Faculty of Mathematics and Natural Sciences

Department of Informatics

Agile teams – roles, practices, challenges and success factors

IN5140

Marthe Nordengen Berntzen

06.09.2023

UNIVERSITY OF OSLO

About me

- Ph.D. In Software Engineering

- Msc in Leadership and Organizational Psychology

-Industry experience in IT consulting

-Worked with research before Ph.D.

marthenb@ifi.uio.no

linkedin.com/in/martheberntzen/

UNIVERSITY
OF OSLO

What I do

- Research topic: *Coordination in large-scale agile software development*
- Fieldwork in a software dev. company in Oslo
- Write papers
- Present work at conferences and workshops
- Teaching and other department work
- Visiting researcher at Monash University in Melbourne

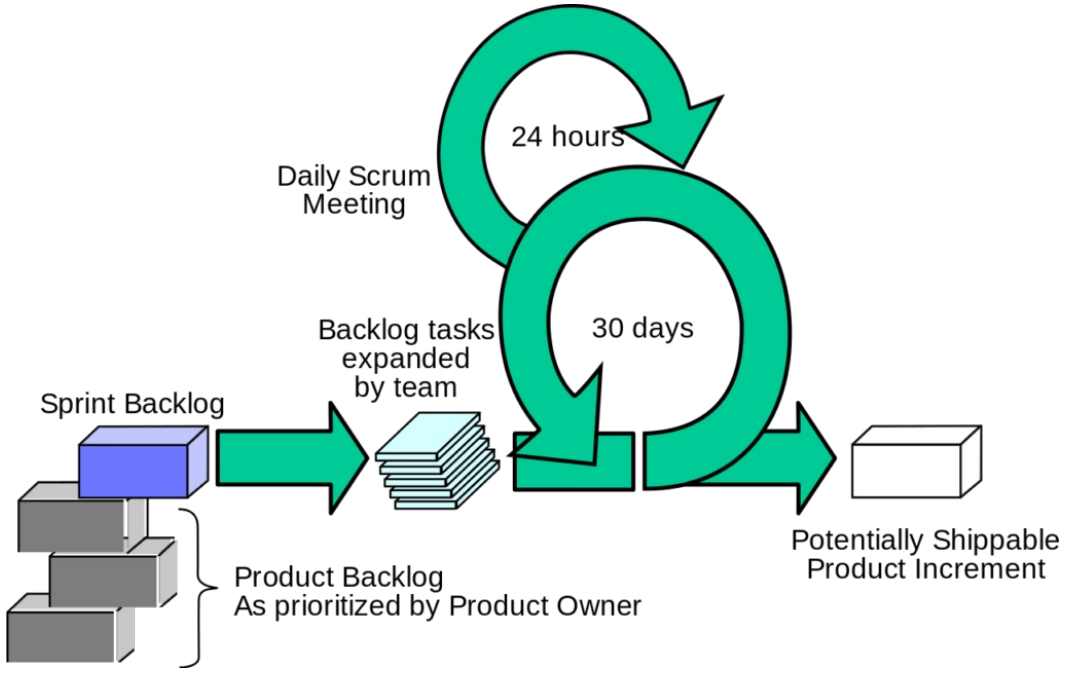
Topics covered in this lecture:

1) What is agile software development?

2) Agile teams
   - Agile team roles
   - The cross-functional agile team

3) Agile teamwork practices

4) Challenges with agile teamwork



Relationship to other lectures and the book

- Related to the coming lectures on Lean and Agile, and Large-Scale Agile.

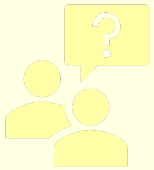
- This lecture will cover ***teamwork* in agile** (and a little bit of large-scale agile)
  - Typical practices and activities
  - Both within and across teams (inter-team)

- Curriculum: Lecture slides, book chapters 5 and 6, and 7

What is agile software development?

Discuss with your neighbour:

Have you worked in an agile team before?



What did you do? What was your role?

What did you think about it? Why?

1. What is Agile software development?

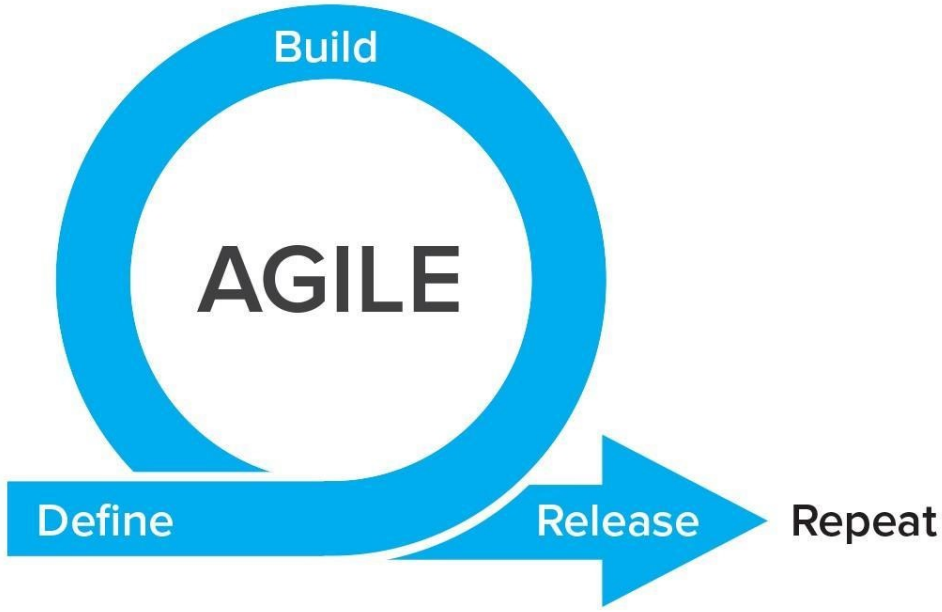
- The word means “able to move quickly and easily”

- The agile manifesto and agile principles was put together 20 years ago

"...AND THIS IS OUR AGILE PROJECT!"

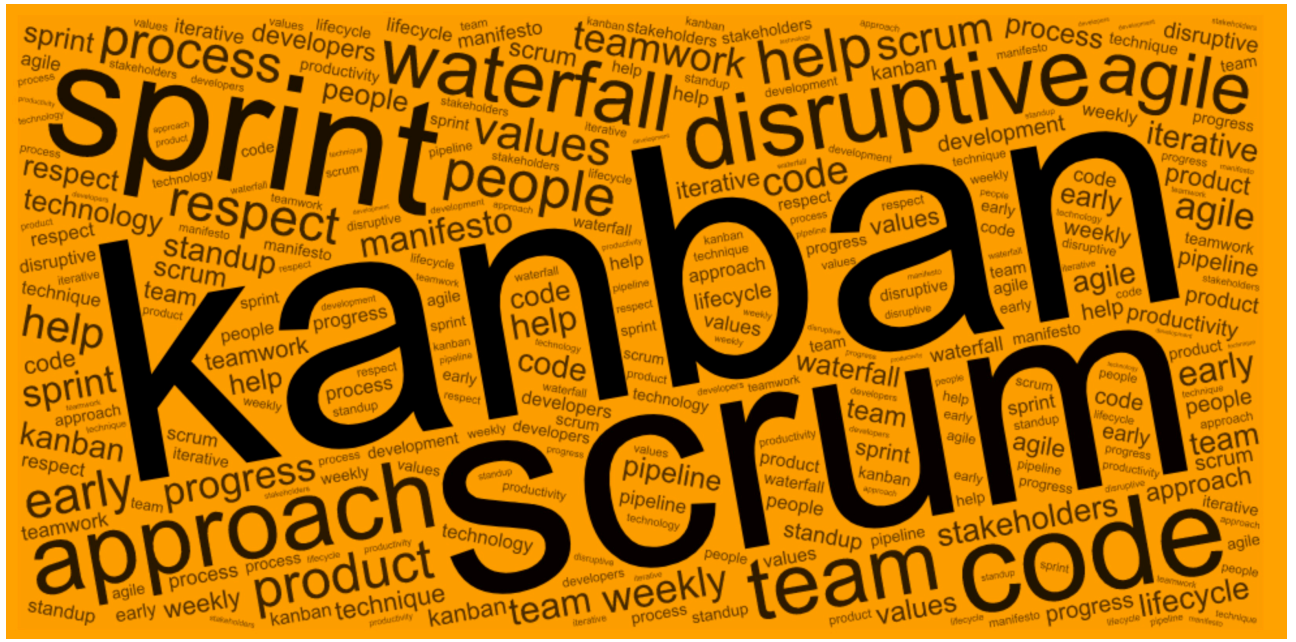
What is agile software development?

