Softwaretechnik / Software-Engineering

Lecture 4: Procedure & Process Models

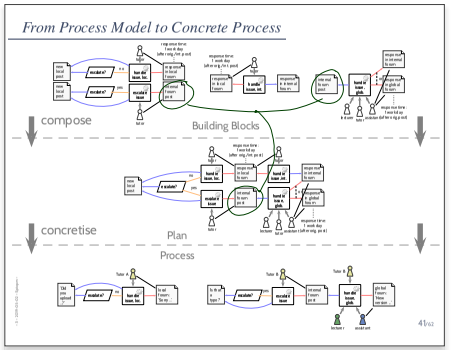
2019-05-06

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

Topic Area Project Management: Content

From Process Model to Concrete Process



Content

Process vs. Procedure Models

Process vs. Procedure Model

(Ludewig and Lichter, 2013) propose to distinguish: **process model** and **procedure model**.

- A **Process model** ('Prozessmodell') comprises
  - (i) **Procedure model** ('Vorgehensmodell')
    Example: "Waterfall Model" (70s/80s).
  - (ii) **Organisational structure** – comprising requirements on
    - project management and responsibilities,
    - quality assurance,
    - documentation, document structure,
    - revision control.

  Examples: V-Modell, RUP, XP (90s/00s).
- Note: In the literature, **process model** and **procedure model** are often used as synonyms; there are (again) no universally agreed terms. . .
- Anticipated **benefits** of using process models:
  - "economy of thought"
  - clear responsibilities
  - fewer errors
  - quantification, reproducibility

Content

- **Procedure and Process Models**
  - Vocabulary:
    - linear / non-linear
    - evolutionary, iterative, incremental
    - prototyping
- **Procedure** Model Examples
  - The (in)famous Waterfall model
  - The famous Spiral model
- **Process** Model Examples
  - Code-and-Fix, Phase Model
  - V-Modell XT
  - Agile
    - Extreme Programming (XP)
    - Scrum
- **Process Metrics**
  - CMMI, Spice

Procedure Model Examples

Linear vs. Non-Linear Procedure Models

- **linear**: basically the strict **Waterfall Model**
  (without feedback between activities)
- **non-linear**: basically **everything else**
  (with feedback between activities)

Iterative, Incremental, Evolutionary

- **Iterative Development**:

iterative software development – software is developed in **multiple iterative steps**, all of them planned and controlled.

Goal: each iterative step, beginning with the second, corrects and improves the existing system based on defects detected during usage.

Each iterative steps includes the characteristic activities **analyse, design, code, test.** **Ludewig & Lichter** (2013)

- **Incremental Development**:

incremental software development – The total extension of a system under development remains open; it is realised in **stages of expansion**. The first stage is the **core system**.

Each stage of expansion extends the existing system and is subject to a separate project. Providing a new stage of expansion typically includes (as with iterative development) an improvement of the old components.
Ludewig & Lichter (2013)

- **Evolutionary Development**:

evolutionary software development – an approach which includes evolutions of the developed software under the influence of practical/field testing.

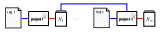
New and changed requirements are considered by developing the software in **sequential steps of evolution**.

Ludewig & Lichter (2013), **ftw.** (Zülighoven, 2005)

Iterative, Incremental, Evolutionary

- **Iterative Development**:

- **Incremental Development**:



- **Evolutionary Development**:

- **Note**: (to maximise confusion) IEEE calls our "iterative" incremental:

> **incremental development** – A software development technique in which requirements definition, design, implementation, and testing occur in an overlapping, iterative (rather than sequential) manner, resulting in incremental completion of the overall software product. **IEEE 610.12** (1990)

- One difference (in our definitions):
  - **iterative**: steps towards fixed goal,
  - **incremental**: goal extended for each step; next step goals may already be planned.

