

## Organizational Matters

Software Engineering II  
WS 2019/20

Enterprise Platform and Integration Concepts

# Communication



## If you haven't yet ...

- Sign up to **mailing list**
- Join **Slack**, teaching team is available
- All links are on the course website

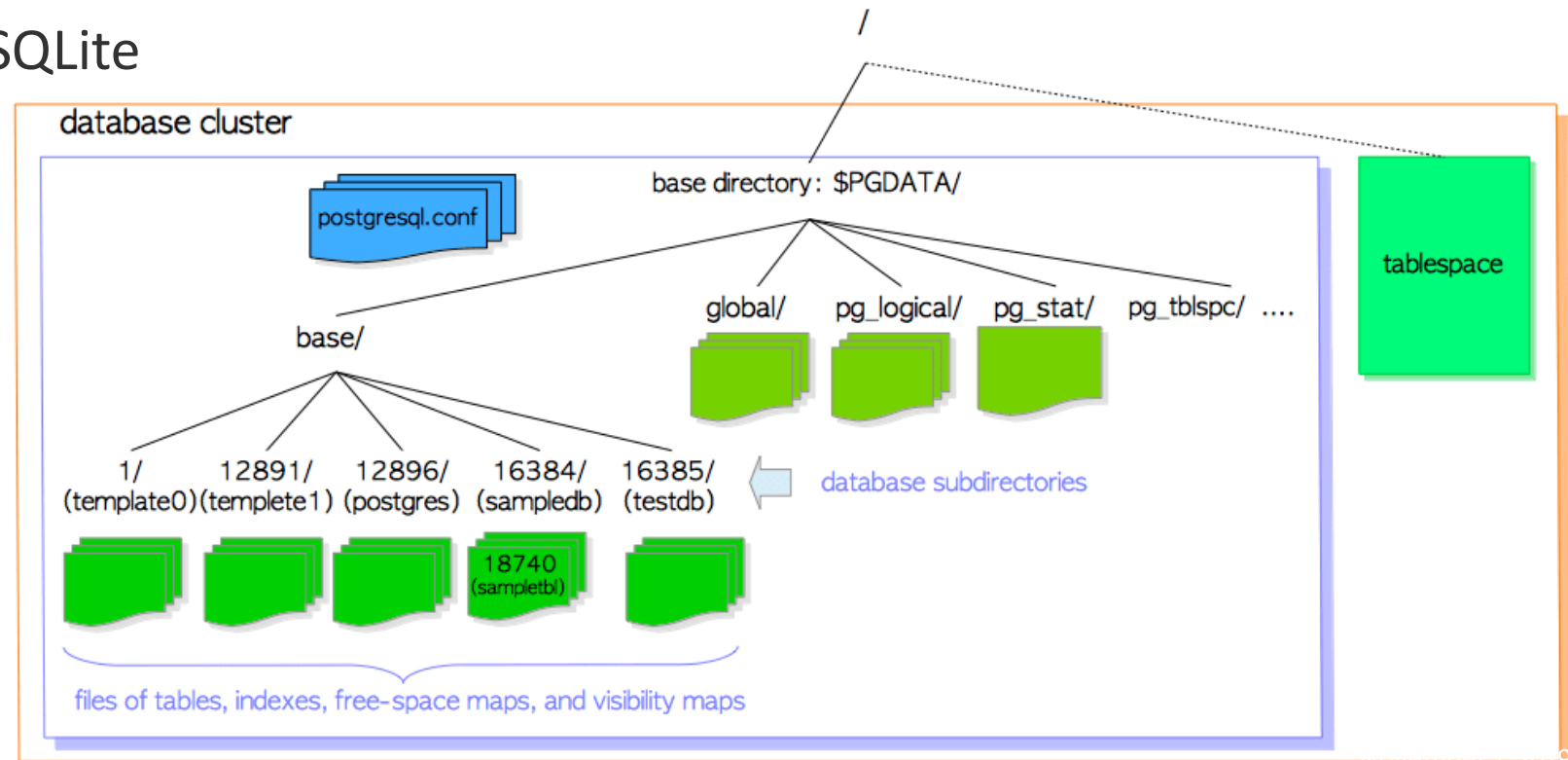
(<https://hpi.de/plattner/teaching/winter-term-201920/softwaretechnik-ii.html>)

A screenshot of a Slack workspace interface. On the left is a dark purple sidebar with the workspace name 'SWTII 2019' and a list of channels including '# general' and '# random'. The main area shows the '#general' channel header with a description: 'You created this channel on October 15th. This is the very beginning of the #general channel. Description: This channel is for workspace-wide communication and announcements. All members are in this channel. (edit)'. Below the header, there are two messages from 'Christoph Matthies'. The first message, dated 'Tuesday, October 15th', says 'joined #general along with 19 others.'. The second message, dated 'Monday, October 28th', says 'Hallo! Die Ruby on Rails Einführungsübung ist nun verfügbar!'. The interface includes a search bar, a member list, and various channel management icons.

# Short Note on DBs

## How databases store data

- Can directly use RAW, i.e. not formatted, partition
  - Oracle, MySQL, Microsoft SQL Server can do this
- The **more common way** is to use the filesystem
  - Own organization, large files can be read and written as required
  - e.g. PostgreSQL, SQLite



<http://www.interdb.jp/pg/pgsql01.html>,

Thanks [Stefan Halfpap](#)

# Next Weeks' Schedule



## Previously

- Introduction lectures
- Testing

## Week Nov 4 – Nov 8

- Finish intro exercise
- Finalize teams + meeting times
- POs: **Customer meeting!**
  - Write initial user stories

## Nov 8 Lecture

- 1st slot: Scrum LEGO Exercise!
  - **Room will be announced!**
- 2nd slot:
  - Kick-off
  - POs present vision
  - Start of project





## Scrum

Software Engineering II  
WS 2019/20

Enterprise Platform and Integration Concepts

# Scrum



- 1. The Case for Agile**
2. The Scrum Process
3. Scaling Scrum

# How Projects Fail



- Delivering late
- Delivering over budget
- Delivering the wrong thing
- Unstable in production
- Costly to maintain

# Why Projects Fail



- **Smart people trying to do good work**
- Stakeholders are well intended

Process in traditional projects



- Much effort for
  - Documents for formalized hand-offs
  - Templates
  - Review committees



# Why Projects Fail



*“The later we find a defect, the more expensive it is to fix it!”*

**Does front-loading a software development process make sense?**

Reality shows:

- Project plans are wonderful
- Adjustments & assumptions are made during analysis, design, code
- Re-planning takes place
- Example: Testing phase at the end
  - Tester raises a defect
  - Programmer claims he followed the specification
  - Architect blames business analyst etc.
  - Exponential cost

# Why Projects Fail



- People are afraid of making changes to project plan
- Unofficial changes are carried out
- Documents get out of sync
- ...

Again, why do we do that!?

*To minimize the risk of finding a defect too late...*

# A Self-Fulfilling Prophecy



- Traditional front-loaded process to minimize costs of change
  - Project plan
  - Requirements specification
  - High-level design documents
  - Low-level design documents
  - Idea: Specify everything, then execute
- This process can cause exponential costs of change
  - A self-fulfilling prophecy

*This makes sense for a bridge, ship, or a building but  
**software (and Lego) are easy to change!***

# The Agile Manifesto



We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

*Individuals and interactions **over** processes and tools*  
*Working software **over** comprehensive documentation*  
*Customer collaboration **over** contract negotiation*  
*Responding to change **over** following a plan*

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

<http://agilemanifesto.org/>

# How Agile Methods Address Project Risks



## **No longer late or over budget**

- Tiny iterations
- Easy to calculate budget
- High-priority requirements first

## **No longer delivering the wrong thing**

- Strong stakeholder communication
- Short feedback cycles

# How Agile Methods Address Project Risks



## **No longer unstable in production**

- Delivering each iteration
- High degree of automation

## **No longer costly to maintain**

- Maintenance mode starting with Sprint 2
- Maintenance of multiple versions during development

