Project Management

HPC-04 Software Engineering

Prof. Dr. Christoph Schober

Definition of a project

[A project is] a piece of planned work or an activity that is finished over a period of time and intended to achieve a particular purpose:

- *a scientific research project*
- *Her latest project is a film based on the life of a 19th-century music hall star.*
- *My next project is decorating the kitchen.*

(Source: Cambridge Dictionary)

A project is a temporary effort with a specific goal, planned activities, and a set timeframe and budget. It's distinct from ongoing operations, has defined start and end points, and aims to create a unique result or outcome.

(Source: ChatGPTv3)

Project Varieties

Standard Projects

Well-experienced, standardized, and straightforward. Examples: technical customer projects, replacement investments.

Acceptance Projects

Well-defined tasks with some standardization, but require effective stakeholder communication. Examples: road construction projects, complex software projects.

Potential Projects

Tasks with open questions, relatively low risk, and a simple project organization. Examples: studies, feasibility assessments, research projects.

Pioneer Projects

Highly innovative, impactful, and risky interventions that span multiple areas. Difficult to estimate task scope. Examples: company mergers, self-driving vehicle development.

(Source: The Project Management Handbook, Kuster et al.)

Project Management

generic term for all **planning, monitoring, coordinating, and controlling activities** that are necessary for successfully finishing a project.

Program management

Coordinate priorities and resources of different interdependent projects

Portfolio management

Coordinate resources for projects within a company or business unit while minimizing risks

Product management

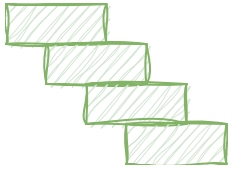
Coordinate all strategic and operational activities performed by an individual or team responsible for a product

→ Product Management often the first contact with project management when working as software engineer!

Types of project management

Different types of projects require different ways to manage them. There is no single best method.

Waterfall

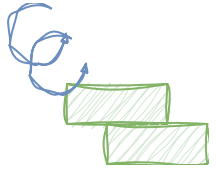


- Classic methodology
- Sequential project phases
- Final product delivered at the end
- Not so popular anymore in software engineering

Agile

- Developed specifically for software projects
- Iterative phases
- Self-empowered teams
- Continuous product increments

Hybrid



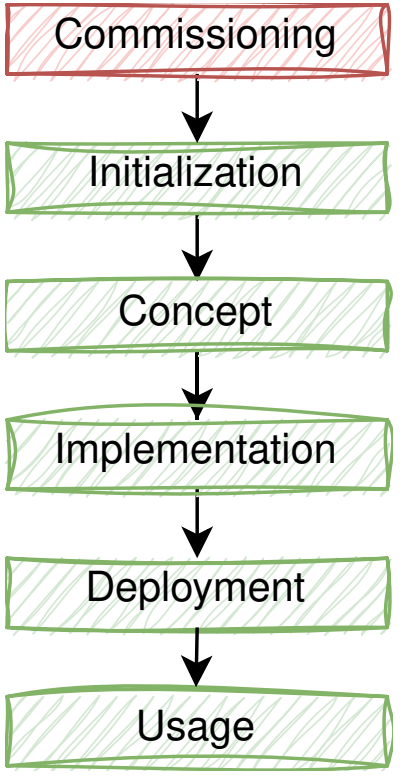
- Mixes agile and waterfall
- Can enable agile methods within a waterfall organization

The Waterfall: Commissioning

Define...

- strategic project goals
- business case
- added value
- preliminary roles

Target: Enable management to make informed decision about starting the project.

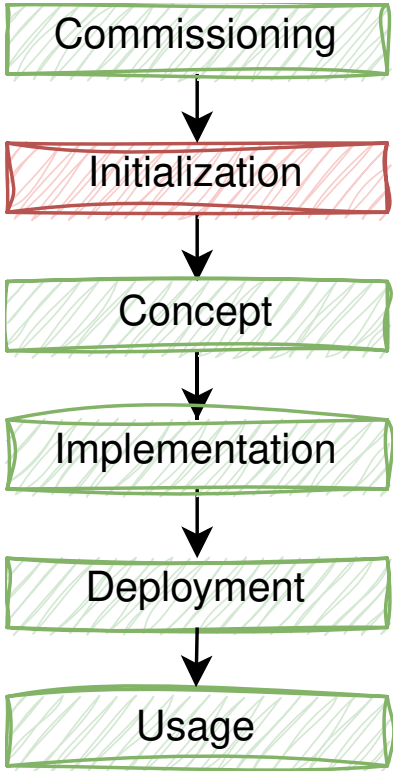


The Waterfall: Initialization

Define project charter with...

- Requirements
- Roles
- Stakeholders
- Risks
- Project plan

At the end of the initialization phase a project can still be stopped if outcome is not convincing

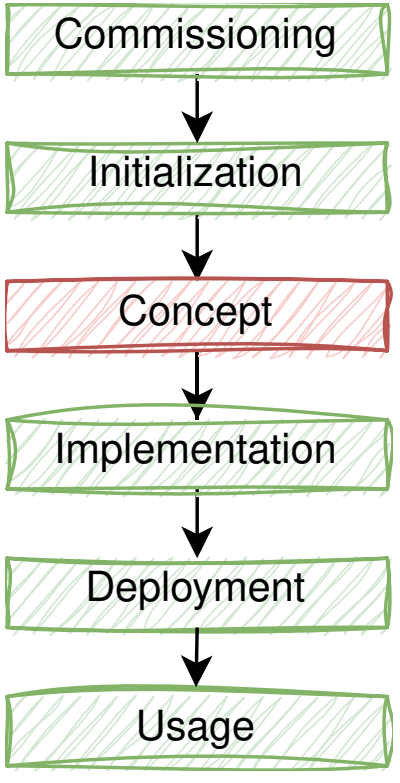


The Waterfall: Concept

Critical phase: Creation of detailed designs and plans based on requirements

- system architecture
- data modelling
- user interfaces
- ...

