Softwaretechnik / Software-Engineering

Lecture 03: Procedure and Process Models

2015-04-30

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

- terms: project, (life) cycle, phase, milestone, role, costs
- project management goals and activities; (cost) estimation: the Delphi method

This Lecture:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - what are the basic conceps of COCOMO, function points?
  - estimate this project using COCOMO, function points
  - what is a process? a model? a process model?
  - give an example for role, activity, artefact, decision point
  - what is a prototype? what is evolutionary, incremental, iterative?
  - what's the fundamental idea of the spiral model? where's the spiral?
  - what is the difference between procedure and process model?
  - what are the constituting elements of "V-Modell XT"? what project types does it support, what is the consequence?
  - what is tailoring in the context of "V-Modell XT"?
  - what are examples of agile process models? what are the agile principles?
  - describe XP, Scrum
- **Content:** cost estimation cont'd (COCOMO, FP); procedure and process models

Cost Estimation Cont’d

Algorithmic Estimation: COCOMO

- **Co**nstructive **Co**st **Mo**del:
  Formulae which fit a huge set of archived project data (from the late 70's).
- Flavours:
  - COCOMO 81 (Boehm, 1981): basic, intermediate, detailed
  - COCOMO II (Boehm et al., 2000)
- All based on estimated program size S measured in DSI or kDSI (thousands of Delivered Source Instructions).
- Factors like security requirements or experience of the project team are mapped to values for parameters of the formulae.

- COCOMO examples:
  - textbooks like Ludewig and Lichter (2013) (most probably made up)
  - an exceptionally large example:
    COCOMO 81 for the Linux kernel (Wheeler, 2006) (and follow-ups)

COCOMO 81

Software Project Type	Characteristics of the Type				a	b	c	d
	Size	Innovation	Deadlines/ Constraints	Dev. Environment				
Organic	Small (<50 KLOC)	Little	Not tight	Stable	2.4	1.05	2.5	0.38
Semi-detached	Medium (<300 KLOC)	Medium	Medium	Medium	3.0	1.12	2.5	0.35
Embedded	Large	Greater	Tight	Complex HW/ Interfaces	3.6	1.20	2.5	0.32

Basic COCOMO:

$$E \text{ (effort required)} = a(S/kDSI)^b \quad \text{[person-months]}$$

$$TDEV \text{ (time to develop)} = cE^d \quad \text{[months]}$$

Intermediate COCOMO:

$$E \text{ (effort required)} = Ma(S/kDSI)^b \quad \text{[person-months]}$$

… where

$$M = \mathit{RELY} \cdot \mathit{CPLX} \cdot \mathit{TIME} \cdot \mathit{ACAP} \cdot \mathit{PCAP} \cdot \mathit{LEXP} \cdot \mathit{TOOL} \cdot \mathit{SCED}$$

factor	very low	low	normal	high	very high	extra high
RELY required software reliability	0.75	0.88	1	1.15	1.40	
CPLX product complexity	0.70	0.85	1	1.15	1.30	1.65
TIME execution time constraint			1	1.11	1.30	1.66
ACAP analyst capability	1.46	1.19	1	0.86	0.71	
PCAP programmer capability	1.42	1.17	1	0.86	0.7	
LEXP programming language experience	1.14	1.07	1	0.95		
TOOL use of software tools	1.24	1.10	1	0.91	0.83	
SCED required development schedule	1.23	1.08	1	1.04	1.10	

COCOMO II

Consists of

- **Application Composition Model** — configuration dominates programming
- **Early Design Model** — adaption of Function Point approach (in a minute)
- **Post-Architecture Model** — like COCOMO 81, needs architecture defined

$$E = 2.94 \cdot \left(\frac{S}{kSLOC}\right)^{X} \cdot M$$

where $X = PREC + FLEX + RESL + TEAM + PMAT$.

So-called **scaling factors** (although not used as scalar):

- Precedentness (PREC) — experience with similar projects
- Development flexibility (FLEX) — development process fixed by customer
- Architecture/risk resolution (RESL) — risk management, architecture size
- Team cohesion (TEAM) — communication effort in team
- Process maturity (PMAT) — see CMM

COCOMO II Cont'd

M now depends on:

group	factor	description
Product factors	RELY	required software reliability
	DATA	size of database
	CPLX	complexity of system
	RUSE	degree of development of reusable components
	DOCU	amount of required documentation
Platform factors	TIME	execution time constraint
	STOR	memory consumption constraint
	PVOL	stability of development environment
Team factors	ACAP	analyst capability
	PCAP	programmer capability
	PCON	continuity of involved personnel
	APEX	experience with application domain
	PLEX	experience with development environment
	LTEX	experience with programming language(s) and tools
Project factors	TOOL	use of software tools
	SITE	degree of distributedness
	SCED	required development schedule

Algorithmic Estimation: Function Points

	Complexity			Sum
Type	low	medium	high	
input	____ $\cdot 3 =$	____ $\cdot 4 =$	____ $\cdot 6 =$	
output	____ $\cdot 4 =$	____ $\cdot 5 =$	____ $\cdot 7 =$	
query	____ $\cdot 3 =$	____ $\cdot 4 =$	____ $\cdot 6 =$	
user data	____ $\cdot 7 =$	____ $\cdot 10 =$	____ $\cdot 15 =$	
reference data	____ $\cdot 5 =$	____ $\cdot 7 =$	____ $\cdot 10 =$	
Unadjusted function points		UFP		
Value adjustment factor		VAF		
Adjusted function points		AFP = UFP · VAF		

