Softwaretechnik / Software-Engineering

Lecture 4: Software Project Management

2016-05-02

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

Albert-Ludwigs-Universität Freiburg, Germany

Topic Area Project Management: Content

VL 2

- **Software Metrics**
  - Properties of Metrics
  - Scales
  - Examples

⋮

VL 3

- **Cost Estimation**
  - "(Software) Economics in a Nutshell"
  - Expert's Estimation
  - Algorithmic Estimation

⋮

VL 4

- **Project Management**
  - Project
  - Process and Process Modelling
  - Procedure Models
  - Process Models

⋮

VL 5

- **Process Metrics**
  - CMMI, Spice

⋮

Content

- **(Software) Project**
- **Project Management**
  - Goals and Activities
  - Common Activities
  - Excursion: Risk
- **Software Project Planning**
  - Costs and Deadlines
    - phase, milestone, deadline
  - Tasks and Activities
    - cycle, life cycle
    - software life cycle
  - People and Roles
    - responsibilities and rights
- **Software Development Process**
- **Procedure and Process Models**

Project

Vocabulary: Project

project – A **temporary activity** that is characterized by **having**

- a **start date**,
- specific **objectives and constraints**,
- established **responsibilities**,
- a **budget and schedule**, and
- a **completion date**.

If the objective of the project is to develop a software system, then it is sometimes called a **software development project** or **software engineering project**. **R. H. Thayer** (1997)

We could refine our earlier definition as follows: a project is **successful** if and only if

- **started** at start date,
- **achieved** objectives, **respected** constraints,
- **adheres** to budges and schedule,
- **stops** at completion date.

Whether, e.g., objectives have been achieved can still be **subjective** (→ customer/user happy).

Vocabulary: Software Project

(software) project – characteristics:

- **Duration** is limited.
- Has an **originator** (person or institution which initiated the project).
  - The **project owner** is the originator or its representative.
  - The **project leader** reports to the project owner.
- Has a **purpose**, i.e. pursue a bunch of goals.
  - The most important goal is usually to create or modify software; this software is thus the result of the project, the **product**.
    Other important goals are extension of know-how, preparation of building blocks for later projects, or utilisation of employees.

  The project is called **successful** if the goals are reached to a high degree.
- Has a **recipient** (or will have one).
  - This recipient is the **customer**.
  - Later **users** (conceptionally) belong to the customer.
- The project **links people**,
  results (intermediate/final products), and **resources**.

  The **organisation** determines their roles and relations, and the **external interfaces** of the project. **Ludewig & Lichter** (2013)

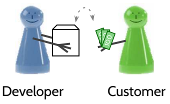
Project Management

Goals and Activities of Project Management

- **Main and general goal**: a **successful** project,
  i.e. the project **delivers**
  - defined **results**
  - in demanded **quality**
  - within scheduled **time**
  - using the assigned **resources**.

  There may be **secondary goals**, e.g.,
  - build or strengthen good **reputation** on market,
  - acquire **knowledge** which is useful for later projects,
  - develop **re-usable components** (to save resources later),
  - be attractive to **employees**.
  - …

