TECHNISCHE
UNIVERSITÄT
DRESDEN

Fakultät Informatik, Institut für Software- und Multimediatechnik, Lehrstuhl für Softwaretechnologie

Softwaretechnologie II

Lecture 2 – Software Development as Engineering Activity:

Software Engineering Scenarios
A run through the engineering life cycle
Engineers and Entrepreneurs

Prof. Dr. U. Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie
http://st.inf.tu-dresden.de

WS 13-0.3, 23.10.2013

Obligatory Reading

- Balzert Introduction
- Maciaszek/Liong Chap. 1
- Ghezzi Chap 5+7 or
- Pfleeger Chap 2+4
- Wolfgang Hesse, Heinrich C. Mayr. Modellierung in der Softwaretechnik: eine Bestandsaufnahme. Informatik Spektrum 31(5), Springer-Verlag 2008
- Ed Seidewitz. What models mean. IEEE Software, 20:26-32, September 2003.

Wie man sich selbständig die Literatur erarbeitet

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:
  - 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.
- JR Abrial, Stephan Hallerstede. Refinement, decomposition, and instantiation of discrete models: Application to Event-B. Fundamenta Informaticae, 2007

Successful Engineers and Entrepreneurs

- Konrad Zuse. Mein Lebenswerk. Springer. A MUST for every student.
- Michael Lewis. The New New Thing. A book about how Jim Clark, Netscape founder, founded Healtheon. Coronet Books, Hodder & Stoughton
- R. Würth. Skript on Entrepreneurship. Interfakultatives Institut für Entrepreneurship. TU Karlsruhe. http://www.iep.uni-karlsruhe.de/260.php
- Klaus Kemper. Heinz Nixdorf. Verlag Moderne Industrie.
  - The Nixdorf foundation donated given 2 chairs to the department (multimedia, computational engineering)
- The Google story.
- Steve Jobs. about Apple. (There are several books available)
- Bill Gates. The Way Ahead. (dtsch. Der Weg nach vorn. Die Zukunft der Informationsgesellschaft) Autobiography. Hoffmann&Campe.
- D. Brandes. Konsequent einfach. Die Aldi Erfolgsstory. Heyne-Verlag.
- David Thielen. Die 12 simplen Erfolgsgeheimnisse von Microsoft. Econ-Verlag
- W. Wiedeking. Anders ist besser. Ein Versuch über neue Wege in Wirtschaft und Politik. Piper-Verlag, München 2006.
- D. Tapscott. Wikonomics. 2007

Scenario of Running Example

- You are a project manager in Hamann/Becker Car Radios, Inc, Karlsruhe, Germany and 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?

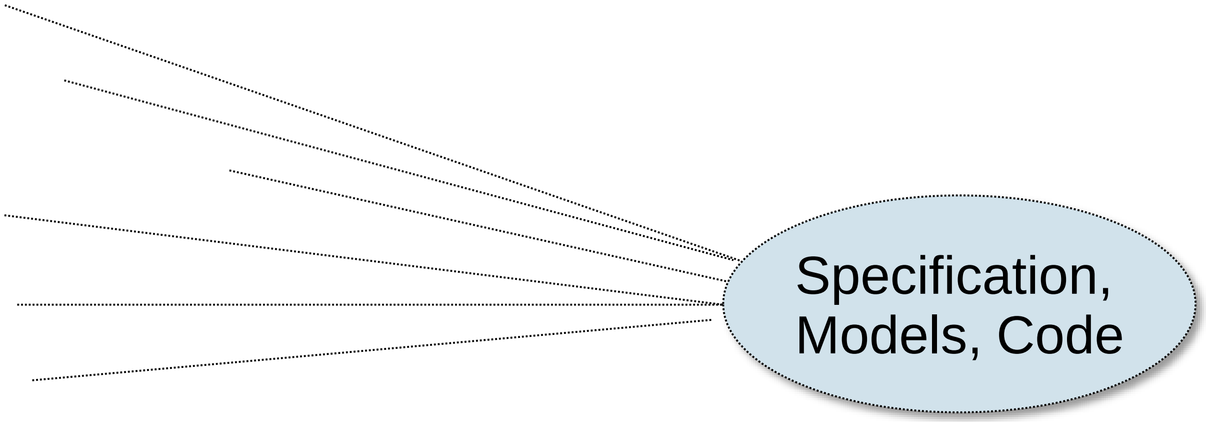
Softwaremanagement

- 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
- Improvement
- Sell



Specification, Models, Code

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

The (Software) Engineer's Toolkit (1)

- Model a reality (a domain or a system in the world): Descriptive modeling
  - Describe or specify
  - World and problem modeling vs. system modeling
- Specify a system: Prescriptive modeling
  - Specifying features and requirements of a system
- Analyze (measure) a reality (a model or a system)
  - Identifying the problem (problem analysis, goal analysis, 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

The (Software) Engineer's Toolkit (2)

- 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)
  - Compose a system from components
  - Describing the infinite and the unknown with finite descriptions
  - Structure a model (making the model more clear)
    - Refinement (making the model more precise and detailed)
    - Abstraction (leaving out detail, focusing on the essential)
    - Domain Transformation (changing representation of model)
  - Reuse parts of products
    - Engineer a product line (product family)