- There are many ways of developing software, and **agile is one of them**.
- Focus on creating small pieces of software at the time (incremental), repeating the process many times (iterative) in **development sprints**.
  - A pre-defined period of time during which development tasks are completed.
  - Typically lasting a few weeks to a month



There are many different approaches to agile, all with similar underlying ideas, values and principles

- Agile methodologies
- Agile methods
- Agile processes
- Agile practices
- Agile principles
- Agile teams
- Agile organizations
- Agile transformations
- **AGILE MINDSET**



The agile manifesto

As software developers, we 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

While there is value in the items on
the right, we value the items on the left more.

The 12 agile principles

1. Our ***highest priority is to satisfy the customer*** through early and continuous delivery of valuable software.

2. ***Welcome changing requirements***, even late in development. Agile processes harness change for the customer's competitive advantage.

3. ***Deliver working software frequently***, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

The 12 agile principles

4. Business people and developers must ***work together daily*** throughout the project.

5. ***Build projects around motivated individuals***. Give them the environment and support their needs, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is ***face-to-face conversation***.

The 12 agile principles

7. ***Working software is the primary measure of progress***.

8. Agile processes promote ***sustainable development***. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. ***Continuous attention*** to technical excellence and good design enhances agility.

The 12 agile principles

10. ***Simplicity***--the art of maximizing the amount of work not done--***is essential***.

11. The best architectures, requirements, and designs emerge from ***self-organizing teams***.

12. At regular intervals, ***the team reflects*** on how to become more effective, then tunes ***and adjusts its behavior*** accordingly.

2. Agile Roles

Agile roles (chapter 5)

The self-organizing team

Manager

Product Owner

Members and observers

The customer

Agile coach/ Scrum Master

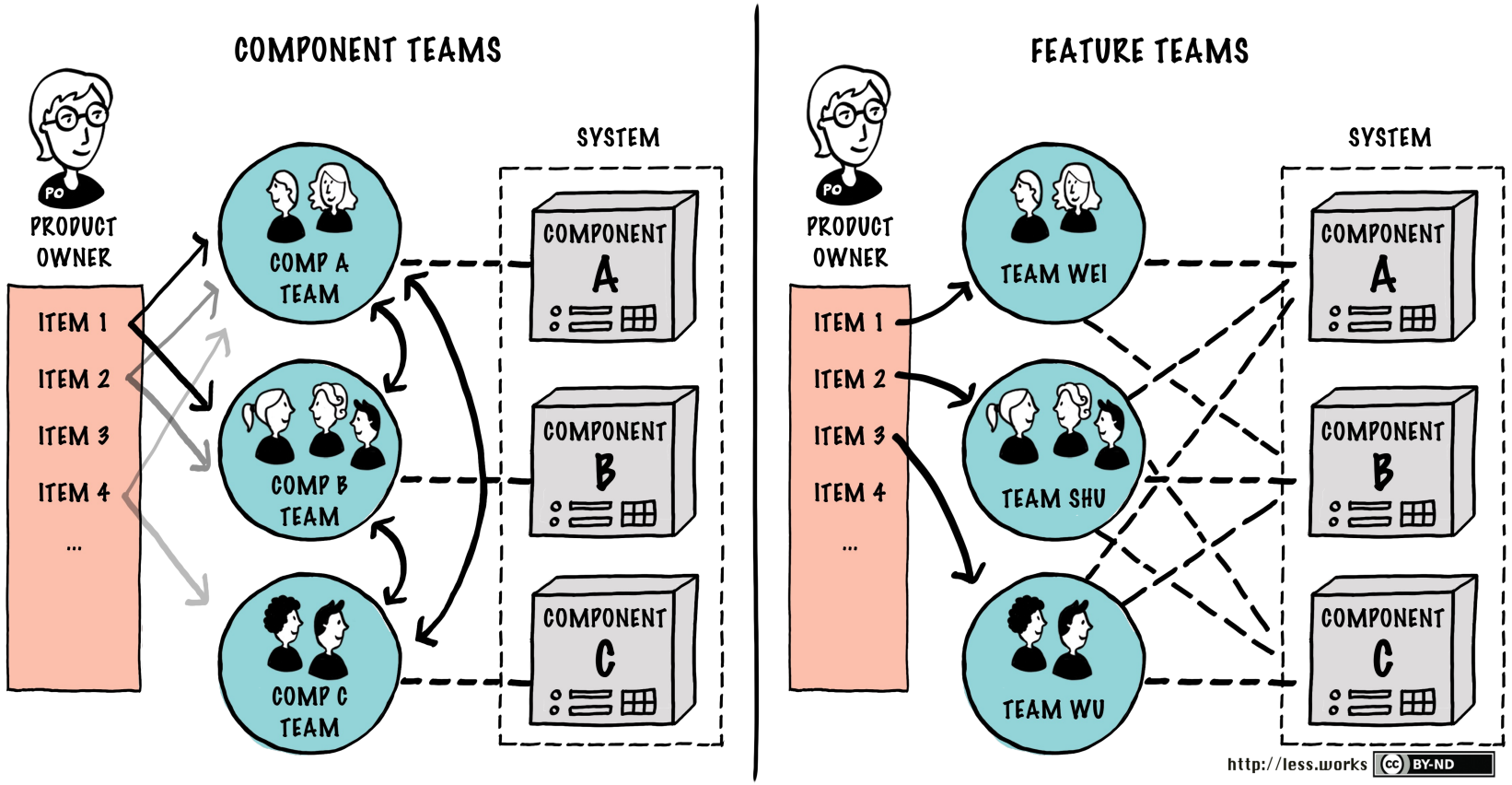
Why is the team so important in agile?

Most software systems and products are developed by **groups of people** who need to **interact to deliver the product**

Agile principle #11:

«The best architectures, requirements and designs emerge from self-organizing teams»

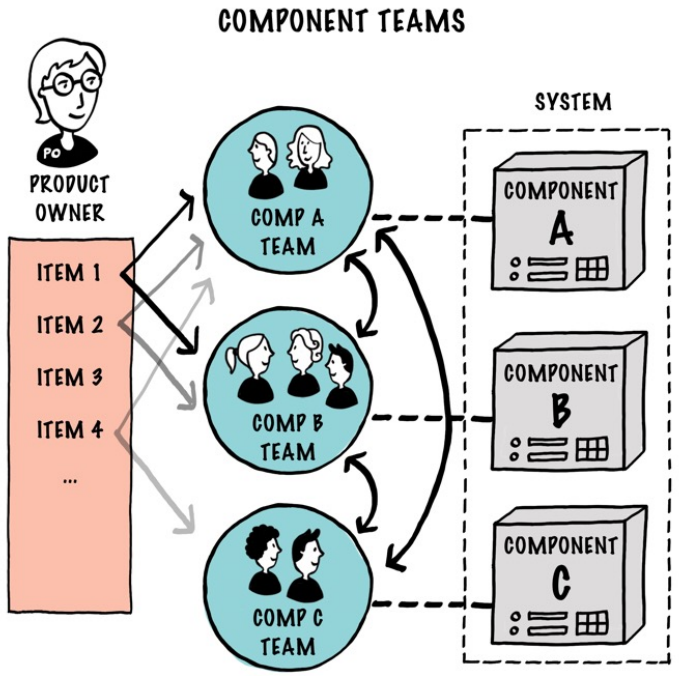
Two types of software development teams



https://www.slideshare.net/LelandNewsom/feb17-feature-vscomponentteams

Specialized teams (component teams)

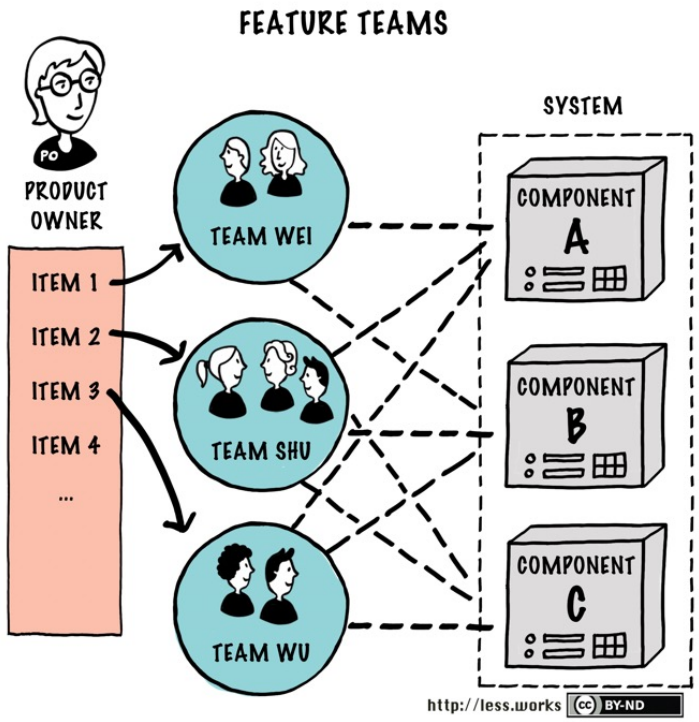
- Focus one component, one part of the overall product, or one part of the development process
- The team members are all specialized in this part of the system
- For example:
  - Backend team
  - Front-end team
  - Test team
- Understand their own component well, but may lack overall system understanding



