Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Agile Recap

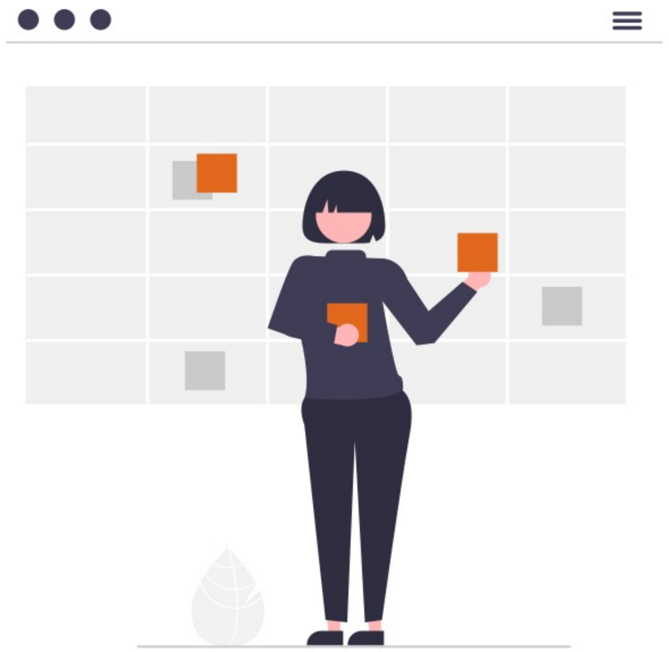
Scalable Software Engineering
WS 2020/21

Enterprise Platform and Integration Concepts

The Case for Agile

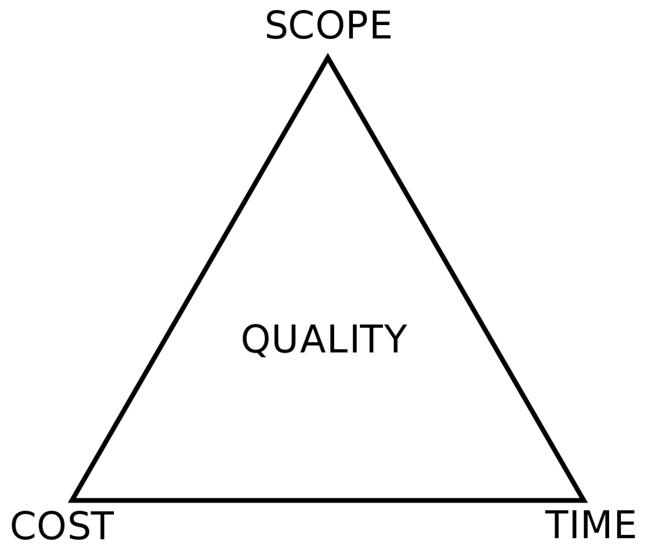
Agile software development methods

- A new/different process *(compared to what?)*
- Defining “agility”
- Advantages and drawbacks



How (Software) Projects Fail

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

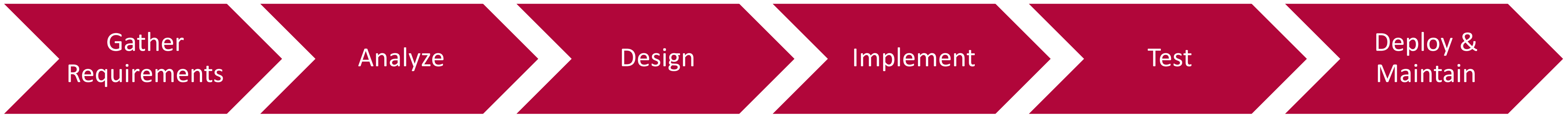


Source: https://en.wikipedia.org/wiki/Project_management_triangle

Why Projects Fail

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

Traditional project process



- Involves efforts regarding
  - Detailed hand-overs from phases
  - Documents specifying work done & to be done
  - Review committees

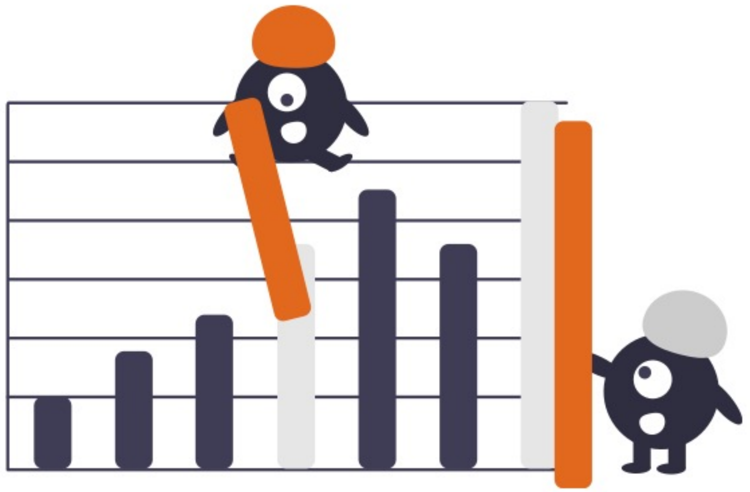
Why Projects Fail

“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:

- Adjustments & assumptions are made during all phases
- **Re-planning must take place**
- Example: Testing phase at the end
  - Tester raises a defect
  - Programmer claims he followed the specification
  - Architect claims he followed business analyst etc.
  - Exponential cost



A Self-Fulfilling Prophecy

- Front-loaded process to minimize late and expensive changes
  - Project plan, requirements specification, high- (& low-)level design documents
  - **Specify everything, then execute**
- This in itself can cause high costs of change

*Highly front-loaded processes make sense for well-understood, well-planned projects, like bridges, ships, or a building but **software is easier to change****

*Exceptions: Software projects that cannot easily be changed, need to follow strict regulations, or where failures can lead to real life disasters, e.g., embedded (offline) medical devices, infrastructure software for power plants, very high security standards in the military domain. But even in *cleanroom development*, iterations are part of the process.

