

Faculty of Mathematics and Natural Sciences
Department of Informatics

Agile teams – roles, practices and challenges

IN5140
Marthe Nordengen Berntzen

01.09.2025



UNIVERSITY
OF OSLO



Key topics covered

1
Recap: What is agile software development?

2
Agile teams

- Types of teams
- Agile team roles

3
Agile teamwork practices

4
Challenges with agile teamwork

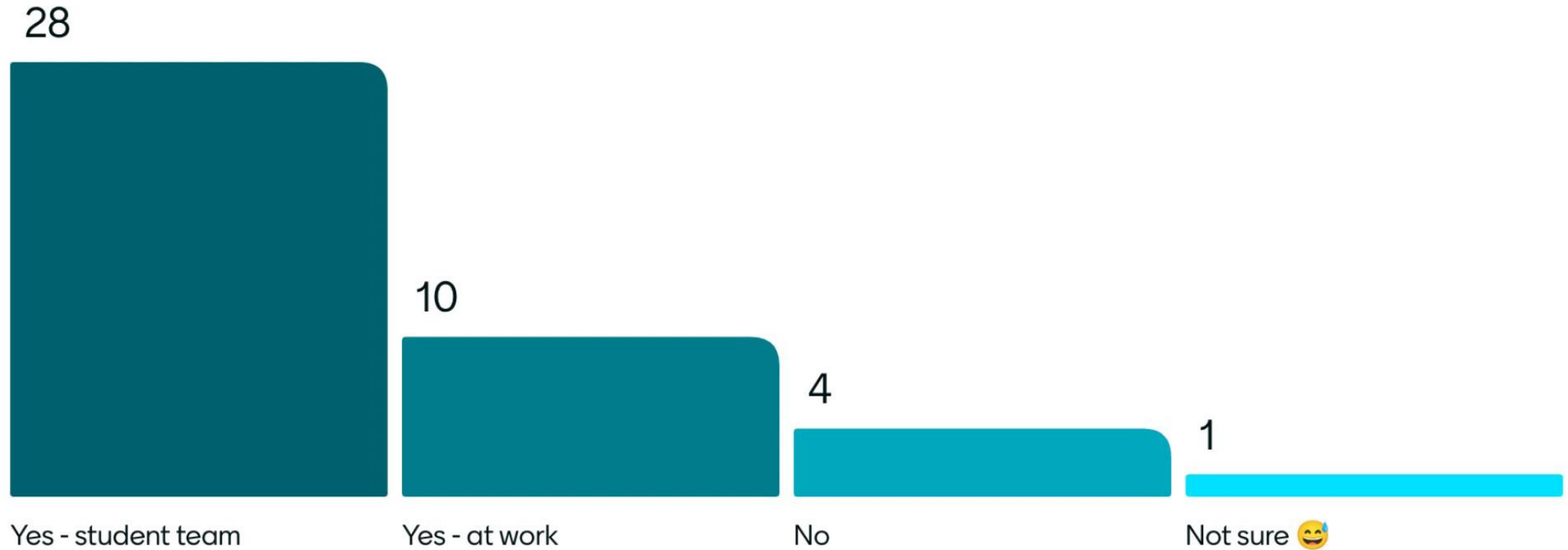
5
Large-scale agile and inter-team coordination

6
Agile teams and AI

Relationship to other parts of the course

- Related to the lectures on Process models (25.08), Lean and Agile (08.09), Large-Scale Agile and agile architecture (13.10), and the lecture on Oct. 20
 - Guest lecture from Knowit on AI in SE + teaching case on large-scale agile
- This lecture focus on *teamwork in agile* (and a little bit of large-scale agile)
 - Typical practices and activities
 - Both within and across teams (inter-team)
- The content relates to chapters 5 and 6, (and 7) in the Meyer book

Have you worked in an agile team before IN5140?

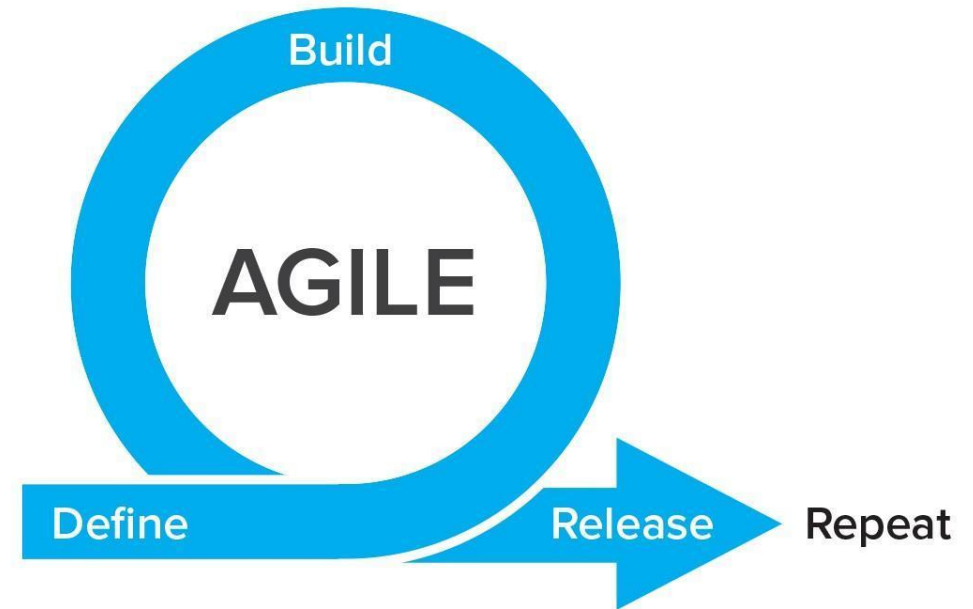


Which words come to mind when you hear the term "agile"



What is agile software development?

- There are many approaches to the software development process, 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

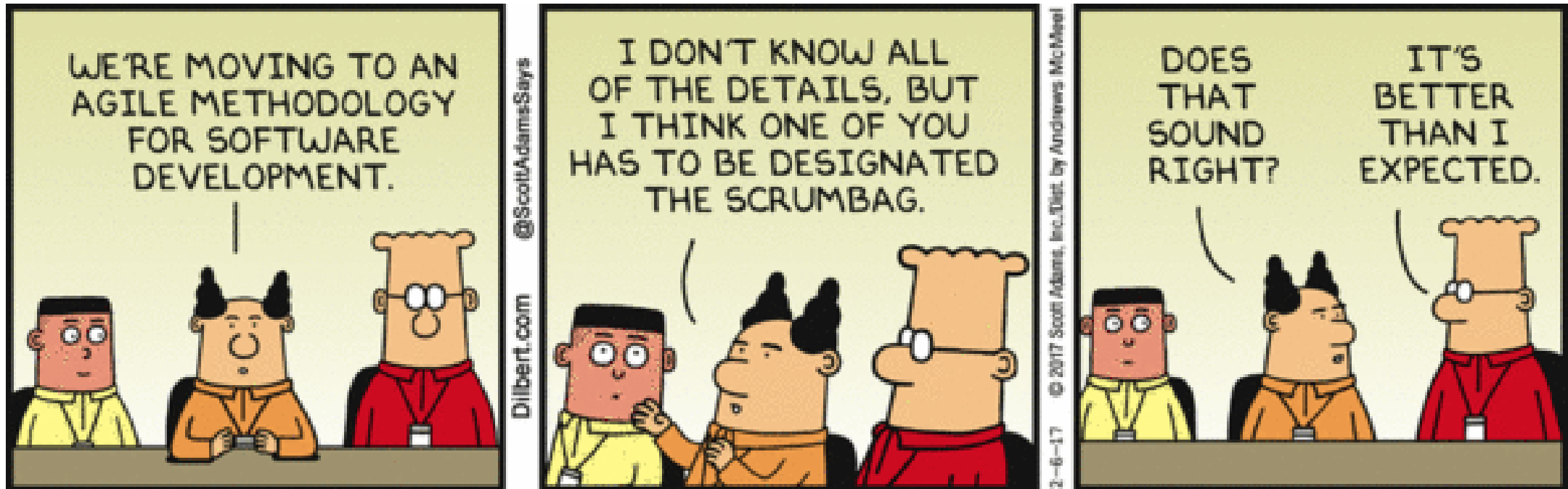


There are many different approaches to agile, all with similar underlying ideas, values and principles