Cross-functional teams (feature teams)

The ideal of agile teams

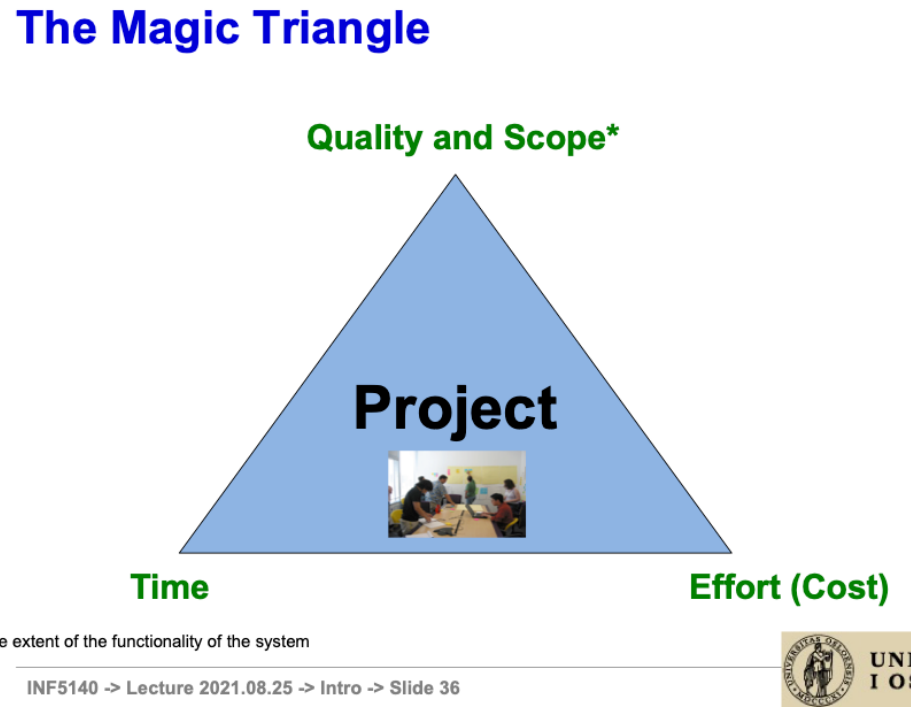
- Build *features of the final product*
- Team members not committed to one specialist area (i.e., they are full-stack)
- Ideally, any developer can work on any task
- Should work together for a longer time period (preferably co-located)
- Typically, 7 ± 2 people



The quality of teamwork influence:

1) The quality of the software product (e.g., Functionality, robustness, reliability, and performance)
2) The quality of the software project (time and cost)
3) Team member satisfaction and learning

Agile teamwork practices help address these quality aspects in different ways



Agile roles: **The manager**

The agile manager does NOT

- Assign tasks
- Decide what functions and features to implement
- Direct the work of teams or team members
- Request status reports through meetings or documentation

Instead, the agile manager

- Establish a good working environment
- Ensure a smooth interaction with the rest of the organization (that may not be doing agile)
- Handle resources, suppliers and outsourcing partners etc.
  - *So that the team can focus on getting their work done*

Agile roles: **The Product Owner**

The core tasks of the role:

- Define and maintain the product backlog (the list of features to be developed)
  - What to be made, not how to make it. The “how” is decided by the team.
- Involved at the start and end of the sprint
  - Select user stories from the backlog
  - Evaluate the results of the sprint

Product Owners do a lot more!

- In large-scale projects, they are often responsible for coordination and prioritization of features across teams (Berntzen, Moe & Stray, 2019)
- They must handle conflicting business needs, groom and prioritize requirements, act as release master, risk assessor, governor, technical architect… and more! (Bass & Haxby, 2019)

Agile roles: **Members and observers**

Members

- Fully committed to the project
- The success of the project is critical to them
- Contributes actively during meetings
- Are part of making decisions

- For example, team members, product owners, customer (sometimes)

Observers

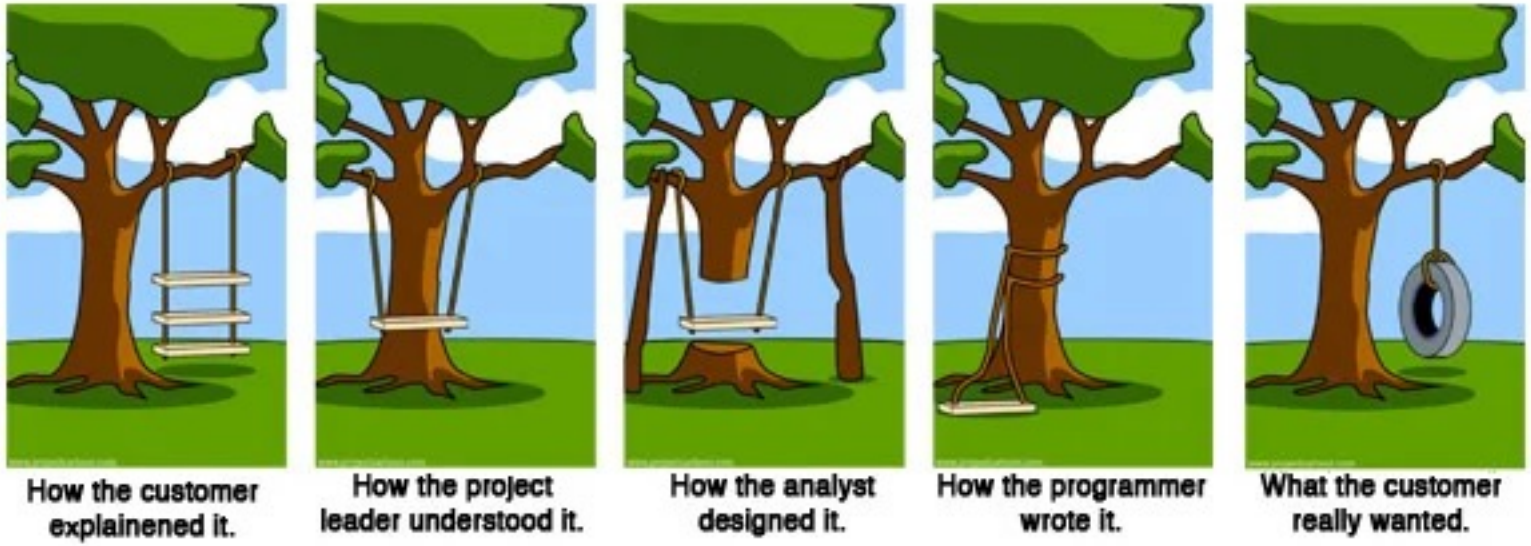
- Involved, but from the sideline
- Follow the discussions during meetings but does not typically contribute
- Can give their opinion if invited
- But have no say on the actual project decisions

- For example, other managers in the organization

Agile roles: **The customer**

- Customer interaction is a key agile value
- Ensure that the customer get the product they actually want
- Changing requirements is communicated early and effectively

- In some agile approaches, the team should include an actual, on-site, customer representative
- Other approaches cover this idea through the product owner roles as a customer representative



- In practice, perhaps not considered an actual part of the team, but the principle remains

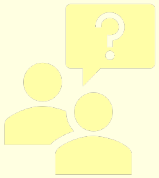
Agile roles: **Agile coach or scrum master**

- Agile methods needs ongoing facilitation and adjustment
- The central task is to remove impediments and distractions for the team
- It is recommended to have a specific role for this
  - “Scrum master” comes from the Scrum methodology
  - “Agile coach” comes from the Extreme Programming methodology

- In practice many teams have a team leader, who may or may not be a Scrum Master!
- Many companies have a separate agile coach that works at the organizational level

Discuss with your neighbour:

Are these the only possible roles in a development team?

