Scrum Deep Dive

Software Engineering II
WS 2020/21

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

Waterfall 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 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! I‘ve a straight flush!“ by Joel Bez (CC BY 2.0) via flickr

Planning Poker

Process

- Product Owner explains backlog item and the business value
- 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**
- Arguments are **discussed** in the group

Planning Poker

- A new vote is conducted
- **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 sized or **time is over**

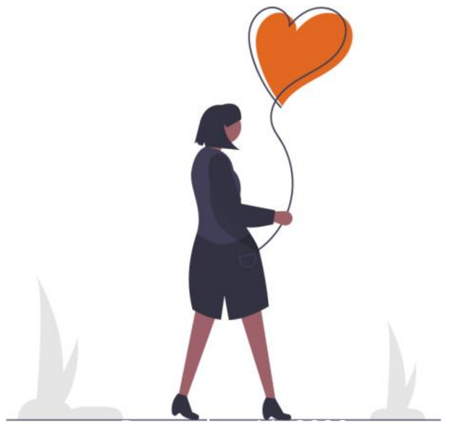
Affinity Estimation

Participants

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

Preconditions

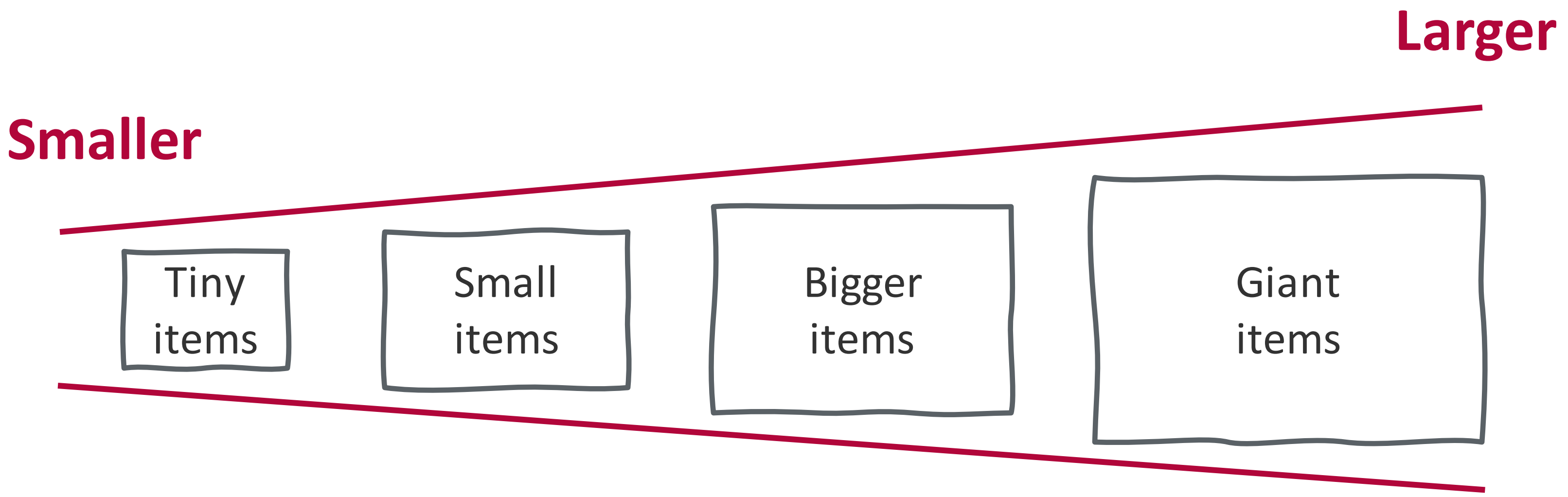
- Product backlog is complete, **prioritized** and understood
- A shared space to work in
- User Stories that can be moved around (post-it notes, printed, in shared workspace)



