Scrum Process in Teams: Details

Scalable Software Engineering
WS 2021/22

Enterprise Platform and Integration Concepts

Effort, Schedule & Cost Estimation

Estimations and schedules in Software Engineering

- Depend on software development process
- Highly **uncertain**, must be negotiated and revised with stakeholders

Traditional effort estimation

- Methods: calibrated estimation model based on historical data, e.g. Function Points, LOC or expert judgment
- *Output*: X man-months

Agile effort estimation

- **Iterative** methods, **shorter** planning horizon
- *Output*: functionality to be implemented in the **next iteration**

Planning Poker

Participants

- **Everyone** operationally involved in creating the software product
- Product Owner (and full-time SMs) might not participate

Preconditions

- Product backlog is filled 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

Process

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

Planning Poker

- **Team agrees** on item size
  - Most occurring or average value might be 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 estimated** or **time is over**

Affinity Estimation

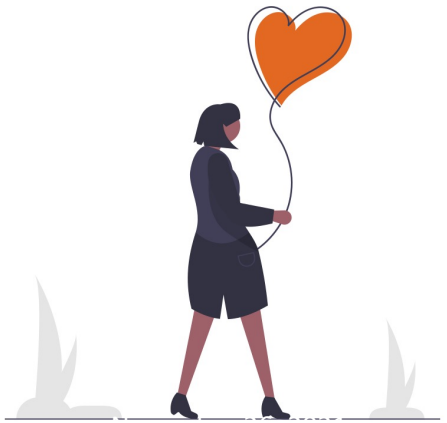
Participants

- **Everyone** operationally involved in creating the software product

Preconditions

- Product backlog is complete, **prioritized** and understood
- A shared space to work in
- User Stories that can be moved around

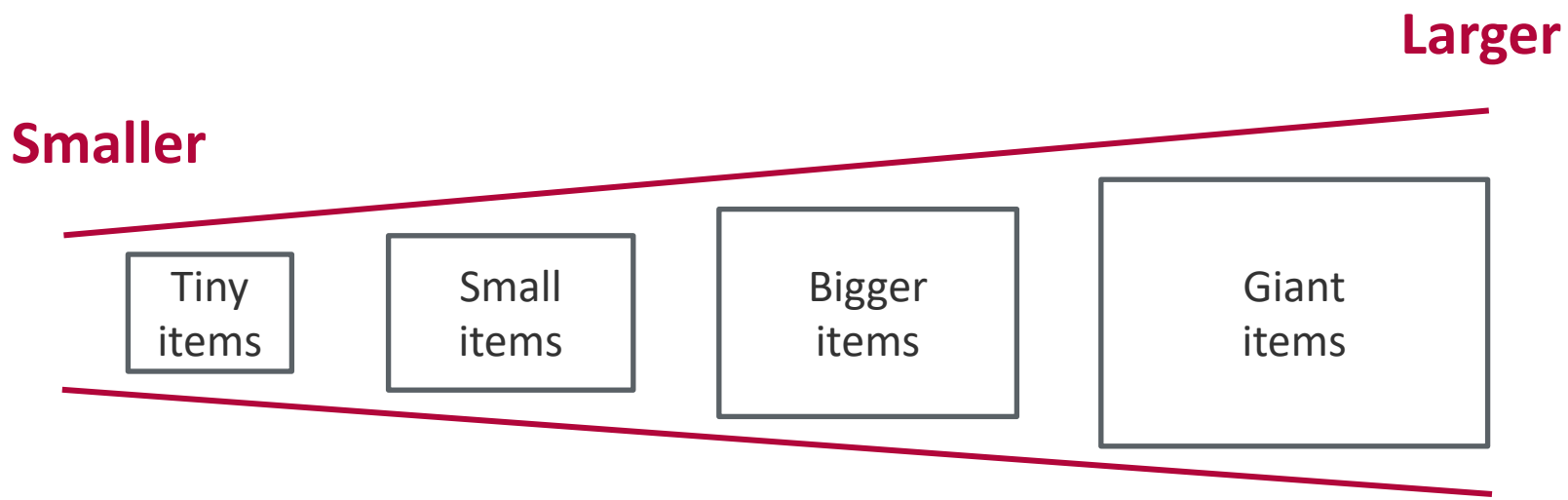
  e.g. post-it notes, print-outs, notes in virtual workspace