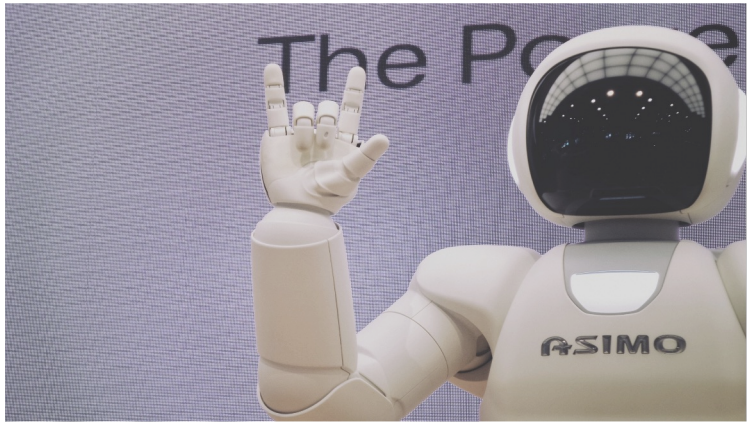


What other roles could be part of the team?

Other possible roles in development projects

- Project manager
- Business analyst
- Team leader
- «Tech lead»
- Architects
- Testers
- Designers
- Data engineers and data scientists

…and probably more



Additional roles can be challenging

Challenges may include

- Part-time vs. fulltime roles
- Differences in competence
- Differences in mindset
- Different prioritizations

Some solutions

- Good communication practices and regular team sessions (e.g., retrospectives)
- Create a team working agreement
- Make it a habit to learn about good teamwork!

3. Agile team Development Practices (chapters 6 and 7)

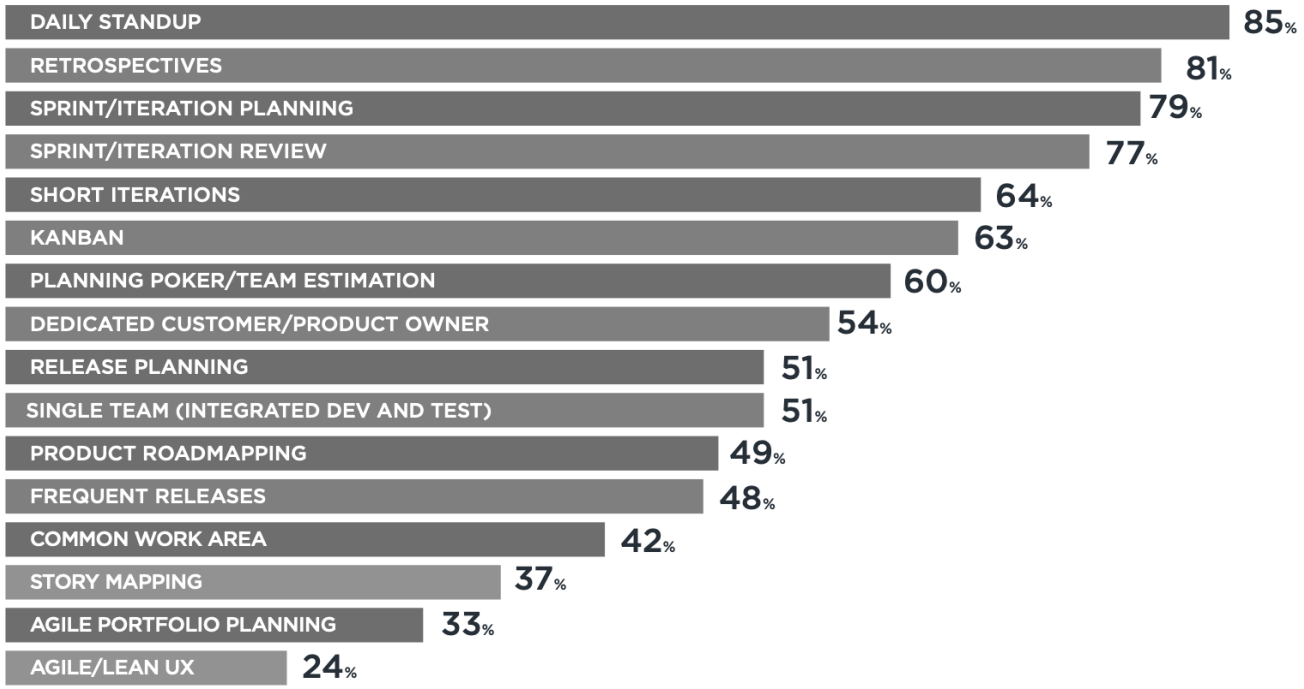
- Agile software development is governed by social rules and practices
  - That is, ***the expectations of certain behaviors*** *in certain contexts, in this case software development, and the* ***associated activities*** *and behaviors*

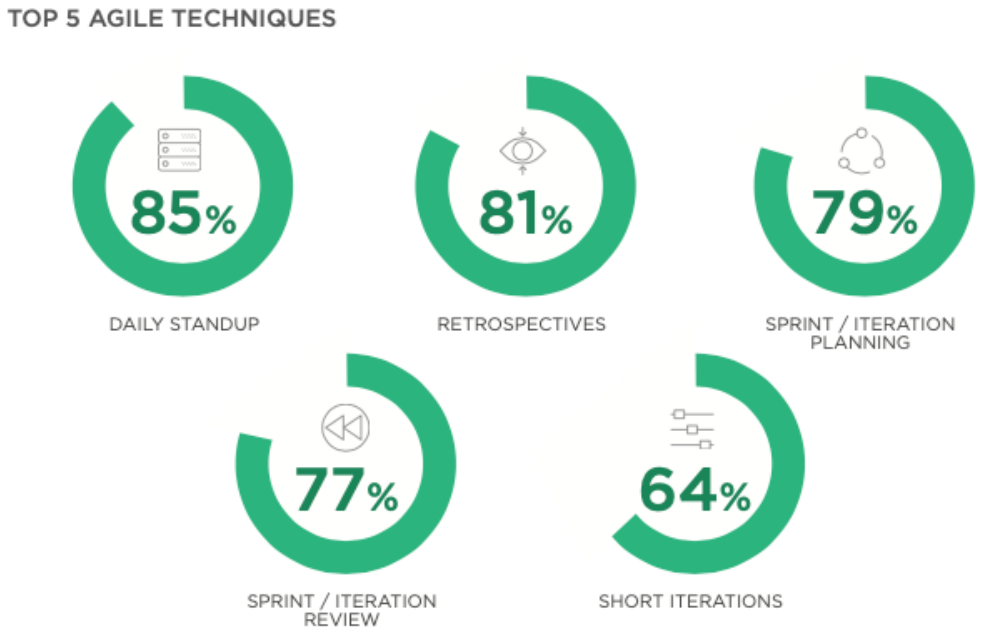
The expectations and behaviors may stem from:

- The agile principles
- Team culture
- Company culture
- Company rules and regulations
- Experiences from previous teamwork
- Industry best practices

…and much more

The most popular agile organizational practices





Digital.ai, (2020) "14th Annual State of Agile Report" https://digital.ai/catalyst-blog/the-14th-annual-state-of-agile-report/

Some of the most popular software development practices

Organizational practices (Chapter 6)

1. Development sprints
2. Daily meetings
3. Retrospective meetings
4. Documentation writing

Technical practices

5. Continuous integration
6. Test-driven development
7. Refactoring
8. Pair programming

Organizational practices: Development sprints

What it is: A pre-defined period of time during which development tasks are completed. Typically lasting a few weeks to a month, the purpose is to advance the development process incrementally in short cycles.

Sprints are started by planning meetings and concluded by sprint review meetings and retrospective meetings.

How it should be done:

- Before each new sprint, the team agrees upon which tasks to solve and how long it will take to solve them
  - Using estimation techniques such as planning poker
- "The closed-window rule": No changes, additions or removal, are to be made during the sprint

In practice:

- It can be hard to say no to urgent incoming tasks from clients (Berntzen et al., 2021).
- Estimation is hard, and it is common to take on too many tasks during sprints (Jørgensen, 2014)

Organizational practices: Daily meetings

What it is: A short status meeting where team members share what they work on.

Also known as the daily stand-up meeting or daily scrum.

How it should be done:

- The meeting is to be held every morning
- Face-to-face
- 15 minutes only
- Everyone must be standing up
- No problem-solving or discussion
- Everyone answer “the three questions” only

In practice:

- Teams should (and do) adapt the daily meeting to meet their specific needs
  - Time zone differences
  - Flexible work schedules and work locations
  - Team size (a large team may need more time)
- For example, having the meeting just before lunch and focus only on the questions “what will I do” and “what impediments” may be more efficient (Stray et al. 2016).

