STUDENT

Jawad Mehmood Khan Qayyum

20233947

jqayyu23@student.aau.dk

INTERVIEWEE

Ahmed Sadiq

Djøf Trade Union

Asq@djoef.dk

EXAM

AALBORG UNIVERSITY

Fall 2025

Agile Software Engineering

EXAM ASSIGNMENT

PROJECT PERIOD

2025

25

Title
Agile Software Engineering

Project period
2025

Student
Jawad Mehmood Khan Qayyum

Semester
Fall 2025

Number of pages
28

Synopsis

This report analyzes the implementation of Agile principles in practice through an interview-based case study. The purpose is to evaluate how Agile methodologies are applied in real-world software development projects and understand their impact on project success. The analysis focuses on examining development processes, stakeholder management, team dynamics, and continuous improvement strategies within an Agile framework.
Through systematic analysis of interview data and theoretical frameworks, this report identifies key success factors, challenges, and recommendations for improving Agile practices in professional software development environments.

Contents

1 Introduction **1**

2 Recognizing Development Challenges **2**
2.1 Key Challenges 2
2.2 Problem Analysis 3
2.3 Root Causes and Readiness for Change 4

3 Adopting Agile Methodologies **6**
3.1 Establishing Product Backlog Refinement 6
3.2 Introducing Scrum Roles and Sprint Structure 7
3.3 Building Collaboration and Transparency 8
3.4 Addressing Change Resistance 9

4 Measuring Team Progress and Adaptation **10**
4.1 Sprint Reviews and Stakeholder Engagement 10
4.2 Retrospectives and Process Refinement 11
4.3 Tracking Metrics and Leadership Support 11

5 Understanding Stakeholder Voices **13**
5.1 Stakeholder Identification and Collaboration 13
5.2 Managing Competing Interests and Communication 14

6 Maintaining Software Quality **16**
6.1 Automated Testing and Code Reviews 16
6.2 Legacy Systems and Feedback Integration 16

7 Deployment Practices and System Observability **18**
7.1 Current Pipeline and Monitoring Assessment 18
7.2 Test Stability Improvements 19
7.3 Building Deployment Capabilities 19

8 Conclusion **20**
8.1 Emerging Challenges 20
8.2 Future Development 21

Bibliography **22**

A Interview Questions **24**
A.1 Welcome and Introduction 24
A.2 Project Experiences - Success and Challenges 24
A.3 Work Methods and Daily Practice 25
A.4 The Team and Collaboration 25
A.5 Measurement and Continuous Improvement 26
A.6 Quality and Testing 26
A.7 AI in Development 26
A.8 CI/CD and Technical Setup 27
A.9 Stakeholders and the Outside World 27
A.10 Closing 28

1

Introduction

This report analysis is part of the exam assignment in the Agile Software Engineering course at Aalborg University Copenhagen. The purpose is to analyze and evaluate the implementation of Agile principles in practice and understand what they mean for a project's success. To gain this insight, I chose to interview Ahmed Sadiq, a software developer with 10 to 12 years of experience at Djøf Trade Union. Ahmed works in their internal development department, building critical systems such as booking platforms, account management, and APIs for both internal employees and external union members **(Transcript: 00:01:05–00:01:34)**.

It's worth noting that the organization previously struggled with legacy systems and unclear processes, which led them to gradually adopt Scrum methodology. However, they haven't followed Scrum strictly. The team dropped story points because stakeholders found them confusing, and they occasionally skip retrospectives when a sprint lacks meaningful progress **(Transcript: 00:08:58–00:10:22)**. This pragmatic approach reflects broader industry trends where teams adapt Agile frameworks to their specific contexts rather than following them rigidly *(State of Agile Report)*.

This report will examine how they implement Agile practices in their daily work, focusing on their refinement processes, quality assurance approaches, deployment pipeline, and stakeholder management. The analysis will use knowledge from the Agile Software Engineering course to understand how theoretical principles manifest in practice.

2

Recognizing Development Challenges

Before Djøf Trade Union adopted Scrum, their development team faced significant challenges that hindered their ability to deliver value efficiently. Understanding these problems is essential to appreciate why they chose Agile practices and how those practices addressed their specific needs. The Scrum framework emphasizes that transparency enables inspection and adaptation *(Schwaber and Sutherland 2020)*, but achieving transparency first requires identifying what is unclear. Research with 2,000 Scrum teams found that team effectiveness depends heavily on addressing fundamental organizational impediments *(Verwijs and Russo 2021)*.

2.1 Key Challenges

Djøf Trade Union faced a fundamental problem with how work arrived and was prioritized. Tasks would emerge ad-hoc without clear consideration for team capacity or strategic importance, creating an environment where developers constantly reacted to incoming requests rather than working proactively toward meaningful goals.

The situation became particularly problematic when dealing with legacy systems. Ahmed described a project where he was asked to add a simple checkbox to the member registration system for union unemployment insurance sign-up **(Transcript: 00:02:23–00:02:34)**. The legacy membership system had security restrictions that prevented making changes without being logged in as an actual member, and creating test members was not permitted for data protection reasons **(Transcript: 00:03:04–00:03:33)**. He spent two weeks to a month working on

this task, only to have it completely abandoned **(Transcript: 00:04:32–00:04:48)**.

The task should never have reached Ahmed in the first place. It needed approval from the legal department regarding GDPR compliance and coordination with the third-party vendor who controlled the membership system's security settings **(Transcript: 00:04:48–00:05:17)**. There was no gate-keeping process to ensure that tasks were actually achievable within existing constraints before developers invested time in them. This pattern repeated itself across various projects. Developers would receive assignments that looked simple on the surface but contained hidden dependencies, unclear stakeholder expectations, or technical constraints that had not been discovered.