# The Cost of Going Agile



## **Outcome-based planning**

- No complete detailed project plan

## **Streaming requirements**

- A new requirements process

## **Evolving design**

- No complete upfront design: flexibility required
- Emergent Design

## **Changing existing code**

- Need for refactoring

# The Cost of Going Agile



## **Frequent code integration**

- Continuous integration

## **Continual regression testing**

- Add nth feature; test n-1 features

## **Frequent production releases**

- Organizational challenges

## **Co-located team**

- Easy communication, keep momentum



# Discuss!



## Pros and Cons

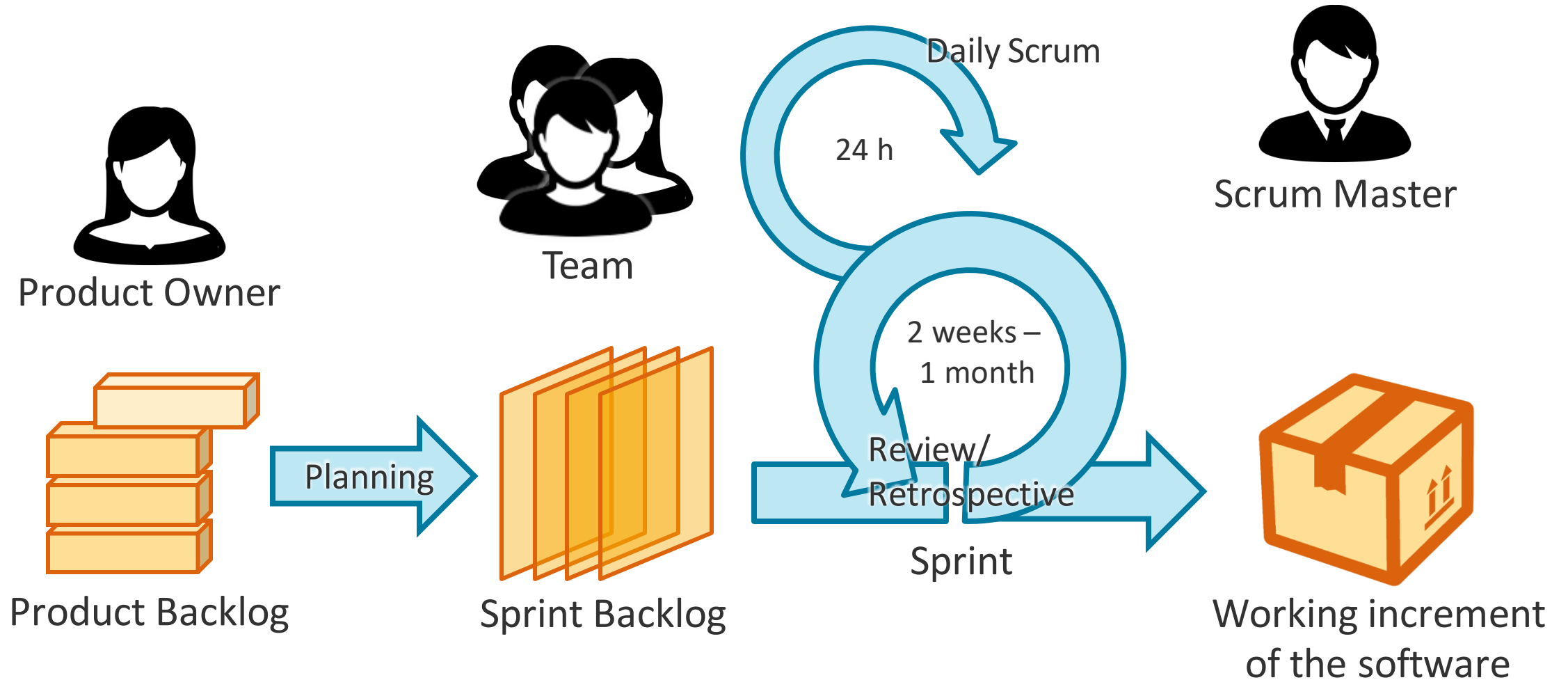
- Short planning horizon
- No up-front design
- Stories instead of requirement documents
- Extreme ideology