Organizational practices: Retrospective meetings

What it is: A meeting held at the end of a sprint or a project where team members share thoughts on the development process and the teamwork. Directly connected to the 12th agile principle: *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly*.

How it should be done:

- The team reviews “what went well”, and “what went less well” and create action points for the next sprint
- Everyone share their opinions on equal terms – typically by using post-its
- The meeting should be long enough for a proper discussion (but not too long)

In practice:

- It can be hard to get everyone to contribute
  - People have to feel safe to share (Andriyani et al., 2017)
- The meeting can often be too short*
- It is challenging for the facilitator to manage the time well, especially if some points cause a lot of discussion*

*Own observations of 14 retrospective meetings during 2018-2020

Organizational practices: Writing documentation

What it is: Software process documentation refers to information that describes the product and how it has been developed. For example, product requirement and feature descriptions, technical manuals, tutorials and how-to-guides, templates and so forth.

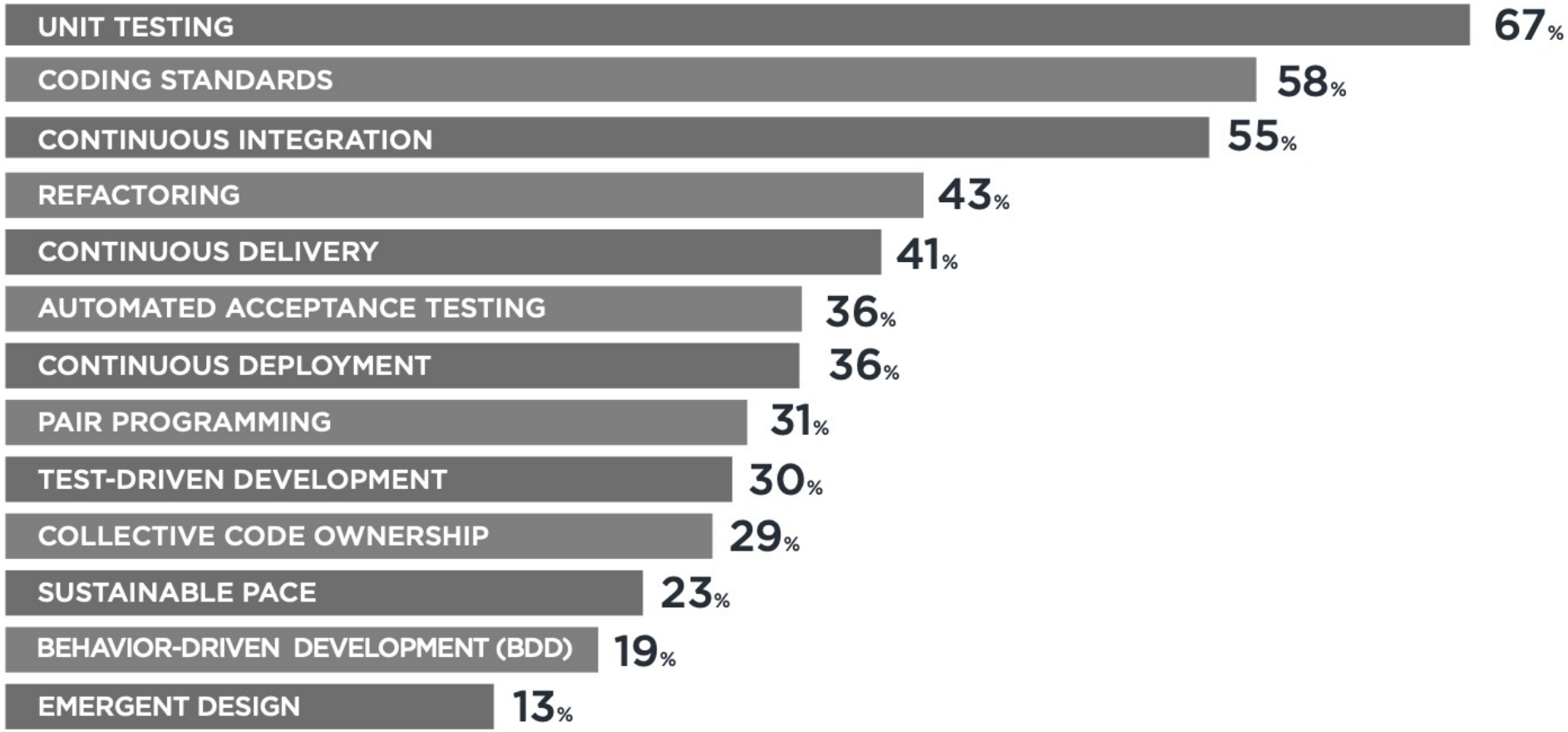
How it should be done:

- The second point in the agile manifesto states that software developers should value *"working software over comprehensive documentation"*
  - This has often been interpreted as "writing documentation is not important", and even that documentation is not needed in agile projects.

In practice:

- Many developers find documentation important, and that too little documentation was available (Stettina & Heijstek, 2011)
- Technical documentation was often consulted more often for information and less for maintenance purposes (Garousi et al., 2015)
- Abandoning documentation was not desirable when implementing agile methods (Hummel et al., 2015)

The most popular technical practices



Digital.ai, (2020) "14th Annual State of Agile Report" https://digital.ai/catalyst-blog/the-14th-annual-state-of-agile-report/

Some of the most popular technical development practices

Organizational practices

1. Daily meetings
2. Development sprints
3. Retrospective meetings
4. Documentation writing

Technical practices (chapter 7)

5. Continuous integration
6. Pair programming
7. Test-driven development
8. Refactoring

Technical practices: Continuous integration/deployment (CI/CD)

What it is: Rather than waiting until the end of the sprint/project before putting "everything" together, continuous integration refers to frequently committing changes, assembling and testing code. Continuous deployment involves releasing the code into production.

How it should be done:

- Principle no 3: **"Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale."
- Integration cycles has increasingly gone shorter
  - Microsoft's daily builds
  - XP: "integrate changes after no more than a couple of hours"

In practice:

- CI is generally considered to increase productivity and efficiency
- But can also bring technical and organizational challenges such as increased complexity in the development environment
- CI is associated with increased confidence in the software quality, sometimes even a false sense of confidence (Soares et al., 2022)

Technical practices: Pair programming

What it is: When two developers work together in real-time at the same workstation, ideally co-located, with one person writing the code (driver) and the other reviewing (navigator).

How it should be done:

- Pair programming is a co-located practice
  - "write *all programs* with two people sitting at one machine" (Beck, 2003)
  - "cover your mouth when you cough"(!)

- Changing roles and changing pairs is encouraged

In practice

- Pair-programming works well in WFH and hybrid work (Smite et al., 2021, Tkalich et al., 2023)
- Many developers prefer to work with the same people most of the time (Tkalich et al., 2023)
- Do not use it as a mentoring activity!
  - Many developers prefer a partner who have complementary skills to their own, (Begel & Nagappan, 2008)

Technical practices: Test-Driven Development (TDD)

What it is: An approach to software development where tests are always written before the actual code. TDD can be considered a software development method in itself.

How it should be done:

1. Quickly add a test
2. Run all tests and see the new one fail.
3. Make a little change.
4. Run all tests and see them all succeed
5. Refactor to remove duplication (Beck, 2003)

In practice

- Very few development projects apply TDD in this original form
- The test-first principle has become widespread
- Practical challenges with TDD include
  - Difficulties with adopting the mindset
  - Lack of knowledge and experience with this way of working
  - Tests are written to be passed rather than actively finding issues (Staegemann et al., 2022)

Technical practices: Refactoring

What it is: The process of restructuring code with the goal of improving the internal quality without changing the code's functionality.

