goal and vision for the merged workstream is challenging in this situation. Team members working in these merged workstreams with less focus on a common goal report lower satisfaction with the process than those working towards a common goal. The severity of this effect depends on the development team's individual environment.
- **Tension between component and workstream assignment**
  Multi-dimensional planning incorporating workstreams and technical components was difficult for individuals to digest at the beginning of framework adoption. The work split between the workstreams and the maintenance responsibilities on the component level created tension as some team members avoided the required maintenance portion. The most visible effect of this tension was an increased bug backlog. Consequently, the team decided to use agenda templates for the regular component circles to avoid missing out on important discussions and to ensure the clear assignment of bugs to developers.

To summarize, individual and team challenges need to be differentiated. Individual challenges are sometimes caused by framework artifacts that provide value to the overall team. Given a solid base of trust among team members, the framework provides the flexibility to discuss and tackle these challenges as part of the iterative improvement process. The bi-weekly retrospective meeting acts as a team-internal safe space to address these challenges.

5. Discussion

After proposing a novel approach to software development, the Workstream-based Development Framework (WDF), in Chapter 3 and summarizing the experience of the prototypical implementation in Chapter 4, this chapter links back to the agile development frameworks introduced in Chapter 2. The discussion concludes with a summary of known limitations and potential future improvements for WDF.

5.1 Comparison with Existing Agile Frameworks

The development framework introduced in this work adopts practices and artifacts from existing frameworks like Scrum, Kanban, or Scrumban. However, it also includes unique concepts such as the matrix organization of work by workstreams and technical components.

Like Boeg's statement that "none of the principles of Kanban restrict you from doing Scrum,"[118] the proposed framework can be embedded into the Kanban methodology. Due to its embedding, teams have the visualization and process health measurements of Kanban at their disposal. However, WDF does not explicitly limit work in progress, which is a central element of Kanban implementations.[119] By restricting developers to focus on a single workstream, the framework reduces parallelization, which is an indirect measure to limit the work in progress. This approach corresponds to setting multi-tasking limits as a transitional step in Scrumban.[120]

Additional factors suggest that WDF may be similar to Scrumban: WDF also leverages late binding of tasks[121] and does not consider estimations. The team at SAP used the status *planned* for development tasks that are ready to be consumed during an iteration, which essentially is the same concept as the ready queue in Scrumban. However, unlike the Scrumban approach, WDF does not introduce further inter-process buffers and does not mandate, but also does not prohibit, defining work standards for the team's process steps.

[118] (Boeg, 2012, p. 17)
[119] see Subsection 2.2.2
[120] see Subsection 2.2.3
[121] see Subsection 3.2.3.3

Beyond the scope of Scrumban, which is founded on agile principles, WDF adds a strong notion of the Software Craftsmanship principles to the development process. With its perspective on the team as a self-responsible collective of entrepreneurial software professionals and its ongoing encouragement of team-internal craftsman swaps, WDF appears as an evolution of Scrumban into the Software Craftsmanship movement. As Ladas describes Scrumban as a transition strategy rather than a framework,[122] WDF can be seen as a continuation of those ideas rather than an alternative or contradiction.

Another key point is how WDF relates to the most commonly implemented agile framework, Scrum. WDF specifically addresses two points of criticism of Scrum highlighted in Section 2.3:

1. The missing comprehension of enterprise realities[123]
2. The meeting overhead commonly experienced by developers[124]

The proposed framework was inspired by the perceived enterprise reality of increased team sizes and the merger of teams due to corporate guidelines.[125] With the framework's unique concept of workstreams, the practicing teams may rebalance staffing non-disruptively, which relaxes the strict assumption of stable teams on a micro level.

Due to the costly training and certification process of a Scrum Master, certified individuals tend to emphasize their knowledge of processes and the associated tooling within the Scrum framework,[126] which creates a tension to the underlying agile principle to value "individuals and interactions over processes and tools."[127] WDF relaxes this tension by generally strengthening the role of developers in the development process and particularly prioritizing their critical view on the ratio between actual working time spent and the experienced overhead of meeting time and process orchestration.