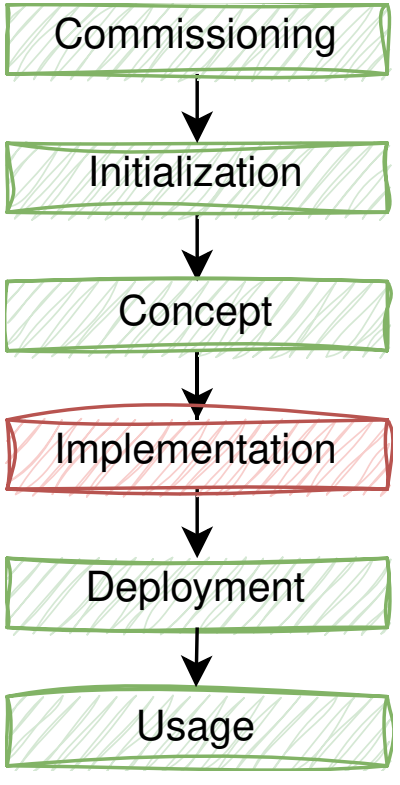
→ Important to take into account all constraints and requirements because a later change is not easily possible



The Waterfall: Implementation

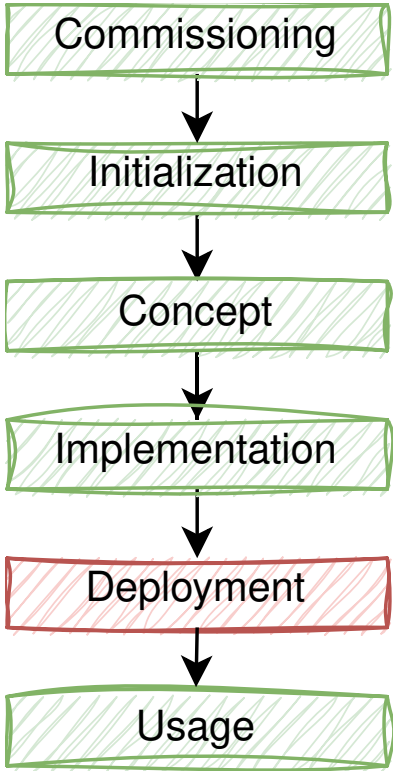
- First code is written by developers according to the design specification from the concept phase
- Tests are written and executed according to the specification
- supplementary material such as documentation or user manual are created

→ The outcome of this phase is the final product!



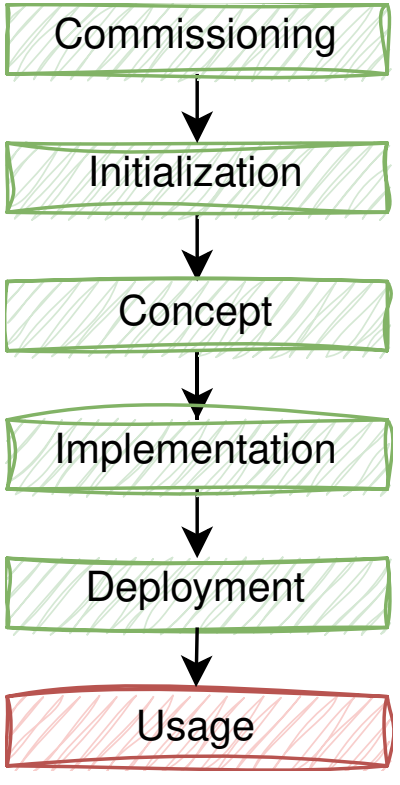
The Waterfall: Deployment

- The product is released and delivered to the customer (internal or external).
- Documentation is handed over to operations team
- Project related resources are finalized and closed (Jira projects, cost centers, ...)



The Waterfall: Usage

- Product is in use
- Often AMS (Application Management Service) contracts exist to handle support requests
- Team might support for critical bug fixes



Agile Project Management

Why another project management technique? What's wrong with waterfall?

Task: Take 5 minutes and discuss this with your neighbor. What problems can you see with waterfall (especially in software projects)?

- not enough knowledge about details when project is started
- requirements evolve faster than in classic industrial projects
- technological pace is much faster
- cost for change is different

Another concept to describe this (see PMI-ACP Exam Prep, M. Griffiths) is the transformation of project work due to the 'Information Revolution' (following in the industrial revolution):

- Knowledge work is less predictable than manufacturing
- More communication and collaboration required

Manifesto for Agile Software Development

https://agilemanifesto.org (2001)

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.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

The Agile Manifesto

That is, while there is value in the items on the right, we value the items on the left more. (Suffix Agile Manifesto)

- a critical addition in the original manifesto that sometimes is forgotten
- decision 'pro/contra agile' is often a heated black-and-white (or 'right-and-wrong') discussion

It is important to remember that while being agile is a mindset, it always has to adapt to the circumstances and has different shades of 'agility'.

