2. Software Development as Engineering Activity

Prof. Dr. rer. nat. habil. Uwe Aßmann

Institut für Software- und Multimediatechnik

Lehrstuhl Softwaretechnologie

Fakultät für Informatik

Technische Universität Dresden

http://st.inf.tu-dresden.de/teaching/swt2

2015-0.9, 16.10.15

Obligatory Reading

- Balzert Introduction
- Maciaszek/Liong Chap. 1
- Ghezzi Chap 5+7 or
- Pfleeger Chap 2+4
- The ACM/IEEE Code of Ethics for Software Engineers:
  - https://www.acm.org/about/se-code
  - https://www.computer.org/cms/Computer.org/Publications/code-of-ethics.pdf
- Wolfgang Hesse, Heinrich C. Mayr. Modellierung in der Softwaretechnik: eine Bestandsaufnahme. Informatik Spektrum 31(5), Springer-Verlag 2008
  - DOI 10.1007/s00287-008-0276-7
  - This explains the concept of a model in general
- Ed Seidewitz. What models mean. IEEE Software, 20:26-32, September 2003.
  - http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1231147&tag=1
  - This explains the different of prescriptive and descriptive modeling

References

- M. Pidd. Tools for Thinking. Modeling in Management Science. Wiley. Gives a good overview on modeling in general (soft and hard models)
- www.omg.org/mda Model driven architecture® is a process that structures refinement-based development, using UML
- Favre's papers on egyptology (i.e., modeling):
  - Jean-Marie Favre. Foundations of model (driven) (reverse) engineering: Models - episode I: Stories of the fidus papyrus and of the solarus. In Jean Bezivin and Reiko Heckel, editors, Language Engineering for Model-Driven Software Development, number 04101 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.
  - Jean-Marie Favre. Foundations of meta-pyramids: Languages vs. metamodels- episode II: Story of thotus the baboon1. In Jean Bezivin and Reiko Heckel, editors, Language Engineering for Model-Driven Software Development, number 04101 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.
- J.R. Abrial, Stephan Hallerstede. Refinement, decomposition, and instantiation of discrete models: Application to Event-B. Fundamenta Informaticae, 2007
  - http://dl.acm.org/citation.cfm?id=1365974&CFID=49627514&CFTOKEN=73132377

Horton about Engineering

- The profession of engineering--which, by the way, is merely the adapting of discoveries in science and art to the uses of mankind--is a peculiarly isolated one.
- But very little is known about it among those outside of the profession. Laymen know something about law, a little about medicine, quite a lot--nowadays--about metaphysics. But laymen know nothing about engineering. Indeed, a source of common amusement among engineers is the peculiar fact ***that the average layman cannot differentiate between the man who runs a locomotive and the man who designs a locomotive*****.** In ordinary parlance both are called engineers.
- Yet there is a difference between them--a difference as between day and night. For one merely operates the results of the creative genius of the other. ***This almost universal ignorance as to what constitutes an engineer serves to show to what broad extent the profession of engineering is isolated.***
- Charles M. Horton. Title: Opportunities in Engineering. 1920, by Harper & Brothers. http://www.gutenberg.org/ebooks/24681

Riddle

"Ironring2005" by Mobilefolk - Own work. Licensed under Public Domain via Commons - https://commons.wikimedia.org/wiki/File:Ironring2005.JPG#/media/File:Ironring2005.JPG

Scenario of Running Example

- You are a project manager in Hamann/Becker Car Radios, Inc, Karlsruhe, Germany
- Your boss comes into your office and says:

"Our competitor Smith Car Radios has a new satellite radio. Their sales are growing, and our customers demand it, too. How quickly can you deliver me a satellite radio?"

First Ideas

- How many people?
  - do we have the right ones?
- Which milestones (deadlines)?
- How many resources?

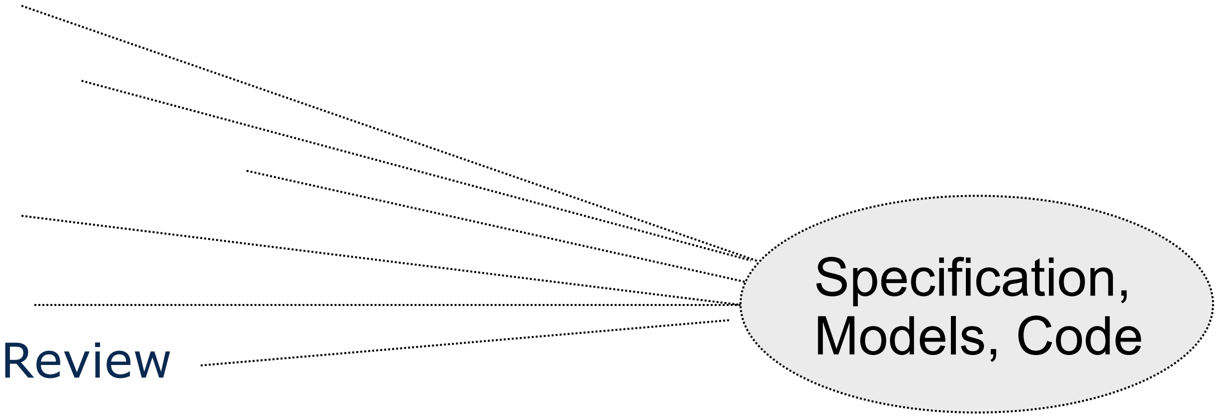
Software-Management
Wie man Projekte macht (SS)

- What should the radio be able to do?
- Why will it better than the competitors? (competitive business edge)
- How can we go the way in a structured way towards the product?
- How can we engineer it?

ST-II

What is Software Engineering?

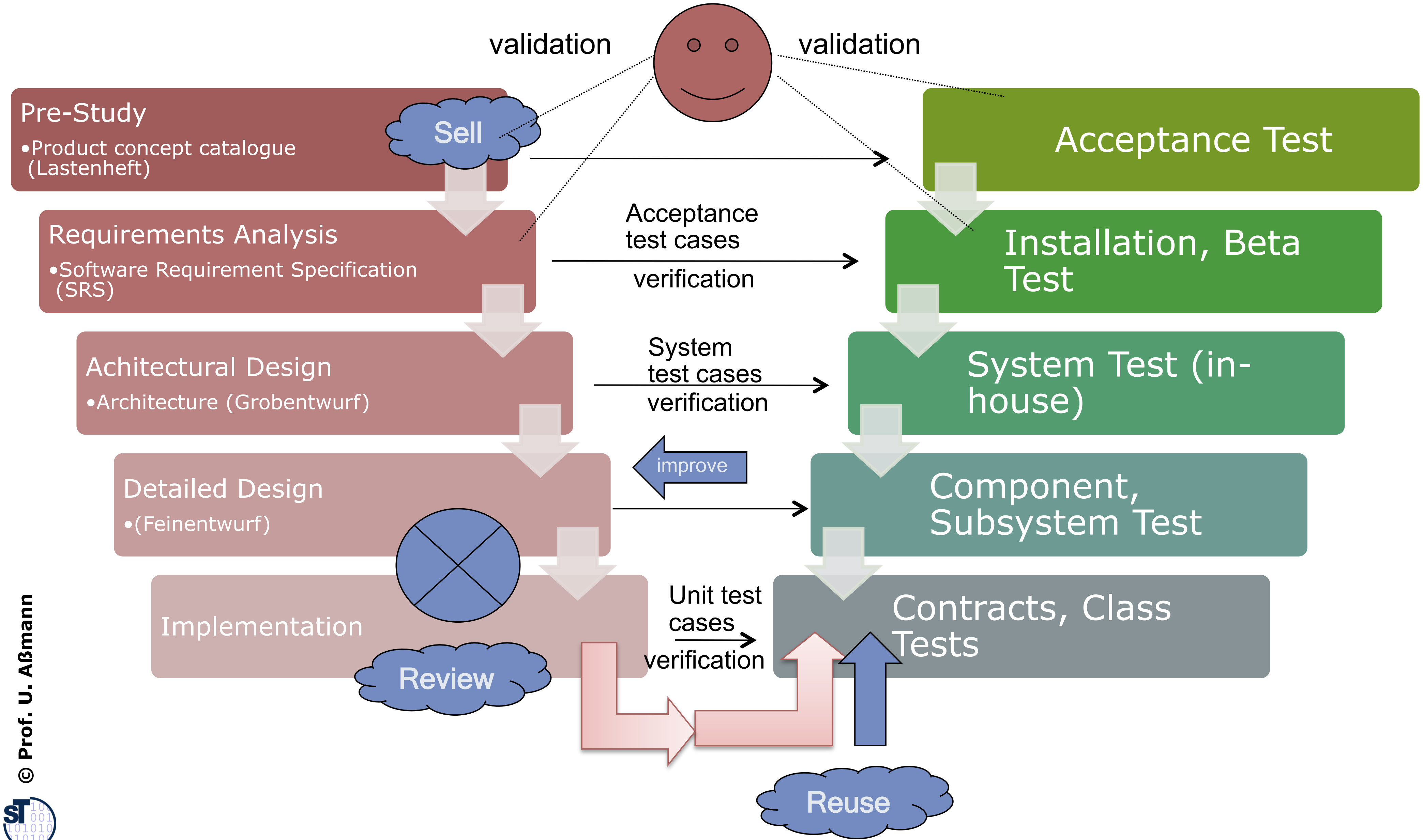
- It teaches the production of software with engineering techniques (the engineer's toolkit)
- Model and Specify
- Analysis and Prediction
- Construction
- Reuse
- Validation, Verification, Review
- Improvement
- Sell



Software engineers model, specify, analyse, predict, build, validate, improve, and sell

Software Engineering in the V-Model

➢ The most simple software development process is the V-model



The (Software) Engineer's Toolkit

Describe a reality (a domain or a system in the world) by a model: ***Descriptive modeling***

- World and problem modeling vs. system modeling

- **Analyze** (measure) a reality by a model
  - Identifying the problem (problem, goal, risk analysis)
  - **Measure** a system (Software metrics)
  - Searching and finding, Controlling
- **Predict** features of a product from the model (form hypotheses, prove)
  - Forming hypotheses about the system

Specify a system: ***Prescriptive modeling***

- Specifying features and requirements of a system

- **Construct** a product (realize, develop, invent, build): apply systematic engineering steps to get a high-quality, evolvable software system
  - **Elaboration** (adding more details to the model to arrive at an implementation)
  - **Refinement** produces a model that *implies* the behavior of the original
    - Refinement renders the model more precise and detailed
  - **Describing** the infinite and the unknown with finite descriptions
  - **Structure** a model (making the model more clear)
    - **Abstraction** (leaving out detail, focusing on the essential)
    - **Architecture** (leaving out application specific details)
    - **Domain Transformation** (changing representation of model)
    - **Refactor** a model: simplify its structure while retaining behavior
  - **Compose** a system from components
  - **Reuse** parts of products
    - Engineer a *product line (product family)*

Model-Driven Software Development in Technical Spaces
How to be productive in software development (WS)

Design Patterns and Frameworks
Architektur objektorientierter Systeme (WS)

Component-Based Software Engineering
Produktlinien mit anderen Komponentenmodellen (SS)

The (Software) Engineer's Toolkit

- **Validate** and **verify** hypotheses on the product
  - Experimentation (empirical software engineering)
  - Consolidate (Checking consistency, integrity, wellformedness, completeness, soundness)
  - Testing
  - Proving (formal software engineering, formal methods)
  - Statistics (not covered here)
  - Review the process of construction (Process review)
  - Review the product (Product review)
- **Improve** the product
  - Reverse engineer towards better structure
  - Restructure
  - Optimize with regard to a value model
- **Sell** the product(s)
  - The software engineer solves problems to earn money for his company and himself
  - How to come to products?
  - How to talk to customers?
  - How to see the problem of the customer?
  - How to reach a market with a product?
  - How to found a startup?
  - Often, engineers are good technicians, but fail to sell the products

Requirements Engineering und Testen (Dr. Demuth)
Wie man Qualität für Software erzielt (WS)