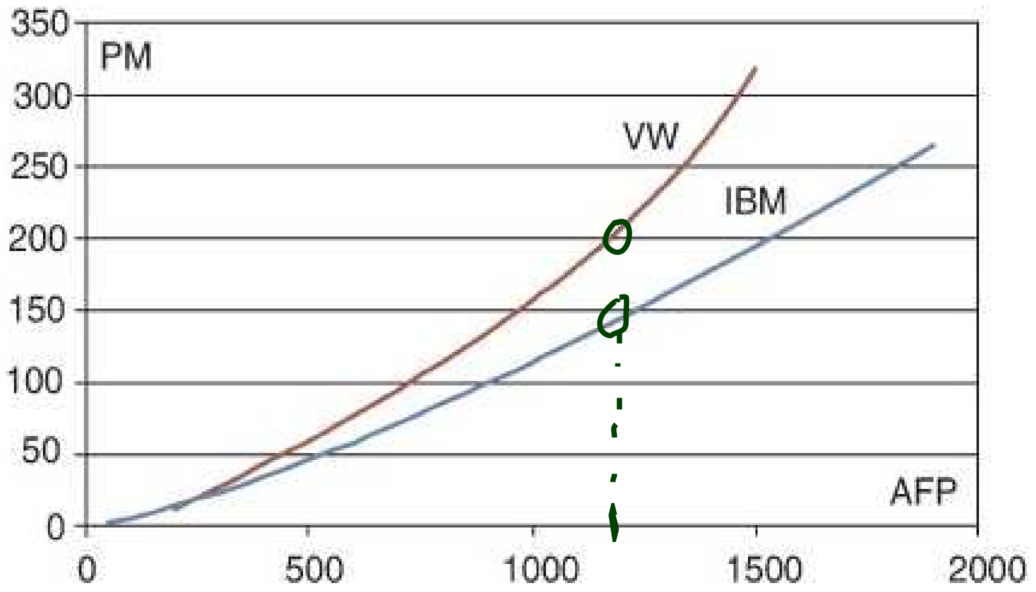
$0.65 \le$ ≤ 1.35

$$VAF = 0.65 + \frac{\sum_{i=1}^{14} GSC_i}{100},$$

$$0 \le GSC_i \le 5,$$

Algorithmic Estimation: Function Points

Type	low		
input	____$\cdot 3 =$		
output	____$\cdot 4 =$		
query	____$\cdot 3 =$		
user data	____$\cdot 7 =$		
reference data	____$\cdot 5 =$		
Unadjusted function points		UFP	
Value adjustment factor		VAF	
Adjusted function points		AFP = UFP · VAF	



IBM and VW curve for the conversion from AFPs to PM according to (Noth and Kretzschmar, 1984) and (Knöll and Busse, 1991).

$$VAF = 0.65 + \frac{\sum_{i=1}^{14} GSC_i}{100},$$

$$0 \leq GSC_i \leq 5,$$

COCOMO vs. Function Points

(Ludewig and Lichter, 2013) says:

- function point approach used in practice, in particular commercial software (business software?)
- COCOMO tends to overestimate in this domain; needs to be adjusted by corresponding factors

In The End. . .

. . . it's **experience**, **experience**, **experience**.

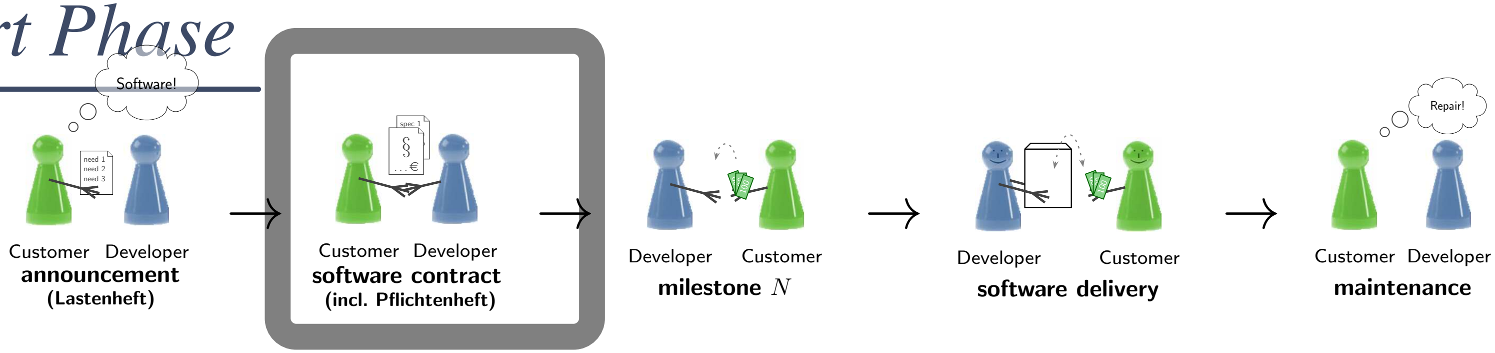
"estimate, document, estimate better" (Ludewig and Lichter, 2013)

Suggestion: start to explicate your experience **now**.

- Take notes on your projects (Softwarepraktikum, Bachelor Projekt, Master Bacherlor's Thesis, Team Projekt, Master's Thesis, . . .)
  - timestamps
  - size of program created
  - number of errors found
  - number of pages written
  - . . . (more measures/metrics: later)
- Try to identify factors: what hindered productivity, what boosted productivity, . . .
- Which detours and mistakes were avoidable in hindsight? How?

Okay, estimation is done, and now. . . ?

The Start Phase



- The phase between **customer's decision** to have software developed and **start of project** is called **start phase**. Start of project: **project plan** is installed.

- **Planning activities**;
  - Clarify extension of deliverables and scope of work.
  - Identify risks.
  - Create budget and time plans.
  - Define and structure tasks.
  - Define who reports what to whom, inside team and towards customer.
  - Estimate needed personnel (number and qualifications).
  - Assign tasks to roles, and roles to personnel.
  - Provide necessary support (e.g. infrastructure).

Some Final Project Management Wisdom...

Most software projects are successful (cf. SUCCESS survey (Buschermöhle et al., 2006)); if they fail, they tend to fail due to:

- insufficient education (in particular project management)
- unrealistic expectations of higher management
- implicit customer expectations
- behaviour of team members
- own expectations (Ludewig and Lichter, 2013)

Rules of behaviour for successful project management:

(i) Think people first, the business is second. All a business is, is its people. Take care of them.

(ii) Establish a clear definition of your project's development cycle and stick to it.

(iii) Emphasize the front-end of the project so that the rear-end won't be dragging.

(iv) Establish baselines early and protect them from uncontrolled change.

(v) State clearly the responsibilities of each person on the project.

(vi) Define a system of documents clearly and early.

(vii) Never give an estimate or an answer you don't believe in.