Prototyping

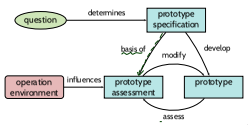
> **prototype** – A preliminary type, form, or instance of a system that serves as a model for later stages or for the final, complete version of the system. **IEEE 610.12** (1990)

> **prototyping** – A hardware and software development technique in which a preliminary version of part or all of the hardware or software is developed to permit user feedback, determine feasibility, or investigate timing or other issues in support of the development process. **IEEE 610.12** (1990)

> **rapid prototyping** – A type of prototyping in which emphasis is placed on developing prototypes early in the development process to permit early feedback and analysis in support of the development process. **IEEE 610.12** (1990)

- classification by **usage**:
  - **demonstration prototype**
  - **functional prototype**
  - **lab sample**
  - **pilot system**, etc.
- classification by **supported activity**:
  - **explorative p.** (analysis)
  - **experimental p.** (design)
  - **evolutionary p.** (product is last prototype)

Prototyping Procedure Model



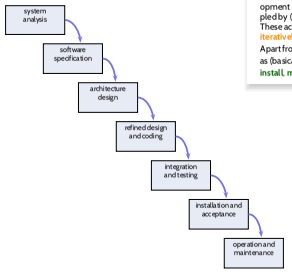
(Ludewig and Lichter, 2013)

Questions towards '**definition of done**':

- Which **purpose** does the prototype have?
  What are the **open questions**?
- Which persons (roles) participate in **development**?
  And, most important, who participates in **assessment** of the prototype?
- What is the **time/cost budget** for prototype development?

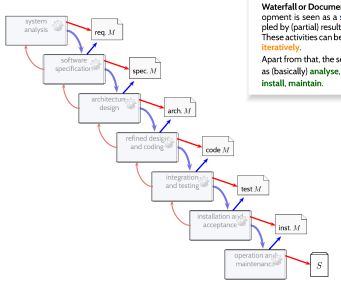
Content

The (In)famous Waterfall Model (Rosove, 1967)



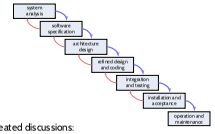
Waterfall or Document-Model– Software development is seen as a **sequence of activities** coupled by (partial) results (documents).
These activities can be conducted **concurrently** or **iteratively**.
Apart from that, the sequence of activities is fixed as (basically) **analyse**, **specify**, **design**, **code**, **test**, **install**, **maintain**. **Ludewig & Lichter** (2013)

The (In)famous Waterfall Model (Rosove, 1967)



Waterfall or Document-Model– Software development is seen as a **sequence of activities** coupled by (partial) results (documents).
These activities can be conducted **concurrently** or **iteratively**.
Apart from that, the sequence of activities is fixed as (basically) **analyse**, **specify**, **design**, **code**, **test**, **install**, **maintain**. **Ludewig & Lichter** (2013)

The Waterfall Model: Discussion



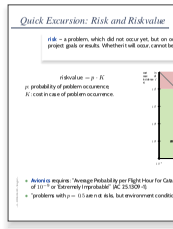
(In)famous?!

- The waterfall model has been subject of heated discussions:
  - Original model without feedback **not realistic**.
  - Gives room for many intepretations; **very abstract**; hardly usable as a "template" for planning real projects.
  - Cycles (and the lack of milestones) makes it hard for project management to **assess a project's process**.
- Maybe best appreciated in the context of its time:

  "Dear people (of the 60's), there is more in software development than coding; and there are (obvious) dependencies."

  That may have been news to some software people back then… (cf. "**software crisis**").
- Everybody knows it (at least the name…).

The Spiral Model (Boehm, 1988)



Barry W. Boehm

Risks in the **software development process** can have various forms and counter-measures, e. g.,

- open **technical questions** (→ prototype?),
- **lead developer** about to leave the company (→ invest in documentation?),
- changed **market situation** (→ adapt appropriate features?),
- …

The Spiral Model (Boehm, 1988) Cont'd

Idea of the **Spiral Model**: iteratively address the (currently) highest risk (instead of planing ahead everything).