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/

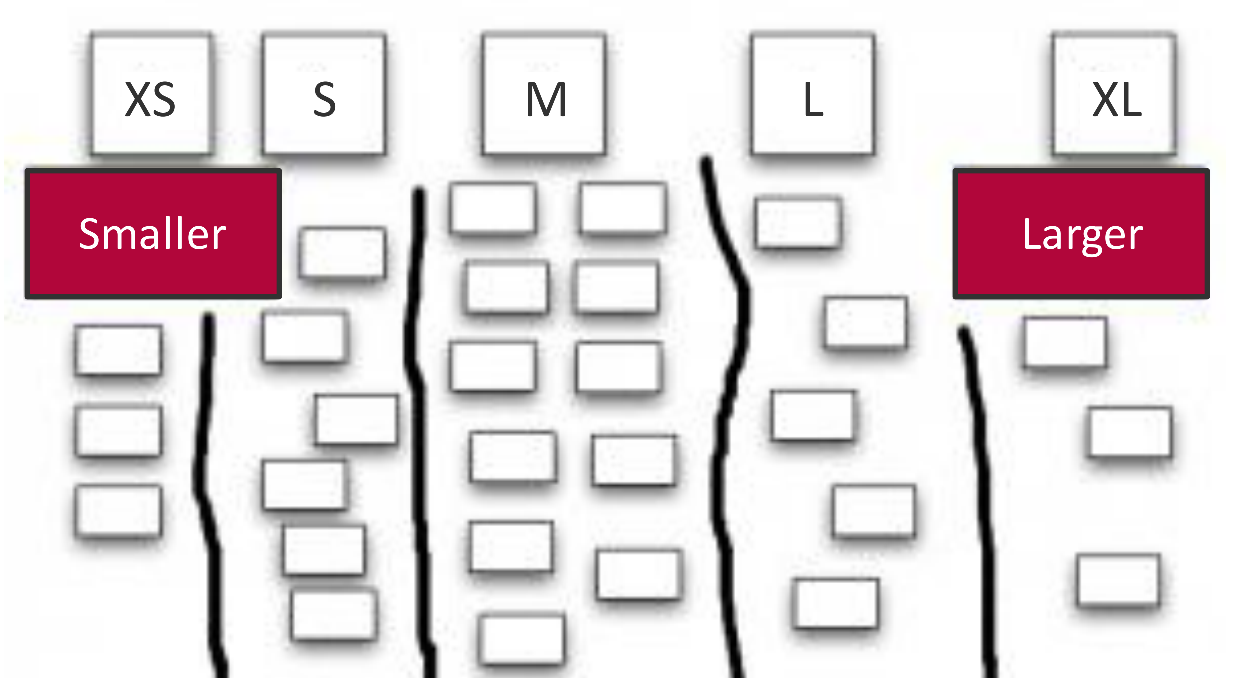
Affinity Estimation

Step 2: Editing

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

Step 3: Place stories into categories

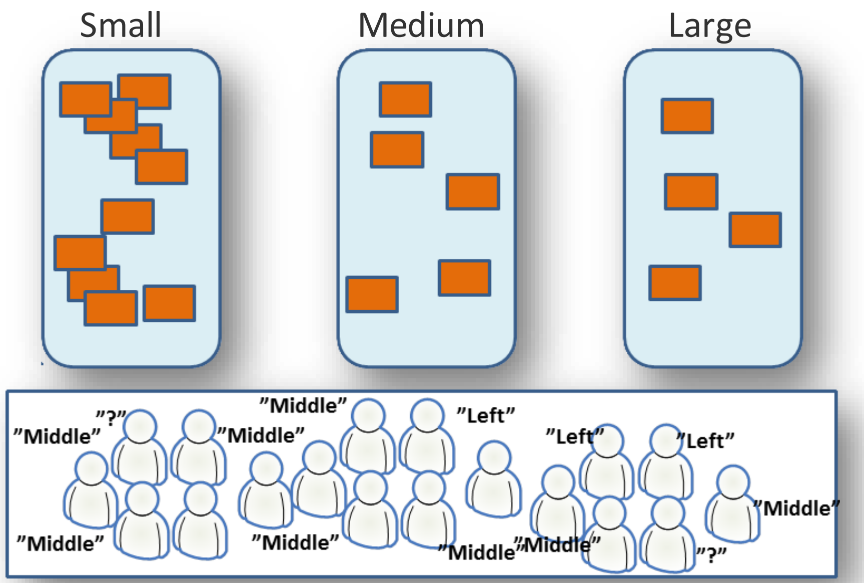
- Place size categories (e.g. Fibonacci sequence) above scale
- Assign each story a size based on location