[122] see Subsection 2.2.3
[123] see Figure 11
[124] see Figure 13
[125] see Section 3.1
[126] see Section 2.3
[127] see Subsection 2.1.2

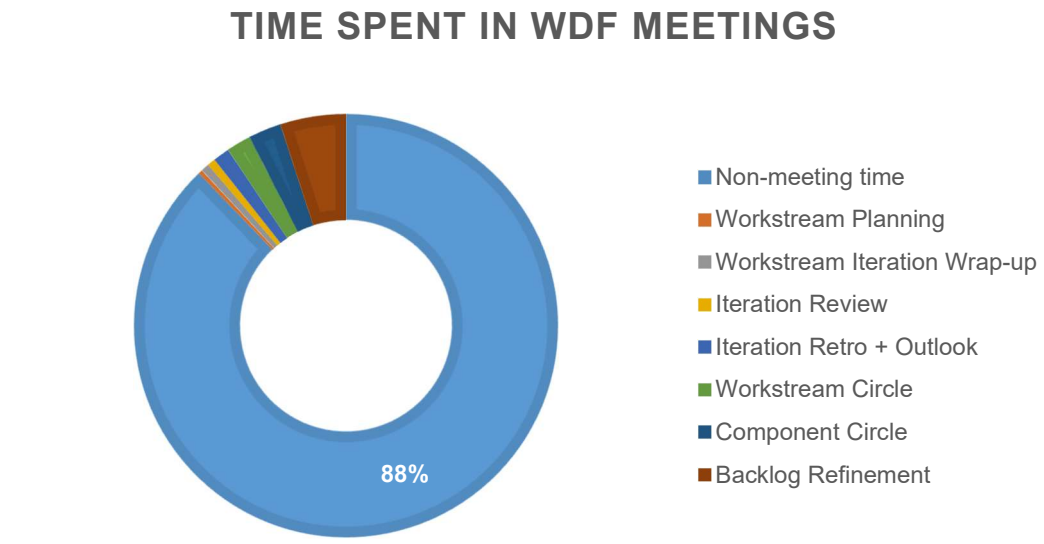

Figure 31: Relative distribution of meeting and non-meeting times for WDF

In analogy to Figure 13, the relative distribution of meeting and non-meeting time for WDF, based on the assumptions in Figure 19, can be seen in Figure 31. It is also assumed that backlog refinement consumes 5% of the iteration time. Since refinement happens primarily at the workstream level and user stories and development tasks are refined at the operational level, the effort is expected to be on the lower end of Scrum's estimation.

While WDF suggests more meetings due to the different planning layers and dimensions, the overall meeting and process orchestration time has almost been cut by half compared to the classic Scrum approach. Overall, the proposed framework largely contradicts the strict interpretation of Scrum (Type A) while incorporating some of its more valuable practices, including retrospective meetings.

WDF can also be seen as an approach to scaling agile development to a larger group, which in this case is a large team with diverse responsibilities. Taking this perspective, WDF can be compared with the scaling frameworks summarized in Subsection 2.2.4. Due to its small scale, the comparison to LeSS or Nexus is explicit. WDF shares the concept of working on a common backlog in synchronized iterations, which is a mandatory prerequisite for prioritizing work across separate topics or teams. However, it avoids the overhead of having an integration team, as

proposed by Nexus, by incorporating integration into the planning hierarchy. Unlike LeSS, WDF tries to make use traditional corporate roles, such as the Team Lead, instead of eliminating them, which is expected to increase its acceptance in large enterprises.

Given its embedding in Kanban and similarities to Scrumban, WDF undeniably follows the lean-agile mindset and the associated principles. Chapter 4 demonstrated how its implementation harmonizes with New Work principles. Autonomy of the workplace and work time can be incorporated by proper use of collaboration technology, as described in Section 4.3. Figure 15 highlights self-organizing teams and an open culture of failure as common New Work artifacts, both of which are strongly encouraged by practicing WDF.

Finally, the framework's central aspect of collective decision-making and seeing developers as entrepreneurial craftsmen within their company is reflected by the New Work principles of "self-responsibility" and "development."

5.2 Limitations and Areas for Future Improvement

The prototypical implementation illustrated in Chapter 4 has been successful because WDF has been conceptualized by the prototyping team who, subsequently, have tailormade the framework to their specific situation and challenges. As agile development methods are primarily based on empiricism,[128] the generic usefulness of this novel approach cannot easily be predicted or extrapolated for other teams. Empiric follow-up research is required to determine whether the specific pain points WDF addresses are of general interest to other software development teams.