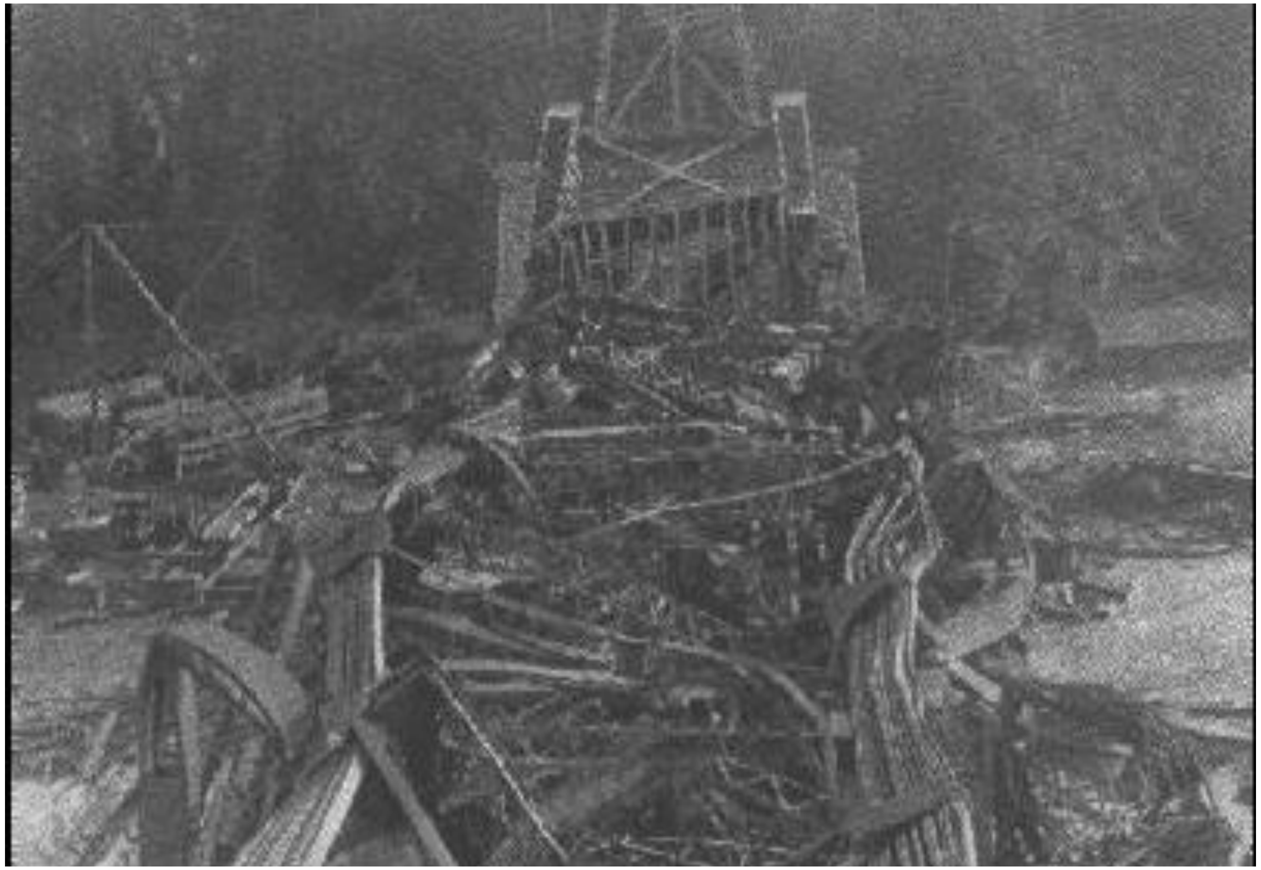
Future-Proof Software Systems (Dr. Furrer)
Evolvable architectures (WS)

Software as a Business
(WS)
How to develop a business model and a startup

The Responsibility of the Engineer

https://en.wikipedia.org/wiki/Quebec_Bridge

http://web.archive.org/web/20110714140148/
http://www.mysteriesofcanada.com/Quebec/
quebec_bridge_collapse.htm



https://en.wikipedia.org/wiki/File:Quebec_Bridge_Collapse_of_1907.jpg

The Engineer‘s Ring and Oath (US, Canada)

- https://en.wikipedia.org/wiki/Engineer%27s_Ring
- https://en.wikipedia.org/wiki/Iron_Ring
- http://www.ironring.ca/background.php

I am an Engineer.
In my profession I take deep pride. To it I owe solemn obligations.

Since the Stone Age, Human Progress has been spurred by the Engineering Genius. Engineers have made usable Nature's vast resources of Materials and Energy for Humanity's Benefit. Engineers have vitalized and turned to practical use the Principles of Science and the Means of Technology. Were it not for this heritage of accumulated experiences, my efforts would be feeble.

As an engineer, I, (full name), pledge to practice Integrity and Fair Dealing, Tolerance, and Respect, and to uphold devotion to the standards and dignity of my profession, conscious always that my skill carries with it the obligation to serve humanity by making best use of the Earth's precious wealth.

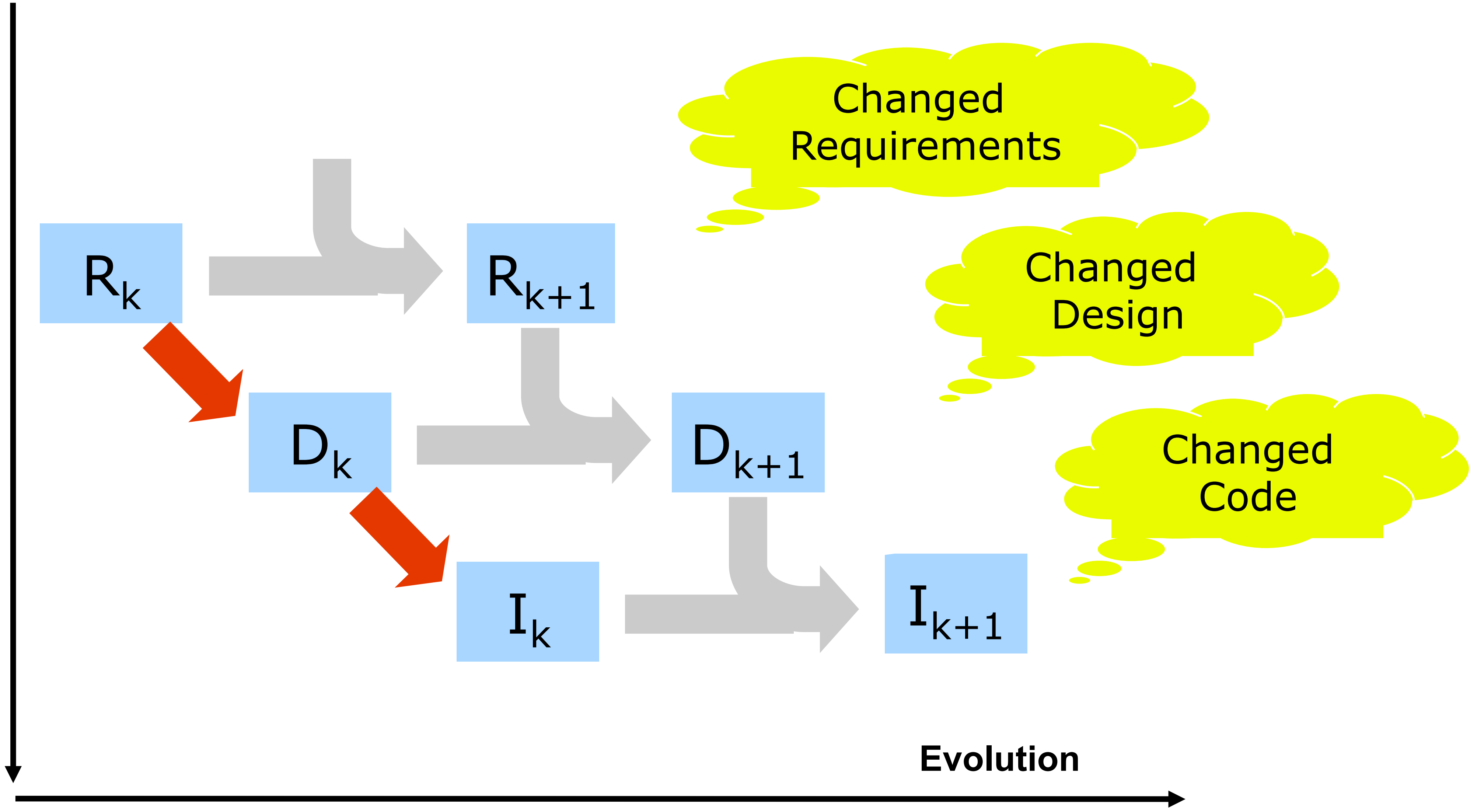
As an engineer, I shall participate in none but honest enterprises. When needed, my skill and knowledge shall be given without reservation for the public good. In the performance of duty, and in fidelity to my profession, I shall give the utmost.

Forward Engineering, Backward Engineering, Improvement, Round-Trip Engineering

2.1. SCENARIOS OF SOFTWARE ENGINEERING

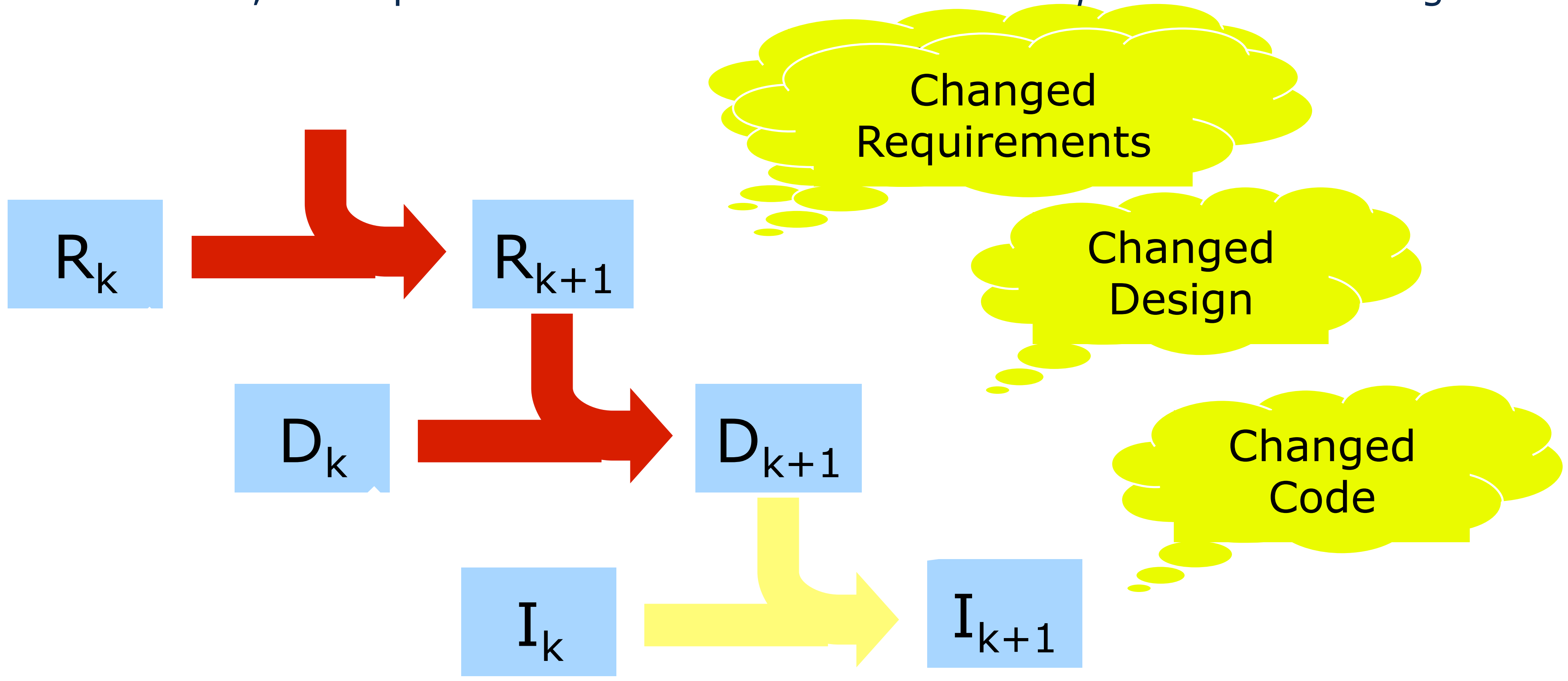
Forward Engineering and Evolution

- With CASE or **MDSD** tools, implementations can be generated from implementation models (**model-driven software development)**