(viii) Never forget Rule (i). *(Metzger, 1981)*

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)**

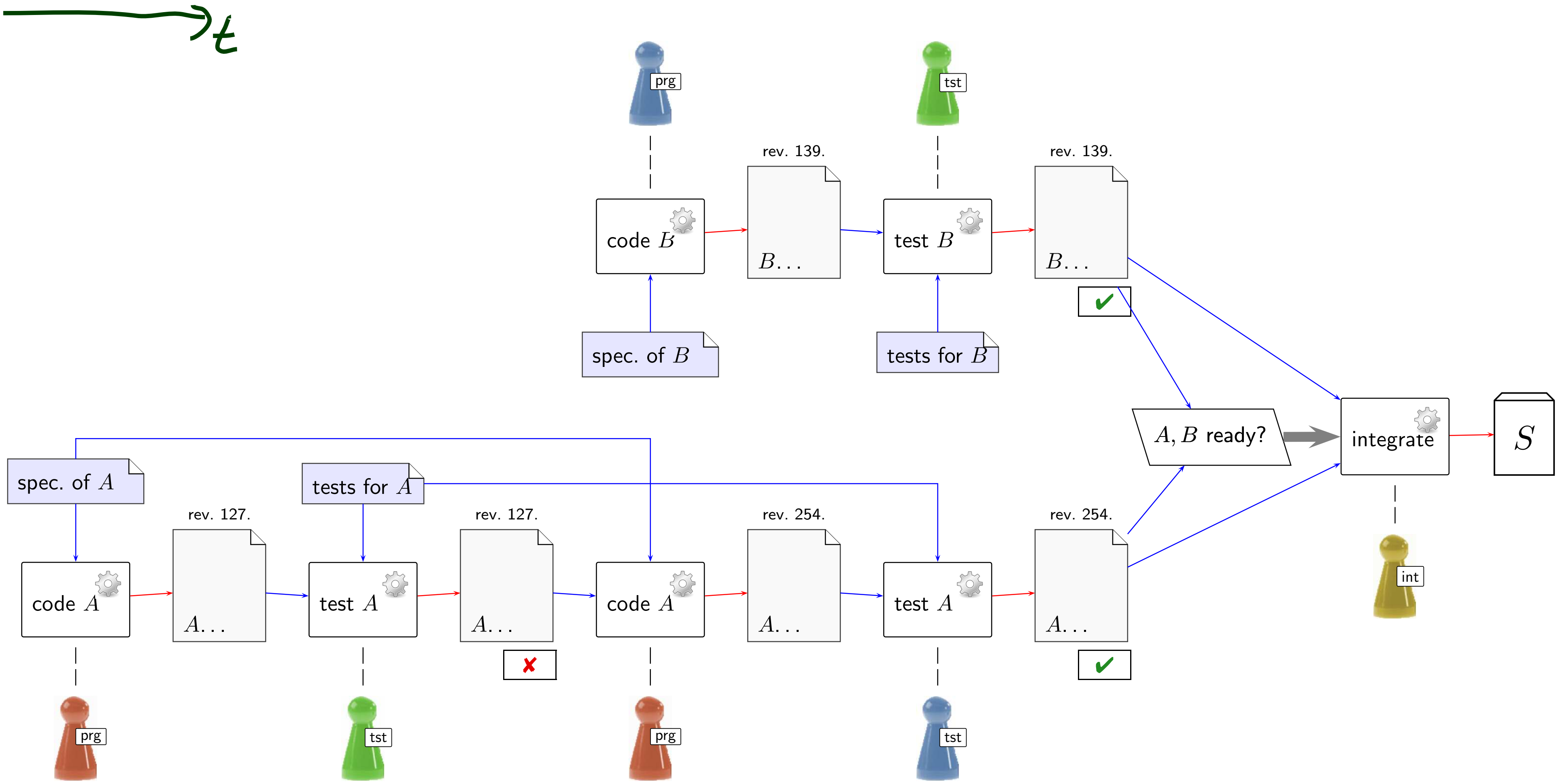
- The process of a software development project may be
  - implicit,
  - informally agreed on, or
  - prescribed (by a **procedure** or **process model**).
- But: each software development **has** a process!

Example Process

Assume

- The desired software product S is obtained from **two modules** A and B.
- There is a **specification** for both A and B.
- There is a **test plan** for both A and B.
- A and B, after having been tested, are **integrated** to obtain S.

One Example Process Creating S



- Can we describe (**model**) the underlying principles?
- In terms of roles, activities, artefacts — abstracting from A, B, and particular people.

Excursion: Model

Model

Definition. [Folk] A **model** is an abstract, formal, mathematical representation or description of structure or behaviour of a (software) system.

Definition. (Glinz, 2008, 425)
A **model** is a concrete or mental image (Abbild) of something or a concrete or mental archetype (Vorbild) for something.

Three properties are constituent:

(i) the **image attribute** (Abbildungsmerkmal), i.e. there is an entity (called original) whose image or archetype the model is,

(ii) the **reduction attribute** (Verkürzungsmerkmal), i.e. only those attributes of the original that are relevant in the modelling context are represented,

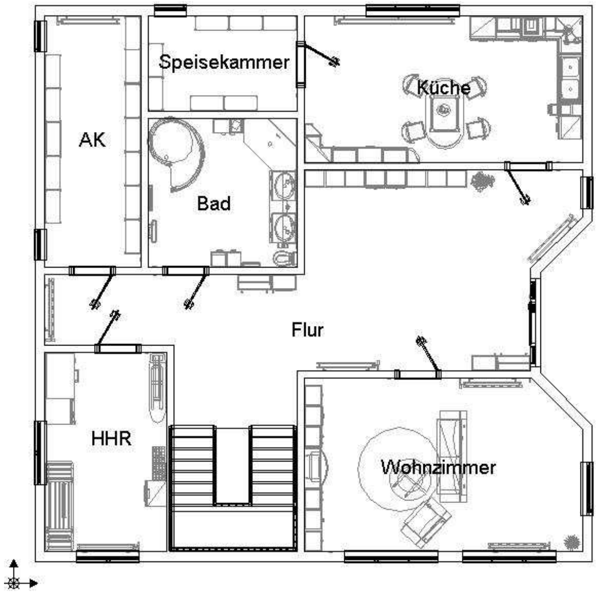
(iii) the **pragmatic attribute**, i.e. the model is built in a specific context for a specific purpose.

Model Example: Floorplan

1. Requirements

- Shall fit on given piece of land.
- Each room shall have a door.
- Furniture shall fit into living room.
- Bathroom shall have a window.
- Cost shall be in budget.

2. Designmodel



http://wikimedia.org (CC nc-sa 3.0, Ottoklages)

3. System

http://wikimedia.org
(CC nc-sa 3.0, Bobthebuilder82)

Observation: Floorplan **abstracts** from certain system properties, e.g. . . .

- kind, number, and placement of bricks,
- subsystem details (e.g., window style),
- water pipes/wiring, and
- wall decoration

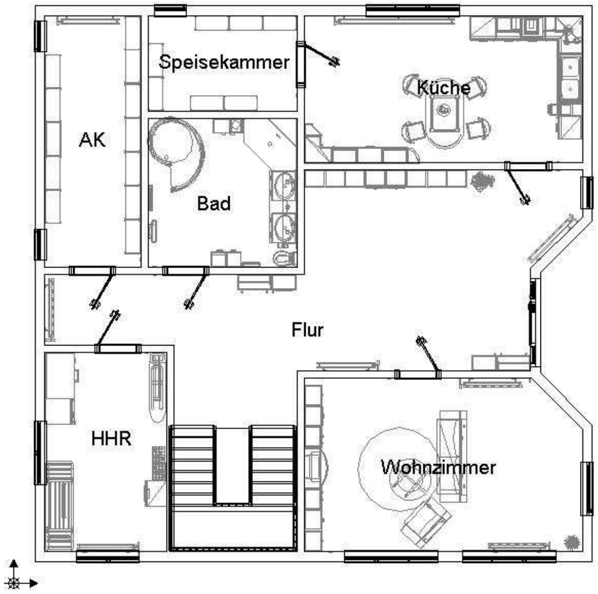
→ architects can efficiently work on appropriate level of abstraction

Model Example: Floorplan

1. Requirements

- Shall fit on given piece of land.
- Each room shall have a door.
- Furniture shall fit into living room.
- Bathroom shall have a window.
- Cost shall be in budget.