The aspect of multi-dimensional planning adds a level of complexity that is not expected to be suitable for small teams in small development organizations. To realize the benefits and justify the increased complexity, the team should have ownership of more than two technical components. Assuming each technical component pursues one or two major development themes, which require at least three developers, the minimum team size is 15 developers. For teams and organizations with more than 30 developers, the large-scale agile frameworks, as introduced in

[128] see Chapter 2

Subsection 2.2.4, are more capable of handling the operational complexity. This situation effectively limits the ideal team size for adopting WDF to ones with between 15 and 30 software developers responsible for up to five technical components.

WDF is a framework for breaking up technical silos—that is, moving away from technical component teams to ones with broader responsibilities and increased flexibility. To get to this state, the teams need an open failure culture and a high willingness to cooperate as prerequisites. This inherently corresponds to a generative team culture, as shown in Figure 2. With its emphasis on self-independent entrepreneurial teams, WDF is unsuitable for implementation in pathological or exclusively bureaucratic organizational cultures, as neither the teams nor the organizations have the surrounding conditions to realize its benefits.

From a practical perspective, software developers are said to be fact-based thinkers willing to collaborate if the collaboration solves an actual problem. A meaningful common vision for the team is a catalyst for collaboration. In the selection process for candidate teams implementing WDF, a focus should be placed on consolidating thematically aligned teams rather than arbitrary groupings within the organization. This approach fosters the potential for cross-component collaboration by leveraging pre-existing synergies between teams contributing to similar non-technical development themes.

One of the insights derived in Subsection 4.4.2 was that developers' satisfaction with WDF corresponds to the workstream they are part of. The satisfaction rate in focused and concise workstreams has typically been high, while the satisfaction in less focused workstreams has been rated lower. The combination of microteams according to their potential for technical cross-component work can mitigate this effect while not completely eliminating it. The existence of unfocused workstreams, where a common goal is hard to define, is the most evident working point for future improvements of WDF.

6. Conclusion

Large software corporations adopt agile development frameworks to act on customer requirements with agility while implementing lean processes for improved margins. On the team level, developers' agile self-conception may contradict the management team's more lean-oriented thinking. The Workstream-based Development Framework (WDF) relaxes this tension by better embedding the developers' perspective into lean and margin-oriented management principles at the corporate level. In addition to the lean and agile principles, the framework makes reference to the principles of New Work and Software Craftsmanship to increase employee satisfaction and intrinsic product quality—both of which have increased relevance for sustainably developing and operating cloud software.

As with other agile frameworks and methodologies, WDF is based on experience and has been proven to work for at least one professional software development team at one point in time. It is not necessarily given that the same framework will work for other teams, nor is it guaranteed to work for the same team in the future. However, based on the fundamentals introduced in Chapter 2, it can be expected that the framework's mechanisms, such as multi-dimensional planning and frequent craftsman swaps, provide value to a broader community of software development experts.

The preconditions for implementing WDF, as outlined in Section 5.2, effectively limit the framework's scope to large teams of between 15 and 30 developers responsible for two to five technical components. The framework's success is encouraged by a generative company culture and a common product vision across the team, which prevents the existence of unfocused workstreams, which have been identified as the main diminisher of individual motivation. The detailed artifacts of WDF were extensively described in Chapter 3 and briefly summarized in Section 3.4.

If WDF were a product, its customers would be software developers, company management, and agile practitioners within companies. Figure 32 shows the main aspects and insights discussed in this thesis transferred into a Business Model Canvas.

