Software Engineering

Prof. Dr. Jóakim von Kistowski

Check-In

Which process models do you know?

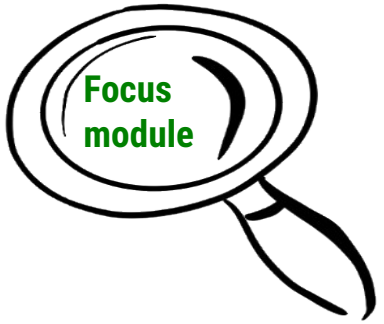
TH Aschaffenburg
university of applied sciences

History of Processes and Process Models

Terms: Process / Procedure

Typical Processes in Software Development

Disciplines in software engineering



Basic topics

Configuration management | Documentation |
Knowledge management | People in the SWE process and digital ethics | Tools

Development

Requirements
- Context analysis
- Requirements Engineering

Design
- Architecture
- Detailed design

Implementation

QualityMgt.

Quality assurance and testing
- Test, inspection, metrics

Processes and procedure models
- Improvement, process model, maturity levels

Evolution
- Roll-Out
- Operation
- Maintenance
- Further development
- Reuse
- Reengineering
- Change management

Management
- Strategy
- Economy
- Team
- Dates
- Risks
- Customer, client/contractor
- Innovation

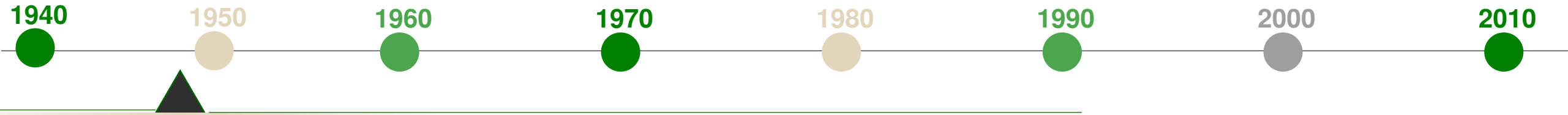
TH Aschaffenburg
university of applied sciences

History of Processes and Process Models

Terms: Process / Procedure

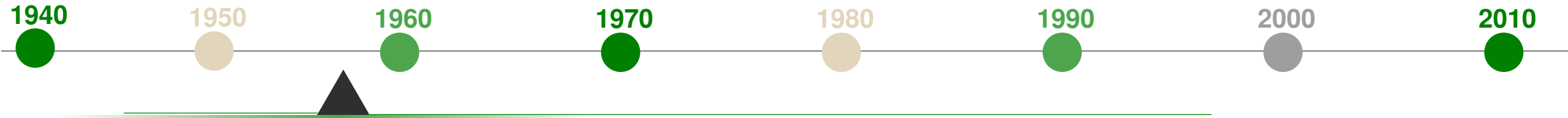
Typical Processes in Software Development

The discovery of procedure/process models



- In the early days, programming was classified as a "trivial task".
- **Software = Code**
- Errors were seen as manageable.

The discovery of procedure/process models



- From the 60s onwards: „**Systems“ and documentation** should ensure that good, low-error software is created with acceptable effort.
- Typical view: **higher programming languages** (Fortran, Algol, COBOL, ...) and elementary programming rules solve the problem.
- But: This only works for **individual** developers, not for projects with several people involved.

TH Aschaffenburg
university of applied sciences

The discovery of procedure/process models

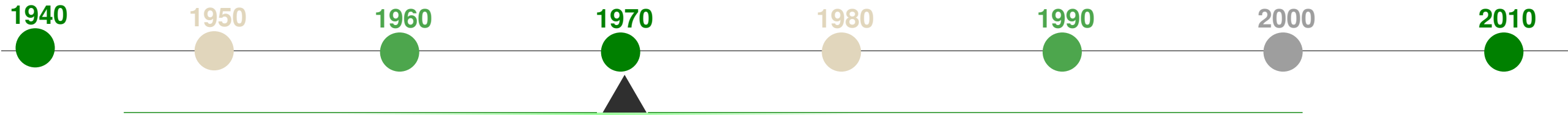
1940 1950 1960 1970 1980 1990 2000 2010

1969

NATO conference in Garmisch Partenkirchen, where F.L. Baur coined the term "software engineering" for the first time.

https://cdn1.vogel.de/unsafe/fit-in/1000x0/images.vogel.de/vogelonline/bdb/1469600/1469671/original.jpg

The discovery of procedure/process models

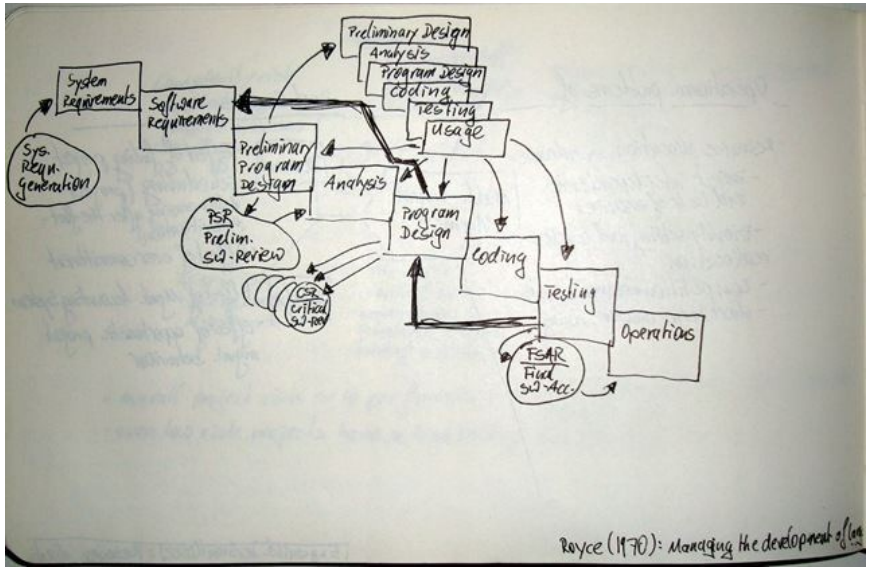


- In addition to programming in smaller contexts, large-scale programming gained in importance and attention.
- **Procedure and process models** were **formulated** and introduced for necessary steps.

https://cdn1.vogel.de/unsafe/fit-in/1000x0/images.vogel.de/vogelonline/bdb/1469600/1469671/original.jpg

The discovery of procedure/process models

1940 1950 1960 1970 1980 1990 2000 2010



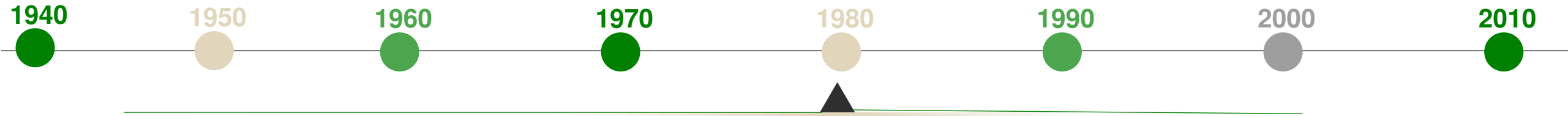
1970

Winston W. Royce, The waterfall model

https://cdn1.vogel.de/unsafe/fit-in/1000x0/images.vogel.de/vogelonline/bdb/1469600/1469671/original.jpg

TH Aschaffenburg
university of applied sciences

The discovery of procedure/process models



- In the 1980s, the **quality of the processes** finally began to be systematically **assessed** and **improved**

https://cdn1.vogel.de/unsafe/fit-in/1000x0/images.vogel.de/vogelonline/bdb/1469600/1469671/original.jpg

History of Processes and Process Models

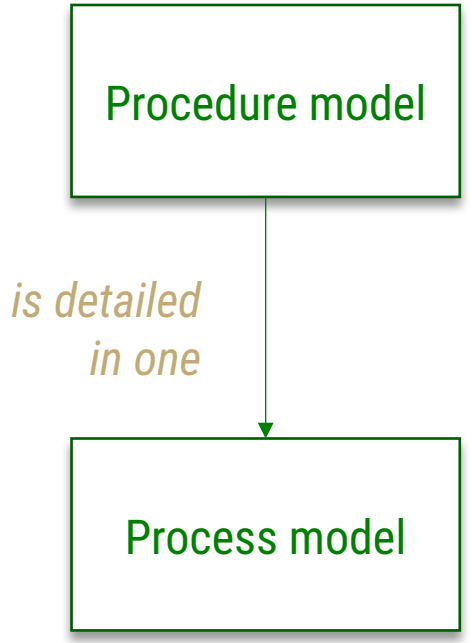
Terms: Process / Procedure

Typical Processes in Software Development

Procedure model vs. process model

- Procedure model
  - Describes the basic procedure, e.g. sequential (e.g. waterfall), iterative (e.g. spiral model)

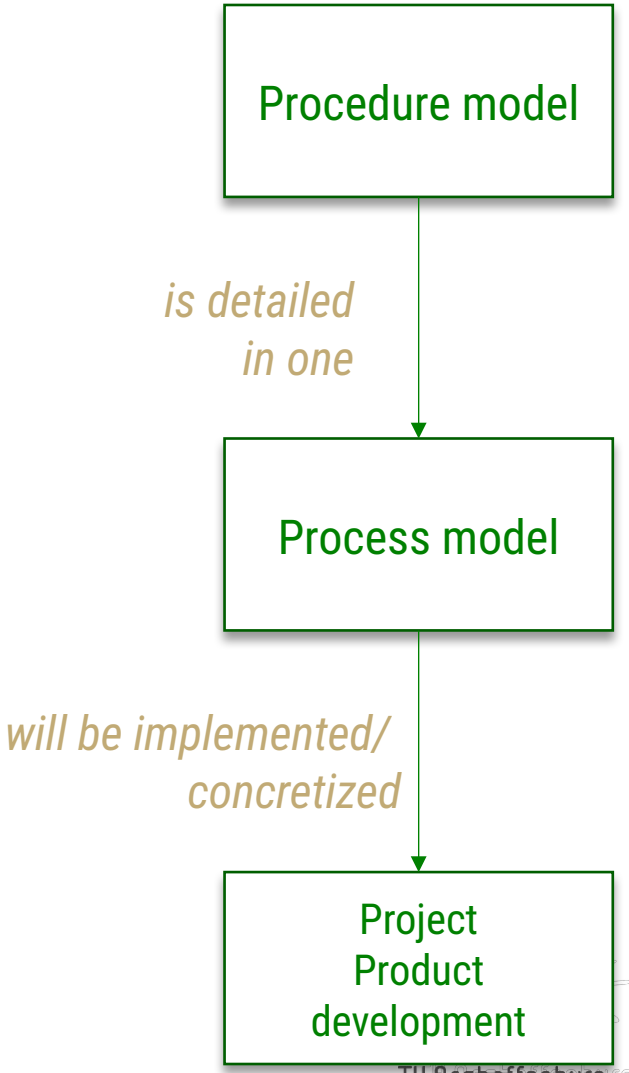
- Process model
  - includes several aspects including procedure model, organizational structure, QA and project management requirements, etc.
  - In a company, processes are defined on the basis of a process model.



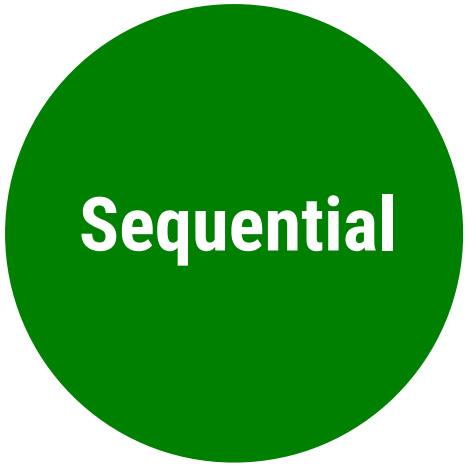
Procedure model vs. process model vs. project/product development

- Procedure model
  - Describes the basic procedure, e.g. sequential (e.g. waterfall), iterative (e.g. spiral model)

- Process model
  - includes several aspects including procedure model, organizational structure, QA and project management requirements, etc.
  - In a company, processes are defined on the basis of a process model.



General process models
→ From PM



Linear

Parallel

Repetitive

TH Aschaffenburg
university of applied sciences

History of Processes and Process Models

Terms: Process / Procedure

Typical Processes in Software Development

Typical Processes in Software Development

1. Code and Fix
2. Waterfall
3. V-Model (Boehm)
4. W-Model
5. V-Model XT
6. UP
7. XP
8. SCRUM
9. KANBAN

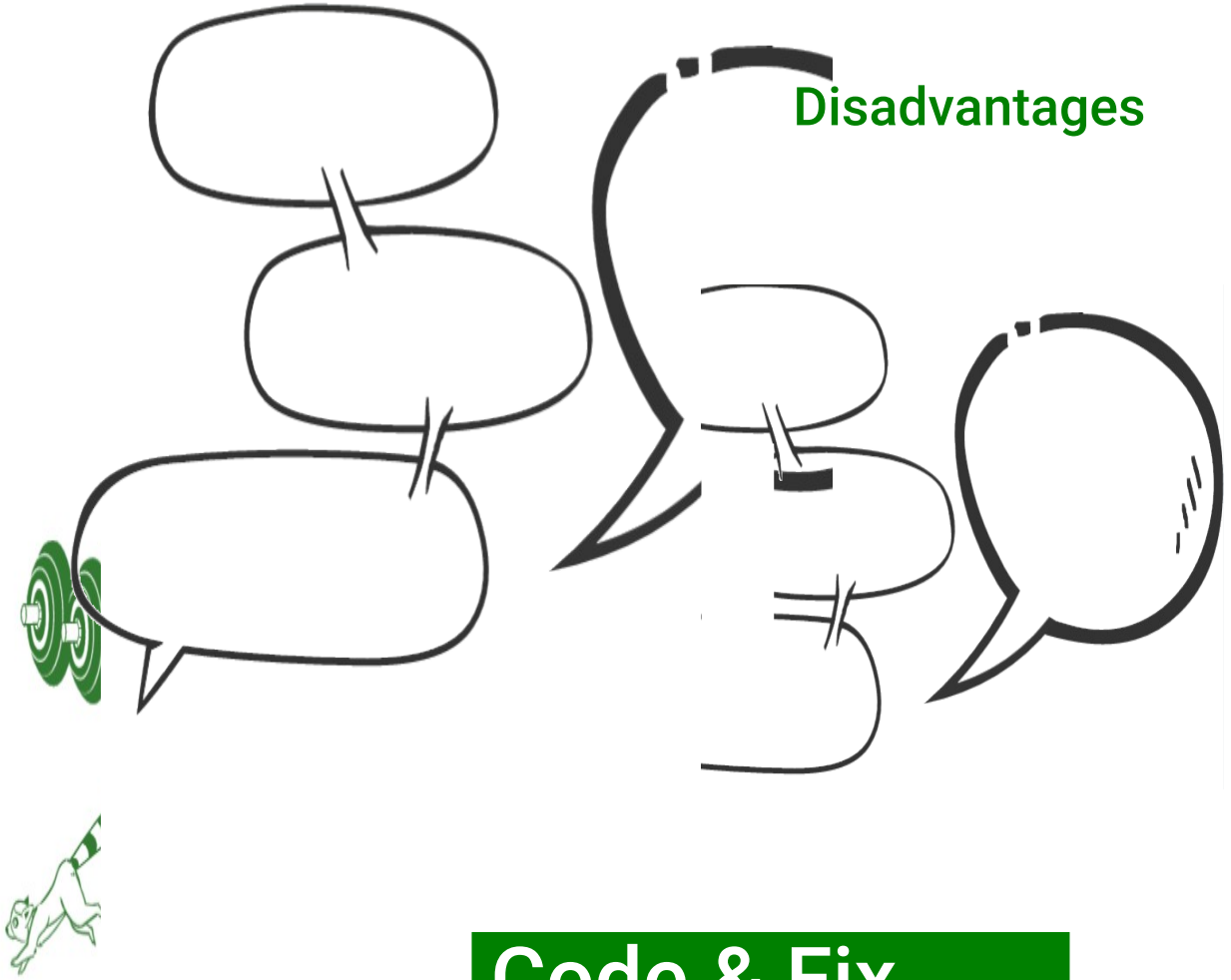
Code and Fix (aka Quick and Dirty)

Code and Fix

Refers to a procedure in which coding or correction, alternating with ad-hoc tests, are the only consciously performed activities in software development. [Ludewig, 2013]

Advantages

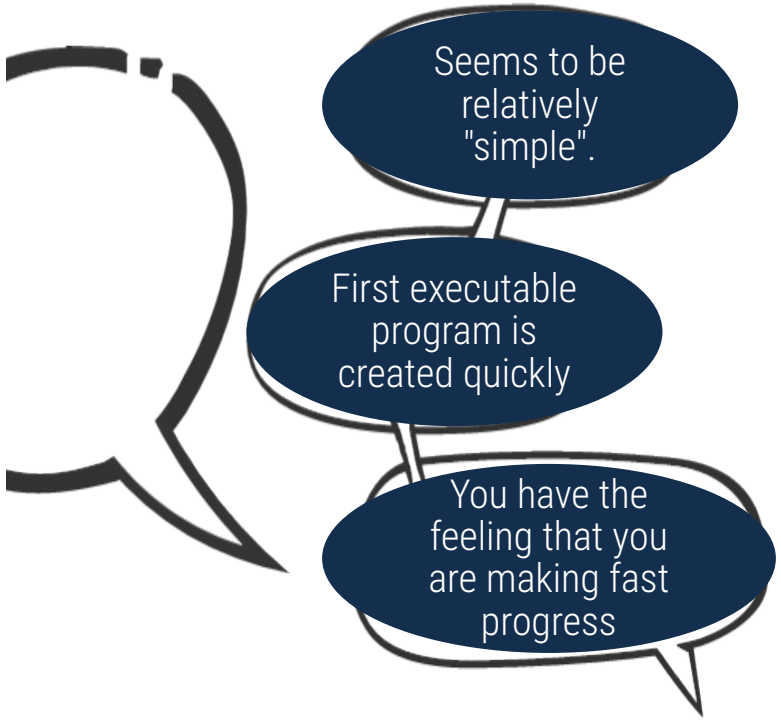
Disadvantages



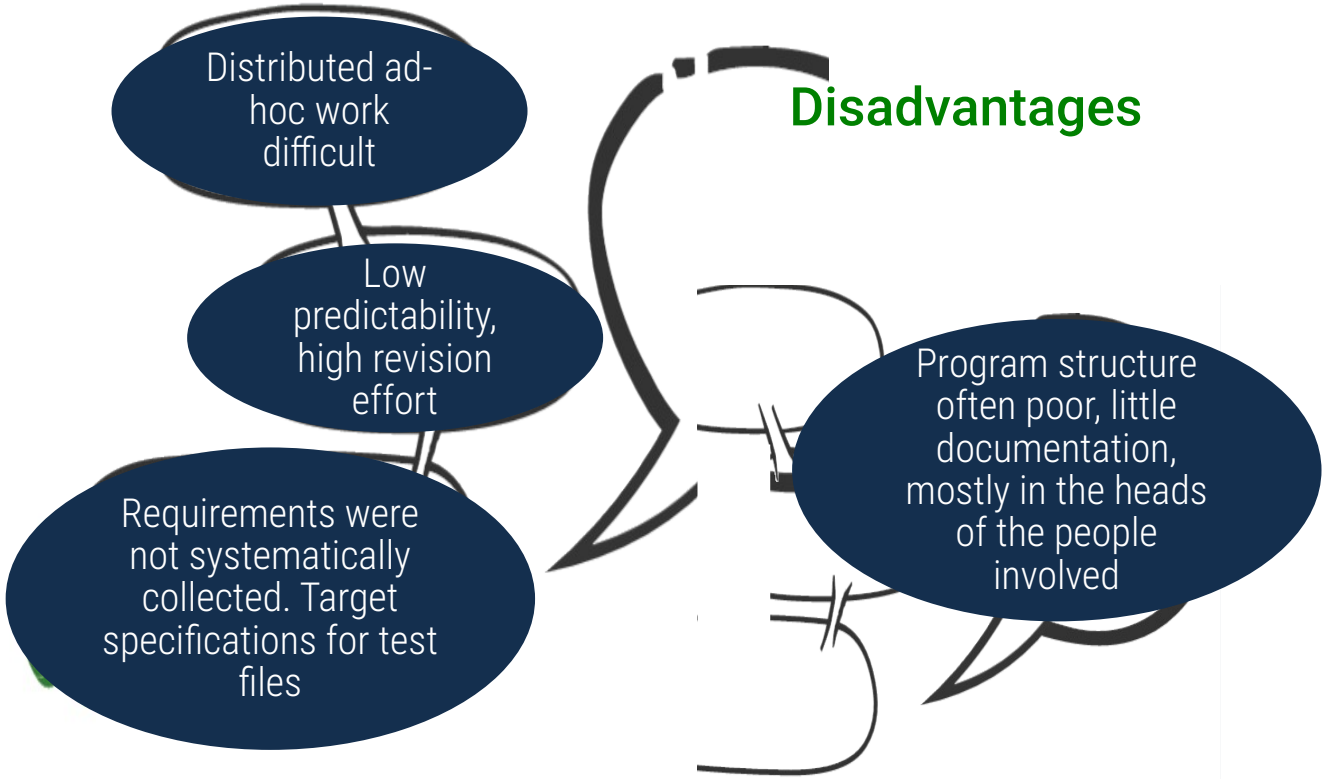
Code & Fix

Why is this used?
What are typical problems?

Advantages



Disadvantages



Code & Fix

Why is this used?
What are typical problems?