2. Designmodel



http://wikimedia.org (CC nc-sa 3.0, Ottoklages)

3. System

http://wikimedia.org (CC nc-sa 3.0, Bobthebuilder82)

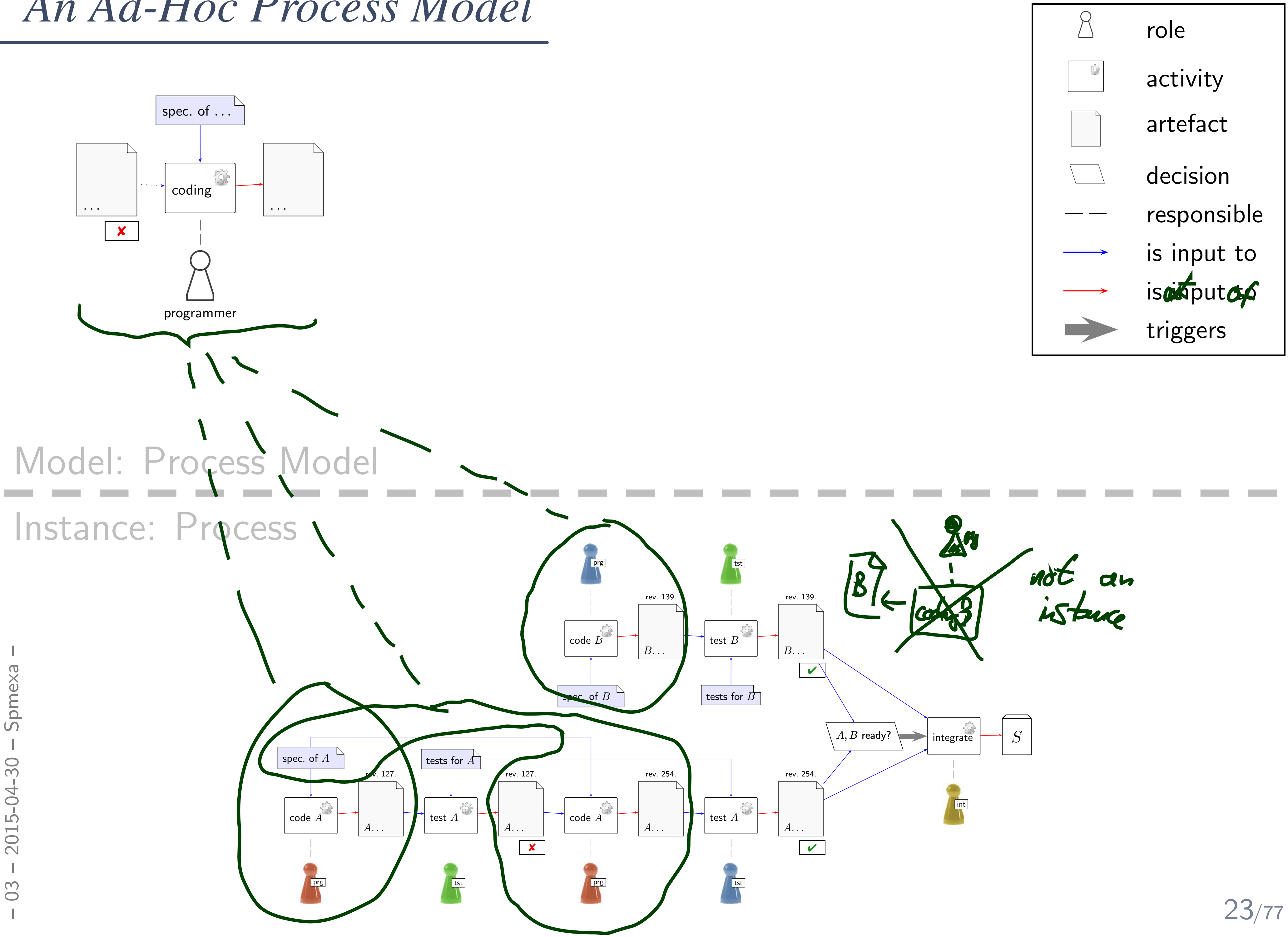
Observation: Floorplan **preserves**/**determines** certain system properties, e.g.,

- house and room extensions (to scale),
- presence/absence of windows and doors,
- placement of subsystems (such as windows).

→ find design errors before building the system (e.g. bathroom windows)

Process and Procedure Models

An Ad-Hoc Process Model

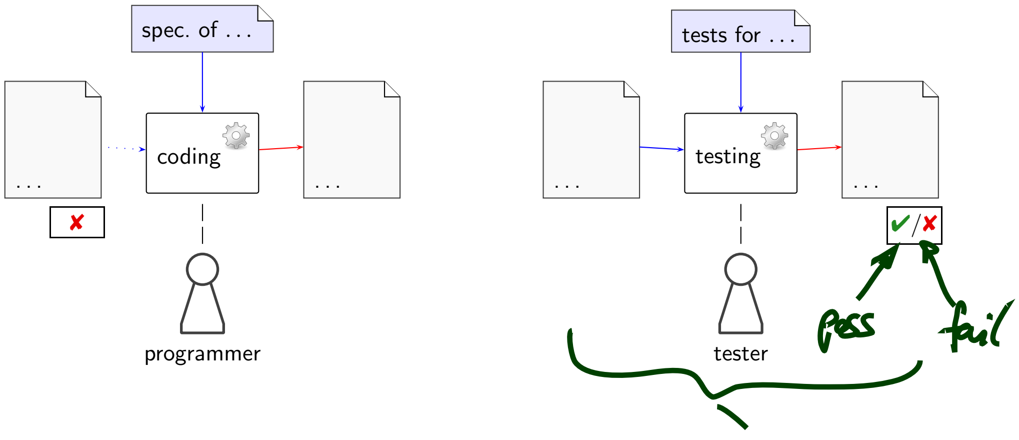


♙	role
□	activity
▯	artefact
▱	decision
— —	responsible
→	is input to
→	is ~~in~~outputto ~~to~~of
➞	triggers

