2. Software Development as Engineering Activity

1. **Software Engineering Scenarios**
2. **A simple run through the life cycle**

Prof. Dr. U. Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
Gruppe Softwaretechnologie
http://st.inf.tu-dresden.de
WS 11-0.3, 17.10.11

Obligatory Reading

- Balzert Introduction
- Maciaszek/Liong Chap. 1
- Ghezzi Chap 5+7 or
- Pfleeger Chap 2+4
- Ed Seidewitz. What models mean. IEEE Software, 20:26-32, September 2003.
  - http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1231147&tag=1

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
  - http://dl.acm.org/citation.cfm?id=1365974&CFID=49627514&CFTOKEN=73132377

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?"

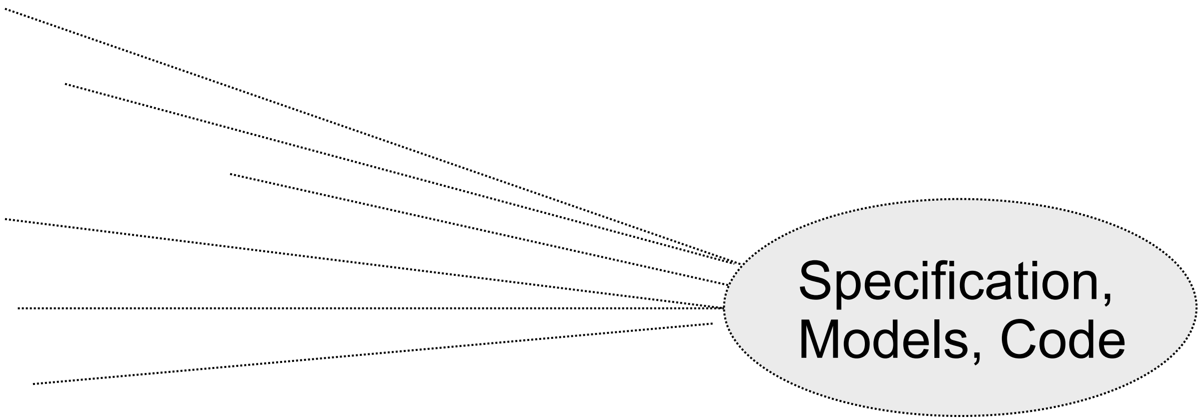
First Ideas

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

What is Software Engineering?

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



> Software engineers model, measure, predict, build, validate, improve, and sell

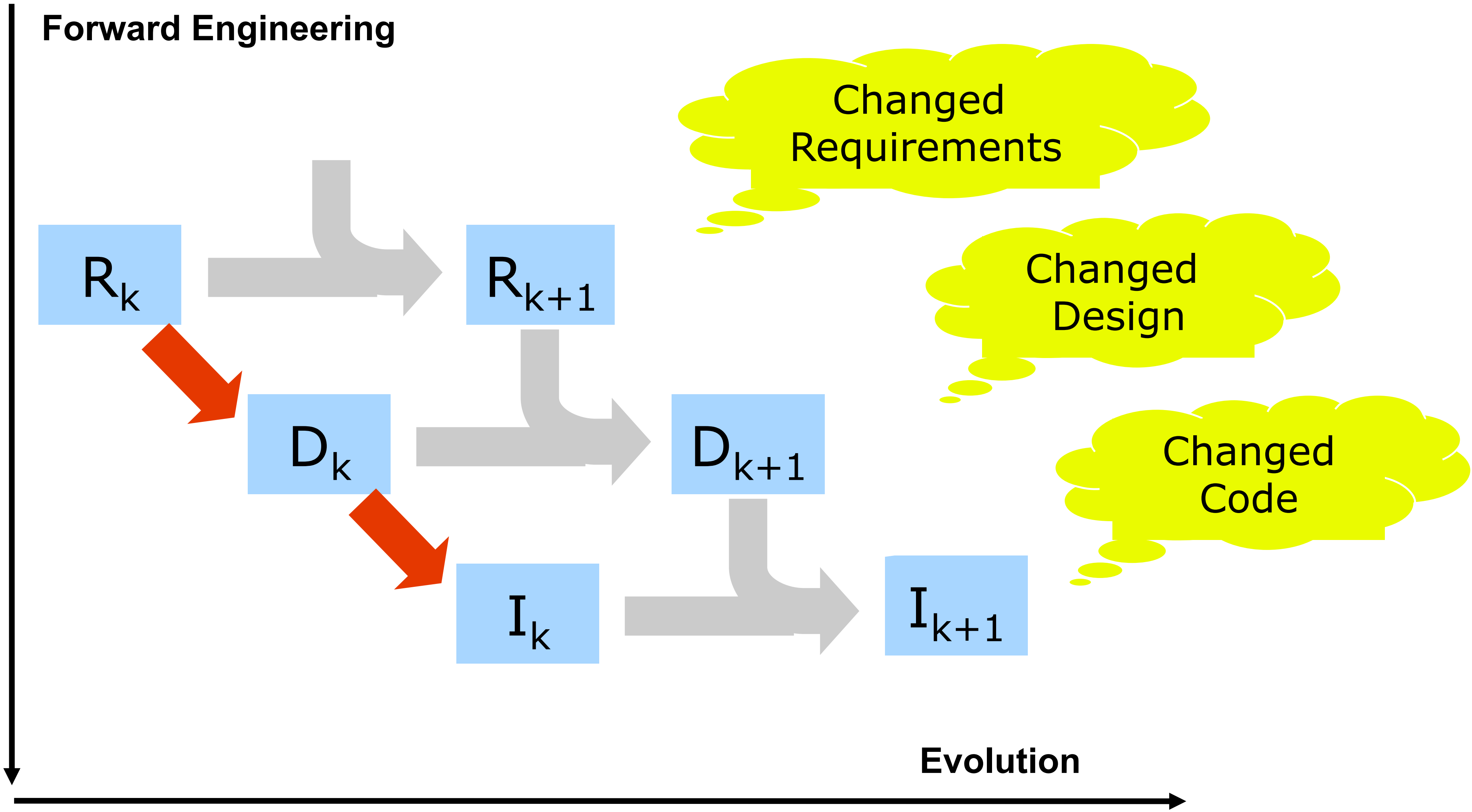
- Model a reality (a domain or a system)
  - Describe or specify
  - World and problem modeling vs. system modeling
- **Analyze** (measure) a reality (a model or a system)
  - Identifying the problem (problem analysis, goal analysis, risk analysis)
  - Measuring (Software metrics)
  - Searching and finding
  - Controlling
- **Predict** features of a product from the model (form hypotheses, prove)
  - Specifying features and requirements of a system
  - Forming hypotheses about the system
- **Construct** a product (realize, develop, invent, build)
  - **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?
  - Often, engineers are good technicians, but fail to sell the products