Repeat until end of project (successful completion or failure):

(i) **determine** the set R of **risks** which are **threatening** the project;
if $R = \emptyset$, the project is successfully completed

(ii) **assign** each risk $r \in R$ a **risk value** $v(r)$

(iii) for the risk r_0 with the **highest risk value**, $r_0 = \max\{v(r) \mid r \in R\}$,
find a way to eliminate this risk, and go this way;
if there is no way to eliminate the risk, stop with project failure

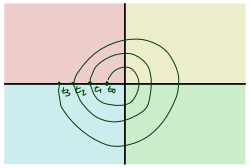
Advantages:

- We know early if the project goal is unreachable.
- Knowing that the biggest risks are eliminated gives a good feeling.

Wait, Where's the Spiral?

A concrete process using the Spiral Model could look as follows:

- investigate goals, alternatives, side conditions
- develop and test the next product part,
- conduct risk analysis,
- plan the next phase,



Content

Process Model Examples

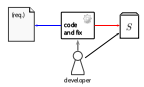
From Procedure to Process Model

A **process model** may describe:

- **steps** to be conducted during development,
  their sequential arrangement,
  their dependencies
  (the **procedure model**)
- **organisation**, responsibilities, roles
- structure and properties of **documents**
- **methods** to be used,
  e.g., for gathering requirements or checking intermediate results
- project phases, **milestones**, testing criteria
- **notations** and languages
- **tools** to be used
  (in particular for project management).

Process models typically come with their **own terminology** (to maximise confusion?),
e.g. what we call **artefact** is called **product** in V-Model terminology.

Trivial Example: Code & Fix



- **Code & Fix** denotes an approach where **coding** (programming) or **fixing** (repairing defects) in alternation with **ad-hoc testing** are the **only consciously** conducted activities.
- **Advantages:**
  - corresponds to the impulse to proceed quickly and solve the problem
  - yields executable programs early
  - simple activities
- **Disadvantages:**
  - project not plannable
  - hard to distribute project over multiple persons or groups
  - often comes without serious requirements and proplem analysis
  - ad-hoc testing lacks expected values ('Soll-Wert')
  - resulting programs often badly structured and hard to maintain
  - high effort (and cost) for corrections; issues often detected late
  - important concepts and decisions usually not documented

→ **sabotages** quality, overall too expensive

The Phase Model: Phases, Milestones

> A **phase** is a continuous, i.e. not interrupted range of time in which certain works are carried out and completed. At the end of each phase, there is a **milestone**.
> A phase is **successfully completed** if the criteria defined by the milestone are satisfied.
> **Ludewig & Lichter** (2013)

- Phases (in this sense) **do not overlap**!
  Yet there may be different "threads of development" running in parallel, structured by different milestones.
- Splitting a project into phases **makes controlling easier**;
  milestones may involve the customer (accept intermediate results) and trigger payments.
- The **granularity** of the phase structuring is critical:
  - very short phases may not be tolerated by a customer,
  - very long phases may mask significant delays longer than necessary.

  If necessary:
  define **internal** (customer not involved) and **external** (customer involved) milestones.

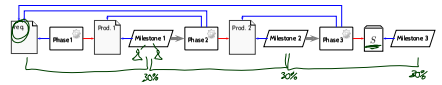
Milestones, Deadlines

> A **phase** is a continuous, i.e. not interrupted range of time in which certain works are carried out and completed. At the end of each phase, there is a **milestone**.
> A phase is **successfully completed** if the criteria defined by the milestone are satisfied.
> **Ludewig & Lichter** (2013)

- Whether a milestone is **reached** (or successfully completed) must be **assessable** by
  - clear,
  - objective, and
  - unambiguous

  criteria.
- The **definition of a milestone** often comprises:
  - a definition of the **results** which need to be achieved,
  - the required **quality** properties of these results,
  - the desired **time** for reaching the milestone (the **deadline**), and
  - the instance (person or committee) which **decides** whether the milestone is reached.
- Milestones can be part of the **development contract**;
  not reaching a defined milestone as planned can lead to **legal claims**.