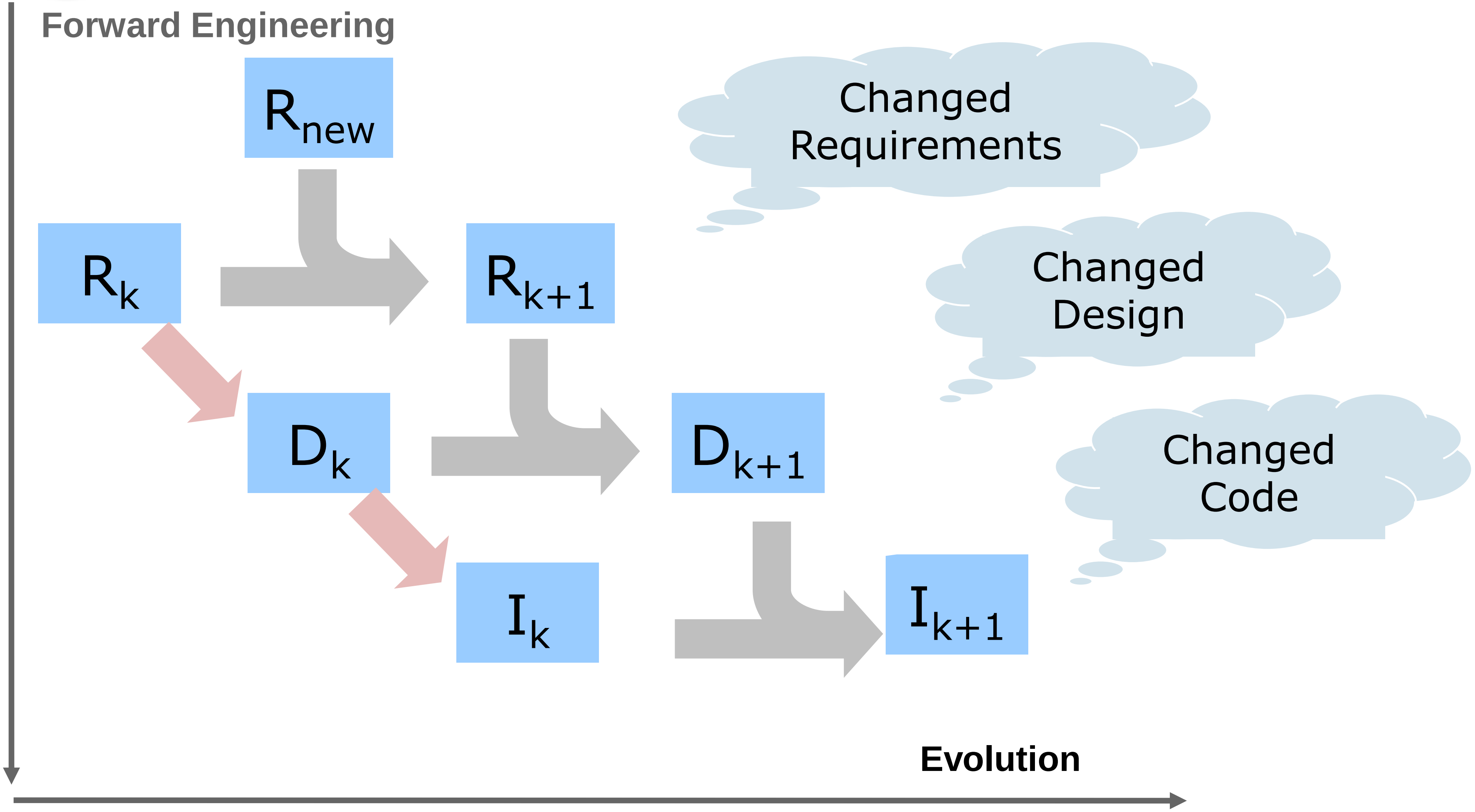
The (Software) Engineer's Toolkit

- Validate hypotheses on the product
  - Experimentation (empirical software engineering)
  - Checking (consistency, integrity, wellformedness, completeness, soundness)
  - Testing
  - Proving (formal software engineering, formal methods)
  - Statistics (not covered here)
- Improve the product
  - Reverse engineer
  - 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?