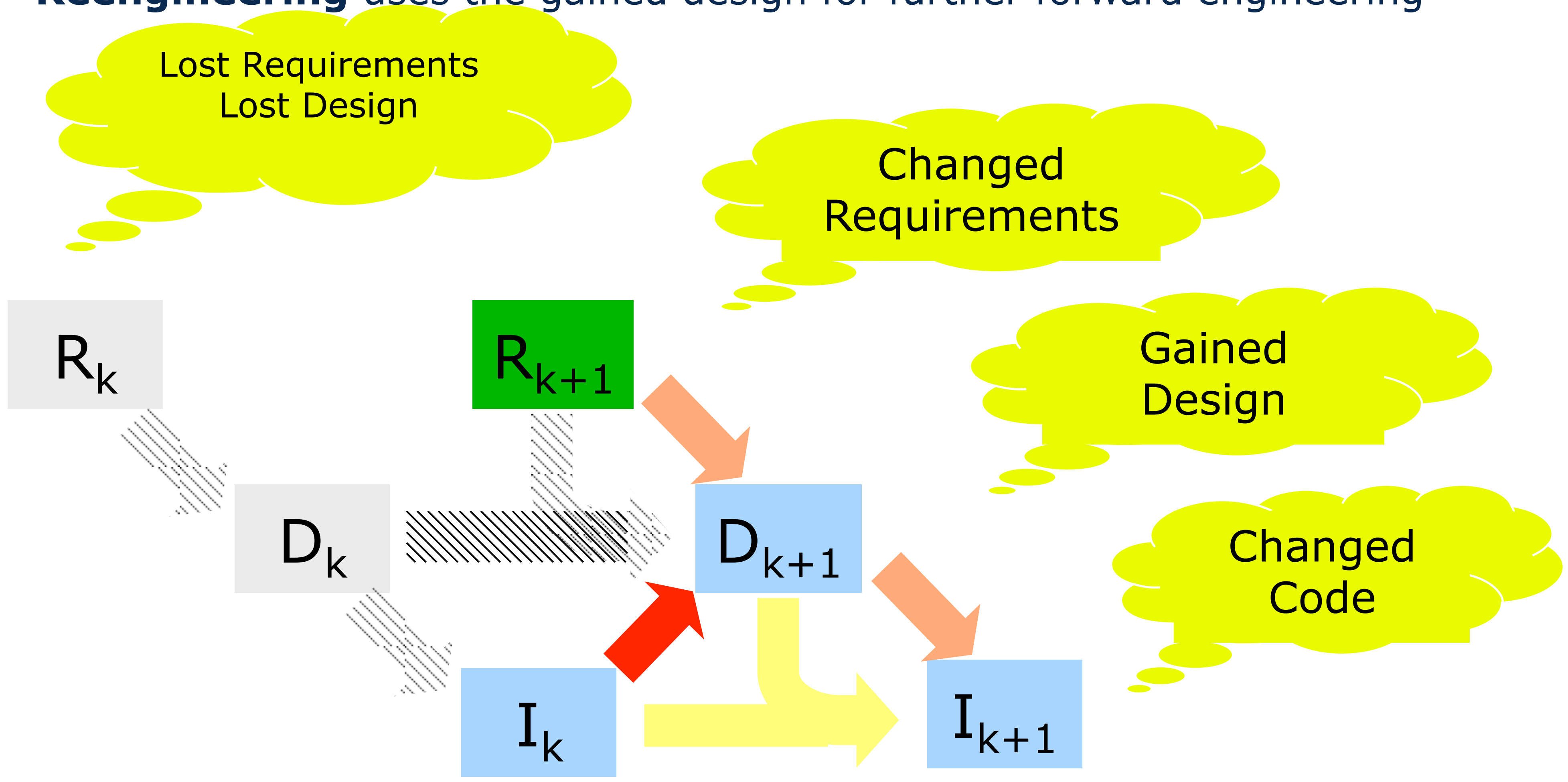
Software Evolution

- Changed requirements require unforeseen refactoring and extensions
- Software must be structured flexibly so that it can be evolved
- Sometimes, more product variants are created and a *product line* emerges



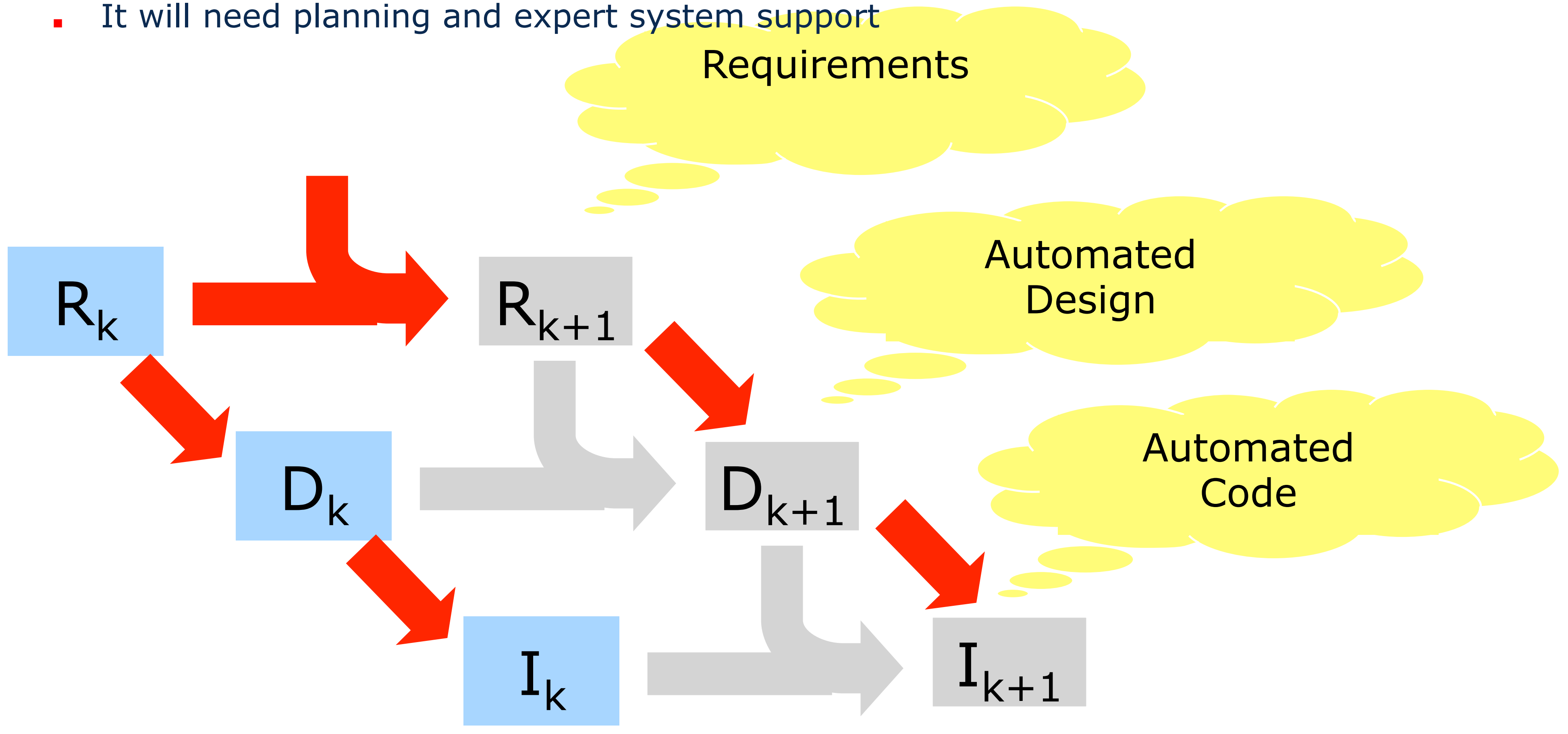
Software Reengineering

- **Reverse Engineering** attempts to recover design from code
- **Reengineering** uses the gained design for further forward engineering



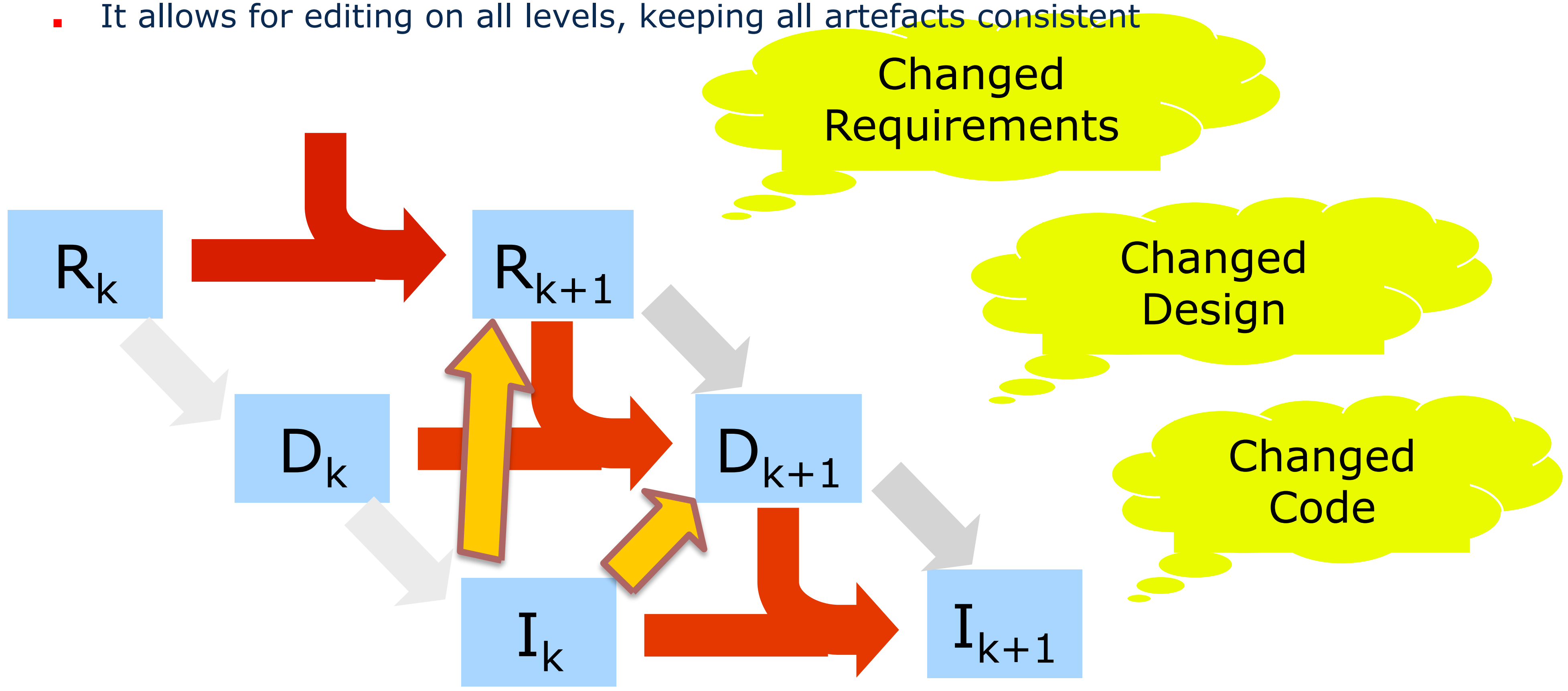
The Dream: Automated Programming

- **Automated programming (generative programming)** generates code from requirements automatically.
  - It will need planning and expert system support