The Phase Model

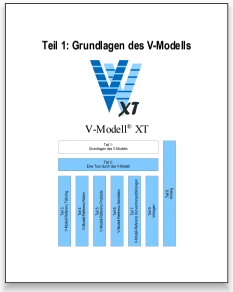


- The project is planned by **phases**,
  delimited by well-defined **milestones**.
- Each phase is assigned a **time/cost budget**.
- Phases and milestones may be part of the development contract;
  partial payment when reaching milestones.
- Roles, responsibilities, artefacts **defined as needed**.
- By definition, there is **no iteration of phases**.
- But **activities may span** (be active during) **multiple phases**.
- Not uncommon for small projects
  (few software people, small product size), and small companies.

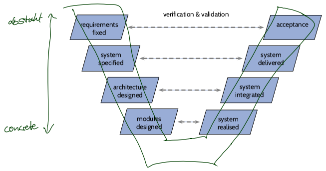
Content

- **Procedure and Process Models**
  - Vocabulary:
    - linear / non-linear
    - evolutionary, iterative, incremental
    - prototyping
- **Procedure** Model Examples
  - The (in)famous Waterfall model
  - The famous Spiral model
- **Process** Model Examples
  - Code-and-Fix, Phase Model
  - V-Modell XT
  - Agile
    - Extreme Programming (XP)
    - Scrum
- **Process Metrics**
  - CMMI, Spice

V-Model XT

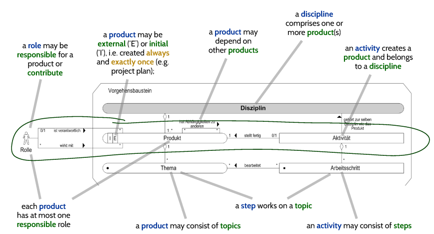


V-Modell XT



- There are different "**V-shaped**" **process models**, we discuss the (German) "V-Modell".
- "**V-Modell**":
  - developed by company IABG in cooperation with the Federal Office for Defence Technology and Procurement ('Bundesministerium für Verteidigung'), released 1998
  - (German) government as customer often **requires** usage of the V-Modell
- 2012: "**V-Modell XT**" Version 1.4 (Extreme Tailoring) (V-Modell XT, 2006)

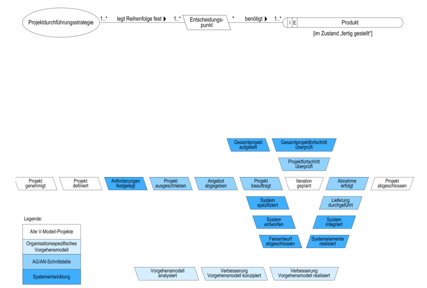
V-Modell XT: Procedure Building Blocks



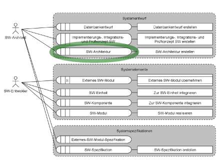
our course	V-Modell XT	explanation
role	role ('Rolle')	
activity	activity ('Aktivität')	
-	step ('Arbeitsschritt')	parts of activities
artefact	product ('Produkt')	
-	topic ('Thema')	parts of products

our course	V-Modell XT	explanation
-	discipline ('Disziplin')	set of related products / activities
phase	project segment (?) ('Projektabschnitt')	

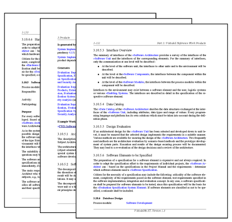
V-Modell XT: Decision Points

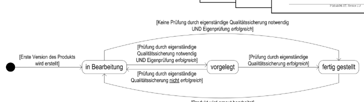


V-Modell XT: Example Building Block & Product State

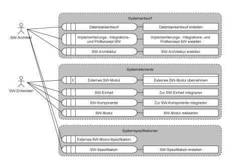


SW-Development ('SW-Entwicklung')