Affinity Estimation

Step 1: Silent Relative Sizing

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



http://www.gettingagile.com/2008/07/04/affinity-estimating-a-how-to/

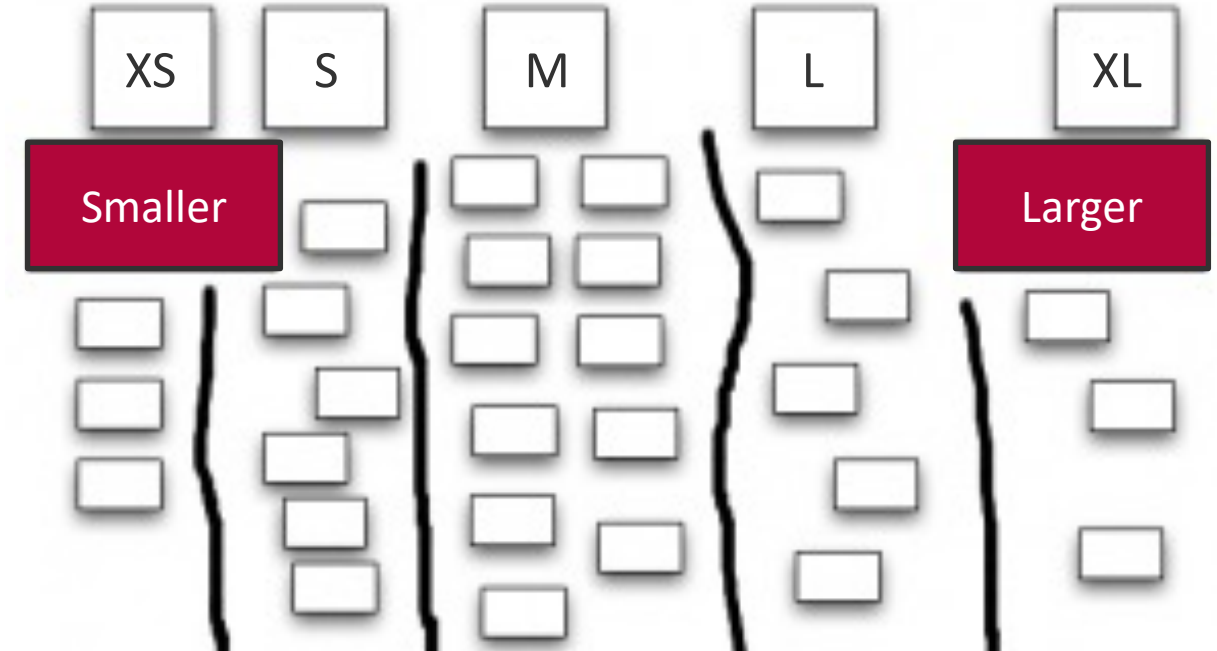
Affinity Estimation

Step 2: Editing

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

Step 3: Place stories into categories

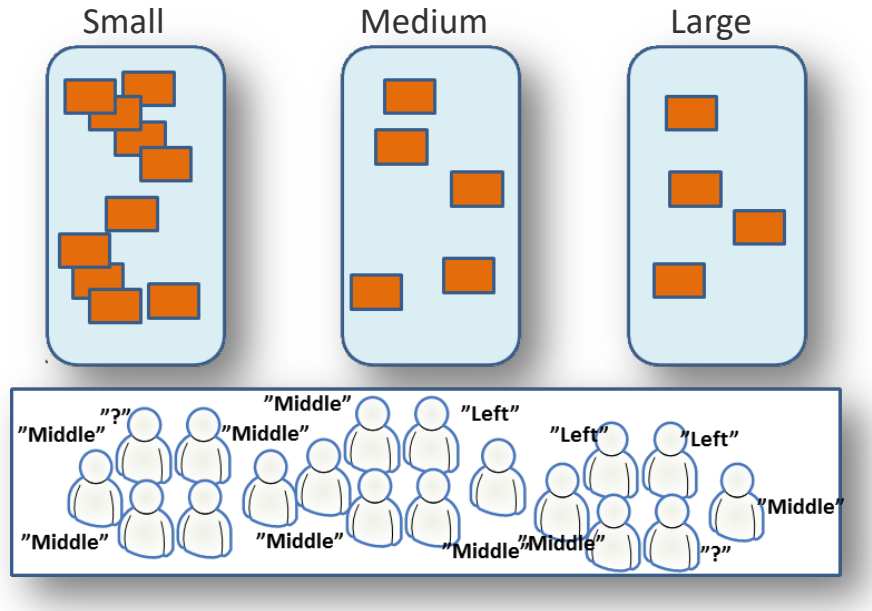
- Place size categories (e.g. Fibonacci sequence) above scale
- Assign each story a size based on location



Estimating Large Backlogs

Bucket Estimation

- **Relative** estimation
- Quickly place items into **few buckets of radically different sizes**
  - E.g. T-Shirt sizes (S, M, L, XL)
  - Quickly present an item, ask the crowd to point to a bucket
- **Estimate sample items** from buckets to determine size of average item
  - Max. 2-3 items per bucket
  - Break up into smaller groups
  - Estimate using a regular approach, e.g. planning poker