2.1. Scenarios of Software Engineering

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

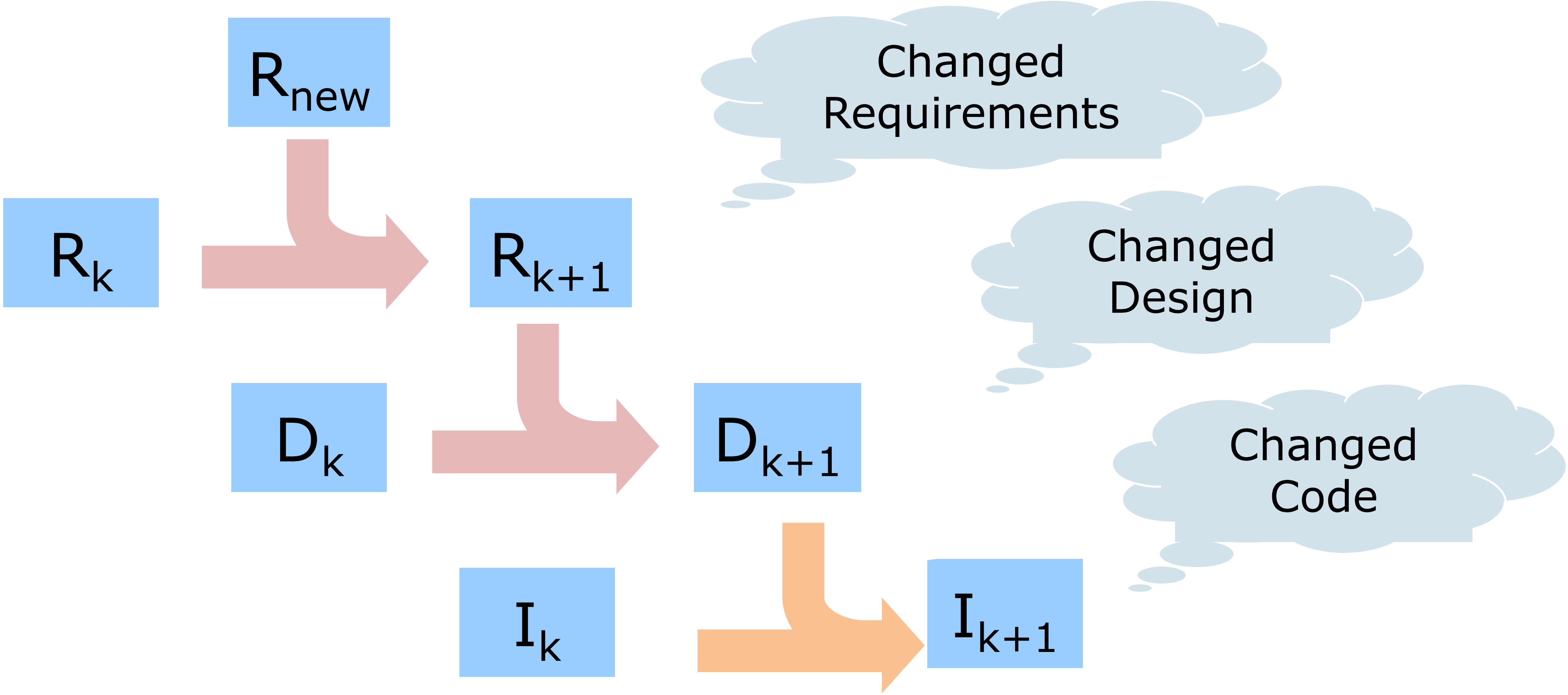
Forward Engineering and Evolution



With CASE tools, implementations can be generated from implementation models

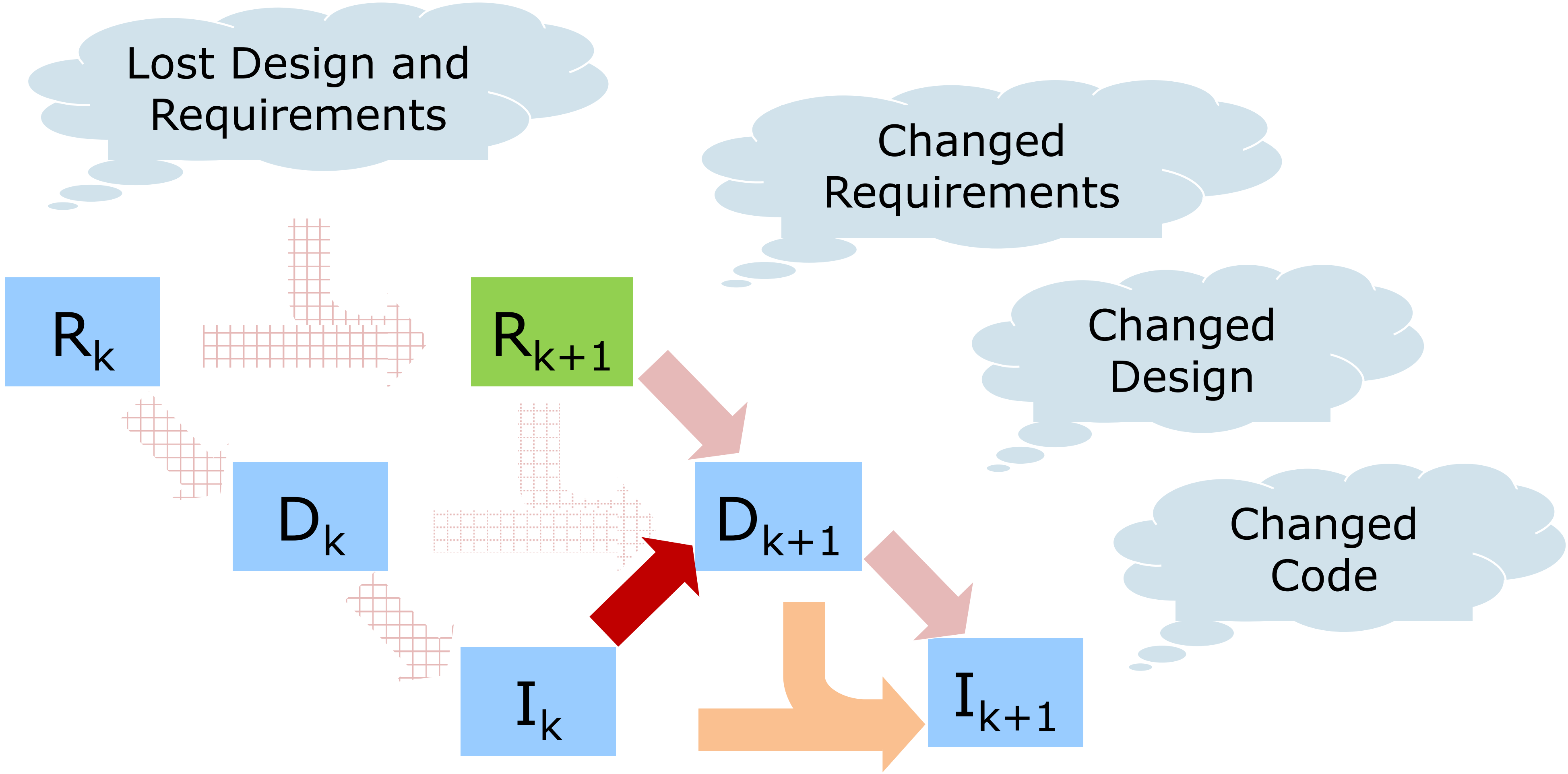
Software Evolution

- 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



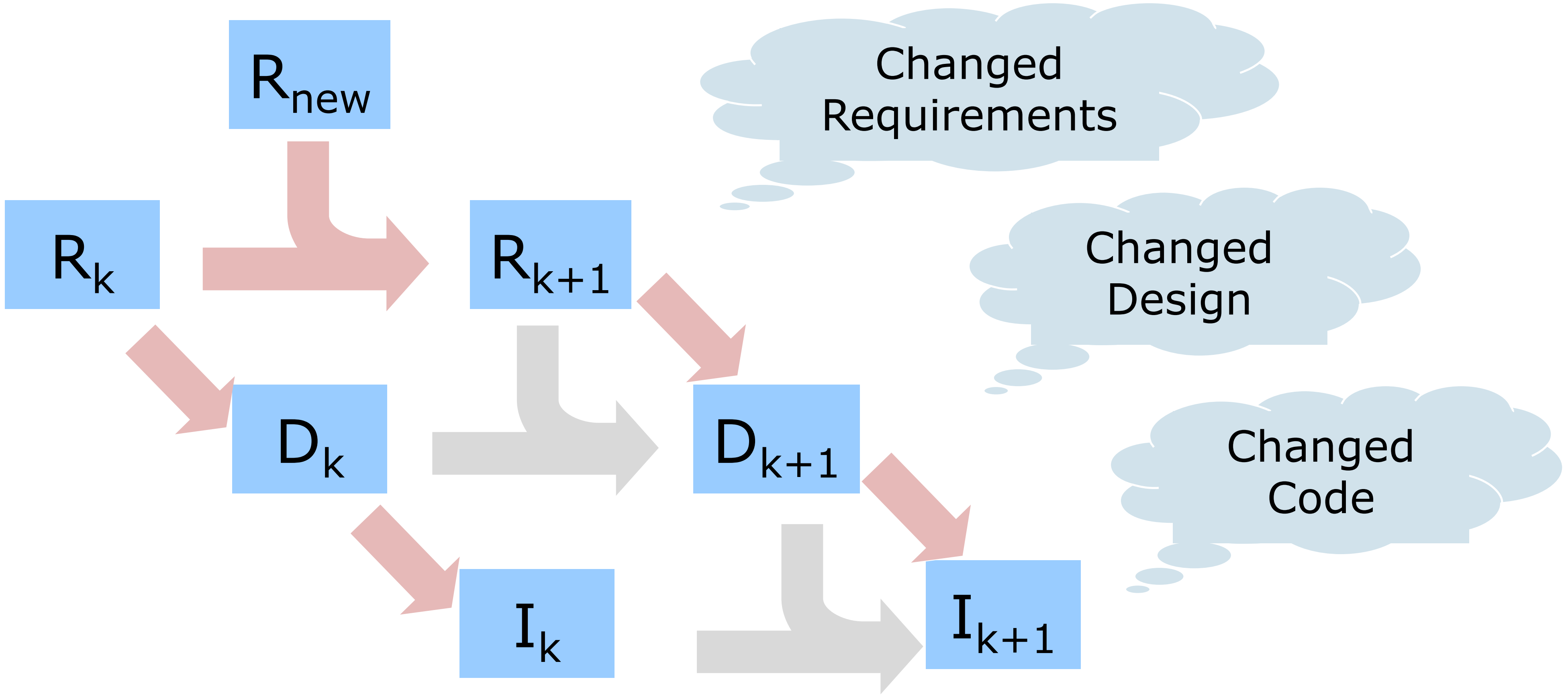
Software Reengineering

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



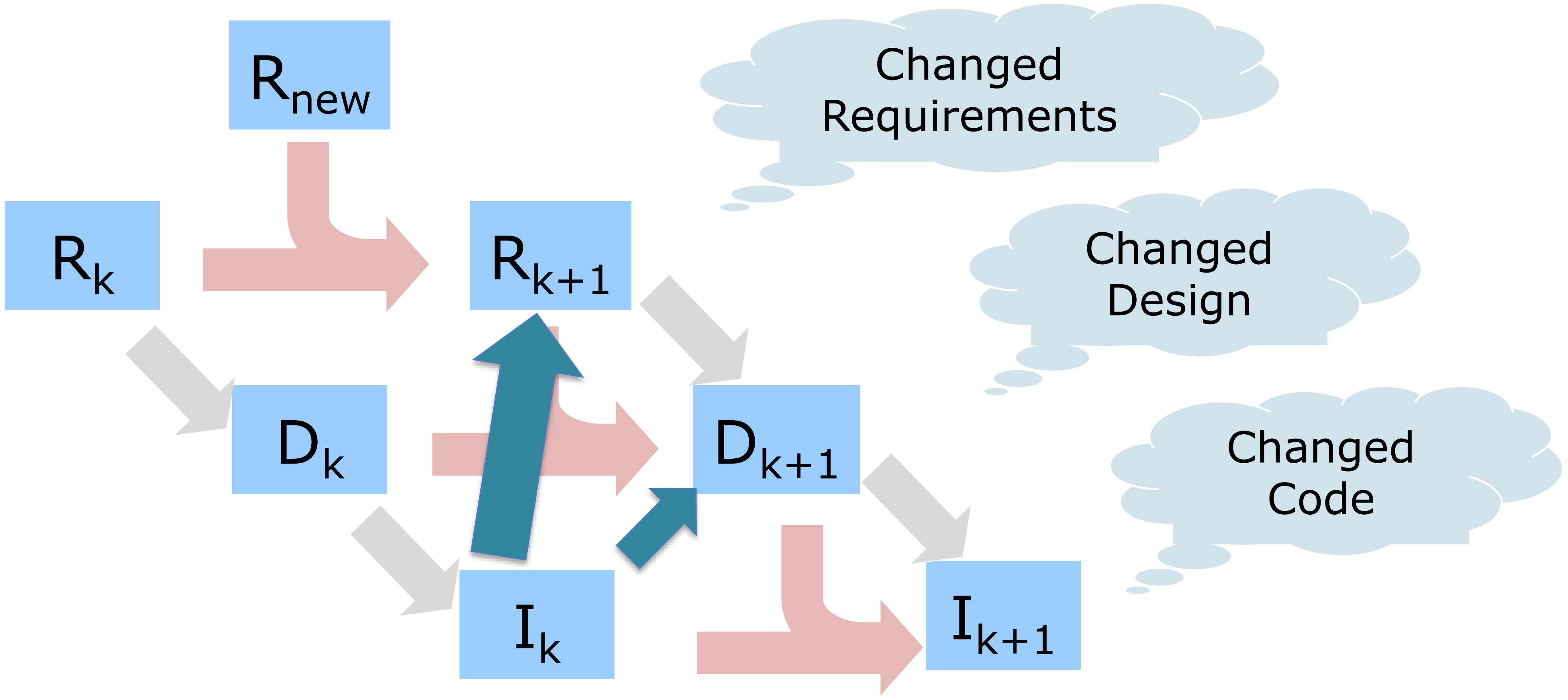
The Dream: Automated Programming

- 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



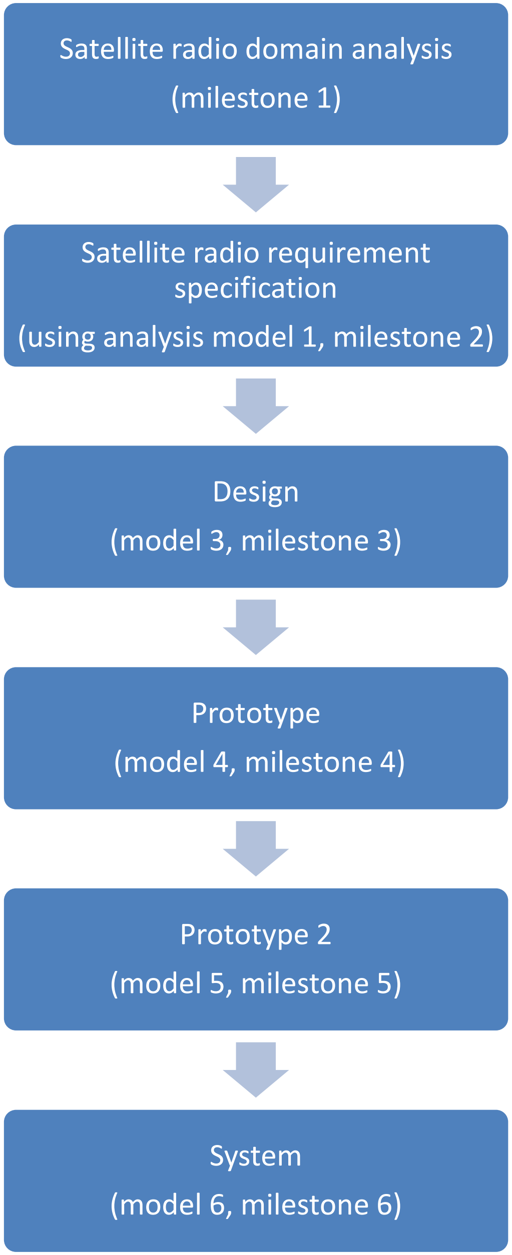
2.2 A Run Through an Engineering Cycle

Engineering Cycle: Steps

1. Analysis
2. Prediction
3. Construction
4. Validation
5. Improvement
6. Selling Software

Analysis

- From requirements to product
  1. Analyze problems to understand what to do
  2. Specify a solution and realize (construct) it