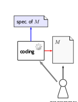


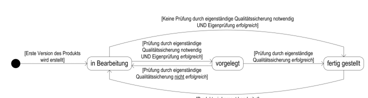
V-Modell XT: Example Building Block & Product State



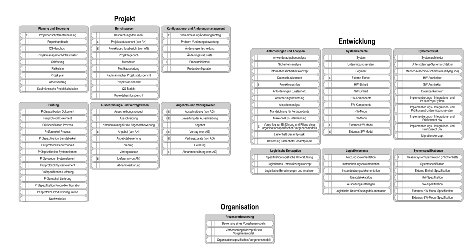
SW-Development ('SW-Entwicklung')

vs.

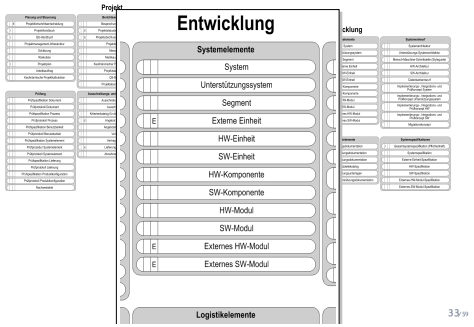




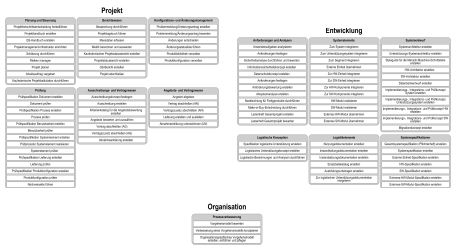
V-Modell XT: (Lots of) Disciplines and Products



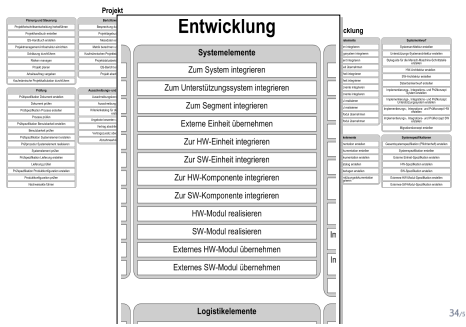
V-Modell XT: (Lots of) Disciplines and Products



V-Modell XT: Activities (as many?!)



V-Modell XT: Activities (as many?!)



V-Modell XT: Roles (even more?!)

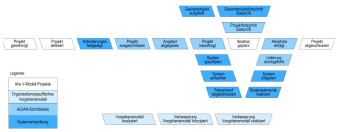
Project Roles:

Änderungssteuerungsgruppe (Change Control Board), Änderungsverantwortlicher, Anforderungsanalytiker (AG), Anforderungsanalytiker (AN), Anwender, Assessor, Ausschreibungsverantwortlicher, Datenschutzverantwortlicher, Ergonomieverantwortlicher, Funktionssicherheitsverantwortlicher, HW-Architekt, HW-Entwickler, Informationssicherheitsverantwortlicher, KM-Administrator, KM-Verantwortlicher, Lenkungsausschuss, Logistikentwickler, Logistikverantwortlicher, Projektkaufmann, Projektleiter, Projektmanager, Prozessingenieur, Prüfer, QS-Verantwortlicher, SW-Architekt, SW-Entwickler, Systemarchitekt, Systemintegrator, Technischer Autor, Trainer

Organisation Roles:

Akquisiteur, Datenschutzbeauftragter (Organisation), Einkäufer, IT-Sicherheitsbeauftragter (Organisation), Qualitätsmanager

What About the Colours?