# Scrum

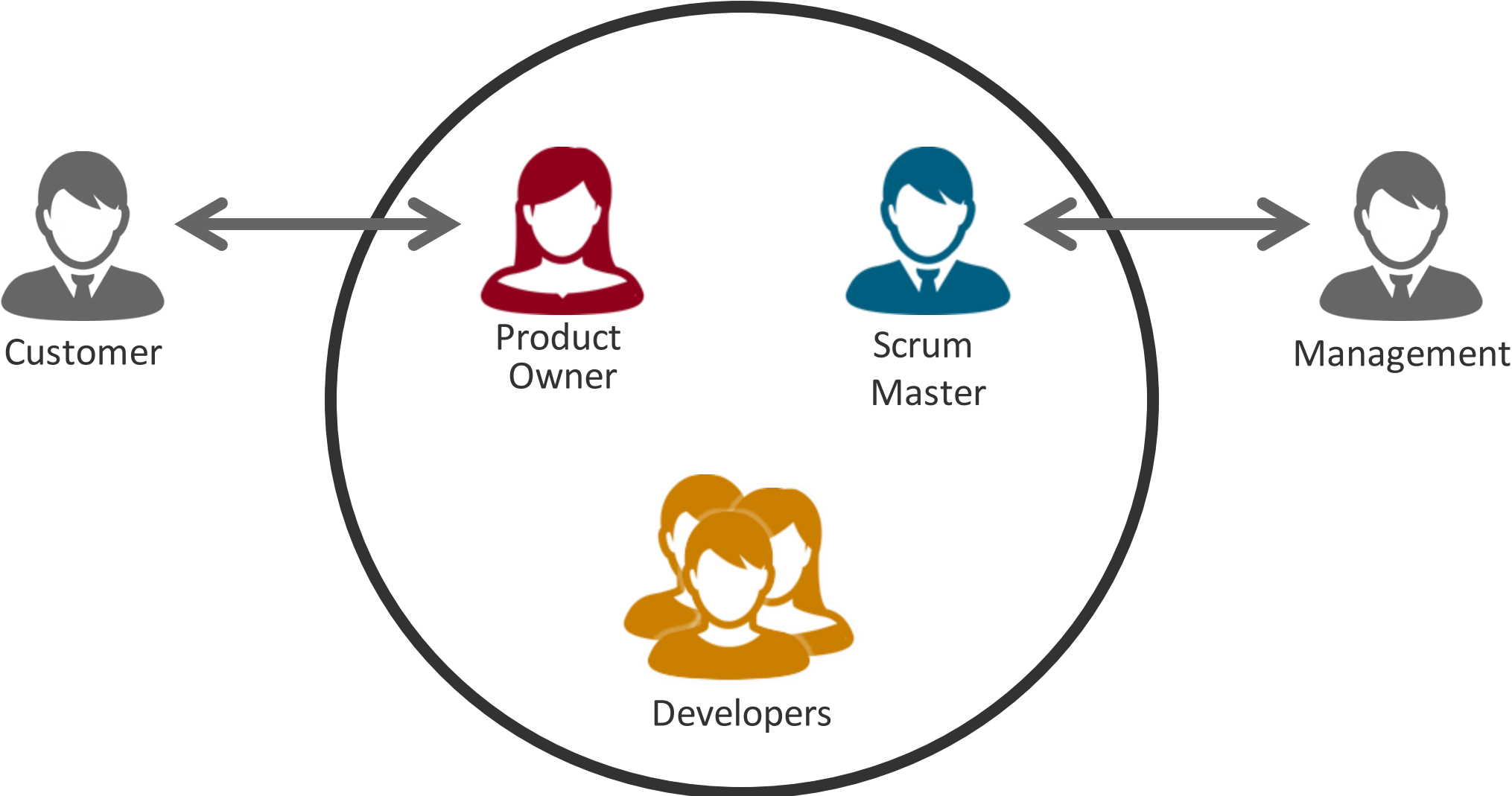


1. The Case for Agile
2. **The Scrum Process**
3. Scaling Scrum

# Scrum



# The Team



Customer

Product  
Owner

Scrum  
Master

Management

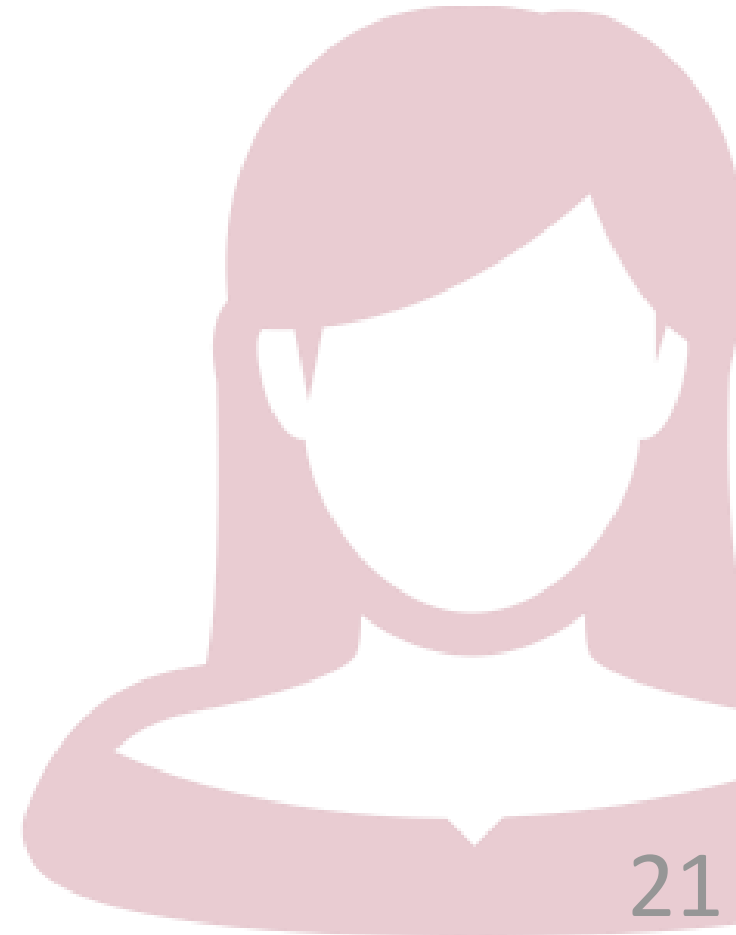
Developers

# Product Owner



## Responsibilities

- **Customer** communication
  - Contact person for team
- Product Backlog
  - **User Stories**
  - Priorities
- Acceptance Criteria & Tests



# Scrum Master



## Responsibilities

- **Process** manager
  - Moderator in meetings
- Management communication
  - Remove **impediments**
- Enabler, not boss



# Developers



## Responsibilities

- Communication
  - **Critically** discuss all inputs
  - Honestly share important information
  - Represent team as expert
- Sprint Backlog
- Developing ;-)

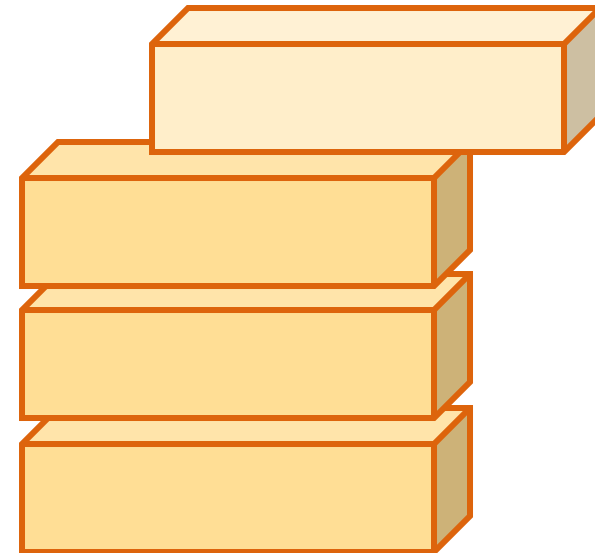


# Product Backlog



## List of work items

- Requirements (modification **requests**)
  - Features
  - Bug fixes
- Ordered/**prioritized**





# Requirements



In Scrum, requirements are often defined as **user stories**:

“As <role>, I want <feature> to <reason>”

Requirements need to fulfill **INVEST** properties:

- I
- N
- V
- E
- S
- T

# Requirements



In Scrum, requirements are often defined as **user stories**:

“As <role>, I want <feature> to <reason>”

Requirements need to fulfill **INVEST** properties:

- I – Independent
- N – Negotiable
- V – Valuable
- E – Estimable
- S – Small
- T – Testable

# Planning Meeting



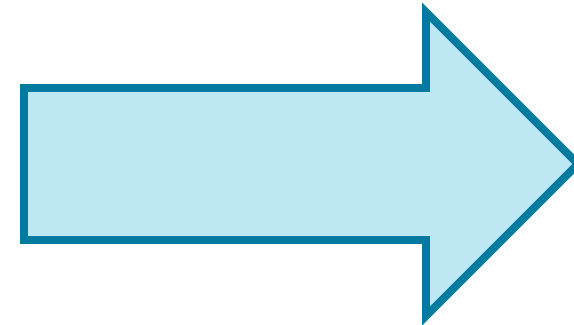
## Filling the sprint

- Estimate Backlog items
- Move items from Product to **Sprint Backlog**

## Defining the work

- **Break down** Backlog items into tasks
- PO not required

Total time: 2 hours per week of sprint



For better planning, stories are broken down into tasks

Tasks should be **SMART**:

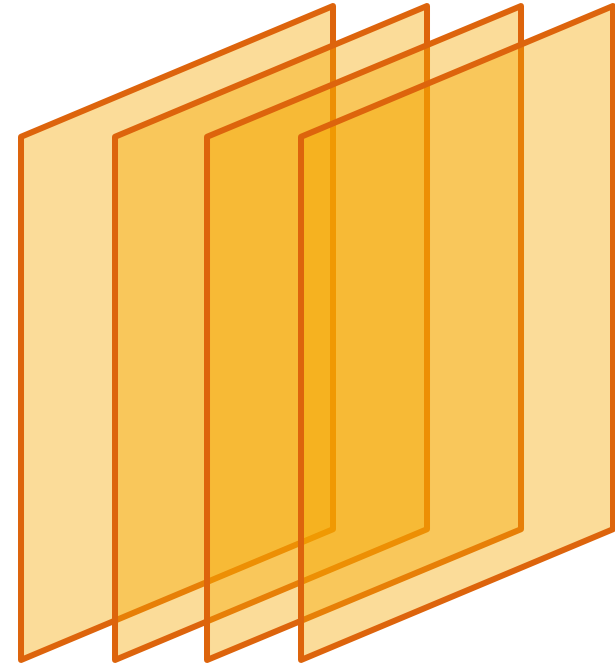
- S – Specific
- M – Measurable
- A – Achievable
- R – Relevant
- T – Time-boxed

# Sprint Backlog



## List of tasks for a sprint

- Tasks are **signed-up** for, not assigned
- During the sprint
  - **No new features**
  - Team may change/add tasks