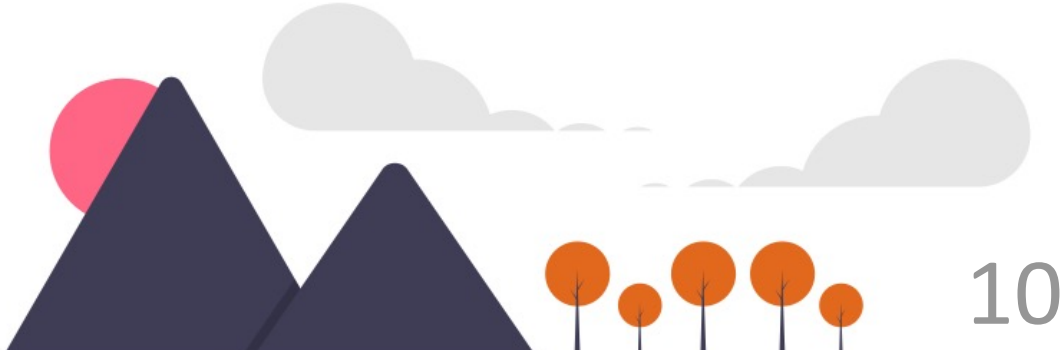
https://blog.crisp.se/2018/06/03/mathiasholmgren/bucket-estimation-how-to-estimate-a-really-large-backlog

Dealing with Uncertainty: Spikes

Spikes should be the exeption, not the rule. Reserve for critical and large unknowns.

Team members lack required knowledge

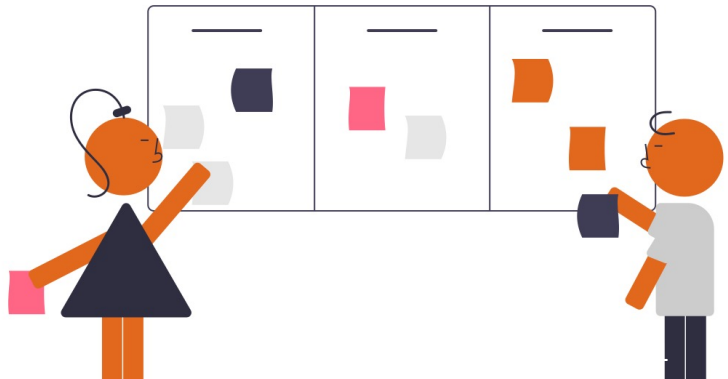
- Hard to estimate with little knowledge or experience
- Take time out of the sprint to research and learn: **Spike**
  - A special kind of user story: produces information, not code
  - E.g. evaluate best persistence strategy, implementation is separate story
  - Maximum time-box size of one iteration
  - Results should be **shared and discussed**
- Try to run spike in separate iteration from the resulting stories