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/

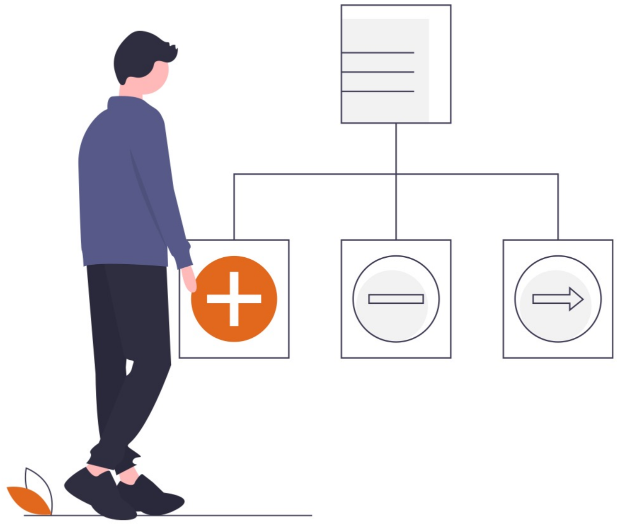
Addressing Project Risks w/ Agile

Budget and Time Constraints

- Plan in **smaller development iterations**
- Budget for iterations, check
- High-priority items first, re-evaluate

Meeting Customer demands

- **Actively involve stakeholders**
- Short feedback cycles
- Reflect demands in prioritizations



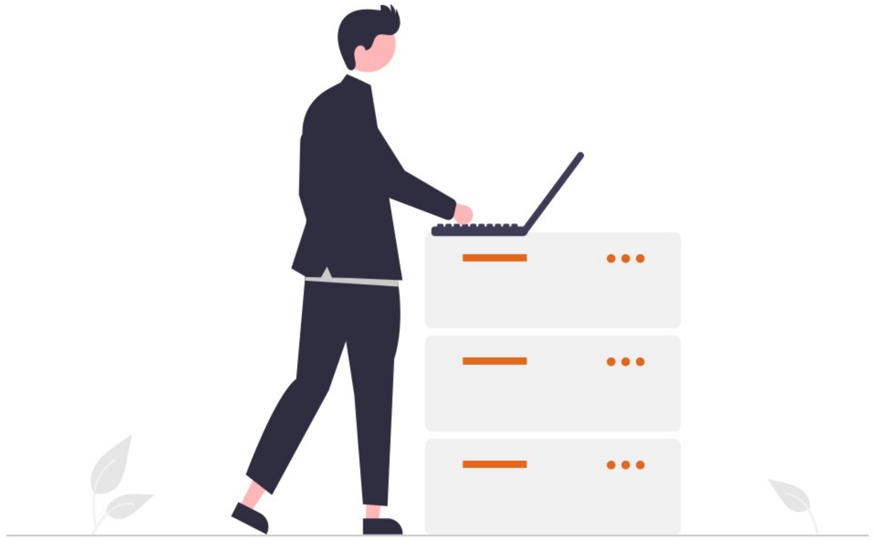
Addressing Project Risks w/ Agile

Production Stability

- Deliver each iteration, identify production issues early
- Suggests **high degree of automation**

Maintenance Costs

- **Maintenance starts immediately**, after first delivery
- Optimize processes iteratively



The Cost of Going Agile

Iterative planning

- **Lack of** complete, detailed **project plan**
- Less knowledge of future deliverables

Streaming requirements

- A new requirements process
- **Continuous refinement** & prioritization of requirements

Evolving design

- No complete upfront architecture: flexibility required
- **Emergent Design**, that may not be completely known at the start

Adapting existing structures

- Need for **refactoring existing code**
- Adapt architecture decisions

The Cost of Going Agile

Frequent code merges

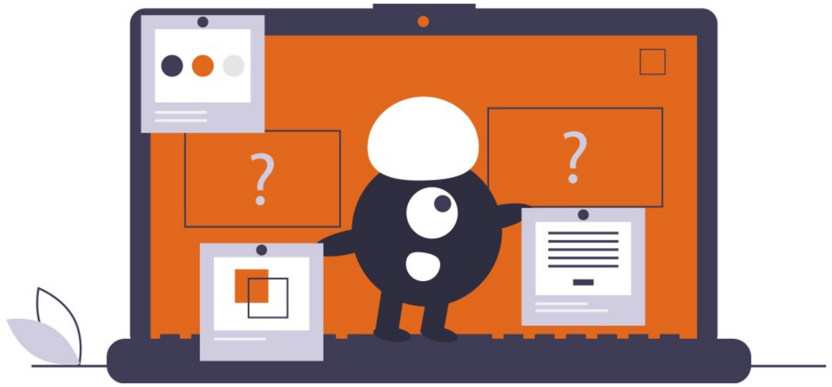
- **Time investment** and frequent source of errors
- Continuous integration

Continuous testing

- Add nth feature; test n-1 features
- Tests are code. Tests need to be **updated & maintained**