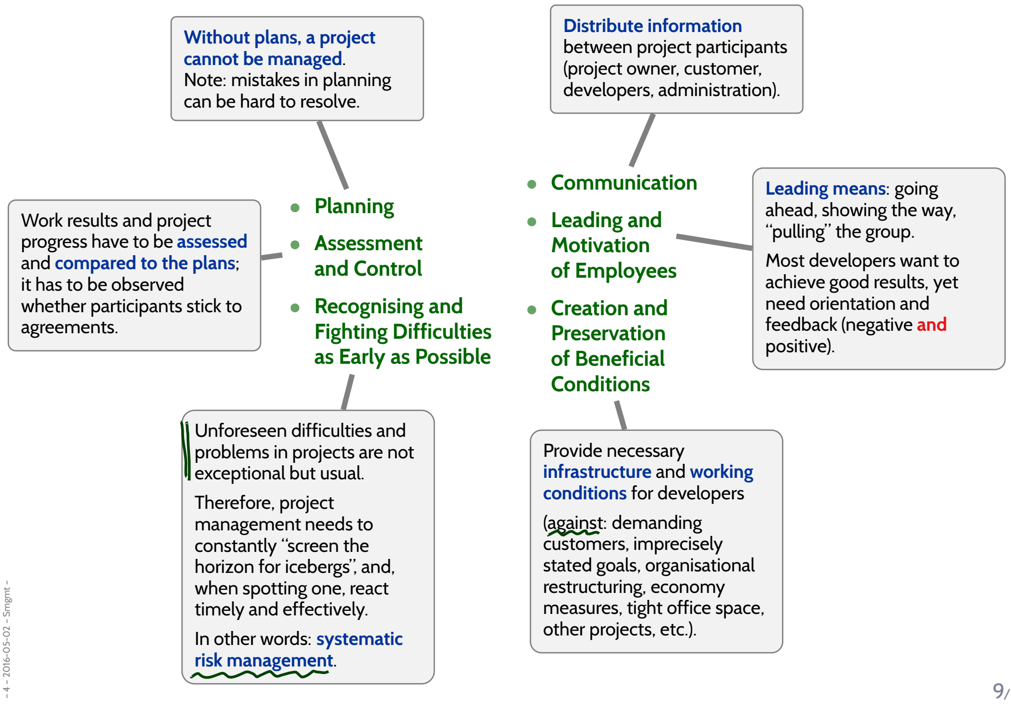
  } may influence estimation



software delivery

- Main **project management activities** (and **responsibilities** of project manager):
  - **Planning**
  - **Assessment and Control**
  - **Recognising and Fighting Difficulties as Early as Possible**
  - **Communication**
  - **Leading and Motivation of Employees**
  - **Creation and Preservation of Beneficial Conditions**

Activities of Project Management



Without plans, a project cannot be managed.
Note: mistakes in planning can be hard to resolve.

Work results and project progress have to be **assessed** and **compared to the plans**; it has to be observed whether participants stick to agreements.

- **Planning**
- **Assessment and Control**
- **Recognising and Fighting Difficulties as Early as Possible**

Unforeseen difficulties and problems in projects are not exceptional but usual.

Therefore, project management needs to constantly “screen the horizon for icebergs”, and, when spotting one, react timely and effectively.

In other words: **systematic risk management**.

Distribute information between project participants (project owner, customer, developers, administration).

- **Communication**
- **Leading and Motivation of Employees**
- **Creation and Preservation of Beneficial Conditions**

Leading means: going ahead, showing the way, “pulling” the group.

Most developers want to achieve good results, yet need orientation and feedback (negative **and** positive).

Provide necessary **infrastructure** and **working conditions** for developers

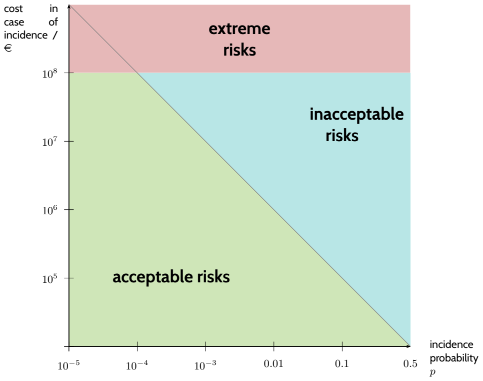
(against: demanding customers, imprecisely stated goals, organisational restructuring, economy measures, tight office space, other projects, etc.).

Quick Excursion: Risk and Riskvalue

> **risk** – a problem, which did not occur yet, but on occurrence threatens important project goals or results. Whether it will occur, cannot be surely predicted.
>
> **Ludewig & Lichter (2013)**

$$\text{riskvalue} = p \cdot K$$

p: probability of problem occurrence,
K: cost in case of problem occurrence.



- **Avionics** requires: “Average Probability per Flight Hour for Catastrophic Failure Conditions of 10^{-9} or ‘Extremely Improbable’” (AC 25.1309-1).
- “problems with $p = 0.5$ are not risks, but environment conditions to be dealt with”

Project Management

Software Engineering
as **defensive discipline**.

Analogy: safety belt;
or hygiene in hospital:

> "Dear patient, we're working hard to protect you from an infection."
> – "Well, doctor, I thought you were working to **get me well** again."