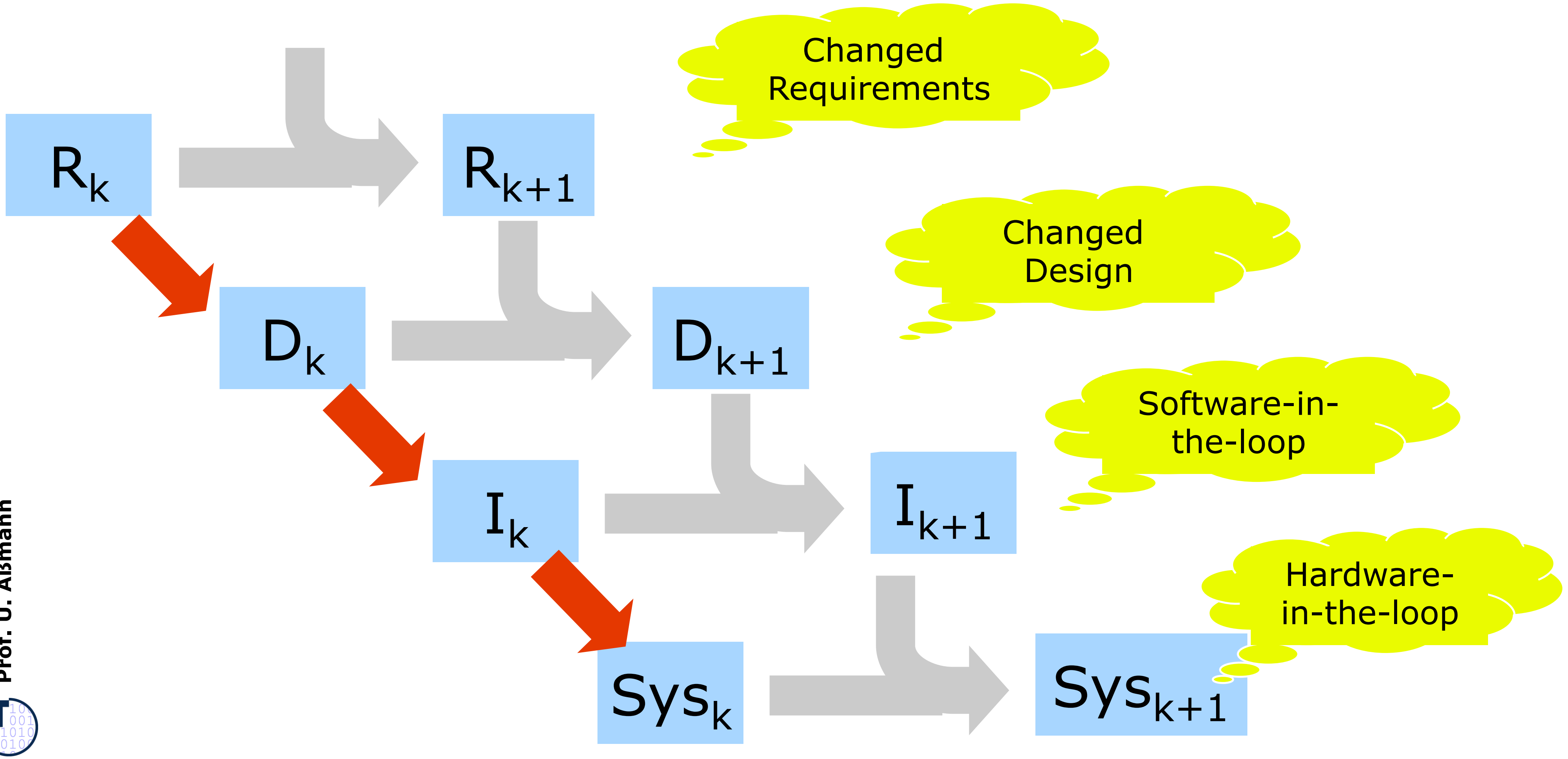
Round-Trip Engineering (Forward and Backward)

- **Round-trip engineering** combines forward and reverse engineering
  - It allows for editing on all levels, keeping all artefacts consistent



Virtual Prototyping

- For safety-critical embedded hw/sw systems, virtual prototyping is used
- Software-in-the-loop (SIL): virtual prototype with simulator of hardware

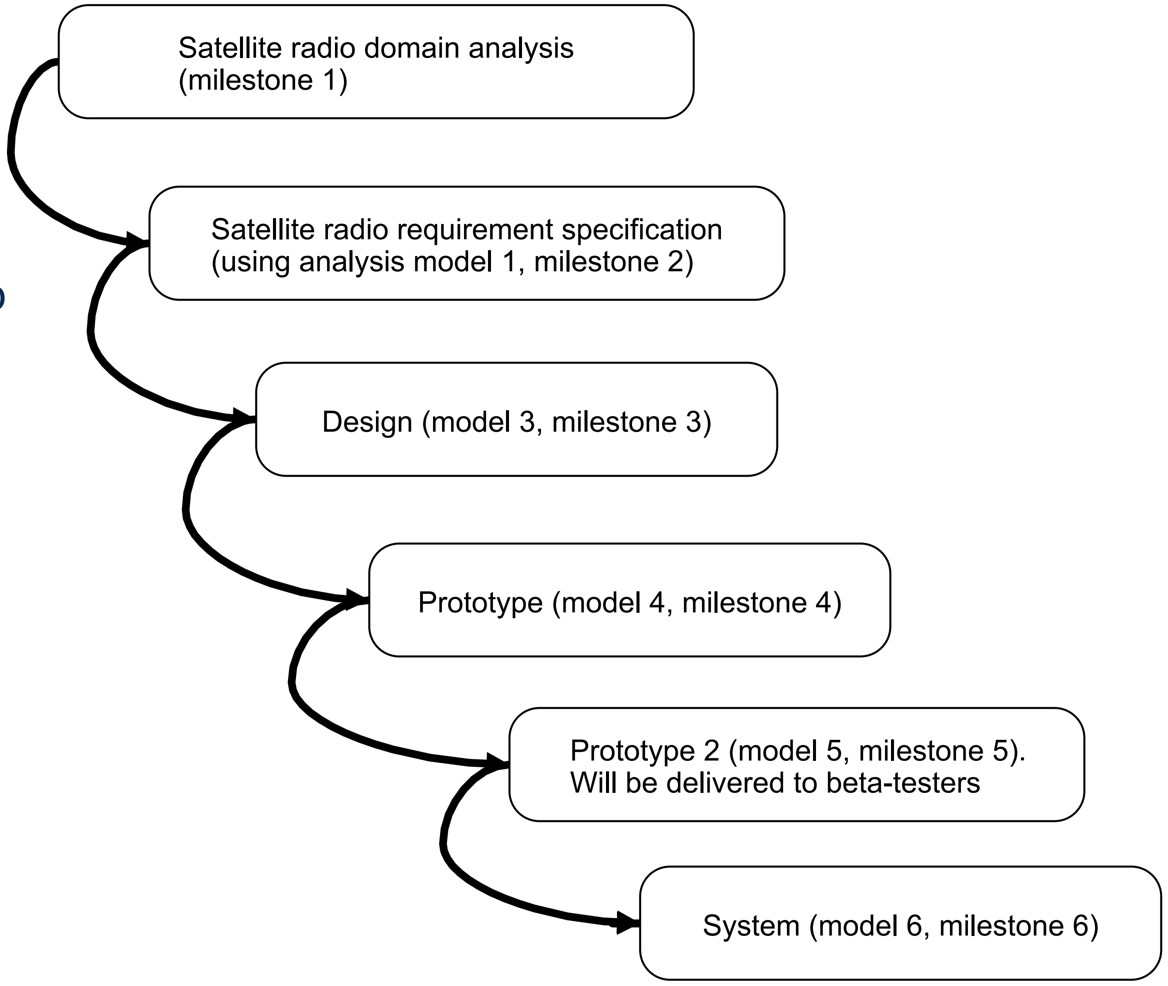


2.2 A RUN THROUGH A FORWARD ENGINEERING CYCLE

2.2.1 FIRST STEP: ANALYSIS

2.2.1 First Step: Analysis

- How do we arrive from the requirements at the product? Let's take an engineer's approach (Analysis steps):
  - Engineers analyze problems to understand what to do
  - Engineers specify a solution and realize (construct) it
  - For both activities, engineers model the world to master it
- Steps
  - We fix the requirements in a requirement specification (requirements models)
  - We go step by step through different design models
  - … until we arrive at the implementation model (which is the system)


Satellite radio domain analysis (milestone 1)
Satellite radio requirement specification (using analysis model 1, milestone 2)
Design (model 3, milestone 3)
Prototype (model 4, milestone 4)
Prototype 2 (model 5, milestone 5). Will be delivered to beta-testers
System (model 6, milestone 6)

But... What Is A Model?

- Pidd suggests a hierarchy of definitions:
  - *A model is a representation of reality*
  - A model is a representation of reality *intended for some definite purpose*
  - A model is a representation of reality intended *to be of use to someone charged with understanding, changing, managing, and controlling that reality*
  - A model is a representation of a part of reality *as seen by the people who wish to use it*
    - *To **understand** that reality **(descriptive model, map)***
    - *To **change, manage, and control** that reality **(prescriptive model, blueprint**)*
- More simply:
  - A model is a representation of a part of a domain, or of a function of a system, its structure, or behavior
  - A model is an abstraction of a system
- A model is *partial*, i.e., *abstract*, and neglects some parts of the reality
- Question: what does this mean for the Satellite radio?

- (J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1991) cited from Dr. Jochen Küster | MDSE 2011

"A model is an abstraction of something for the purpose of understanding before building it"

To Produce Software, We Model

- Software construction uses two kinds of models

The World

Problem Domain

Problem Analysis

What is the problem?

Problem model

(Analysis model)

Models the *problem reality*

Understand a problem

Descriptive (analytic) models

Software Systems

System Domain

System Design

What is the solution?

System model

(Design model)

Models the *system reality*

Manage that reality

Prescriptive models (specifications)