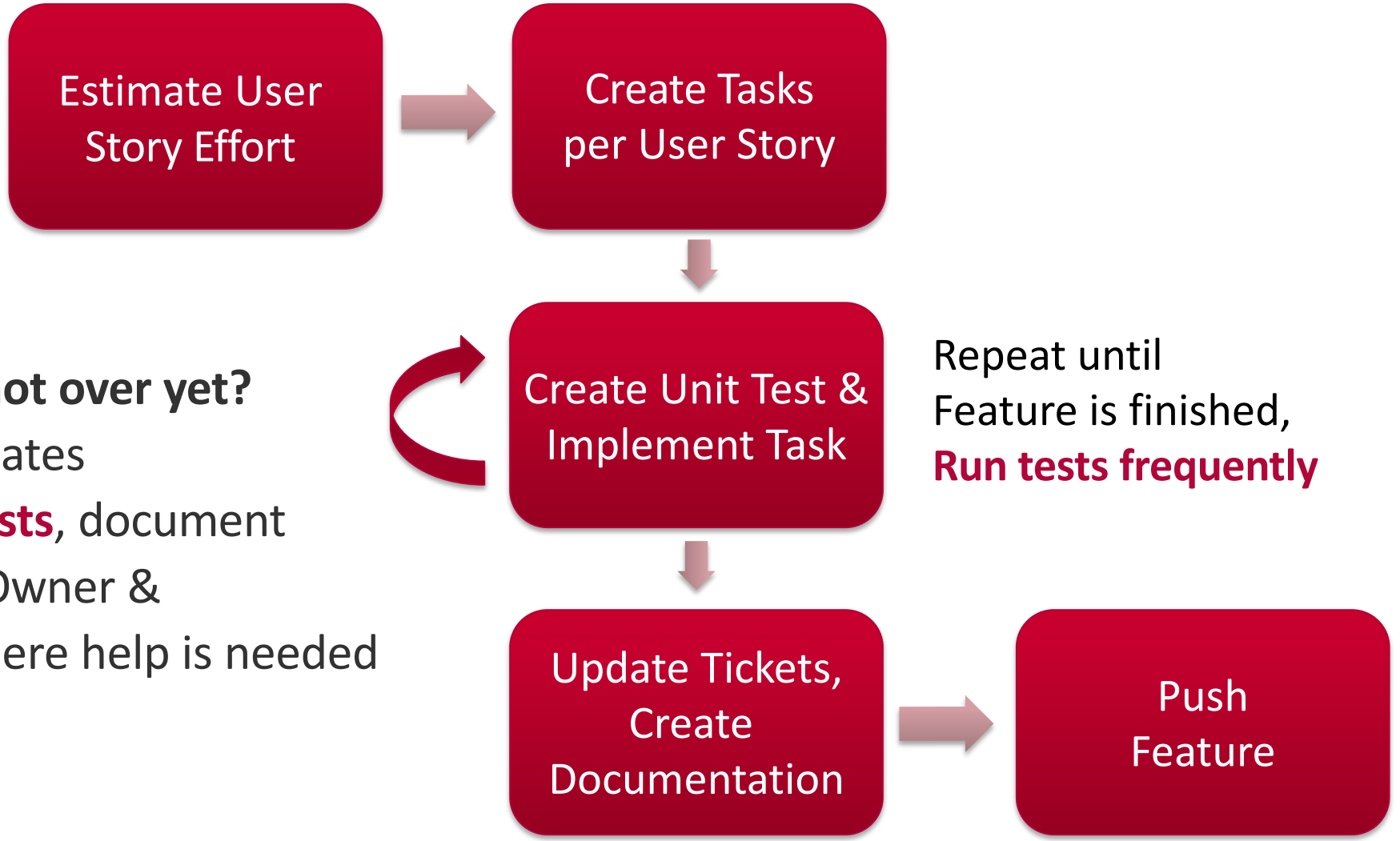
After the Planning Meeting

Begin the sprint

- Break down stories into tasks and fill your **Scrum Board**
  - Keep acceptance criteria in mind
  - Keep Definition of Done in mind
- Developers assign stories to themselves
- **Implement** the stories task by task
  - Communicate what you are working on
  - e.g. Draft Pull Requests



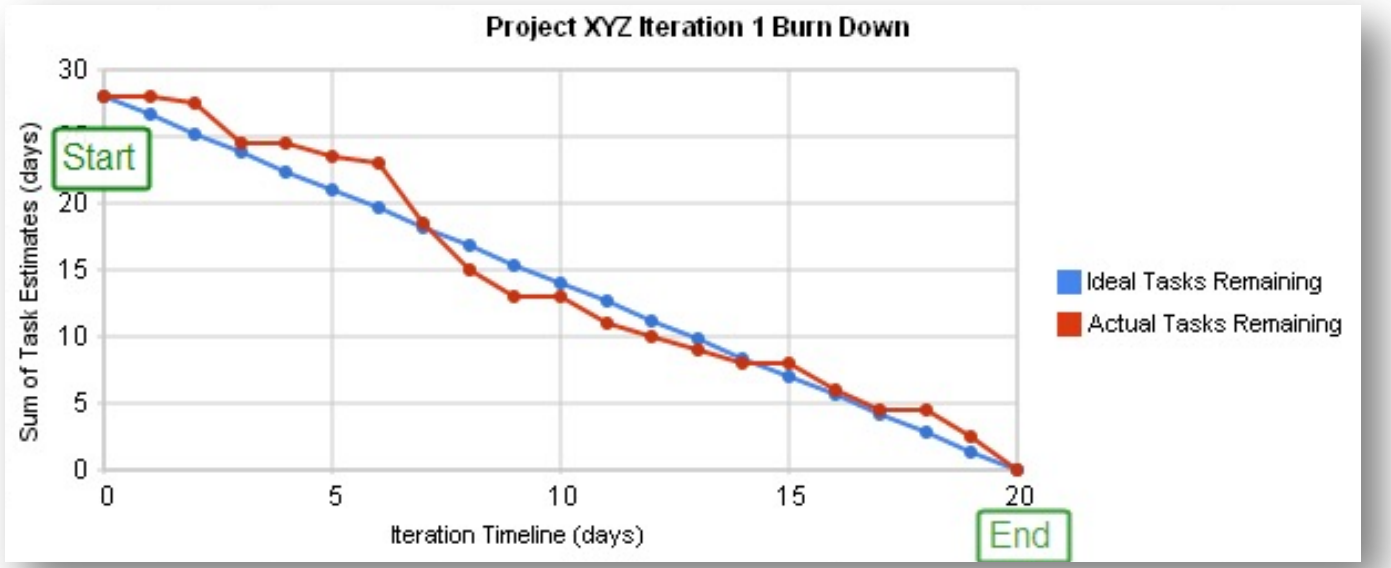
Project Workflow: Developers



Done and Sprint is not over yet?