The Agile Manifesto: Value 1

Individuals and interactions over processes and tools

- focus on the team and interactions, not processes and tools
- problems get solved by people, not processes

The Agile Manifesto: Value 2

Working software over comprehensive documentation

- focus on the purpose and business value of the project over paperwork
- required documentation (e.g., regulatory) is part of the project work

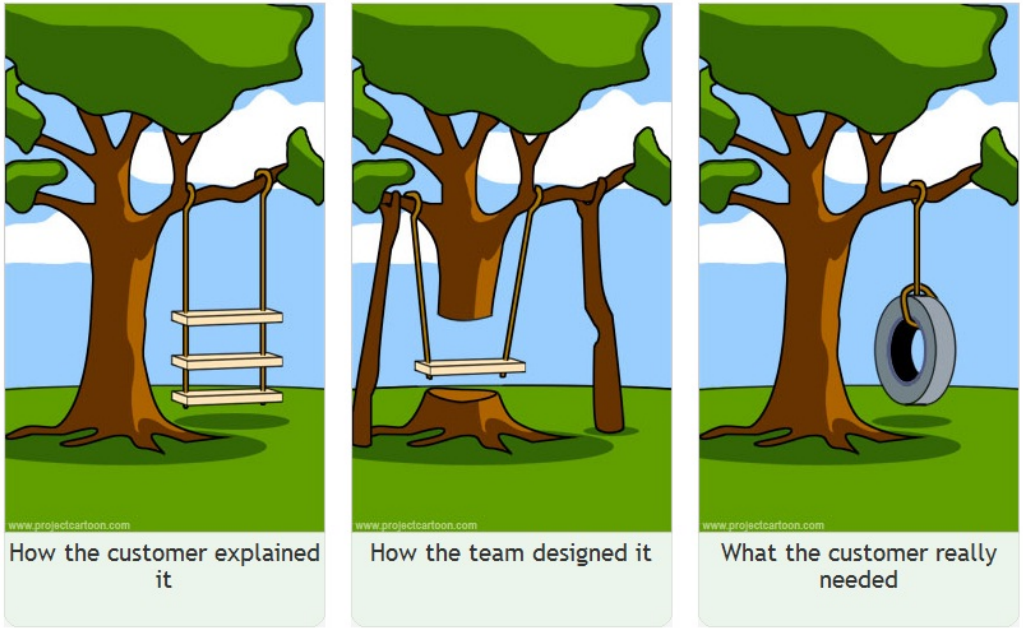
Or: just enough, just in time — and sometimes, just because (from PMI-ACP)

- Just enough documentation to focus on the outcome of the project: Documentation without functionality is not useful!
- Just in time or in the last responsible moment: Avoid spending extra time to update documentation when the product changes
- Just because: Sometimes it is important to decide on which battles to fight and especially in hybrid organizations it might be the most responsible approach.

The Agile Manifesto: Value 3

Customer collaboration over contract negotiation

- the customer is not your enemy!
- be flexible and accommodating
- work with the customer to create the solution the customer needs



https://projectcartoon.org

The Agile Manifesto: Value 4

Responding to change over following a plan

- initial plans are done when knowledge about the project is lowest
- adapting the plan during the project is required and okay
- agile methods account for this by fast iterations, feedback and visibility of work

The twelve agile principles

In addition to the four core values a set of 12 principles was defined: https://agilemanifesto.org/principles.html

Task: Go to the page and read all 12 principles. Pick one that you find most interesting, unexpected or useful and summarize why. Hint: You can also actively decide to not pick one, then please explain why!

(1) Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

(2) Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

(3) Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

(4) Business people and developers must work together daily throughout the project.

(5) Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

(6) The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

(7) Working software is the primary measure of progress.

(8) Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

(9) Continuous attention to technical excellence and good design enhances agility.

(10) Simplicity--the art of maximizing the amount of work not done--is essential.

(11) The best architectures, requirements, and designs emerge from self-organizing teams.

(12) At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Methodologies

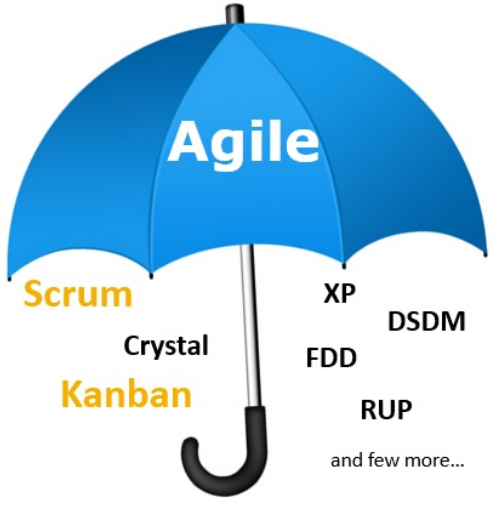
Which ones do you know?

- Scrum
- Kanban
- Extreme Programming
- Lean Programming
- Crystal Family (Crystal, ...)
- Scaled Agile Framework (SAFe®)
- Large Scale Scrum (LeSS)
- ... and some more!