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



2. Agile Teams and roles



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**»*



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



*the extent of the functionality of the system

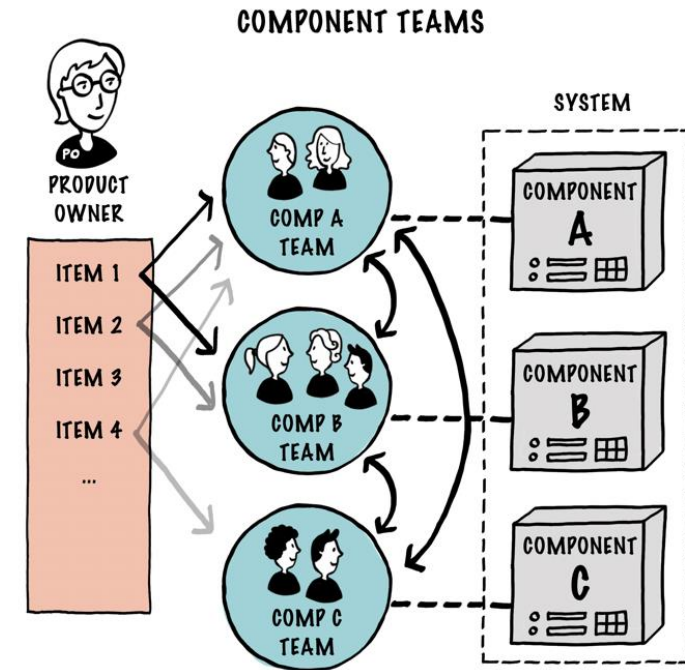
INF5140 -> Lecture 2021.08.25 -> Intro -> Slide 36

Types of software development teams



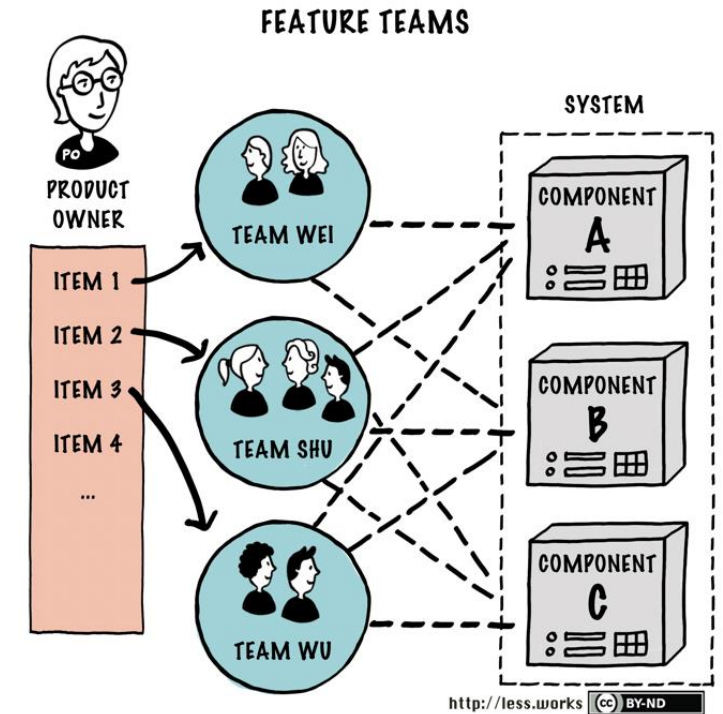
Specialized teams (component teams)

- Focus one component, one part of the overall software system
- Their components are integrated with other teams' components across the whole system
- For example:
 - Backend team
 - Front-end team
 - Test team
- Understand their own component well, but may lack overall system understanding



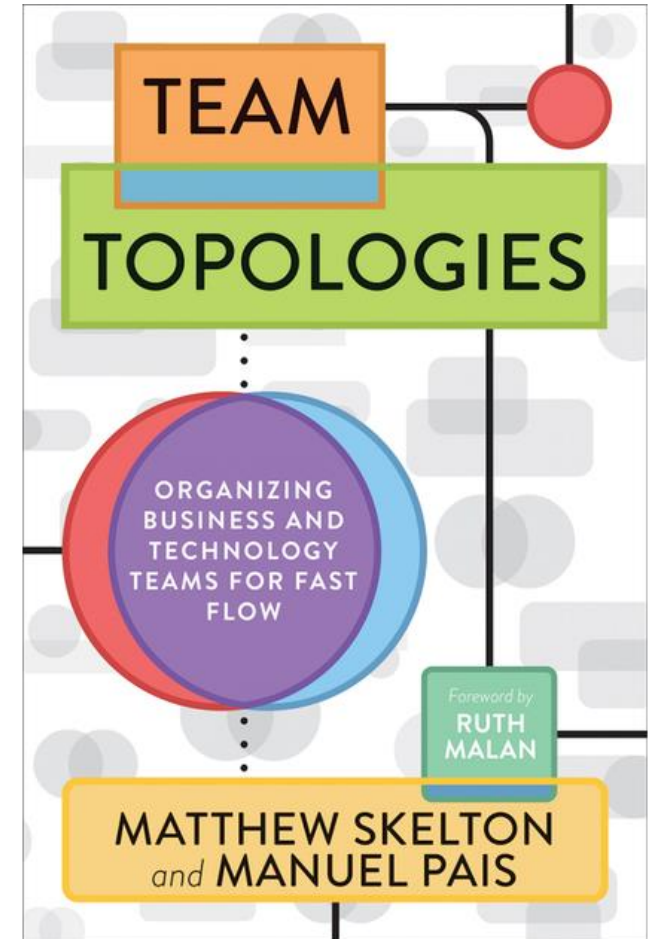
Cross-functional teams (feature teams)

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



Team typologies (Skelton & Pais, 2019)

- 1) **Stream-aligned teams** – cross-functional product teams responsible for building and delivering specific products
- 2) **Platform teams** – a type of specialized team that support and enable product teams by handling e.g., shared technical platforms and overall infrastructure
- 3) **Complicated subsystem teams** – (often temporary) teams that solve very specific, complex, one-off tasks (e.g., one specific system integration)
- 4) **Enabling teams** – assist, coach, and/or support other teams as needed. Typically management teams