The Satellite Radio as Example

The World

Problem Domain

Problem Analysis

No FM in USA

Digital radio quality required everywhere

Software Systems

System Domain

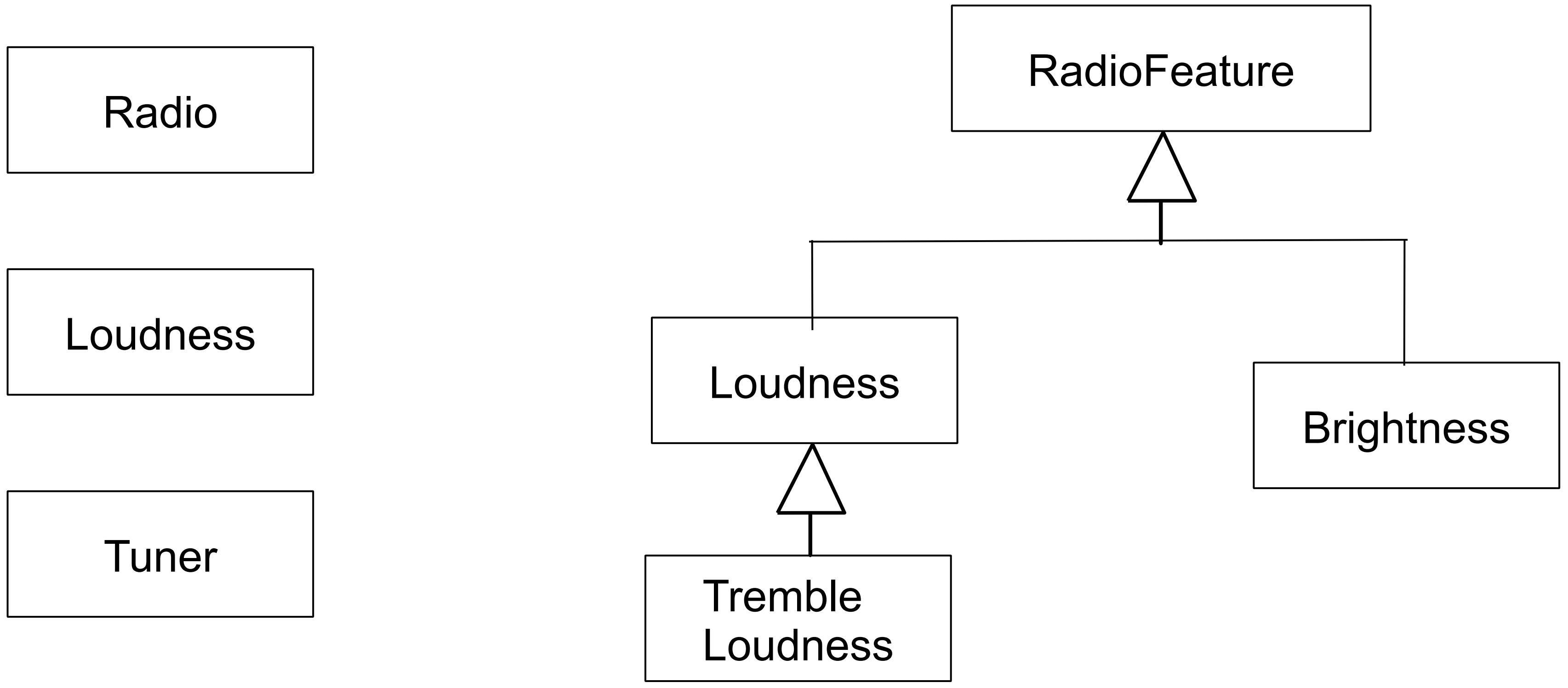
System Design

Satellite Radio

Software-controlled embedded system

Descriptive Models: Glossaries, Classifications, Taxonomies, and Ontologies

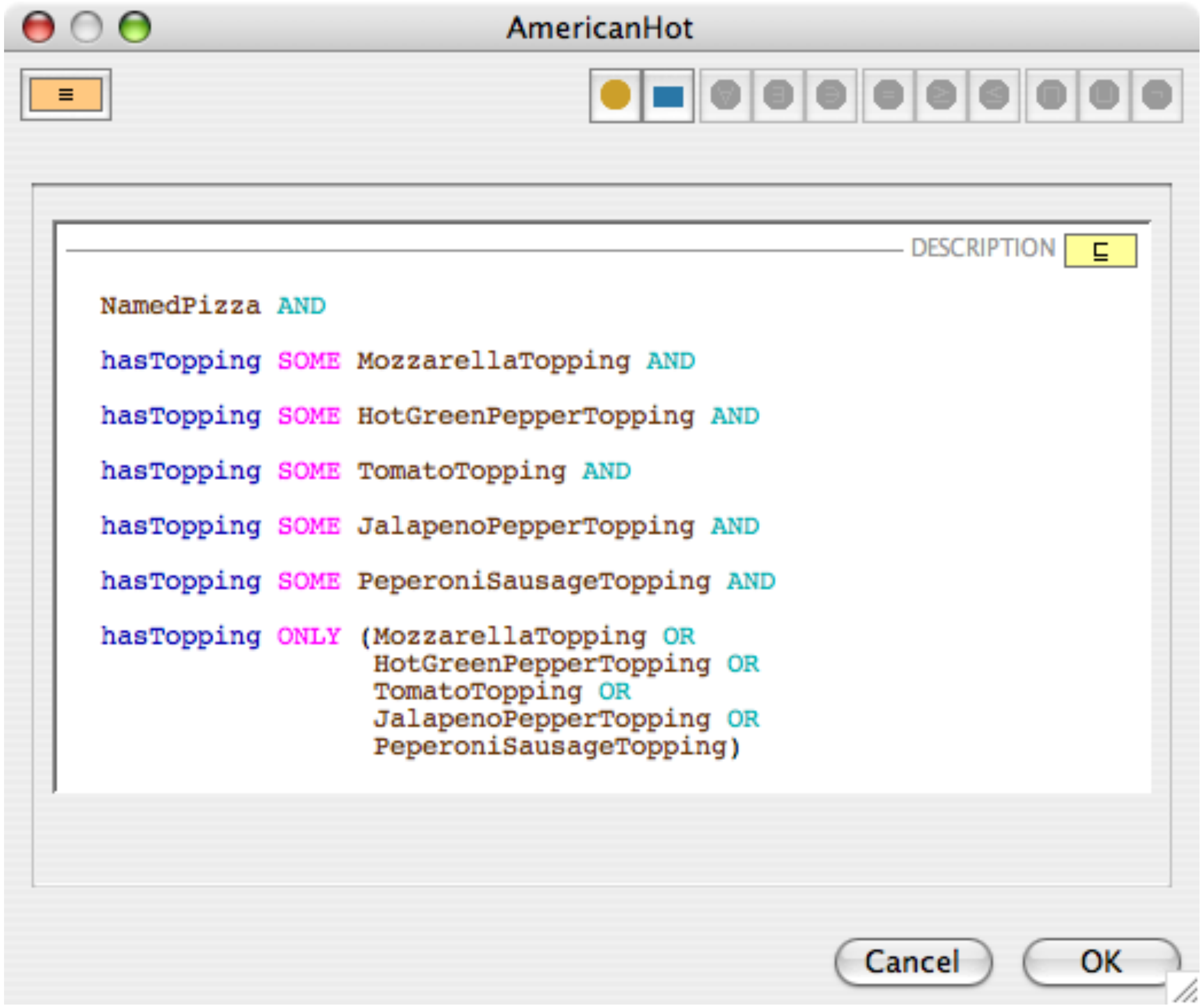
- A **glossary** is a set of explained terms
- A **classification** is a grouping of the concepts of a domain into classes
- A **taxonomy** (Begriffshierarchie) superimposes a hierarchical or acyclic is-a relationship
  - Analyse similarity (commonality-variability analysis)
- A **ontology** adds associations, class and relation expressions, and well-formedness constraints



Ontologies as Standardized Domain Models

- A **(domain) ontology** is a shared, standardized model for a domain, consisting of a taxonomy and integrity constraints (consistency constraints) constraining the hierarchy
  - Rules to produce *derived parts* of the hierarchy. The derived parts are *intentionally* specified
- Ontologies are standardized domain models and play an important role in domain analysis
  - In general, a domain model need not necessarily be standardized
  - For many domains, domain modeling will start from these ontologies
  - *Domain engineers* produce domain ontologies
- Example:
  - Dublin Core ontology with concepts such as Date, Author, Comment
  - Medical ontologies, such as gopubmed.org
  - Upper ontologies (conceptual ontologies), such as SUO suo.ieee.org
  - Biochemical ontologies (Gene ontology www.geneontology.org)
- Ontologies in the Semantic Web
  - In 2003, the W3C has standardized the first ontology language for the web: OWL (web ontology language)

Ontology in OWL „Manchester Syntax“



```
NamedPizza AND
hasTopping SOME MozzarellaTopping AND
hasTopping SOME HotGreenPepperTopping AND
hasTopping SOME TomatoTopping AND
hasTopping SOME JalapenoPepperTopping AND
hasTopping SOME PeperoniSausageTopping AND
hasTopping ONLY (MozzarellaTopping OR
                 HotGreenPepperTopping OR
                 TomatoTopping OR
                 JalapenoPepperTopping OR
                 PeperoniSausageTopping)
```

What is a Specification (Prescriptive Model)?

- A ***specification*** is a prescriptive model (blue print) of the system, i.e., a precise description what a system
  - should deliver (service, delivery, postconditions, guarantees)
  - requires for the delivery (requirements, preconditions, assumptions)
  - “the truth lies in the model” (J.M. Favre)
- A specification must be *realized (implemented)*. An implementation can be *verified* with regard to a specification
  - showing that the implementation derives the delivery from the requirements
- A specification may contain one or several *prescriptive models* of the system
  - Because models are abstract, they are partial representations of partial knowledge
  - Several dimensions of abstractions can be used, so many specifications for a system can be constructed
- However, often, the word specification and model are used interchangeably (which is not precise)

Summary: Different Kinds of Specifications and Models

- **Descriptive (Analysis) models**
- Domain model:
  - Domain analysis is the process of identifying and organizing knowledge about the application domain
- "Real"-Problem model:
  - Usually, the requirement specification includes a problem model – to support description and solution of these problems
- Goal models
  - What do we want to achieve with the system?

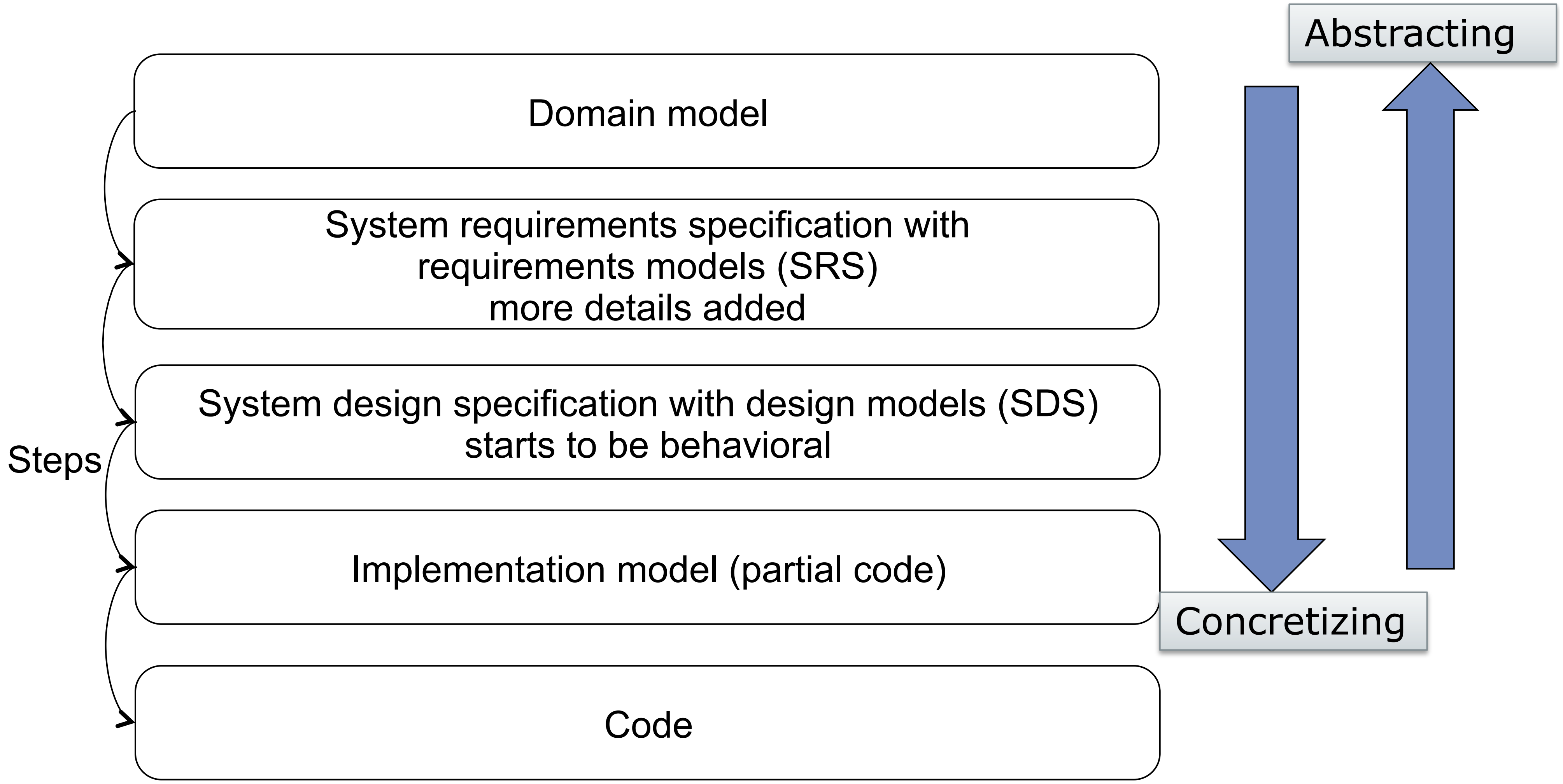
- **Prescriptive models (system models, specifications)**
  - From the analysis models, we derive the system models.
- Requirements specification (SRS):
  - the specification what the system should deliver.
  - Functional requirement model: system functions
  - Non-functional requirement model: system qualities
- Design models:
  - abstract representation of a system on the level of a design language
- Architecture models
  - Describing the software architecture
- Implementation models:
  - partial representation of the system on the level of an implementation language

Structural vs. Behavioral Models

- A **structural model** captures the structure of a reality
  - Integrity constraints for well-formedness
- A **behavioral model** captures its behavior
  - A behavioral model uses a structural model and adds a model how a reality reacts
    - operations (functions, procedures, methods, ...)
    - event-condition-action rules,
    - a state space
- Objects have a state space, often represented by
  - Petri-nets (see later) and their specializations:
    - a finite state machine
    - a hierarchical state machine (state chart)
    - data-flow diagrams
  - Process algebra

Specifications and Models in Software Engineering

- Developing from declarative to behavioral models
- Earlier models should be *abstractions* of later ones, later models should be *concretizations* or *refinements* of earlier ones


Abstracting
Domain model
System requirements specification with requirements models (SRS) more details added
System design specification with design models (SDS) starts to be behavioral
Steps
Implementation model (partial code)
Concretizing
Code

2.2.2 SECOND STEP: PREDICTION

Second Step: Prediction

- Prediction is important for **critical software:**
  - Real-time software in embedded systems
  - Safety
  - Security and privacy
  - Energy efficiency
- Behavioral models allow for **prediction**
  - Graph-based models can be consistency-checked with logic reasoners
    - Integrity constraints constrain the object sets (object extents) of the classes
    - Structural constraints (reducibility, layering)
  - Petri nets can be verified with matrix theory
    - Resource consumption (memory consumption)
    - Liveness, Fairness, Deadlocking of the processes
  - Statecharts can be checked with model checkers
- **Virtual prototyping** is used to predict the behavior of a system:
  - Simulation is also called **model-in-the-loop (MIL)**
  - Simulation of system behavior with **simulation languages** (see www.openmodelica.org)
- This area is called *formal methods* of software engineering

How to come to the next model?