- **Help** your teammates
- Refactor, **write tests**, document
- Ask the Product Owner & Scrum Master where help is needed

Scrum Burn-Down Chart



- Graphical representation of **work left to do vs time**
- X-Axis: sprint timeline, e.g. 10 days
- Y-Axis: work that needs to be completed in sprint (e.g. time or story points)
- "Ideal" work remaining line: straight line from start to end
- Actual work remaining line
  - Above ideal: behind schedule, below ideal: ahead schedule

Definition of Done

Defining when a User Story is finished

- Acceptance criteria fulfilled
- All related tests are green
- Code meets agreed quality standards
- Code was reviewed (by whom?)
- Implementation meets non-functional requirements
  - Internationalization
  - Security, legal
  - Documentation

The Definition of Done is the team's consensus of what it takes to complete a feature.

Definition of Ready

When is a user story ready for implementation?

- Similar to Definition of Done, but for user stories

Examples

- Estimated
- Acceptance criteria
- Mockups for UI stories

Beyond Scrum

Scrum critique:

- Scrum and agile are **by no means universally accepted** as "the way" to do software engineering (“Agile Hangover”)
- Michael O. Church - *Why “Agile” and especially Scrum are terrible (2015)*
  https://michaelochurch.wordpress.com/2015/06/06/why-agile-and-especially-scrum-are-terrible/
  - *Business-driven engineering*
    Scrum increases the feedback frequency while giving engineers no real power
  - *Terminal juniority*
    Architecture and R&D and product development aren’t part of the programmer’s job
  - *It’s stupidly, dangerously short-term*
    engineers rewarded solely based on completion of current sprint

Beyond Scrum

David Heinemeier Hansson created Ruby on Rails

Scrum critique:

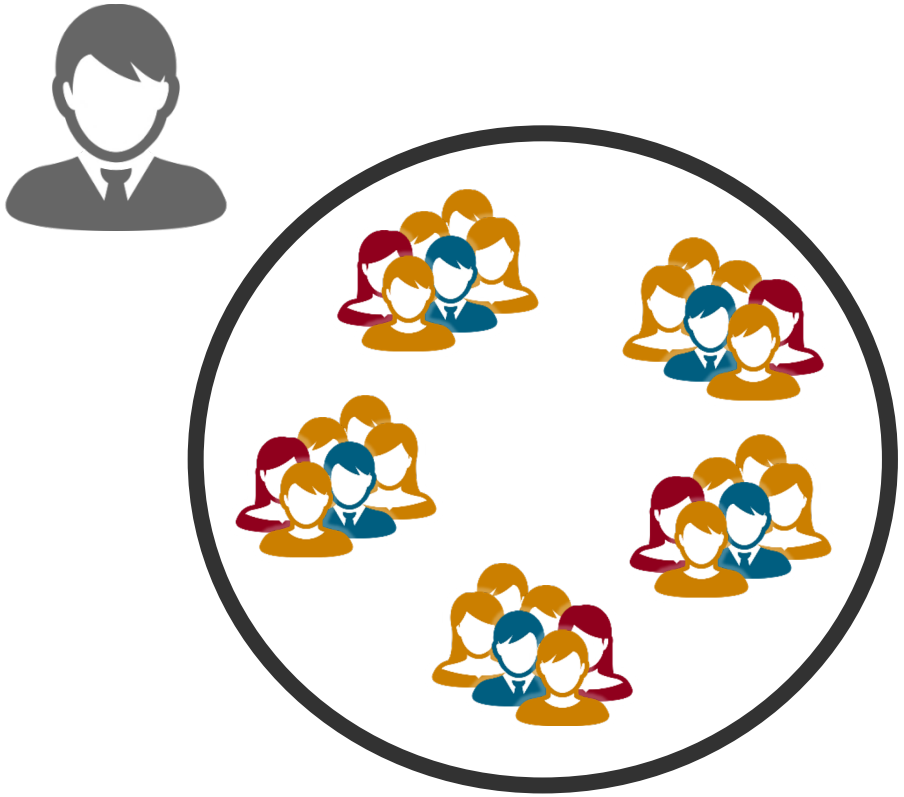
- Building Software with David Heinemeier Hansson
  https://medium.com/computers-are-hard/computers-are-hard-building-software-with-david-heinemeier-hansson-c9025cdf225e
  - *"estimation is bullshit. It's so imprecise as to be useless"*
  - *"No one is ever able to accurately describe what […] software should do before they see the piece of software."*
  - *"Agile was sort of onto this idea that you need running software to get feedback but the modern implementations of Agile are not embracing the lesson they themselves taught."*

Practical Ideas for Scaled Scrum

Scalable Software Engineering
WS 2021/22

Enterprise Platform and Integration Concepts

Recap: Project Structure



What's needed in such an environment?