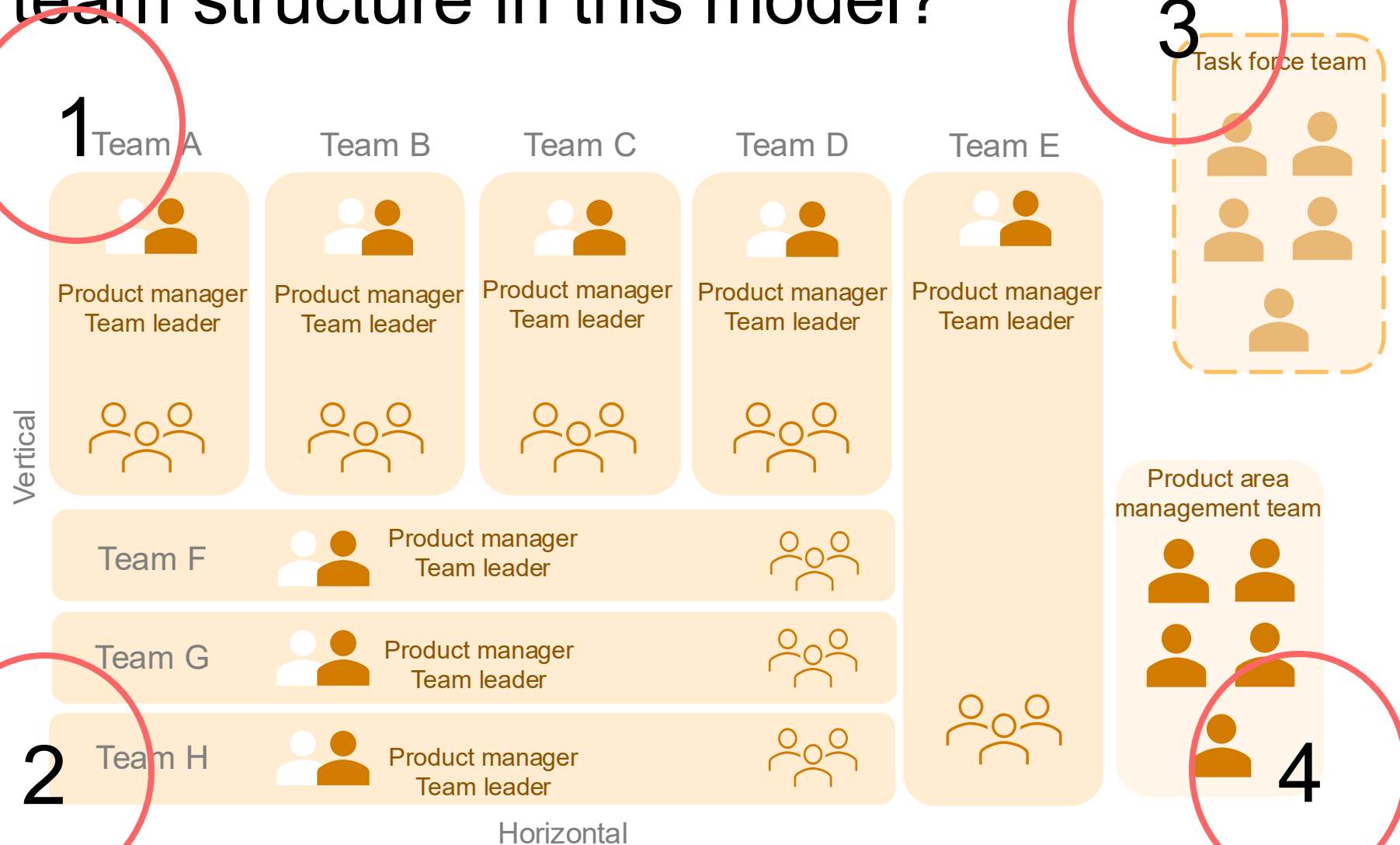
Discuss in pairs: which team typologies can be mapped to the team structure in this model?

Stream-aligned teams – cross-functional product teams responsible for building and delivering specific products

Platform teams – specialized teams that support and enable product teams by handling e.g., shared technical platforms and overall infrastructure

Complicated subsystem teams – (often temporary) teams that solve very specific, complex, one-off tasks (e.g., one specific system integration)

Enabling teams – assist, coach, and/or support other teams as needed.



Agile roles (chapter 5)

**The self-
organizing
team**

Manager

Product
Owner

Members
and
observers

The
customer

Agile coach/
Scrum
Master

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

Original 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)
- The product **manager** is an increasingly common role working in tandem with the product owner to take on many of these roles
 - Often, POs handle more product-technical aspects and PMs handle more strategic-organizational aspects (Barbala, Berntzen & Moe, 2025)

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 core 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



How the customer explained it.



How the project leader understood it.



How the analyst designed it.



How the programmer wrote it.



What the customer really wanted.

- In practice, typically 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
- Two distinct roles or parallel terms?
 - “Scrum master” comes from the Scrum methodology
 - “Agile coach” comes from the Extreme Programming (XP) 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



Other possible roles in development projects

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



Please bear in mind:
The roles mentioned in your text book
are NOT EXHAUSTIVE!!
Roles change all the time 😊

Additional roles can be challenging

Challenges may include

- Part-time vs. fulltime roles
- Differences in competence and experience
- Differences in mindsets
- 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 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***

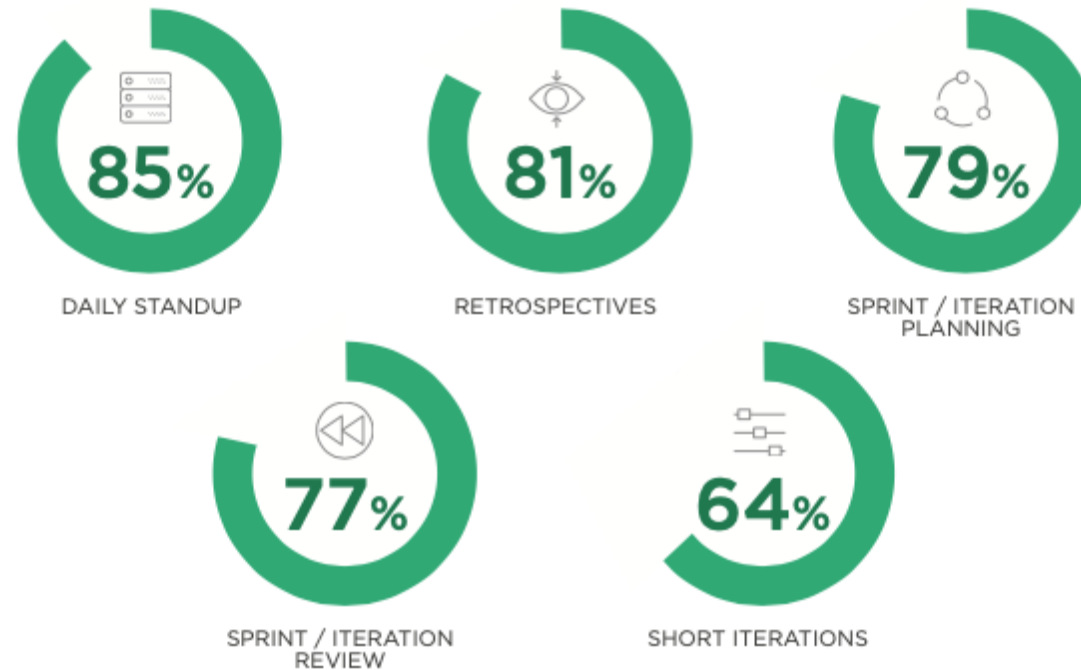
The practices we use may stem from:

- 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

TOP 5 AGILE TECHNIQUES



Agile development team 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

Please bear in mind:
The practices mentioned in your text
book are NOT EXHAUSTIVE!!
Practices change all the time 😊

Organizational development practices: 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.*

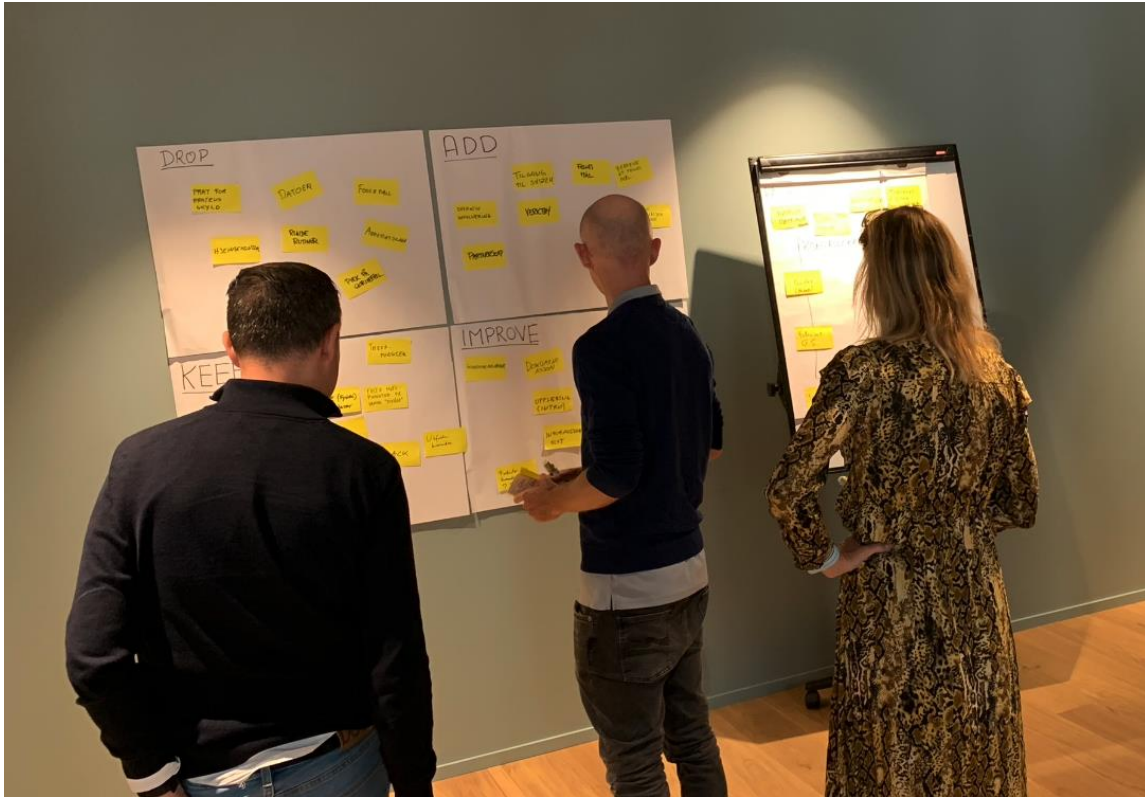
How it should be done:

- 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)

A Typical Sprint Retrospective Model



Organizational practices: Retrospective meetings



In practice:

- It can be hard to get everyone to contribute
 - People need 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*

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. – *NOT an agile practice per se! Just a necessary part of the development process!*

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)

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

Please bear in mind:
The practices mentioned in your text
book are NOT EXHAUSTIVE!!
Practices change all the time 😊

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:

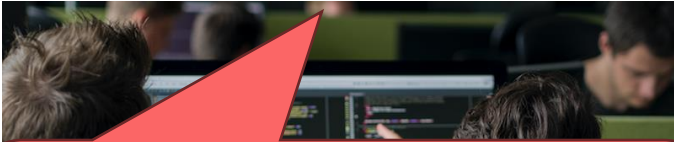
- CI is generally considered to increase productivity and efficiency
- But can also bring technical and organizational challenges such as increased complexity in the development environment
- CI is associated with increased confidence in the software quality, sometimes even a false sense of confidence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

This may be of those slightly “extremist” early agile texts that Meyer criticize 😊

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)

This may also be of those slightly “extremist” early agile texts that Meyer criticize 😊

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

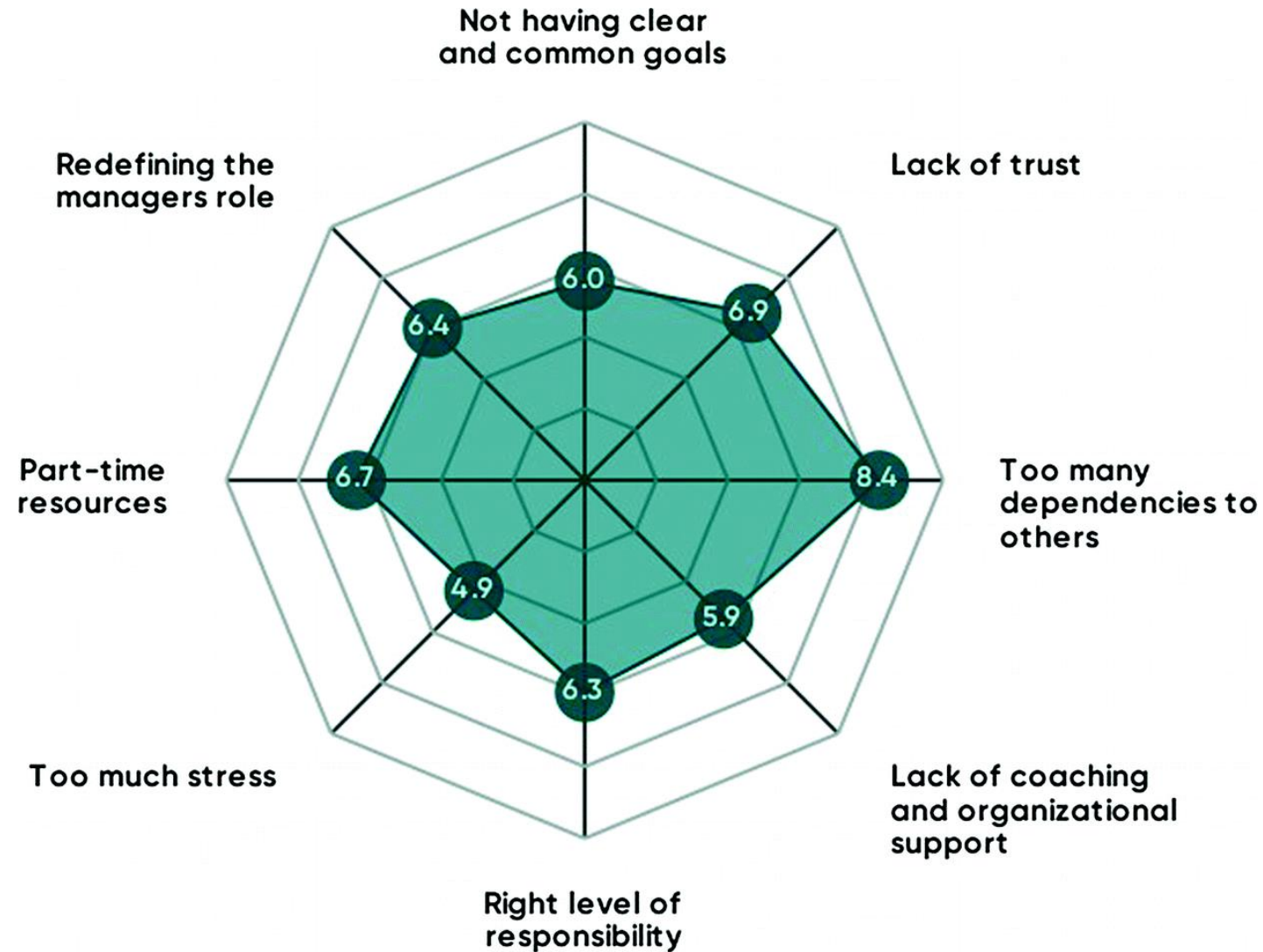
In practice:

- 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)
 - More on agile and architecture in a later lecture! 😊

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

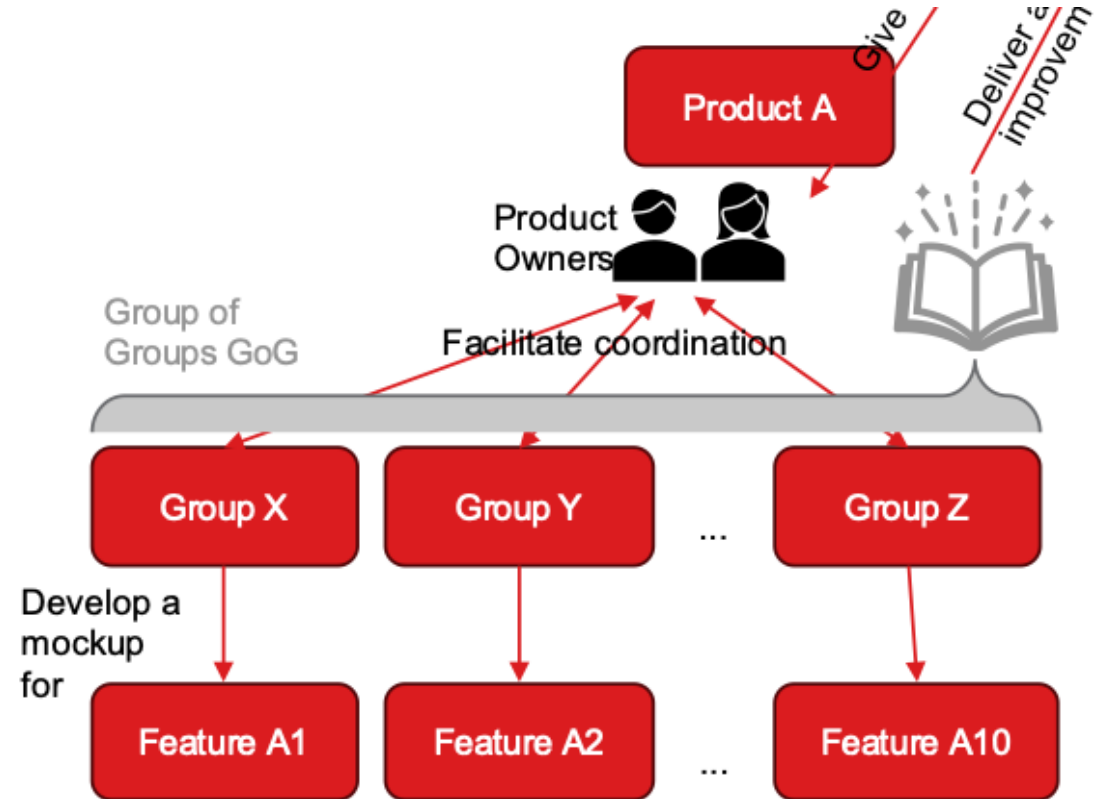


4. Challenges with agile teamwork



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 inter-team coordination 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





LARGE-SCALE AGILE



What is large-scale agile?