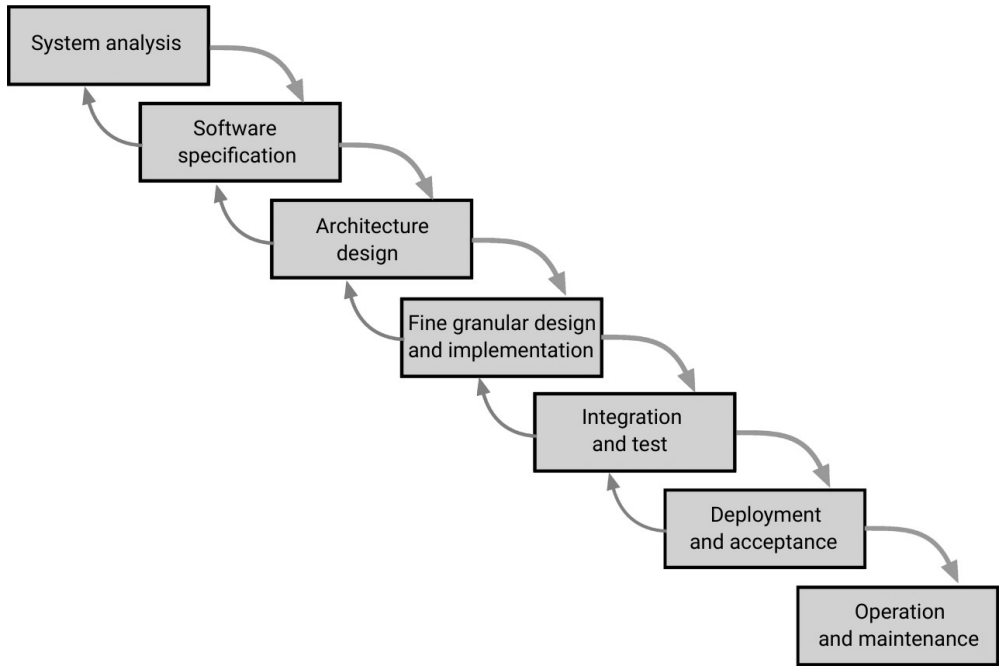
TH Aschaffenburg
university of applied sciences

Typical Processes in Software Development

1. Code and Fix
2. Waterfall
3. V-Model (Boehm)
4. W-Model
5. V-Model XT
6. UP
7. XP
8. SCRUM
9. KANBAN

The waterfall model

→Properties

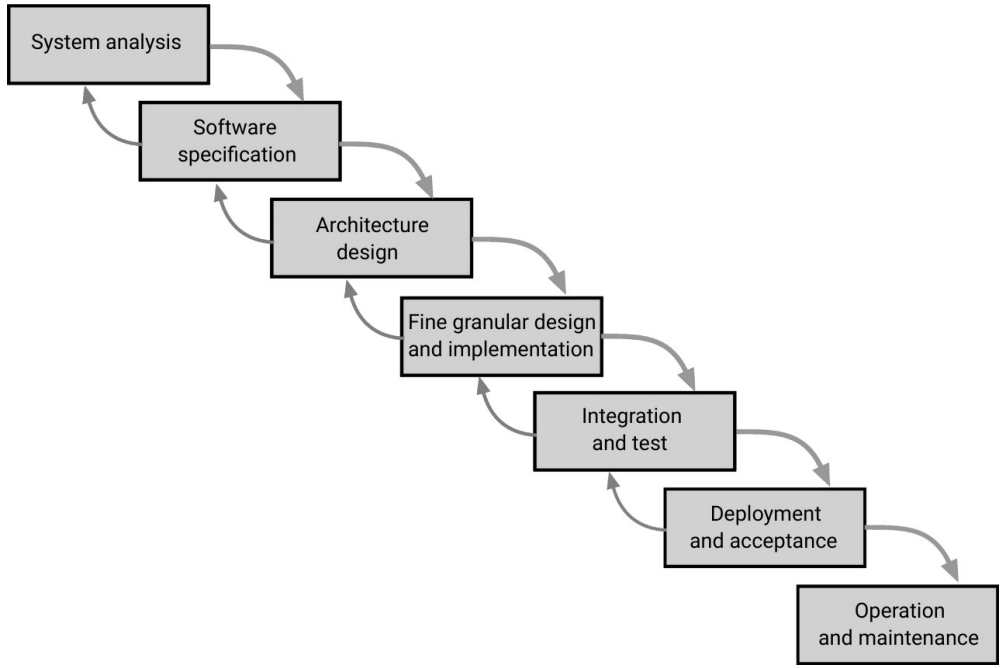


- Represents the software development process as a **sequence of phases.**
- Because of the cascade from one phase to the next, the model was given the name "waterfall".
  - **Rectangles** = activities/phases
  - **Down arrows**: desired transitions
  - **Arrows upwards** (eddies in the waterfall): Jumps back if errors are detected and need to be rectified.
  - There is no way from operations back to development!

Originally published by Royce [Royce, 1970].

The waterfall model

→ Notes



Originally published by Royce [Royce, 1970].

- Strict sequence was not intended. Steps back are also included in the original model.
- "Historically" they were more or less omitted. A strictly sequential model has become established in linguistic usage.
- What has (unfortunately) been created and practiced is a one-way street model.

The waterfall model

- → Discussion
  - **High quality:** How well is quality supported?
  - **Satisfied customers:** How well does it ensure usefulness for users?
  - **Maintainability:** How well is efficient development and further development supported?
  - **Cost/time:** How well is resource compliance supported?

The waterfall model

- → Discussion

  - **High quality:** How well is quality supported?
    - Strong dependency on early phases (requirements elicitation) and late testing are bad
  - **Satisfied customers:** How well does it ensure usefulness for users?
    - Late implementation => late tangible result is bad for communication
  - **Maintainability:** How well is efficient development and further development supported?
    - No special features, further development is not the focus
  - **Cost/time:** How well is resource efficiency (cost/time) supported?
    - Poor visibility of project progress (not visible for a long time) due to late implementation

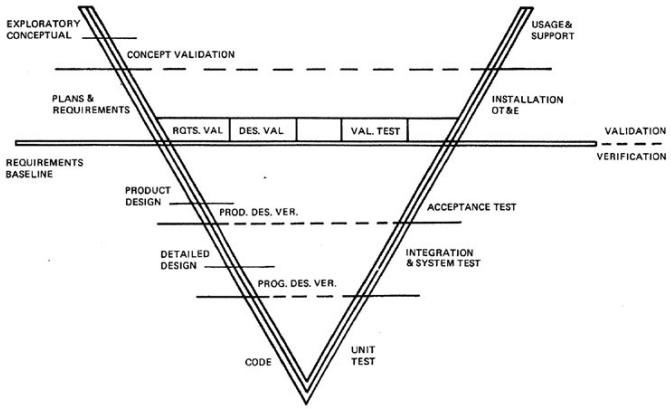
Typical Processes in Software Development

1. Code and Fix
2. Waterfall
3. V-Model (Boehm)
4. W-Model
5. V-Model XT
6. UP
7. XP
8. SCRUM
9. KANBAN

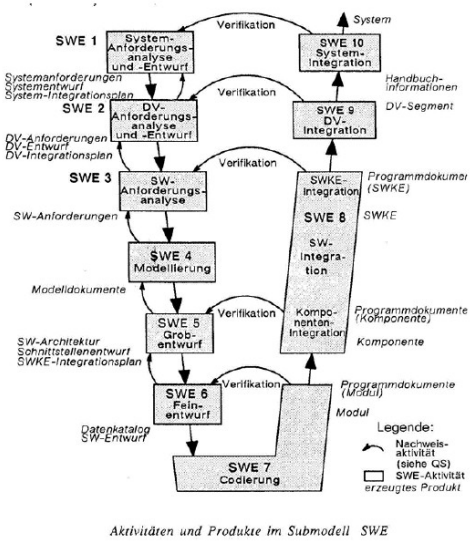
The V-model

→ Clarification of terms

- The V-model by B. Boehm **[Boehm, 1979]**
  - Sequential model, shown in V-shape



- The German federal V-model (binding since 1992)
  - Developed on behalf of the Federal Ministry of Defense and mandatory since 1992

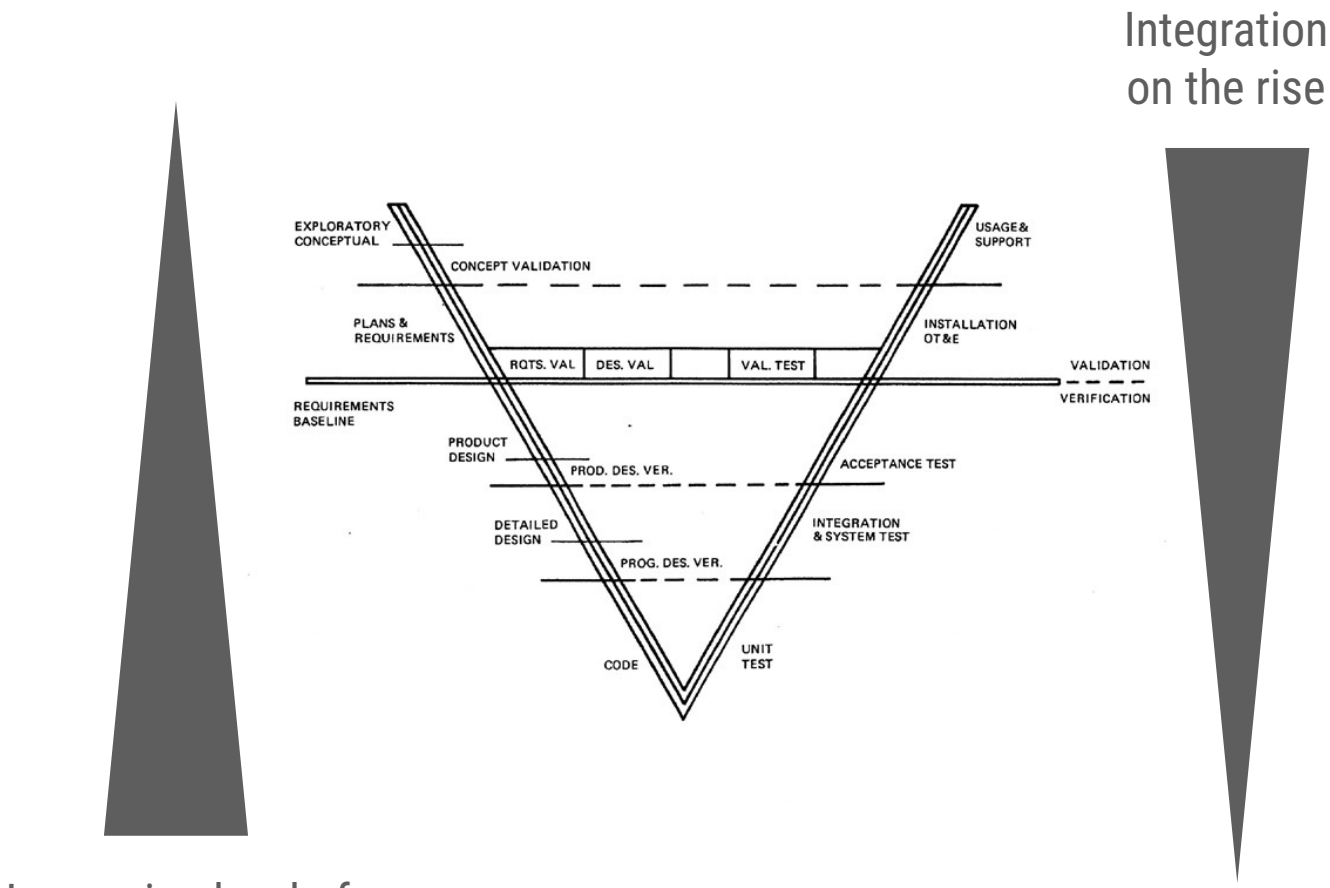


Aktivitäten und Produkte im Submodell SWE

[Federal Office of Works Technology and Procurement, 1989]

The V-model [Boehm, 1979]

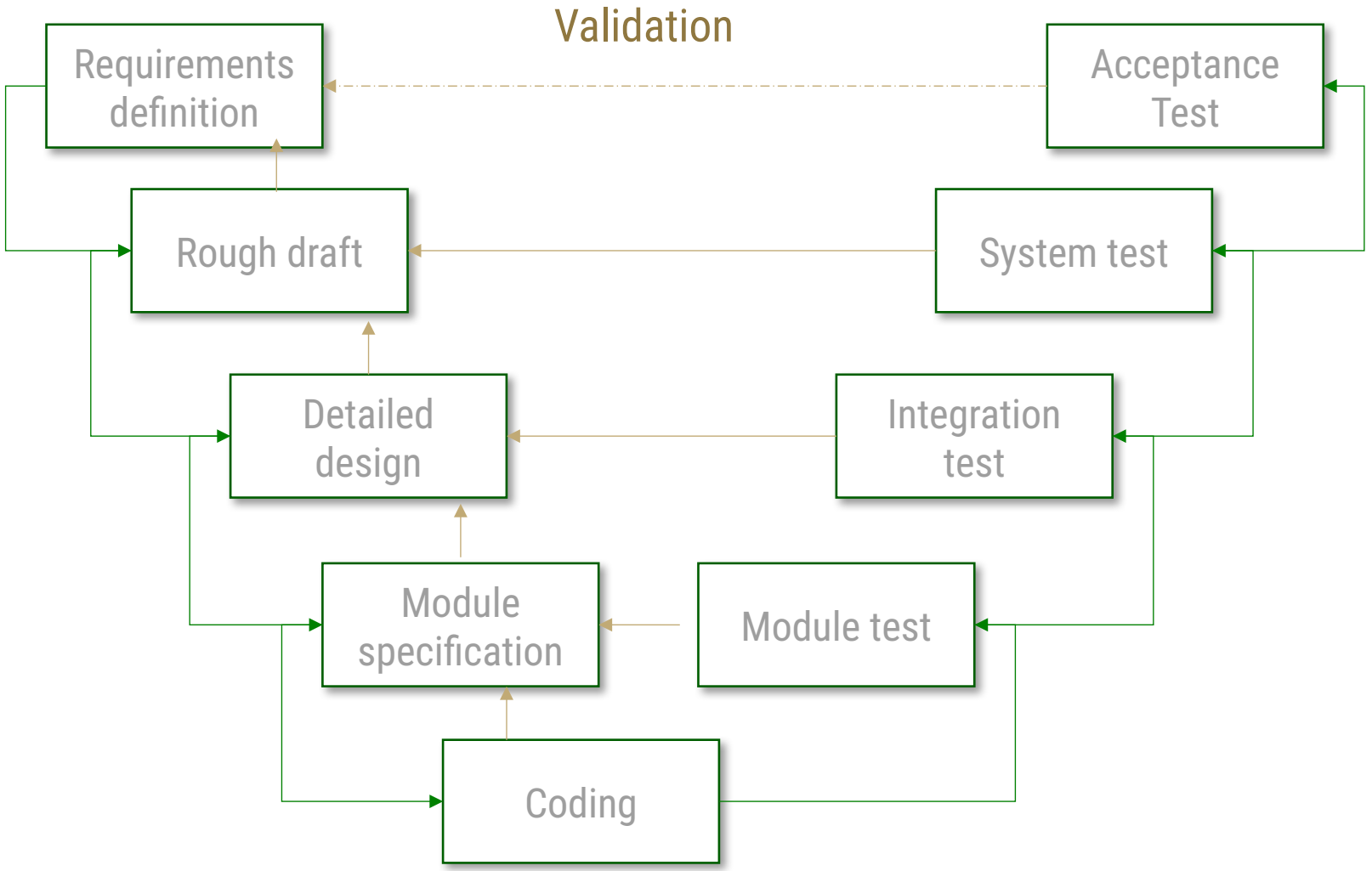
→ Origin



- Further development of the waterfall model
- Even if the illustration suggests otherwise, the V-model is a sequential model.
- Left side of the V ("constructive branch"): Requirements are becoming more and more **detailed**.
- Right side of the V ("analytical branch for **test and integration activities**"): Each level in the left branch is assigned a corresponding test level in the right arm.

The V-model [Boehm, 1979]

→ Origin



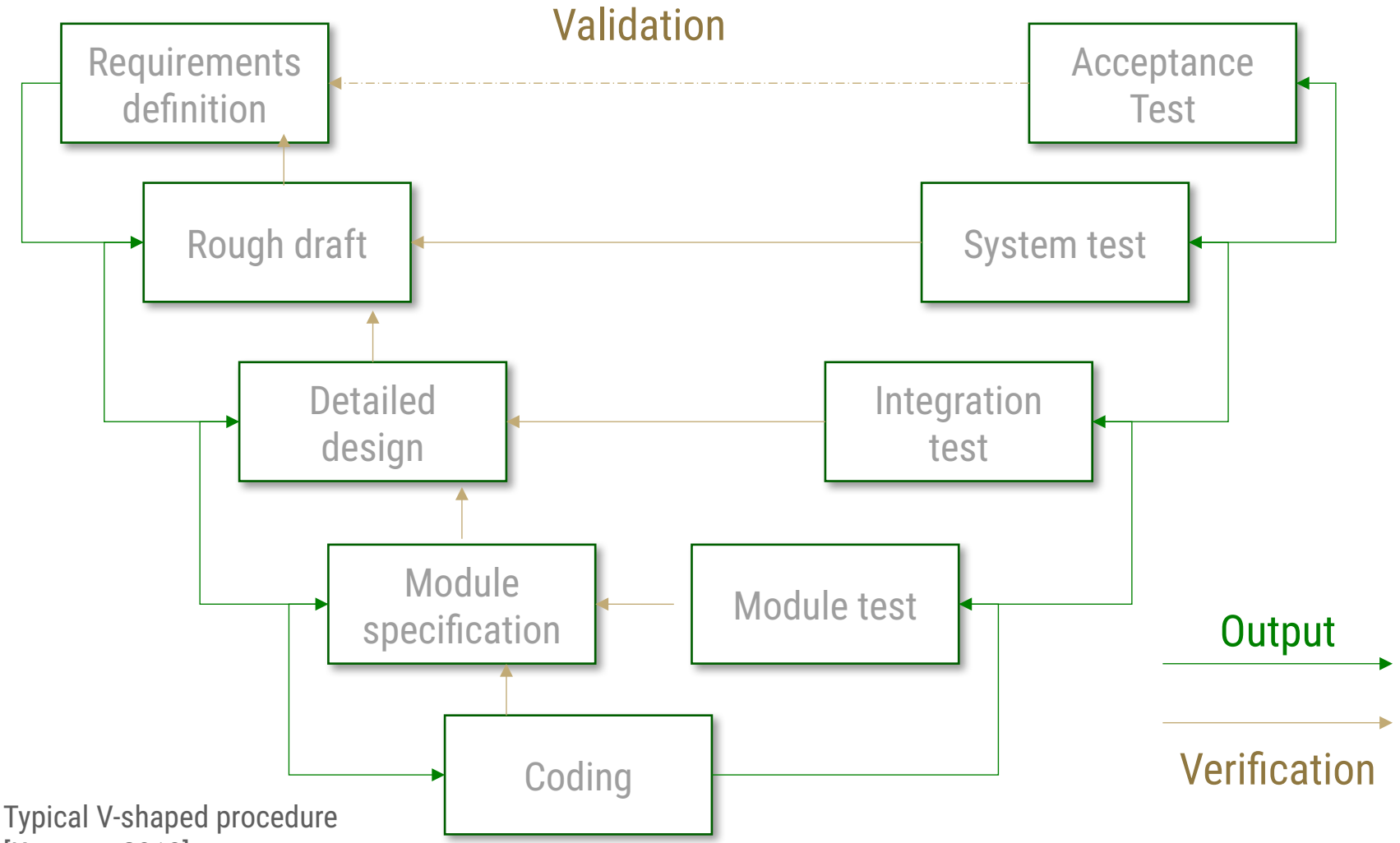
Typical V-shaped procedure [Kneuper, 2018]

Output

Verification

The V-model [Boehm, 1979]

→ Origin



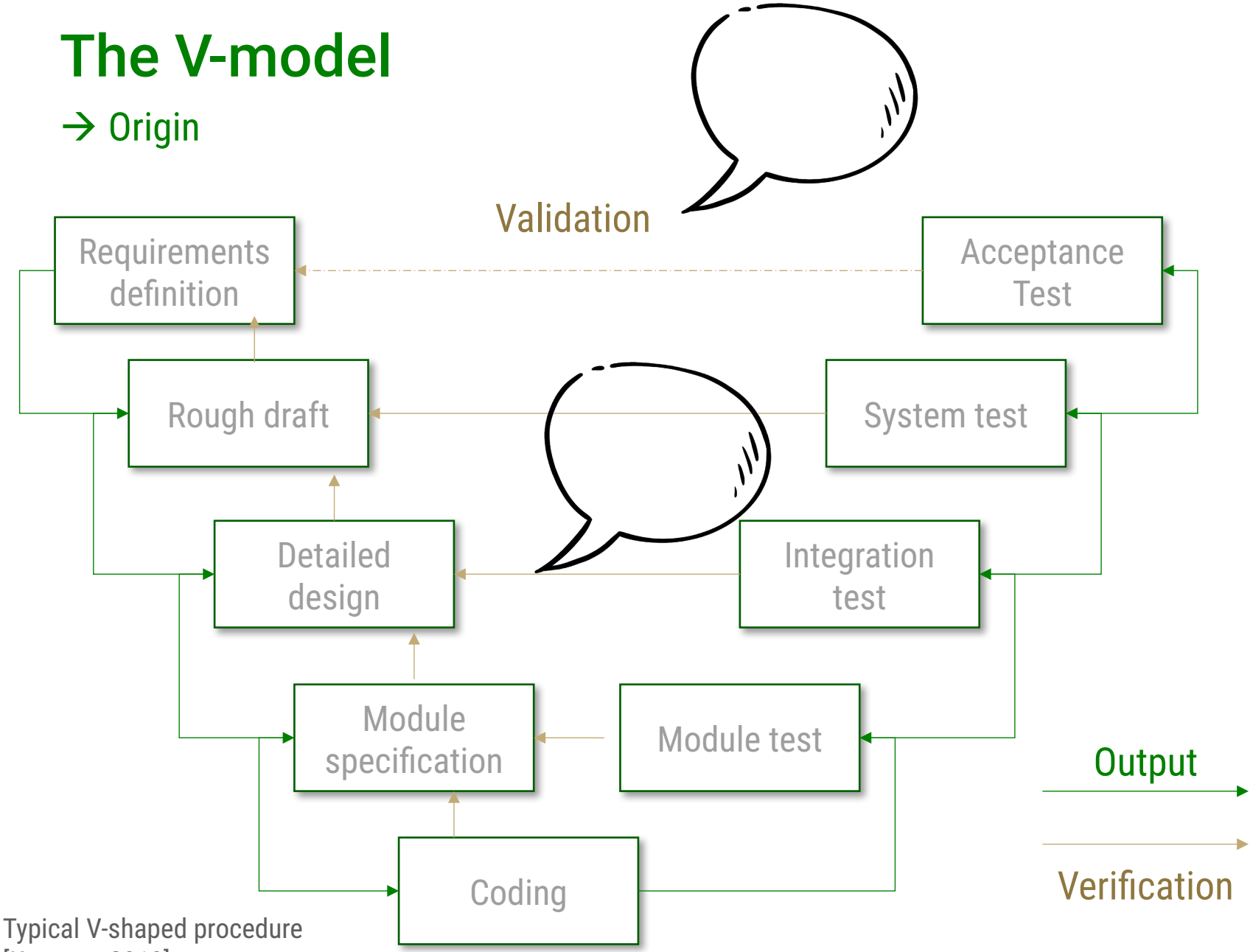
Typical V-shaped procedure [Kneuper, 2018]

Verification: *Confirmation by providing objective evidence that specified requirements have been met.*

Validation: Confirmation by providing objective evidence that the requirements for a specific intended use or application have been met.

The V-model

→ Origin



Typical V-shaped procedure [Kneuper, 2018]

Are we doing the right thing?