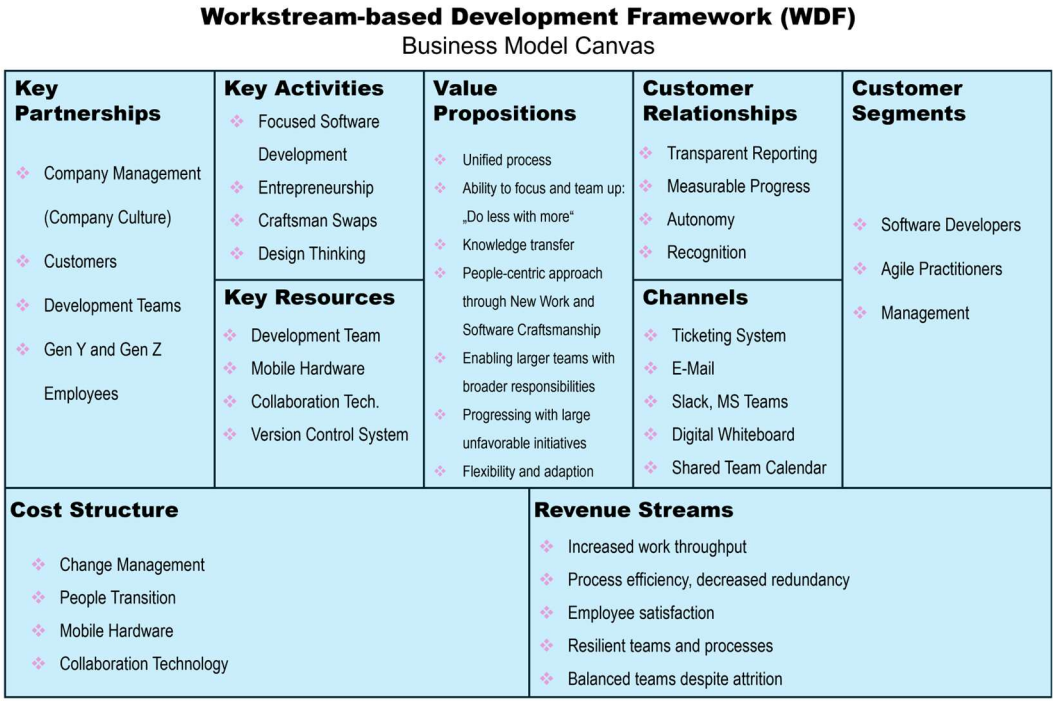

Figure 32: Business Model Canvas, assuming WDF as a product

The company management team, customers, development teams, and especially their Generation Y and Generation Z employees are affected by the introduction of WDF. From a management perspective, the expected improvement in reporting and progress measurement is appealing, whereas developers may benefit from increased autonomy and internal recognition due to visible and exposed roles, such as Workstream Lead, within the development process.

On the technical side, an effective framework implementation in a hybrid work environment requires proper mobile hardware and the respective software licenses for digital collaboration software, such as communication platforms, whiteboarding software, ticketing systems, and more. Thus, collaboration technology is one of the cost drivers next to the costs associated with change management and the people transition process.

The financial gain directly or indirectly caused by the new development process is expected to exceed its cost structure. Contributing factors are increased work throughput, decreased redundancy amongst the teams, and an optimized balance of available human resources according to business priorities.

Compared to other agile frameworks, WDF provides value by unifying processes across large teams and establishing a culture of transparency and ongoing knowledge sharing. The focused and people-centric approach is expected to increase teams' flexibility and resilience and make them capable of effectively managing a broader variety of topics.

In his "Remarks on the Original Scrumban Essay," Ladas states that "Scrumban was never intended to be a defined, prescriptive process" but a "demonstration of a way of thinking about cooperative knowledge work."[129] As a close relative of Scrumban, as per Section 5.1, the same may be true of WDF. Depending on the level of future external adoption, WDF may retrospectively be categorized as a demonstration of applying agile principles or as a defined framework for replication across several teams and companies.

Irrespective of the actual outcome, the external adoption, and the future use of WDF, the team at SAP benefited from the explicit team discussions on the development process. From this perspective, it can be concluded that the current framework, at the very least, is a suitable intermediate step to a less wasteful process in the terminology of Kanban.

[129] (Ladas, 2021)

Bibliography

Almeida, F. & Espinheira, E., 2021. Large-Scale Agile Frameworks: A Comparative Review. *Journal of Applied Sciences, Management and Engineering Technology,* 2(1), pp. 16-29.

Amazon Web Services, 2023. *AWS Whitepaper: Introduction to DevOps on AWS,* s.l.: Amazon Web Services, Inc..

American Psychological Association, 2006. *Multitasking: Switching costs.* [Online] Available at: https://www.apa.org/topics/research/multitasking [Accessed 14 July 2024].

Anderson, D. J., 2010. *Kanban: Successful Evolutionary Change for Your Technology Business.* Sequim: Blue Hole Press.