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

Many scaled frameworks exist:
LeSS, SAFe, Nexus,
Scrum@Scale

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 (split initial team)

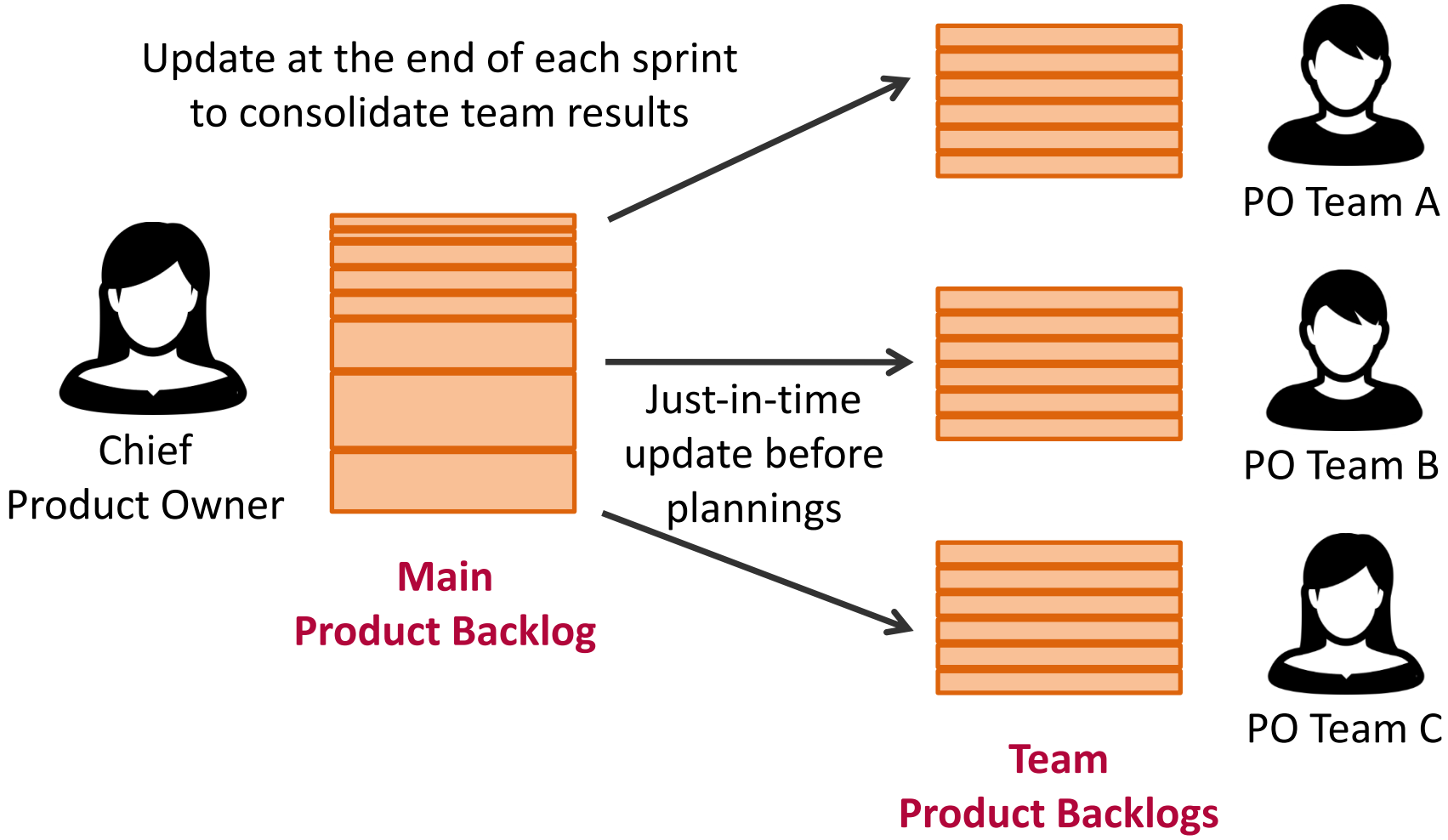
Internship Model

- Add people to high performance team, preserve team, then new people start their own teams
- Share knowledge, share culture across teams when growing number of teams