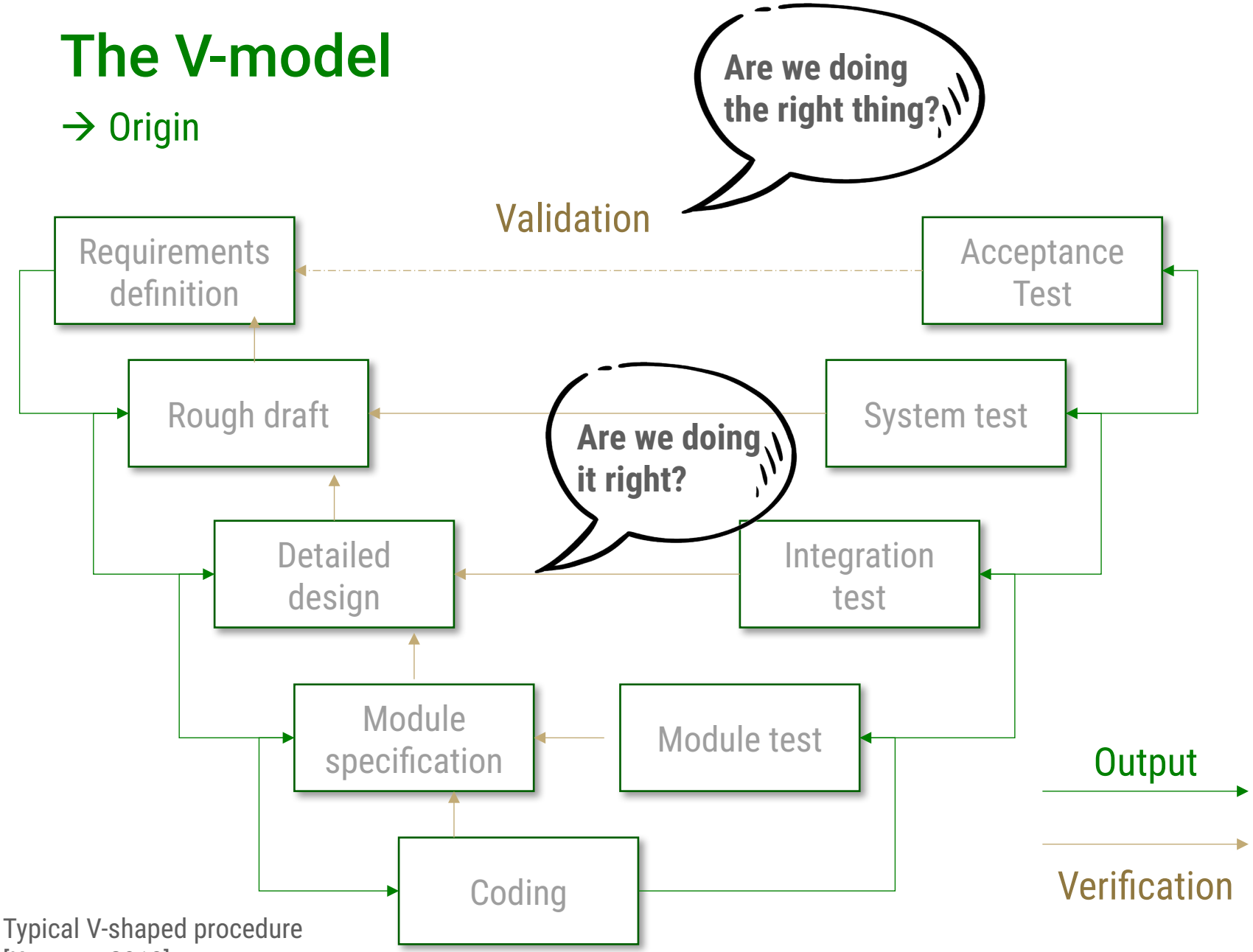
Are we doing it right?

Verification: *Confirmation by providing objective evidence that specified requirements have been met.*

Validation: Confirmation by providing objective evidence that the requirements for a specific intended use or application have been met.

The V-model

→ Origin



Typical V-shaped procedure [Kneuper, 2018]

Verification: *Confirmation by providing objective evidence that specified requirements have been met.*

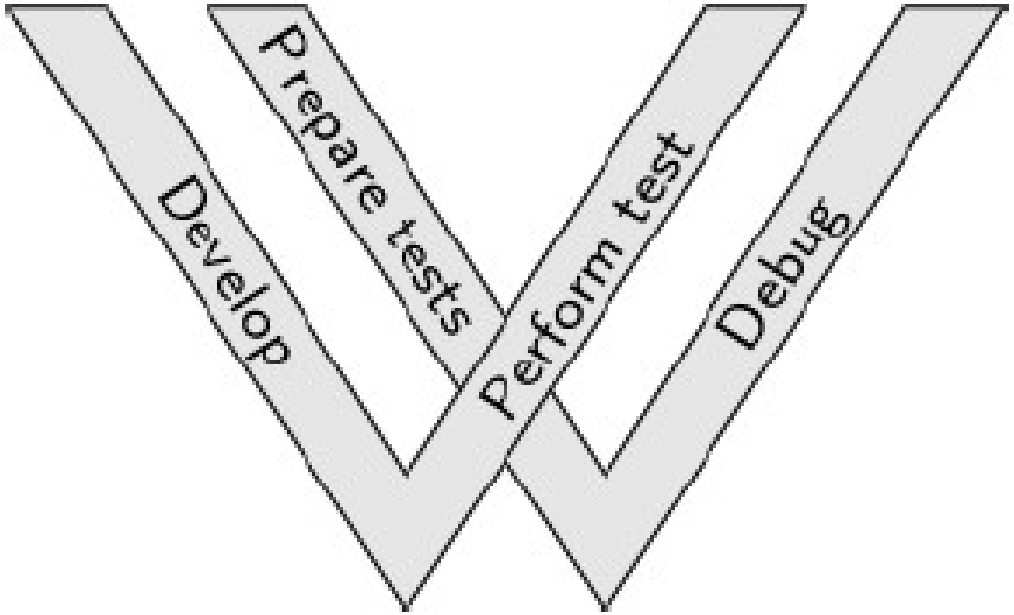
Validation: Confirmation by providing objective evidence that the requirements for a specific intended use or application have been met.

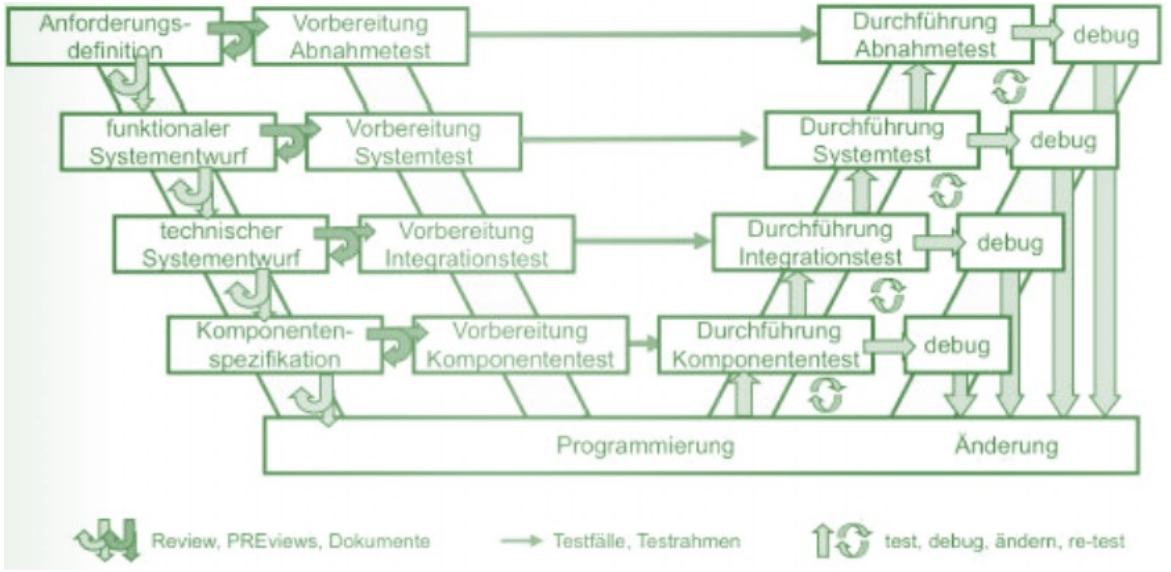
Typical Processes in Software Development

1. Code and Fix
2. Waterfall
3. V-Model (Boehm)
4. W-Model
5. V-Model XT
6. UP
7. XP
8. SCRUM
9. KANBAN

Further development to the W-model [Spillner, 2002]

→ Preparation of test activities parallel to the "constructive branch"





The V-model/W-model

→ Discussion

- **High quality:** How well is quality supported?
- **Satisfied customers:** How well does it ensure usefulness for users?
- **Maintainability:** How well is efficient development and further development supported?
- **Cost/time:** How well is resource efficiency supported?

The V-model/W-model

→ Discussion

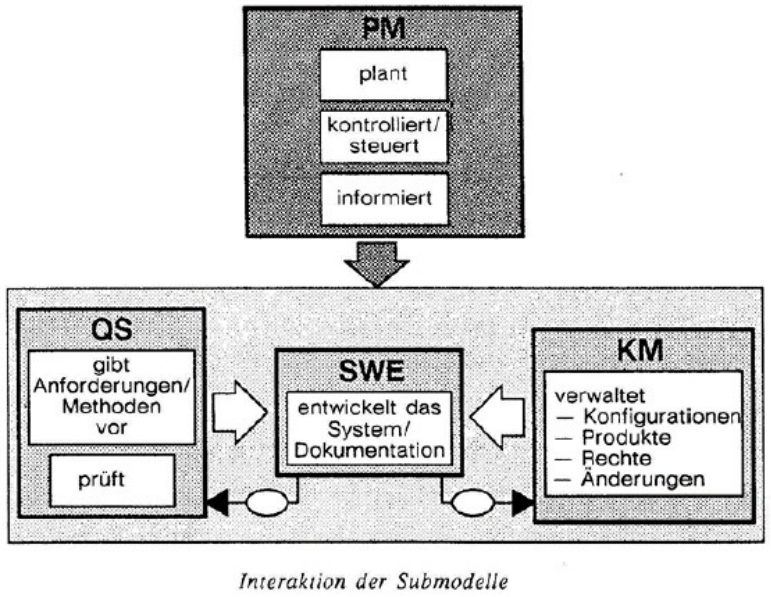
- **High quality:** How well is quality supported?
  - Positive: Support through early QA planning
- **Satisfied customers:** How well does it ensure usefulness for users?
  - As with waterfall, implementation too late, no prototyping, no iterations
- **Maintainability:** How well is efficient development and further development supported?
  - No special features
- **Cost/time:** How well is resource efficiency supported?
  - Early QA planning facilitates appropriate planning of QA efforts and early identification of quality risks

Typical Processes in Software Development

1. Code and Fix
2. Waterfall
3. V-Model (Boehm)
4. W-Model
5. V-Model XT
6. UP
7. XP
8. SCRUM
9. KANBAN

The V-model of the German federal government

→ Origin



Interaktion der Submodelle

Abb. 3: Die Submodelle im Prototyp des V-Modells des Bundes. Quelle: Bundesamt für Wehrtechnik und Beschaffung (1989)

Developed on behalf of the Federal Ministry of Defense and (further developed and) mandatory since 1992.

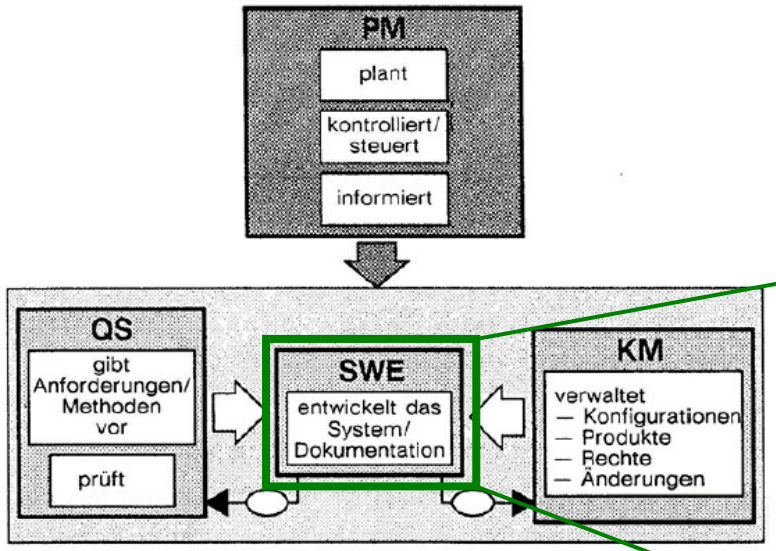
- Later also on behalf of the federal administration and the Federal Ministry of the Interior for the civilian sector

- Three main versions
  - 1991/1992: Software development standard V-model (**V-model 92**)
  - 1997: **V-model 97**
  - 2005: **V-Model XT**

The V-model of the German federal government

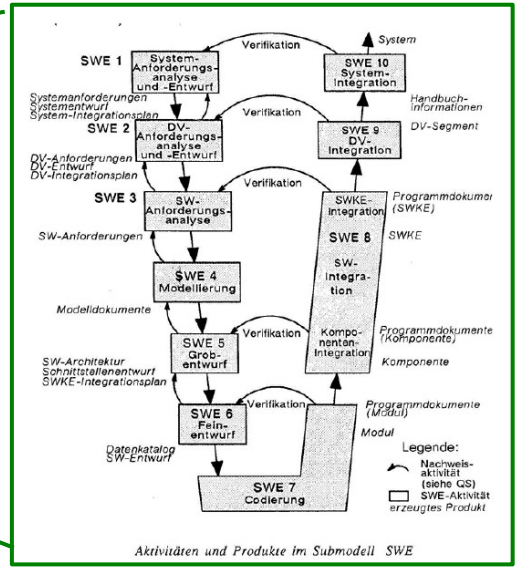
→ V-model 92/97

- First prototype **1989**
- Process of SW development projects from the **contractor**'s perspective
- Includes submodels, whereby the **submodel "SWE"** was described **in a V-shape** and contributed to the name.



Interaktion der Submodelle

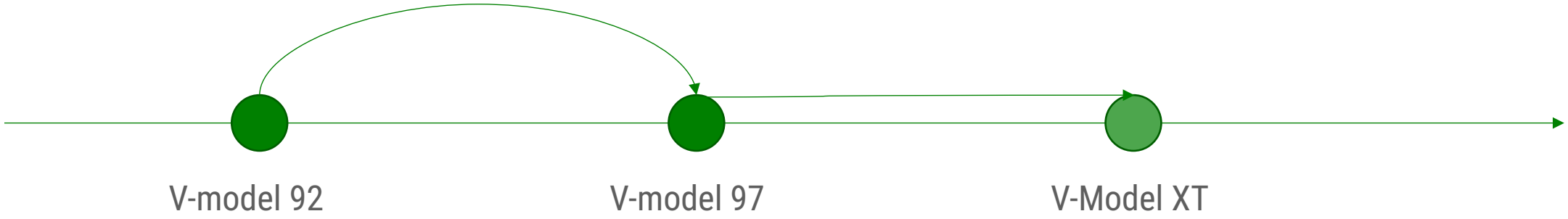
Abb. 3: Die Submodelle im Prototyp des V-Modells des Bundes. Quelle: Bundesamt für Wehrtechnik und Beschaffung (1989)



Aktivitäten und Produkte im Submodell SWE

The V-model of the German federal government

→ Changes over time



- System and hardware development
- Flexibilization of procedure

- No more submodels
- Developed by a consortium of universities and industry
- XT: eXtreme **Tailoring** allows the model to be adapted to specific framework conditions

The V-model XT

Current version, 350 pages

- First publication **2004**
- Development standard for federal IT projects: **V-Modell XT Bund**(Version 2.3., 350 pages, **2020**)
  - http://www.cio.bund.de/Web/DE/Architekturen-und-Standards/V-Modell-XT-Bund/vmodellxt_bund_node.html

The V-model XT

→ Overview of documents

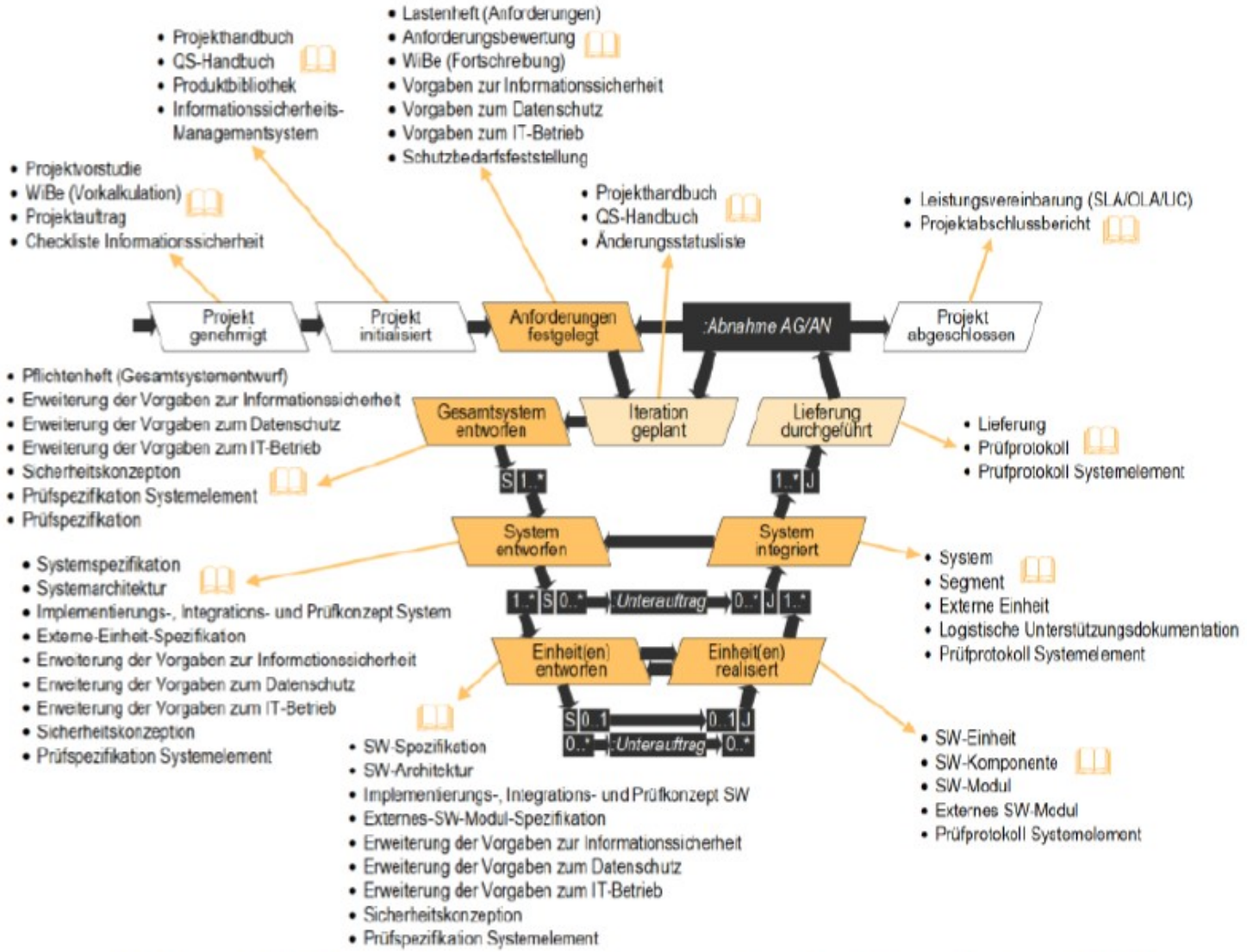


Abbildung 19: Überblick über Entscheidungspunkte und spezifische Produkte aus Sicht eines AG/AN

The V-model XT

→ Discussion

- **High quality:** How well is quality supported?
  - Early QA planning as with V-model
- **Satisfied customers:** How well does it ensure usefulness for users?
  - Depends on implementation strategy (e.g. agile, incremental)
- **Maintainability:** How well is efficient development and further development supported?
  - Can become very bureaucratic if necessary
- **Cost/time:** How well is resource efficiency supported?
  - Very bureaucratic, document-heavy
  - Customizability for project possible, possibly complex

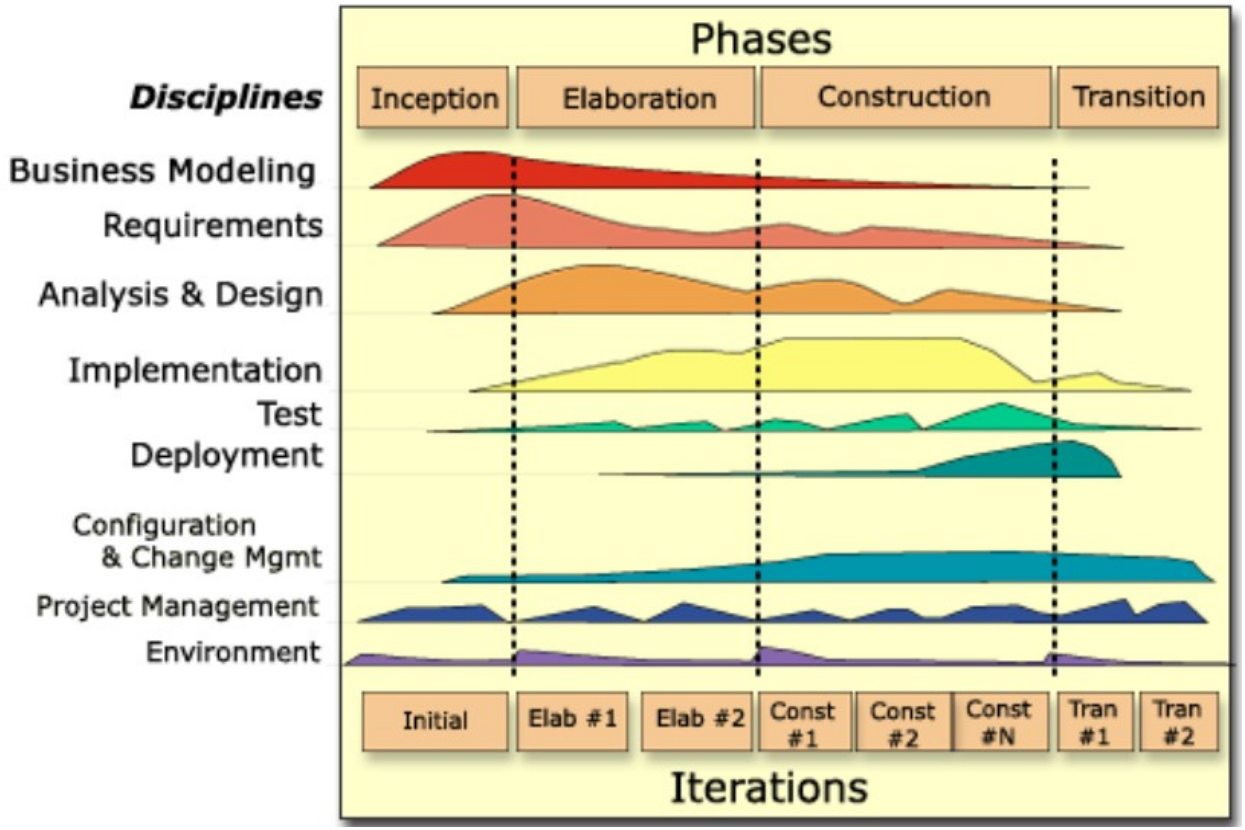
Typical Processes in Software Development

1. Code and Fix
2. Waterfall
3. V-Model (Boehm)
4. W-Model
5. V-Model XT
6. UP
7. XP
8. SCRUM
9. KANBAN

UP - the Unified Process

→ Background and definition

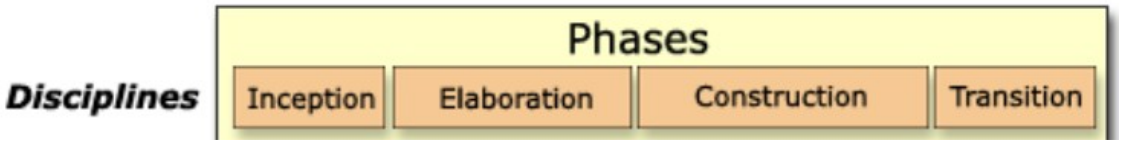
- OOP became popular towards the end of the 80s
- Realization that previous models did not fit the new paradigm
- Published after further development stages in 1999
  [Jakobson, Booch, Rumbaugh, 1999]
- RUPP is a version of the generic UP developed by IBM.