  3. For 1. and 2. Model the world to master it
- Steps
  - Put requirements in a requirement specification (requirements models)
  - Step by step through different design models
    - ... until we arrive at the implementation (system) model



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)

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

Other definition

- “A model is an abstraction of something for the purpose of understanding before building it” (J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1991)

To Produce Software, We Model

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

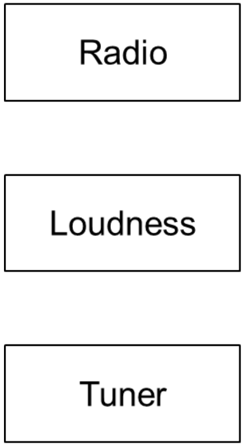
System Design

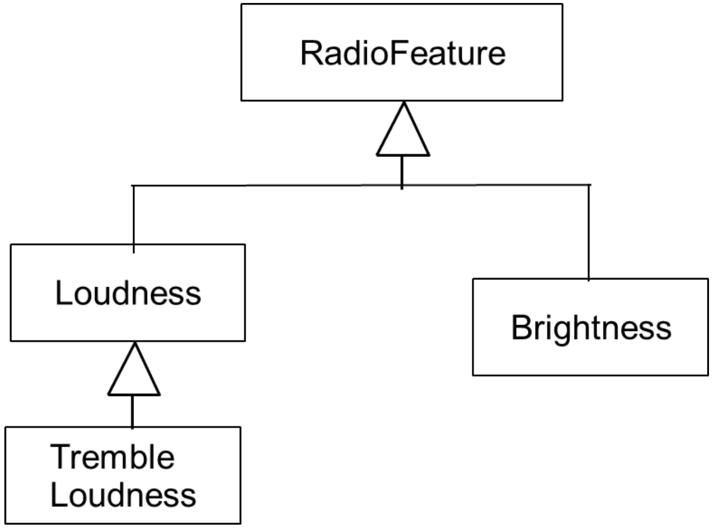
Satellite Radio

Software-controlled embedded system

Descriptive Models: Glossaries, Classifications and Taxonomies

- 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** = **taxonomy** + associations, class and relation expressions, and well-formedness constraints