How it should be done:

- Refactoring is only about improving the quality of the architecture
- Refactoring should "make the design simpler"
  - No duplication of code
  - As few classes and methods as possible
- Refactoring should be done in small, incremental steps

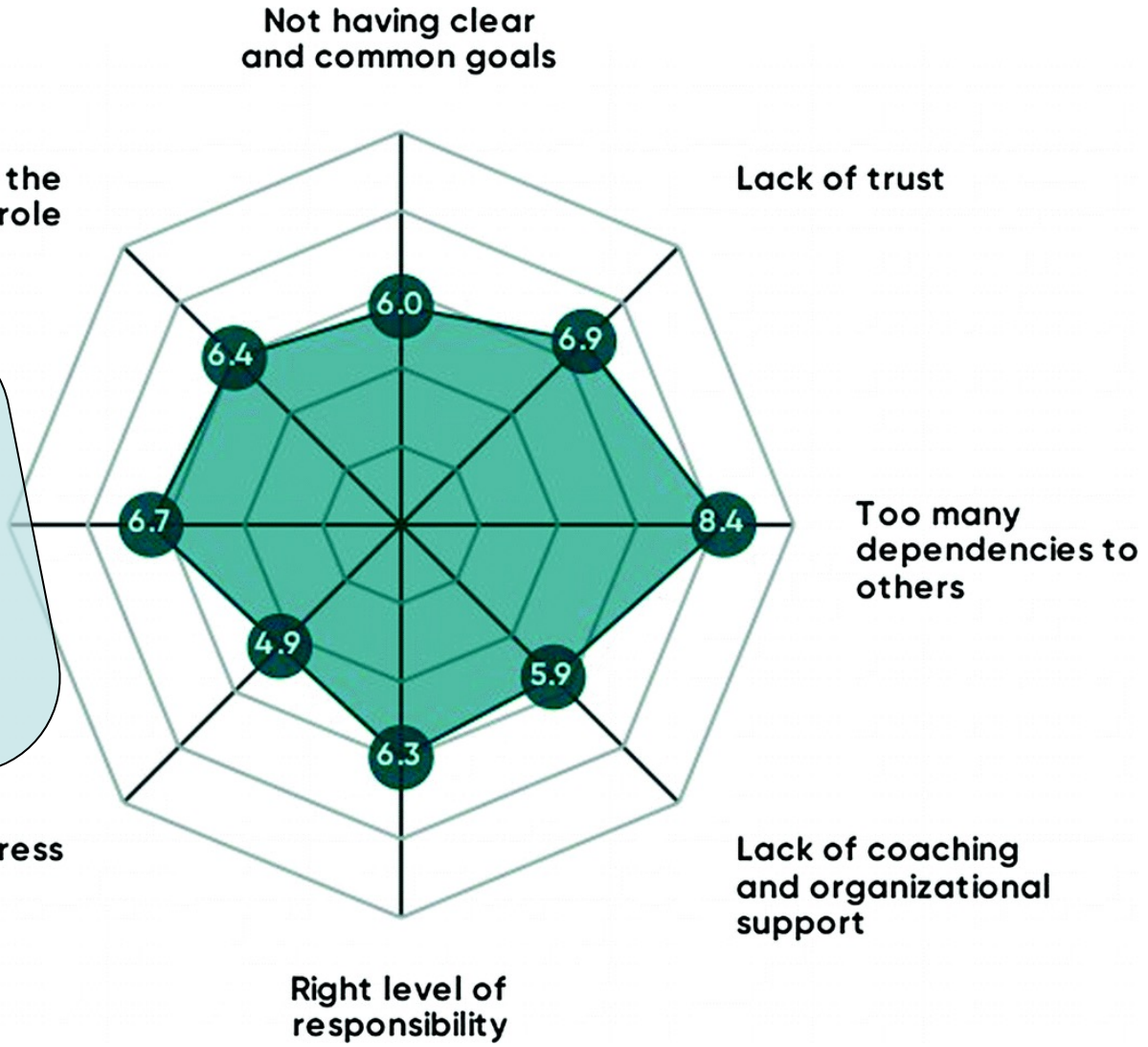
In practice:

- Over time, technical debt is likely accumulate, and a "Big refactoring" may be needed to fix it
- The cost of managing technical debt, including refactoring, in large software organizations is estimated to be, on average, 25% of the whole development time (Martini et al., 2018)
- Refactoring is increasingly being automated (Baquais & Alshayeb, 2020)

Discuss with your neighbour:

What do you think can be challenging about working in an an agile development team?

4. Challenges with agile teamwork



Hybrid work arrangements?

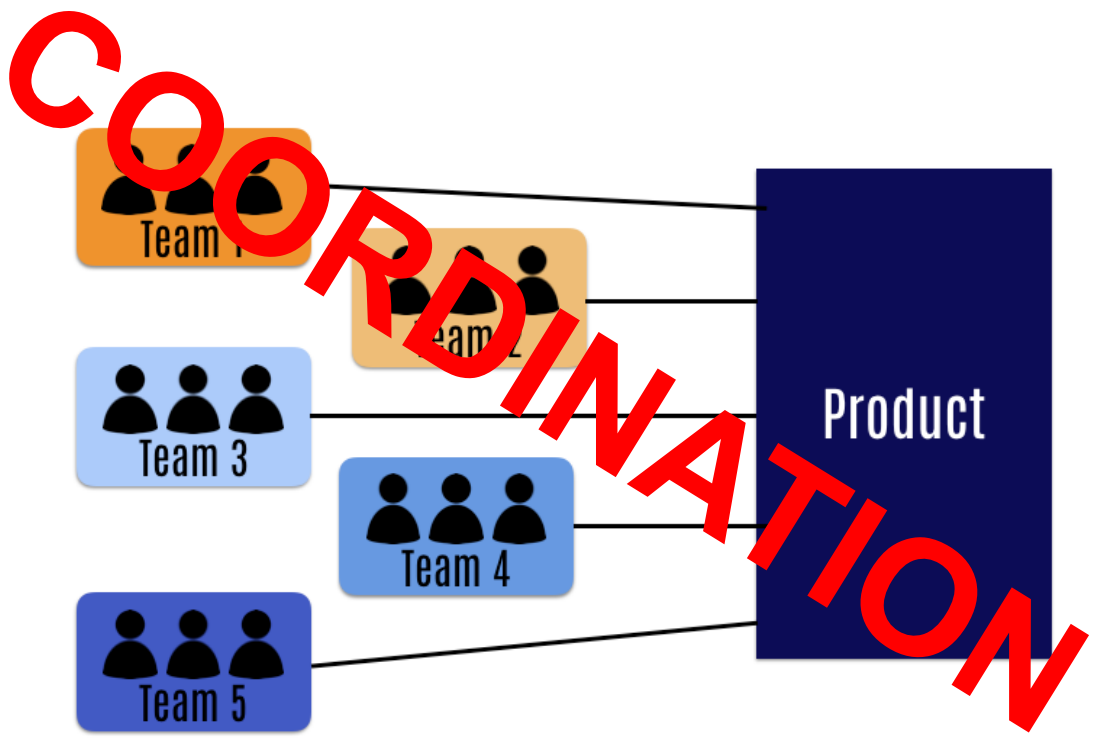
Part-time resources

For example
Different work location
Different work schedule

Moe, N. B., Stray, V., & Hoda, R. (2019).

Inter-team challenges

- Very few real-life agile teams work in isolation. They are part of larger development projects or organizations with many teams (Large-scale agile)
- All the above (and many other!) challenges re-appear *between teams!*
- *Above all: Coordination challenges*



Coordination challenges between teams

We found four types of challenges:

1) Teams work with different agile methods and different routines
2) Having overview across teams
3) Prioritizing tasks and clients across teams
4) Managing technical dependencies between teams

Berntzen, M., Stray, V., & Moe, N. B. (2021).

1) Teams work with different agile methods and different routines

If there are sixteen teams here, there are sixteen different ways of doing things" [Manager].

"*It is great that the teams are free and have a lot of responsibility. But it is also essential to have arenas where we can discuss and share knowledge across teams so that it's not spinning out of control"* [Tech Lead].