[Jakobson, Booch, Rumbaugh, 1999]

UP - the Unified Process

→ Basic concepts: 1) **Phases**

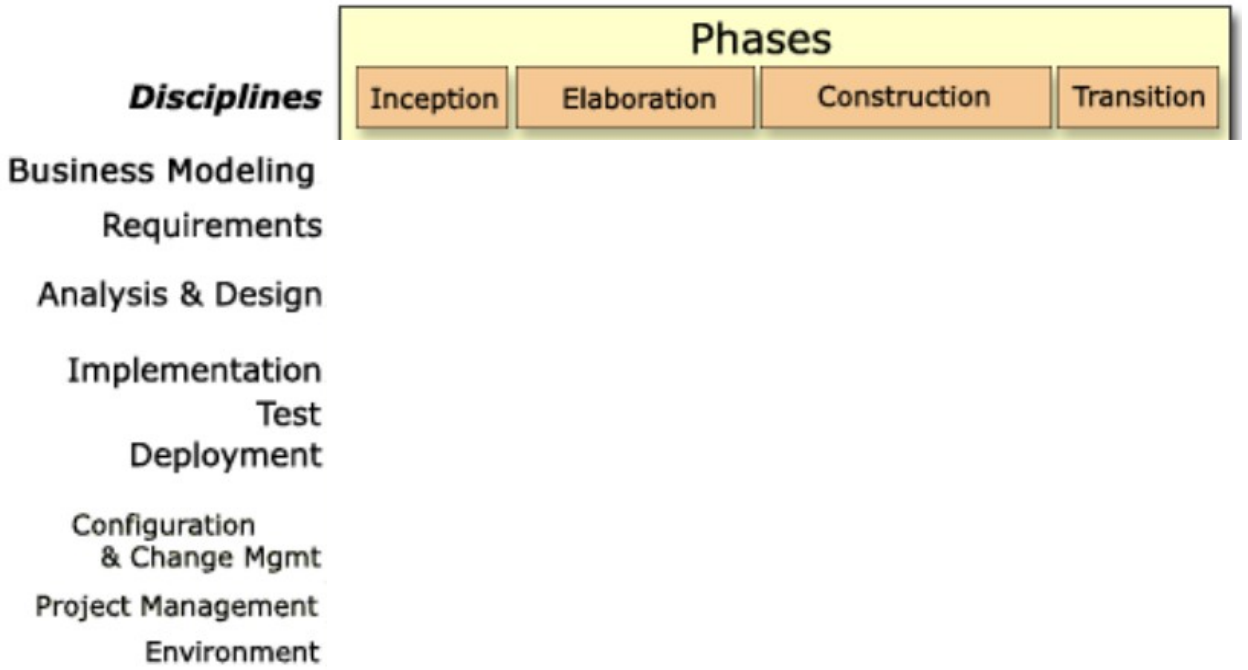


- Inception
  - Product idea, feasibility, understanding rough requirements, business case.
- Elaboration
  - Refined vision, Missing requirements, Core architecture, Effort
- Construction
  - Implementation, integration, testing
- Transition
  - Delivery, piloting, stabilization.

[Jakobson, Booch, Rumbaugh, 1999]

UP - the Unified Process

→ Basic concepts: 2) **Workflows**

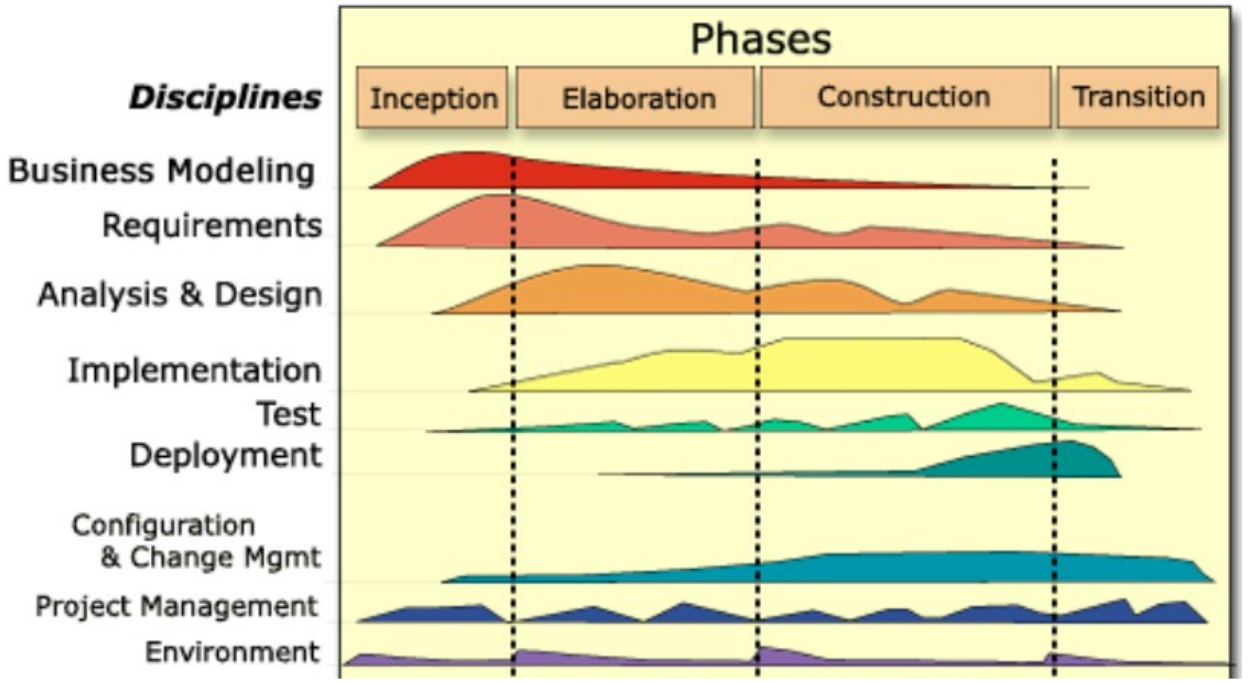


- Combine related activities
- They are carried out in ALL phases, only with varying intensity

[Jakobson, Booch, Rumbaugh, 1999]

UP - the Unified Process

→ Basic concepts: 2) **Workflows**

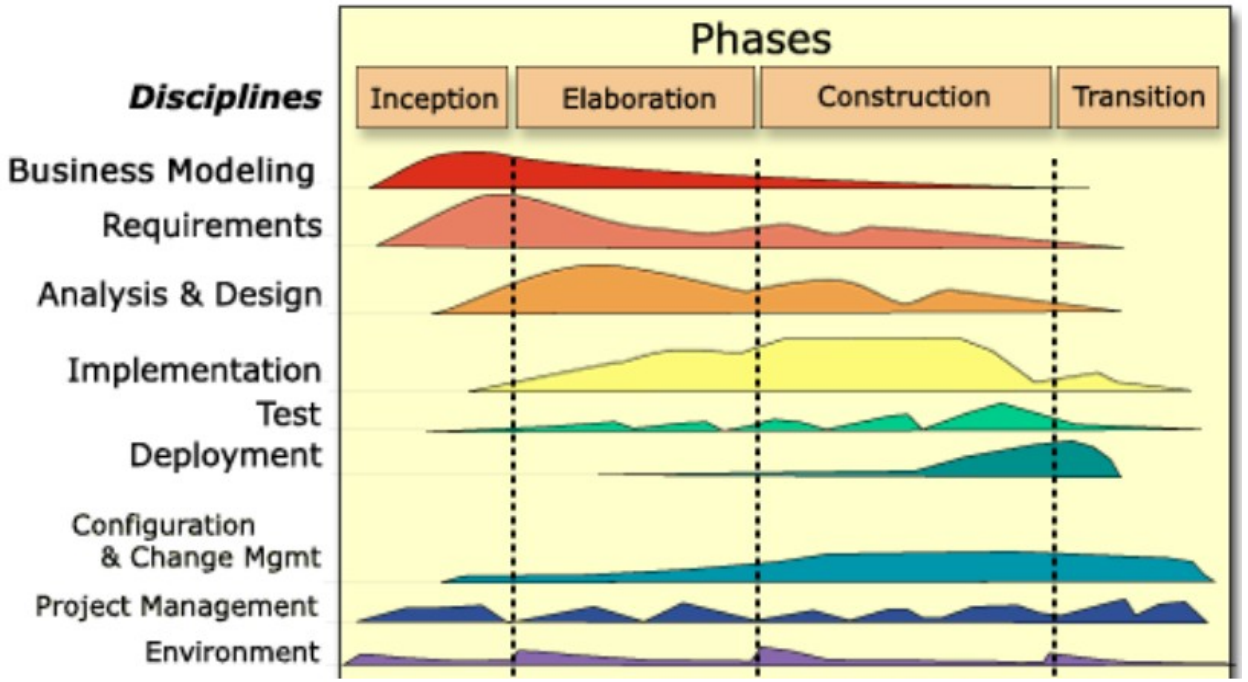


- Combine related activities
- They are carried out in ALL phases, only with varying intensity

[Jakobson, Booch, Rumbaugh, 1999]

UP - the Unified Process

→ Basic concepts: 3) **Iterations**

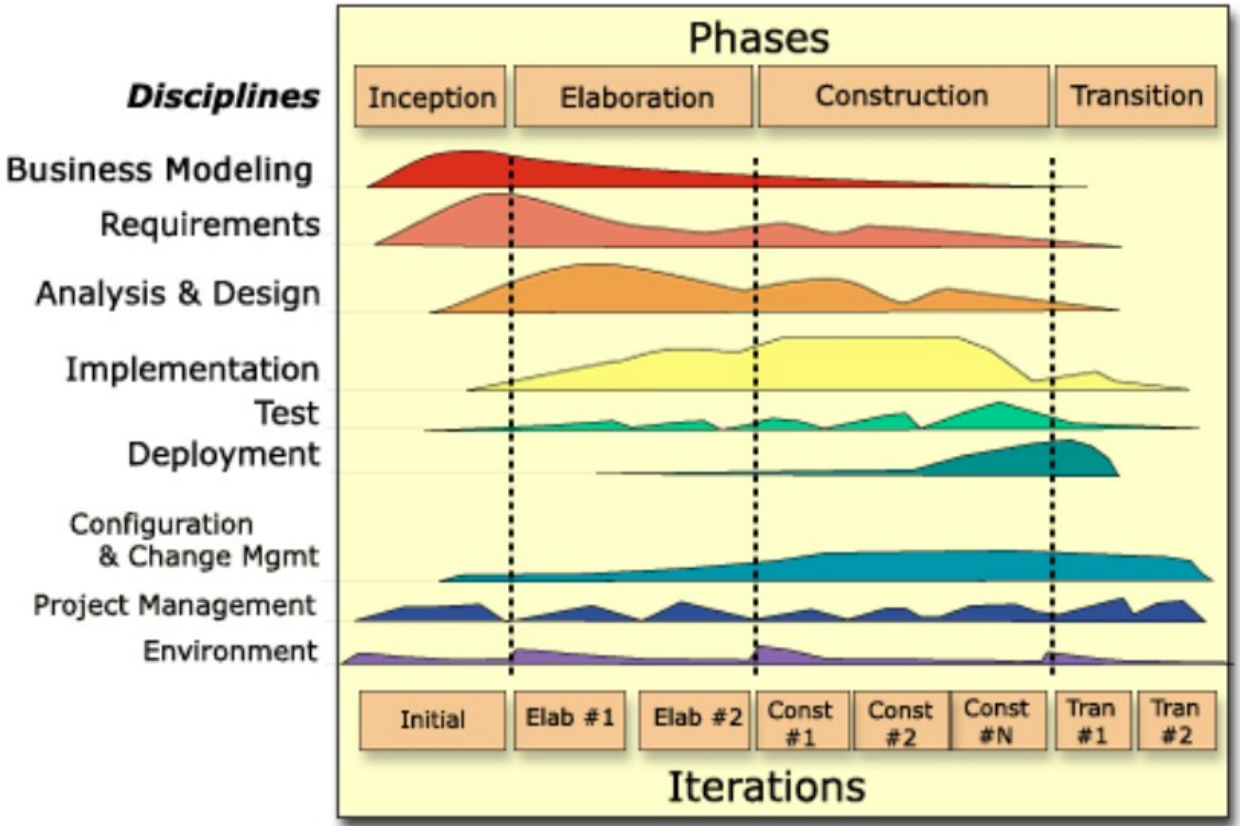


- Each phase can be subdivided into several iterations.
- In each phase, the artifacts relevant to the phase are developed incrementally.
- Each phase is concluded with an internal milestone.

[Jakobson, Booch, Rumbaugh, 1999]

UP - the Unified Process

→ Basic concepts: 3) **Iterations**

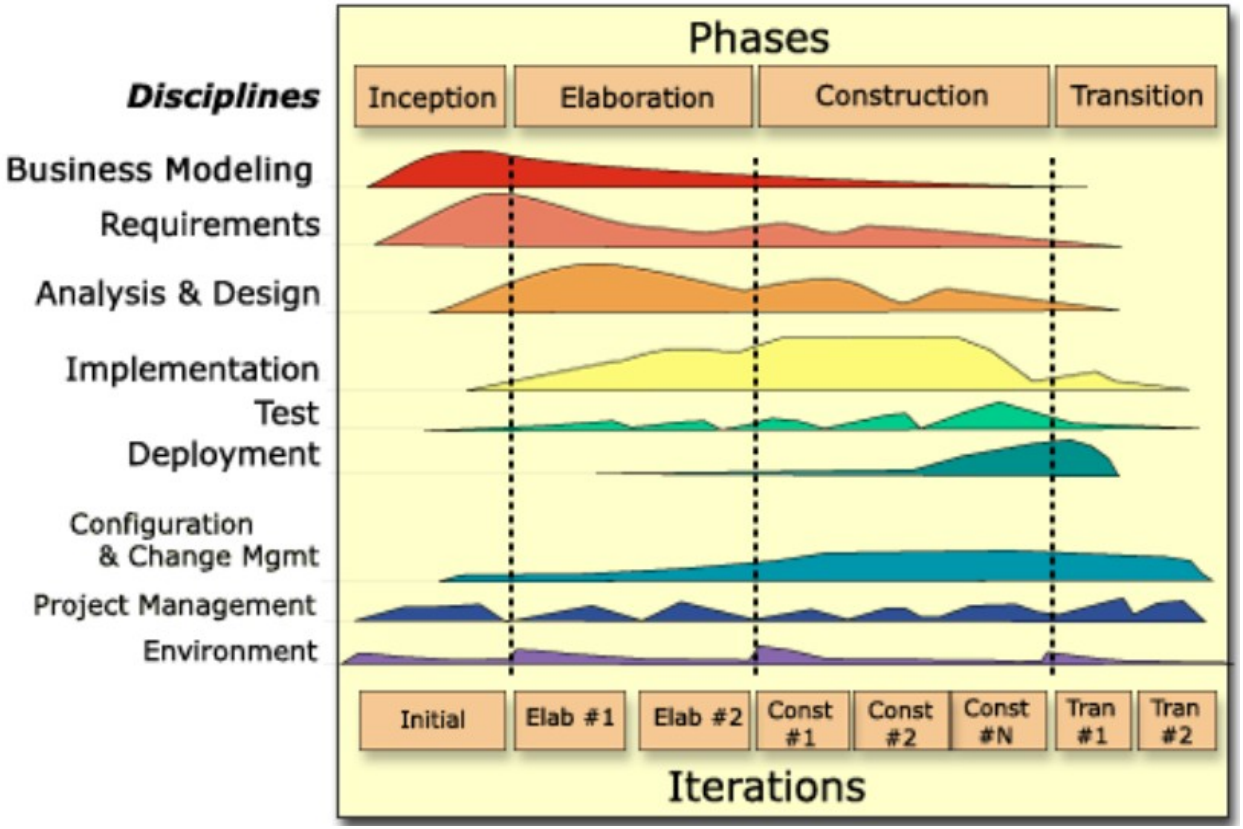


- Each phase can be subdivided into several iterations.
- In each phase, the artifacts relevant to the phase are developed incrementally.
- Each phase is concluded with an internal milestone.

[Jakobson, Booch, Rumbaugh, 1999]

UP - the Unified Process

→ Basic concepts: 4) **Increments**



- If a product is to be delivered in expansion stages → Development cycle is run through several times
- Prototyping possible

[Jakobson, Booch, Rumbaugh, 1999]

UP - the Unified Process

→ Discussion

- **High quality:** How well is quality supported?
- **Satisfied customers:** How well does it ensure usefulness for users?
- **Maintainability:** How well is efficient development and further development supported?
- **Cost/time:** How well is resource efficiency supported?

UP - the Unified Process

→ Discussion

- **High quality:** How well is quality supported?
  - QA (even if with varying intensity) is part of EVERY phase
- **Satisfied customers:** How well does it ensure usefulness for users?
  - Supports prototyping
- **Maintainability:** How well is efficient development and further development supported?
  - Focus on architecture, modeling, OOA, OOP
- **Cost/time:** How well is resource efficiency supported?
  - Iterative development is explicitly supported

Typical Processes in Software Development

Agile!

1. Code and Fix
2. Waterfall
3. V-Model (Boehm)
4. W-Model
5. V-Model XT
6. UP
7. XP
8. SCRUM
9. KANBAN

https://agilemanifesto.org/iso/de/manifesto.html

Manifesto for Agile Software Development

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.

XP

eXtreme Programming

- **Most influential work that has changed the culture of software development is xP**
- The book by **Kent Beck** is published in **1999**

Why "extreme"?

XP

eXtreme Programming

- **Most influential work that has changed the culture of software development is xP**
- The book by **Kent Beck** is published in **1999**
- "extreme" because previous good practices (e.g. iterative software development) have been taken to the extreme.
- Example: regular integration was known as good practice to find errors quickly. XP suggests integrating several times a day.

XP

→ Practices

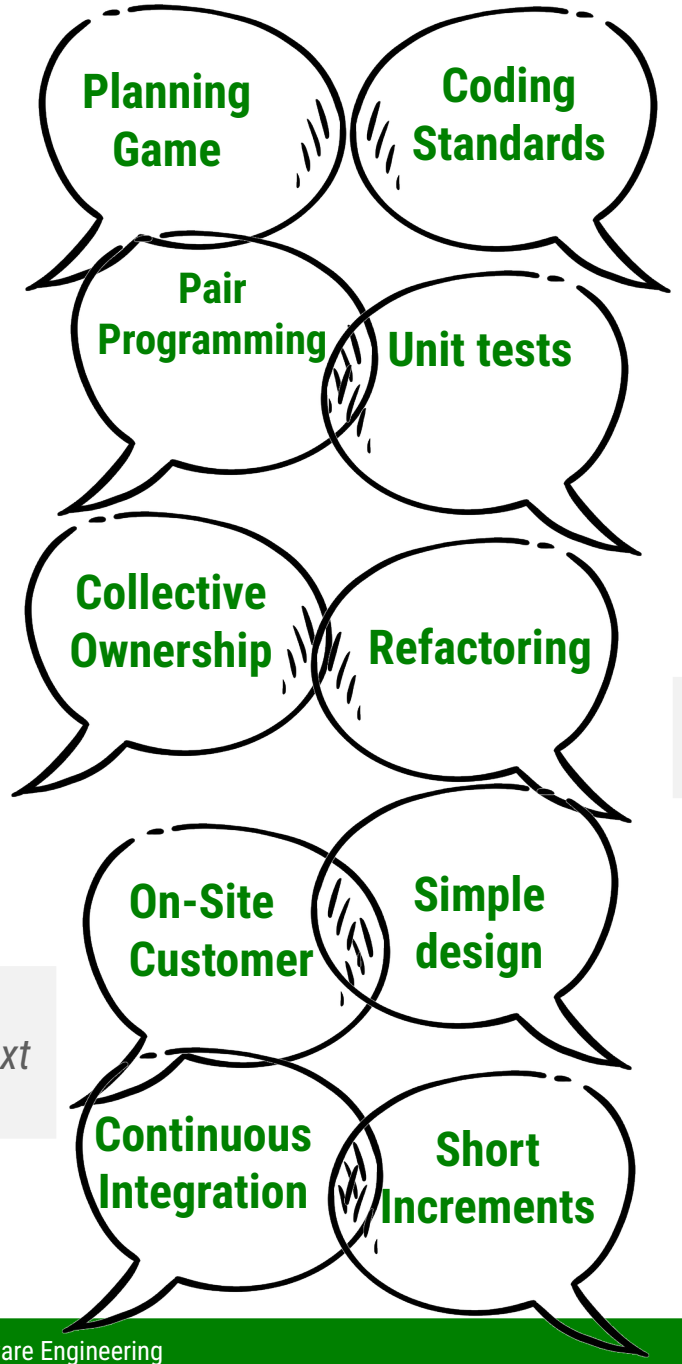
Code belongs to the entire development team, everyone is responsible for the code. The teams rotate cyclically.

Code produced by different team members is merged very often.

Competent representatives from the customer side are with the developers throughout the entire development period.

Consultation between customer and developer to determine the scope of the next increment.

Two developers work together on one piece of code at a time



The code is restructured as early as possible and on a regular basis.

No unnecessary features are implemented.

Frequent incremental delivery of the system with maximum value for the user.

The smallest units are tested automatically.

Strict adherence to coding guidelines.

XP

→ Practices

Collective Ownership

Code belongs to the entire development team, everyone is responsible for the code. The teams rotate cyclically.

Continuous Integration

Code produced by different team members is merged very often.

On-Site Customer

Competent representatives from the customer side are with the developers throughout the entire development period.

Planning Game

Consultation between customer and developer to determine the scope of the next increment.

Pair Programming

Two developers work together on one piece of code at a time

Refactoring

The code is restructured as early as possible and on a regular basis.

Simple design

No unnecessary features are implemented.

Short Increments

Frequent incremental delivery of the system with maximum value for the user.

Unit tests

The smallest units are tested automatically.

Coding Standards

Strict adherence to coding guidelines.

XP

→ Practices → Pair programming in detail

- In pair programming, two developers sit at a computer and work together on a task.
- The two developers take on different roles:
  - **Pilot**: writes
  - **Navigator**: thinks along, "navigates", checks, thinks about alternative solutions
- The roles change regularly, e.g. every 15-30 minutes.
- Advantages: Knowledge exchange, quality is increased (dual control principle), skill building, errors are recognized earlier, etc.