# Daily Scrum Meeting



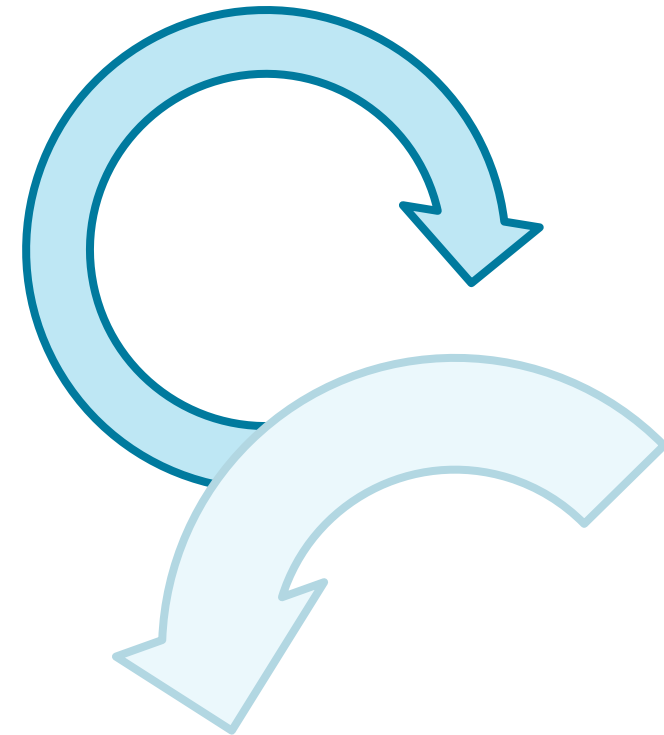
## Status update

- Last achievements
- Next steps
- Problems

Max. **2 min** per person

## Discussions?

- Schedule **subsequent** expert's meeting

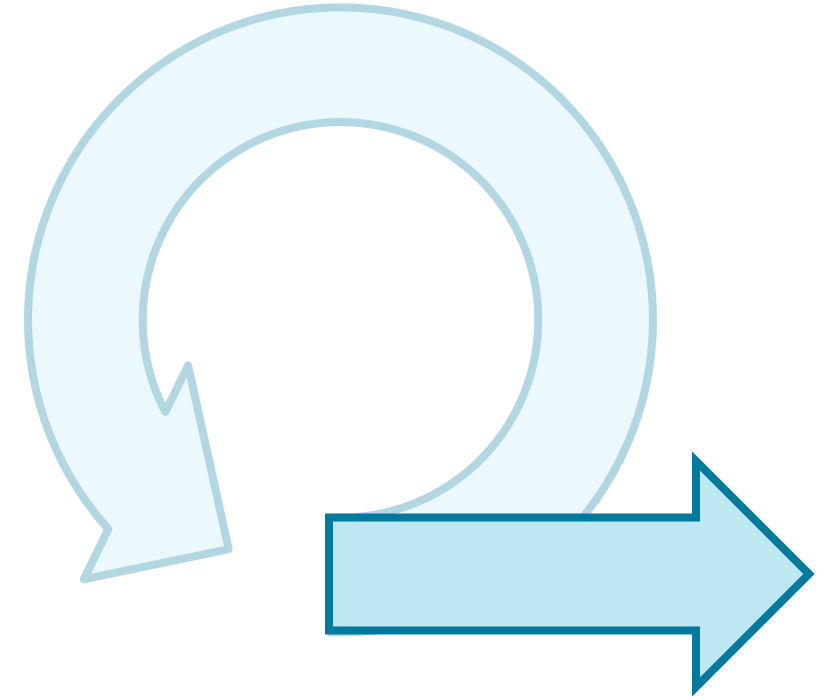


# Review Meeting



## Acceptance of Features

- Demo to PO
  - PO should be prepared
  - Optional: invite other stakeholders
- Comments by developers

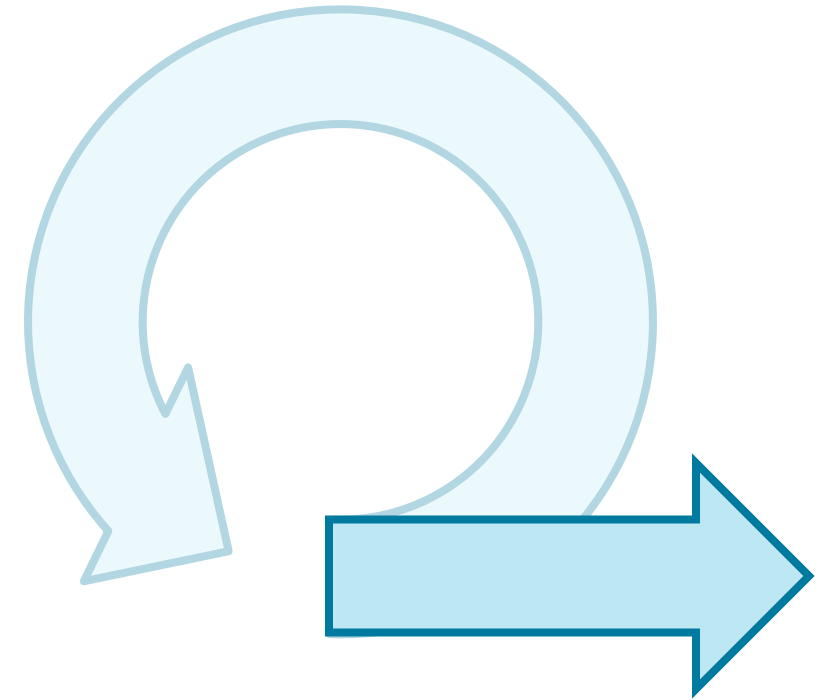


# Retrospective Meeting



## Internal team evaluation

- PO not required
- Discuss process and problems
- **Measure** improvements



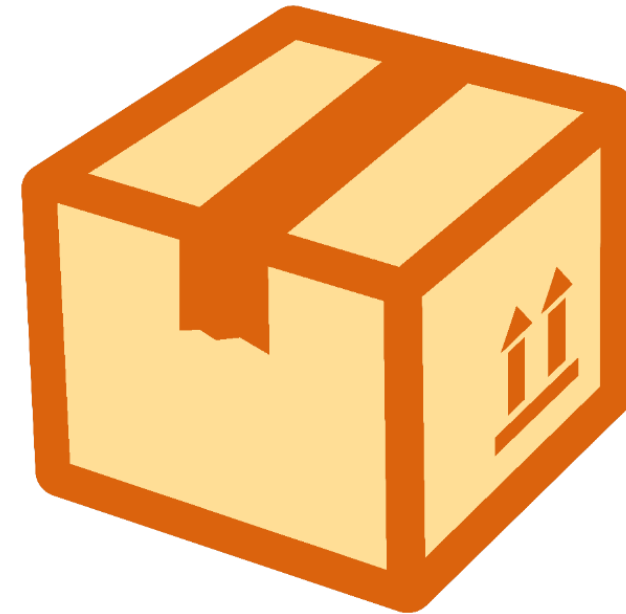


# Product Increment



## Potentially shippable increment

- Complete according to **Definition of Done**
  - Even if not actually released
- **No regrets** if project ended now



# Scrum



## Team

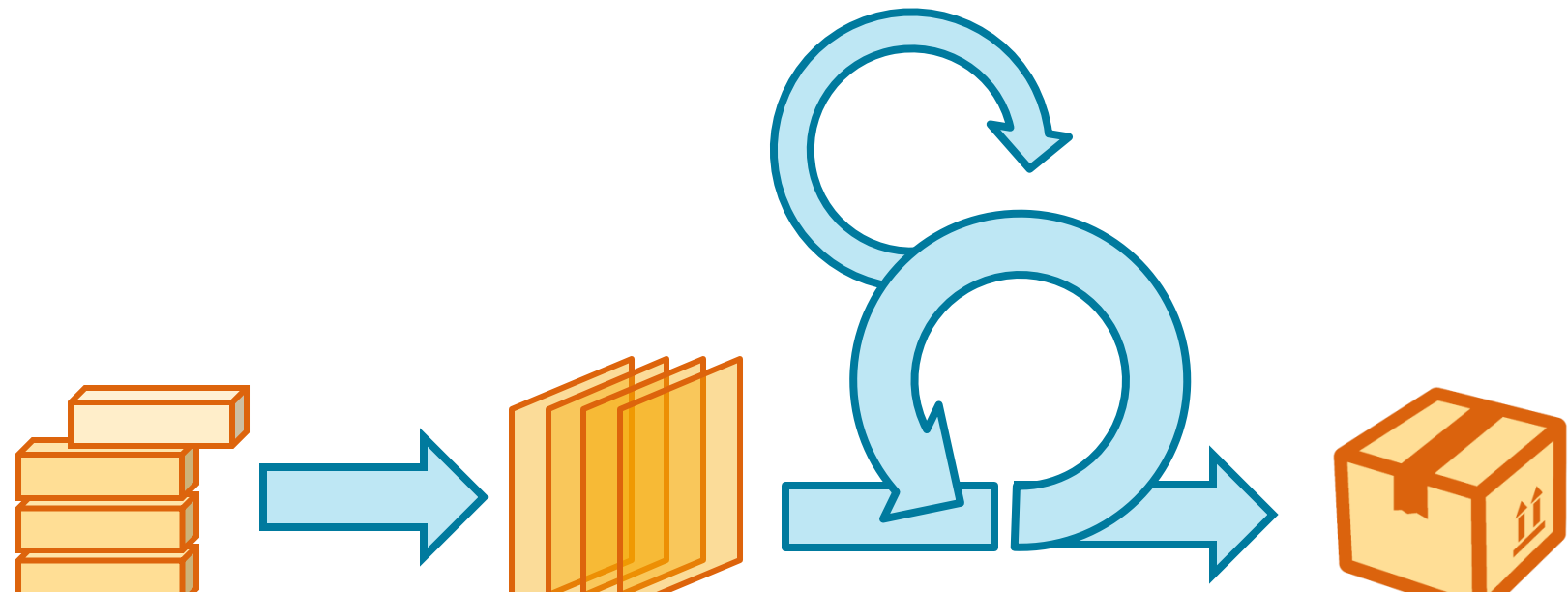
- Product Owner
- Scrum Master
- Developers