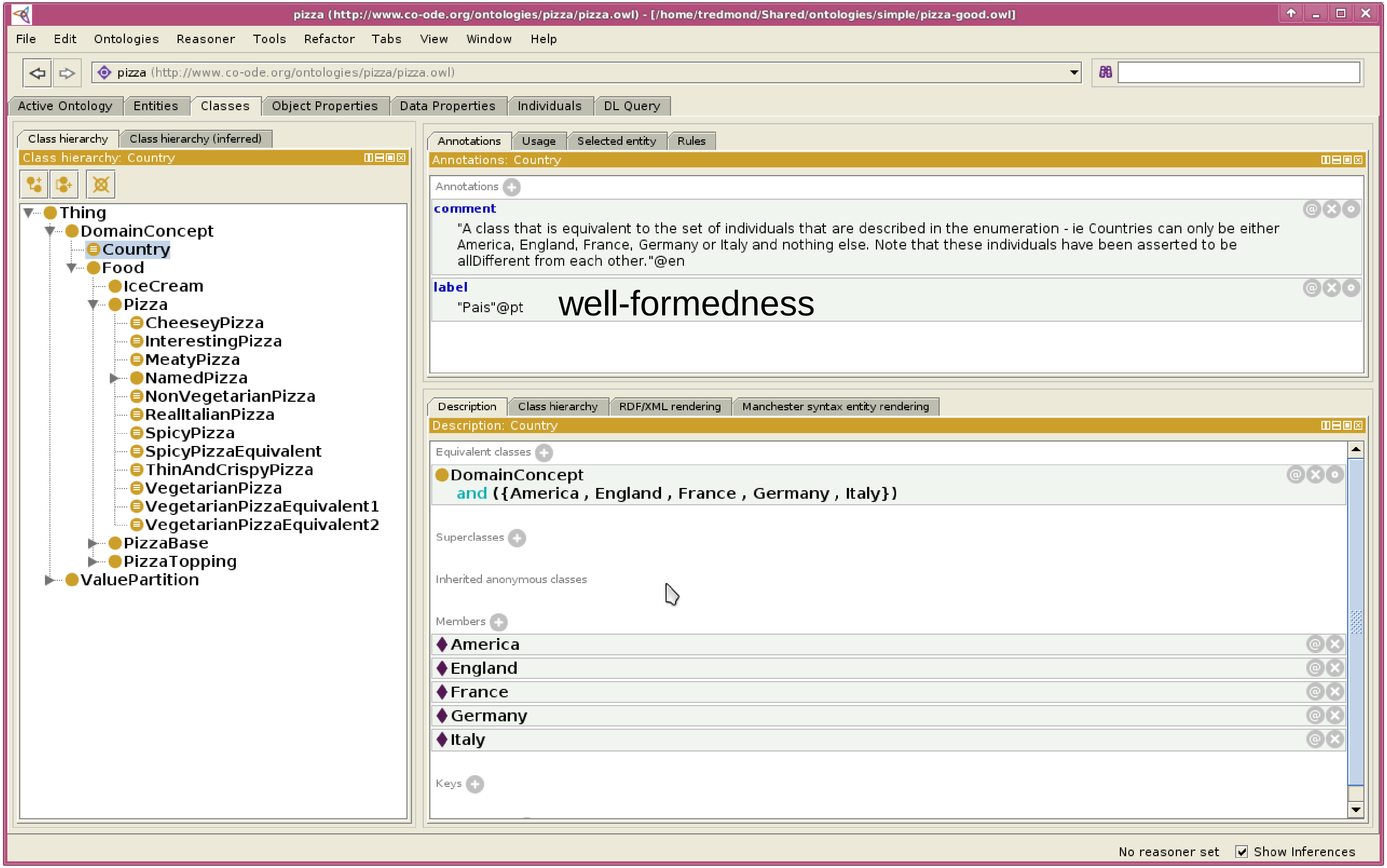




Ontologies as Standardized Domain Models

- 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
  - 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)

Protege Editor


pizza (http://www.co-ode.org/ontologies/pizza/pizza.owl) - [/home/tredmond/Shared/ontologies/simple/pizza-good.owl]
File Edit Ontologies Reasoner Tools Refactor Tabs View Window Help
pizza (http://www.co-ode.org/ontologies/pizza/pizza.owl)
Active Ontology | Entities | Classes | Object Properties | Data Properties | Individuals | DL Query
Class hierarchy | Class hierarchy (inferred)
Class hierarchy: Country
Annotations | Usage | Selected entity | Rules
Annotations: Country
Description | Class hierarchy | RDF/XML rendering | Manchester syntax entity rendering
Description: Country
No reasoner set
☑ Show Inferences

```
Thing
  DomainConcept
    Country
    Food
      IceCream
      Pizza
        CheeseyPizza
        InterestingPizza
        MeatyPizza
        NamedPizza
        NonVegetarianPizza
        RealItalianPizza
        SpicyPizza
        SpicyPizzaEquivalent
        ThinAndCrispyPizza
        VegetarianPizza
        VegetarianPizzaEquivalent1
        VegetarianPizzaEquivalent2
      PizzaBase
      PizzaTopping
  ValuePartition
```

Annotations

comment

"A class that is equivalent to the set of individuals that are described in the enumeration - ie Countries can only be either America, England, France, Germany or Italy and nothing else. Note that these individuals have been asserted to be allDifferent from each other."@en

label

"Pais"@pt

well-formedness

Equivalent classes

DomainConcept
and ({America , England , France , Germany , Italy})

Superclasses

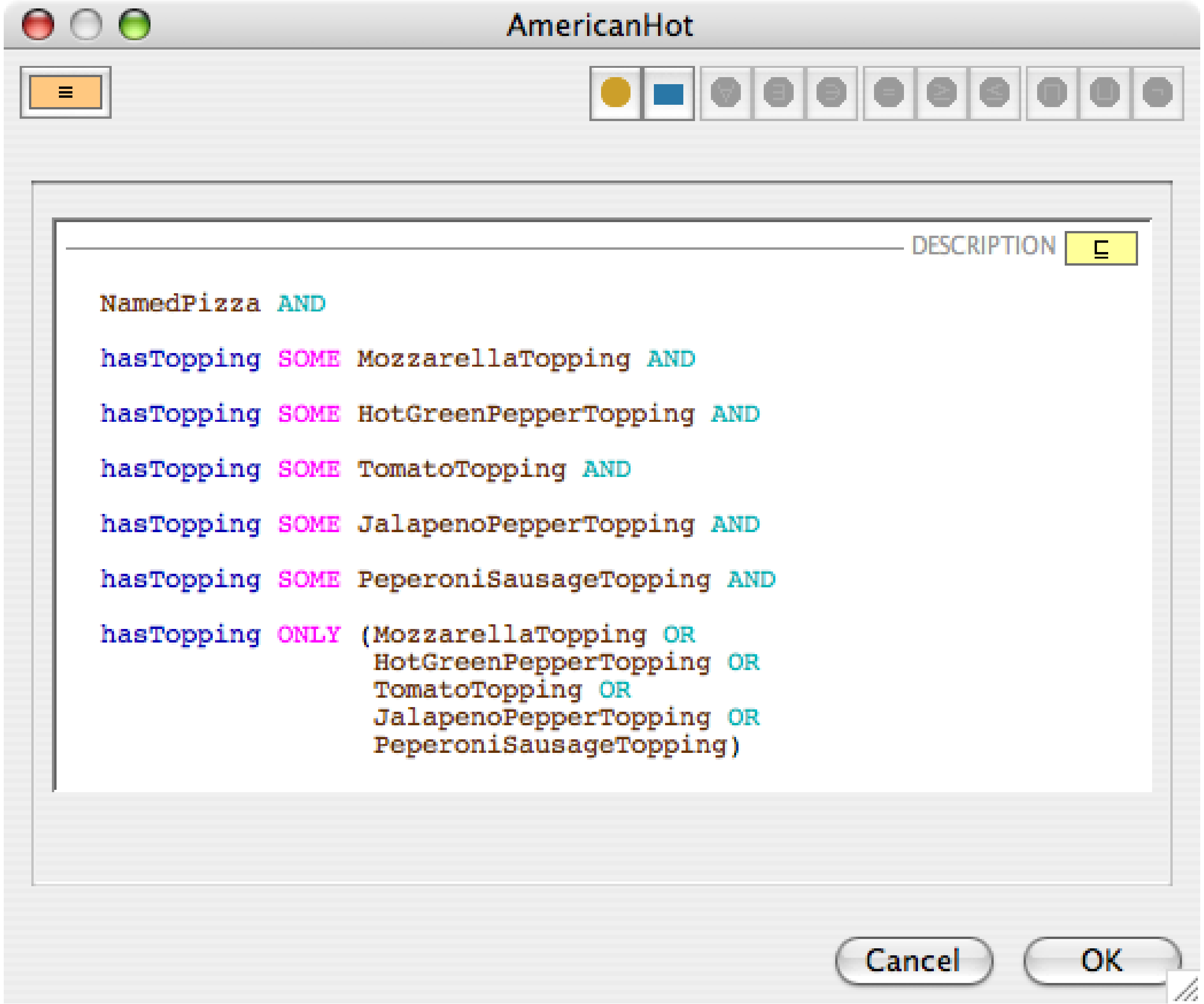
Inherited anonymous classes

Members

- America
- England
- France
- Germany
- Italy

Keys

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?