Literature

- [Ludewig 2013] Ludewig, Lichter: Software Engineering. Basics, people, processes, techniques, dpunkt.verlag.
- [Paech 2021] Barbara Paech: Lecture Software Engineering, Uni-Heidelberg.

Typical Processes in Software Development

Agile!

1. Code and Fix
2. Waterfall
3. V-Model (Boehm)
4. W-Model
5. V-Model XT
6. UP
7. XP
8. SCRUM
9. KANBAN

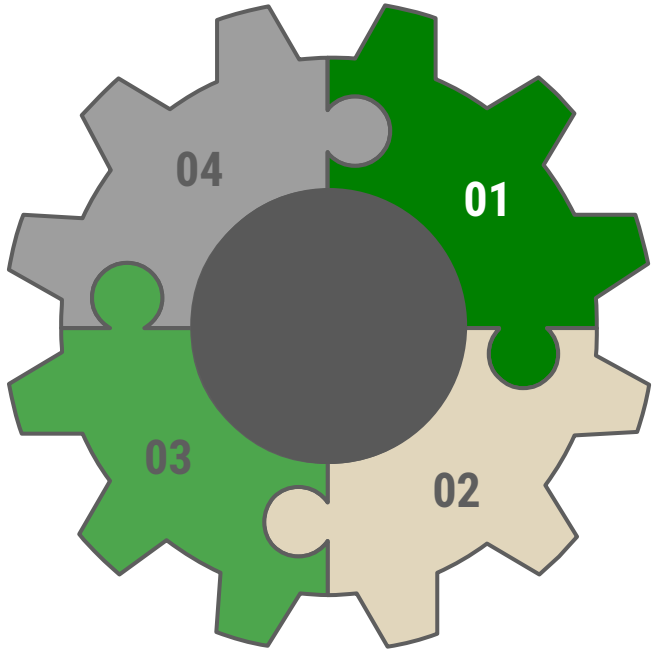
Learning Objectives

- You can explain the principles of empirical process control using the example of SCRUM
- You know roles, artifacts and events in the SCRUM process
- You can explain the places in the SCRUM process at which the values of SCRUM are particularly important
- You can compare planning in a classic and agile context

SCRUM

04
Value-based
Cooperation is not based on sophisticated processes but on **values**

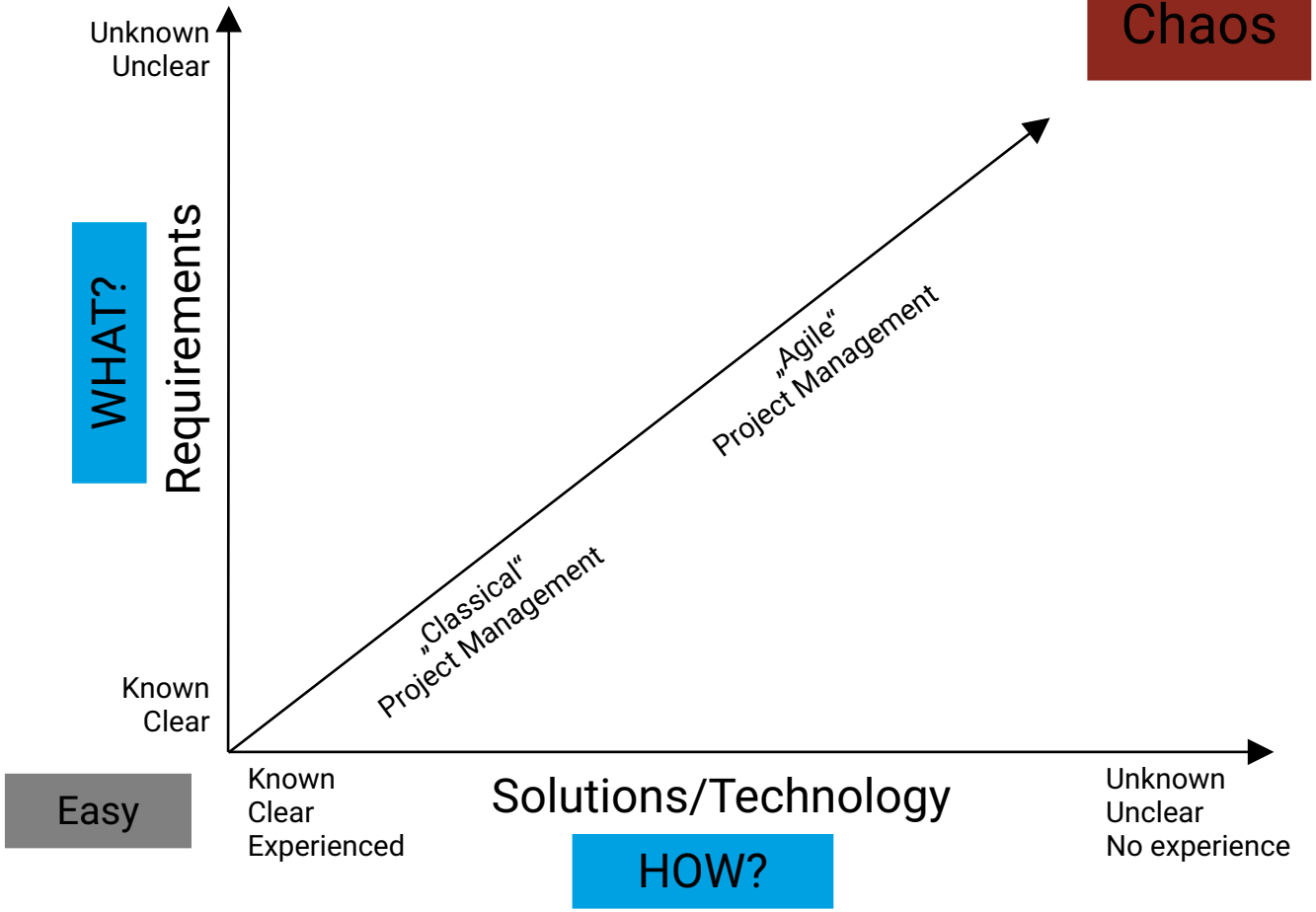
03
Lightweight
Only requires a minimal set of rules.

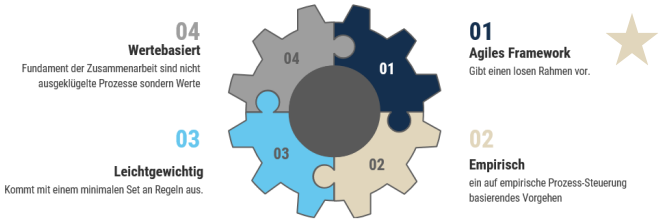


01
Agile framework
Provides a loose framework.

02
Empirical
an approach based on empirical process control

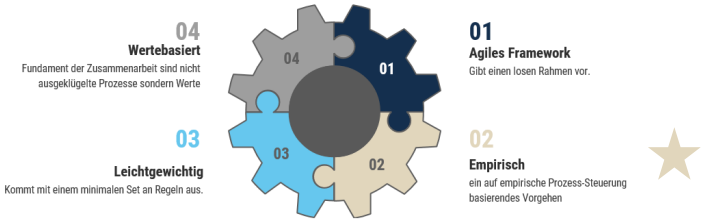
SCRUM is an agile framework



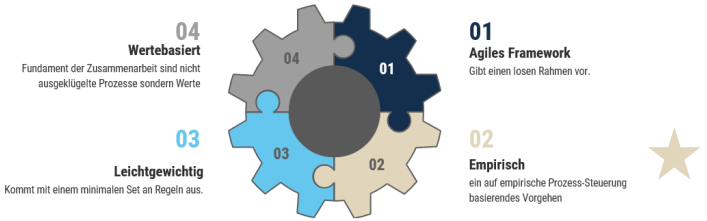


Check-In

What goes through your mind when you read the manifesto?



- Three fundamental concepts
  - Transparency
  - Inspection
  - Adaptation (adapt)
- For successful completion of complex projects, the current status of the solution must be
  - **transparent** at all times
  - developed and inspected in small, regular steps
  - able to be adapted in small, regular steps.

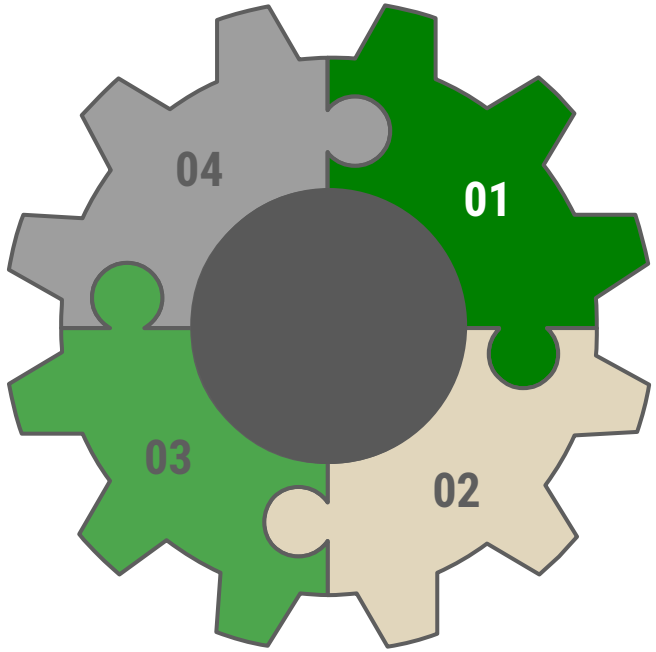


- Transparency
  - Results are visible (and therefore available)
  - **Timeboxing** (the (interim) results available at regular, closely timed points in time)
- Inspection
  - **Acceptance criteria** for the acceptance of an (interim) result
  - Status is not regularly "queried" and "reported upwards" but checked daily
- Adaptation (adapt)
  - Continuous **learning** and **adaptation**

SCRUM

04
Based on values
Cooperation is not based on sophisticated processes but on values

★ **03**
Lightweight
Only requires a minimal set of rules.

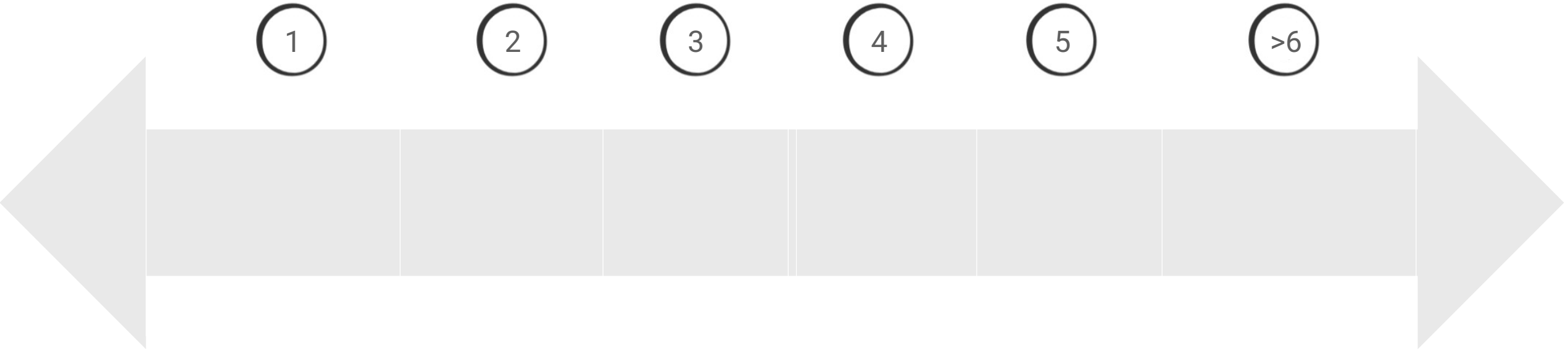


01
Agile framework
Provides a loose framework.

02
Empirical
an approach based on empirical process control

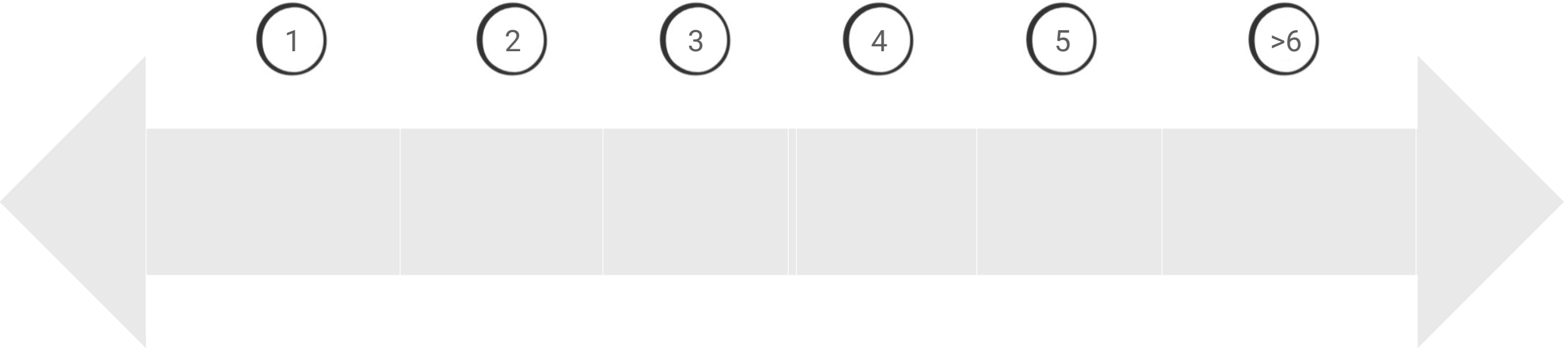
How many roles are there in SCRUM?

* according to the SCRUM-GUIDE



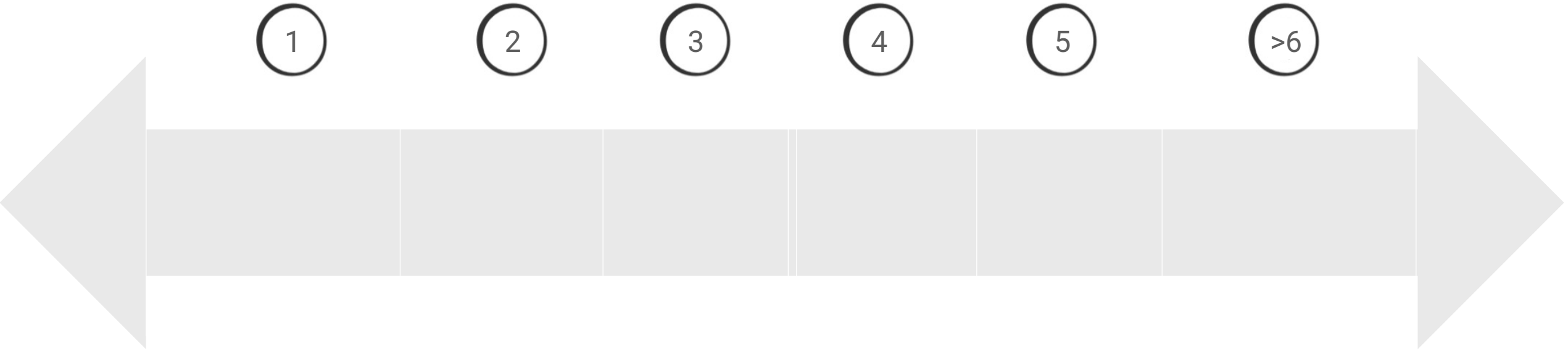
How many artifacts are there in SCRUM?

* according to the SCRUM-GUIDE



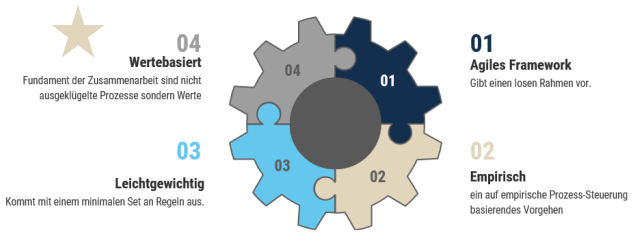
How many events are there in SCRUM?

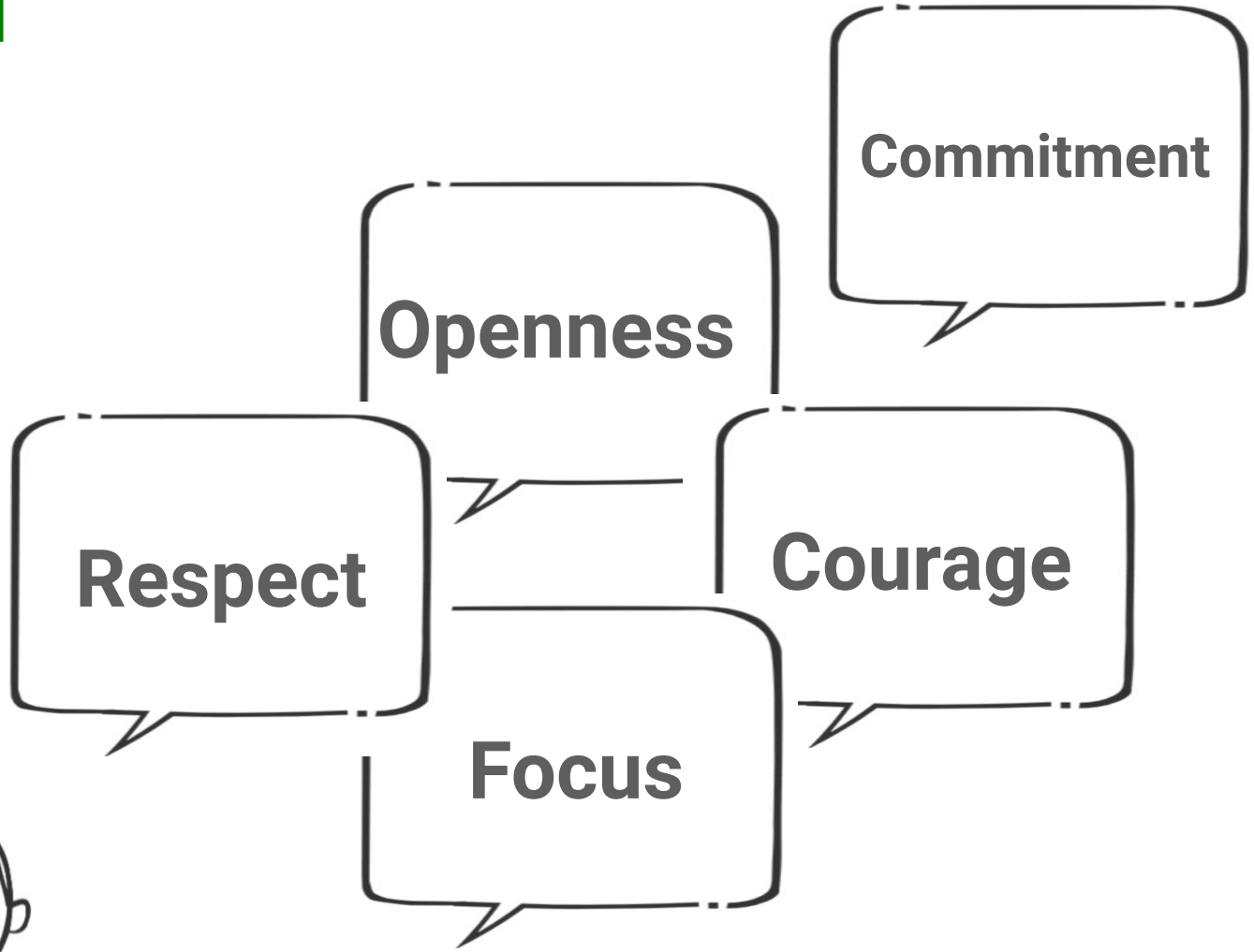
* according to the SCRUM-GUIDE

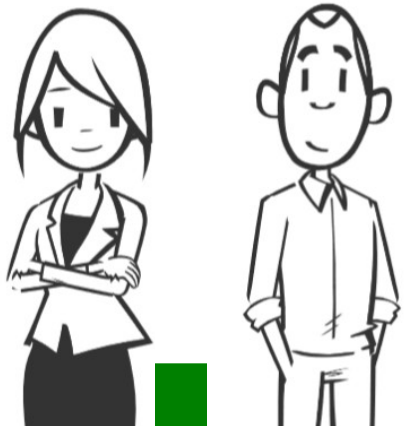


- Example "Pocket calculator"
  - Typically, you get faster from sprint to sprint. Why?
  - When does it not happen? Or: What prevents you from getting faster?