"Software Engineering is boring and frustrating for people who do not value the defense of failures as a positive achievement."
(Ludewig and Lichter, 2013)

"Tanker Summit Europe" von world24 in der Wikipedia auf Deutsch. Lizenziert unter CC BY-SA 3.0 über Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Tanker_Summit_Europe.JPG#/media/File:Tanker_Summit_Europe.JPG

Software Project Planning

What to (Plan and) Manage?

Planning and managing software projects involves

- **costs** and **deadlines**,
- **tasks** and **activities**,
- **people** and **roles**.

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)

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 **decide**s whether the milestone is reached.

    or tool
- Milestones can be part of the **development contract**;
  not reaching a defined milestone as planned can lead to **legal claims**.

What to (Plan and) Manage?

Planning and managing software projects involves

- **costs** and **deadlines**,
- **tasks** and **activities**,
- **people** and **roles**.

Cycle and Life Cycle

cycle – (1) A period of time during which a set of events is completed. See also: ...

IEEE 610.12 (1990)

system life cycle – The period of time that begins when a system is **conceived** and ends when it is **no longer available for use**. **IEEE 610.12 (1990)**

software life cycle – The period of time that begins when a software product is **conceived** and ends when the software is no longer **available for use**. [...]**IEEE 610.12 (1990)**

UI

software development cycle – The period of time that begins with the **decision to develop** a software product and ends when the software is **delivered**. [...]

IEEE 610.12 (1990)

Software Life and Development Cycle

software life cycle – The period of time that begins when a software product is **conceived** and ends when the software is no longer **available for use**.

The software life cycle typically includes

- a concept phase,
- a requirements phase,
- a design phase,
- an implementation phase,
- a test phase,
- an installation and checkout phase,
- on operation and maintenance phase, and,
- sometimes, a retirement phase.

Note: These phases may overlap or be performed iteratively. **IEEE 610.12 (1990)**

Ul

software development cycle – The period of time that begins with the **decision to develop** a software product and ends when the software is **delivered**.

This cycle typically includes

- a requirements phase,
- a design phase,
- an implementation phase,
- a test phase, and
- sometimes an installation and checkout phase.

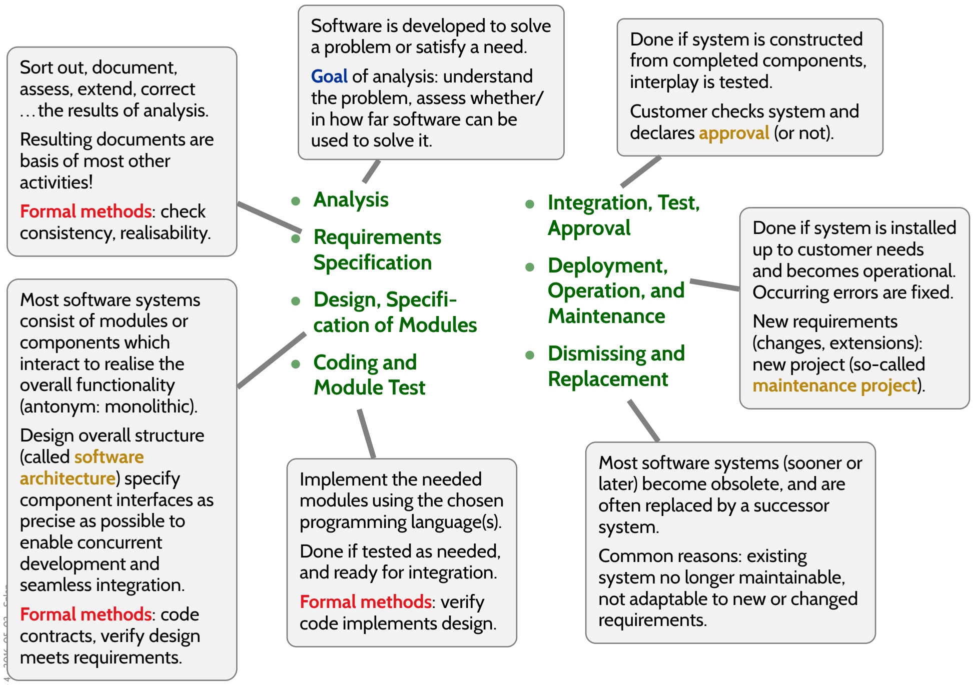
Notes:

(1) the phases listed above may overlap or be performed iteratively, depending upon the software development approach used.