A different view: The Agile Umbrella

Agile Umbrella

Methods introducing Agility

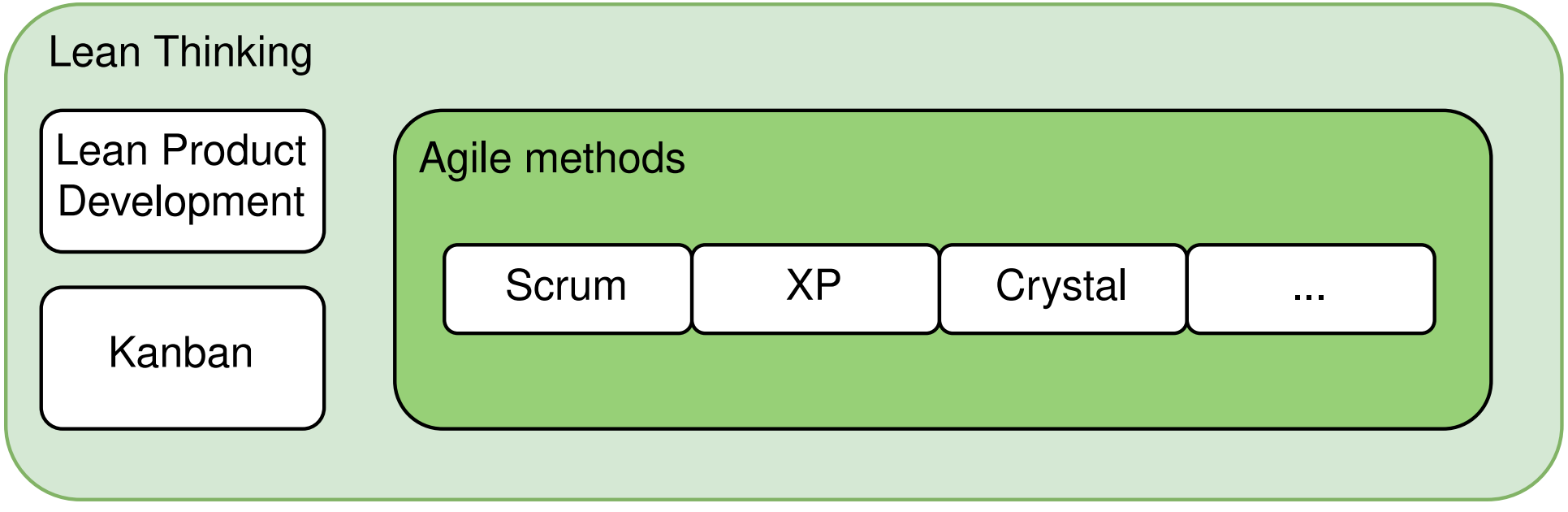


More Prescriptive
more rules to follow

- **RUP (120+)**
  RUP has over 30 roles, over 20 activities, and over 70 artifacts
- **XP (13)**
- **Scrum (9)**
- **Kanban (3)**

More Adaptive
fewer rules to follow

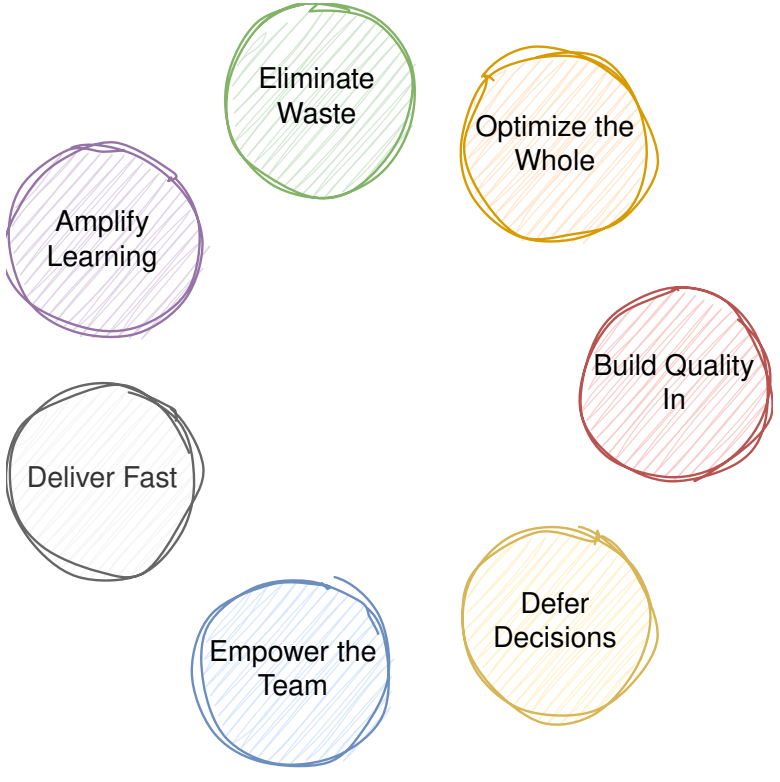
And yet another classification: Lean and Agile



→ Take-away: There are many approaches to define, explain and categorize agile software development, but many core concepts are shared by agile methodologies.

Lean

Derived from Lean Manufacturing and the Toyota Production System (1950s+1960s)



The 7 core principles of Lean

Eliminate Waste

Identify and remove waste from project work: Partially done work, delays, handoffs, etc. Most important principle behind the original lean manufacturing approach

Empower the Team

Trust the team to make the right local decisions for their work items

Deliver Fast

Produce valuable deliverables and iterate to find the best solution.

Optimize the Whole

Don't see work isolated, always take into account the bigger picture and how it works towards the goals of the organization

Build Quality In

Quality (and testing) is not done afterwards, but built into the process: Unit tests, refactoring and continuous integration

Defer Decisions

Balance early planning with late commitments. Allow reprioritizing!

Amplify Learning

Communicate early and often, get feedback from peers and stakeholders as soon as possible

The Wastes of Lean

The original lean defined wastes (japanese *'muda'*) that have been adapted to software engineering

Partially done work
: Code waiting for testing or specs waiting for development

Extra processes
: Work that does not add value to the product like unused documentation

Extra features
: Features not strictly required but considered 'nice-to-have'