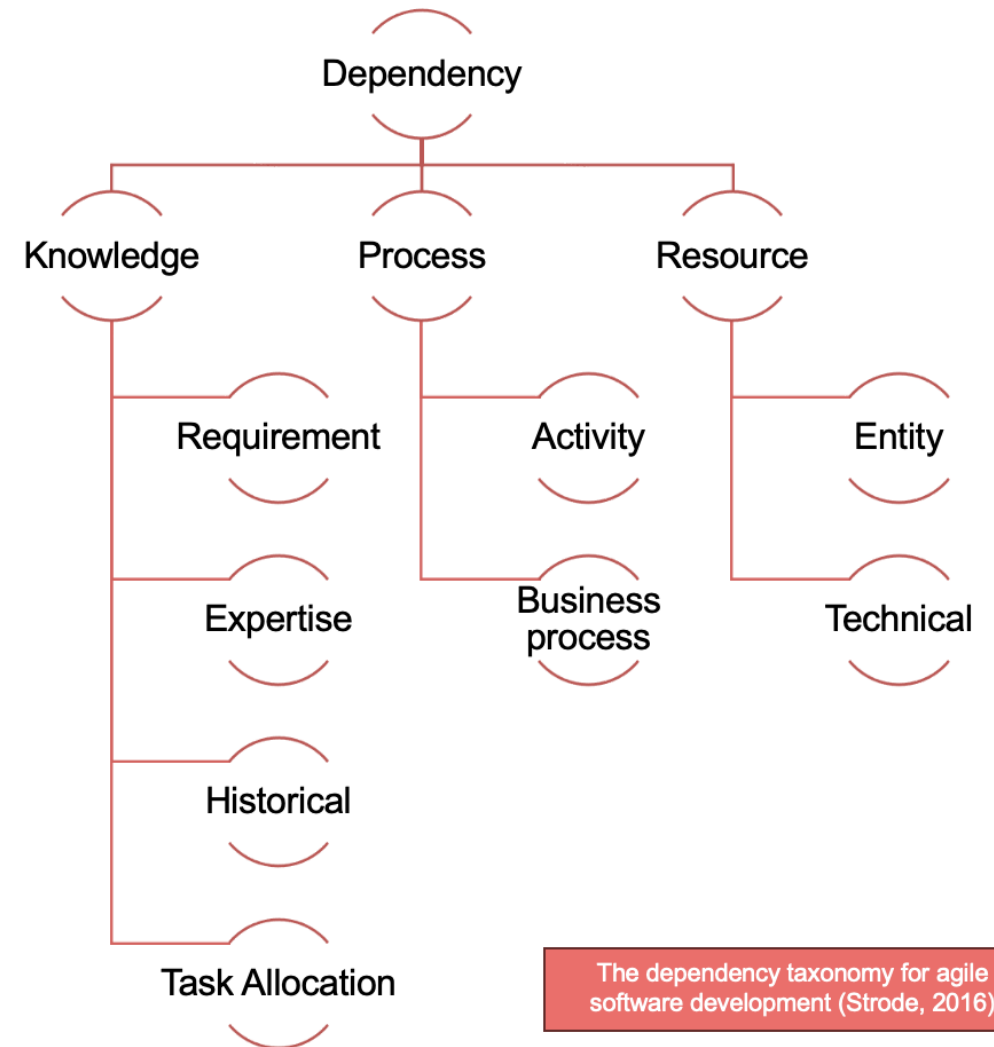
- *A software development organization with 50 or more developers or at least six teams (Dikert et al., 2016)*
- *Large-scale = 2-9 teams, very large-scale = 10 or more teams and that needs inter-team coordination for a (Dingsøy et al., 2014)*
- *A complex integration of several internal and external information systems and several stakeholders with different priorities (Rolland et al., 2016)*
- *The adoption of agile methods in settings with at least two teams, or the adoption of agile methods at the organizational level across several multi-team constellations (Uludag et al., 2020)*

What is large-scale agile?

- **ChatGPT:** «**Large-scale agile software development** is *the application of agile principles and practices* in projects or organizations involving *multiple teams working collaboratively* on a shared product or system. It typically includes coordination across teams, complex system architecture, diverse stakeholder needs, and often operates under regulatory or organizational constraints. *The goal remains to deliver working software iteratively and incrementally, while maintaining agility, transparency, and responsiveness at scale.*»

Large-scale agile requires coordination!

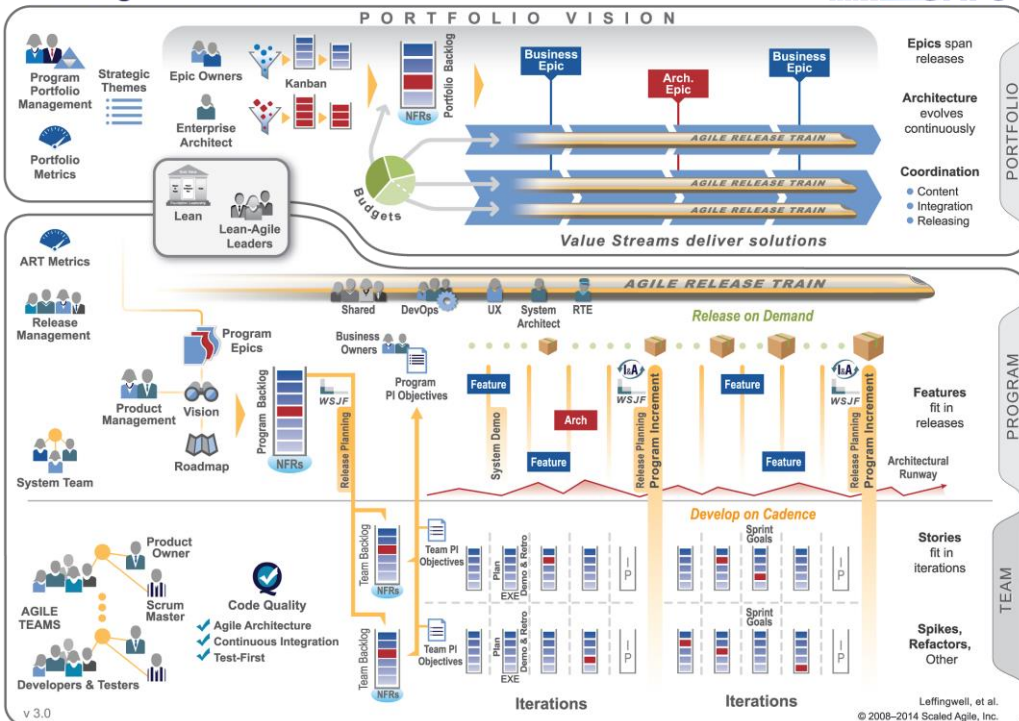
- Many teams = many dependencies!
 - May hinder speed and flexibility
 - Both within and across teams
- Teams can be dependent on:
 - Information from another team/person
 - That some things occur before others
 - Specific tools or infrastructures
 - Specific code
- We use coordination mechanisms to manage dependencies



The dependency taxonomy for agile software development (Strode, 2016)

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

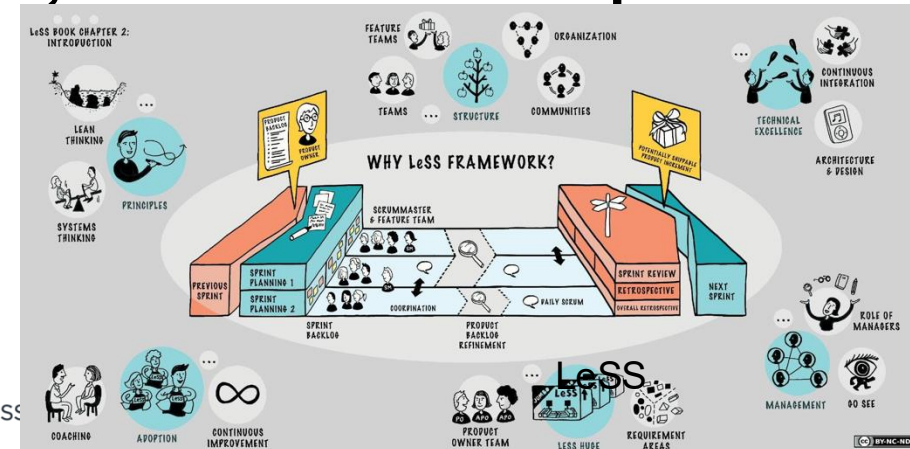




SAFe

Scaled Agile Framework® (SAFe®)

Large-scale agile frameworks provide coordination mechanisms, but at the expense of agility?