## Meetings

- Planning
- Daily Scrum
- Review
- Retrospective

## Artifacts

- Product Backlog
- Sprint Backlog
- User Stories
- Software Increment



# Effort, Schedule, and Cost Estimation



- Depends on software engineering process
- Highly **uncertain**, must be negotiated and revised with stakeholders
  
- Waterfall effort estimation
  - Methods: calibrated estimation model based on historical size (Function Points, LOC, ...); expert judgment; ...
  - Output: X man-months
- Agile effort estimation
  - **Iterative** methods, **shorter** planning horizon
  - Output: functionality to be implemented in the **next iteration**
  - Different methods exist

# Effort Estimation: “Planning Poker”



## Participants

- **Everyone** operationally involved in creating the software product
- Product Owner (and Scrum Master) are not playing

## Preconditions

- Product backlog is complete and **prioritized**
- Backlog items are known by the team
- The effort for a small backlog item was determined as a **reference**
- Every participant has a set of sizing cards



# Planning Poker 1/2



- Product owner explains backlog item
- Product owner **answers questions** of team members
- Participants estimate complexity of item and choose a card (**hidden**)
- All cards shown simultaneously
- Participants with highest and lowest number **explain choices**
- The arguments are **discussed** in the group

# Planning Poker 2/2



- A new vote is conducted
- **Team agrees** on item size
  - Most occurring or average value is acceptable
  - If not, another round is played
- The moderator notes size of backlog item in the product backlog
- The game ends if all backlog items are sized or **time is over**

# Effort Estimation: “Affinity Estimation”

## ■ Participants

- **Everyone** operationally involved in creating the software product
- Product Owner (and Scrum Master) are not participating, but are present for questions

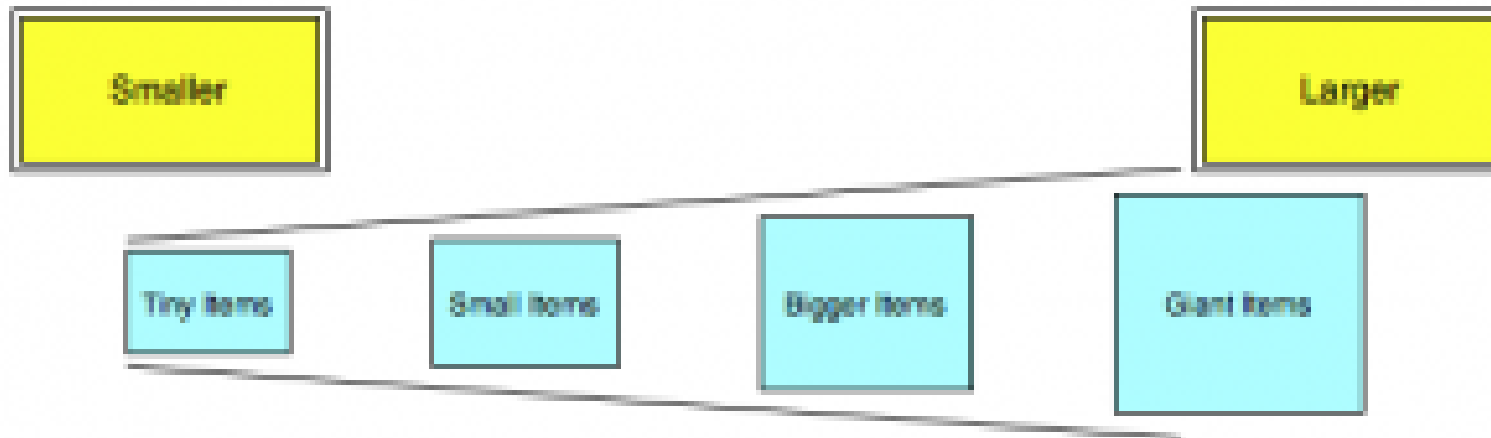
## ■ Preconditions

- Product backlog is complete, **prioritized** and understood
- A shared space to work in
- User Stories in physical form (e.g. post-it notes or printed)

# Affinity Estimation 1/2

## ■ Step 1: **Silent** Relative Sizing

- Team members place backlog items on scale of “smaller” to “larger”
- No discussion at this point





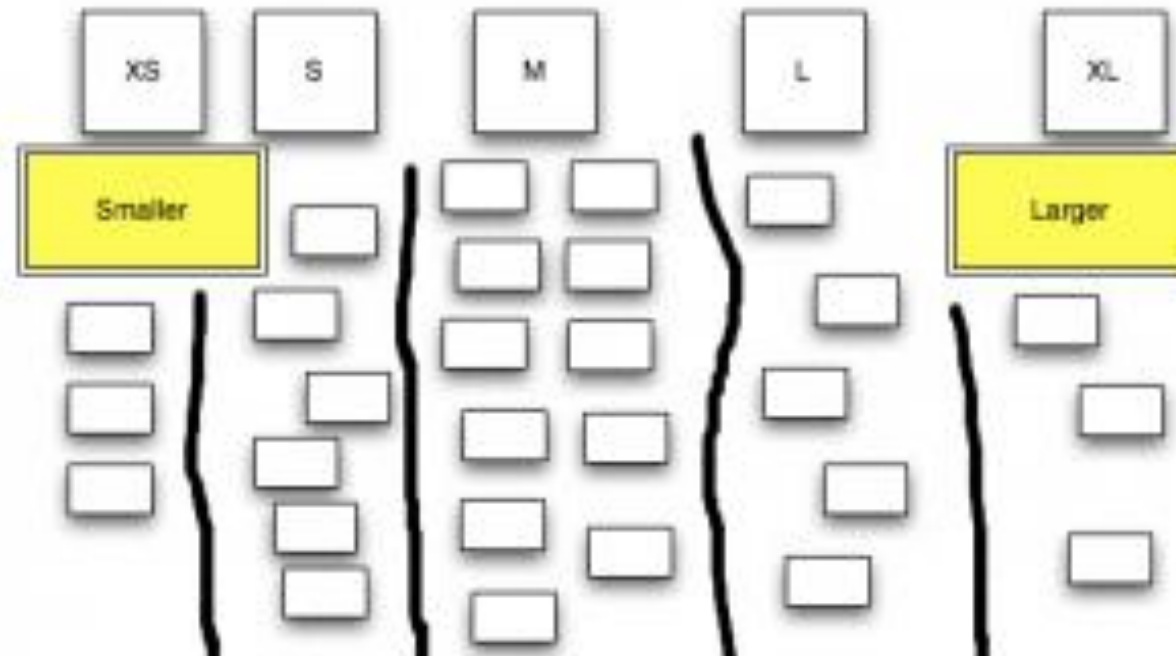
# Affinity Estimation 2/2

## ■ Step 2: **Editing**

- Team members rearrange stories on the scale, discuss changes
- Clarifications from PO

## ■ Step 3: Place stories into **categories**

- Place size categories (e.g. Fibonacci sequence) above scale
- Assign each story a size based on location