Frequent (production) releases

- **Organizational challenges**
- User training, updated user documentation

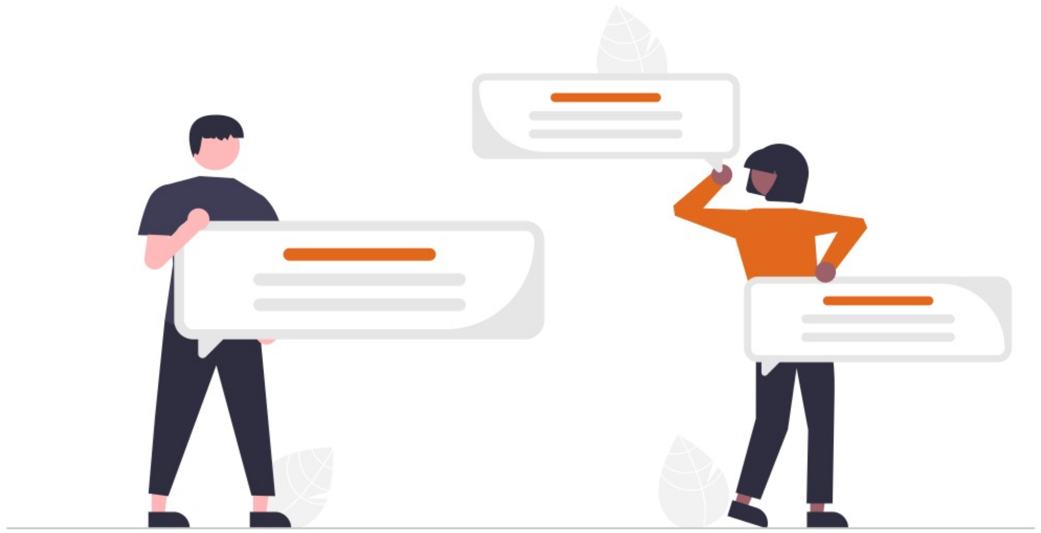


Discussion

Advantages and drawbacks

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

Almost feels like something you‘d ask in an exam?!