Disciplined Agile Delivery (DAD)

Large Scale Scrum (LeSS)

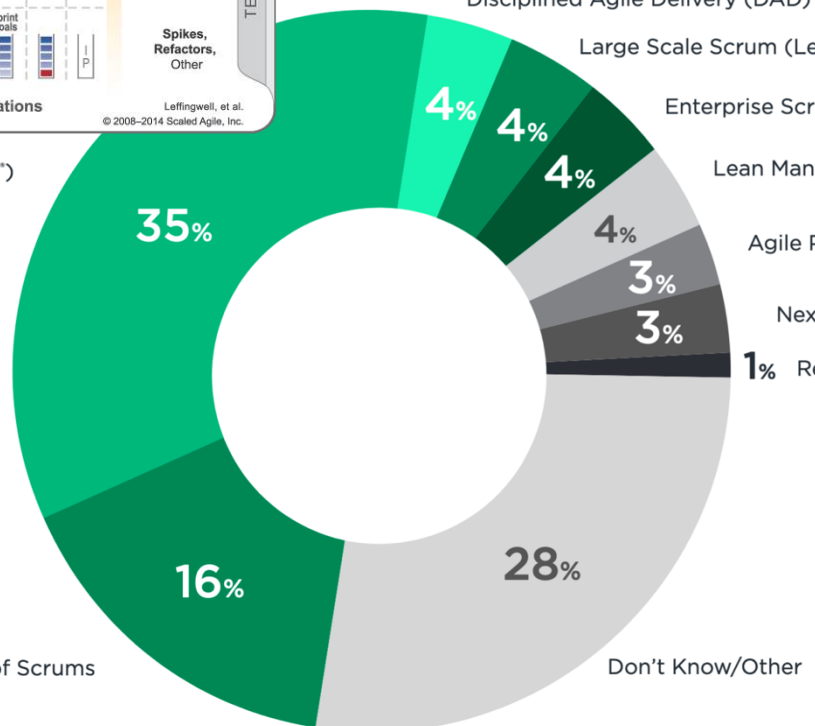
Enterprise Scrum

Lean Management

Agile Portfolio Management (APM)

Nexus

1% Rec

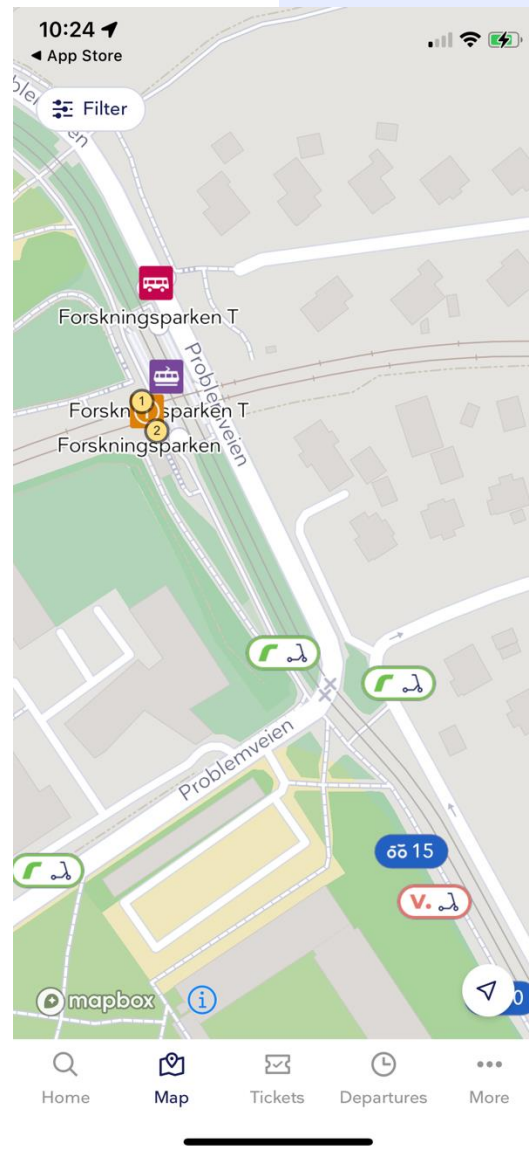
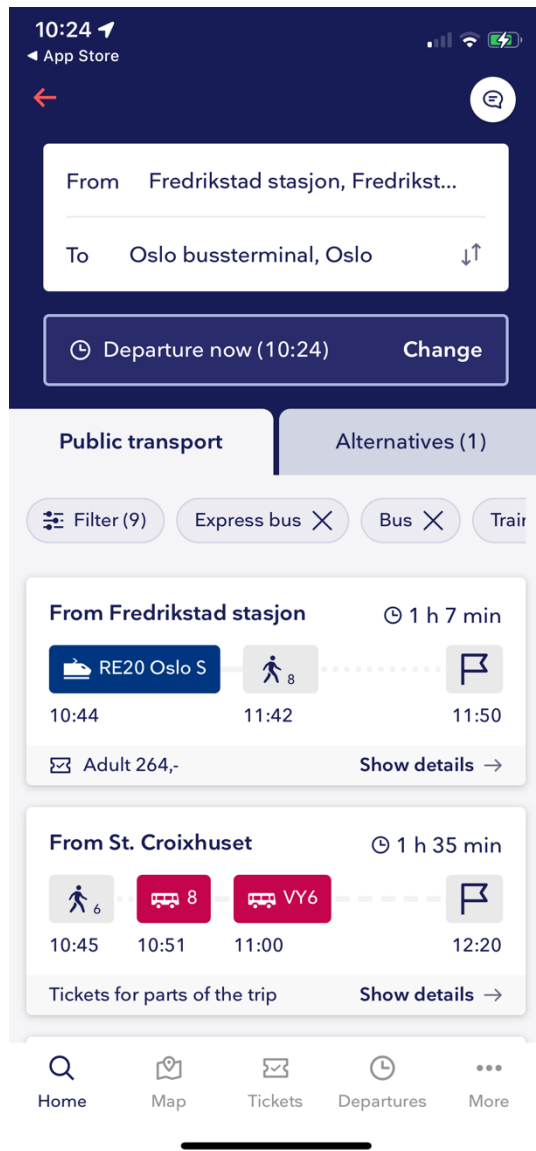


Scrum of Scrums

Don't Know/Other

A large study from 2020 showed that most organizations adapt or use some kind of hybrid version (Edison et al., 2020)

You will hear more about some of the popular frameworks in the lecture on Large-Scale agile and Architecture