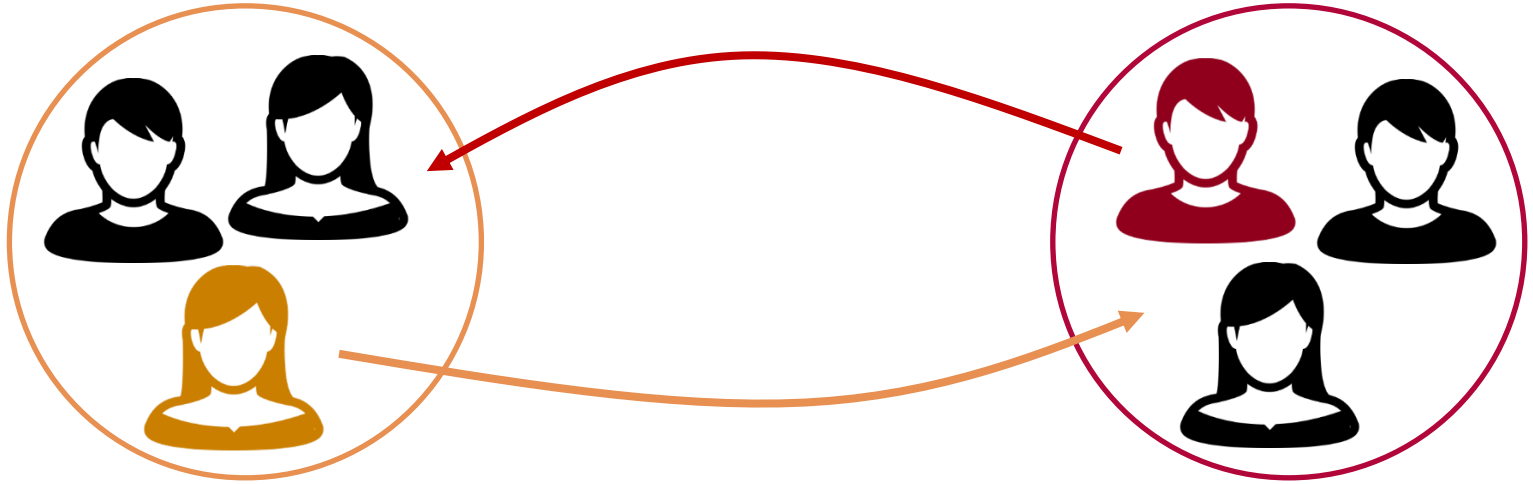
This course has already reached a scaling point: multiple collaborating teams

https://www.scrum.org/scaled-professional-scrum-nexus-practices

Product Backlog Hierarchy



Dealing with Dependencies: Ambassadors



Mutual Exchange of team members

- Improve **efficiency of communications**
- Allow deeper understanding of (other teams') problems
- Prevents coordination problems early
  - Ambassadors should be fully integrated team members
  - Especially useful for API development, design, etc.

[Pichler, Scrum – Agiles Projektmanagement erfolgreich einsetzen, 2007]

Scaled Sprint Planning

Preparation

- Individual review and retrospection meetings
- Sprint Planning 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)