- 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 contains one or several models of the system
  - Models are abstract, partial representations of partial knowledge

Different Kinds of Specifications and Models (1)

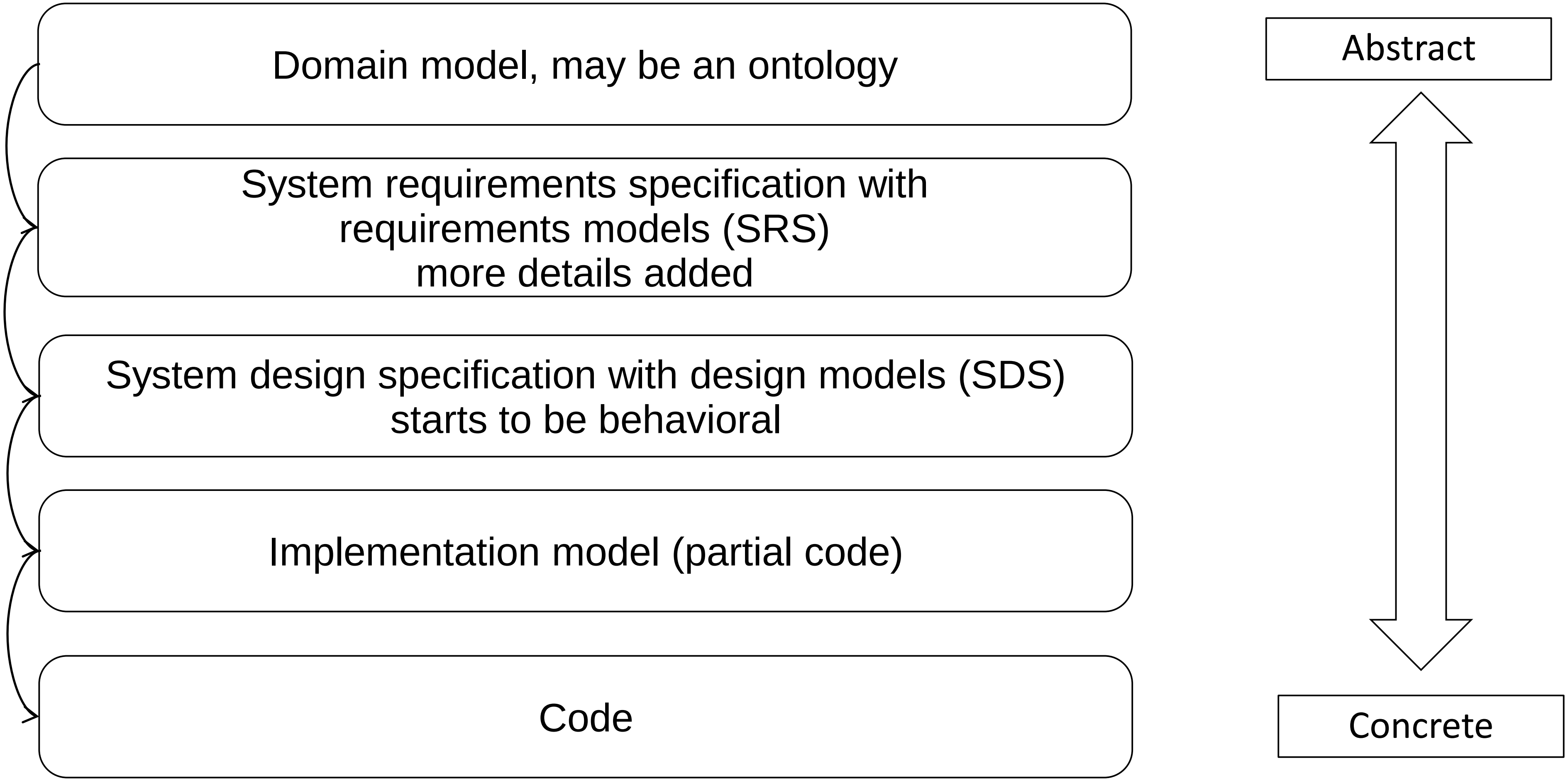
- Descriptive (Analysis) models
- Domain model:
  - Domain analysis is the process of identifying and organizing knowledge about the application domain
- “Real”-Problem model:
  - Usually, the requirement specification includes a problem model – to support description and solution of these problems
-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

Different Kinds of Specifications and Models (2)

- 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

Specifications and Models in Software Engineering

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


Domain model, may be an ontology
System requirements specification with requirements models (SRS) more details added
System design specification with design models (SDS) starts to be behavioral
Implementation model (partial code)
Code
Abstract
Concrete

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

Engineering Cycle: Steps

1. Analysis
2. **Prediction**
3. Construction
4. Validation
5. Improvement
6. Selling Software

Prediction

- 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** of the processes
    - **Fairness** of the processes
    - **Deadlocking** processes
  - Statecharts can be checked with model checkers
  - Real-time statecharts can be time-checked with real-time model checkers
- This area is called formal methods of software engineering
- Prediction is important for **critical software**:
  - Real-time software in embedded systems
  - Safety
  - Security and privacy
  - Energy efficiency

Engineering Cycle: Steps

1. Analysis
2. Prediction
3. **Construction**
4. Validation
5. Improvement
6. Selling Software

Construction with Refinement-Based Development

- Refinement
  - From Domain Model → Requirement Specification → Design Specification → Implementation Model → Code
    - Develop the next specification, starting from the previous ones
- Construction steps
  - 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** consistency of models
    - **Measure** quality and quantity of models
    - **Compose** from components

Questions for the Methods of Development (1)

- Elaboration (concretizations): Elaborate more details
  - Which Elaboration steps exist?
  - How do I know in which direction to elaborate?
- Refinements
  - Syntactic refinement
    - Replace a part of the model by something more fine-grained
  - Semantic refinement
    - Prove for a syntactic refinement that it is correct, i.e., either preserves semantics, or enrichtes semantics
  - 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

Questions for the Methods of Development (2)

- Rotations (Symmetry operations): Apply a semantics-preserving change
  - Rotate
    - Symmetry operations
    - Semantics-preserving operations
  - Restructure (refactor)
    - Rearrange 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