(2) This term is sometimes used to mean a longer period of time, either the period that ends when the software is no longer being enhanced by the developer, or the entire software life cycle.

IEEE 610.12 (1990)

Common Activities in Order to Develop or Adapt Software



- Analysis
- Requirements Specification
- Design, Specification of Modules
- Coding and Module Test
- Integration, Test, Approval
- Deployment, Operation, and Maintenance
- Dismissing and Replacement

Software is developed to solve a problem or satisfy a need.

Goal of analysis: understand the problem, assess whether/in how far software can be used to solve it.

Sort out, document, assess, extend, correct …the results of analysis.

Resulting documents are basis of most other activities!

Formal methods: check consistency, realisability.

Most software systems consist of modules or components which interact to realise the overall functionality (antonym: monolithic).

Design overall structure (called software architecture) specify component interfaces as precise as possible to enable concurrent development and seamless integration.

Formal methods: code contracts, verify design meets requirements.

Implement the needed modules using the chosen programming language(s).

Done if tested as needed, and ready for integration.

Formal methods: verify code implements design.

Done if system is constructed from completed components, interplay is tested.

Customer checks system and declares approval (or not).

Done if system is installed up to customer needs and becomes operational. Occurring errors are fixed.

New requirements (changes, extensions): new project (so-called maintenance project).

Most software systems (sooner or later) become obsolete, and are often replaced by a successor system.

Common reasons: existing system no longer maintainable, not adaptable to new or changed requirements.

What to (Plan and) Manage?

Planning and managing software projects involves

- **costs** and **deadlines**,
- **tasks** and **activities**,
- **people** and **roles**.

The Concept of Roles

In a software project, at each point in time,
there is a set R of (active) **roles**, e.g. $R = \left\{ \boxed{\text{mgr}}, \boxed{\text{prg}}, \boxed{\text{tst}}, \boxed{\text{ana}} \right\}$.

A role has **responsibilities** and **rights**, and necessary skills and capabilities.

For example,

- $\boxed{\text{mgr}}$: project manager
  - has the **right** to raise issue reports
  - is **responsible** for closing issue reports
- $\boxed{\text{prg}}$: programmer
  - has the **right** to change the code
  - is **responsible** for reporting unforeseen problems to the project manager
  - is **responsible** for respecting coding conventions
  - is **responsible** for addressing issue reports
- $\boxed{\text{tst}}$: test engineer
  - has the **right** to raise issue reports
  - is **responsible** for quality control

The Concept of Roles Cont'd

Given a set R of roles, e.g. $R = \left\{ \boxed{\text{mgr}}, \boxed{\text{prg}}, \boxed{\text{tst}}, \boxed{\text{ana}} \right\}$,
and a set P of people, e.g. $P = \left\{ ♟, ♟, ♟, ♟, ♟ \right\}$, each with **skills** or **capabilities**.

An aspect of project management is to assign (a set of) people to each role:

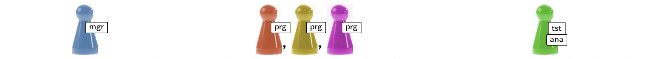
$$assign : R \to 2^P \longleftarrow \text{powerset of } P$$

such that each person $p \in assign(r)$ assigned to role r
has (at least) the skills and capabilities required by role r.

Note: $assign$ may change over time, there may be different assignments for different phases.

Sanity check: ensure that $assign(r) \neq \emptyset$ for each role r.