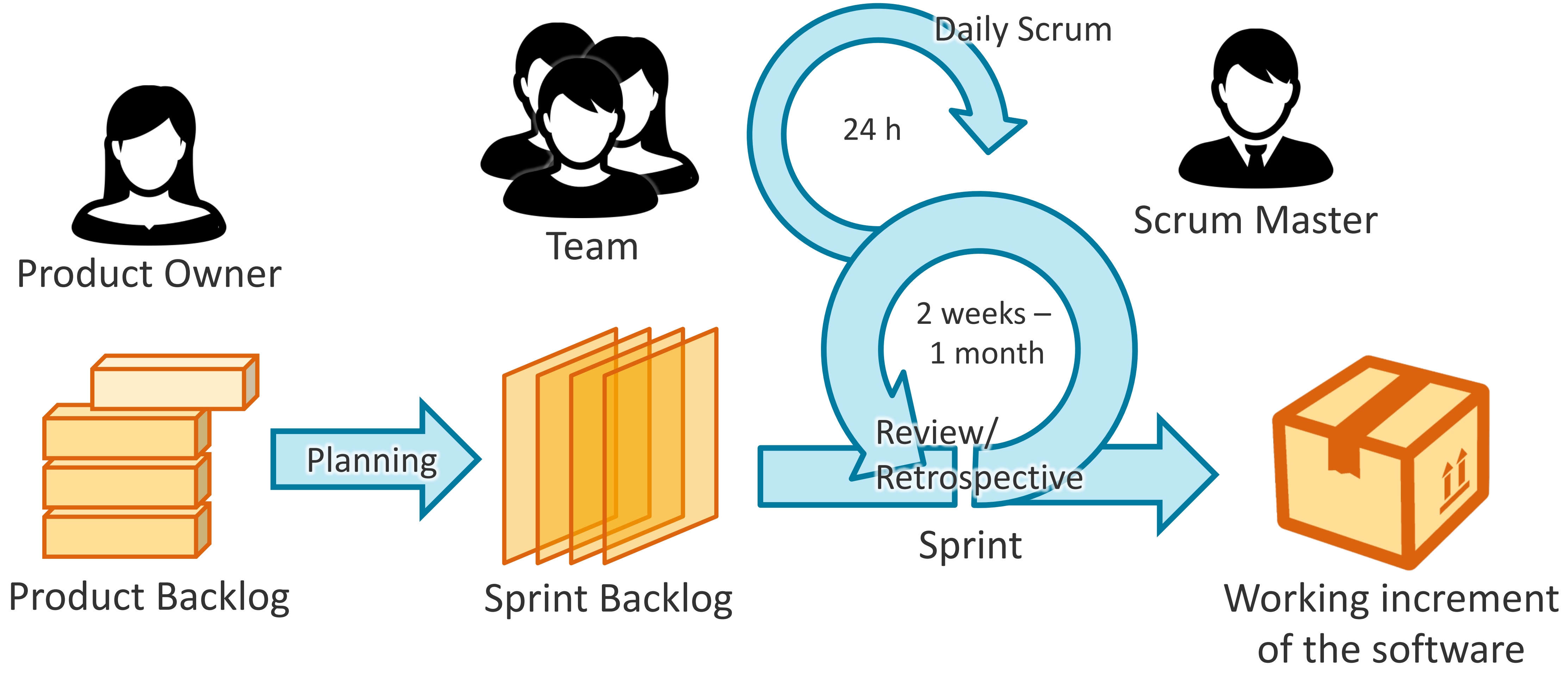
Scrum Recap

Scalable Software Engineering
WS 2021/22

Enterprise Platform and Integration Concepts

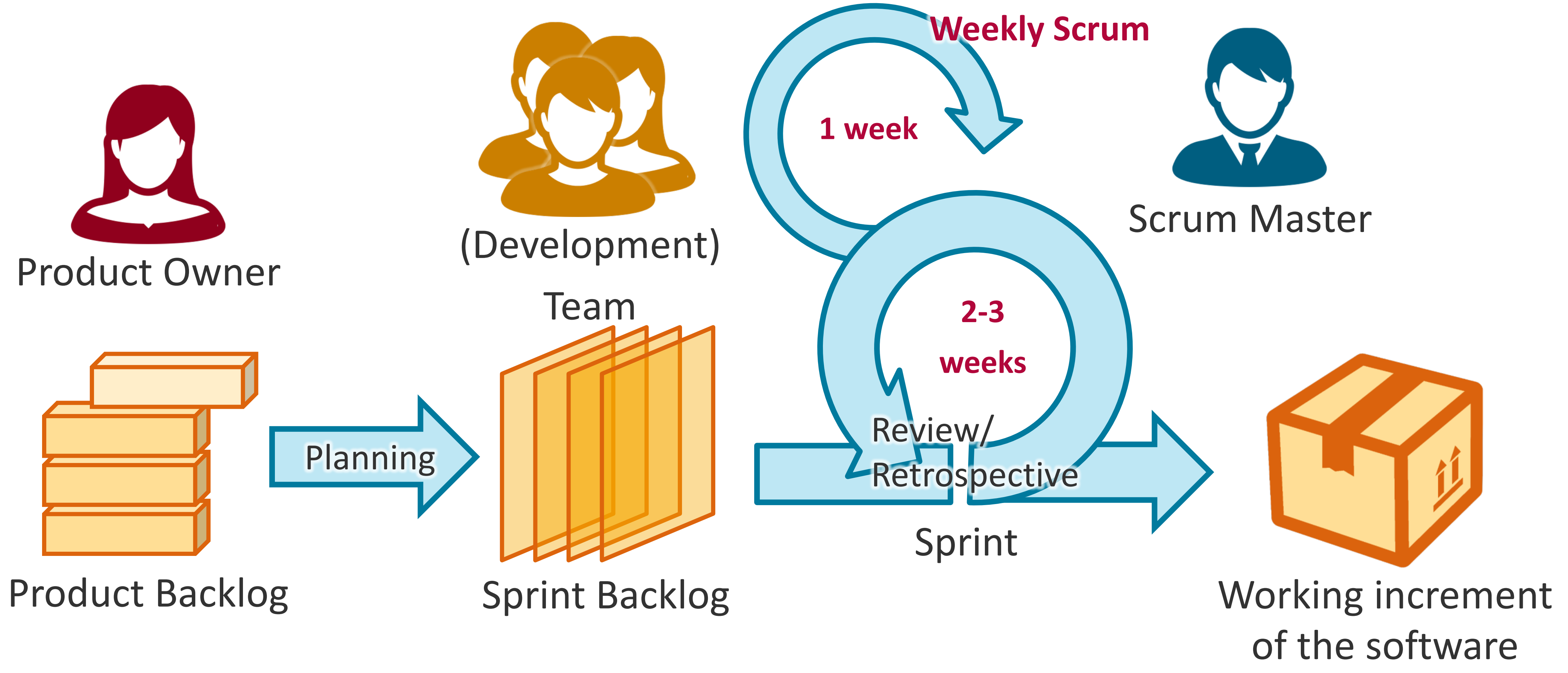
Image by *Four Bricks Tall* on flickr: https://flickr.com/photos/fourbrickstall/48716532018/ (CC BY-NC-ND 2.0)