Task switching
: Multi-tasking between different projects or responsibilities

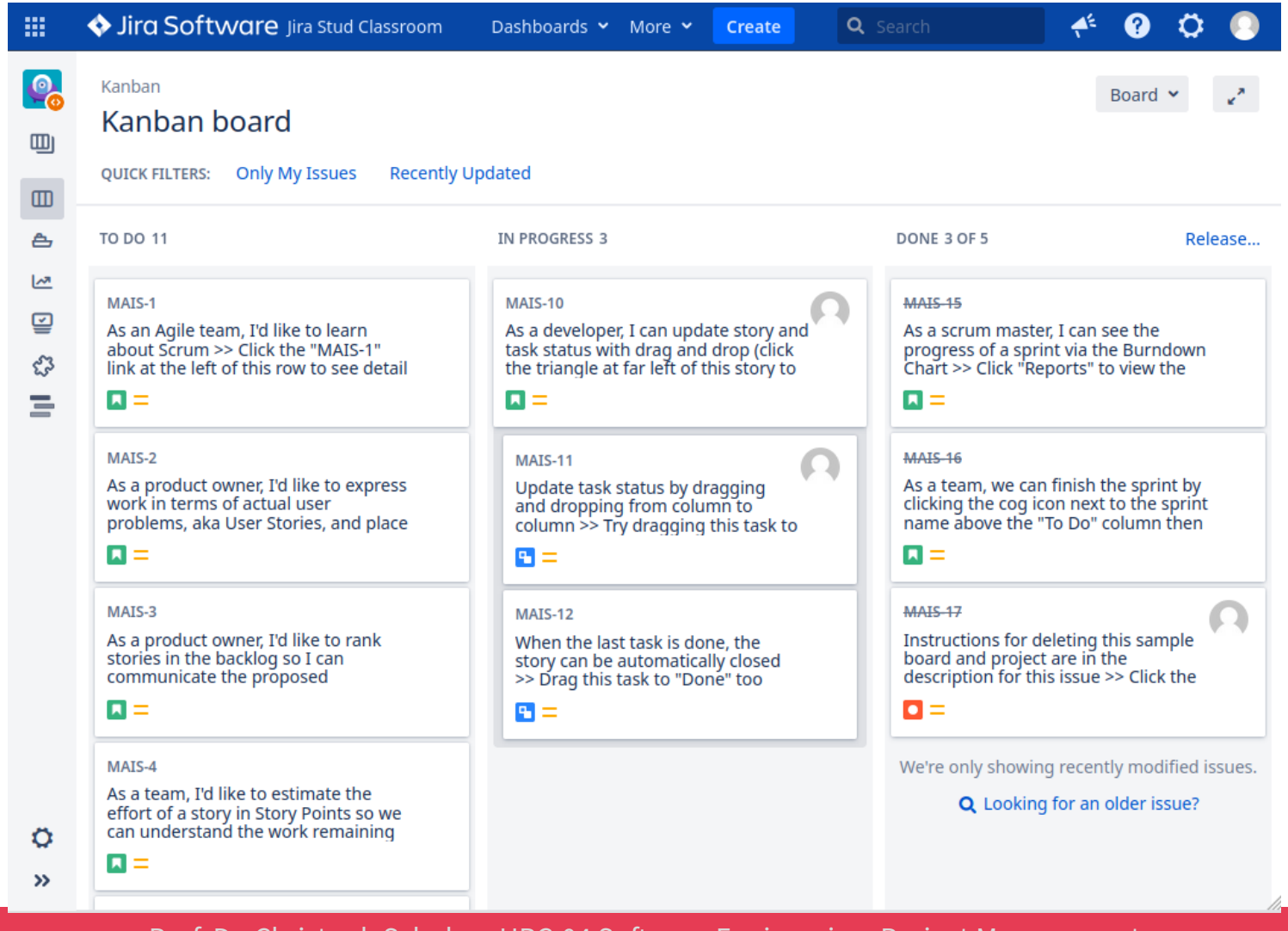
Waiting
: Delays waiting for (code) reviews

Motion
: Effort required to communicate and move information between teams

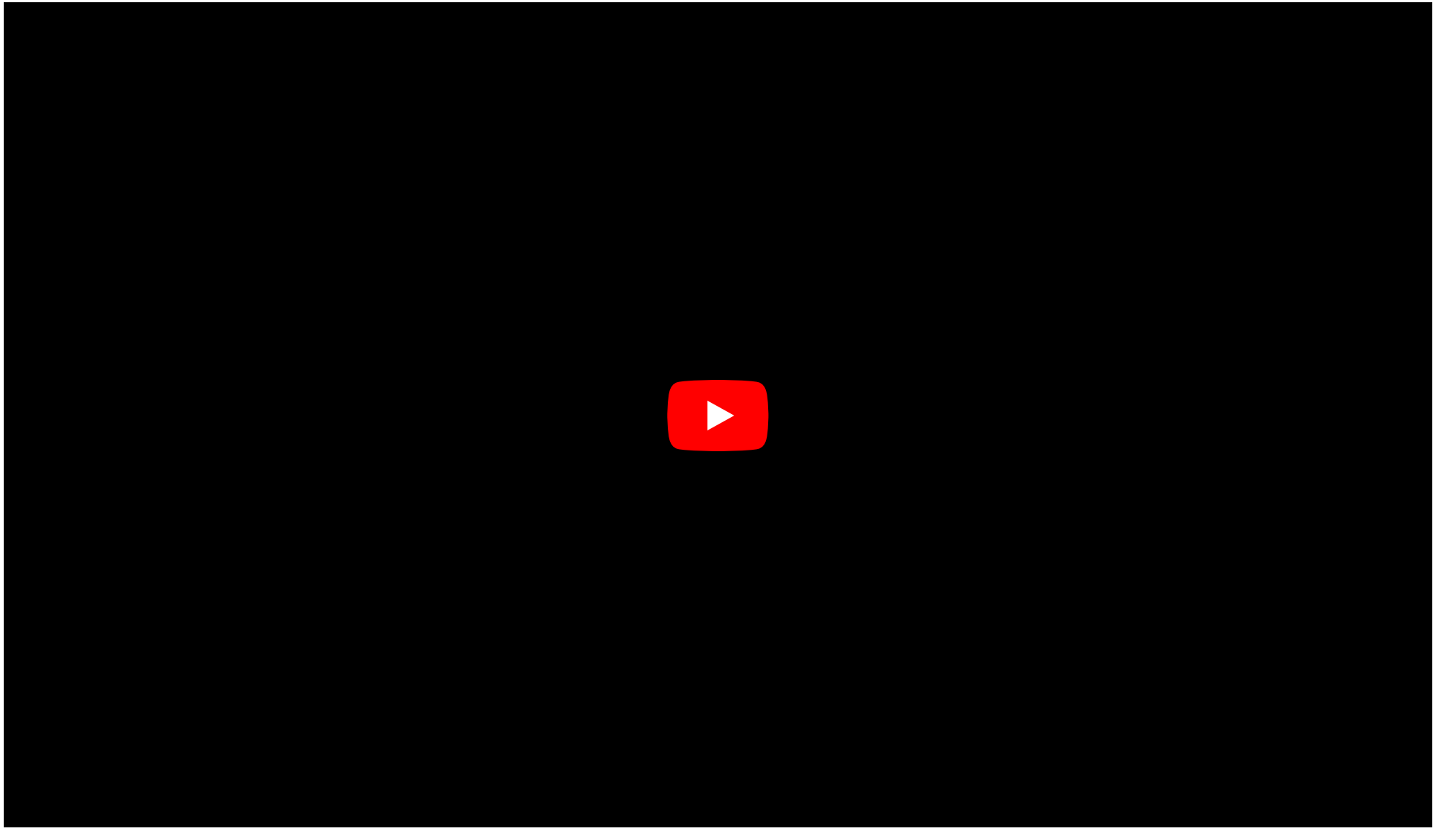
Defects
: Bugs in the product that must be fixed and distract from the current work