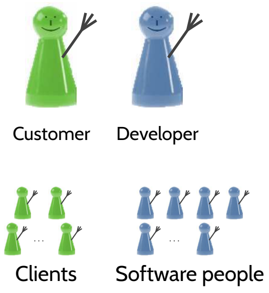
- **Example**:



one person, one role — multiple persons, one role — one person, multiple roles

$$assign = \left\{ \boxed{\text{mgr}} \mapsto \{♟\}, \boxed{\text{prg}} \mapsto \{♟, ♟, ♟\}, \boxed{\text{tst}} \mapsto \{♟\}, \boxed{\text{ana}} \mapsto \{♟\} \right\}$$

Useful and Common Roles



Recall: **roles** "Customer" and "Developer" are assumed by **legal persons**, which often represent many people.

The same legal person may act as "Customer" and "Developer" in the same project.

Useful and common roles in software projects:

- **customer**, **user**
- **project manager**
- **(sytems) analyst**
- **software architect**, **designer**
- (**lead**) **developer**
  programmer, **tester**, ...
- **maintenance engineer**
- **systems administrator**
- **invisible clients**: legislator,
  norm/standard supervisory committee

Content

- **(Software) Project**
- **Project Management**
  - Excursion: Risk
- **Software Project Planning**
  - Costs and Deadlines
  - Tasks and Activities
  - People and Roles
- **Software Development Process**
  - Vocabulary: role, artefact, activity
  - Describing & prescribing processes
- **Procedure and Process Models**
  - Procedure Model Examples
    - The (in)famous Waterfall model
    - The famous Spiral model
    - Procedure classification
  - Process Model Examples
    - V-Modell, RUP
    - Agile (XP, Scrum)

Software Development Process

Process

Process –

(1) A sequence of steps performed for a given purpose; for example, the software development process.

(2) See also: task; job.

(3) To perform operations on data.

IEEE 610.12 (1990)

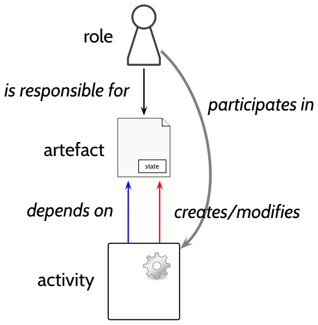
Software Development Process –
The process by which user needs are translated into a software product.
The process involves **translating** user needs into **software requirements**, **transforming** the software requirements into **design**, **implementing** the design in **code**, **testing** the code, and sometimes, **installing and checking out** the software for **operational use**.

IEEE 610.12 (1990)

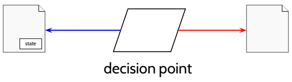
- The process of a software development project may be
  - implicit,
  - informally agreed on, or
  - explicitly prescribed (by a **procedure** or **process model**).
- **Note**: each software development project **has** a process!

Describing Software Development Processes

Over time, the following **notions** proved useful to describe and model (→ in a minute) software development processes:

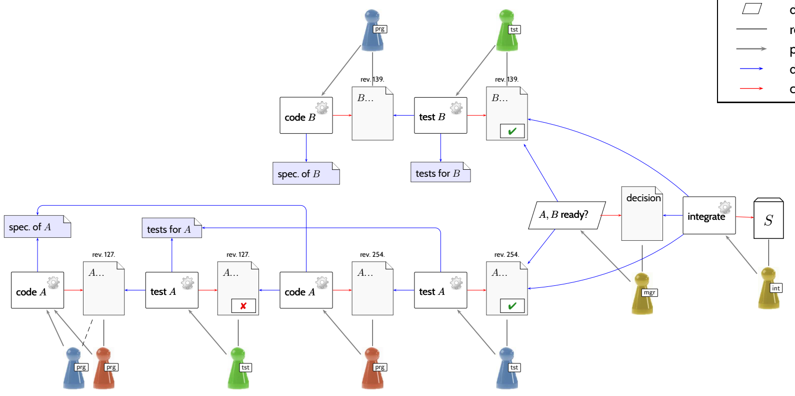


- **role** – has resposibilities and rights, needs skills and capabilities. In particular: responsibility for **artefacts**, participates in **activities**.
- **artefact** – all documents, evaluation protocols, software modules, etc., all products emerging during a development process. Is processed by **activities**, may have **state**.
- **activity** – any processing of artefacts, manually or automatic. Depends on **artefacts**, creates/modifies **artefacts**.
- **decision point** – special case of activity: a decision is made based on **artefacts** (in a certain state), creates a **decision artefacts**. Delimits phases, **corresponds to milestone**.