V-Modell XT: Project Types

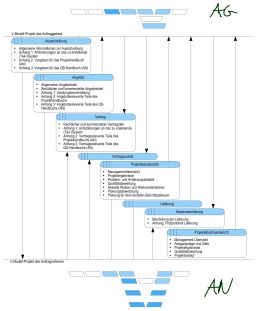
V-Modell XT considers four different **project types**:

- **AG**: project from the perspective of the customer (create call for bids, choose developer, accept product)
- **AN**: project from the perspective of the developer (create offer, develop system, hand over system to customer)
- **AG/AN**: customer and developer from same organisation
- **PM**: introduction or improvement of a process model

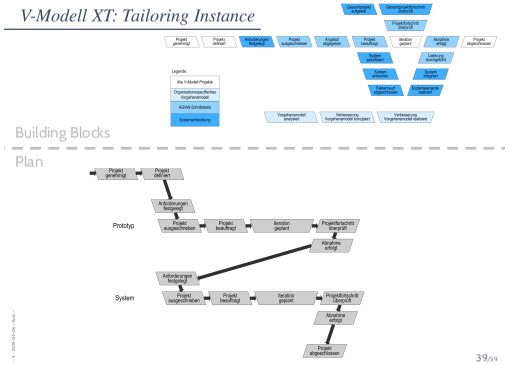
Project type variants: one/many customer(s); development/improvement/migration; maintenance

project role	customer 'Auftraggeber'	developer 'Auftragnehmer'	customer/developer 'Auftragg./'Auftragn.'	customer/developer 'Auftragg./'Auftragn.
project type	system development project (AG)	system development project (AN)	system development project (AG/AN)	introduction and maintenance of specific process model
project subject	HW system	SW system	HW-SW system/embedded	System integration	introduction and maintenance of specific process model

V-Modell XT: Customer/Developer Interface



V-Modell XT: Tailoring Instance



V-Modell XT: Development Strategies

V-Modell XT mainly supports three **strategies**,
i.e. principal **sequences between decision points**,
to develop a system:

incremental component based prototypical

V-Modell XT: Discussion

Advantages:

- certain **management related building block** are part of each project,
  thus they may receive **increased attention** of management and developers
- publicly **available**, can be used **free of license costs**
- very **generic**, support for **tailoring**
- **comprehensive**, **low risk of forgetting** things

Disadvantages:

- **comprehensive**, tries to cover everything; tailoring is supported, but may need high effort
- tailoring is **necessary**, otherwise a huge amount of useless documents is created
- description/presentation leaves **room for improvement**

Needs to prove in practice, in particular in small/medium sized enterprises (SME).

Agile

The Agile Manifesto

"Agile – denoting 'the quality of being agile; readiness for motion; nimbleness, activity, dexterity in motion' – software development methods are attempting to offer an answer to the eager business community asking for lighter weight along with faster and nimbler software development processes.

This is especially the case with the rapidly growing and volatile Internet software industry as well as for the emerging mobile application environment." (*Abrahamsson et al., 2002*)

The Agile Manifesto (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.

Agile Principles

- *"continous / sustainable delivery"*
  - *Our highest priority is to **satisfy the customer** through early and **continuous delivery** of valuable software.*
  - ***Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
  - *Agile processes promote **sustainable development**.*
    The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- *"simplicity"*
  - ***Simplicity** – the art of **maximizing the amount of work not done** – is essential.*
  - ***Working software is the primary measure** of progress.*
- *"changes"*
  - ***Welcome changing requirements**, even late in development.*
    Agile processes harness change for the customer's competitive advantage.
- *"people"*
  - *The best architectures, requirements, and designs emerge from **self-organizing teams**.*
  - ***Build projects around motivated individuals**.*
    Give them the environment and support they need, and trust them to get the job done.
  - ***Business people and developers must work together** daily throughout the project.*
  - *The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**.*
- *"retrospective"*
  - *Continuous **attention to technical excellence** and good design enhances agility.*
  - *At regular intervals, **the team reflects** on how to become more effective, then tunes and **adjusts its behavior accordingly**.*

Similarities of Agiles Process Models

- **iterative**: cycles of a few weeks, at most three months.
- Work in small groups (6–8 people) proposed.
- Dislike the idea of large, comprehensive documentation (radical or with restrictions).
- Consider the customer important;
  recommend or request customer's presence in the project.
- Dislike dogmatic rules.