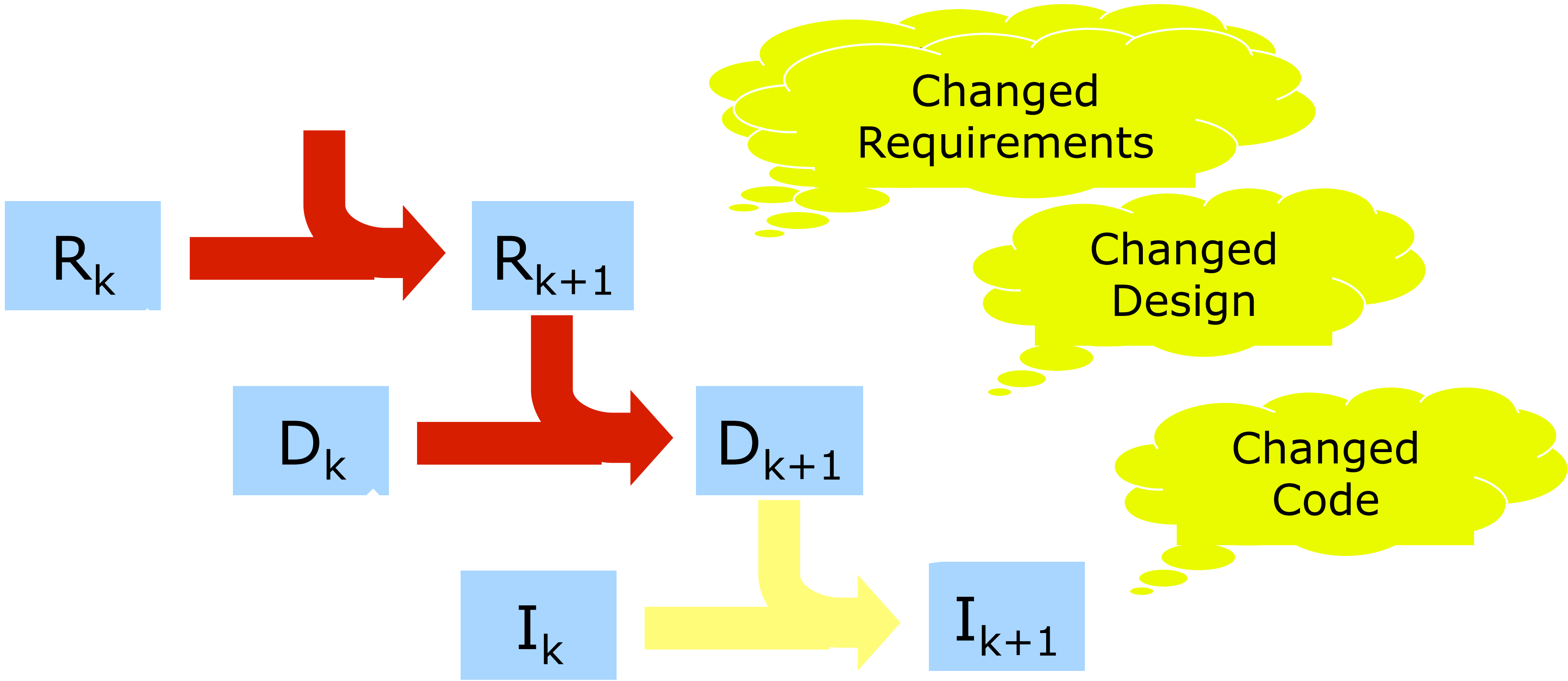
Forward Engineering, Backward Engineering, Improvement, Round-Trip Engineering