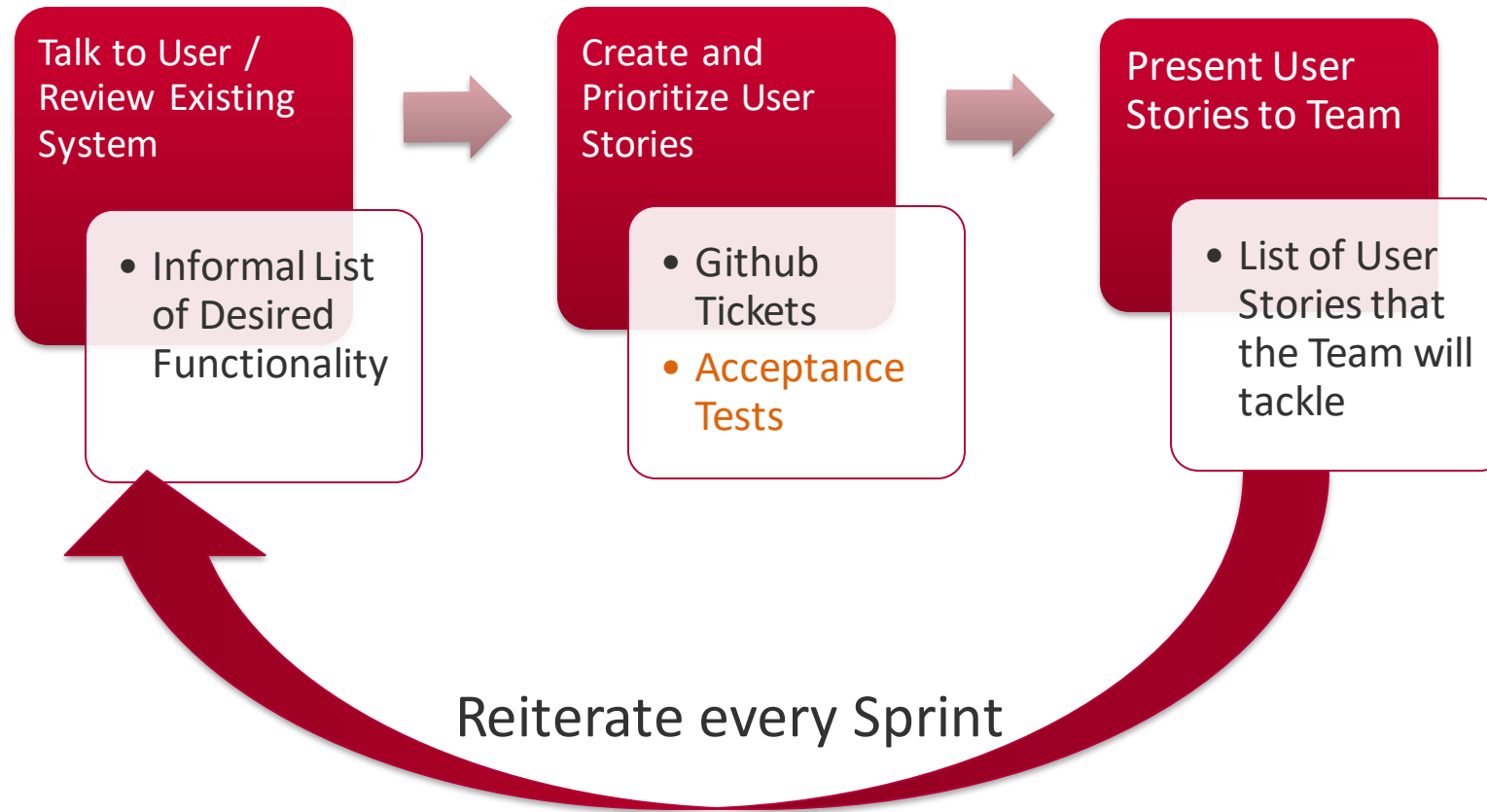
# After the Planning Meeting



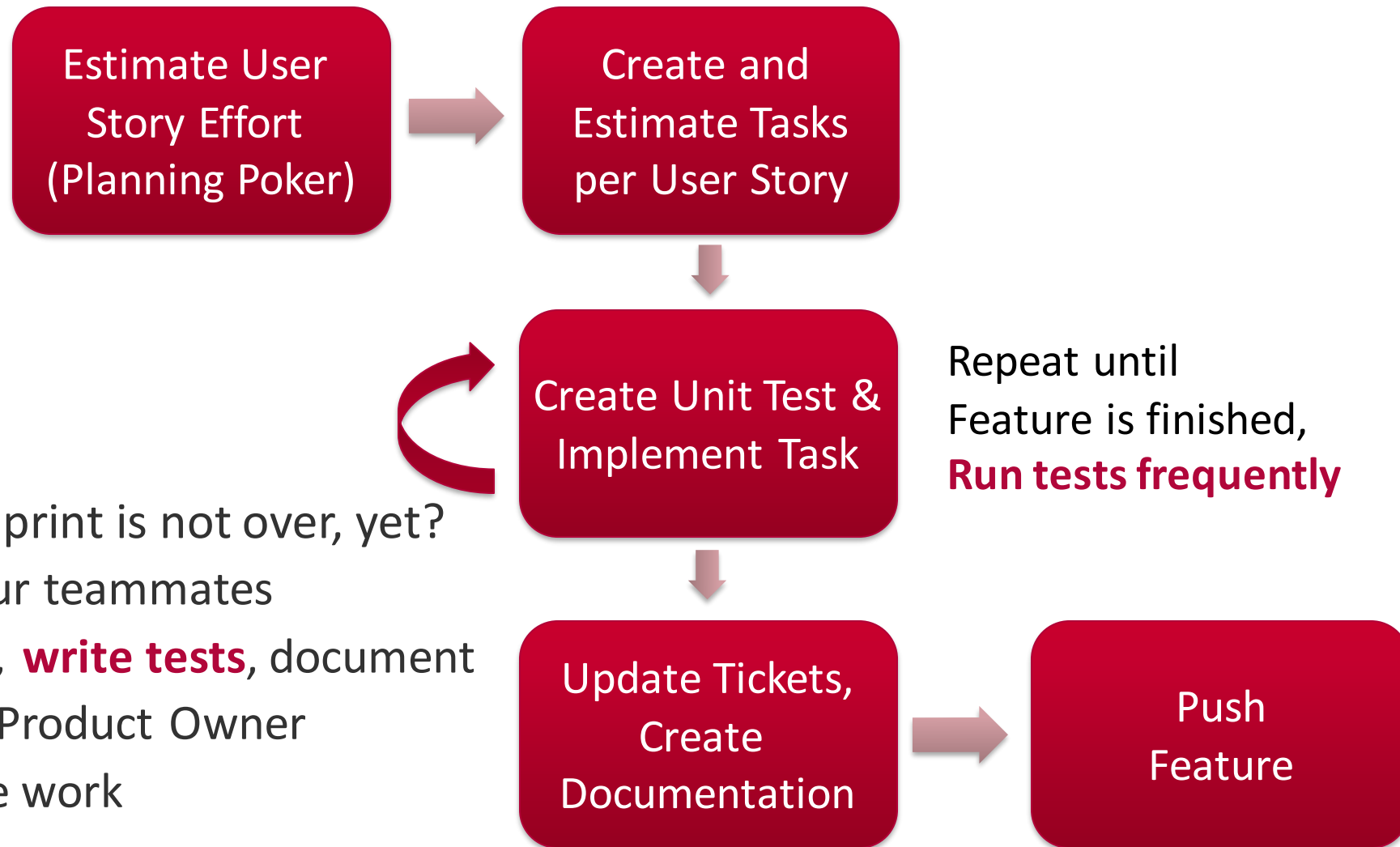
## Begin the sprint

- Select stories until sprint is full
- Break down stories into tasks and fill your **Scrum Board**
- Assign stories to developer(s)
- **Implement** the stories task by task