(Ludewig and Lichter, 2013)

Agile
— Extreme Programming (XP) —

Extreme Programming (XP) (Beck, 1999)

XP values:

- **simplicity**, **feedback**, **communication**, **courage**, **respect**.

XP practices:

- **management**
  - integral team (including customer)
  - planning game (→ Delphi method)
  - short release cycles
  - stand-up meetings
  - assess in hindsight
- **team**:
  - joint responsibility for the code
  - coding conventions
  - acceptable workload
  - central metaphor
  - continuous integration
- **programming**
  - test driven development
  - refactoring
  - simple design
  - pair programming

Agile
— Scrum —

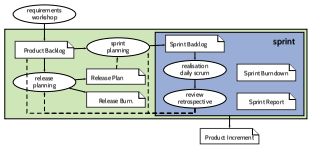
Scrum

- First published 1995 (Schwaber, 1995), based on ideas of **Takeuchi** and **Nonaka**.
- Inspired by **Rugby** (yes, the "hooligan's game played by gentlemen"): get the ball in a **scrum**, then **sprint** to score.
- Role-based; iterative and incremental;
  in contrast to XP no techniques proposed/required.

Three roles:

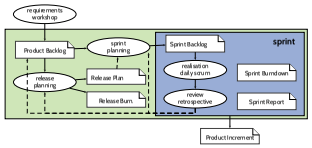
- **product owner**:
  - representative of customer.
  - maintains requirements in the product backlog.
  - plans and decides which requirement(s) to realise in next sprint.
  - (passive) participant of daily scrum.
  - assesses results of sprints
- **scrum team**:
  - members capable of developing autonomously.
  - decides how and how many requirements to realise in next sprint.
  - distribution of tasks self-organised, team decides who does what when.
  - environment needs to support communication and cooperation, e.g. by spatial locality
- **scrum master**:
  - helps to conduct scrum the right™ way.
  - looks for adherence to process and rules.
  - ensures that the team is not disturbed from outside.
  - moderates daily scrum, responsible for keeping product backlog up-to-date.
  - should be able to assess techniques and approaches

Scrum Process



- **product backlog**
  (maintained by product owner)
  - comprises all requirements to be realised,
  - priority and effort estimation for requirements,
  - collects tasks to be conducted,
- **release plan**
  - based on initial version of product backlog,
  - how many sprints, which major requirements in which sprint,
- **release-burndown report**
  - see sprint-burndown report
- **sprint backlog**
  - requirements to be realised in next sprint, taken from product backlog,
  - more precise estimations,
  - daily update (tasks done, new tasks, new estimations)
- **sprint-burndown report**
  - completed/open tasks from sprint backlog,
  - should decrease linearly, otherwise remove tasks from sprint backlog,
- **sprint report**
  - which requirements (not) realised in last sprint,
  - description of obstacles/problems during sprint

Scrum Process



- **daily scrum**:
  - daily meeting, 15 min.
  - discuss progress, synchronise day plan, discuss and document new obstacles
  - team members, scrum master, product owner (if possible)
- **sprint**:
  - at most 30 days, usually shorter (initially longer)
- **sprint review**:
  - assess amount and quality of realisations; product owner accepts results
- **sprint retrospective**:
  - assess how well the scrum process was implemented; identify actions for improvement (if necessary)

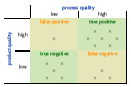
Scrum: Discussion

- Has been used in many projects, experience in majority positive.
- Team size bigger 7–10 may need **scrum of scrums**.
- Competent **product owner** necessary for success.
- Success depends on motivation, competence, and communication skills of team members.
- Team members are responsible for planning, and for adhering to process and rules, thus **intensive learning and experience** necessary.
- Can (as other process models) be combined with techniques from XP.