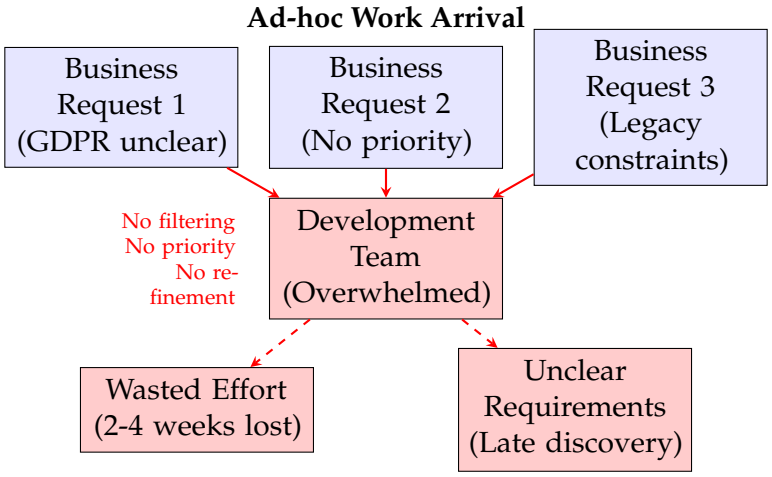

Figure 2.1: *Chaotic work arrival before Scrum adoption. Business requests flowed directly to developers without proper refinement, prioritization, or feasibility analysis, leading to wasted effort and late discovery of blockers.*

2.2 Problem Analysis

The core issue was the absence of structured processes to connect business needs with development capacity. When the business side wanted something built, there was no intermediary to translate those desires into well-defined, technically feasible requirements. Business stakeholders assumed their requests were feasible. Developers accepted tasks without fully understanding the business context or questioning whether all necessary approvals were in place. Both sides operated with incomplete information.

Without a clear backlog or product owner role, everything seemed equally urgent. There was no mechanism to evaluate trade-offs or make conscious

decisions about what should be done first based on business value and technical dependencies. Ahmed's team also struggled with legacy code maintenance: systems built years earlier by developers no longer with the company, with sparse documentation **(Transcript: 00:16:07–00:16:37)**. This hidden complexity was rarely accounted for when estimating work.

2.3 Root Causes and Readiness for Change

The fundamental problem was the absence of empirical process control. Djøf Trade Union made decisions based on assumptions rather than validated information. None of these assumptions were regularly tested against reality until it was too late. Teams didn't have regular checkpoints to demonstrate what had been built and gather feedback. There were no retrospectives where teams could reflect on what went wrong and agree on changes to prevent similar issues in the future.

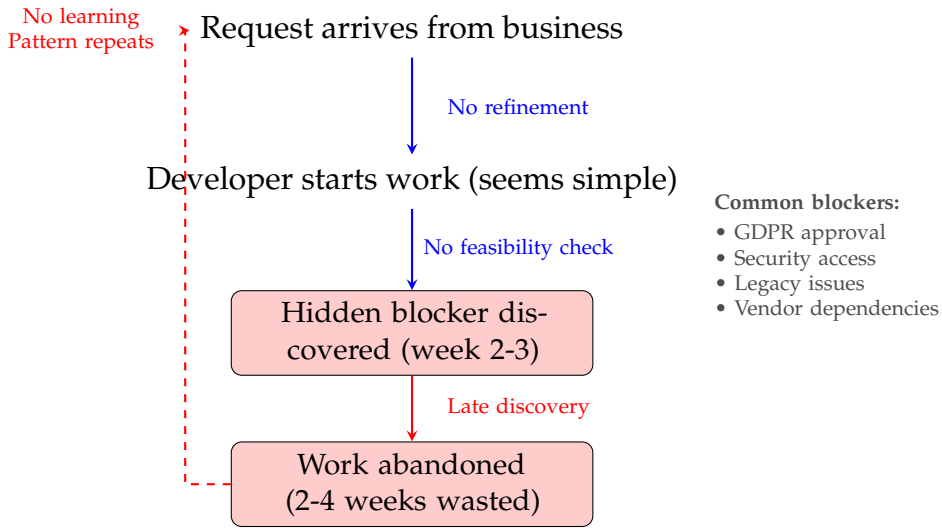

Figure 2.2: *Wasted effort cycle before Scrum adoption. Work requests arrived without refinement or feasibility checks, developers started tasks that seemed simple, hidden blockers emerged late in development, and work was abandoned after 2-4 weeks of effort. Without retrospectives or process changes, this pattern repeated continuously.*

Part of the root cause was also the absence of dedicated roles with clear accountability. No one owned the product backlog or facilitated the development process. They also lacked transparency into their capacity and work in progress. Without a visible backlog or sprint board, stakeholders couldn't see what the team was working on, creating frustration on both sides.

Finally, there was insufficient management support for changing how work was done. Research on Agile transformations demonstrates that management support is the second most critical success factor, creating conditions that enable developer effectiveness and continuous improvement to flourish *(Russo 2021)*. Without executive buy-in and resources dedicated to process improvement, teams had little leverage to push for changes.

These root causes (lack of empirical process control, absence of defined roles, insufficient transparency, and limited management support) combined to create an environment where software development was chaotic and unpredictable. However, recognizing these problems created readiness for change. Failed projects like the membership checkbox task made the costs of their current approach visible to stakeholders. This awareness became the catalyst for exploring structured frameworks - precisely the context that led Djøf Trade Union to explore Scrum as a potential solution.