Some solutions:

-Having shared documentation routines

-Shared definition of done

-Common testing routines

-Inter-team meetings

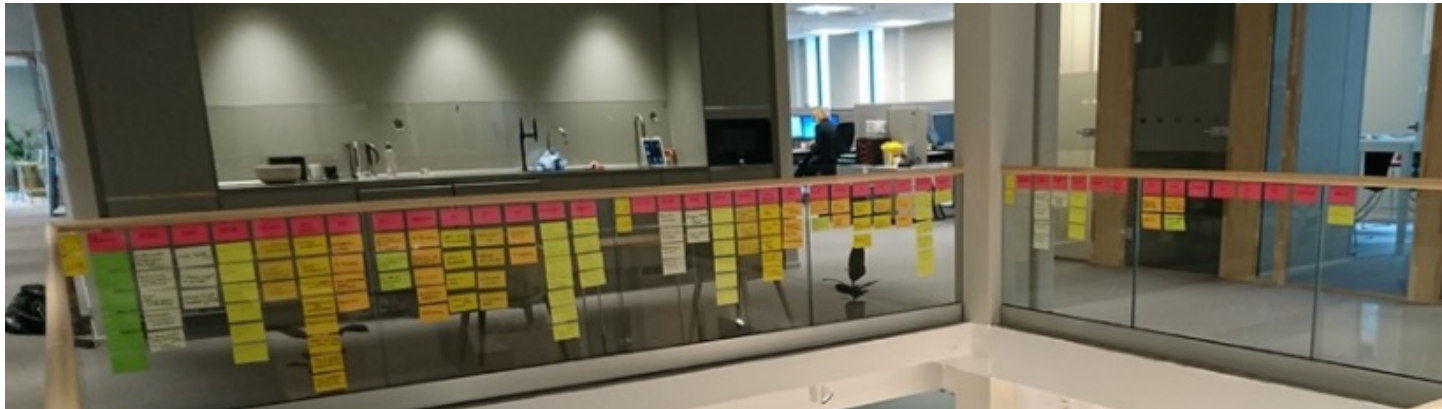
2) Overview across teams

Many developers found it hard to know **who, in which team, has what information, where, and how to find them.**

"*Right now, it is a bit hard to know the status of any given team. I don't know where to find it. You need to play detective to find out*"
[Product Owner].

Some solutions:

- Shared backlog
- Confluence page with everyone's name, team and photo
- Open office spaces
- Slack channels for each team and other relevant topics



3) Prioritizing tasks and clients across teams

"*Things come up from different clients that they all expect us to solve. Sometimes we have not managed the expectations well enough, and we may simply not have finished on time*" [Manager].

"*We were so close to finishing the feature! And then one of the teams had to prioritize something else*" [Tech Lead].

Some solutions:

- Prioritization task board
- Inter-team stand-up meetings (for team leaders, product owners)
- Temporary "task force" teams that work on specific features

4) Managing technical dependencies between teams

"*If one application goes down, the whole system goes down*"
[Team Leader].

"*I think the forum is great! It is very good to learn about other teams and what they do and what challenges they have. It's very helpful*" [Tech Lead].

Some solutions:

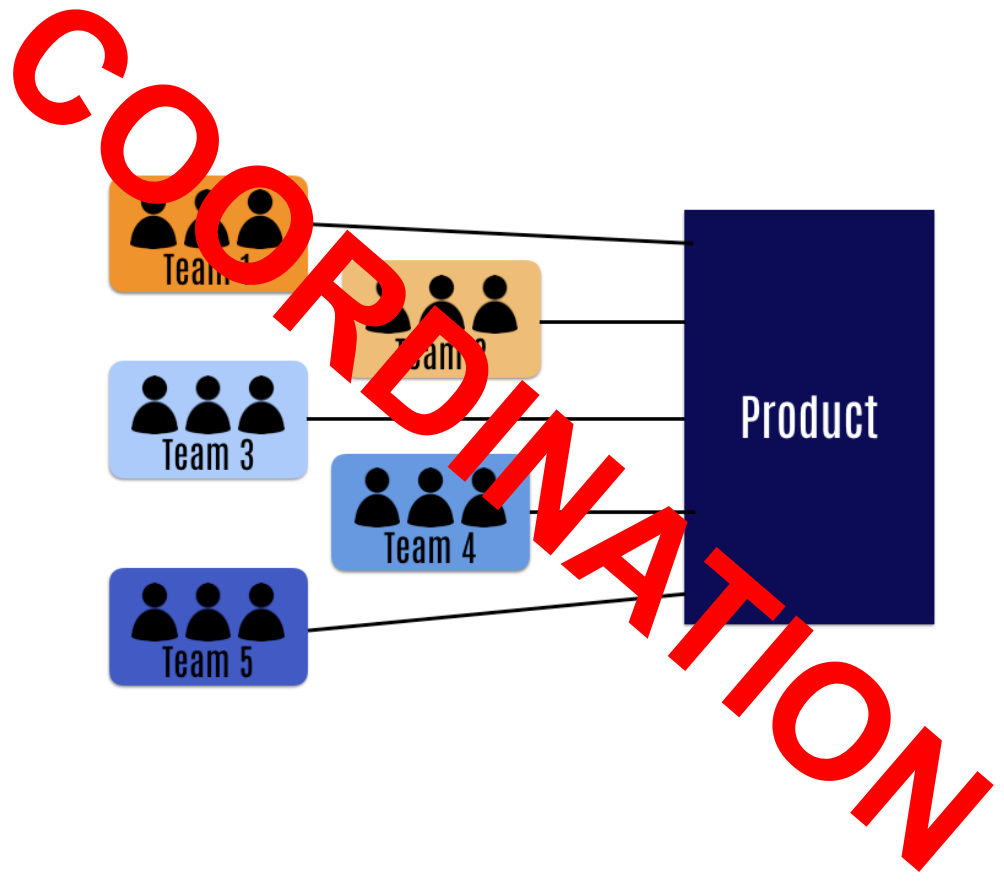
- Mapping all technical challenges across teams
- Tech lead forum for knowledge sharing about architecture across teams
- A platform team that supported development teams with shared technologies

Large-scale agile bring additional challenges

- These challenges are characteristic of large-scale agile software development

- There are approaches to large-scale agile that aims to address inter-team coordination challenges

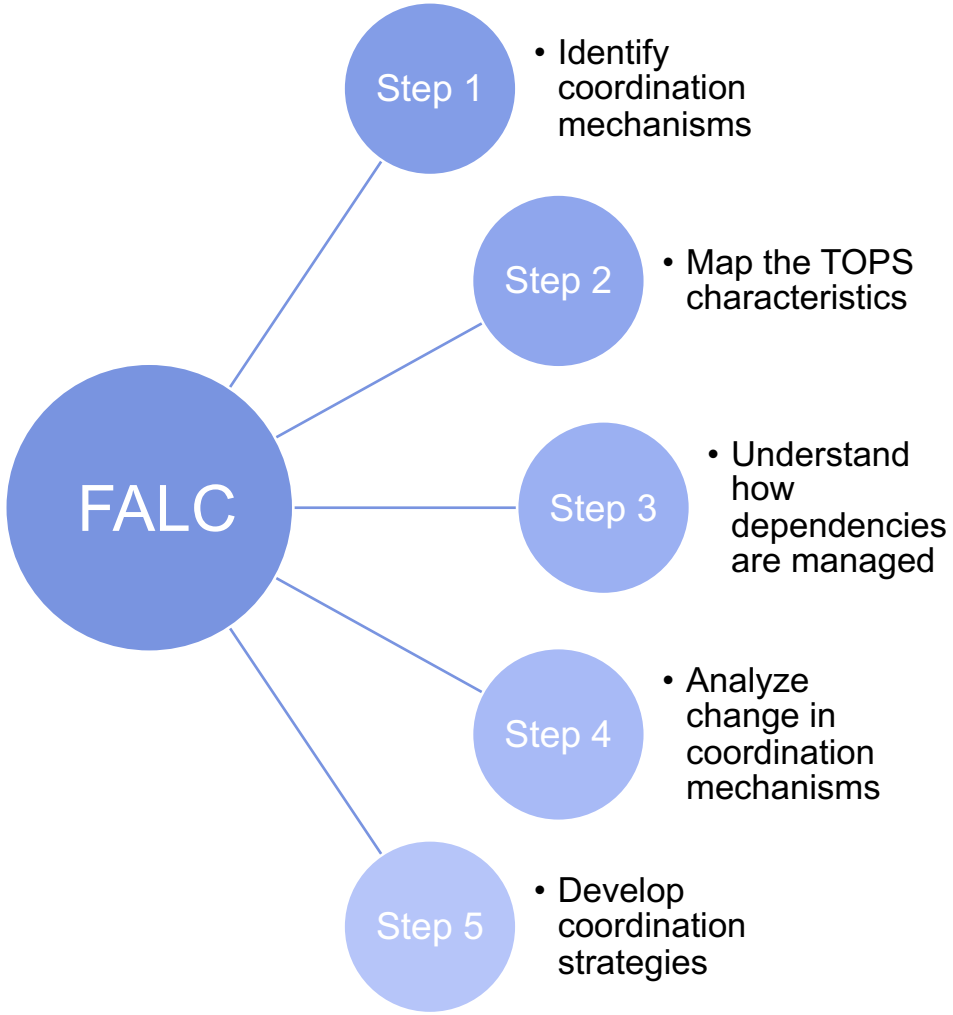
More on this in the lectures on large-scale agile (04.10) and agile transformation (11.10)



We need to use “the right” mechanisms to manage dependencies between teams

Framework for Analyzing Large-scale agile Coordination mechanisms (FALC)