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 an average item
  - Max. 2-3 items per bucket
  - Break up into smaller diverse groups
  - Estimate using a fitting approach



Dealing with Uncertainty
Spikes

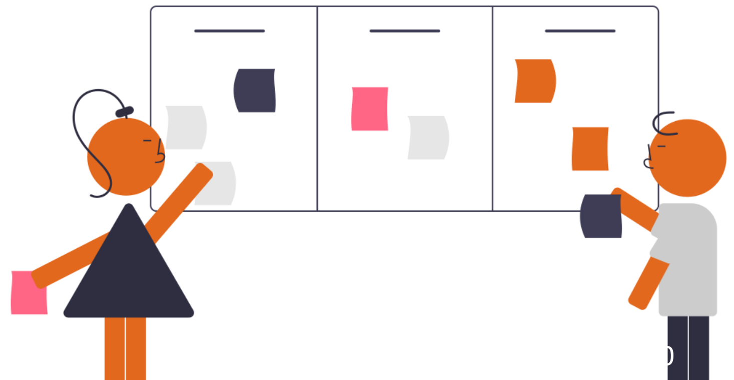
What can we do if no team members lack knowledge in a particular domain?

- Hard to estimate with little knowledge
- Take time out of the sprint to research and learn
- Spike
- For example, evaluate new technologies