3

Adopting Agile Methodologies

Djøf Trade Union addressed their development challenges by adopting Scrum practices that introduced structure, transparency, and empirical process control. The Scrum framework emphasizes empiricism through transparency, inspection, and adaptation *(Schwaber and Sutherland 2020)*. Research demonstrates that successful Agile adoption requires teams to adapt frameworks to their organizational realities *(Verwijs and Russo 2021)*. This chapter examines how they implemented Scrum practices to establish predictability and deliver value more consistently.

3.1 Establishing Product Backlog Refinement

The introduction of structured refinement sessions fundamentally changed how work flowed into the development team. Before refinement became routine, tasks arrived ad-hoc without proper analysis of feasibility or business value. The refinement process gave the team a mechanism to break down large initiatives systematically and identify issues before developers invested time. Product Backlog refinement is an ongoing activity where the team adds detail, estimates, and order to backlog items *(Schwaber and Sutherland 2020)*.

Ahmed described how refinement works: before starting any project, the team runs refinement sessions where they break down epics into user stories **(Transcript: 00:05:30–00:06:00)**. The team takes business needs from stakeholders and decomposes them into manageable tasks. This allows them to deliver features incrementally. When the business side has done their preparation well and the team conducts thorough refinement where each user story has a clear purpose, projects tend to succeed **(Transcript: 00:06:00–00:06:24)**.

This practice directly addressed the legacy system problem Ahmed encoun-

tered earlier. The checkbox task that consumed weeks should never have reached the development team without first verifying legal approval and vendor accommodation. Refinement created a gate-keeping process that prevented such situations. They now question assumptions, identify dependencies, and validate feasibility during refinement rather than discovering blockers mid-implementation. The refinement process also improved transparency for stakeholders. When epics broke down into twenty user stories, stakeholders gained realistic insight into the complexity involved.

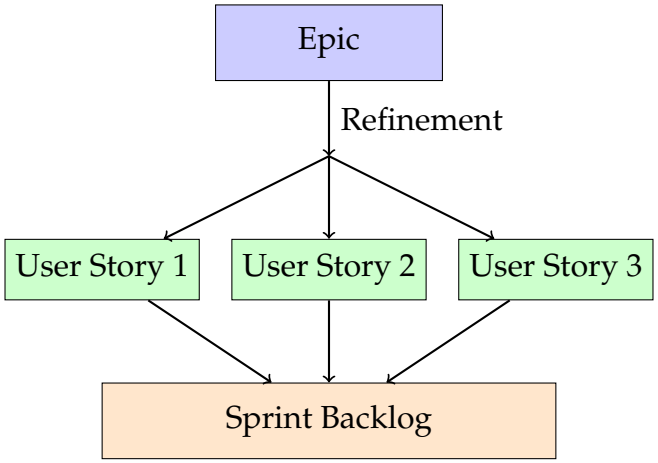

Figure 3.1: *Product Backlog Refinement Process: epics are decomposed into manageable user stories through collaborative refinement sessions.*

3.2 Introducing Scrum Roles and Sprint Structure

The adoption of defined Scrum roles brought accountability and clarity. Scrum defines three accountabilities: Product Owner, Scrum Master, and Developers, each with specific responsibilities for delivering value *(Schwaber and Sutherland 2020)*. The Product Owner became responsible for prioritizing the backlog in collaboration with the business, evaluating which work delivers the most value while considering technical dependencies **(Transcript: 00:14:52–00:15:22)**. This resolved the previous situation where everything seemed equally urgent. The Scrum Master role focused on maintaining the process and removing obstacles **(Transcript: 00:16:52–00:17:22)**.

Djøf Trade Union adopted two-week sprints after experimenting with different lengths. Three-week sprints proved too long, while one-week sprints were too short to deliver meaningful increments **(Transcript: 00:14:07–00:14:37)**.

The two-week cadence provided sufficient time to complete user stories while maintaining rapid feedback cycles.

The team adapted Scrum pragmatically. They dropped story points because the concept proved difficult to communicate consistently *(Russo 2021)*. Scrum Masters experimented with alternative estimation approaches to move away from estimates that stakeholders mistakenly interpreted as time commitments **(Transcript: 00:08:58–00:09:27)**. The team also occasionally skipped retrospectives when a sprint lacked meaningful progress to reflect on **(Transcript: 00:09:59–00:10:22)**, demonstrating their practical approach.

Sprint reviews after each sprint gave the team regular checkpoints to evaluate whether they succeeded and learn from failures **(Transcript: 00:06:48–00:07:31)**. Retrospectives provided space to refine how the team worked together. These ceremonies established inspection and adaptation cycles that were completely absent previously.

3.3 Building Collaboration and Transparency

Scrum practices fundamentally improved transparency. The visible backlog allowed stakeholders to see what the team was working on and understand capacity constraints. When stakeholders wanted new features, they could observe how those requests fit into existing priorities rather than assuming development resources were unlimited.

The sprint structure created predictability. Stakeholders knew when to expect completed features. The flexibility Ahmed mentioned as the best aspect of their approach stems from having a structured process that can accommodate change **(Transcript: 00:15:37–00:15:52)**. When business priorities shift, the team can adjust in the next sprint planning session.

Code review processes and the Azure DevOps pipeline with automated testing provided technical safeguards **(Transcript: 00:12:52–00:13:22)**. Automated tests run before code can merge. A colleague must review and approve the code before it progresses. The deployment pipeline moves code through development, test, and production environments systematically, creating quality gates at each stage. Delivering a Done increment every Sprint reduces the risk of undone work accumulating and disrupting future development *(Verwijs, Overeem, et al. 2020)*.