Model: Process Model

Instance: Process

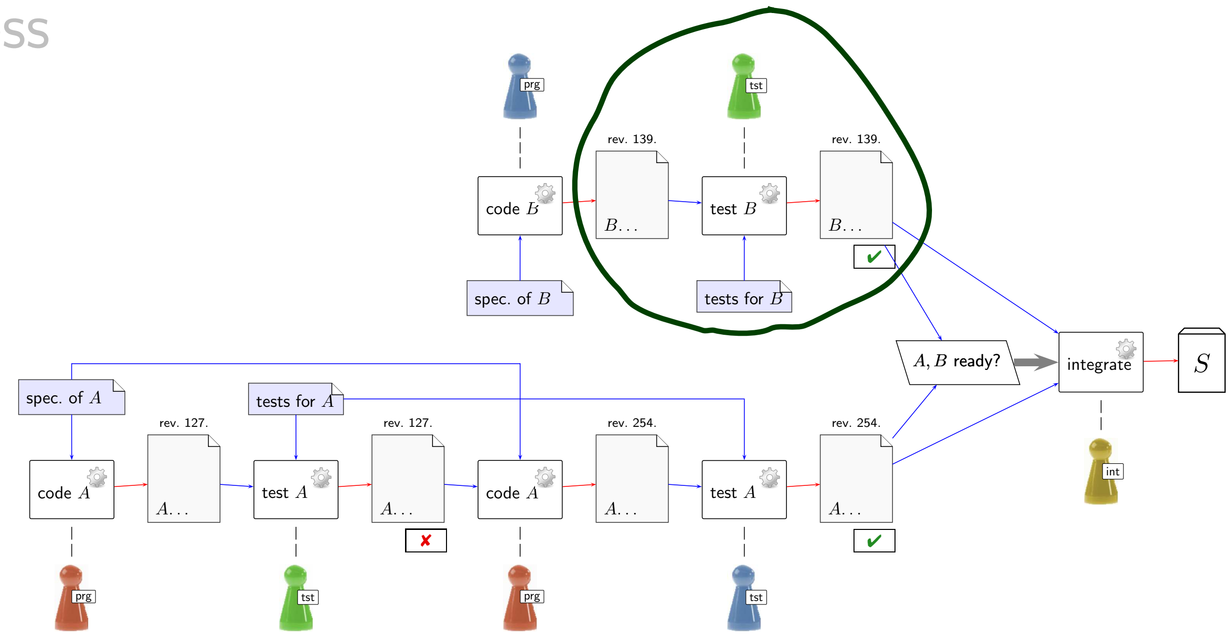
An Ad-Hoc Process Model



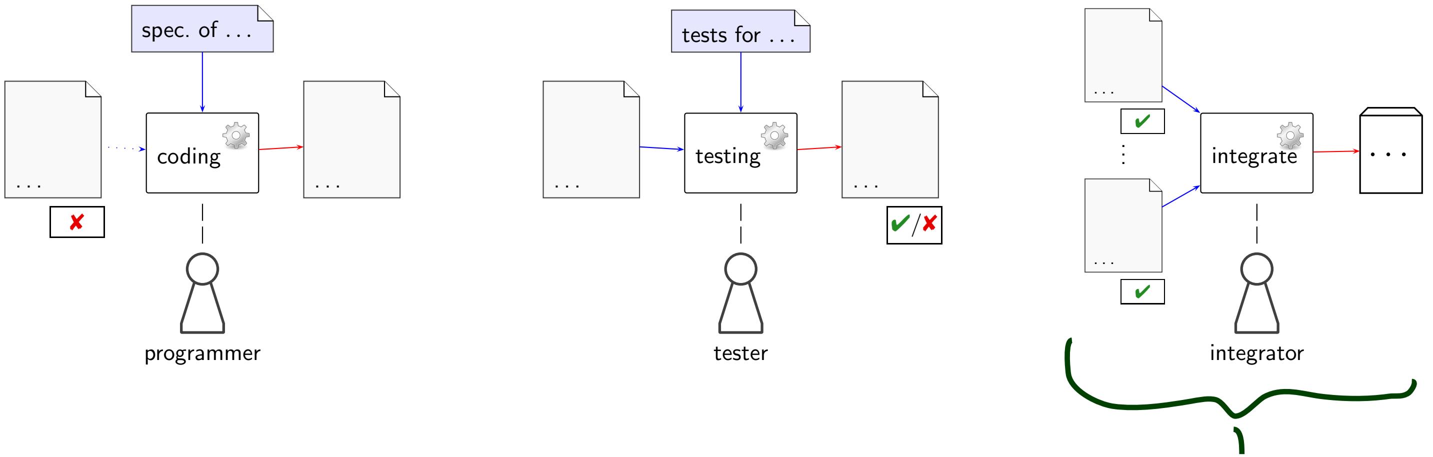
- role
- activity
- artefact
- decision
- responsible
- is input to
- is ~~in~~output ~~to~~ of
- triggers