Scrum



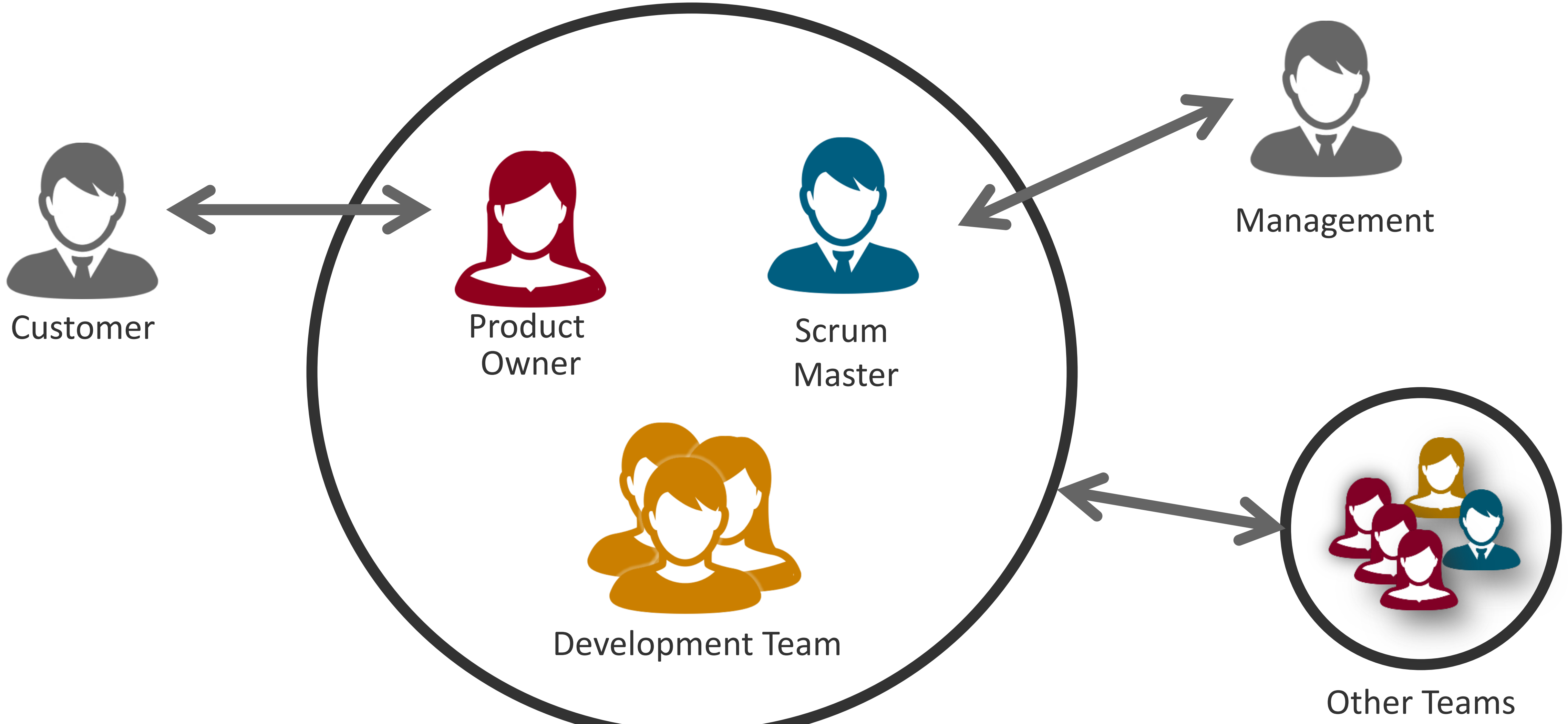
“Scrum process” by Lakeworks. (CC BY SA 3.0) via Wikimedia Commons

Our Scrum in the SSE Course



“Scrum process” by Lakeworks. (CC BY SA 3.0) via Wikimedia Commons

A Team in our SSE Course

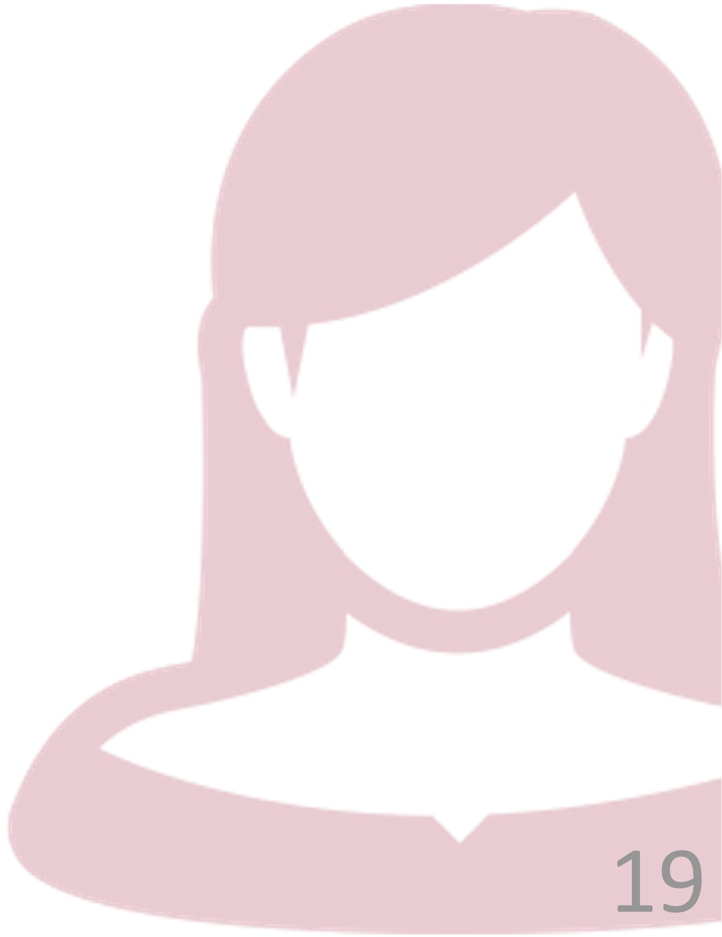


Product Owner

Responsibilities

- Acts as a **Stakeholder Proxy** for the team
  - Knowledge of **business value** of requirements
- **Customer** communication
  - Contact for team members on domain information
  - Customer testing
- Managing the Product Backlog
  - Maintaining **User Stories**
  - Knowledge on **work priorities**
  - Acceptance criteria & acceptance tests

http://agilemodeling.com/essays/productOwner.htm



Scrum Master

Responsibilities

- **Coaches** to the rest of the team
  - (Delegates) moderation in meetings
  - Enable rest of the team to work effectively
  - Provide transparency within (and outside) the team
- **Process focus** (vs. product direction of PO)
  - Focus on the "how" of getting work done
  - Work on workflow problems, remove **impediments**

https://www.atlassian.com/agile/scrum/scrum-master

Development Team

Responsibilities

- Communication
  - **Critically** discuss inputs and provide feedback
  - Discover & share (technical) information
- Sprint Backlog
  - Evaluate and refine requirements
  - Initiate required work items (e.g. bugs, refactorings)
  - Highlight (technical) requirement dependencies
- Strong **focus of software development**