- What are “the right” coordination mechanisms will vary by context, and over time.
- It is useful for organizations to be aware of which mechanisms they use, when and for what
- So that they may change and adjust their coordination as needed

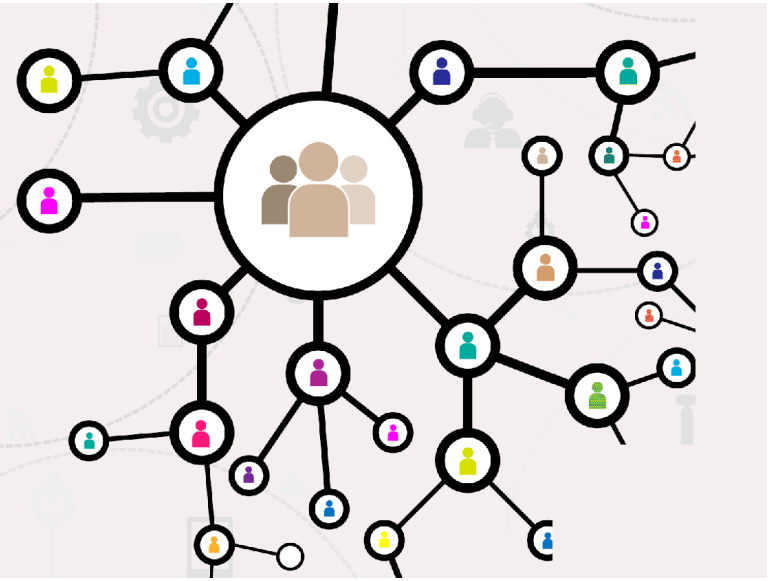


Berntzen, M., (2023).

What about when people work from other places?

There are additional challenges when working as a distributed team…

…And hybrid versions which may be part of the post-pandemic work life



More on this in the lecture on Global Software Development (18.10)

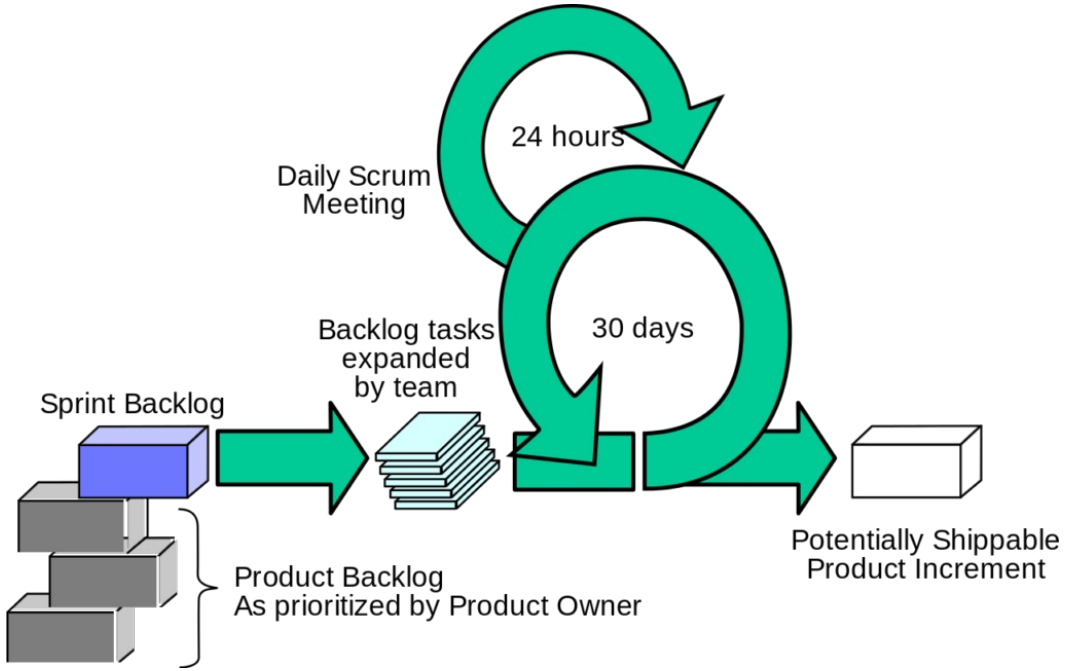
Topics covered in this lecture:

1) What is agile software development?

2) Agile teams
   - Agile team roles
   - The cross-functional agile team

3) Agile teamwork practices

4) Challenges with agile teamwork



There will be also be a lecture on agile and lean software engineering on November 15th!

References

- **Meyer, B. (2014). *Agile!: The good, the hype and the ugly*. Springer Science & Business Media**
- Berntzen, M., Moe, N. B., & Stray, V. (2019). The product owner in large-scale agile: an empirical study through the lens of relational coordination theory. In *Agile Processes in Software Engineering and Extreme Programming: 20th International Conference, XP 2019, Montréal, QC, Canada, May 21–25, 2019, Proceedings 20* (pp. 121-136). Springer International Publishing.
- Bass, J. M., & Haxby, A. (2019). Tailoring product ownership in large-scale agile projects: managing scale, distance, and governance. *IEEE Software*, *36*(2), 58-63.
- Berntzen, M., Stray, V., & Moe, N. B. (2021, June). Coordination strategies: managing inter-team coordination challenges in large-scale agile. In *International Conference on Agile Software Development* (pp. 140-156). Cham: Springer International Publishing.
- Jørgensen, M. (2014). What we do and don't know about software development effort estimation. *IEEE software*, *31*(2), 37-40.
- Stray, V., Moe, N. B., & Sjoberg, D. I. (2018). Daily stand-up meetings: start breaking the rules. *IEEE Software*, *37*(3), 70-77
- Andriyani, Y., Hoda, R., & Amor, R. (2017). Reflection in agile retrospectives. In *Agile Processes in Software Engineering and Extreme Programming: 18th International Conference, XP 2017, Cologne, Germany, May 22-26, 2017, Proceedings 18* (pp. 3-19). Springer International Publishing.
- Stettina, C. J., & Heijstek, W. (2011, October). Necessary and neglected? An empirical study of internal documentation in agile software development teams. In *Proceedings of the 29th ACM international conference on Design of communication* (pp. 159-166).
- Garousi, G., Garousi-Yusifoğlu, V., Ruhe, G., Zhi, J., Moussavi, M., & Smith, B. (2015). Usage and usefulness of technical software documentation: An industrial case study. *Information and software technology*, *57*, 664-682.
- Hummel, M., Rosenkranz, C., & Holten, R. (2015). The role of social agile practices for direct and indirect communication in information systems development teams. *Communications of the Association for Information Systems*, *36*(1), 15.

References

- Soares, E., Sizilio, G., Santos, J., da Costa, D. A., & Kulesza, U. (2022). The effects of continuous integration on software development: a systematic literature review. *Empirical Software Engineering*, *27*(3), 78.
- Smite, D., Mikalsen, M., Moe, N. B., Stray, V., & Klotins, E. (2021, June). From collaboration to solitude and back: Remote pair programming during COVID-19. In *International Conference on Agile Software Development* (pp. 3-18). Cham: Springer International Publishing.
- Tkalich, A., Moe, N. B., Andersen, N. H., Stray, V., & Barbala, A. M. (2023). Pair Programming Practiced in Hybrid Work.
- Begel, A., & Nagappan, N. (2008, October). Pair programming: what's in it for me?. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement* (pp. 120-128).
- Beck, K. (2003). *Test-driven development: by example*. Addison-Wesley Professional.
- Staegemann, D., Volk, M., Perera, M., Haertel, C., Pohl, M., Daase, C., & Turowski, K. (2022). A Literature Review on the Challenges of Applying Test-Driven Development in Software Engineering. *Complex Systems Informatics and Modeling Quarterly*, (31), 18-28.
- Martini, A., Besker, T., & Bosch, J. (2018). Technical Debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations. Science of Computer Programming, 163, 42-61.
- Baqais, A. A. B., & Alshayeb, M. (2020). Automatic software refactoring: a systematic literature review. Software Quality Journal, 28(2), 459-502.
- Moe, N. B., Stray, V., & Hoda, R. (2019). Trends and updated research agenda for autonomous agile teams: a summary of the second international workshop at XP2019. In *International Conference on Agile Software Development* (pp. 13-19). Springer, Cham.
- Berntzen, M (2023). Coordination Mechanisms in Large-Scale Agile Software Development: A Longitudinal Empirical Investigation. *Doctoral thesis, University of Oslo.* https://www.duo.uio.no/handle/10852/103637