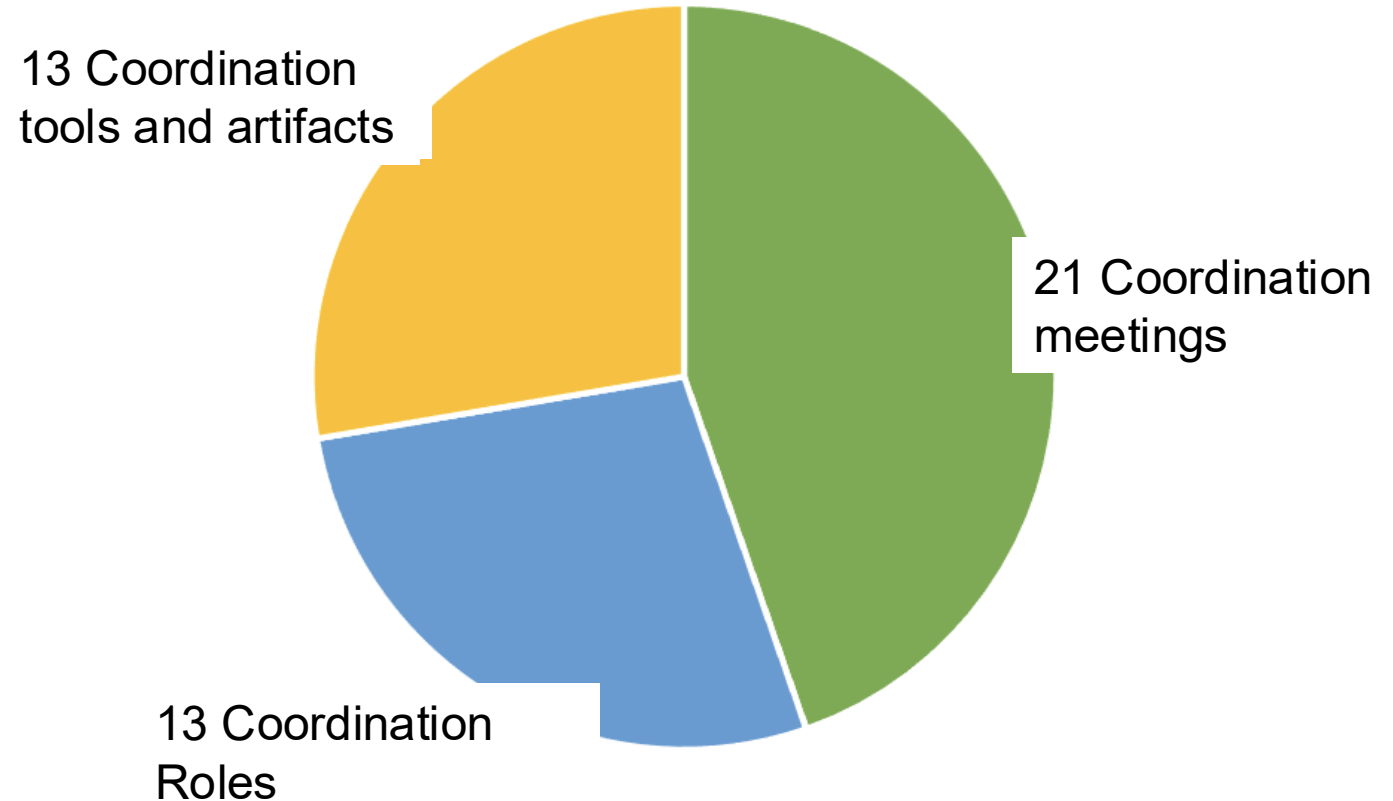
ENTUR



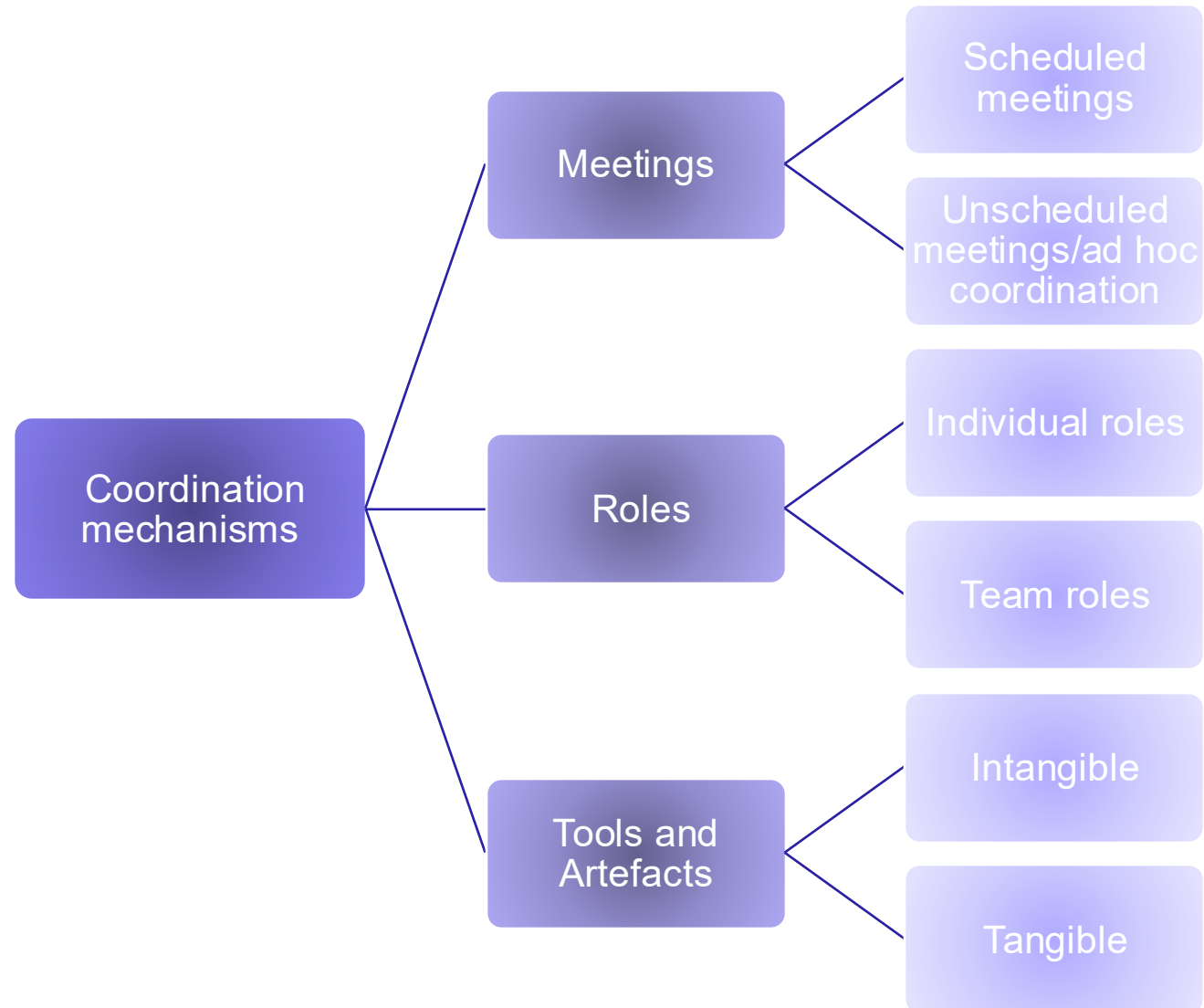
UNIVERSITY
OF OSLO

Coordination mechanisms at Entur

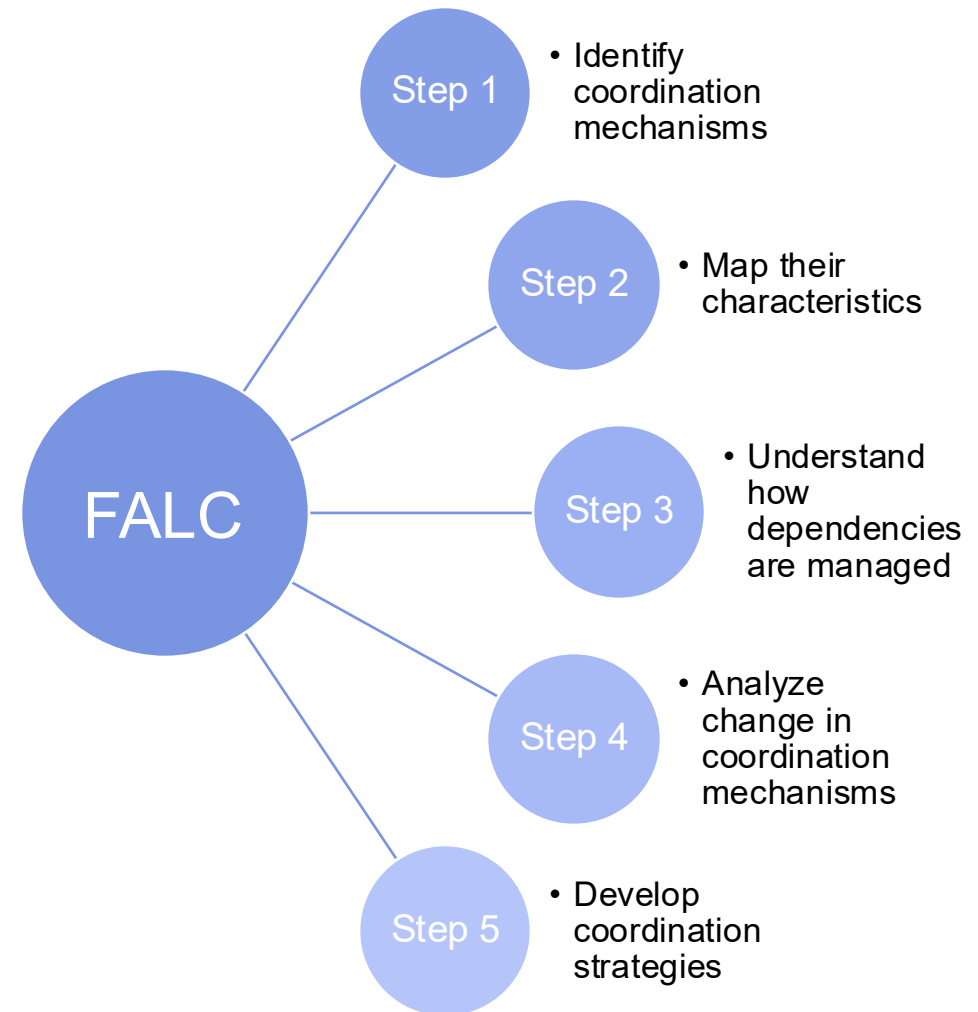
- In total 47 different mechanisms used for some form of inter-team coordination, of which 21 (!) were meetings
- Why so many?
 - Actual high coordination needs
 - Easy to keep status quo,
 - Remember to ask “why” we use the practices we do! 😊
 - *(Attributes of the research process– more on that on Sep 22!)*
- Fewer mechanisms in use today 😊



A taxonomy of inter-team coordination mechanisms (Berntzen et al., 2022)



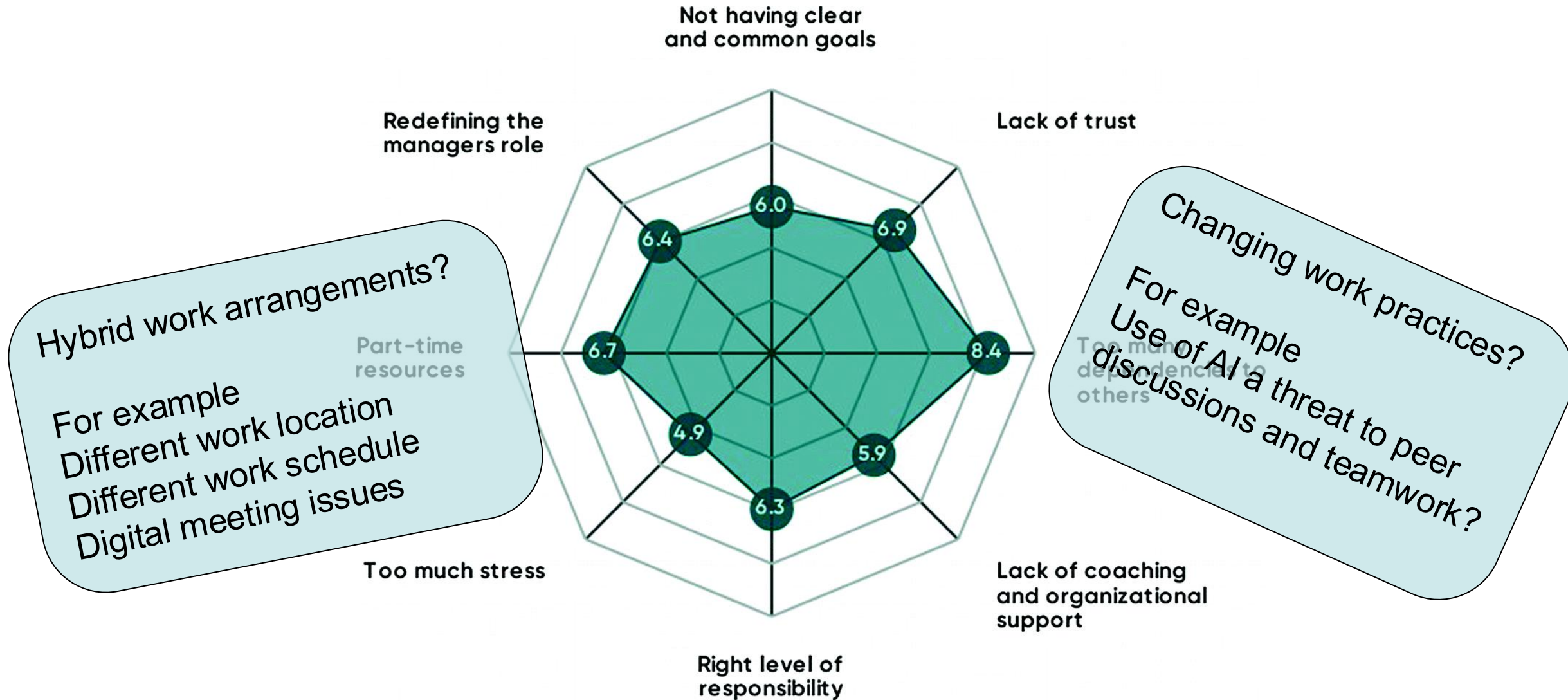
Framework for Analyzing Large-scale agile Coordination mechanisms (Berntzen, 2023)



On October 20th there will be a teaching case about Entur that may or may not be relevant for your exam ;-)

Don't miss out! 😊

Evolving challenges in agile teamwork



AI as a **helpful tool**, or as a **threat** to teamwork?

- How does AI influence software development teamwork?
 - For example, what happens to knowledge sharing/asking your colleague?
 - Those peer discussions are valuable, how do they change when we can ask our AI in stead?



Attitude differences between users and non-users of GitHub Copilot

Copilot users Non-Copilot users

I view Copilot as a valuable tool for developers



I get demotivated by having to complete repetitive programming tasks

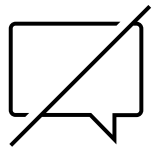


I am dependent on colleagues for help and guidance



I enjoy reading others' code as part of a code review





GenAI can threaten teamwork



This summer, when I tried to work with the one person on the team who likes to use AI, he was more interested in talking to the AI than me. It was comical, but very frustrating, because I felt that then there is no point in us working together. Instead of asking the question to me, he asked the question to ChatGPT.