2.2.3 THIRD STEP: CONSTRUCTION

Construction with Refinement-Based Development

- The construction of systems starts off from Domain Model over Requirement Specification and Design Specification to Implementation Model to Code:
  - Develop the next specification, starting from the previous ones

- **Construction steps:** For every model, start with some simple form. Then, apply construction steps:
  - **Elaboration:** Elaborate more details – enrich the model with more semantics
    - **Concretization**: add concrete details
    - **Refinement**: Refine an existing specification/model, by detailing an abstract concept
  - **Check:** Check consistency of models
  - **Measure** quality and quantity of models
  - **Compose** from components
- We can distinguish several methods of constructive development

Questions for the Methods of Development

- **Elaboration (concretizations)**: Elaborate more details
  - Which Elaboration steps exist?
  - How do I know in which direction to elaborate?
- **Refinements:** With and without correctness proofs that the semantics of the abstract concept is provided by the refinement
    - **Syntactic refinement:** replace a part of the model by something more fine-grained
    - **Semantic (behavioral) refinement:** prove for a syntactic refinement that it is correct, i.e., implies semantics of the original (either preserves semantics, or enriches)
  - Pointwise Refinements: detailing an abstract concept by a net of more concrete ones
  - Regional Refinements: detailing a region of the model by a net
  - Crosscutting Refinements: detail a slice of the model
- **Rotations (Symmetry operations)**: Apply a semantics-preserving change
  - **Rotate:** Symmetry operations (semantics-preserving operations)
  - **Restructure** (**refactor):** rearrange for more structure, but keep requirements and delivery, i.e., semantics
    - Which restructuring? (when is a specification too complex?)
  - **Semantic refinement:** prove for a syntactic refinement that it is correct, i.e., either preserves semantics, or enriches semantics
  - **Transform Domains** (change representation, but keep semantics)
    - Which representation change? (which representations are appropriate for which purpose?)

Reuse of Models and Code in Construction

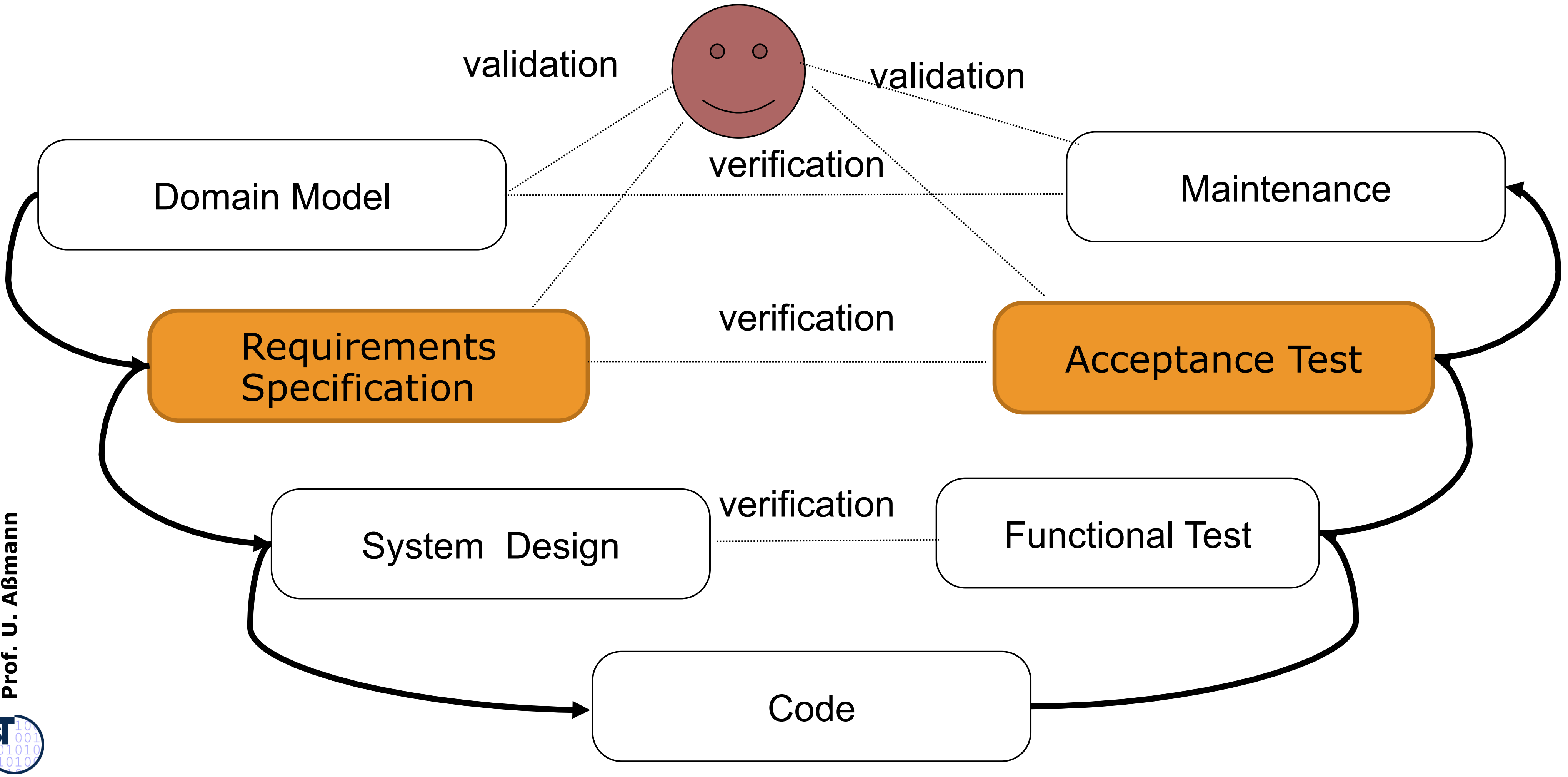
- **Reuse by composition:** Engineers try to reuse well-established solutions
  - Components (CBSE)
  - Design patterns (-> DPF)
  - Models (model-driven architecture) (-> MOST)
  - Best practices
- Reuse: to simplify system construction
  - To save costs
  - To reduce testing effort

Software without test architecture is called a *program.*

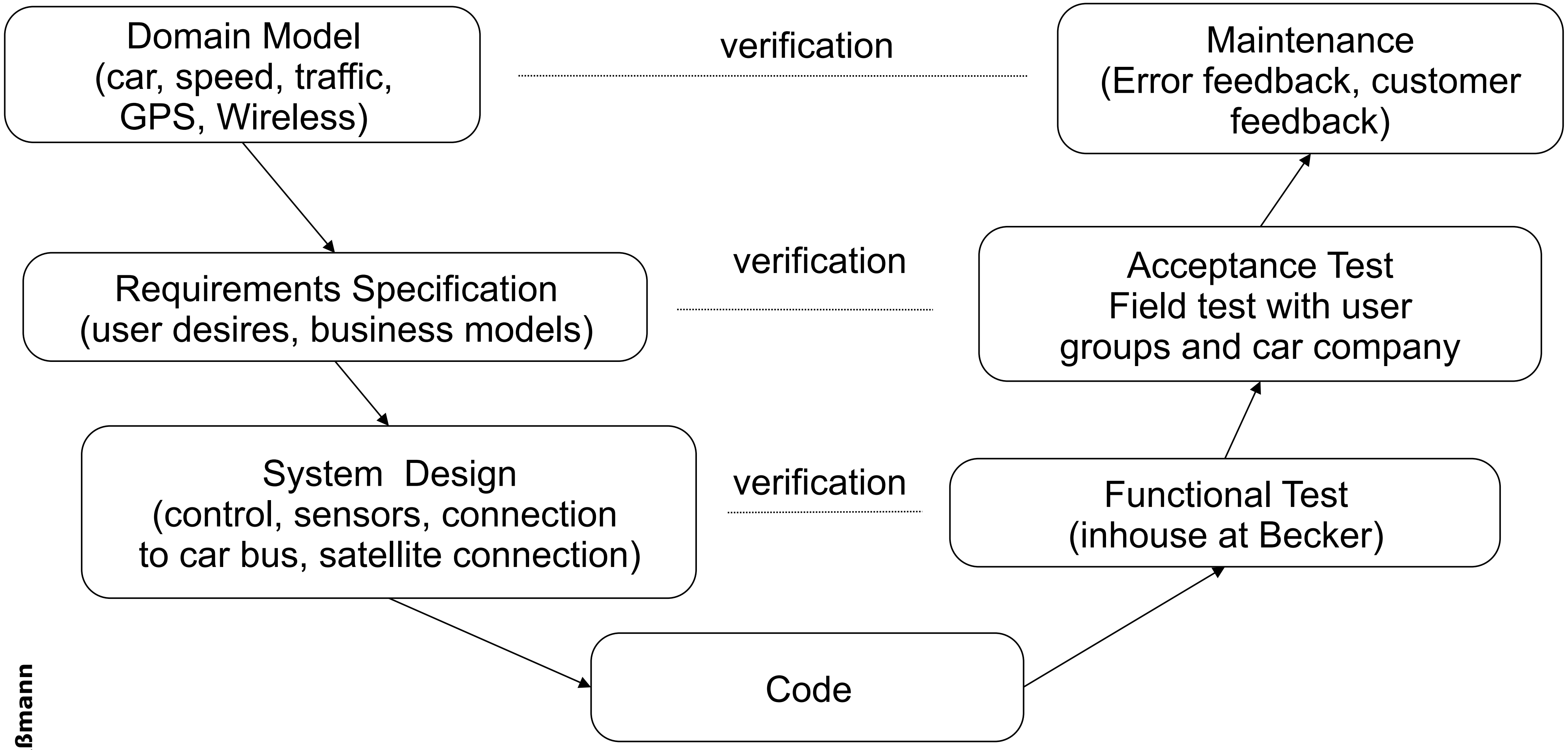
2.2.4. 4TH STEP: VALIDATION

Validation in a Software Development Process (V-Process Model)

- All specifications and models have to be validated (demonstrated to the customer), verified, or event formally verified.
  - Implementations against specifications
- Result: A V-like software development process



Validation of the Satellite Radio in the V-Model



Domain Model (car, speed, traffic, GPS, Wireless)
verification
Maintenance (Error feedback, customer feedback)
Requirements Specification (user desires, business models)
verification
Acceptance Test Field test with user groups and car company
System Design (control, sensors, connection to car bus, satellite connection)
verification
Functional Test (inhouse at Becker)
Code

- [JezequelMeyer] Jean-Marc Jézéquel, Bertrand Meyer. Put it in the contract: The lessons of Ariane.
  - https://www.irisa.fr/pampa/EPEE/Ariane5.html
  - This article appeared in a slightly different form in IEEE Computer, as part of the Object-Oriented department, in January of 1997 (vol. 30, no. 2, pages 129-130).
- The reaction of Ladkin on [JezequelMeyer]
  - http://www.rvs.uni-bielefeld.de/publications/Reports/ariane.html
- https://fr.wikipedia.org/wiki/Ariane_5
- http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html

2.2.4.1 THE ARIANE 5 LAUNCH FAILURE

The Ariane 5 Launch Failure

June 4th 1996

Total failure of the Ariane 5 launcher on its maiden flight

The following slides are used by permission from

[Ian Summerville, Software Engineering]

https://upload.wikimedia.org/wikipedia/commons/0/05/Ariane_5_Le_Bourget_FRA_001.jpg

Ariane 5 Launch Failure

June 4, 1996 maiden flight

- Ariane 5 was designed to launch commercial payloads (e.g.communications satellites, etc.) into orbit
  - Heavier payload than Ariane 4
  - More thrust (Schub), launches steeper
- 37 seconds after a lift-off, the Ariane 5 launcher lost control
  - Incorrect control signals were sent to the engines
  - These swivelled so that unsustainable stresses were imposed on the rocket
  - Rocket started to break up and self-destructed
- → $0.5 bio. Loss
- → software failure

Ian Summerville, Software Engineering

The Problem

- The attitude and trajectory of the rocket are measured by a inertial reference system
  - This transmits commands to the engines to maintain attitude and direction
  - The software of this system failed: Diagnostic commands were transmitted to the engines
    - ..which interpreted them as real data and which swivelled to an extreme position
  - The backup system shut down the rocket
- **Integer overflow failure** occurred during converting a 64-bit floating point number to a signed 16-bit integer
  - There was no exception handler
  - So the system exception management facilities shut down the software
- The backup software was a copy and behaved in exactly the same way.

Software Reuse Error: No Contract Checking

- The software that failed was reused from the Ariane 4 launch vehicle.
- Contract of Ariane-4 modules were not checked, neither statically nor dynamically [JezequelMeyer]
  - The computation that resulted in overflow was not used by Ariane 5
- Decisions were made
  - Not to remove the facility as this could introduce new faults
  - Not to test for overflow exceptions because the processor was heavily loaded.
  - For dependability reasons, it was thought desirable to have some spare processor capacity

A **Contract** is a specification summarizing the behavior of a software module that can be checked statically or dynamically for flawless use of the module

Ian Summerville, Software Engineering

Why not in Ariane 4?