Collaboration within the team also improved. Developers could focus on defined sprint goals rather than constantly reacting to incoming requests. The Product Owner acted as a buffer, managing stakeholder expectations and protecting the team from excessive interruptions. This allowed developers to work more proactively and build solutions with proper consideration for technical

quality.

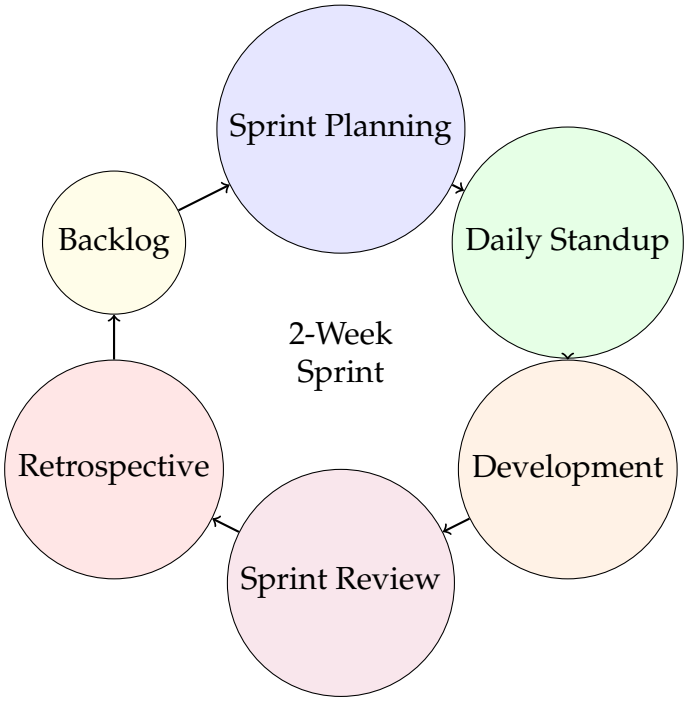

Figure 3.2: *Sprint Cycle: Ahmed's team follows a two-week sprint cadence with regular ceremonies for planning, coordination, review, and continuous improvement.*

3.4 Addressing Change Resistance

Not everyone embraced Scrum immediately. Stakeholders accustomed to making direct requests to developers found the Product Owner intermediary frustrating. Developers who valued autonomy sometimes viewed ceremonies as unnecessary overhead. The organization addressed resistance through demonstrated results. When sprints consistently delivered working software, skeptics recognized the value. Leadership support proved critical. Management allocated time for Scrum ceremonies and protected teams from pressure to bypass the process during urgent situations.

4

Measuring Team Progress and Adaptation

Adopting Agile practices represents only the beginning. Teams must continuously evaluate whether their implementation delivers value and where further improvement is possible. Empiricism requires inspection and adaptation based on transparent information *(Schwaber and Sutherland 2020)*. Research shows that teams demonstrating responsiveness through frequent releases achieve higher stakeholder satisfaction *(Russo, ASE Lecture 3, 2025)*. This chapter examines how Djøf Trade Union measures their Agile effectiveness and evolves their practices.

4.1 Sprint Reviews and Stakeholder Engagement

Djøf Trade Union established regular sprint reviews to evaluate completed work and gather feedback. After each sprint, they review what they accomplished, whether goals were met, and what can be learned **(Transcript: 00:06:48–00:07:31)**. These reviews provide structured moments for reflection rather than pushing forward blindly.

Sprint reviews involve stakeholders from the business side. The team invites interested parties to see what they built, and quarterly demos showcase three months of accumulated progress to broader audiences **(Transcript: 00:20:37–00:21:07)**. This regular demonstration rhythm keeps stakeholders informed without constant interruptions to the development team.

The review process helps the team understand whether their work delivers actual value. When features reach users, the Product Owner and business teams collect feedback and bring it back to development **(Transcript: 00:21:07–00:21:37)**. This creates a feedback loop where user experience influences future

prioritization. Success is not just completing tasks but solving real problems for members and internal users.

4.2 Retrospectives and Process Refinement

Beyond reviewing product increments, Ahmed's team holds retrospectives to examine their working methods. These sessions focus on how the team collaborates and what obstacles prevent smooth workflow **(Transcript: 00:07:31–00:07:48)**. Sprint Retrospectives enable teams to reflect on their process and identify improvements for future iterations *(Russo, ASE Lecture 3, 2025)*.

One concrete improvement emerged from retrospective discussions: code reviews were taking too long because colleagues lacked time to examine pull requests promptly **(Transcript: 00:23:07–00:23:37)**. The team agreed that when someone creates a pull request and announces it in Teams, a reviewer must examine it within four hours **(Transcript: 00:23:37–00:24:07)**. This simple commitment dramatically improved their development flow.

The team's pragmatic adaptation of Scrum also appears in retrospectives. When a sprint lacks meaningful progress, they sometimes skip the retrospective because there is little to reflect upon **(Transcript: 00:09:59–00:10:22)**. This flexibility demonstrates their focus on value rather than ceremony for ceremony's sake.

4.3 Tracking Metrics and Leadership Support

Djøf Trade Union tracks concrete indicators of progress. They measure how many tasks complete per sprint, providing insight into team capacity **(Transcript: 00:21:52–00:22:22)**. They also track lead time from when a task starts until it deploys to production, highlighting bottlenecks in their process.