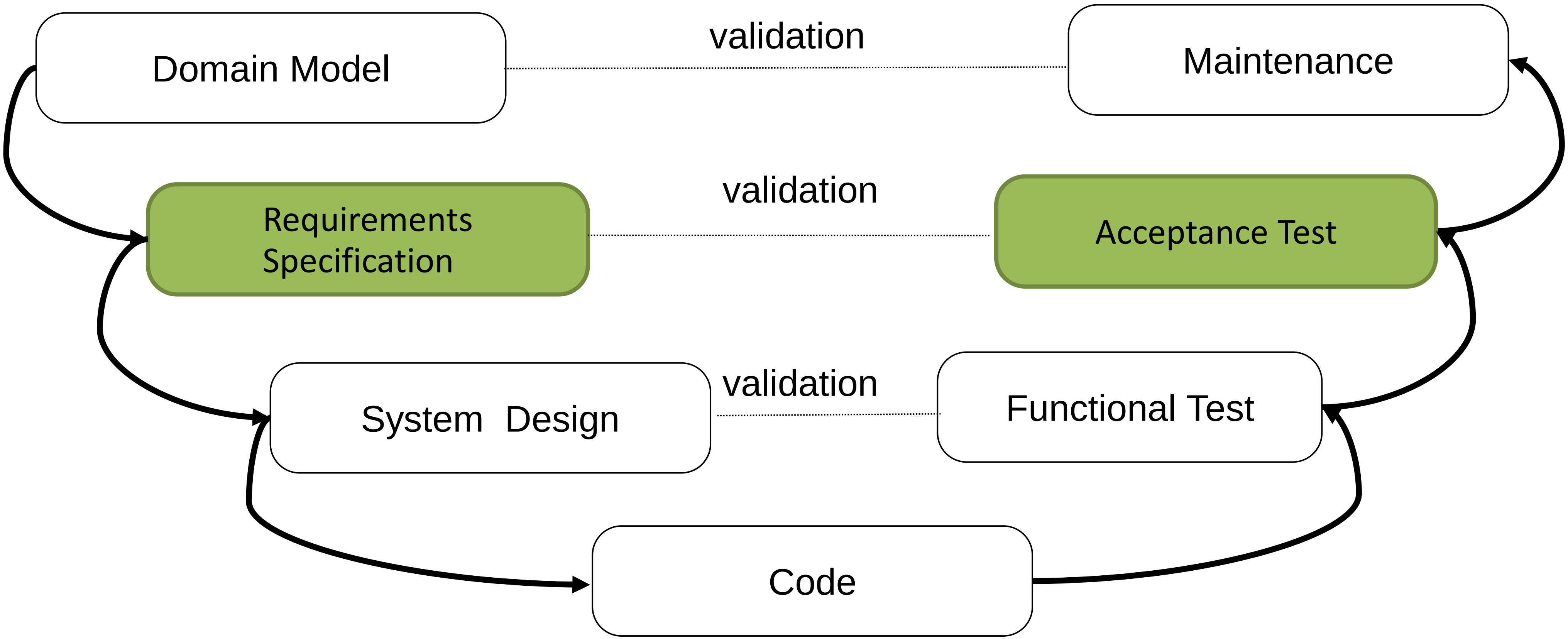
- Reuse by composition: Engineers try to reuse well-established solutions
  - Components (CBSE)
  - Design patterns
  - Models (model-driven architecture)
  - Best practices
- Reuse: to simplify system construction
  - To save costs
  - To reduce testing effort

Engineering Cycle: Steps

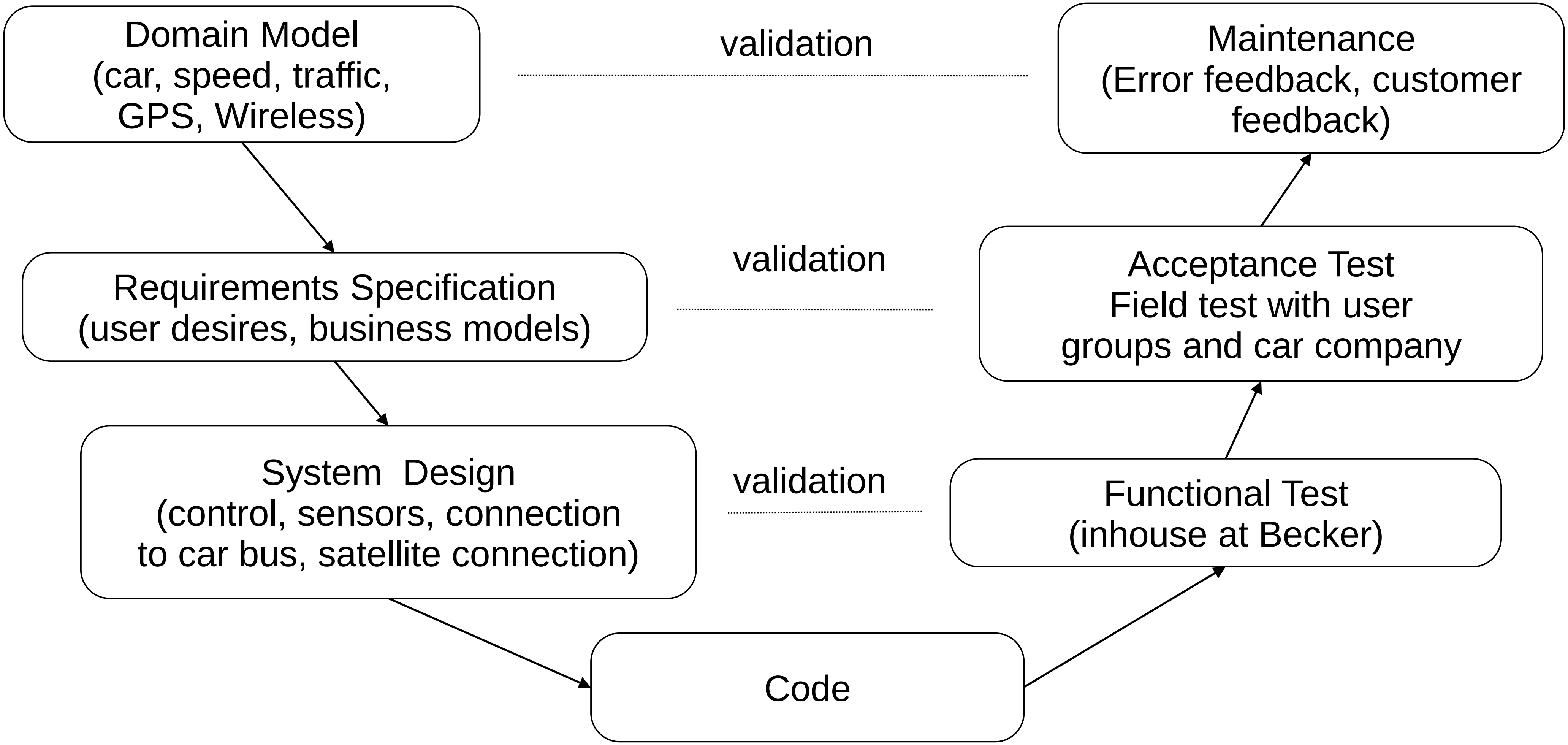
1. Analysis
2. Prediction
3. Construction
4. **Validation**
5. Improvement
6. Selling Software

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

- All specifications and models have to be validated or formally verified.
  - Detailed models against more abstract models
  - Implementations against specifications
- Result: A V-like software development process



Validation of the Satellite Radio in the V-Model



Engineering Cycle: Steps

1. Analysis
2. Prediction
3. Construction
4. Validation
5. **Improvement**
6. Selling Software

Improvement

- 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

Engineering Cycle: Steps

1. Analysis
2. Prediction
3. Construction
4. Validation
5. Improvement
6. Selling Software

The Best Seller Is...

- .. the one who solves a problem best
- .. 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

??

2.3 Solving Problems – 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

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
- Aims realistic?