Process Metrics

Assessing Process Quality

- A **good process**, in general, does not stop us from creating **bad products**,
- (the hope is, that) bad products are less likely when using a good process, i.e. that there is a correlation like:

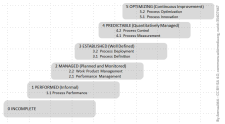


- Some customers would like to only work with contractors with **good processes**.
- But **how to measure** the quality of a process?

SPICE (Hörmann et al., 2006) and CMMI (Team, 2010)

- **SPICE / ISO 15504** (Software Process Improvement and Capability Determination)
  - can be seen as a specification for process **pseudo-metrics**; ISO/IEC 15504 Part 5 gives one example implementation
  - idea:
    - define considered **process areas**
    - **assess** each process for so-called **process attributes**
    - map results to **maturity level**

    assessment conducted by specially trained assessors (→ subjective metrics)



- **CMMI** (Capability Maturity Model Integration)
  - considers 5 **process categories** (project magmt., support, engineering, process mgmt.),
  - each consisting of 5–7 **process areas**,
  - each process area can be assigned a **capability level** (0: incomplete, 1: performed, 2: managed, 3: defined)
  - capability levels can be **aggregated** to organisation's **maturity level** (1: initial, 2: managed, 3: defined, 4: quantitatively managed, 5: optimizing)
  - flavours: CMMI-DEV, CMMI-ACQ, CMMI-SVC

Content

Discussion

Recall: Anticipated Benefits of Process Modelling:

- **"economy of thought"**
- **quantification, reproducibility**
- **fewer errors**
- **clear responsibilities**

- **Process model-ing** is easily **overdone** – the best process model is **worthless** if your software people don't "live" it.
- Before introducing a process model
  - understand what you have, understand what you need.
  - process-model as much as needed, not more (→ tailoring).
  - assess whether the new/changed process model makes matters better or worse (→ metrics)
- **Note**: customer may require a certain process model.

Tell Them What You've Told Them...

- **Classification** of processes
  - **linear, non-linear**
  - **evolutionary, iterative, incremental**
  - **prototyping**: needs purposes and questions
- **Procedure Models**
  - **Waterfall** (very well-known, very abstract, of limited practical use)
  - **Spiral** (iterated risk assessment, e.g., for very innovative projects)
- **V-Model XT**
  - slightly different vocabulary,
  - quite comprehensive,
  - may serve as inspiration for, e.g., definition of roles,
  - can be tailored in various ways
- **Agile** approaches
  - **Extreme Programming** (XP) (proposes methods and approaches)
  - **Scrum** (focuses on management aspects)
- Measure **process quality**: **CMMI**, **Spice**

References

References

Abrahamsson, P., Salo, O., Ronkainen, J., and Warsta, J. (2002). Agile software development methods. review and analysis. Technical Report 478.

Beck, K. (1999). *Extreme Programming Explained - Embrace Change*. Addison-Wesley.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *IEEE Computer*, 21(5):61–72.

Hörmann, K., Dittmann, L., Hindel, B., and Müller, M. (2006). *SPICE in der Praxis: Interpretationshilfe für Anwender und Assessoren*. dpunkt.verlag.

IEEE (1990). *IEEE Standard Glossary of Software Engineering Terminology*. Std 610.12-1990.

Ludewig, J. and Lichter, H. (2013). *Software Engineering*. dpunkt.verlag, 3. edition.

Rosove, P. E. (1967). *Developing Computer-based Information Systems*. John Wiley and Sons.

Schwaber, K. (1995). SCRUM development process. In Sutherland, J. et al., editors, *Business Object Design and Implementation, OOPSLA'95 Workshop Proceedings*. Springer-Verlag.

Team, C. P. (2010). Cmmi for development, version 1.3. Technical Report ESC-TR-2010-033, CMU/SEI.

V-Modell XT (2006). *V-Modell XT*. Version 1.4.

Züllighoven, H. (2005). *Object-Oriented Construction Handbook - Developing Application-Oriented Software with the Tools and Materials Approach*. dpunkt.verlag/Morgan Kaufmann.