How Software S May Have Been Created...

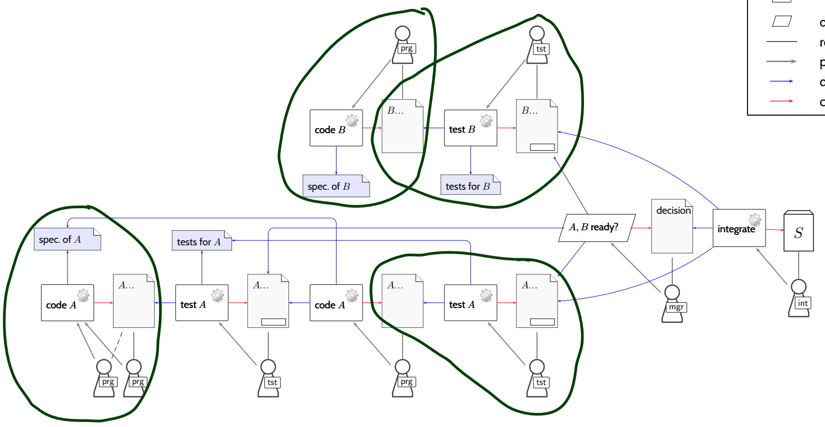
- role
- artefact
- activity
- decision point
- responsible
- participates
- depends on
- creates/modifies



- S consists of modules A and B.
- Assume: specifications and test cases for A and B were available.
- Person 🔵 coded B (according to spec.), then person 🟢 tested B (with test cases), no errors found.
- Person 🔵 coded A, with the help of person 🔴. Then person 🟢 tested A, some errors found.
- Person 🔴 fixed A, person 🔵 tested again, no errors found.
- A and B ready caused a positive decision, then person 🟡 integrated A and B and obtained S.

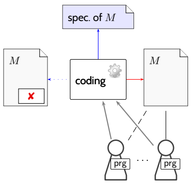
How the Plan for Creating S May Have Looked Like...

- role
- artefact
- activity
- decision point
- responsible
- participates
- depends on
- creates/modifies

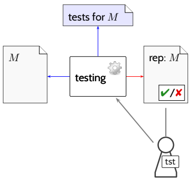


- S consists of modules A and B; specifications and test cases for A and B are available.
- Some prg codes B (according to spec.), then some tst tests B (with test cases), and creates test report.
- Some prg codes A, with the help of some prg. Then some tst tests A, and creates test report.
- If errors in A found, some single prg fixes A, some tst tests again, and creates test report.
- If A and B ready causes a positive decision, then some int integrates A and B and obtains S.

How the Plan for Creating S May Have Been Created...

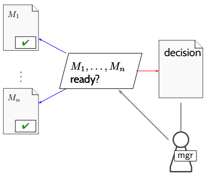


- A **software module** M has a responsible $\boxed{\text{prg}}$, any number of $\boxed{\text{prg}}$ may **help** with work on M.
- A **software module** M is created/modified by activity **coding**.
- Activity **coding** depends on a **specification** of M, and may consider a **positive test report** for M.
- The responsible $\boxed{\text{prg}}$ (and the helper $\boxed{\text{prg}}$'s) participate in activity **coding**.
- Activity **coding** is done, if M exists and there is a negative **test report** for M (all tests passed).

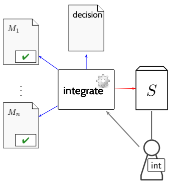


- A **test report** for a module M has a responsible $\boxed{\text{tst}}$.
- A **test report** is created/modified by activity **testing**.
- Activity **testing** depends on **software module** M and **tests** (in state "finished") for M.
- The responsible $\boxed{\text{tst}}$ participates in activity **testing**.
- Activity **testing** is done, if M exists and there is a negative **test report** for M (all tests passed).

How the Plan for Creating S May Have Been Created...