Other Possible Project Roles

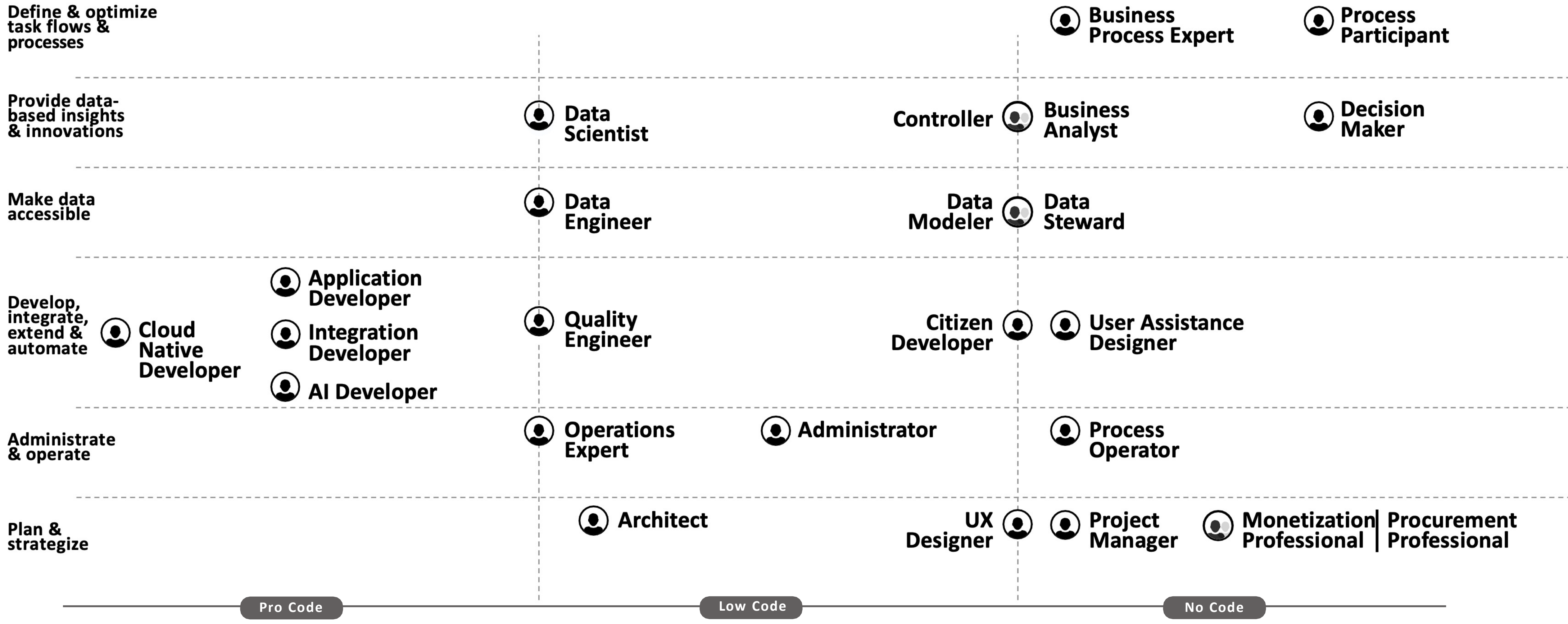
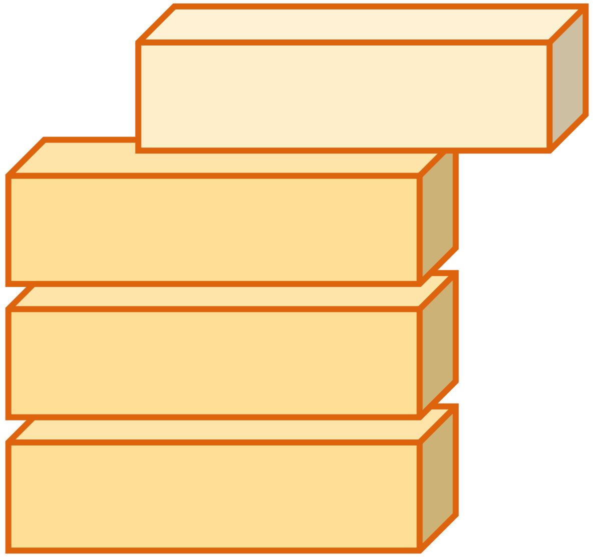


Image courtesy of SAP

Product Backlog

List of work items

- Requirements (modification **requests**)
  - Features
  - Extensions
  - Bug fixes
  - Ideas that need to be tracked
- Ordered/**prioritized**
- Varying levels of polish



Requirements

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

"As <role>, I want <feature> to <reason>"

User Stories should fulfill **INVEST** properties:

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

Further reading:
http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

Requirements

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

"As <role>, I want <feature> to <reason>"

User Stories should fulfill **INVEST** properties:

- I – **Independent** (ability to schedule and implement in any order)
- N – **Negotiable** (captures the essence, not the fine details)
- V – **Valuable** (in terms of business value to stakeholders)
- E – **Estimatable** (enough info to rank the story's effort)
- S – **Small** (understandable scope)
- T – **Testable** (sufficient info so that you *could* write a test)