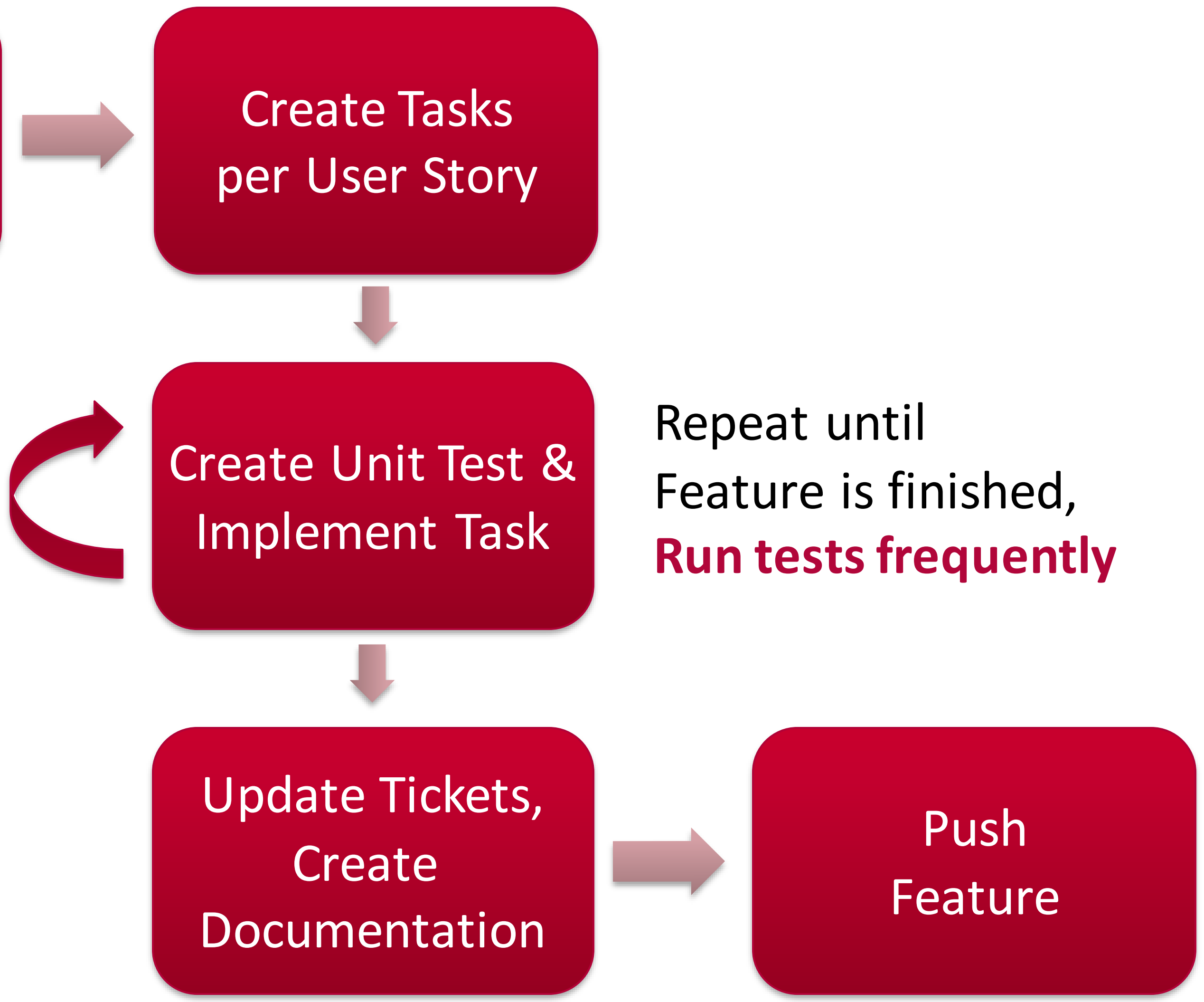
After the Planning Meeting

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

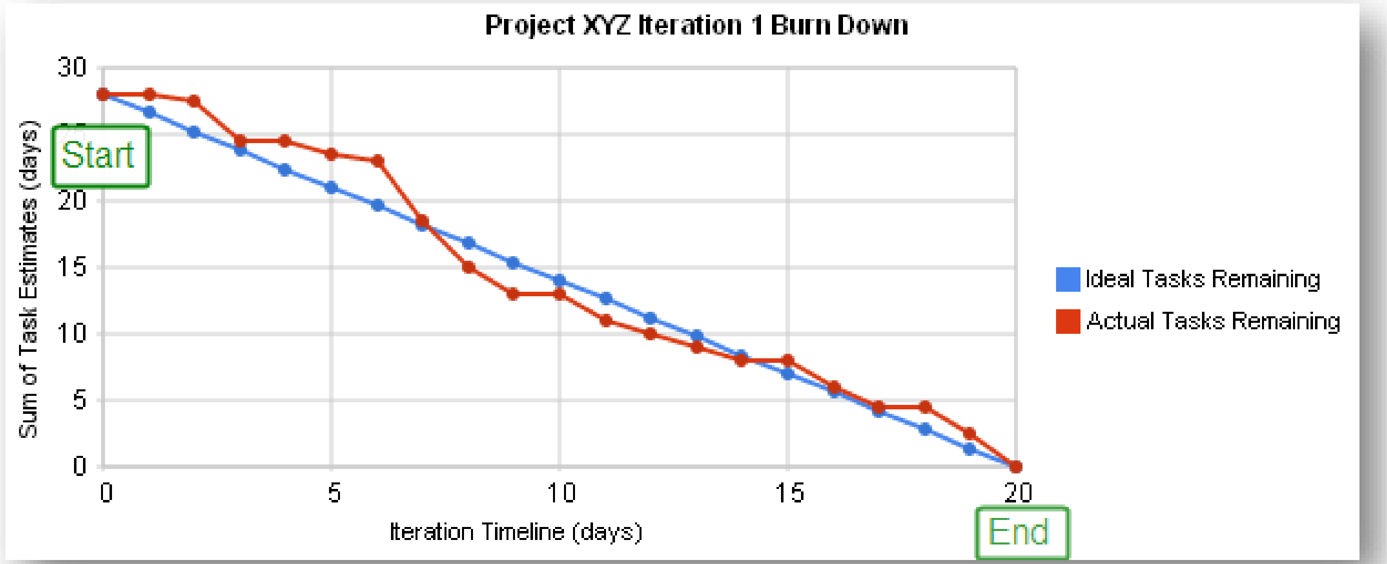

Estimate User Story Effort
Create Tasks per User Story
Create Unit Test & Implement Task