What are potential usage areas of (generative) AI in agile software development teams?

code reviews

To be helpful, not
responsible

cross checking

efficiently separate key
tasks and assign them to
the correct members.

Code review

Planning

Structuring

Planning and checking
goals

What are potential usage areas of (generative) AI in agile software development teams?

Better? structure

better ideas to share
with team

Code reviews

break down tasks

Summarizing meetings

Documentation

to do list according to
priorities

Help in estimating the
duration of tasks so we
have a general idea how
long it will take

List some reasons why NOT to use GenAI in agile software team

Hallucinations

too much dependency

Less communication

You become unaware of
how your own code
works

Lack of context

Don't think like human

little to no context of the
codebase (unless ?)

Data privacy concerns

List some reasons why NOT to use GenAI in agile software team

Trust

no self improvement,
lack of creative mindset

Teambuilding

Bias

Bad for creativity and
communication

Reduced creativity

no cross questioning

Responsibility relies on
the team.

List some reasons why NOT to use GenAI in agile software team

Slows down
collaboration and
teamwork

peer involvement?

Lack of overarching view
of project

How to incorporate AI in agile teams?

Some research questions we are currently exploring:

- How should AI agents interact with humans and with each other?
- What roles can AI agents fill in teams?
- What team activities can an AI agent participate in?
- How should autonomy and control be distributed between AI-based and human agents?
- How can the team build the skills to include the AI agent in daily tasks?
- How should teams onboard an AI agent?



More in the guest lecture on October 20th! 😊

Thank you for listening!

Marthenb@ifi.uio.no