- Ariane 4 has a lower initial acceleration and build up of horizontal velocity than Ariane 5
  - The value of the variable on Ariane 4 could never reach a level that caused overflow during the launch period.
  - That had been proved (for Ariane 4)!
- As the facility that failed was not required for Ariane 5,
  - there was no requirement associated with it.
- As there was no associated requirement,
  - there were no tests of that part of the software and hence no possibility of discovering the problem.
- During system testing, simulators of the inertial reference system computers were used.
  - These did not generate the error as there was no requirement!

Ian Summerville, Software Engineering

What Kind of Engineering Error Was It?

- Requirements Analysis Error
- Reuse Error, Reuse Specification Error: no contract checking
- Review Failure (→ defensive programming)
  - The design and code of all software should be reviewed for problems during the development process
  - Either the inertial reference system software was not reviewed because it had been used in a previous version
    - Or the review failed to expose the problem or that the test coverage would not reveal the problem
    - Or the review failed to appreciate the consequences of system shutdown during a launch

Reuse without a contract is sheer folly. [Jezequel, Meyer]

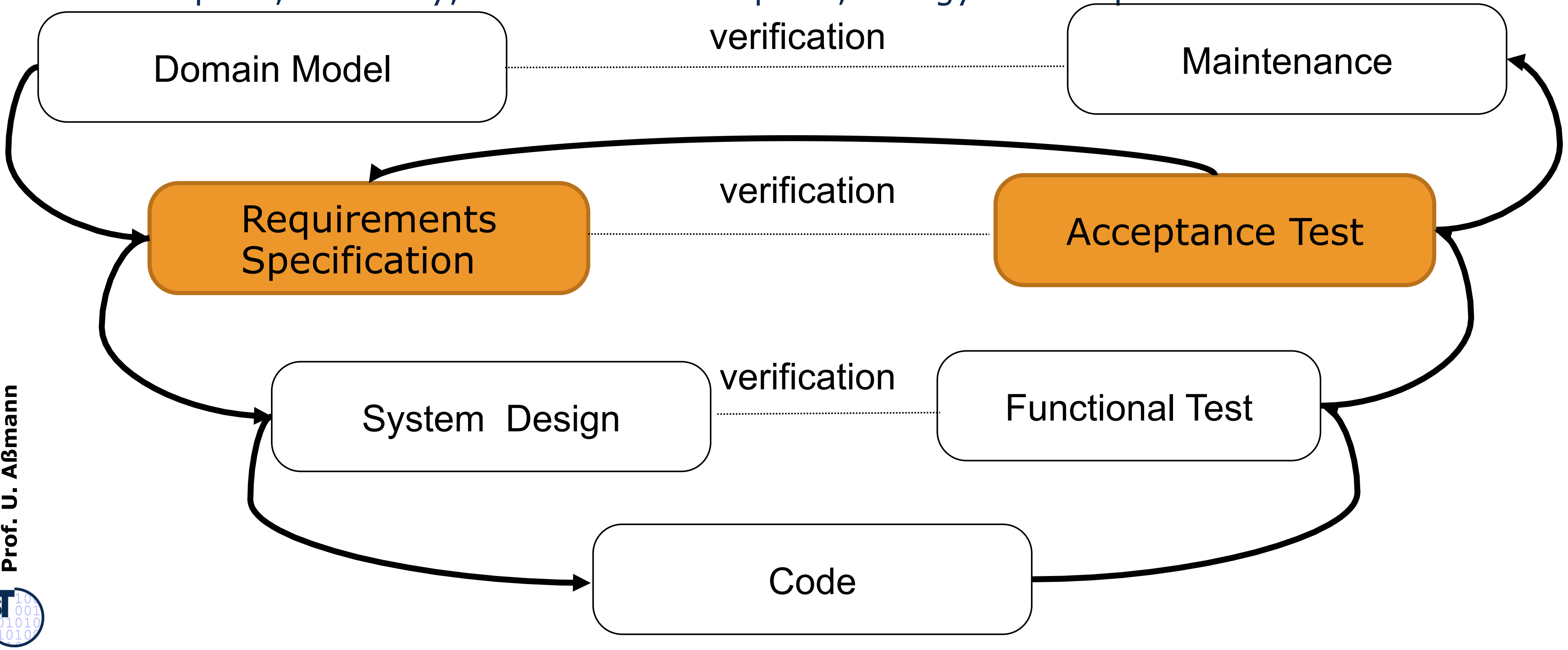
Lessons Learned

- In critical systems **don’t run software unless it is actually needed**
  - Return best effort values if the absolutely correct values cannot be computed
  - Do not have system shut-down as default exception handler in systems that have no fail-safe state
  - Do run-time contract checking
- Test for what the system should do, (good test cases)
  - **and** what the system should **not** do (error test cases)
  - **And** what the system does in case of exceptions (failure test cases)
- Use **Virtual Prototyping** for **prediction**
  - First, use models **(Model-in-the-loop, MIL)**
  - Then, use simulations **(Software-in-the-loop, SIL)**
  - Finally, use real equipment **(Hardware-in-the-loop, HIL)**
- **Improve the review process** to include external participants and review all assumptions made in the code

2.2.5 5TH STEP: IMPROVEMENT

5th Step: Improvement (Optimization)

- Done via iteration, and ad-hoc
  - Not in the focus of the course.
- Section "Product Lines" will treat some aspects of software evolution, namely when new products should be derived from an existing product or product family.
- Optimization means: Improve on the qualities of the system
  - Speed, reliability, resource consumption, energy consumption



Some aspects in section “Earning Money with Software”.

2.2.6 6TH STEP: SELLING SOFTWARE AS REWARD FOR THE SOLUTION OF PROBLEMS

The Best Seller Is...

- .. the company who solves a problem best, i.e., provides the best **value proposition** for the customer
- .. the one who pretends to solve a problem best
- .. the one who solves a problem just good enough
- .. the one who solves a problem reliably

Software can be sold as a reward for solving problems of people

Solving problems is a task for Engineers and Entrepreneurs

Why do we need to care about money?

- Calculating the cost and the price of a product is essential for an engineer
- While usually other people distribute the products on the markets („Vertrieb“), engineers must give a price for a product!

Was sich nicht verkaufen lässt, will ich nicht erfinden.

Thomas Alva Edison http://www.gratis-spruch.de/

It is difficult to Earn Money with Software

- "The winner takes it all"
- OSS is cheap
- Product lines is the only way out
  - They encapsulate enough knowledge of a domain which forms a "sellable core"
  - They help to follow market changes quickly

Entrepreneurship

- The difference of entrepreneurship and capitalism is
  - A capitalist wants to earn money
  - An entrepreneur solves problems
- Central question: Which problems can I solve for other people?
  - Get rid of a negative life: What do people need? Where is their **pain**?
  - Enabler for a positive life: What do people care about? Where is a **value** for the customer?

An entrepreneur solves problems of people.

“Make things that remove people’s **pain**”

Pain removers

An entrepreneur creates a **value** in the life of the customer.

“Make things that people need”

Happiness enablers

Grameen

- „Die Grameen-Bank ermuntert die Kinder ihrer Kreditnehmer auch zum Schulbesuch. .. Derzeit studieren mehr als 50000 Studentinnen und Studenten mithilfe von Ausbildungskrediten der Grameen-Bank...
- Wir ermuntern diese jungen Leute, sich fest vorzunehmen, dass sie sich niemals als Arbeitssuchende auf den Arbeitsmarkt begeben werden. Sie sollen später einmal Arbeitsplätze schaffen, nicht sich um Arbeit bewerben. Wir sagen ihnen: Euren Müttern gehört eine große Bank, die Grameen-Bank. Die hat einen Haufen Geld, mit dem sich jedes Unternehmen eurer Wahl auf den Weg bringen lässt. ***Warum wollt Ihr Zeit mit Arbeitssuche vergeuden, um dann für jemand anderen zu arbeiten? Werdet lieber Arbeitgeber, keine Arbeitnehmer.***
- ***Die Grameen-Bank ermutigt die Menschen von Bangladesh zur unternehmerischen Selbständigkeit und wirtschaftlichen Unabhängigkeit – weg von der Abhängigkeit.***"
- Mohammad Yunus – Social Business. Von der Vision zur Tat. Hanser 2010.

The Entrepreneurial Type

- Hard work: do you want to spent 5 years in business until your company has survived?
- An entrepreneur must long for freedom and independence
  - Uncertainty vs longing for freedom
  - People appear in two classes:
    - **Security type:** tends to avoid risks. Likes to be told what to do
    - **Independence type:** loves freedom, independence.
- Self discipline
- Are your aims realistic?

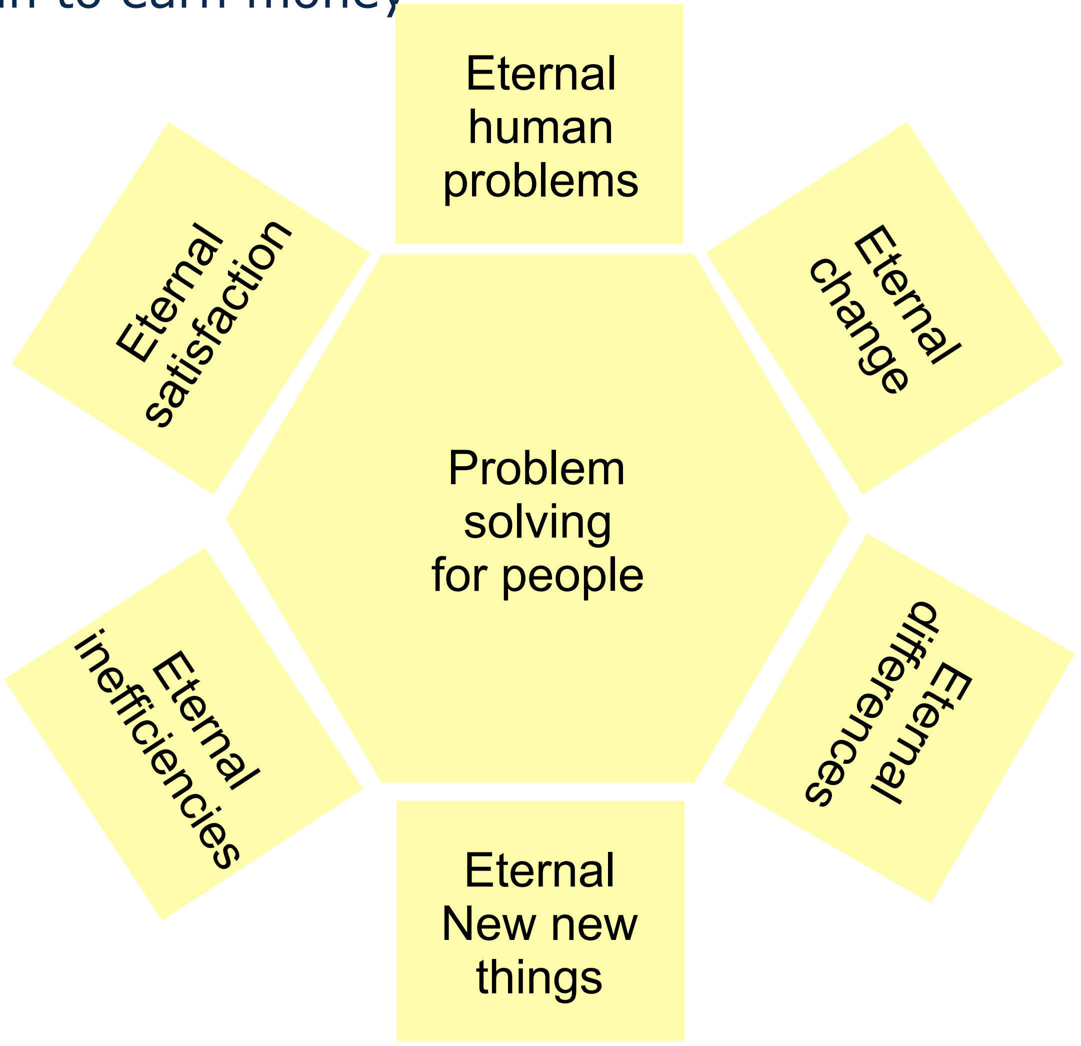
Problem Solving – A Task for the Engineer and also the Salesman

- Successful engineers and salesmen also solve problems for their customers.
  - A successful engineer or salesman can always return to a customer because he has created *satisfaction* in the customer (Kundenzufriedenheit)
- The engineer solves problem with an engineering technology
- The salesman solves problem by mediating the customer’s financial situation and the engineer’s solution
- In small companies, software engineers have to play the role of a salesman, too [Konrad Zuse, Mein Lebenswerk] [Klaus Kemper. Heinz Nixdorf]
- Some of the greatest entrepreneurs of the 20th century have been engineers: Werner von Siemens, Konrad Zuse, Heinz Nixdorf, Robert Bosch