Quality metrics complement velocity tracking. The team monitors how many bugs surface during development versus how many escape to production **(Transcript: 00:22:22–00:22:52)**. This indicates whether their testing catches issues before users encounter them. Quality assurance practices and clear Definition of Done criteria help teams maintain consistent standards *(Russo, ASE Lecture 5, 2025)*. High production bug rates signal problems with testing or code review processes.

The organization's measurement approach remains practical rather than exhaustive. They gather enough data to spot trends and identify problems without drowning in metrics. The goal is actionable insight, not comprehensive dashboards. When code review delays appeared in their data and team discussions,

they took concrete action rather than simply documenting the problem.

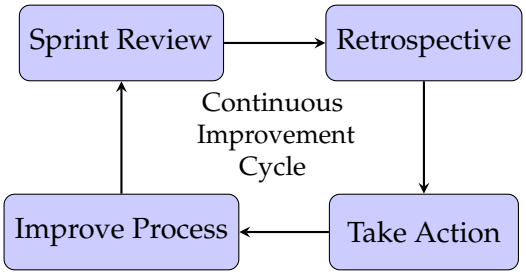

Figure 4.1: *Continuous improvement cycle through regular sprint reviews and retrospectives*

The combination of sprint reviews, retrospectives, and lightweight metrics creates continuous improvement. The organization demonstrates that Agile evaluation doesn't require sophisticated tools or extensive data collection. Regular inspection of both product and process, coupled with willingness to adapt when issues surface, drives meaningful improvement over time.

Management support reinforces this learning orientation **(Transcript: 00:24:22–00:25:22)**, allowing them to invest time in addressing problems rather than only delivering features. Leadership provides resources for skill development and respects process boundaries. When they request tools or training, management evaluates requests seriously. This support creates psychological safety where teams can acknowledge problems without fear, enabling honest retrospectives and genuine improvement.

5

Understanding Stakeholder Voices

Successful Agile implementation depends on understanding and engaging stakeholders effectively. Research with 2,000 Scrum teams found that stakeholder concern is a key driver of team effectiveness *(Verwijs and Russo 2021)*. Djøf Trade Union gradually learned to identify stakeholders and build collaborative relationships that support value delivery.

5.1 Stakeholder Identification and Collaboration

The organization serves two groups: internal employees and union members. Internal employees use systems for booking management and account administration **(Transcript: 00:01:34–00:02:04)**. Union members interact with member-facing systems. These two stakeholder groups create challenges because internal and member needs don't always align.

The Product Owner became the bridge between stakeholder groups and the development team. Before Scrum, tasks arrived chaotically without clear understanding of who needed the work or why. The Product Owner now gathers needs from both groups, translates them into backlog items, and ensures the team understands context **(Transcript: 00:11:22–00:11:52)**.

They recognized that stakeholders have different influence and interest levels. Leadership cares about member growth and operational efficiency. They set budget priorities and decide which initiatives receive funding. Daily system users provide practical feedback about what works and what frustrates them.

Refinement created opportunities for stakeholders to shape solutions before development begins. The team now breaks down epics collaboratively with input from users **(Transcript: 00:05:30–00:06:00)**. When stakeholders participate in defining acceptance criteria, they gain realistic expectations.

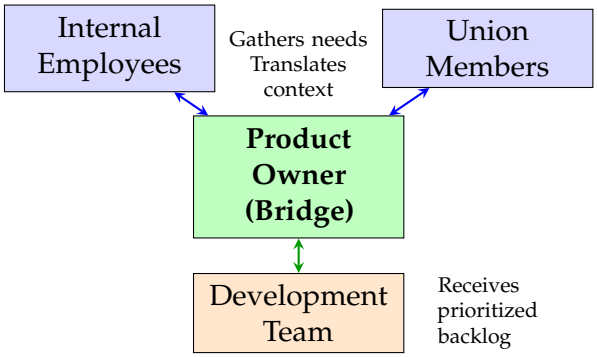

Figure 5.1: *Stakeholder landscape showing two groups served by Ahmed's organization. The Product Owner bridges stakeholder groups and the development team, gathering needs from both internal employees and union members, then translating them into prioritized backlog items.*

Sprint reviews became regular touchpoints where stakeholders see completed work and provide feedback. The team invites business representatives to reviews, demonstrating features and gathering reactions **(Transcript: 00:20:37–00:21:07)**. Quarterly demos expanded this transparency by showcasing three months of progress to broader audiences.

Direct stakeholder-to-developer communication still happens. Ahmed adapted practically: when users approach him with ideas, he listens but redirects them to the Product Owner for prioritization **(Transcript: 00:29:22–00:30:21)**. This protects the team while recognizing valuable ideas.

5.2 Managing Competing Interests and Communication

Stakeholders do not always agree. Economic constraints take priority: when leadership determines something costs too much, the project stops regardless of user enthusiasm **(Transcript: 00:36:17–00:36:47)**. Technical feasibility also matters. When requests would compromise system integrity, developers push back with Product Owner support.

The team learned to find middle ground. Understanding underlying needs often reveals alternative solutions. Successful Agile transformations require balancing multiple influences, with developer skills and management support being critical factors *(Russo 2021)*.

Stakeholder management improved significantly with Agile practices. Structured refinement, regular reviews, transparent backlog, and clear Product Owner ownership created predictability that reduces frustration. Stakeholders know

when to expect features. Management sees progress through transparent metrics rather than status reports that may hide problems.

The team established multiple feedback channels tailored to different stakeholder groups. Members receive newsletters announcing new features and changes, creating awareness without direct development team involvement. Internal stakeholders participate in sprint reviews and can observe the Jira backlog. Leadership receives quarterly progress updates aligned with strategic objectives. These layered communication approaches ensure each stakeholder group receives appropriate information without overwhelming the development team with coordination demands.