Update Tickets, Create Documentation
Push Feature

Repeat until Feature is finished, **Run tests frequently**

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 (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

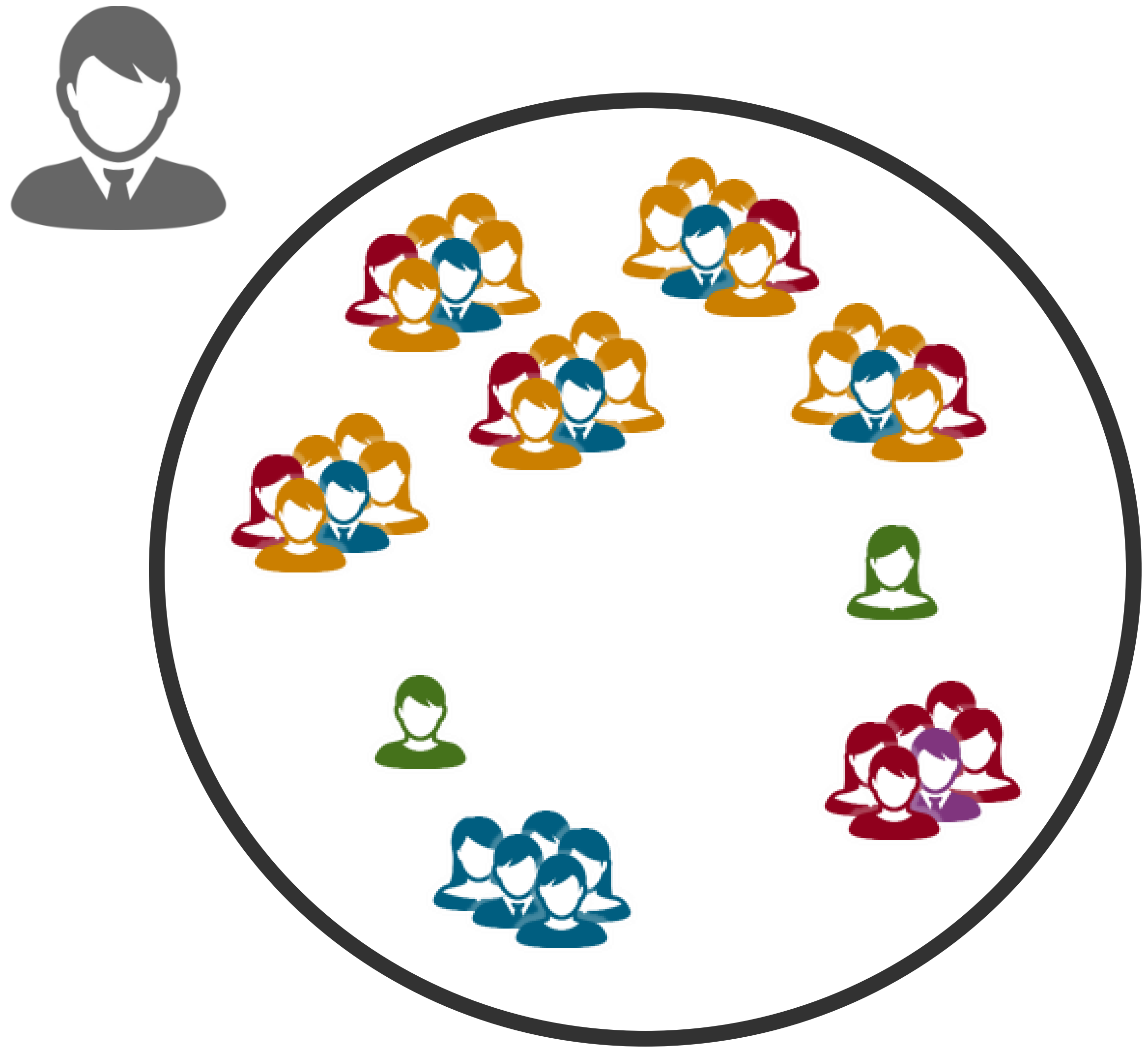
Scrum critique:

- 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."*

David Heinemeier Hansson created Ruby on Rails

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

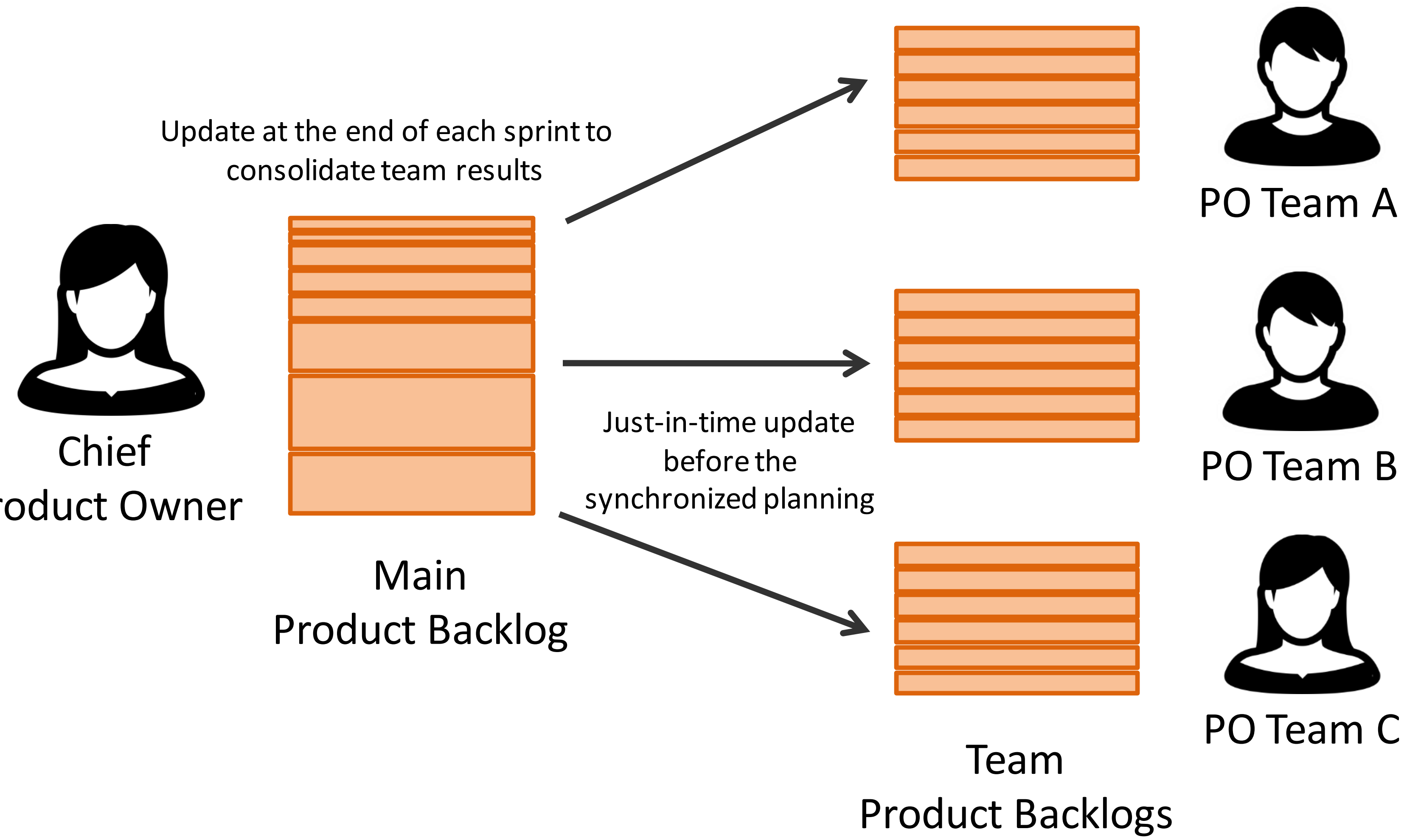
Start small and grow organically

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

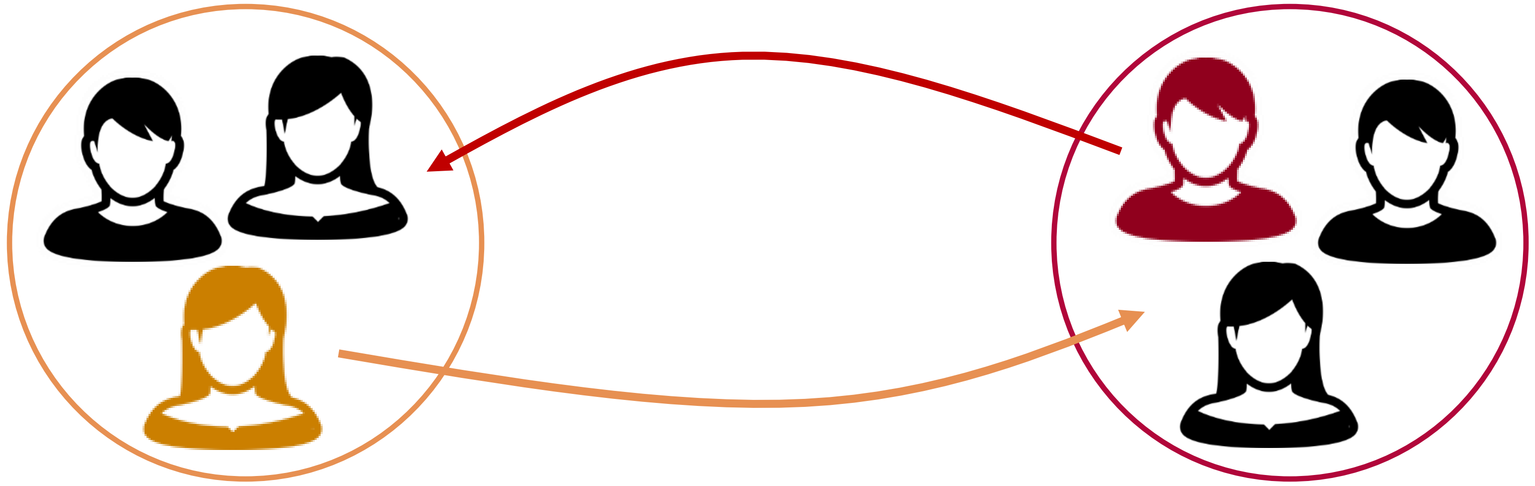
SWTII is already at a scaling point

- Multiple collaborating teams

Product Owner / Backlog Hierarchy


Update at the end of each sprint to consolidate team results
Chief Product Owner
Main Product Backlog
Just-in-time update before the synchronized planning
Team Product Backlogs
PO Team A
PO Team B
PO Team C

Dealing with Dependencies & Scale
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]

Scaling Scrum: 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)