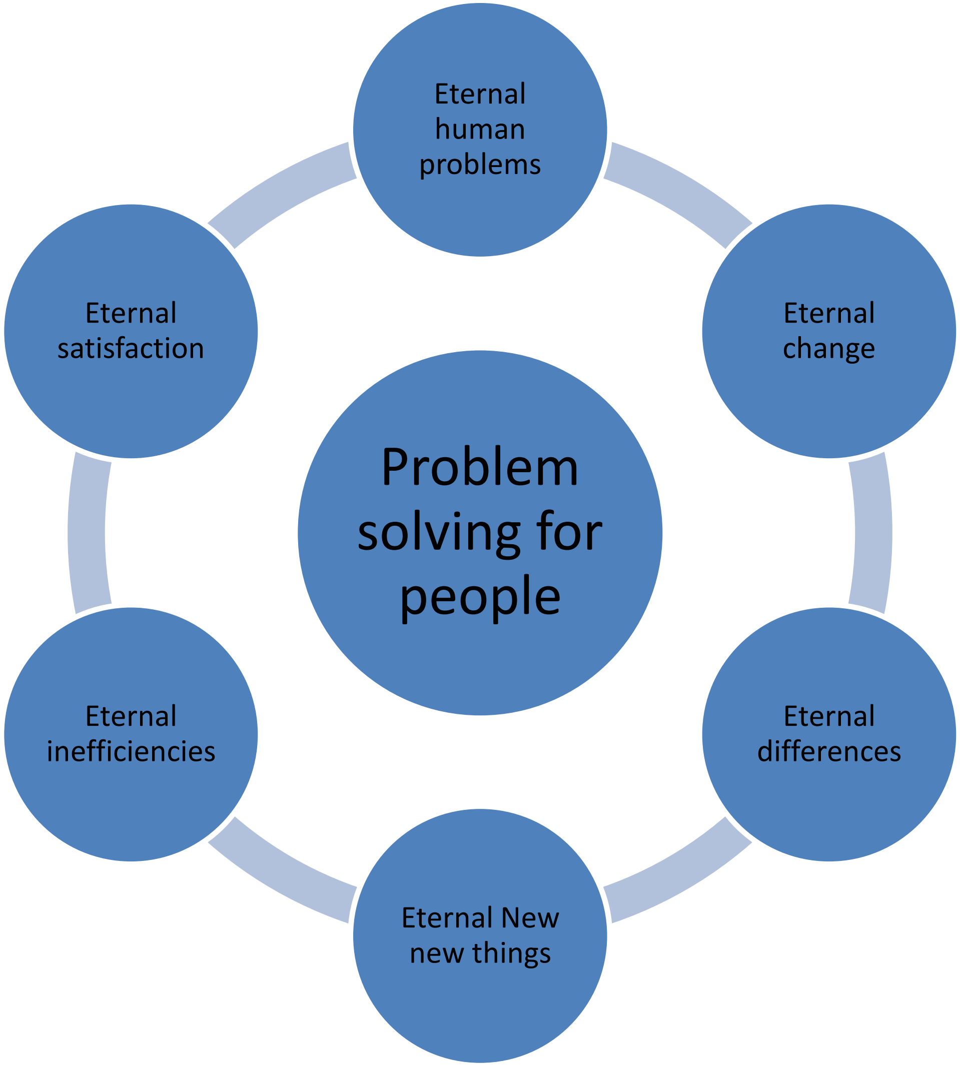
Problem Solving – A Task for the Engineer and also the Salesman

- 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

2.3.2 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
  - Apply ZOPP to the Problem Area
    - Stakeholder analysis is important
    - Find out about the problems of a stakeholder group
    - Find out about their goals
    - From there, derive the product
- Try to find pain problems, because they create pressure on the customer

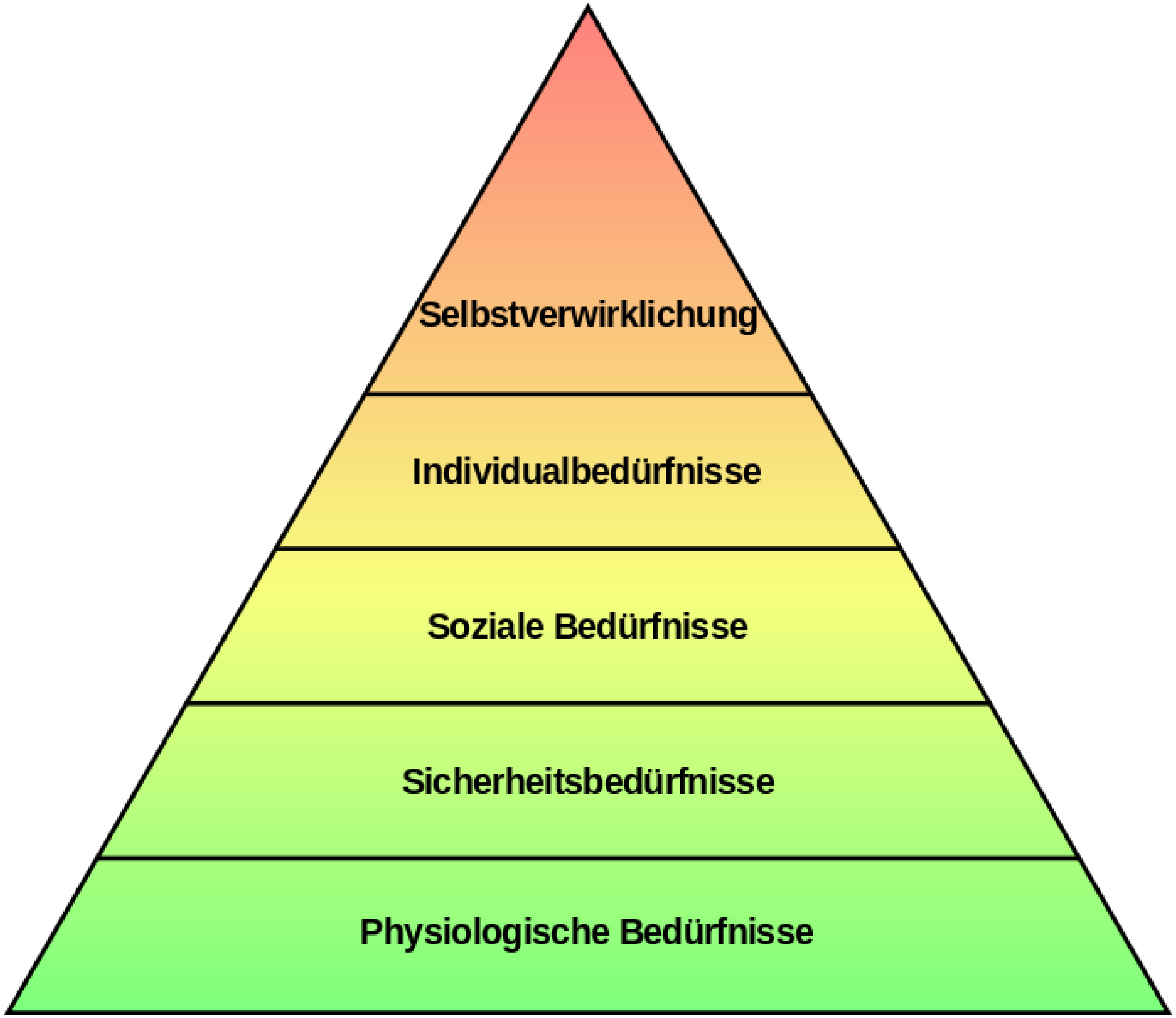
Exploit the Eternal Human Problems and Needs

- Hunger, Food, Restaurants, ...
- Love, Relationship
- Hobby
- Beauty
- Exhibiting oneself
- Housing
- Save money
- Holidays
- Overcoming the Space problem: Car, Flights,...
- Simplifying complex things
  - Overcoming bureaucracy
- Communication
- 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 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

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.

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.

2.3.2 What to Sell as a Software Engineer

Different Types of Things to Sell

- 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
- 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 (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 n. 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
  - Easy to switch to another company
  - 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.
  - 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

- What you might also sell:
  - Consultancy
    - Sell your know-how
      - Analysis studies on a market, trend or strategy
  - Service
    - Exampl: Requ.analysis, testing, maintenance, modernization, reengineering
    - Many big companies have their focus there: IBM
  - 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 (Anwendungslandschaft) with integration of many tools**

Chapter testing, requirements analysis, modeling, model structuring

Chapter “design methods”

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)”.