2.2.6.2 STRATEGIES OF SOLVING PROBLEMS AND SELLING

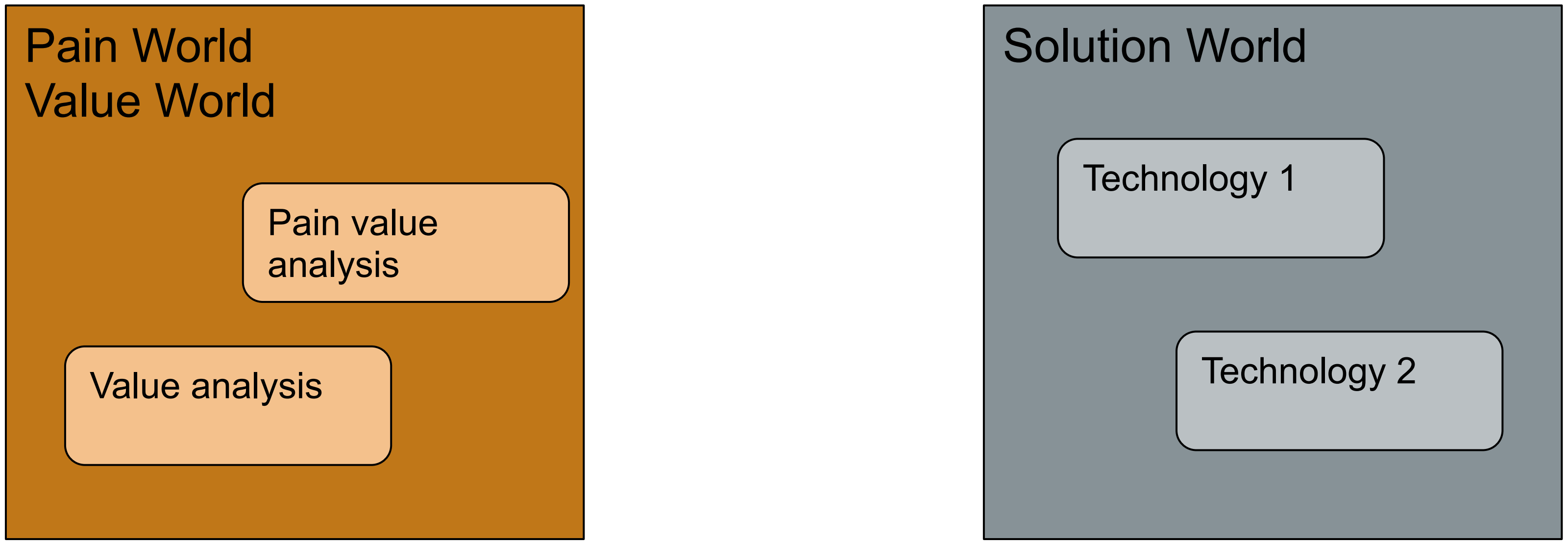
Chances

- „When you find inefficiency, you find opportunity“ [Barrack]
- „Make things people need“
- „Remove pain to earn money“



Provide Problem Solving

- "Knowing a good problem is half the business"
- "Problems are my best friends" (Robert Fritz)
- "Selling drilling machines is not as important as selling holes, but these are completely different businesses" (H. Kagermann, SAP)
- Problem analysis of customers: Find out about problems, and you will earn money
  - → see ZOPP Problem-Objective Analysis
- Try to find **pain problems**, because they create pressure on the customer



Exploit the Eternal Human Problems and Needs

- Hunger, Food, Restaurants, ...
- Love, Relationship
- Hobby
- Beauty
- Exhibiting oneself (Flickr, youtube)
- Housing
- Save money
- Holidays “Ab-in-den-Urlaub.de”
- Overcoming the Space problem: Car, Flights,...
- Simplifying complex things
  - Overcoming bureaucracy
- Communication (Nokia “Connecting people”)
- Being different from others (individualism)
- Lazyness
- Searching knowledge (expert portals)
- Relaxing
  - Tourism, Travel,..
- Events
  - Party, meeting people

Which of these problems is a need, if satisfied, makes the customer happy (happiness problem)?

Which of these problems is a pain problem?

Bedürfnisse nach Lichtenberg

- Wikipeida: (d) Lichtenberg unterscheidet fünf verschiedene Motivationssysteme,[4] die miteinander in Konflikt treten können, auf die er durch Ergebnisse der Säuglingsbeobachtung schließt:

- die biologische Notwendigkeit, physiologische Bedürfnisse zu befriedigen
- ein elementares Bedürfnis nach Bindung, das sich später zu einem Bedürfnis nach Zugehörigkeit erweitert
- das Bedürfnis, Dinge zu erforschen und sich selbst zu behaupten
- das Bedürfnis, auf unangenehme Stimuli aversiv zu reagieren, durch Widerspruch oder Rückzug
- das Bedürfnis nach sinnlichem Vergnügen, Zärtlichkeit und sexueller Erregung

Bedürfnisse nach Maslow

Exploit the Eternal Change

- The markets, the customers, the competitors change.
  - Find out about change, and you will earn money
  - Old players do not recognize change, but often are too immutable
- The stock market principle: “sell when high, buy when low”
  - Investments in a crisis create value
- Embrace change
  - Use it for your purposes, or change will roll you over.
  - Some markets die after some time. Recognize the change, and change your market.
- Which of the expected changes will create pain? (**pain change**)
  - Year 2000 problem was a pain change problem with deadline.
  - Lots of problems had to be solved
- Investigate the future
  - By looking at market change forecasts, e.g., [Canton]
- Look out for **goldrushs:** A **goldrush** is a change with disruptive changes, opening many new changes
  - The German “Energiewende” is a goldrush change with deadline in 2020

Which of these changes is a pain change?

"**Nenne einen Markt, der mit 20 bis 70 Prozent waechst, und es ist fast sicher:**" meint Dr. Konrad Seitz: "**Die Deutschen sind nicht dabei.**" -

Which of these changes is a gold rush change?

Exploit Eternal Differences

- Know-how vs absent know-how
  - Consultance
- Differences in knowledge:
  - Wikonomics: sharing knowledge in a web community
- Cultural differences
- Export from one region; import to the other
  - Asian restaurants, Gyros, Döner
  - Teleconferencing

Eternal Satisfaction
IBM's secret: Customer Satisfaction

- Satisfy your customer (Customer satisfaction)
  - IBM: T. Watson, “THINK”
- Dont' loose a customer. Try to please him so that she returns.
  - It is much more easy to gain somebody who was customer before than getting a new customer
  - Quality and confidence pays off.
- It is 10 times harder to aquire a new customer than to keep an old customer (Stammkundschaft)

The New New Thing

- Innovation creates new new things for which customers may pay higher prices
- "New New Things" are goldrush changes:
- Michael Lewis. The New New Thing. A Silicon Valley Story. Coronet Books. Hodder and Stoughton.
  - Tells the story about Jim Clark, founder of Netscape, how he founds another company, Healtheon, end of the 90s.
  - Fascinating to read about the hypes in the Silicon Valley.

Over time, things become commodity which is given away very cheap or for free

2.2.6.3 What to Sell as a Software Engineer

Different Types of Things to Sell

What you might sell (types of products):

- Consultancy: sell your know-how
  - Analysis studies on a market, trend or strategy
- Service (Requ.analysis, testing, maintenance, modernization, reengineering)
  - Many big companies have their focus there: IBM
- Individual projects for "individual software"
  - SD&M, Accenture, Saxonia systems, …
- Software Product (Licence)
- Product line (product family)
  - Horizontal product line: one product idea in several markets
  - Vertical product line: several products in one market
- Software platform for software ecosystem
- Enterprise landscapes (Anwendungslandschaft) with integration of many tools

Guidelines

- "Go directly to the product" (Prof. Hufenbach)
  - Always consider: which unit of my work will others want to sell?
  - What can be made to a product?
  - For products, licences can be sold
- However, it is difficult to get a software product
  - Software is often considered as a commodity, for which people do not want to pay
  - If a software technology (tool, framework, etc.) is not used, it does not immediately create pain in the customer
- Software is "Soft":
  - Does not have a production cost
  - Others may be able to easily rebuild it
- How can we nevertheless have "software products"?

What to Sell: How SAP Earns Money

- Figures of 2005 in Mrd. Euro [IX Magazine, 3/2006]
- Products

Software licences	2.7 (18% growth)
Products incl. maintenance	5.9 (ERP 1.2, CRM 0.6, SRM 0.12)

- Service

Consultancy	2.1
Training	0.3

Turnaround (Umsatz)	8.5
Win (Gewinn vor Steuern)	2.3
Win net (Gewinn nach Steuern)	1.5

- Market size:
  - Currently targeted: 40 Mrd Euro
  - In 2010, with an extended product portfolio: 70 Mrd Euro

Maturity Levels of Companies

- Class 1 (work hour business): Consultancy and service and individual projects have no income out of licenses, and do not generate a dependency on vendor (vendor lock-in).
  - It is easy to switch them
  - They earn money by selling work hours
- Class 2 (licensing business): Products, product lines, software platforms, and enterprise landscapes generate license incomes
  - Ex. Kontron (embedded systems vendor) is a product and product line company, without vendor lock-in.
- Class 3 (vendor lock-in): Product lines, software platforms, and enterprise landscapes generate dependencies on the vendor.
  - Vendors are hard to switch
  - Ex. SAP is class 3

Software companies are called **mature,** if they generate license fees or maintain a vendor lock-in

Software Engineers: What They May Sell (Types of Products)

Chapter testing, requirements analysis, modeling, model structuring

What you might sell:

- Consultancy: sell your know-how
  - Analysis studies on a market, trend or strategy
- Service (Requ.analysis, testing, maintenance, modernization, reengineering)
  - Many big companies have their focus there: IBM

Chapter "design methods"

- Individual projects for "individual software"
  - SD&M, Accenture, Saxonia systems, ...
- Product
- Product line (product family)
  - Horizontal product line: one product idea in several markets
  - Vertical product line: several products in one market
- Software platform for software ecosystem
- Enterprise landscapes (Anwendungslands tools

Chapter "Product lines"

Chapter "Earning money with software"

What Have We Learned?

- Specifications (complete representations of what the problem is or the system should do) consist of models (abstract representations of worlds)
  - Analysis models in the problem domain
  - System models in the system domain
- Engineers analyze, form hypotheses, construct, validate, improve, sell
  - Detailed models are validated against their more abstract ancestors
  - Implementations are validated against specifications
- Software companies earn money with different forms of activities.
  - Mature companies have revenues based on licensing and vendor-lock-in.
  - Product lines are important for selling
- The course is structured along these activities

Remark: Software and Systems Engineering

- Software Engineering is closely related to a twin, the Systems Engineering
  - Building software into a system (embedded system)
  - Many concepts can be used in both areas.
    - See study line “Distributed Systems Engineering (DSE)”.

The End

- Why is virtual prototyping important for safety-critical systems? Explain its steps.
- Explain the aspects of the ethics of the software engineer.
- Why is the profession of software engineering often so misunderstood by society?

Was ist ein Profi?

- http://www.dwds.de/pages/pages_textba/selbst_out/Selbsteinwechslung.html

Wortform: Selbsteinwechslung

- Unten der rechte Schuh, Größe 47, den Günter Netzer im Pokalfinale 1973 nach seiner berühmten Selbsteinwechslung trug - und auch hier irrt der Katalog, denn es ist eben nicht der Schuh, mit dem er kurz danach das Siegtor für Borussia Mönchengladbach schoß.
- In: o.A., Beidfüßige Lektüre, in: Frankfurter Allgemeine 24.08.2000, S. 46
- Nach: o.A., Beidfüßige Lektüre, in: F.A.Z.-Buchkritik 2000, Frankfurt a.M.: Frankfurter Allgemeine Zeitung GmbH 2000

https://www.youtube.com/watch?v=yPVpJIYWTIg
http://www.zehn.de/netzer-wechselt-sich-selbst-ein-2105602-2

Fun

- http://www.zehn.de/netzer-erhaelt-den-adolf-grimme-preis-2105602-9
- Netzer: "Ich sag ja, Sie hören mir nie zu!" Delling: "In Ihrem Alter merken Sie doch gar nicht mehr, ob jemand Ihnen zuhört.“

- http://www.zehn.de/netzer-bezwingt-england-2105602-3