Bach, T. et al., 2022. Testing Very Large Database Management Systems: The Case of SAP. *Datenbank Spektrum*, 2 June, pp. 195-215.

Beck, K. et al., 2001. *Manifesto for Agile Software Development.* [Online] Available at: https://agilemanifesto.org/ [Accessed 14 July 2024].

Bergmann, F., 2004. *Neue Arbeit, Neue Kultur.* Freiamt: Arbor Verlag.

Boeg, J., 2012. *Priming Kanban - A 10 step guide to optimizing flow in your software development system.* 2nd Edition ed. Aarhus: Trifork A/S.

Bria, M., 2008. *Craftsmanship - the Fifth Agile Manifesto Value?.* [Online] Available at: https://www.infoq.com/news/2008/08/manifesto-fifth-craftsmanship/ [Accessed 14 July 2024].

Cambridge Dictionary, 2024. *Meaning of collective in English.* [Online] Available at: https://dictionary.cambridge.org/dictionary/english/collective [Accessed 14 July 2024].

Conboy, K. & Carroll, N., 2019. Implementing Large-Scale Agile Frameworks: Challenges and Recommendations. *IEEE Software,* 36(2), pp. 44-50.

Dam, R. F. & Siang, T. Y., 2024. *What is Design Thinking and Why Is It So Popular?.* [Online]
Available at: https://www.interaction-design.org/literature/article/what-is-design-thinking-and-why-is-it-so-popular
[Accessed 14 July 2024].

Diepstraten, M., 2022. *The Role of Procrastination in Agile IT Projects.* s.l., Open University of the Netherlands.

Digital Template Market, 2020. *Why Agile is important for Software Development.* [Online]
Available at: https://digitaltemplatemarket.com/agile-important-software-development/
[Accessed 14 July 2024].

European Commission, 2024. *Commission sends Statement of Objections to Microsoft over possibly abusive tying practices regarding Teams.* [Online]
Available at: https://ec.europa.eu/commission/presscorner/detail/en/ip_24_3446
[Accessed 14 July 2024].

Fath, R., 2018. *Unsplash.* [Online]
Available at: https://unsplash.com/photos/people-building-structure-during-daytime-ymf4_9Y9S_A
[Accessed 14 July 2024].

Grass, A., Backmann, J. & Hoegl, M., 2021. *Research Project: Success Factors for Agile Team Collaboration.* Munich, Institute for Leadership and Organization, LMU Munich.

Hewlett Packard Enterprise, 2017. *Agile is the new normal: Adopting Agile project management,* s.l.: Hewlett Packard Enterprise Development LP.

Humble, J., Molesky, J. & O'Reilly, B., 2020. *Lean Enterprise - How High Performance Organizations Innovate at Scale.* First Release ed. Sebastopol: O'Reilly Media, Inc..

Koch, J., Drazic, I. & Schermuly, C. C., 2023. The affective, behavioural and cognitive outcomes of agile project management: A preliminary meta-analysis. *Journal of Occupational and Organizational Psychology*, 22 February, 96(3), pp. 678-706.

Ladas, C., 2008. *Scrumban - Essays on Kanban Systems for Lean Software Development.* Seattle: Modus Cooperandi Press.

Ladas, C., 2021. *Remarks on the Original Scrumban Essay.* [Online] Available at: https://www.agilealliance.org/remarks-on-the-original-scrumban-essay/ [Accessed 14 July 2024].

Leffingwell, D., 2023. *Say Hello to SAFe 6.0!.* [Online] Available at: https://scaledagileframework.com/blog/say-hello-to-safe-6-0/ [Accessed 14 July 2024].

Liker, J. K., 2003. *The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer.* Reissue Edition ed. New York: McGraw-Hill Professional.

Lucena, P. & Tizzei, L. P., 2016. Applying Software Craftsmanship Practices to a Scrum Project: an Experience Report. *arXiv.org*, 17 November.

Oppermann, A., 2023. *What Is the V-Model in Software Development?.* [Online] Available at: https://builtin.com/software-engineering-perspectives/v-model [Accessed 14 July 2024].

Quino Al, 2017. *Unsplash.* [Online] Available at: https://unsplash.com/photos/grayscale-photography-of-men-playing-rugby-on-muddy-land-ce79TRf3Fyw [Accessed 14 July 2024].