Kanban

Also a technique from the Toyota Production System. Kanban means 'signboard':



What is Kanban?



Five Principles of Kanban

Visualize the workflow

Knowledge work is often invisible. Kanban boards visualize the work being done

Limit WIP (Work In Progress)

Restricting the work done in parallel improves productivity and increases the visibility of issues and bottlenecks

Manage flow

Track the flow of items through the system to identify issues and measure effectiveness of changes (also see "lead time")

Make process policies explicitly

Everyone must know how things work to follow the process and discuss improvements (e.g., coding standards, handling of bugs or new features, backlog)

Improve collaboratively

A team should improve continuously by learning from errors or by testing new ideas.

→ Kanban does not enforce specific ideas how work is actually done, but how work progresses through the system.
→ Kanban can easily be combined with other agile techniques! allows

Finally, Scrum!

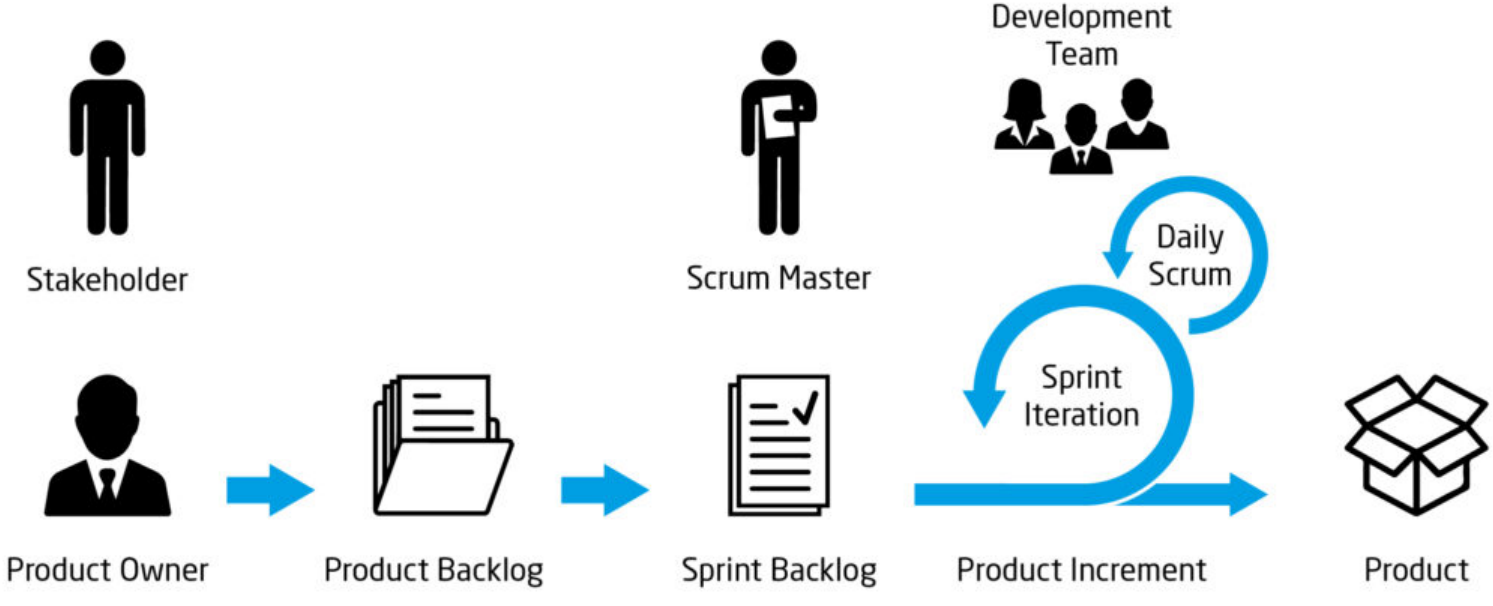
Introduced **1986(!)** by Hirotaka Takeuchi and Ikujiro Nonaka in an article in the Harvard Business Review ("The New New Product Development Game")

- Systematic analysis of new product development approaches in the US and Japan (Fuji-Xerox, 3M, Hewlett-Packard, ...)
- Named 'Scrum' in an analogy to rugby (Anyone knows why?)
- Multi-disciplinary teams work continuously together (in contrast to experts working sequentially (Architect → Developer → Tester)