Model: Process Model

Instance: Process



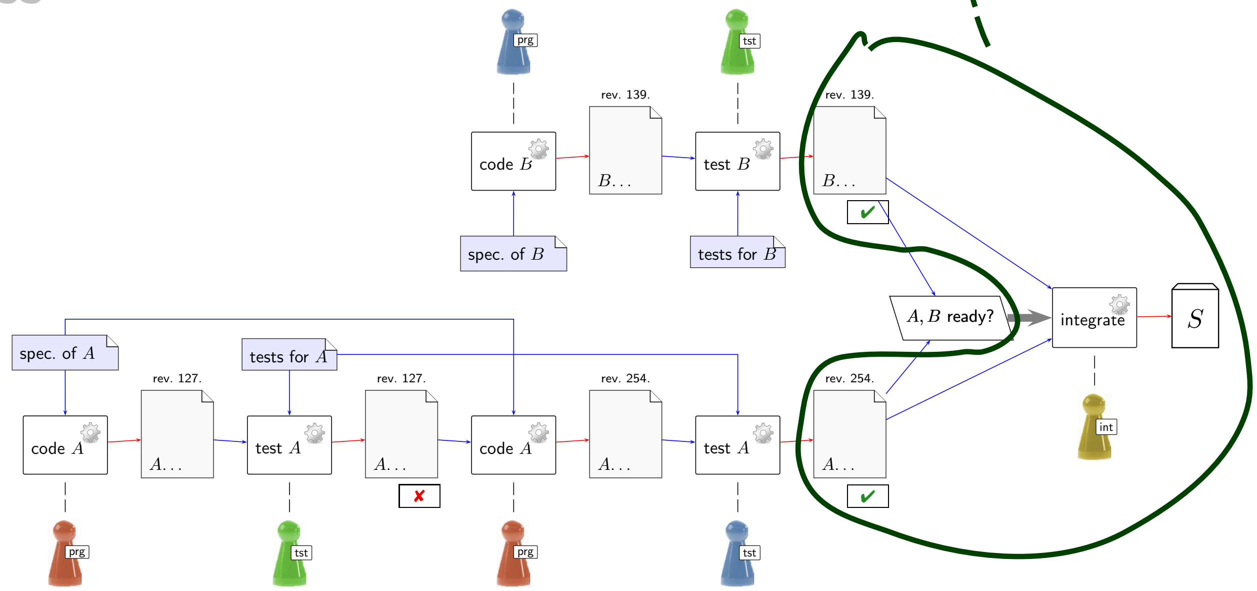
An Ad-Hoc Process Model



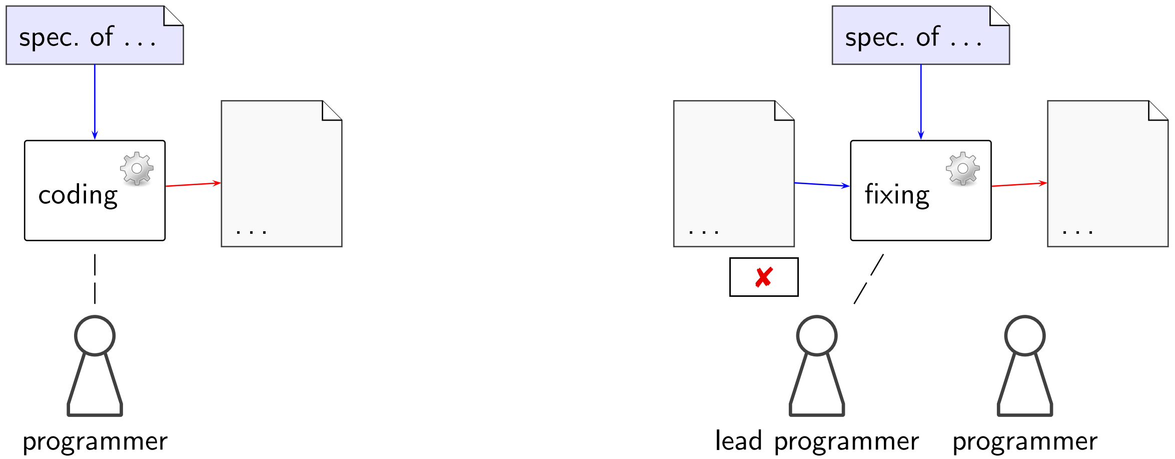
♙	role
⚙	activity
🗎	artefact
▱	decision
— —	responsible
→	is input to
→	is ~~in~~output ~~to~~ of
➡	triggers

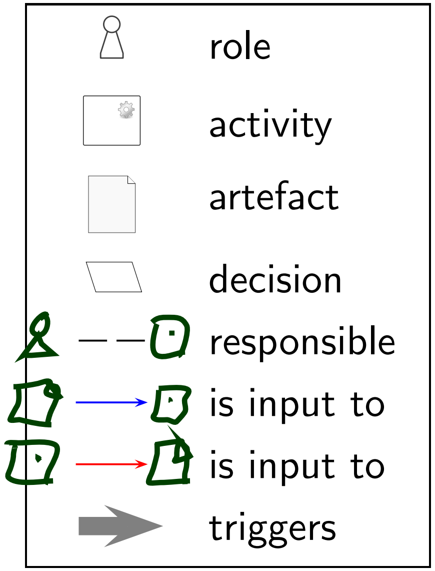
Model: Process Model

Instance: Process



Ad-Hoc Process Model Refined

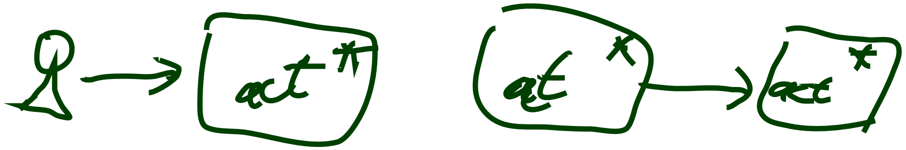




- an (ordinary) **programmer** creates the initial version according to the **specification**
- if testing discovers an error, the **lead developer** is responsible for fixing it, a programmer supports the activity

NOT A REFINEMENT

No:



Process and Procedure Models: The Bigger Picture

Anticipated Benefits...

... of process models:

- **“economy of thought”** — don’t re-invent principles
- one can **assess the quality** of **how** products are created (→ CMMI)
  - identify weaknesses, learn from (bad) experience, improve the process
- **fewer errors** — testing a module cannot be forgotten because integration depends on module with “test passed” flagged
- **clear responsibilities** — “I thought **you**’d fix the module!”

Process model is a topic as **emotional** as **progamming language**, **editor**, **operating system**, **soccer club**, ... — let’s try to keep the discussion objective.

- **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 new/changed process model makes matters better or worse (→ metrics)
  - but: customer may require a certain process model

Process vs. Procedure Model

- In the literature, **process model** and **procedure model** are often used as synonyms.

- (Ludewig and Lichter, 2013) propose to distinguish:
  - **process model** ('Prozessmodell') — 90s/00s "RUP, XP": comprises
    - **procedure model** ('Vorgehensmodell') — 70s/80s "waterfall model"
    - **organisational structure** comprising
      - requirements on project management and responsibilities,
      - requirements on quality assurance,
      - requirements on documentation, document structure,
      - requirements on revision control.

- Analogy: theatre (Ludewig and Lichter, 2013)
  - **procedure model**: script
  - **process model**: staging (script **plus** cast, costumes, stage setting)

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)

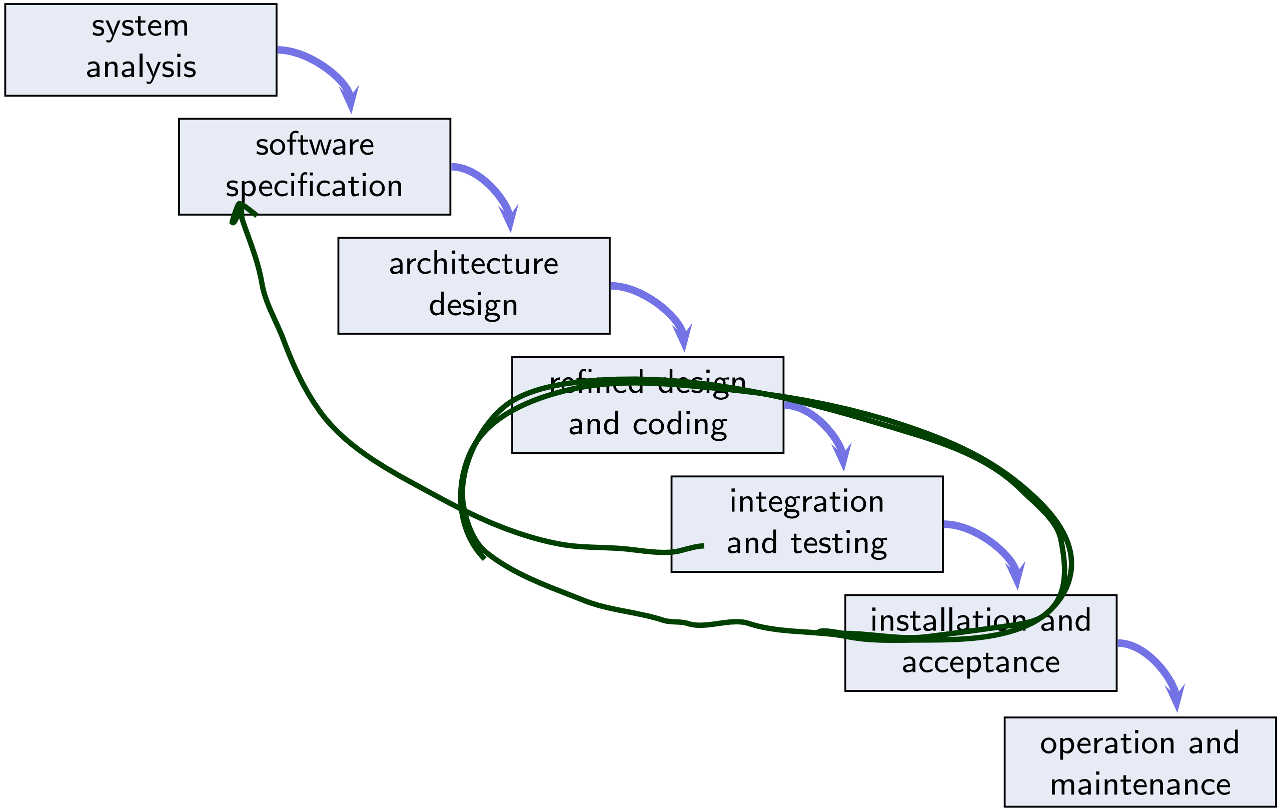
Advantages:

- Corresponds to our desire to “get ahead”, solve the stated problem quickly.
- There are executable programs early.
- The conducted activities (coding and ad-hoc testing) are easy.

Disadvantages:

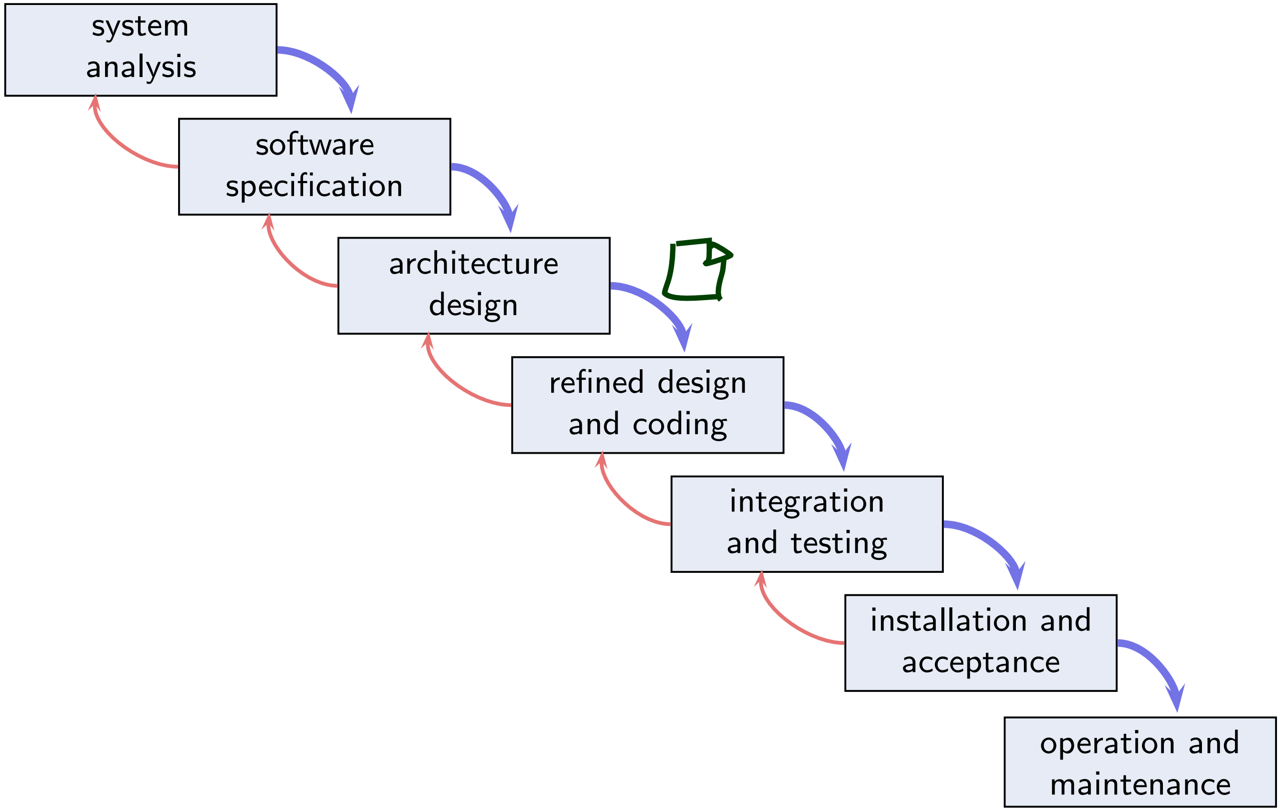
- It is **hard to plan** the project, there are no rational/explicit decisions.
- It is **hard to distribute** work over multiple persons or groups.
- If requirements are not stated, there is **no notion of correctness** (= meeting requirements).
- Tests are **lacking expected outcome** (otherwise, e.g., 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.

Procedure Model: The Waterfall Model



(Rosove, 1967)

Procedure Model: The Waterfall Model

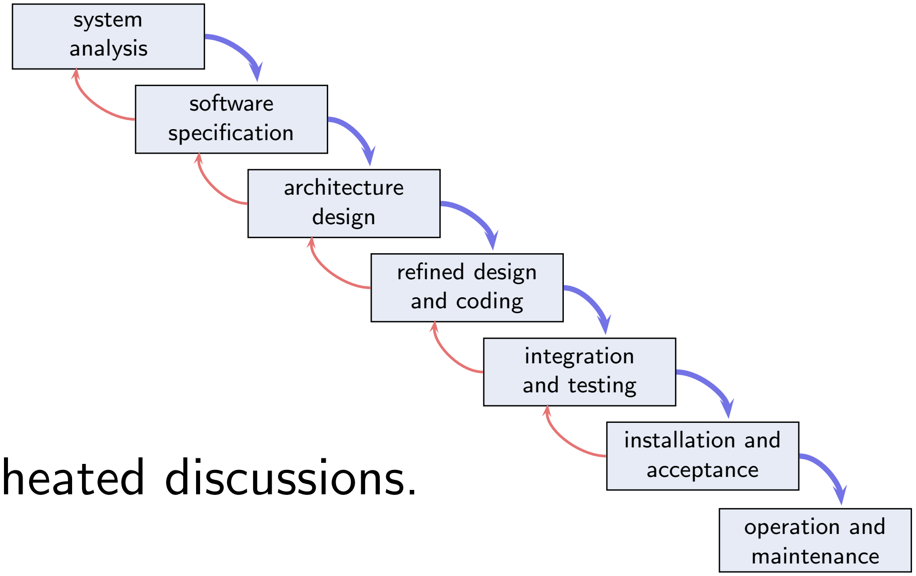


(Rosove, 1967)

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



- The waterfall model has been subject of heated discussions.
- With 40 years of distance:
  - it gives room for numerous interpretations:
    a very abstract model, hardly usable as a "template" for real projects,
  - in particular the cycles (and the lack of milestones) makes is hard for project management to **assess a project's process**
  - still there's some truth in it:

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

    That may have been news back then. . . (→ software crisis)
  - Given the computers and software of that time (no* personal computers, no* smart phones, no* web-shops, no* computer games, no* automotive software, no* . . .).