# Project Workflow: Product Owner



# Project Workflow: Developers



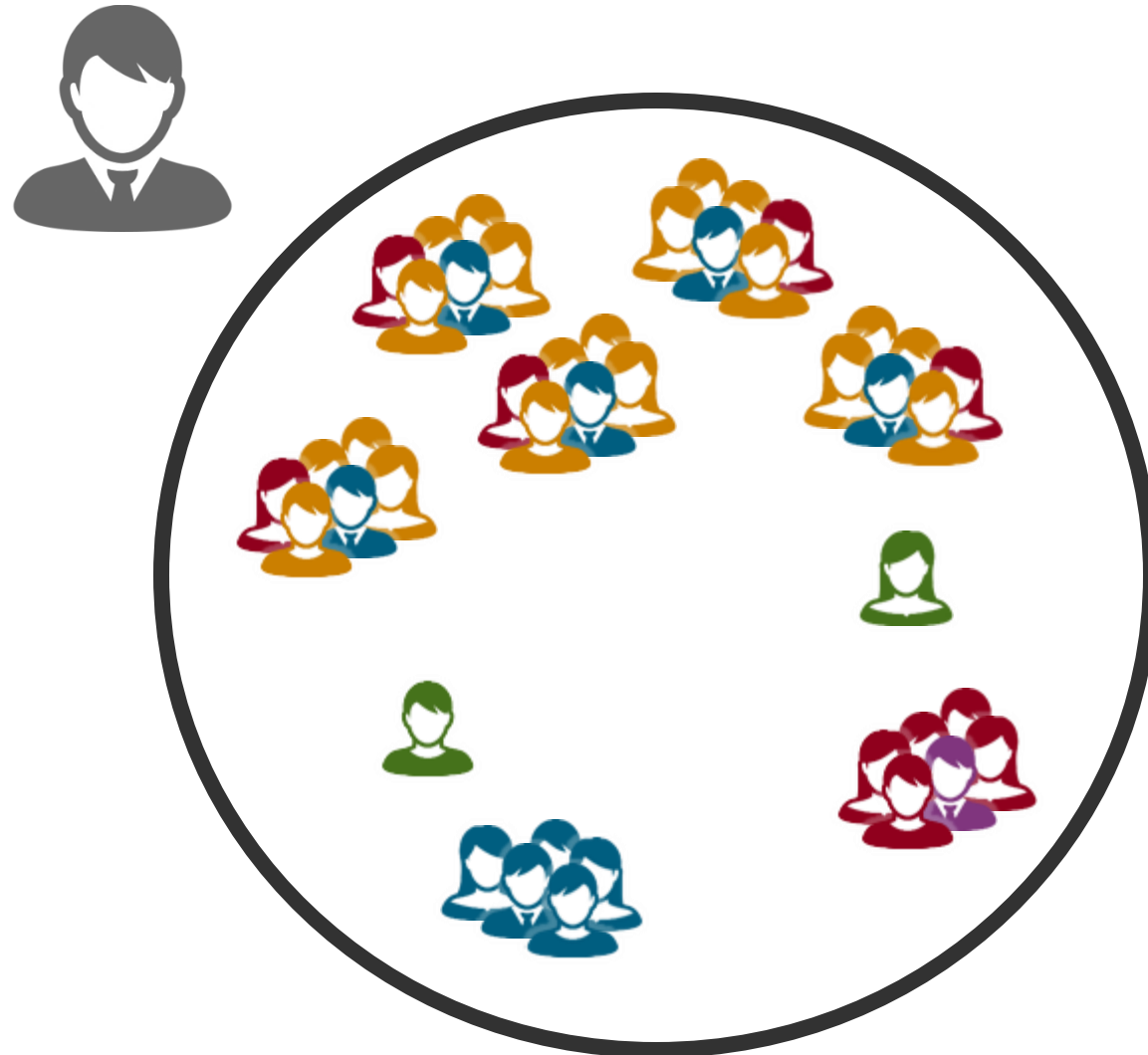
Done and sprint is not over, yet?

- **Help** your teammates
- Refactor, **write tests**, document
- Ask the Product Owner for more work



## Scaling Scrum

# Recap: SWTII High-level Overview



# Implications of the Setup



## What's needed in such an environment?

- Development **process**
- **Communication** on multiple levels
- Infrastructure for **collaboration**

# Scaling Scrum: Project Start



## **Start small and grow organically**

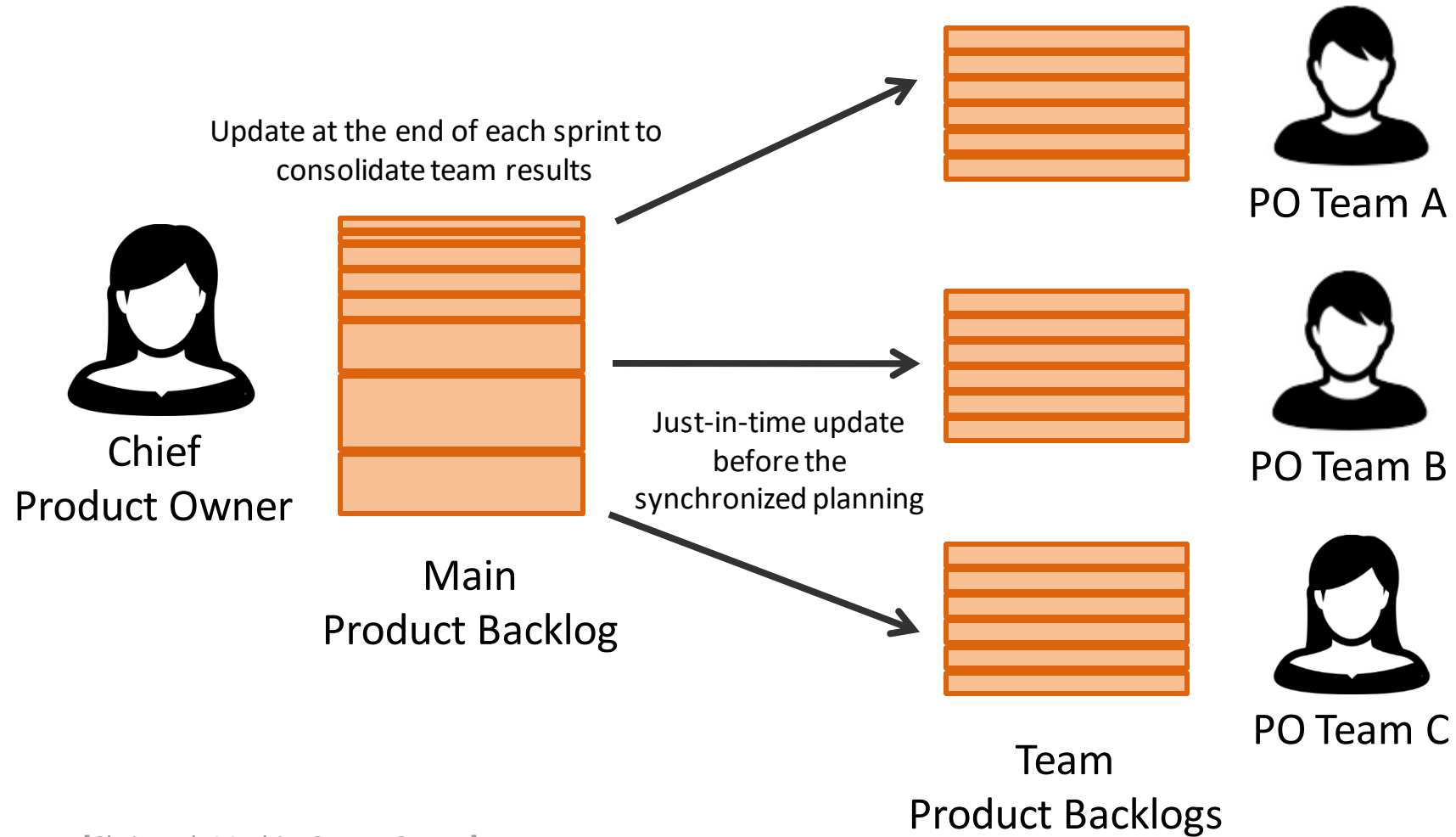
- Single Scrum (teaching) team for preparation
- Work out foundation for the first sprints
- Scale when it becomes necessary

## **We are now at the first scaling point**

- SWT II participants take over!