Scrum in Software Development

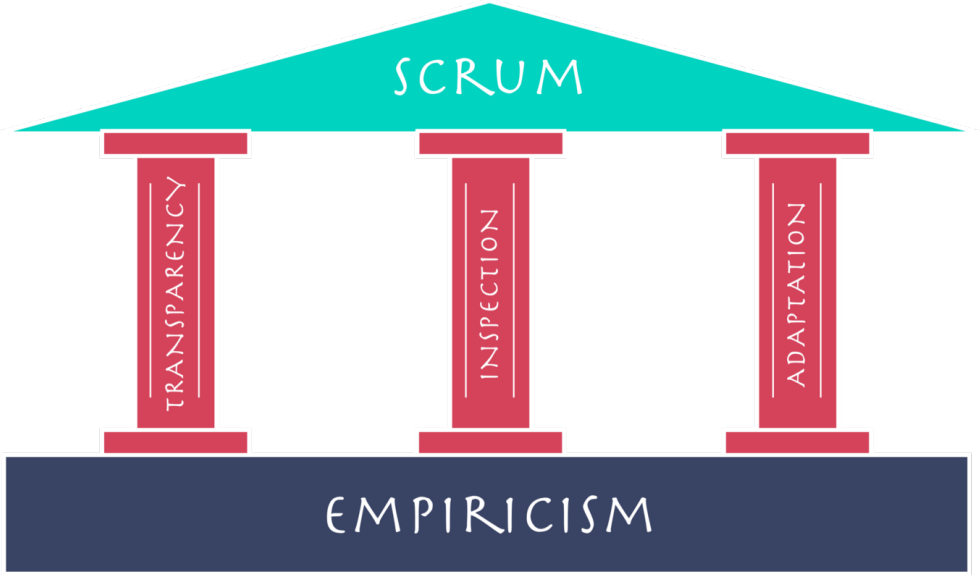
In A Nutshell (Source: The Scrum Guide)

1. A Product Owner orders the work for a complex problem into a Product Backlog.
2. The Scrum Team turns a selection of the work into an Increment of value during a Sprint.
3. The Scrum Team and its stakeholders inspect the results and adjust for the next Sprint.
4. Repeat.



visioneleven.com

What does this require?



seriousscrum.com

- **Empiricism**: Knowledge comes from experience and decisions are made based on observations (the opposite of empiricism is rationalism)
- **Transparency**: Work must be visible to developers and stakeholders alike; important decisions depend on it
- **Inspection**: Progress and current state of work must be inspected regularly (see Scrum events!) to detect problems and to adapt (see next)
- **Adaption**: Consequence of inspection must be adaption: If something deviates from the goal, the Scrum team must adjust its current plan

Scrum Values

Scrum relies on a set of values, but they are not required to start with Scrum - they are expected to be developed by a team following the principles.

Commitment
to achieve the goals and support each other

Focus
on the work of the sprint and nothing else

Openness
towards the work, challenges and any problems

Respect
each other and the stakeholder

Courage
to tackle tough problems and learn from mistakes

Scrum Overview

The Team

"We try to create teams that are no larger than can be fed by two pizzas. We call that the two-pizza team rule." (Jeff Bezos)

An ideal Scrum team consists of 5-10 people:

- 1(-x) Product Owner
- 1 Scrum Master
- z Developers

The Events

"Each event in Scrum is a formal opportunity to inspect and adapt Scrum artifacts." (The Scrum Guide)

Big goal: Minimize the need for other meetings!

1. Sprint
2. Sprint Planning
3. Daily Scrum
4. Sprint Review
5. Sprint Retrospective

The Artifacts

"Scrum's artifacts represent work or value. They are designed to maximize transparency of key information." (The Scrum Guide)

Each artefact stands for a specific commitment

1. Product Backlog
2. Sprint Backlog
3. Increment

Scrum Events

Literature Task

Download 'The Scrum Guide' at https://t1p.de/f840e or from the iLearn course and read the introduction and your assigned role/event/artefact (and as much context as you require).

In the group

1. Write a summary of the event using the template provided in iLearn (e.g., "Scrum_Event1_Summary.docx")
2. Upload it in iLearn ("Summary of Scrum..")
3. Present and explain it to the audience

Example Role

The Product Owner

Responsible for the success of the development project.

Dos

- create and communicate the product vision and goal
- manage stakeholders
- manage and prioritize product backlog items

Don'ts

- make technical decisions
- change the sprint backlog during a sprint
- develop functionality

Example Event

The Sprint

Cadence of scrum: The container for everything else.

- **Duration:** 1 to 4 weeks
- **Frequency:** continuous
- **Goal:** Reaching the sprint goal and deliver an *Increment*
- **Involves:** Scrum Team

Description:

The sprint is the defining event of Scrum. All other events take part during a sprint.

Example Artefact

The Product Backlog

Commitment: Product Goal

- **Responsible:** Product Owner
- **Purpose:** Give everyone transparency about the product vision and potential next steps
- **Used in event(s):** Sprint planning

Details

- contains the product vision broken down into small work items (e.g., user stories)
- single source of truth for team
- continuously adapted by Product Owner during 'backlog grooming' / 'backlog refinement'

User Stories

"Software requirements is a communication problem. Those who want the new software (either to use or to sell) must communicate with those who will build the new software."
Mike Cohn, User Stories Applied, 2004

One common way to communicate requirements in agile are user stories and story cards.

A user story is

- a short, simple description of a feature
- told from the perspective of the person who desires the new capability

As a [persona] I [want to], [so that]...