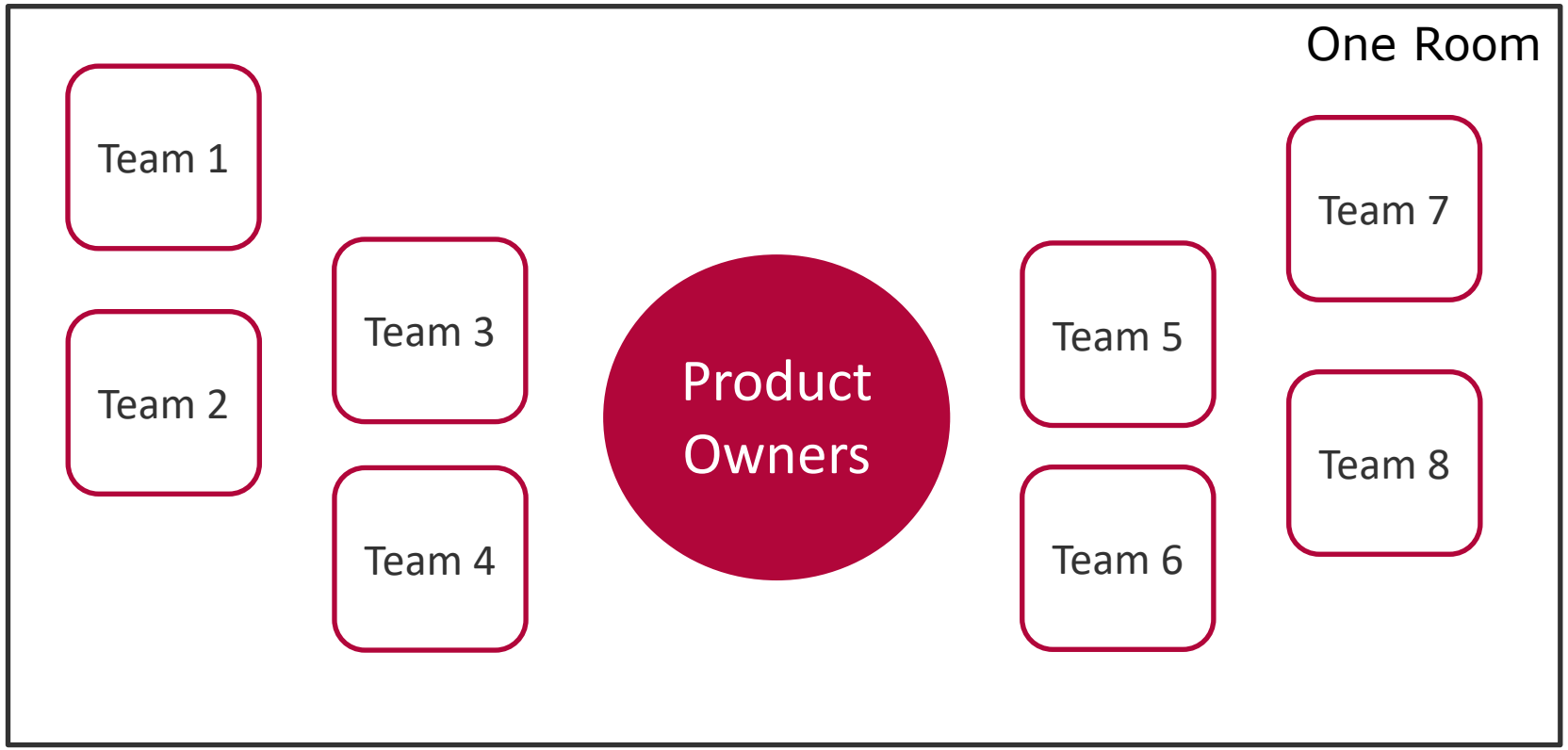
Execution

- Individual Plannings in teams
- Discussion of identified additional input or output dependencies
- Final Sprint Planning

Problem: Time consuming & high degree of coordination needed!

Scaled Sprint Planning

Co-located planning



Scrum of Scrums

Synchronize team efforts

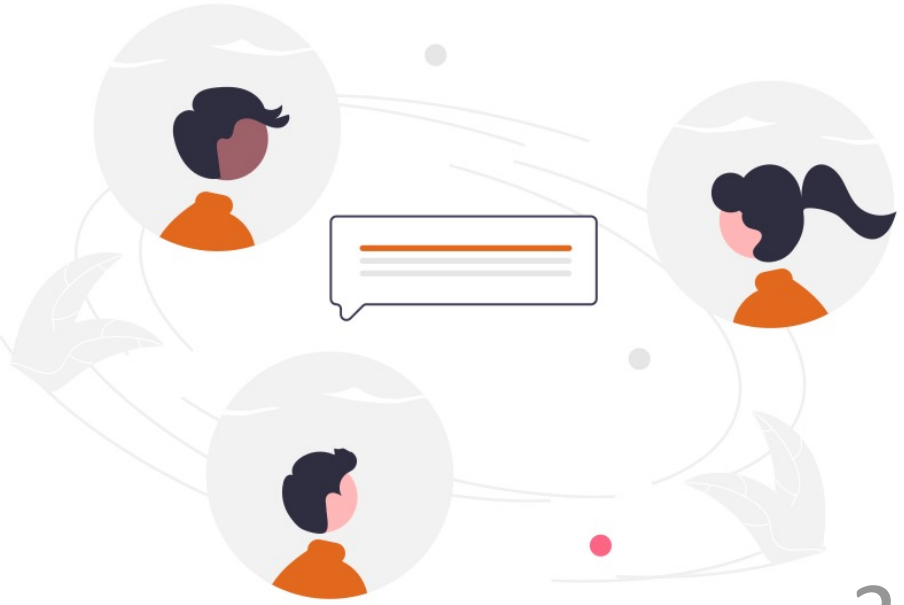
- Regular meeting of Scrum Masters/process interested
  - Developers may join if necessary (**ambassador principle**)
- Participants
  - Share their learnings
  - Report completions & next steps
  - Coordinate inter-team dependencies
  - Negotiate responsibility
- Developers discuss technical interfaces across teams
- Distribute information back into the teams
- (work toward fully integrated set of potentially shippable increment)

https://www.atlassian.com/agile/scrum/scrum-of-scrums

Communities of Practice

Organize around topics

- Form and support groups of **people with common interests**
  - From different teams, from different backgrounds
- Share learnings, discuss details, improve skills
  - **Tackle current (shared) issues in teams**
  - Explore new ideas
- E.g. people interested in UI design, software architecture, security, user research



https://www.scrum.org/scaled-professional-scrum-nexus-practices

Summary

Effort estimation

- Planning Poker
- Affinity Estimation
- Bucket Estimation

Scrum Concepts

- Spikes
- Developer workflow
- Burn-Down Chart
- Definition of Done
- Definition of Ready
- Scrum critique

Scaling Scrum

- Backlog Hierarchy
- Ambassadors
- Scaled Sprint Planning
- Scrum of Scrums
- Communities of Practice