The values of SCRUM

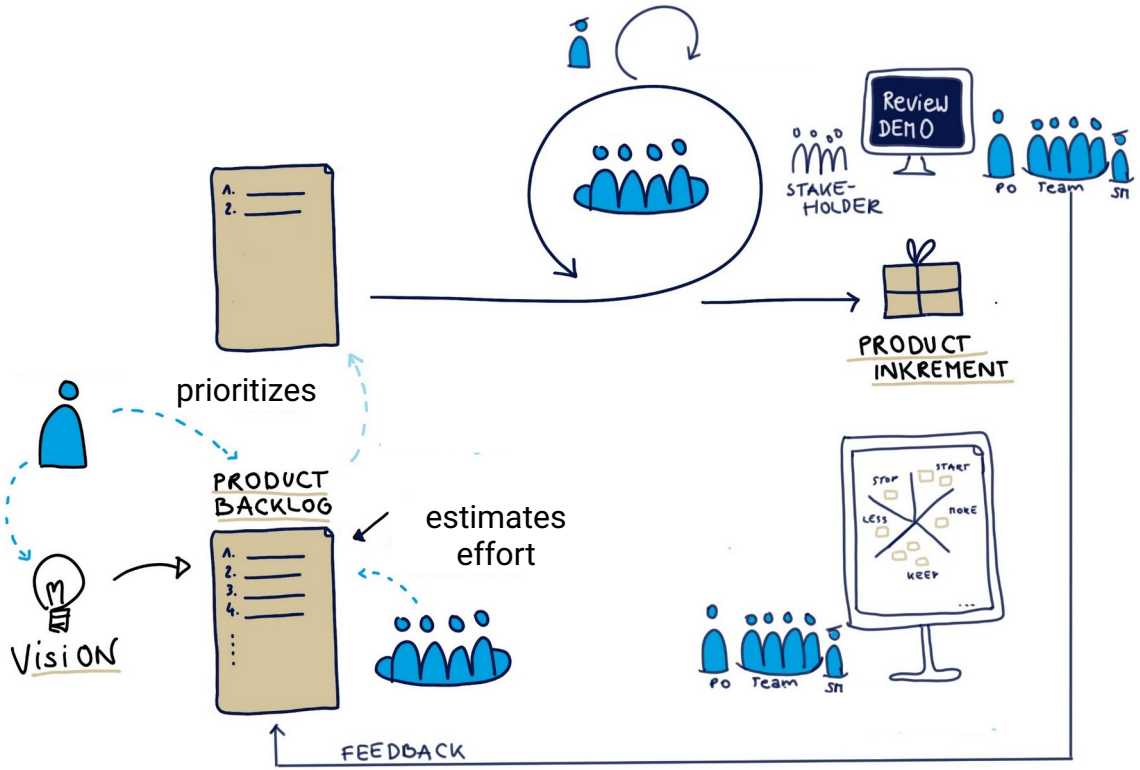






SCRUM

SCRUM in a nutshell



Rolls
→Product Owner

- Responsible for the **product** (product vision, product strategy)
- "Product visionary"
- **Responsible for the budget**, responsible for the economic success of a product
- Identifies all stakeholder **requirements**
- Defines and **prioritizes** requirements, which are recorded in a **product backlog**.
- Creates a **release plan**
- **Reduces** sprint results and the end product

- All persons involved in the development of the product (whether software, explanatory video, training, service)
- Shares the tasks in order to implement the requirements (in the given priority)
- Breaks down the requirements into so-called "tasks"
- Commits to implementing a set of requirements by the end of the sprint
- Interdisciplinary composition
- **Self-responsible and self-organized**

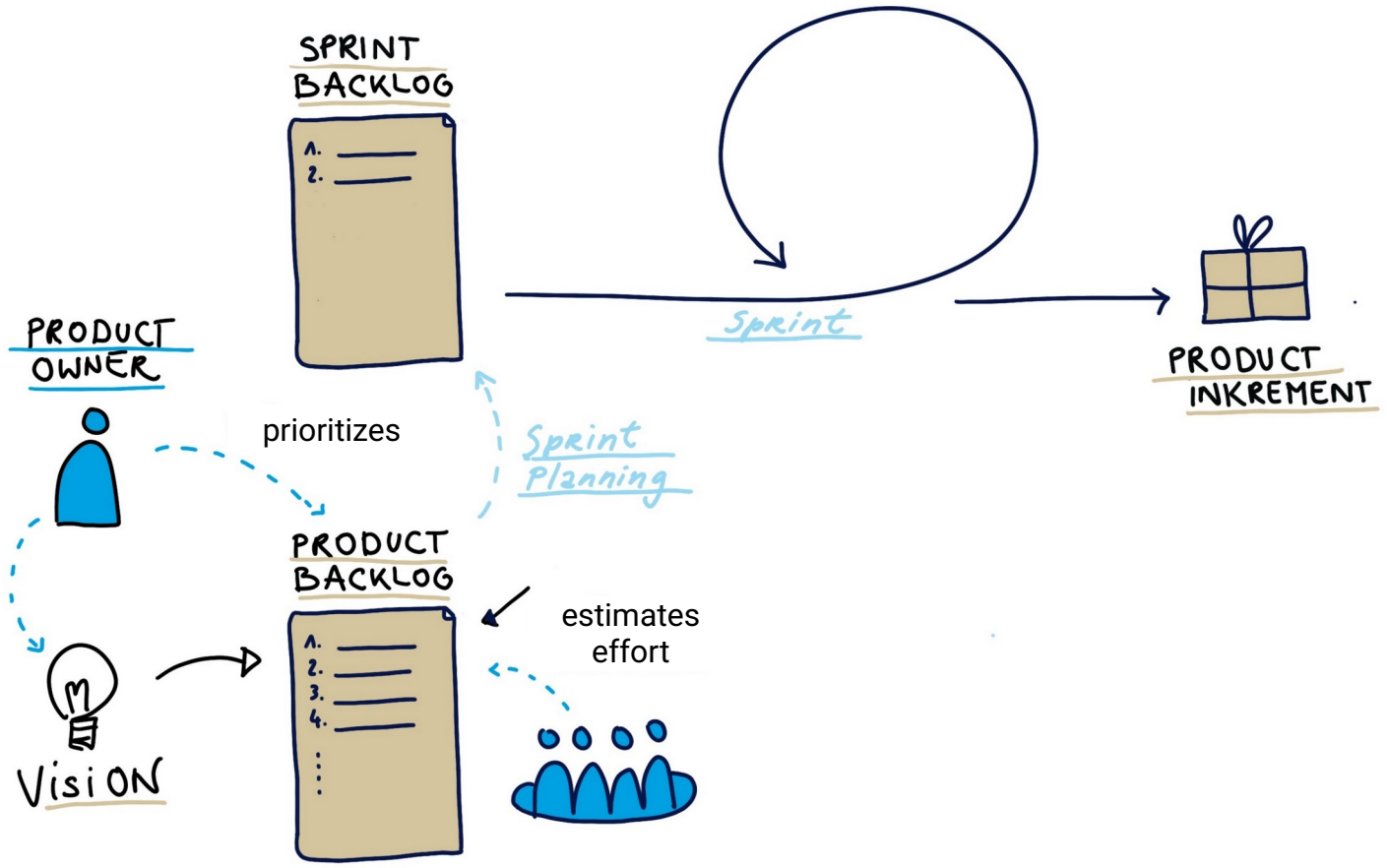
- Responsible for the process
  - Responsible for the correct implementation of the Scrum process
  - Values, rules, principles
- Coaching role
- Protects the team, removes obstacles (**impediments**)
- Product Owner and Scrum Master NEVER in personal union

The SCRUM process
→ How a product is created

PRODUCT BACKLOG

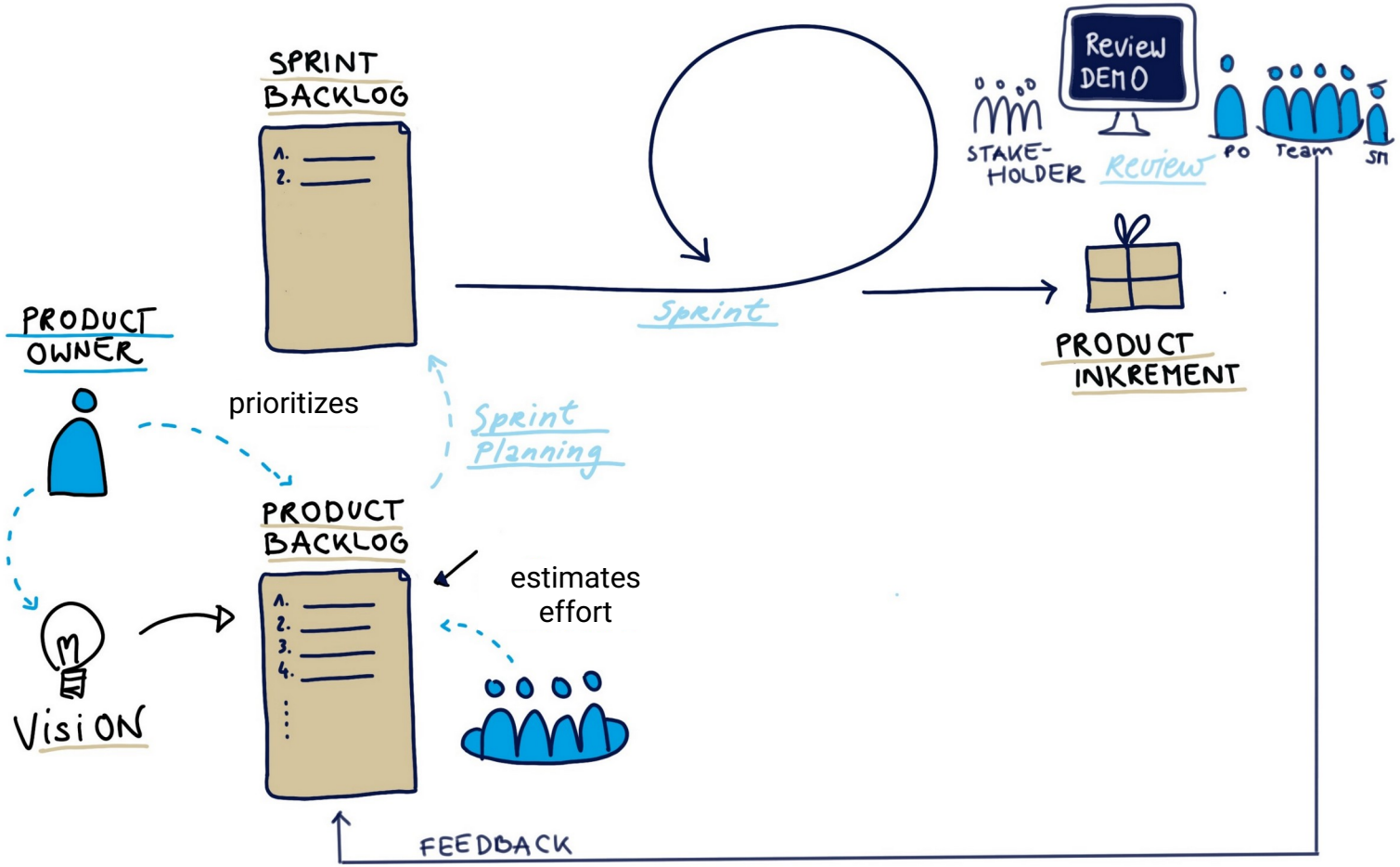
- Contains requirements for the product
- Is prioritized by the product owner
- And appreciated by the team
- Scrum does not specify how the requirements should be described
- Describing the backlog in the form of user stories has proven successful

The SCRUM process
→ How a product is created



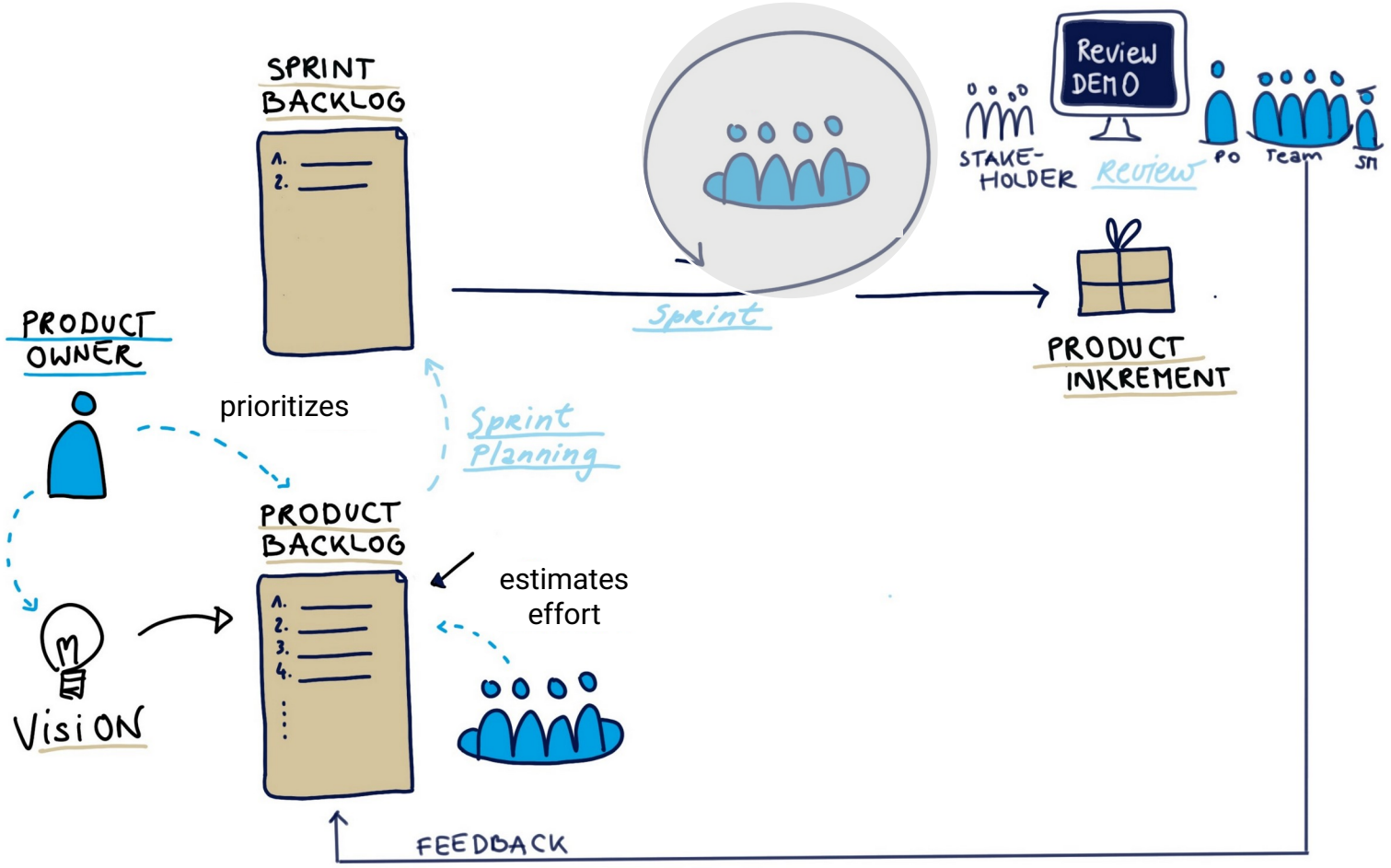
- Product development takes place in iterations, in "sprints"
  - Sprints have the same length
  - Maximum 4 weeks
  - Typically 2 weeks
- Content of the sprint is defined in *Sprint Planning*
- The high-priority elements are taken from the backlog
- Team commits to implementation within the sprint
- The result is a finished product increment

The SCRUM process
→ How a product is created



- Interim results are shown in the sprint review (demo), participants can be all stakeholders
- The finished increment is approved by the product owner

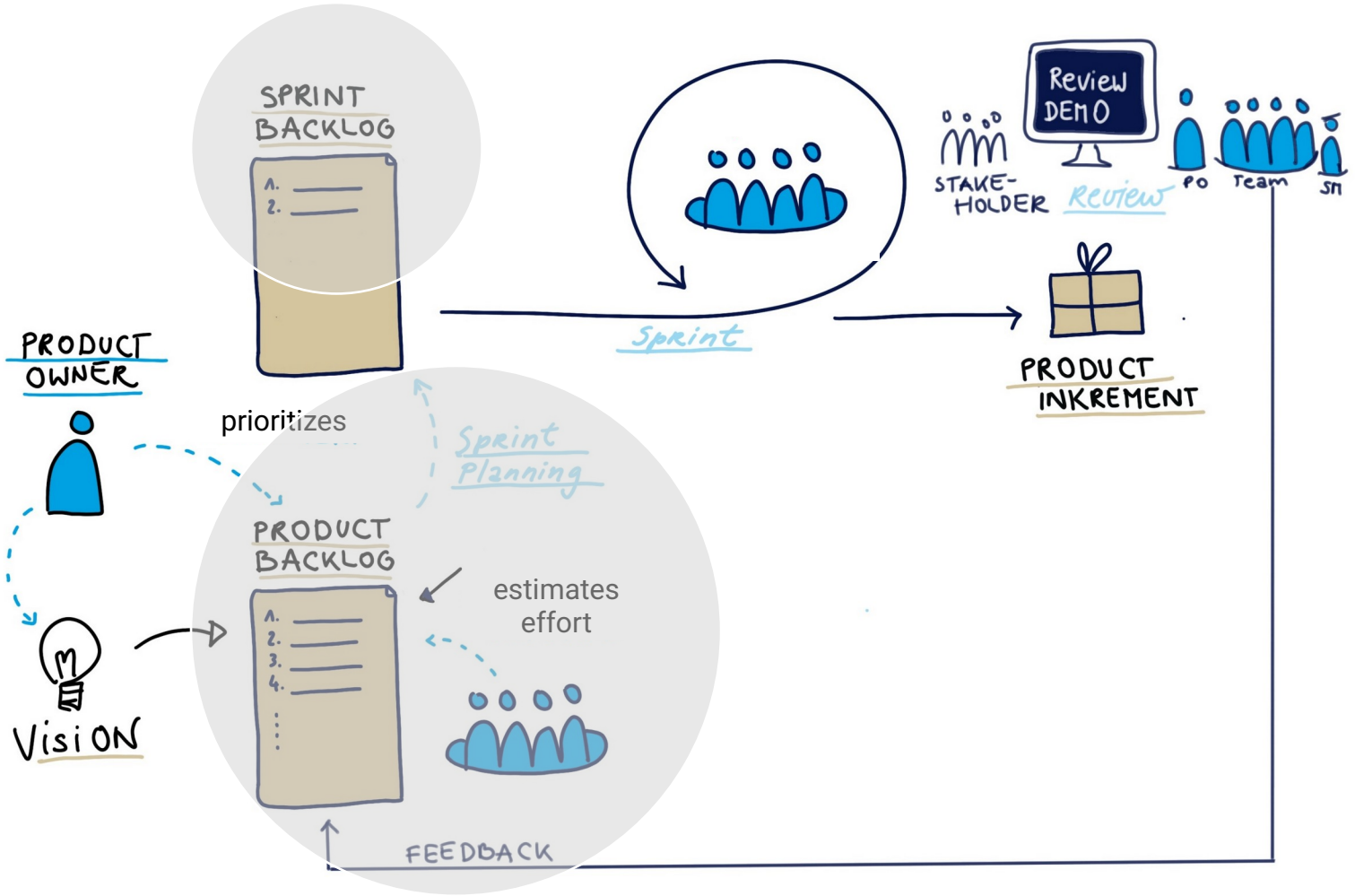
The SCRUM process
→ Product development in a sprint



In the sprint...

- The team consists of 3 - max. 10 interdisciplinary members
- Has all the skills to create a product increment
- Organizes itself
- No prominent roles/positions/hierarchies

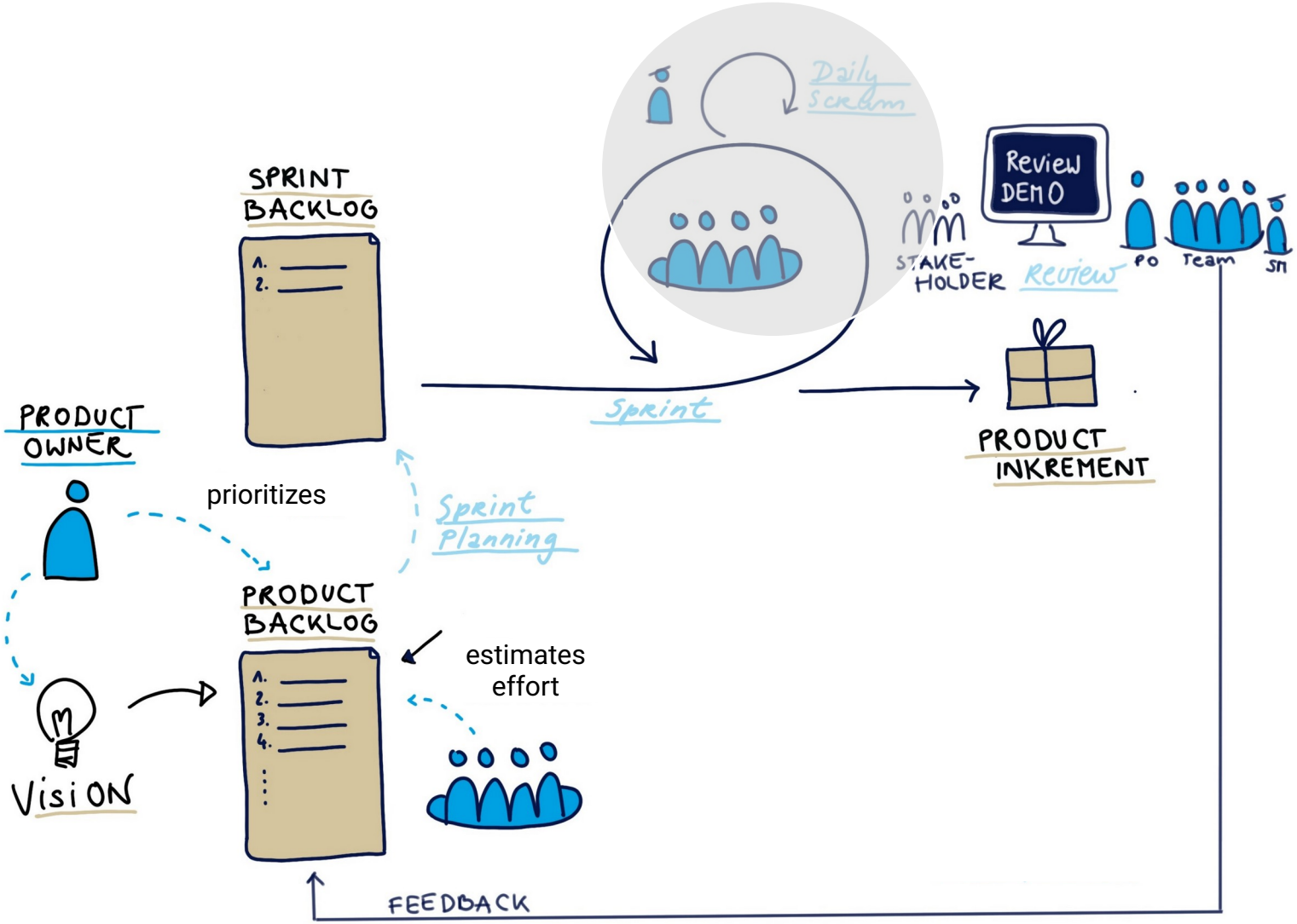
The SCRUM process
→ Product development in a sprint



In the sprint...

- The team determines how much content of the backlog is feasible in the sprint
- The content that is to be implemented in a sprint is recorded in the sprint backlog
- The contents of the sprint CANNOT be changed during the sprint.
- Principle: Pull

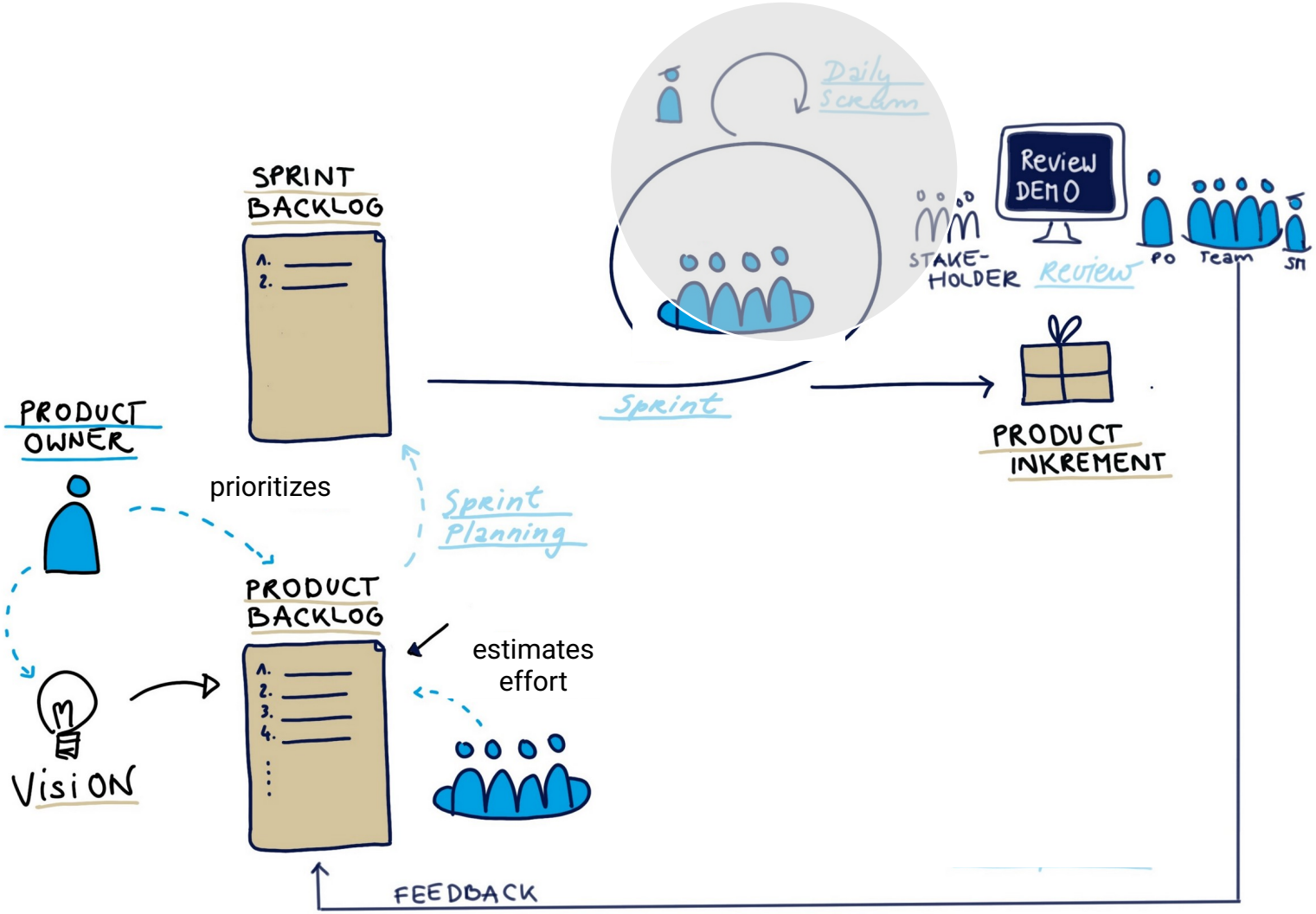
The SCRUM process → Product development in a sprint



In the sprint...