6

Maintaining Software Quality

Quality became a shared responsibility as Djøf Trade Union adopted Agile practices. Rather than relying on end-of-cycle testing, they built quality checks into every development step. Tests run automatically, code reviews happen before merge, and features deploy to multiple environments.

6.1 Automated Testing and Code Reviews

Djøf Trade Union uses Azure DevOps pipelines to catch problems early. When developers finish a feature, they push code to a branch **(Transcript: 00:12:52–00:13:22)**. The pipeline runs unit tests, integration tests, and end-to-end tests immediately. Failed tests notify developers right away.

Code reviews add another quality layer. A colleague must review every pull request before merge **(Transcript: 00:13:22)**. Their Definition of Done makes quality concrete: code reviewed, tests passing, deployed to staging, and validated **(Transcript: 00:18:37)**.

Bugs still appear. When they do, they fix them immediately without estimation. If too many bugs arrive in one sprint, they reduce feature commitments for the next sprint.

6.2 Legacy Systems and Feedback Integration

Legacy code presents ongoing challenges. Some systems lack documentation, making changes difficult **(Transcript: 00:16:07)**. The team isolates new code from legacy systems where possible.

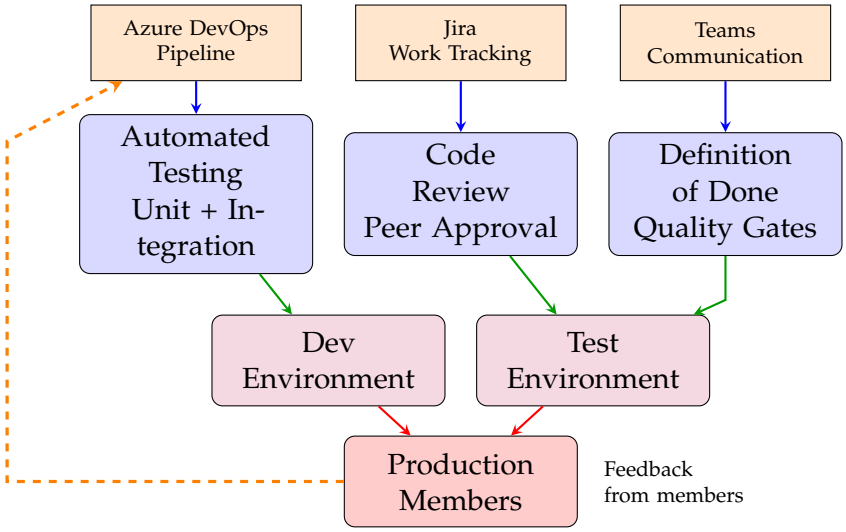

Figure 6.1: *Quality management across tools, practices, and environments. Azure DevOps runs automated tests, Jira tracks work items, and Teams handles communication. Code moves through dev and test environments before reaching production. Member feedback loops back to inform improvements.*

The pipeline improved over time. Manual deployments evolved into automated stages. Developers deploy to dev, then test, then production **(Transcript: 00:13:22)**. Each environment catches different problems.

Azure DevOps manages the build pipeline and testing automation. Teams handles daily communication, and Jira tracks work items **(Transcript: 00:19:22)**.

Sprint reviews and quarterly demos provide quality feedback from stakeholders **(Transcript: 00:20:37–00:21:07)**. The Product Owner gathers member feedback through newsletters and brings it back to the team.

Quality improved through consistent practices. Automated tests, code reviews, multiple deployment stages, and regular feedback work together to catch problems before production.

7

Deployment Practices and System Observability

Djøf Trade Union's deployment pipeline functions reliably but reveals opportunities for improvement. They manually approve each deployment stage, tests sometimes fail unpredictably, and monitoring gaps delay problem detection **(Transcript: 00:37:52)**.

7.1 Current Pipeline and Monitoring Assessment

Their current pipeline moves code through dev, test, and production environments **(Transcript: 00:31:37–00:32:07)**. Automated builds run unit tests, integration tests, and static analysis. However, deployment requires manual approval at each stage **(Transcript: 00:34:07–00:34:37)**.

Tests occasionally fail without reason and require reruns **(Transcript: 00:37:52)**. This instability wastes developer time and erodes confidence in test results. When tests fail randomly, they can't distinguish real problems from false alarms.

Monitoring relies on SMS alerts for critical failures **(Transcript: 00:36:37)**. If problems occur overnight, the team discovers them the next morning. Limited logging makes root cause analysis difficult **(Transcript: 00:37:52–00:38:22)**.

The most impactful improvement would be comprehensive logging and monitoring **(Transcript: 00:38:37)**. Detailed logs capture system behavior. Proactive alerts detect anomalies before they escalate. This reduces response time and prevents user impact.