# Product Owner / Backlog Hierarchy



[Christoph Mathis, Scrum Center]

# Scaling Scrum: Sprint Planning

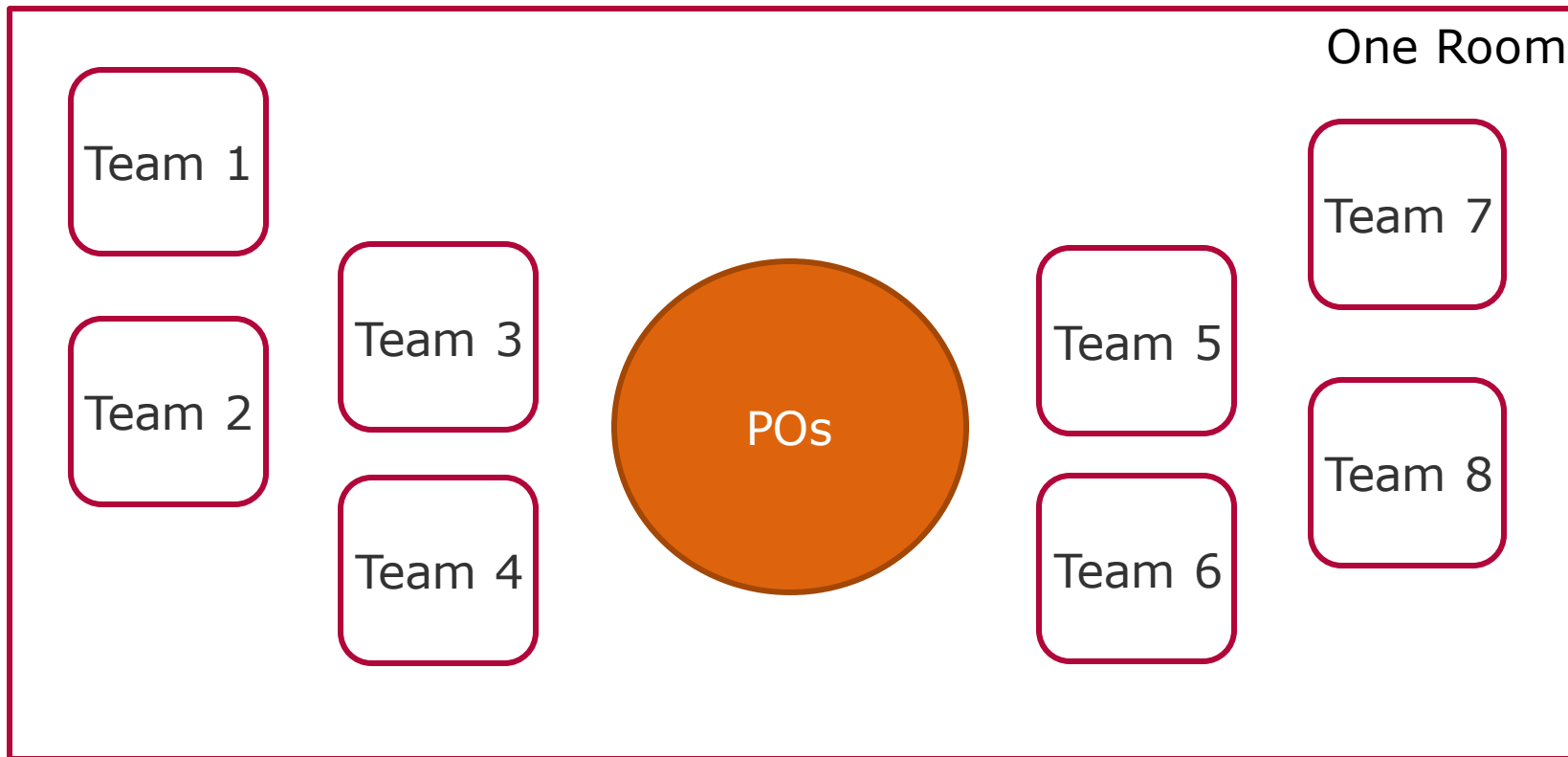


- Preparation
  - Individual review and retrospection meetings
  - Meeting of all teams with 1-2 members each:
    - Review of the last sprint
    - Input dependencies (What is needed)
    - Output dependencies (What needs to be delivered)
- Execution
  - Individual plannings (strict timeboxing)
  - Discussion of identified additional input or output dependencies
  - Final sprint planning
- Problem: Time consuming & high degree of coordination needed!

# Scaling Scrum: Sprint Planning



Another Option: Co-located planning



# Scrum of Scrums



## Goal: Synchronize team effort with minimal coordination overhead

- Regular meeting of all Scrum masters.
  - Developers join if necessary (**ambassador principle**)
- Scrum masters
  - Share their learnings
  - Report completions & next steps
  - Coordinate inter-team dependencies
  - Negotiate responsibility
- Developers discuss technical interfaces across teams
- Distribute information back into the teams

# Scrum

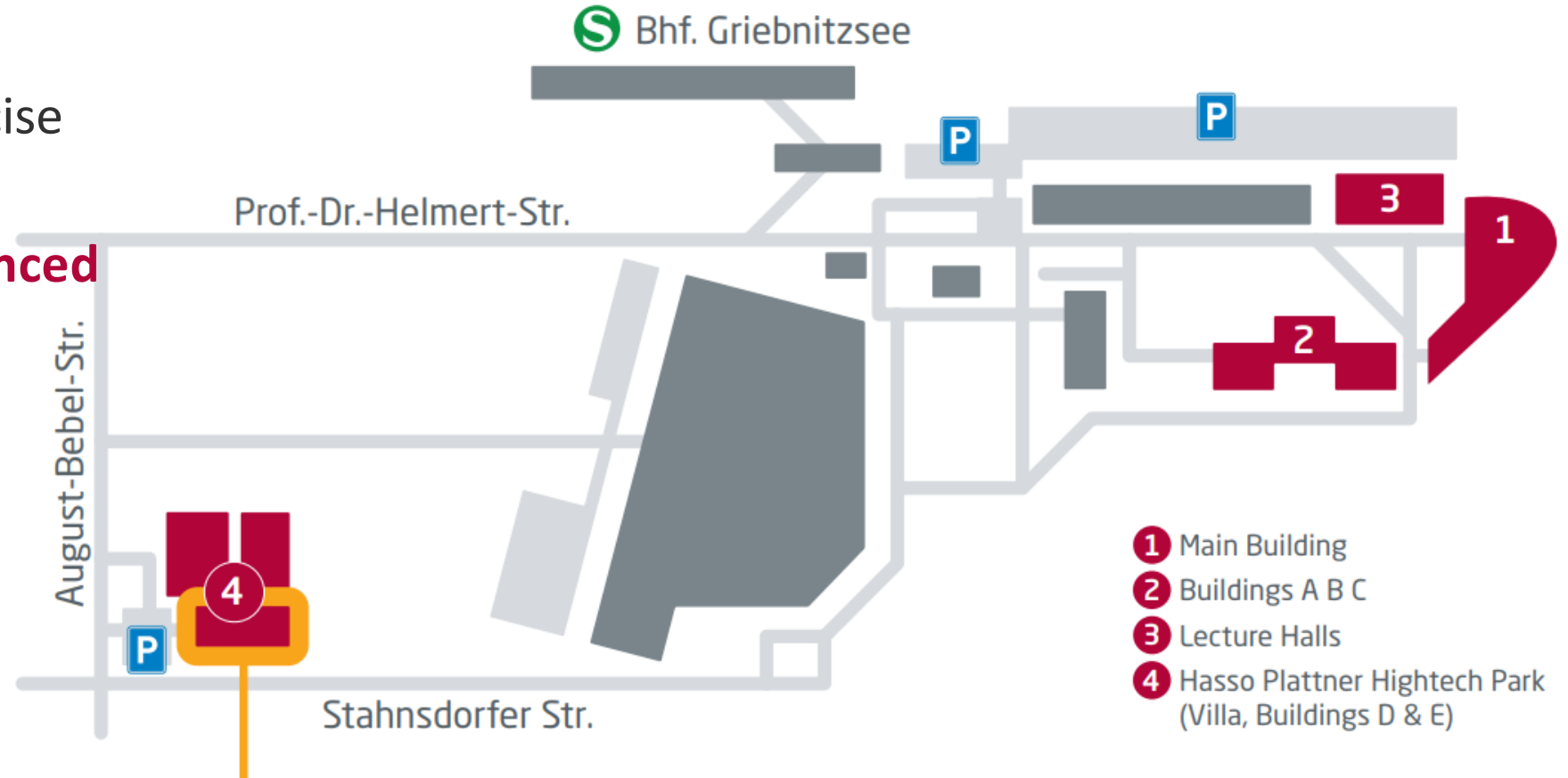
1. The Case for Agile
2. The Scrum Process
3. Scaling Scrum

Questions?

# Next: Scrum Exercise



- Practical Scrum Exercise
- **09:15**
- **Room will be announced**  
Not the lecture hall!



**HPI D-School**  
Hasso Plattner Hightech Park  
Building D