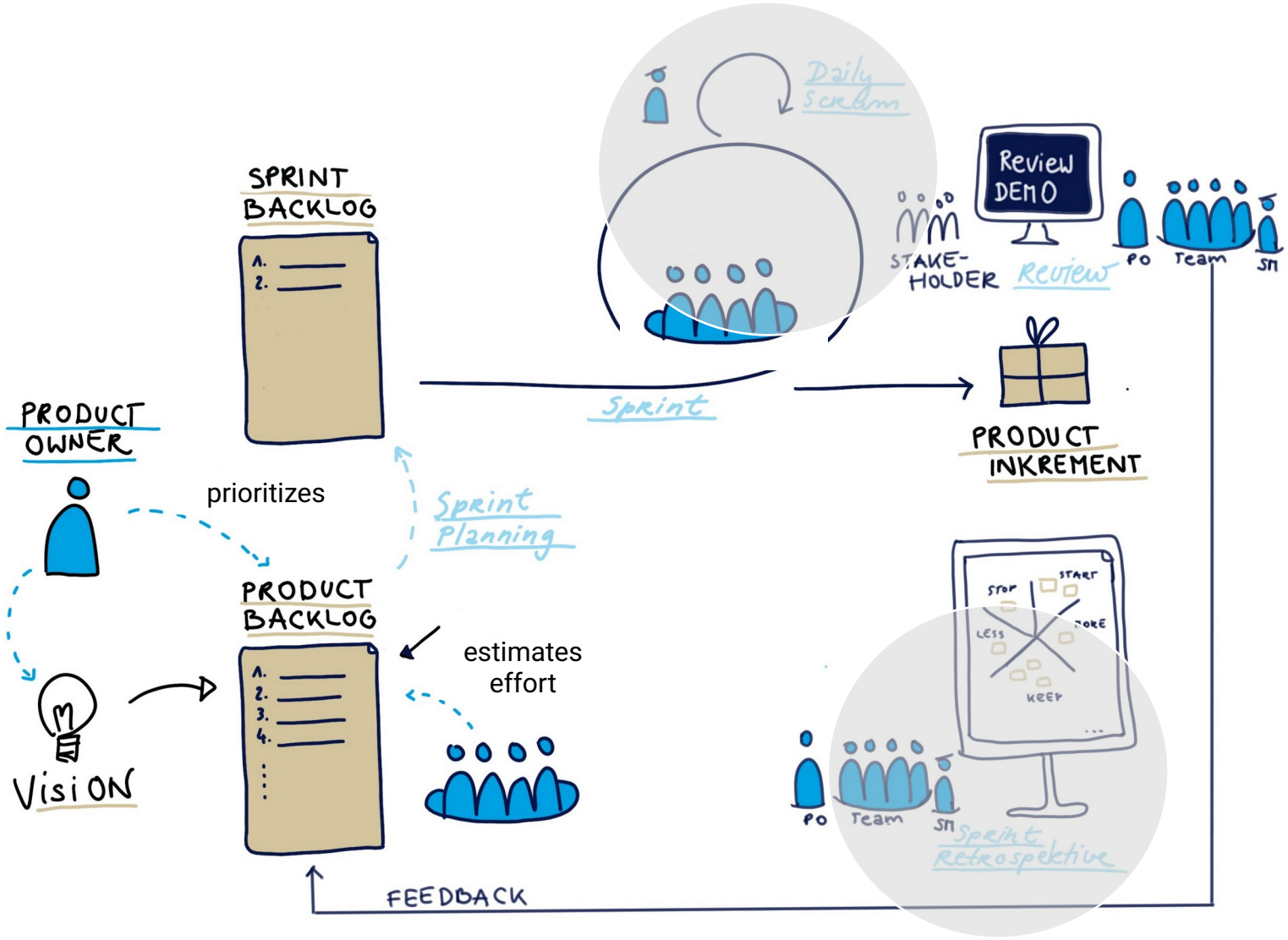
- The team meets daily for the so-called *Daily Scrum*
- Time and length of the meeting constant
- Questions
  - What have I done since the last Daily Scrum?
  - What obstacles do I see on the way to my sprint goal?
  - What do I plan to do before the next Daily Scrum that will bring us closer to the sprint goal?
- Participants: Scrum Master, Team. PO optional

The SCRUM process
→ Learning in the SCRUM process (inspect & adapt)



Learning is an essential part of SCRUM...

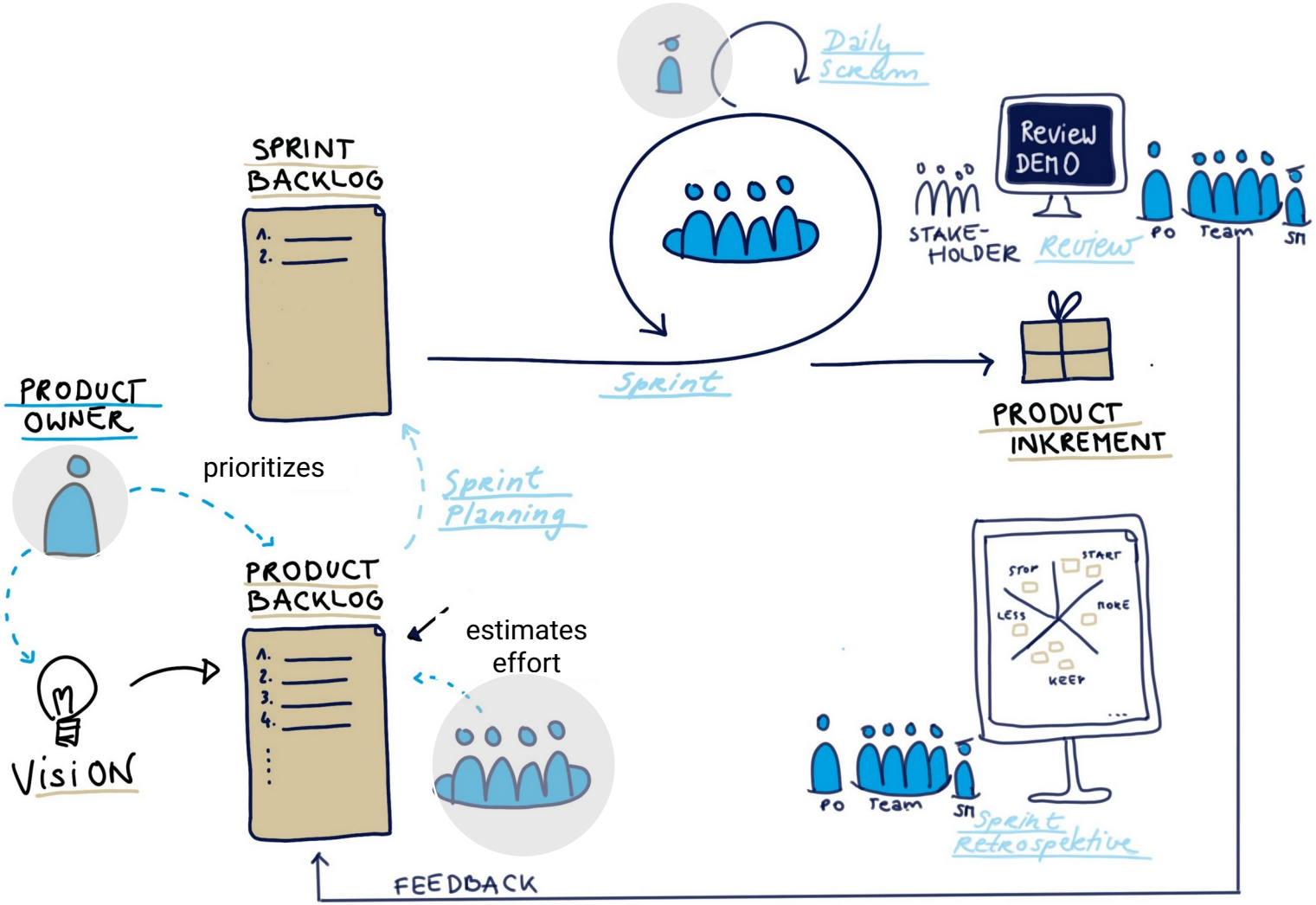
- The SCRUM master is a coach for the team
- Ensures compliance with SCRUM rules, time-boxing, etc.
- Supports the team in identifying and removing obstacles



Learning is an essential part of SCRUM...

- Obstacles are identified in the *Daily Scrum* and removed if possible
- Obstacles hinder work in the current sprint
- Every sprint ends with a *retrospective*
- Here the team reflects on what went well and what went less well in the sprint.
- The Scrum Master moderates the retrospective

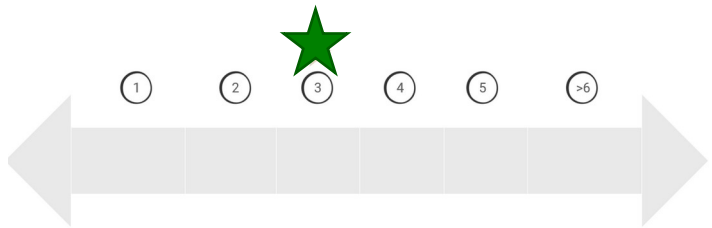
SCRUM roles



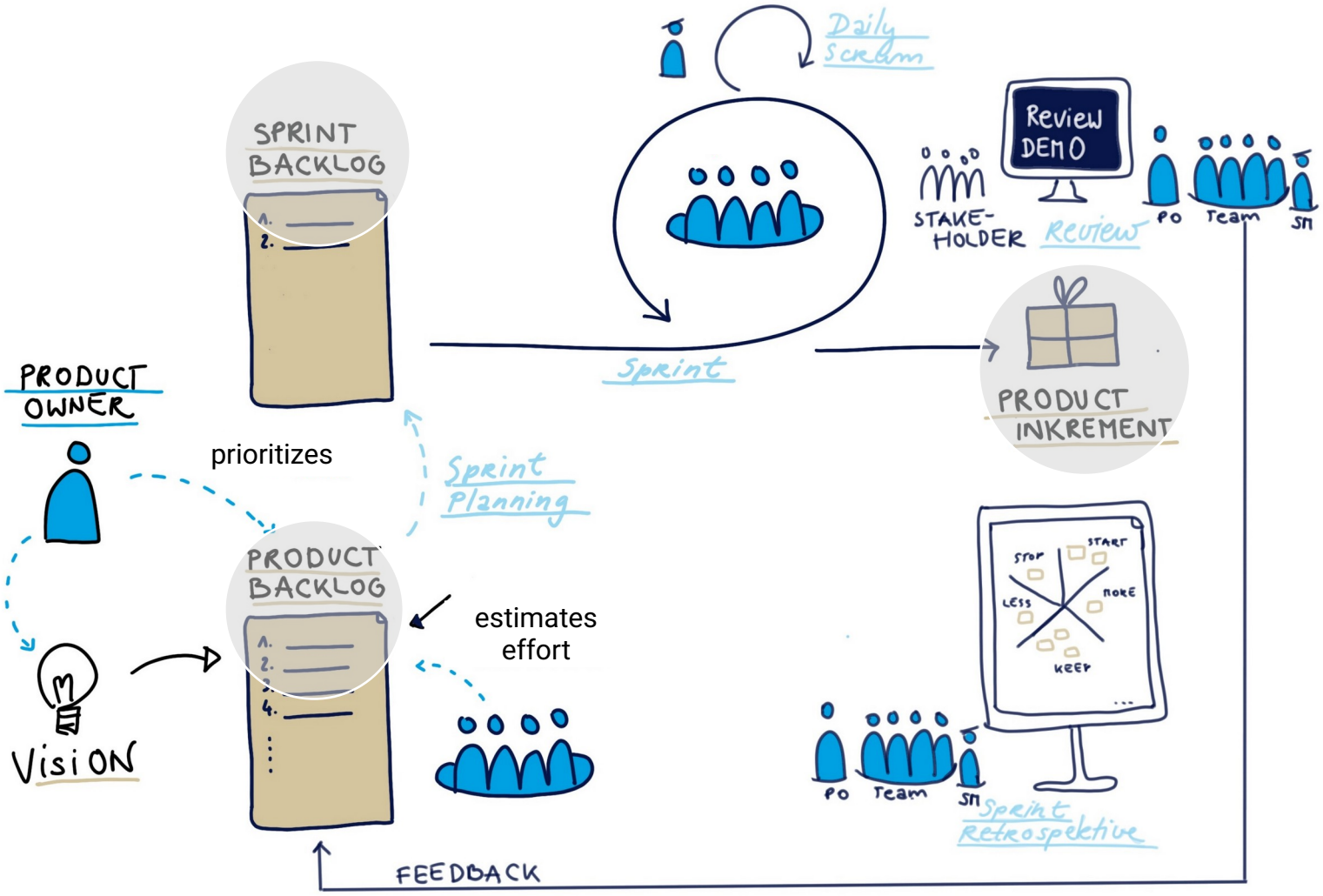
Product Owner

Scrum Master

Team



SCRUM artifacts

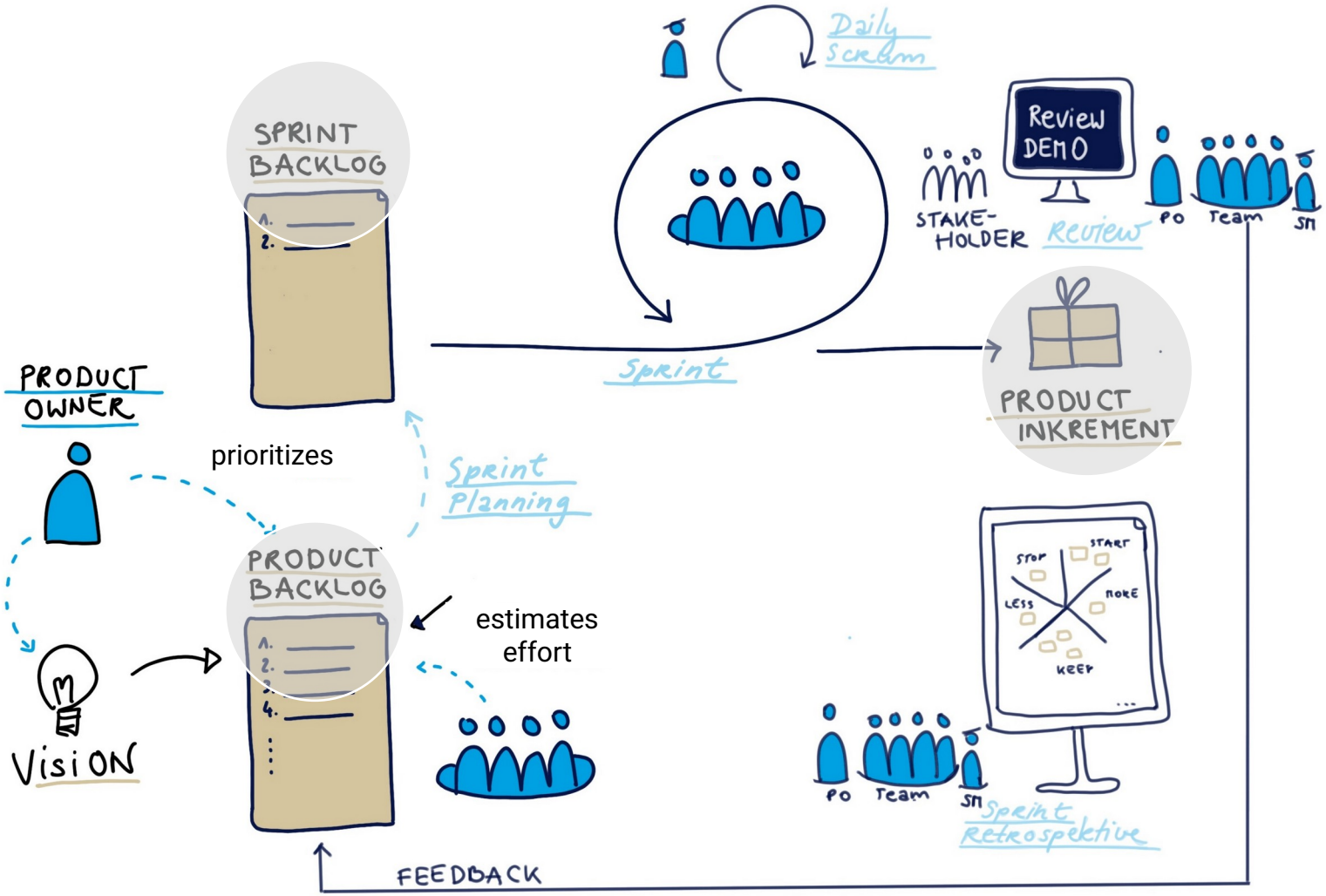


- Product backlog
- Sprint backlog
- Product Increment
- → **DoD - Definition of Done**
  - → Describes the team's quality requirements for the increment ("quality gate"), When are we finished? E.g. when we have carried out all test cases, when the documentation has been updated, when code reviews have taken place, etc.

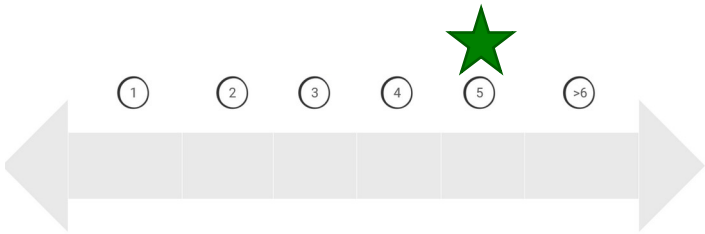
→ **Acceptance criteria**

- → Describes criteria that lead to the acceptance of the increment by the PO. Often referred to as acceptance test cases.

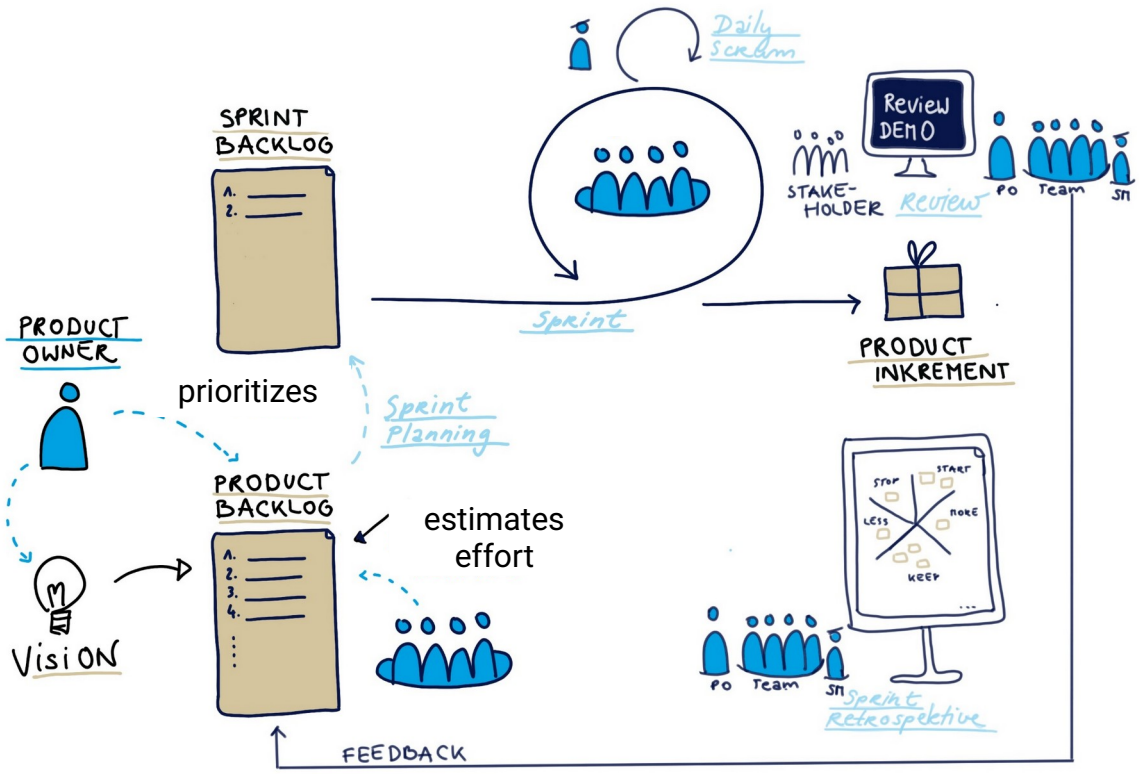
SCRUM events



- Sprint
- Sprint Planning
- Daily Scrum
- Sprint Review
- Sprint Retrospective



SCRUM in a nutshell



Discuss in small groups where the values of SCRUM

- Openness
- Respect
- Courage
- Commitment
- Focus

are important?

XP, Scrum

→ Discussion

- **High quality:** How well is quality supported?
  - Very good through QA practices (acceptance test cases/acceptance criteria, unit tests, continuous integration, in SCRUM as a quasi-standard in the DoD)
- **Satisfied customers:** How well does it ensure usefulness for users?
  - Good communication: Onsite-Customer (XP), ProductOwner(Scrum)
- **Maintainability:** How well is efficient development and further development supported?
  - XP has many good practices but too little documentation
  - More documentation with SCRUM, but too little for long-term development
- **Cost/time:** How well is resource compliance supported?
  - Very good practices such as 40h week (XP), timeboxing (Scrum)
  - Too little structure for very large projects

Typical Processes in Software Development

Agile!