Further reading:
http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

Planning Meeting

Filling the Sprint

- Define a **Sprint Goal** (overarching theme that guides the iteration)
- Estimate Backlog items (can also happen before)
- Move items from Product to **Sprint Backlog**
- **Keep in mind the team’s capacity**
- Get the team’s commitment

Defining the specific work items

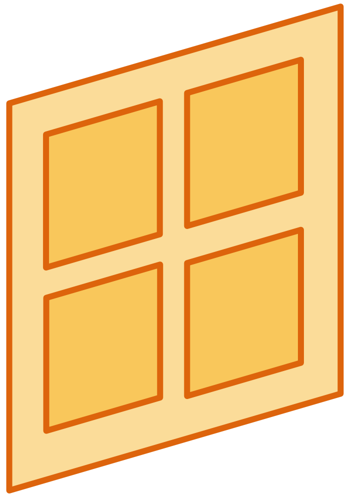
- **Break down** Backlog items into specific tasks
- PO not necessarily required

Tasks

For better planning, stories can be broken down into tasks

Tasks should be **SMART**:

- S – **Specific** (everyone understands what's involved)
- M – **Measurable** (clear what is required to mark as done)
- A – **Achievable** (task owner can be expected to fulfil task)
- R – **Relevant** (task contributes to story)
- T – **Time-boxed** (clear expectation when to seek help)



In XP stories have 3 parts: Cards (physical medium), Conversation (discussion around them) & Confirmation (tests that verify)

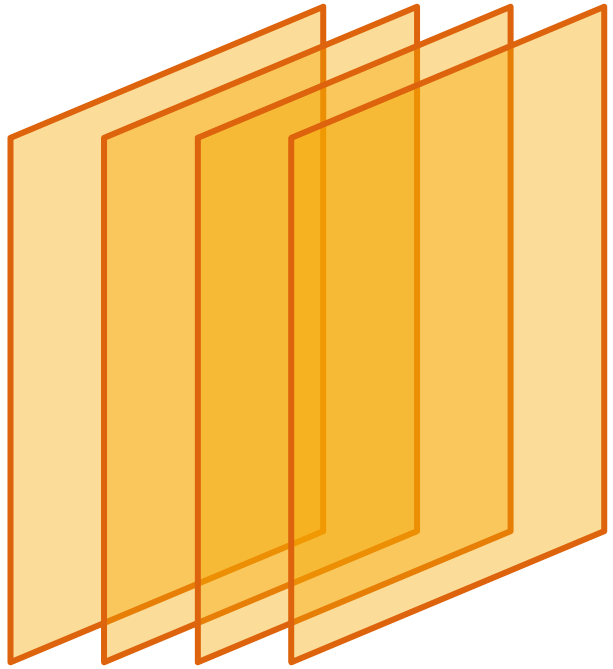
Further reading:
http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

Sprint Backlog

List of work items & tasks for a Sprint

- Work to be done to fulfill Sprint Goal
- Tasks should be **signed-up** for, not assigned

- During the sprint
  - Focus on building value
  - **No new features and stories**
  - Team may change/add tasks



Weekly Sprint Meeting

With full-time employees, this would occur daily

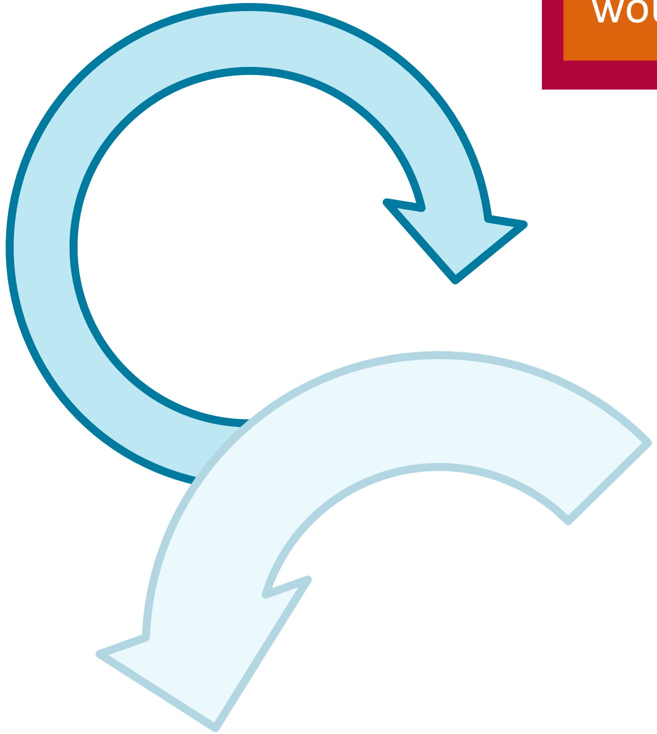
Status update & team sync

- Last achievements
- Next steps
- Problems

Keep it **as focused as possible**

Discussions?