Rauch, N., 2016. *Craftsman Swap and Journeyman Tour.* [Online] Available at: https://www.nicole-rauch.de/posts/2016-08-29-craftsman-swap-and-journeyman-tour.html [Accessed 14 July 2024].

Scaled Agile, Inc., 2024. *Lean-Agile Mindset.* [Online]
Available at: https://scaledagileframework.com/lean-agile-mindset/
[Accessed 14 July 2024].

Schell, V., 2019. *Agile Skalierungsframeworks: Safe, Less und Nexus im Vergleich.* [Online]
Available at: https://t3n.de/news/agile-skalierungsframeworks-safe-less-nexus-1150190/
[Accessed 14 July 2024].

Schermuly, C. C. & Meifert, M., 2022a. Auf dem Weg ins postagile Zeitalter?. *Personalmagazin*, September, pp. 24-30.

Schermuly, C. C. & Meifert, M., 2022b. *Ergebnisbericht zum New Work-Barometer,* Berlin: SRH Berlin University of Applied Sciences.

Schwaber, K. & Sutherland, J., 2020. *The Scrum Guide.* [Online]
Available at: https://scrumguides.org/
[Accessed 14 July 2024].

Scrum.org, 2024. *What is Scrum?.* [Online]
Available at: https://www.scrum.org/resources/what-scrum-module
[Accessed 14 July 2024].

Scrum.org, 2024. *What is ScrumBut?.* [Online]
Available at: https://www.scrum.org/resources/what-scrumbut
[Accessed 14 July 2024].

Shaffer, C. A. & Kazerouni, A. M., 2021. The Impact of Programming Project Milestones on Procrastination, Project Outcomes, and Course Outcomes: A Quasi-Experimental Study in a Third-Year Data Structures Course. *SIGCSE '21: Proceedings of the 52nd ACM Technical Symposium on Computer Science Education,* pp. 907-913.

Statista, 2024. *Number of software developers worldwide in 2018 to 2024.* [Online]
Available at: https://www.statista.com/statistics/627312/worldwide-developer-

population/
[Accessed 14 July 2024].

Sutherland, J., 2005. *Scrum II: Better, Faster, Cooler!,* s.l.: s.n.

Sutherland, J., 2010. *Scrum Handbook.* Somerville: Scrum Training Institute.

Tactivos, Inc., 2024. *Mural - Visual collaboration made easy.* [Online]
Available at: https://www.mural.co/features
[Accessed 14 July 2024].

the undersigned, 2009. [Online]
Available at: https://manifesto.softwarecraftsmanship.org/
[Accessed 14 July 2024].

Tregubov, A., Boehm, B., Rodchenko, N. & Lane, J. A., 2017. Impact of Task Switching and Work Interruptions on Software Development Processes. *ICSSP 2017: Proceedings of the 2017 International Conference on Software and System Process,* July.pp. 134-138.

Väth, M., Soballa, A. & Gstöttner, A., 2019. *New Work Charta.* [Online]
Available at: https://humanfy.de/new-work-charta/
[Accessed 14 July 2024].

Waworuntu, E. C., Kainde, S. J. R. & Mandagi, D. W., 2022. Work-Life Balance, Job Satisfaction and Performance Among Millennial and Gen Z Employees: A Systematic Review. *Society,* 10(2), pp. 286-300.

Westrum, R., 2005. A Typology of Organisational Cultures. *Quality & safety in health care*, January, pp. ii22-ii27.

Womack, J. P. & Jones, D. T., 2003. *Lean Thinking: Banish Waste and Create Wealth in Your Corporation.* 2. ed. New York: Free Press, Simon & Schuster Inc..

Wong, W., 2009. Employee swap gives two firms new perspectives. *Ventura County Star*, 13 July.

World Rugby Limited, 2024. *World Rugby - The scrum.* [Online]
Available at: https://www.world.rugby/the-game/beginners-guide/scrum
[Accessed 14 July 2024].

Declaration of Authenticity

I declare that I completed the master's thesis independently and used only the materials that are listed. All materials used, from published as well as unpublished sources, whether directly quoted or paraphrased, are duly reported.
Furthermore, I declare that the master's thesis, or any abridgement of it, was not used for any other degree-seeking purpose.

Germersheim, 17 July 2024 

Place, Time Signature