1. Code and Fix
2. Waterfall
3. V-Model (Boehm)
4. W-Model
5. V-Model XT
6. UP
7. XP
8. SCRUM
9. KANBAN

Learning Objectives

- You can explain the basic principle of KANBAN
- You can explain and apply core practices of Kanban "in knowledge work"

Kanban

→ Background

- Originally from industrial manufacturing, first used in the Toyota Production System.
- Until then, individual process steps of a production process were optimized; Kanban aims to **optimize the entire process.**
- Kanban = kan = signal, ban = card ("order slip" → Triggers production process)
  - A signal card (Kanban) is sent to the upstream location when parts are required. Only then are these parts produced or provided.
  - The card is therefore a **signal** to produce more parts or to stop because there is enough.
  - Decentralized production control
- First introduced by David J. Anderson **outside of production**, i.e. for information technology. [Anderson, 2017]

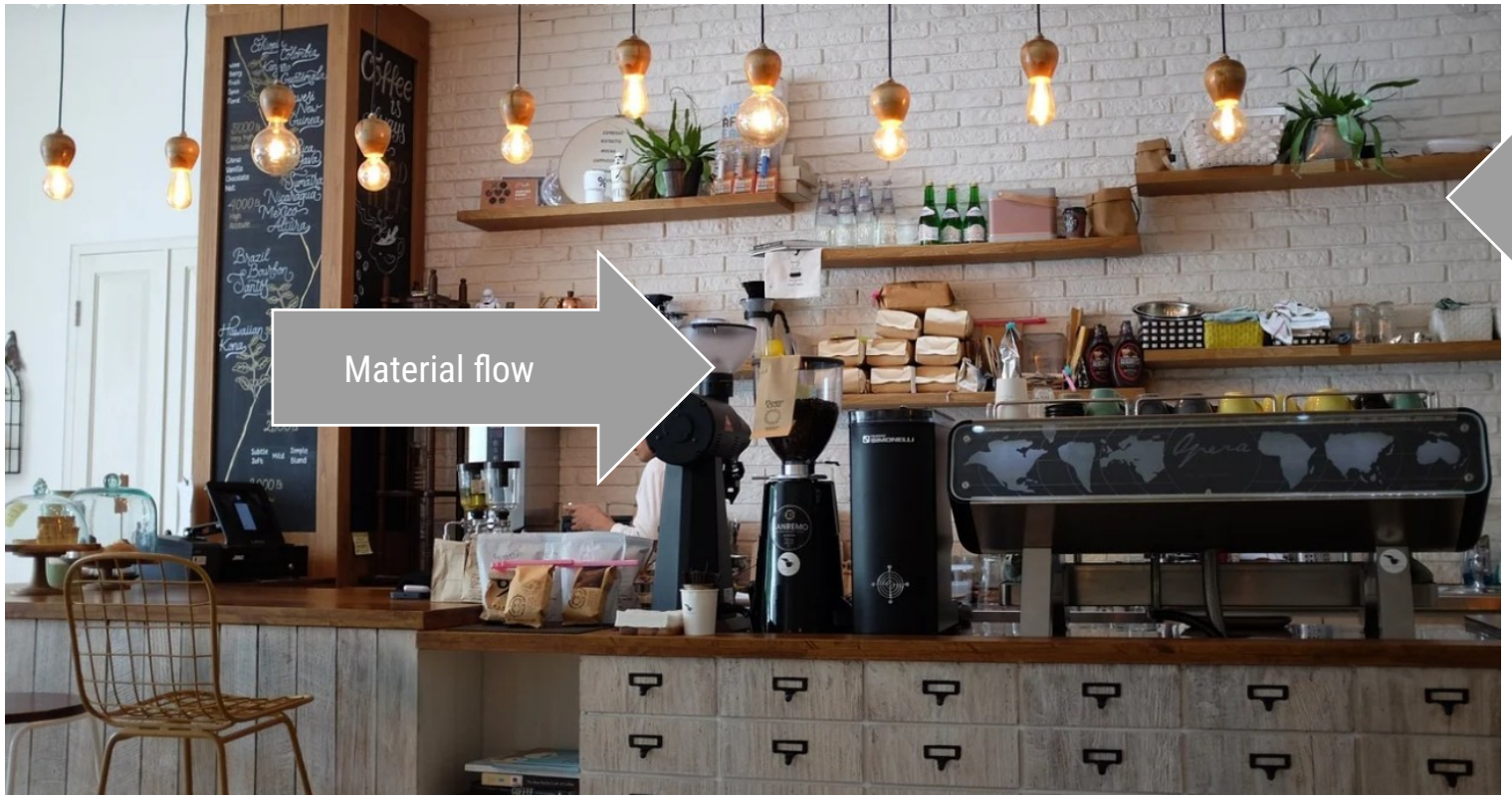
Kanban

→ Example

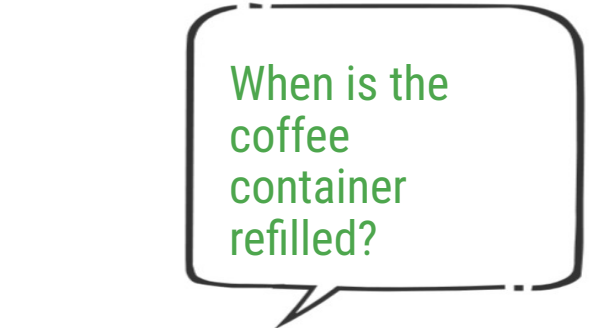
https://pixabay.com/de/photos/coffee-shop-barista-cafe-1209863/

Kanban

→ Example

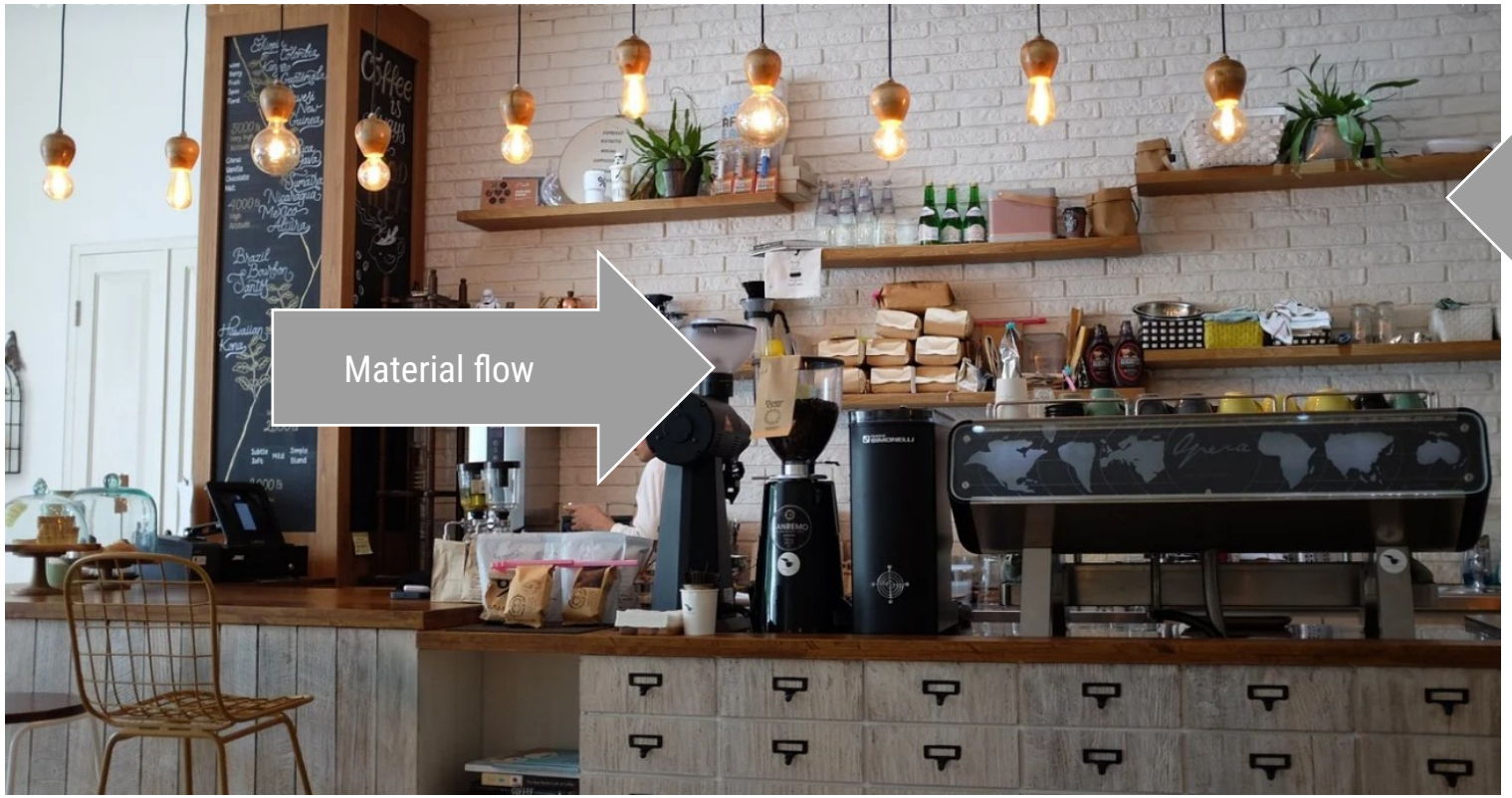


https://pixabay.com/de/photos/coffee-shop-barista-cafe-1209863/



Kanban

→ Example



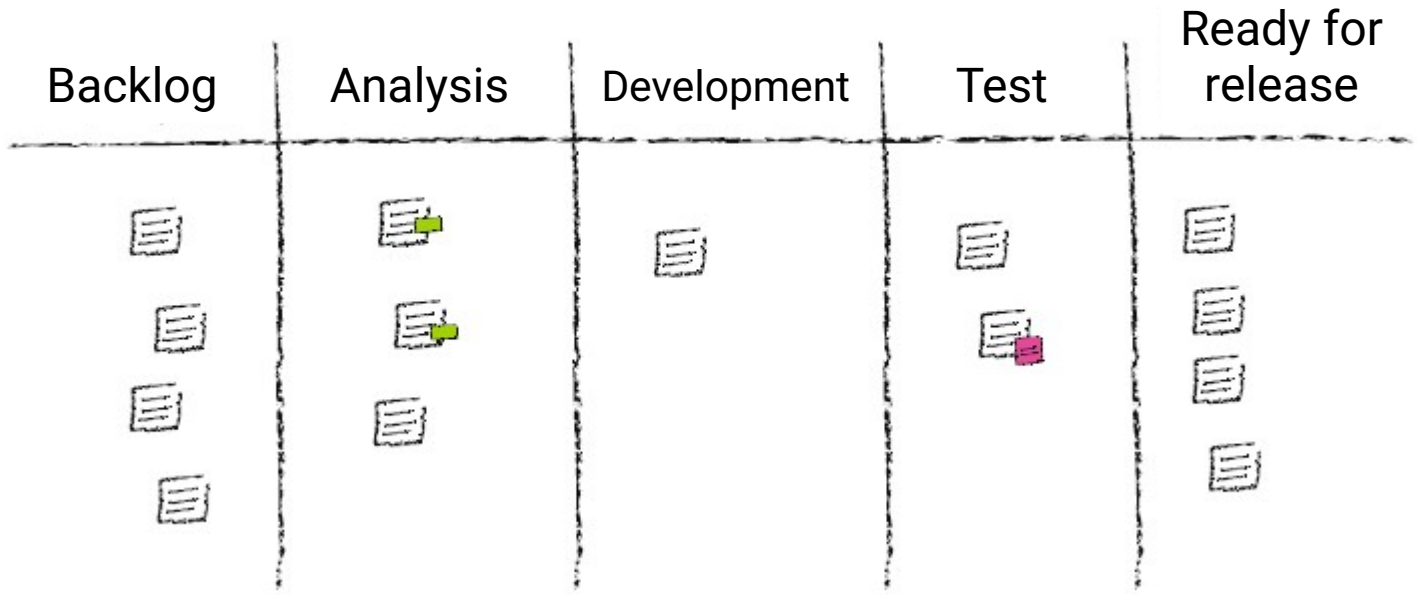
https://pixabay.com/de/photos/coffee-shop-barista-cafe-1209863/

- A downstream process (ordering coffee) tells the upstream process (making coffee) that new parts (coffee) are needed.
- Typically, the barista receives "kanbans" (= orders), which they process.
- Rarely is production "on stockpile"
- No parts are produced without a Kanban signal (=order slips).

Kanban

→ In "knowledge work"

- Controls the "invisible" amount of work in a system, which is made visible by "signal cards".



[Leopold, Kaltenecker, 2018]

Kanban

→ Core practices in knowledge work [Anderson, 2010]

In general: No specifications "HOW" something should be done, rather "WHAT" should be done.

1. Make work **visible**
2. **Limit** the work in progress
3. Manage the **flow**
4. Make **process rules** explicit
5. Implement **feedback mechanisms**
6. Carry out **joint** improvements

Kanban

→ 1. Make work visible

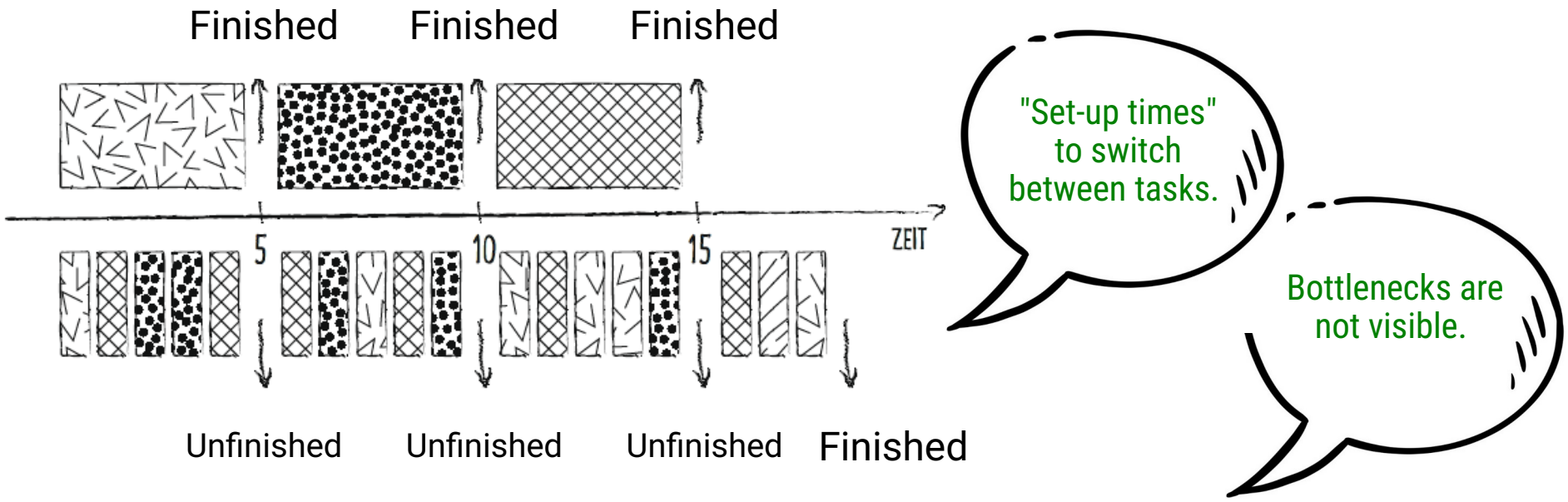
In general: No specifications "HOW" something should be done, rather "WHAT" should be done.

- Kanban works with boards on which the work is visualized. To set up a board, a few questions need to be clarified
  - What is the first **step in the process**?
  - What the last one?
  - Which steps are run through within the system boundaries?
  - What are the **rules** that govern the work? (→ See also "Make the rules explicit"). When can a "card" be moved to the next column?
  - In general: **Pull** NOT push!!!

Kanban

→ 2. Limit the work in progress

Unfinished products are tied-up capital. The greater the number of "packages" started, the longer the throughput time → Goal: Limit the number of tasks that are carried out simultaneously in one work step.

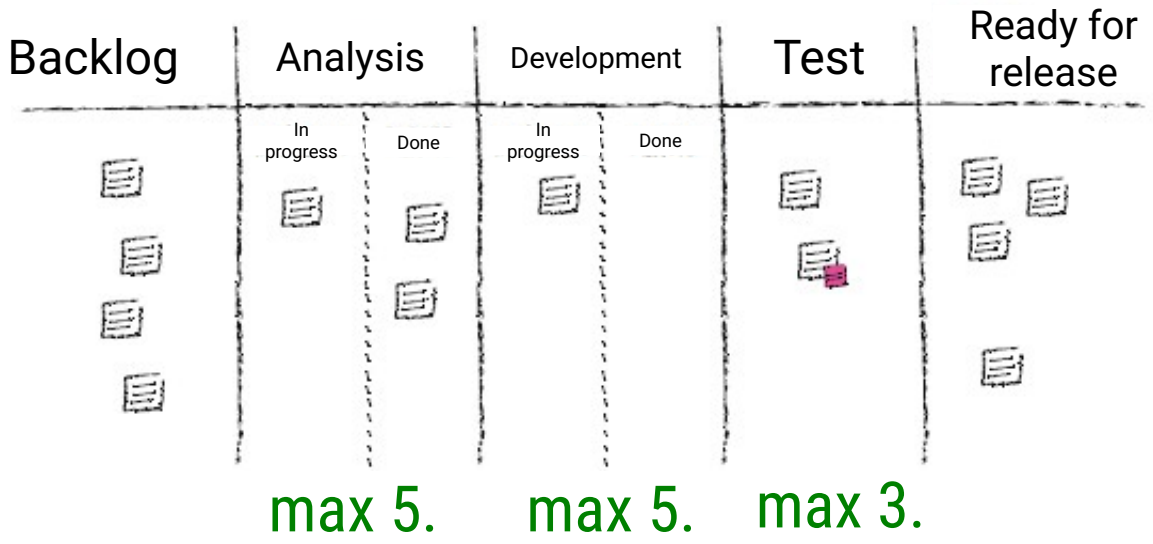


[Leopold, Kaltenecker, 2018]

Kanban

→ 2. Limit the work in progress

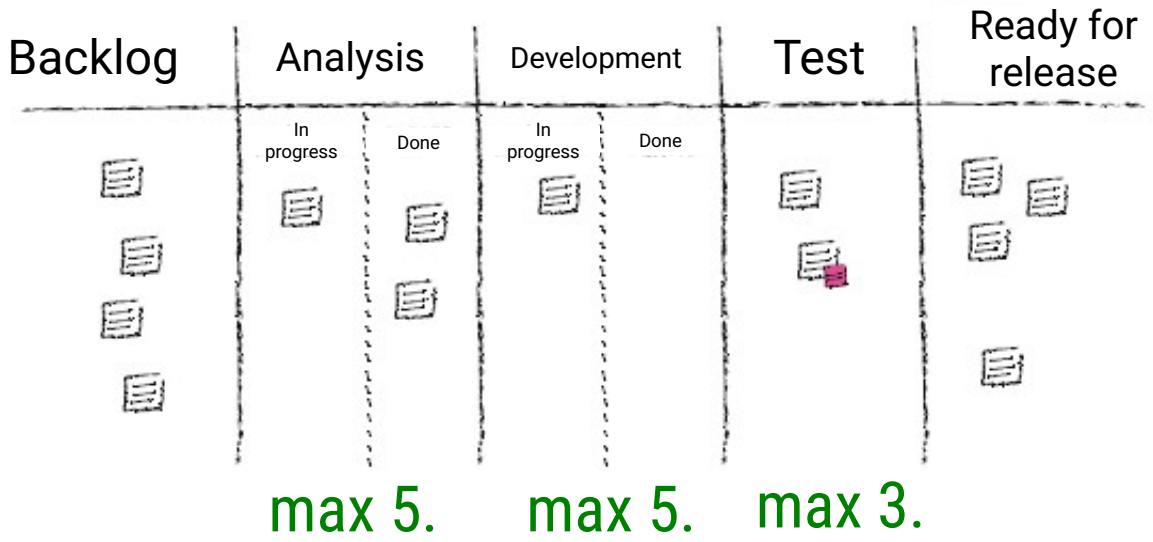
- The focus is on "getting ready".
- Assuming there is a limit on the "Test" column. Then only the specified number of cards may be there.
- If more cards are "actually" available, the team must first ensure that the congestion in the test is cleared and the flow of work is restored.



Kanban

→ 2. Limit the work in progress

- In other words, not everyone continues with "their" work, but they **work together to** ensure that it continues.
- Only **finished** work counts.
- Also means: "free space" on the board so that another work package could be **drawn** in this area.
- Once again: PULL NOT Push!!!



Kanban

→ 3. Manage the flow

- Good workflow is characterized by **uniformity** and **predictability**.
- Work on your problems first (= everything that causes the workflow to stall, e.g. bottleneck in a step) before you start new work.
- Communicate (several times) a day, e.g. in the Daily Standup

Kanban

→ 4, 5, 6

4. Make process rules explicit

- **Few rules**, but they are followed without exception.
- The principle of visualization also applies to rules: Make them visible!
- Rules take "emotions" out of processes. It is jointly agreed what happens if, for example, too many tickets have the status "Test".
- Rules are decided jointly (→ 6 Team Commitment).
- Rules are changed immediately if they have not proven themselves. (→6)

5. Implement feedback mechanisms

- Daily, Retro, etc.

6. Carry out joint improvements

- Start with the **current situation** and improve it **in small steps**.
- A Kanban board is therefore **never static**.

TH Aschaffenburg
university of applied sciences

KANBAN

→ Discussion

- **High quality:** How well is quality supported?
  - Framework too generic, explicit integration of QA practices in the workflow necessary
- **Satisfied customers:** How well does it ensure usefulness for users?
  - See point above
- **Maintainability:** How well is efficient development and further development supported?
  - Very high transparency of the process and process bottlenecks
  - Joint processing of bottlenecks → High exchange of knowledge
- **Cost/time:** How well is resource efficiency supported?
  - Limitation of work-in-progress, steady pace

Process models - Who am I?

→ These are: **Waterfall, V-model, XP, V-model XT, Kanban, UP**

- Plan everything carefully, it will pay off later
- Take everything to the extreme
- I give you a maximum model, you have to adapt it yourself.
- Mistakes are easiest (cheapest) to find at the level at which they were made.
- Stop starting, start finishing
- Carry out all meaningful practices with the necessary intensity
- Improve continuously

Process models

→ Who I am: This includes: **Waterfall, V-model, XP, V-model XT, spiral model, Kanban, UP**

- Plan everything carefully, it pays off later (waterfall, V-model)
- Take everything to the extreme → XP
- I give you a maximum model, you have to adapt it yourself. → V-Model XT
- Errors are easiest (cheapest) to find at the level at which they were made. → V-model
- Stop starting, start finishing → Kanban
- Carry out all meaningful practices with the necessary intensity → UP
- Improve continuously (Kaizen → Kanban, SCRUM)

Literature

- [Ludewig 2013] Ludewig, Lichter: Software Engineering. Basics, people, processes, techniques, dpunkt.verlag.
- [Paech 2021] Barbara Paech: Lecture Software Engineering, Uni-Heidelberg.

Questions?

TH Aschaffenburg
university of applied sciences

Thank you for your attention!

Software Engineering

Prof. Dr. J. v. Kistowski