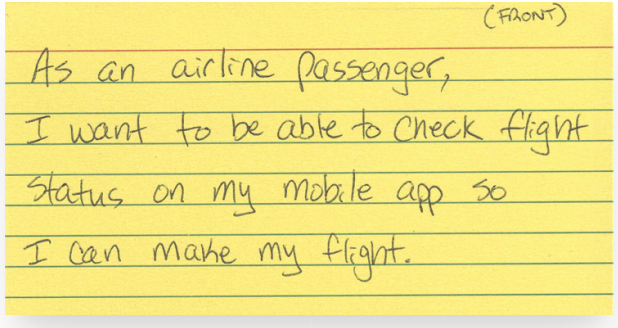
As a student I want to download the slides so that I can review the lecture at home.

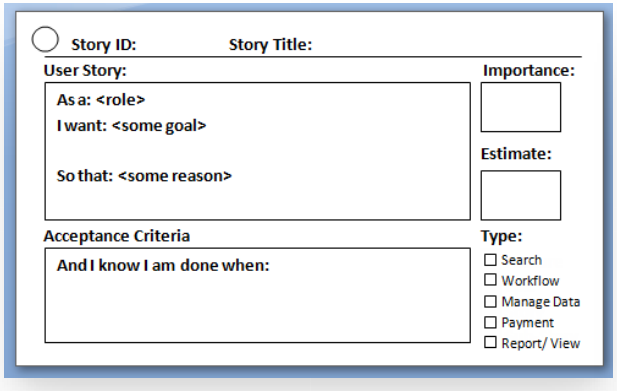
As a professor I want to save my slides as pdf so that I can upload them to iLearn.

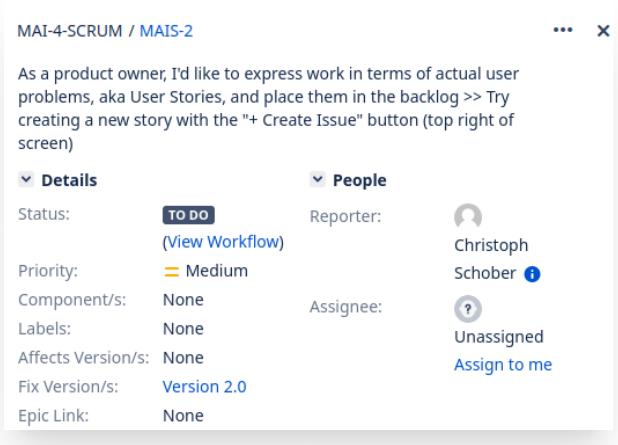
User Stories and Story Cards

User stories are often used together with story cards. A story card can be

- a paper card with hand-written user story
- a real template card with user story and additional information (acceptance criteria, estimates, ...)
- an entity in a digital system such as Jira, Trello, Miro and others







→ The right format depends on the way a team is working together (single office, hybrid or fully remote) and the maturity of a story (paper cards in brainstorming and planning, later digital card for development.

What is a good user story?

A good user story can be described by the INVEST principles (Wake, 2003):

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

I - Independent

An ideal story is independent of other stories. This gives maximum flexibility when prioritizing and scheduling the work.

- not always obvious how to make stories independent
- not always possible without increasing the size/complexity too much (see "Small"!)
- best effort and worth some headaches!

N - Negotiable

An ideal story covers the essence of the task, not the details. They are added during the development.

- story is a living document and not a legal contract
- can be changed during the development

V - Valuable

An ideal story is valuable to the customer, not the developer.

- different way of thinking: Not the technical requirement, but the user needs
- each story should deliver value to the customer (not the developer)
- when splitting stories each story shall be valuable on its own (slice into smaller features, but not technical layers such as database layer and presentation layer!)

E - Estimable

An ideal story is understood by the team and can be estimated with sufficient accuracy.

- not too complex (if, then split!)
- team has required knowledge (if not, a **spike** can help)
- estimates are no promises or deadlines (teams need to feel safe to make realistic estimates, otherwise they will add too much safety time!)

Spike: A time-boxed exploratory story to acquire knowledge necessary for estimation of the original task.

S - Small

An ideal story is small enough to be understood, estimated and implemented with sufficient confidence.

- definition of "small enough" depends on team: From few person days to weeks
- smaller tasks are easier to discuss
- tradeoff with story independence: Too small tasks will often depend on others

T - Testable

An ideal story has clear and objective acceptance criteria that can be tested (automatically).

- clear "definition of done" for developer when acceptance criteria are fulfilled
- unclear acceptance criteria (or difficulty of PO/Stakeholder to formulate them) indicate problems with the user story itself

User Story Exercise

You will be roleplaying in pairs in order to complete this exercise. One of you will be a developer, the other a client (and then you will switch).

When you are playing the client:

- Think of a piece of software that you would like to create or enhance.
- Think of a feature that you would like to add or change
- Don't make it too easy for the developer. Describe the features in natural language

When you are playing the developer:

- Ask questions and practice active listening techniques while you take notes.
- Get enough detail so that you could write up a User Story (per person)
- Write down a user story on a piece of paper (and keep it, you'll need it later!)

As a [persona] I [want to], [so that]...

Example for a conversation

Important: This is an example, you are not expected to write down your conversation!

[Client, a professor] I need a feature in iLearn to upload more files at once to a course instead of uploading file after file and clicking all the time with my mouse.

[Developer, ITC] Can you describe where you need to upload multiple files?

[Client, a professor] Eh yes, when I upload my slides using this "Folder" thingy and I have 2 PDF files for a lecture, I need to edit it, click "upload file", select one, upload, click "upload file", select the other, upload it.

[Developer, ITC] Ok, so you want to be able to select two files in the file selector and upload them in one go?

[Client, a professor] YES! Can you do that until tomorrow morning? That would be great!

[Developer, ITC] Weeeeeeell. Let's first check if I got everything correct and then I'll estimate it with my team in the next iteration!

As a lecturer in iLearn I want to select and upload multiple files from my computer in the Folder block so that I save time when uploading lecture slides.