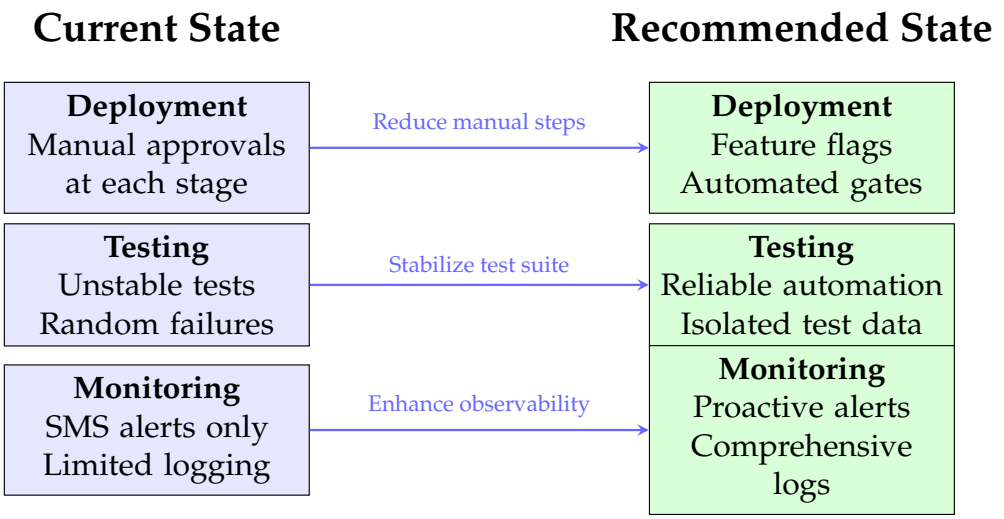

Figure 7.1: *Comparison of current and recommended pipeline practices. Current state relies on manual approvals and reactive monitoring. Recommended improvements focus on automation, test stability, and comprehensive observability to enable faster feedback and reduce operational overhead.*

7.2 Test Stability Improvements

Stabilizing tests eliminates false failures. Isolating test data, managing test dependencies, and identifying flaky tests creates reliable automation. When teams trust their tests, they deploy confidently.

Gradually reducing manual approvals accelerates deployment. Feature flags enable safe production releases without manual gates. Teams can toggle features independently from deployment.

7.3 Building Deployment Capabilities

Enhanced observability transforms operations. Distributed tracing reveals request flows across services. Centralized logging aggregates data for analysis. Custom dashboards surface critical metrics in real time.

Improving deployment frequency builds organizational capability. More frequent deployments reduce batch size, making problems easier to diagnose and fixes faster to deploy.

Investing in test infrastructure pays dividends. Container-based test environments ensure consistency. Parallel test execution reduces feedback time. Regular test suite maintenance removes obsolete tests.

8

Conclusion

Djøf Trade Union transformed chaotic development practices into structured, predictable delivery through pragmatic Scrum adoption. Their journey from ad-hoc task assignment to refined backlog management illustrates how Agile frameworks address fundamental organizational impediments when adapted thoughtfully to context.

The failed membership checkbox project crystallized the costs of absent process controls. Two weeks of development investment vanished because no gatekeeping process verified legal compliance or vendor capability before work began. This painful lesson catalyzed the adoption of refinement sessions that now surface blockers proactively. Their refinement practice demonstrates that breaking down epics collaboratively with clear acceptance criteria prevents wasted effort and aligns stakeholder expectations.

Their flexibility in adapting Scrum reveals maturity beyond rigid framework adherence. Dropping story points when stakeholders misunderstood them and occasionally skipping retrospectives when sprints lacked meaningful progress shows they value outcomes over ceremony. This pragmatism reflects understanding that frameworks serve teams rather than constrain them.

8.1 Emerging Challenges

Legacy system complexity and test instability represent ongoing impediments. Flaky tests that fail randomly erode confidence and waste developer time. Limited monitoring visibility delays problem detection, particularly for overnight failures. These technical gaps suggest areas for continued investment.

8.2 Future Development

The organization built strong foundations through refinement discipline, role clarity, and stakeholder collaboration. Enhancing observability, stabilizing test automation, and gradually reducing manual deployment gates will extend their Agile maturity. Their demonstrated willingness to adapt practices positions them well for continued evolution.

Bibliography

Digital.ai (2024). *State of Agile Report*. Tech. rep. Annual industry survey on Agile adoption and practices. Digital.ai.

Rogers, Philip (Feb. 2024). *Identifying and Engaging Stakeholders*. Medium - A Path Less Taken. Describes usage of the 2x2 Stakeholder Matrix technique for Agile projects. URL: https://medium.com/agile-outside-the-box/identifying-and-engaging-stakeholders-c16c6557317d.

Russo, Daniel (2021). "The Agile Success Model: A Mixed-Methods Study of a Large-Scale Agile Transformation". In: *ACM Transactions on Software Engineering and Methodology*. Study examining success factors for Agile transformations, finding developer skills and management support as most critical factors. URL: https://dl.acm.org/doi/10.1145/3571849.

— (2025a). *ASE Lecture 3: The Scrum Framework*. Course lecture slides. Topics: Scrum roles, events, values, empiricism, responsiveness, stakeholder concern, team effectiveness, continuous improvement, team autonomy.

— (2025b). *ASE Lecture 5: Requirements, Done & Risk Management*. Course lecture slides. Topics: Product Backlog refinement, User Stories, Acceptance Criteria, Definition of Done, risk management.

Sadiq, Ahmed (Fall 2025). *Interview with Ahmed Sadiq - Experienced Software Developer*. Personal interview conducted by Jawad. Trade union organization (internal development department). 10-12 years of experience. Interview conducted in Danish, transcript available in Appendix.

Schwaber, Ken and Jeff Sutherland (Nov. 2020). *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*. Scrum.org. URL: https://scrumguides.org/.

Verwijs, Christiaan, Barry Overeem, and Johannes Schartau (2020). *Scrum: A Framework to Reduce Risk and Deliver Value Sooner*. The Liberators on Medium. Practical guide explaining Scrum framework from empirical process control perspective. URL: https://medium.com/the-liberators/scrum-a-framework-to-reduce-risk-and-deliver-value-sooner-c49fa80902e7.