A Classification of Software

Lehmann (Lehman, 1980; Lehman and Ramil, 2001) distinguishes three classes of software (my interpretation, my examples):

- **S-programs**: solve mathematical, abstract problems; can exactly (in particular formally) be specified; tend to be small; can be developed once and for all.

  Examples: sorting, compiler (!), compute π or $\sqrt{\cdot}$, cryptography, textbook examples, ...

- **P-programs**: solve problems in the real world, e.g. read sensors and drive actors, may be in feedback loop; specification needs **domain model** (cf. Bjørner (2006), "A tryptich software development paradigm"); formal specification (today) possible, in terms of domain model, yet tends to be expensive

  Examples: cruise control, autopilot, traffic lights controller, plant automatisation, ...

- **E-programs**: embedded in socio-technical systems; in particular involve humans; specification often not clear, not even known; can grow huge; delivering the software induces new needs

  Examples: basically everything else; word processor, web-shop, game, smart-phone apps, ...

Non-linear Procedure Models

	Analysis of Requirements	Use on Target System	Defined Steps	Preliminary Results Used	Complete Plan
Rapid Prototyping	■				
Evolutionary Development	▦	■			
Iterative Development	░	■	▦	░	
Incremental Development	░	■	■	▦	░
. . .					

- ■ yes
- ▦ to some amount
- ░ to a low amount

(Rapid) 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)**

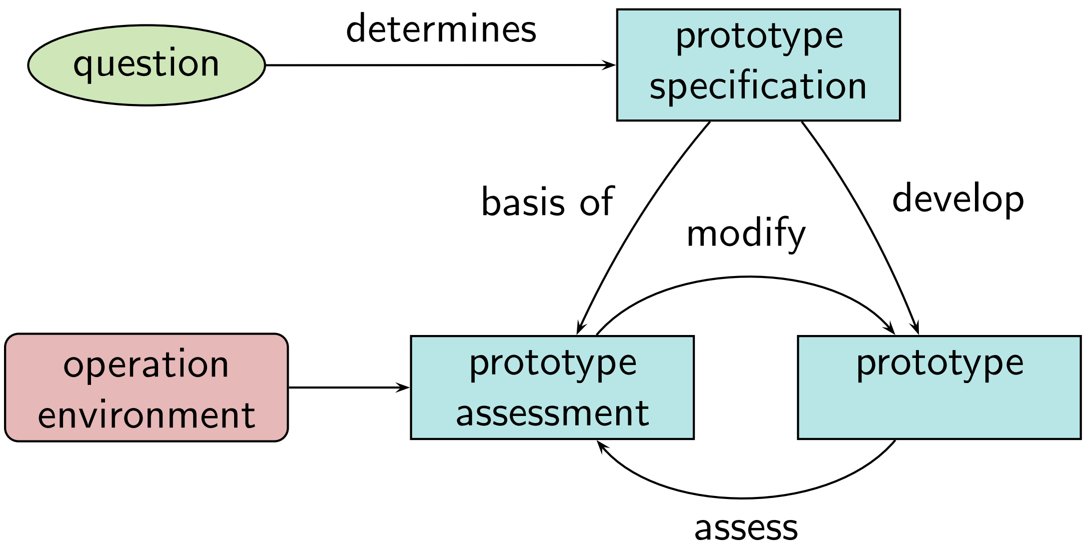
rapid prototyping — A type of prototyping in which emphasis is placed on developing protoypes early in the development process to permit early feedback and analysis in support of the development process. **IEEE 610.12 (1990)**

- A prototype realises **selected aspects** of the final system.
- A prototype is **checked/tested and assessed**, at best involving the customer.
- Sometimes distinguished from **prototype** (which is executable):
  mock-up, like a drawing of the intended graphical user interface, or game menu screens and structure simulated with HTML pages (→ requirements engineering)

Prototyping

Approach:

- clarify: which purpose does the prototype have, what are the open questions?
- which persons (roles) participate in **development** and, most important, **assessment** of the prototype?
- what is the time/cost budget for prototype development?



(Ludewig and Lichter, 2013)

Kinds of Prototypes and Goals of Prototyping

- **demonstration prototype** ('Demonstrationsprototyp')
  "throw away products" to demonstrate look-and-feel or potential usage of proposed product; can be "quick and dirty"
- **functional prototype** ('Funktionaler Prototyp')
  usually regarding (graphical) user interface; maybe many separate prototypes for specific questions
- **lab sample** ('Labormuster')
  addresses open technical questions, proof-of-concept; need not be part of the final system
- **pilot system** ('Pilotsystem')
  functionality and quality are at least sufficient for a (temporary) use in the target environment

Floyd Taxonomy:

- **explorative prototyping** — support analysis
- **experimental prototyping** — develop new techonology
- **evolutionary prototyping** — the product is the last prototype

(categories may overlap)

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.

Bjørner, D. (2006). *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Springer-Verlag.

Boehm, B. W. (1981). *Software Engineering Economics*. Prentice-Hall.

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

Boehm, B. W., Horowitz, E., Madachy, R., Reifer, D., Clark, B. K., Steece, B., Brown, A. W., Chulani, S., and Abts, C. (2000). *Software Cost Estimation with COCOMO II*. Prentice-Hall.

Buschermöhle, R., Eekhoff, H., and Josko, B. (2006). success – Erfolgs- und Misserfolgsfaktoren bei der Durchführung von Hard- und Softwareentwicklungsprojekten in Deutschland. Technical Report VSEK/55/D.

Glinz, M. (2008). Modellierung in der Lehre an Hochschulen: Thesen und Erfahrungen. *Informatik Spektrum*, 31(5):425–434.

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

Knöll, H.-D. and Busse, J. (1991). *Aufwandsschätzung von Software-Projekten in der Praxis: Methoden, Werkzeugeinsatz, Fallbeispiele*. Number 8 in Reihe Angewandte Informatik. BI Wissenschaftsverlag.

Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE Special Issue on Software Engineering*, 68(9):1060–1076.

Lehman, M. M. and Ramil, J. F. (2001). Rules and tools for software evolution planning and management. *Annals of Software Engineering*, 11(1):15–44.

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

Metzger, P. W. (1981). *Managing a Programming Project*. Prentice-Hall, 2 edition.

Noth, T. and Kretzschmar, M. (1984). *Aufwandsschätzung von DV-Projekten, Darstellung und Praxisvergleich der wichtigsten Verfahren*. Springer-Verlag.

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.

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

Wheeler, D. A. (2006). Linux kernel 2.6: It's worth more!

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