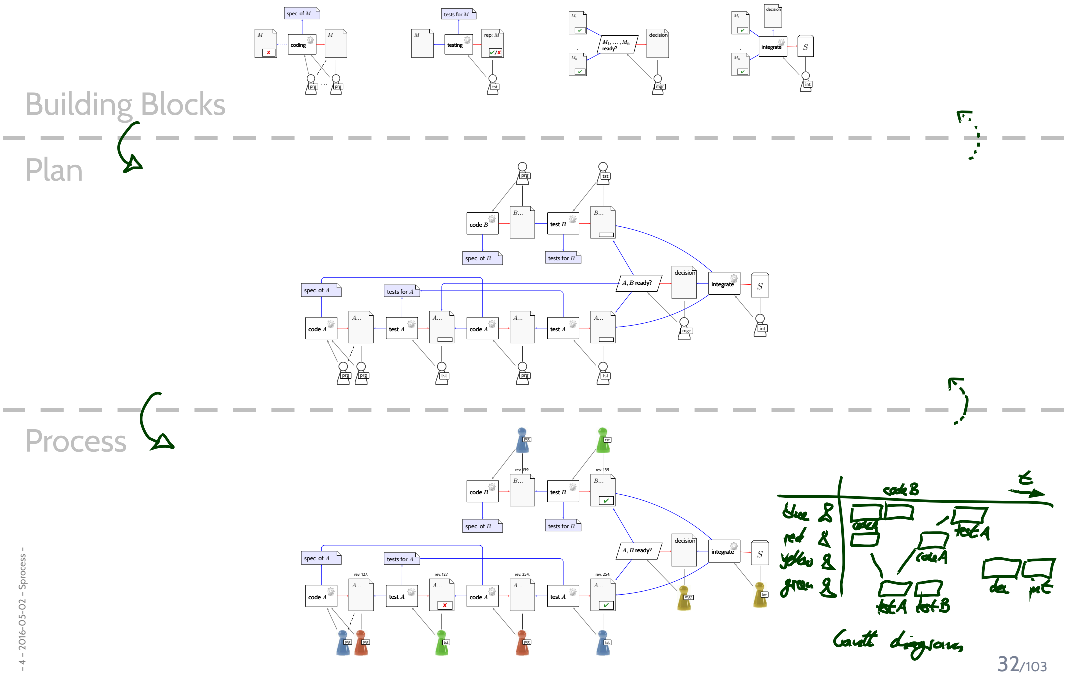


- A **ready decision** for a modules $M_1, \ldots, M_n$ has a responsible $\boxed{\text{mgr}}$.
- A **ready decision** is created/modified by decision point **ready?**.
- Decision point **ready?** depends on negative **test reports** for $M_1, \ldots, M_n$.
- The responsible $\boxed{\text{mgr}}$ participates in decision point **ready?**.
- Decision point **ready?** is done, if a positive decision exists.



- A **software** S has a responsible $\boxed{\text{int}}$. is created by integrating modules $M_1, \ldots, M_n$
- A **software** is created/modified by activity **integration**.
- Activity **integration** depends on **software modules** $M_1, \ldots, M_n$ in state "finished".
- The responsible $\boxed{\text{int}}$ participates in activity **integrate**.
- Activity **integration** is done, if S exists.

From Building Blocks to Process (And Back)



Building Blocks

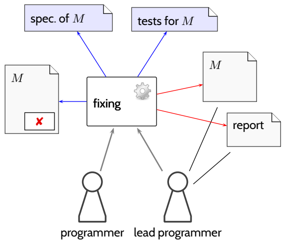
Plan

Process

Gantt diagrams

Building Blocks Can Be Arbitrarily Complicated

- **Example**: Distinguish **coding** and **fixing** software.



- If there is a negative test result for M,
- a leadprogrammer is responsible for fixing M,
- the programmer who was responsible for the initial version assist;
- fixing depends on the test cases, in addition to the specification of M,
- a report (analysis of the error, documentation of the fix) is created.

- Using such **building blocks**, the project management
  - can prescribe particular procedures,
  - analyse, which **roles** need to be filled in a project,
  - avoid to “forget” things.