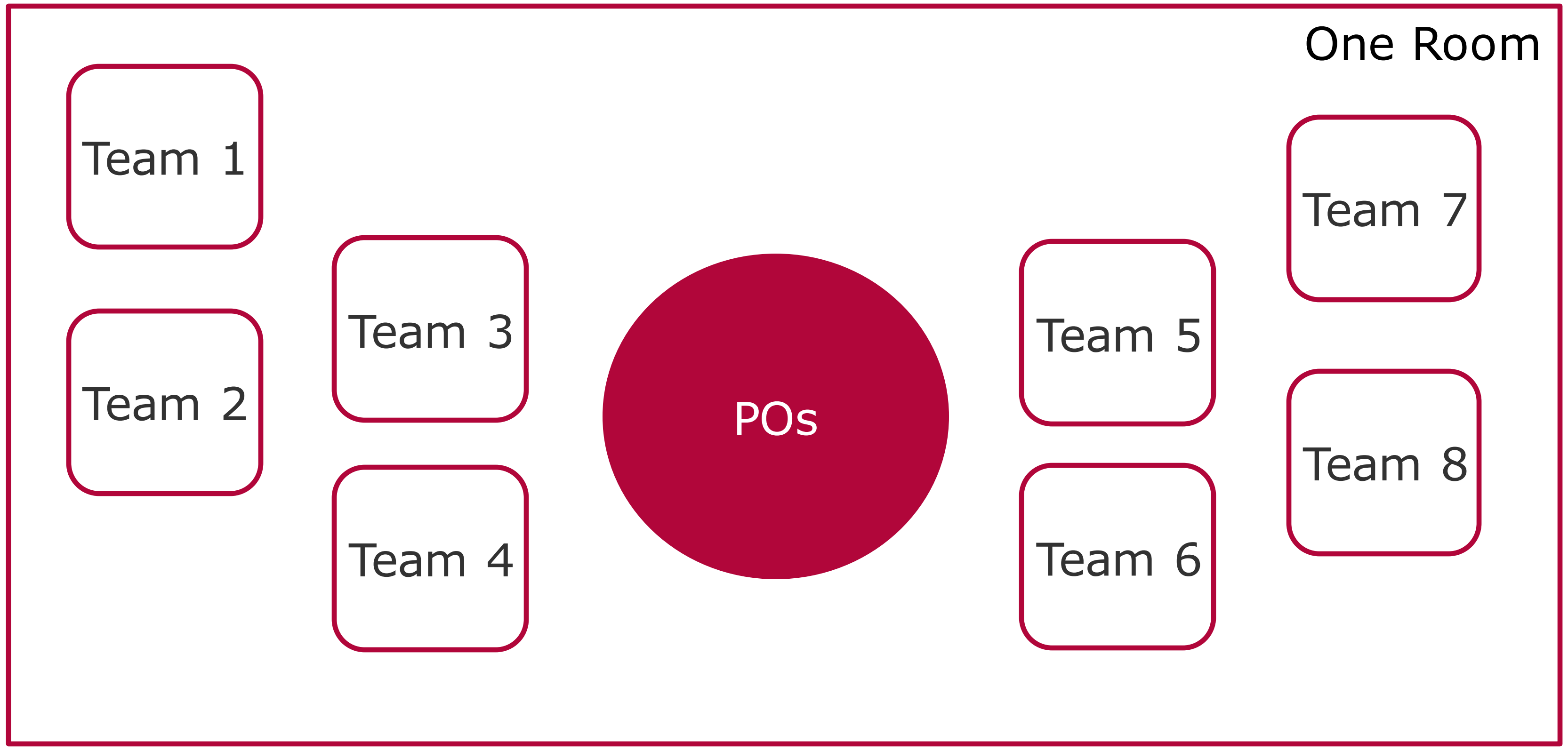
Execution

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

Problem: Time consuming & high degree of coordination needed!

Scaling Scrum: Sprint Planning

Another Option: Co-located planning



Scrum of Scrums

Goal: Synchronize team effort with minimal coordination overhead

- Regular meeting of Scrum Masters / process interested
  - Developers join if necessary (**ambassador principle**)
- Scrum Masters or those interested
  - 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

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