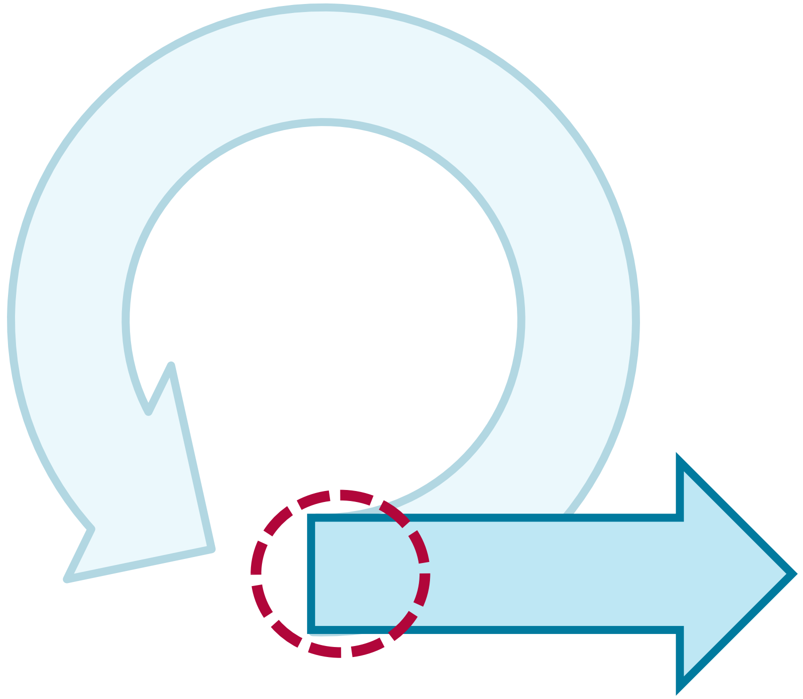
- Note follow-ups (possibly not everyone is required)
- Schedule **subsequent** expert meeting



Review Meeting

Review of iteration results

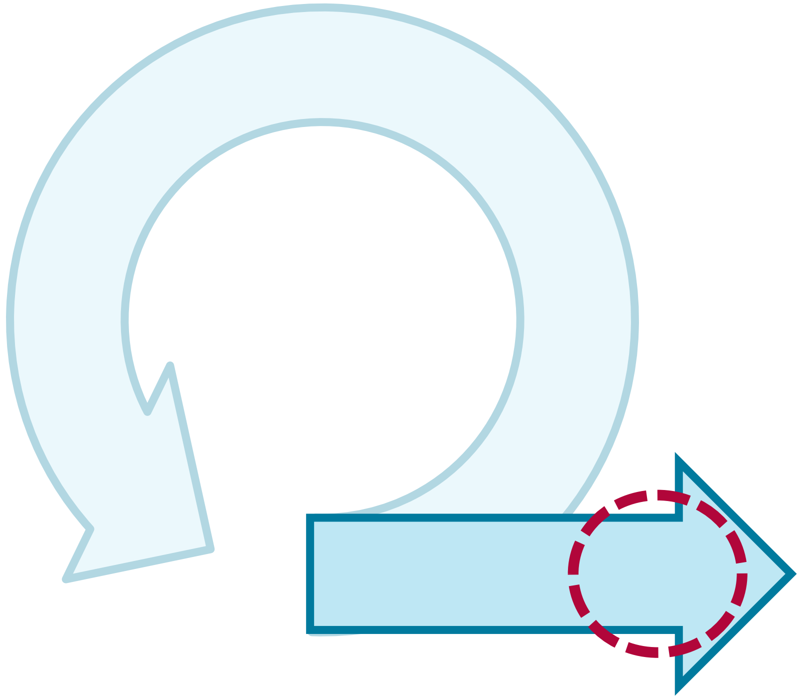
- **Celebration** of results
- **Demo** of accomplished functionality
  - Developers present their work
  - A prepared PO is able to assess (according to the User Story)
  - Optional: invite other stakeholders
- Was the **Sprint Goal** achieved?
- What potentially needs to go back into the Product Backlog?



Retrospective Meeting

Evaluation of the past iteration

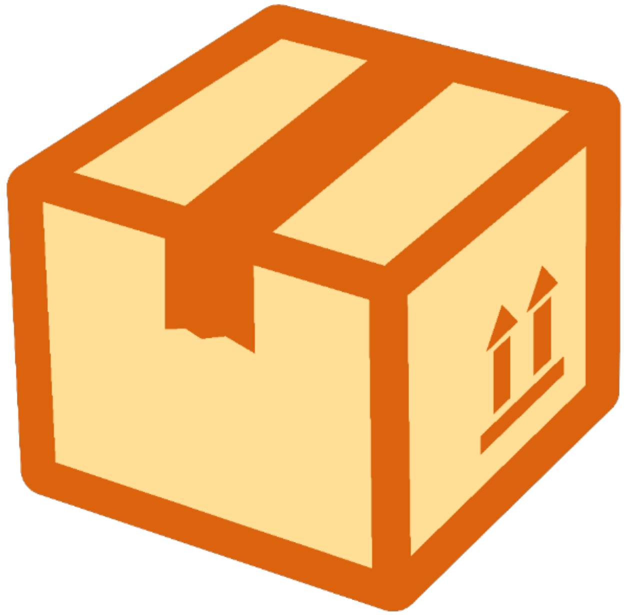
- Discuss process and improvements
  - What went well?
  - What could be improved?
  - What were impediments for us?
- **Decide and document action items**
- Discuss previous action items



Product Increment

Potentially shippable increment

- Complete according to **Definition of Done**
  - Even if not actually released to users
- **No regrets** if project ended now
  - Value was delivered to the customer
  - Customer more likely to hire company again



Scrum Basics

Team

- Product Owner
- Scrum Master
- Developers

Meetings

- Planning
- Daily Scrum
- Review
- Retrospective

Artifacts

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

These are the basics. Customize for your team!