Content

- **(Software) Project**
- **Project Management**
  - Excursion: Risk
- **Software Project Planning**
  - Costs and Deadlines
  - Tasks and Activities
  - People and Roles
- **Software Development Process**
  - Vocabulary: role, artefact, activity
  - Describing & prescribing processes
- **Procedure and Process Models**
  - Procedure Model Examples
    - The (in)famous Waterfall model
    - The famous Spiral model
    - Procedure classification
  - Process Model Examples
    - V-Modell, RUP
    - Agile (XP, Scrum)

Process vs. Procedure Models

Process Description and Reference Model

process description – documented expression of a set of activities performed to achieve a given purpose.

NOTE: A process description provides **an operational definition of the major components of a process**.

The description specifies, in a **complete, precise, and verifiable** manner, the requirements, design, behavior, or other characteristics of a process.

It also may include **procedures for determining** whether these provisions have been satisfied.

Process descriptions can be found at the **activity, project, or organizational level**. **IEEE 24765** (2010)

process reference model – a model comprising definitions of processes in a life cycle described in terms of process purpose and outcomes, together with an architecture describing the relationships between the processes. **IEEE 24765** (2010)

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')

    e.g., "waterfall model" (70s/80s).
  - (ii) **Organisational structure** – comprising requirements on
    - project management and responsibilities,
    - quality assurance,
    - documentation, document structure,
    - revision control.

  e.g., V-Modell, RUP, XP (90s/00s).

- In the literature, **process model** and **procedure model** are often used as synonyms; there is not universally agreed distinction.

Anticipated Benefits of Process Models

- **"economy of thought"**
  – don't re-invent principles.
- **quantification, reproducibility**
  – one can **assess the quality** of **how** products are created (→ CMMI).

  Identify weaknesses, learn from (bad) experience, improve the process.
- **fewer errors**
  – e.g., testing a module cannot be forgotten because the "ready" decision point depends on module with "test passed" flagged.
- **clear responsibilities**
  – fewer "I thought **you**'d fix the module!"

- **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.

Procedure Models

Procedure Model (?!): Code and Fix

> **Code and Fix** – denotes an approach, where coding and correction alternating with ad-hoc tests are the only consciously conducted activities of software development.
>
> **Ludewig & Lichter (2013)**

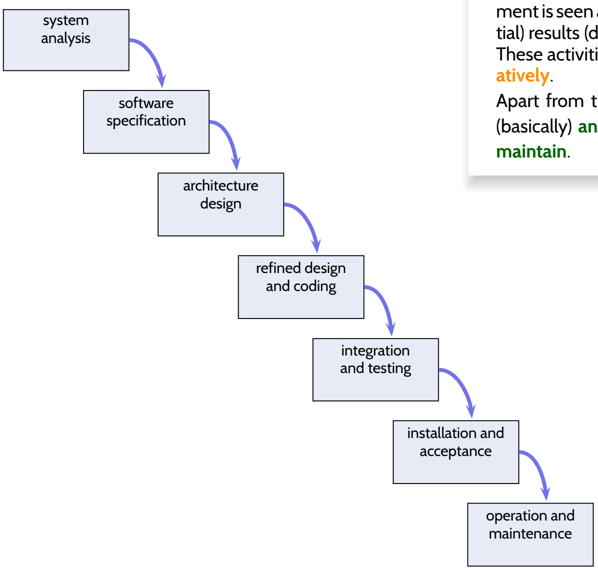
Advantages:

- Corresponds to our desire to "get ahead", to solve the stated problem quickly.
- The conducted activities (coding and ad-hoc testing) are easy.

Disadvantages:

- It is hard to plan the project, there are no rational/explicit decisions.
- It is hard to distribute work over multiple persons or groups. (→ responsibilities)
- If requirements are not stated, there is no notion of correctness (= meeting requirements).
- Tests are lacking expected outcome (otherwise, e.g., derived from requirements).
- Resulting programs often hard to maintain.
- Effort for maintenance high: most errors are only detected in operation.
- Important concepts and decisions are not documented, but only in the heads of the developers, thus hard to transfer.
- …

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)**

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.

ISO/IEC/IEEE (2010). *Systems and software engineering – Vocabulary*. 24765:2010(E).

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.

Thayer, R. H. (1997). *Tutorial – Software Engineering Project Management*. IEEE Society Press, revised edition.

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.