Verwijs, Christiaan and Daniel Russo (2021). "A Theory of Scrum Team Effectiveness". In: *ACM Transactions on Software Engineering and Methodology*. Research based on data from 1,978 Scrum teams examining factors of team effectiveness. URL: https://dl.acm.org/doi/10.1145/3571849.

Verwijs, Christiaan and Daniel Russo (May 2022). *In-Depth: What Makes Scrum Teams Effective?* The Liberators on Medium. Non-technical version explaining 5 core factors and 13 sub-factors for team effectiveness from research with 2,000 Scrum teams. URL: https://medium.com/the-liberators/what-makes-scrum-teams-effective-7ca9f56a82c7.

Zombie Scrum Resistance (Feb. 2023). *8 Workshops To Involve Your Stakeholders And Deliver More Value Sooner*. The Liberators on Medium. Practical workshops using Liberating Structures for Sprint Reviews, Product Backlog refinement, and stakeholder involvement. URL: https://medium.com/the-liberators/8-workshops-to-involve-your-stakeholders-and-deliver-more-value-sooner-78d07a94efcd.

A

Interview Questions

Note: The following questions were originally asked in Danish. English translations are provided below.

A.1 Welcome and Introduction

1. Hello and thank you for taking the time to meet. My name is Jawad and I study software engineering at AAU Copenhagen. I am working on an exam where I need to interview an experienced developer about how work is conducted as a software developer, and we will spend some time on this. If it's okay with you, I will record the conversation so I can transcribe it afterwards. Is that okay?

A.2 Project Experiences - Success and Challenges

1. First of all, I would like to talk a bit about project experiences. If you were to show a new team member an example of how things are done at your workplace, which project would you choose that you work with? And why that one specifically?

2. Have you experienced having a project that did not live up to expectations? And if you were to return to that project together with your company, what would you have done differently? Have you had any bad experiences with the way work is conducted in your projects?

3. That makes good sense. When you look back at all the projects and tasks you have been involved in, what characterizes those that just flow and

have really good momentum? Are there any principles from Agile that you use, or some priorities, something that makes it easier and more pleasant to work with?

4. Are there other things you measure - for example customer satisfaction? How do you assess whether your project was a success, not only for your goals within development, but also whether it is something that has delivered value in the real world?

A.3 Work Methods and Daily Practice

1. Now I would like to talk more about work methods and daily practice. How would you describe your approach to development? Is there a specific methodology you use, such as Scrum?
2. How has it evolved over time? Have there been changes in the way you have worked within Scrum, or have you just kept it from the start?
3. Are there things you have changed about your work method with Scrum over time?
4. Can you tell me about the process that is gone through when you have received a task? What happens from when you receive the task until it reaches the users? Which steps must you go through for the task to be carried out in reality?
5. How long are your sprints? Have you experimented with other lengths?
6. Who decides what should be done first? How does that prioritization work?
7. What is the best thing about your current way of working?
8. And what is most frustrating in the daily work?

A.4 The Team and Collaboration

1. Can you paint a picture of the team for me - who is involved, and what are their roles?
2. How do you ensure that everyone is pulling in the same direction - that everyone knows what is important right now?

3. How do you communicate on a daily basis - standups, Slack, ad hoc meetings?
4. How much contact do you have with those who actually use the product?

A.5 Measurement and Continuous Improvement

1. How do you keep track of whether you are on the right path? Do you track velocity, lead time, or other metrics?
2. Do you hold retrospectives? Can you give an example of something you have changed based on a retro?
3. Do you feel that management supports your way of working? Do you have the freedom and resources you need?
4. How good are you at changing direction when requirements change?
5. How do you ensure that competencies in the team stay sharp?

A.6 Quality and Testing

1. When is a task 'done' for you - what needs to be in place before you say finished?
2. What do you do to ensure quality along the way - code review, pair programming, static analysis?
3. Can you describe your test approach - which types of tests do you run, and when?
4. How much of your testing is automated? Do you trust that tests catch the problems?
5. What happens when you find a bug - what is the process from when it is discovered until it is fixed?

A.7 AI in Development

1. Have you adopted AI tools for development - Copilot, Claude, Cursor, or similar?

2. What do you primarily use it for? Where does it provide the most value in everyday work?
3. Has it noticeably changed how you work?
4. Have you experienced situations where AI-generated code has created problems?
5. Has management said anything about what you may and may not do with AI in the codebase?
6. How do you see AI in the future for your workplace? Could you see it taking over more and perhaps reducing the number of employees, or will it still be the human element that delivers quality?

A.8 CI/CD and Technical Setup

1. Can you describe the journey from when you push code until it runs in production - which steps does it pass through?
2. Which tools do you use in your CI/CD setup - GitHub Actions, Jenkins, Azure DevOps, or something else?
3. How much runs automatically vs. how much requires manual approval?
4. How often do you deploy to production? Is it continuous, daily, weekly, or after a specific sprint?
5. What happens when something goes wrong in production - who gets the alarm, and what is the process?
6. Where do you experience the biggest challenges in your pipeline?
7. If you had free hands to improve one thing in your setup, what would it be?

A.9 Stakeholders and the Outside World

1. Who do you primarily make the product for? Who are your most important stakeholders?
2. What happens when different stakeholders have opposing interests? Let's say members want something, but it conflicts with what the organization wants, or it is too costly. Who has the final say?

3. How do you keep different groups updated on what is happening in the project?

A.10 Closing

1. How many years have you been doing software development?
2. If you were to give advice to someone who has just started - what is important to focus on to have a good work environment and be able to function well and deliver something?