2.1. SCENARIOS OF SOFTWARE ENGINEERING

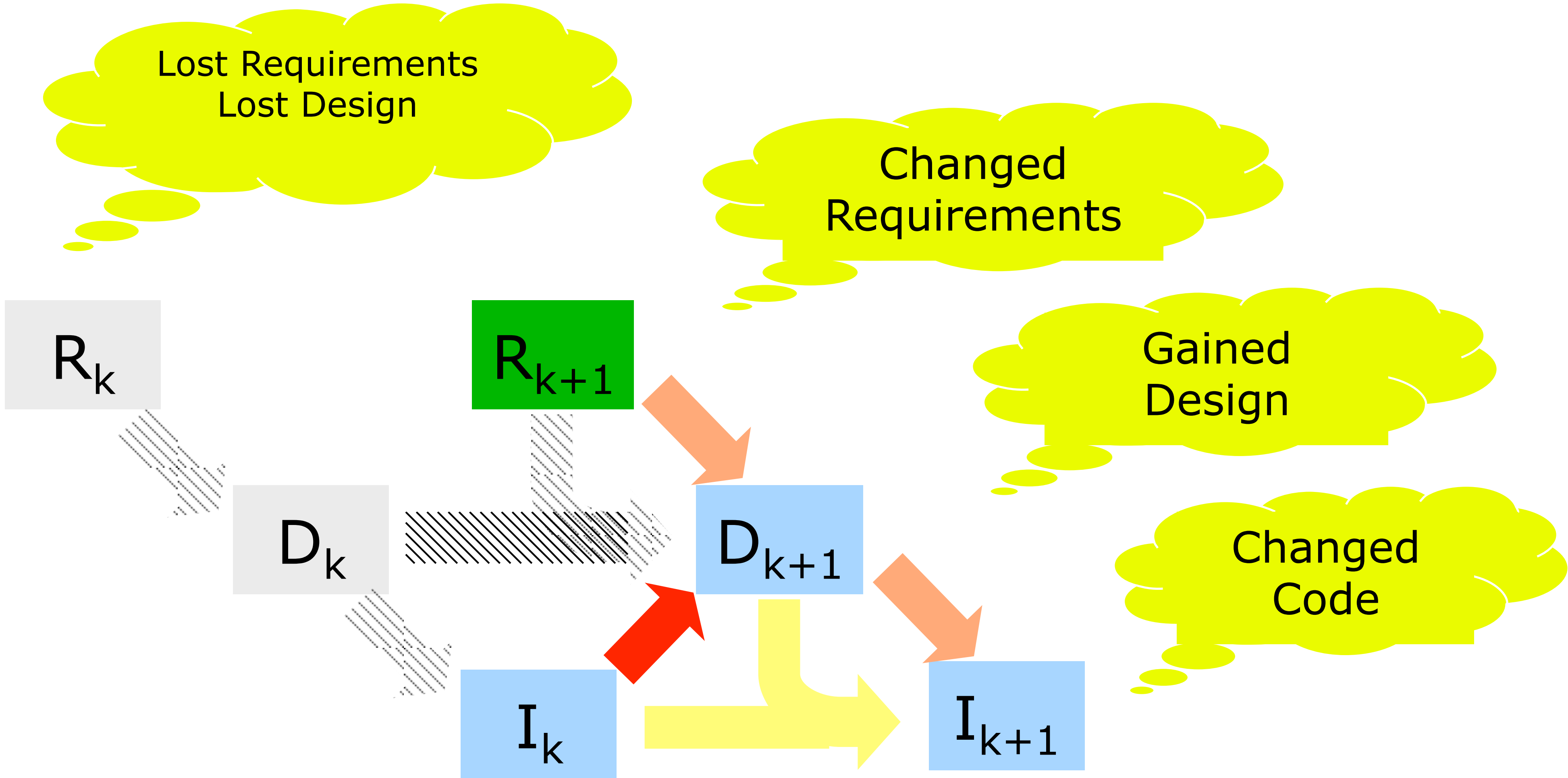


With CASE tools, implementations can be generated from implementation models

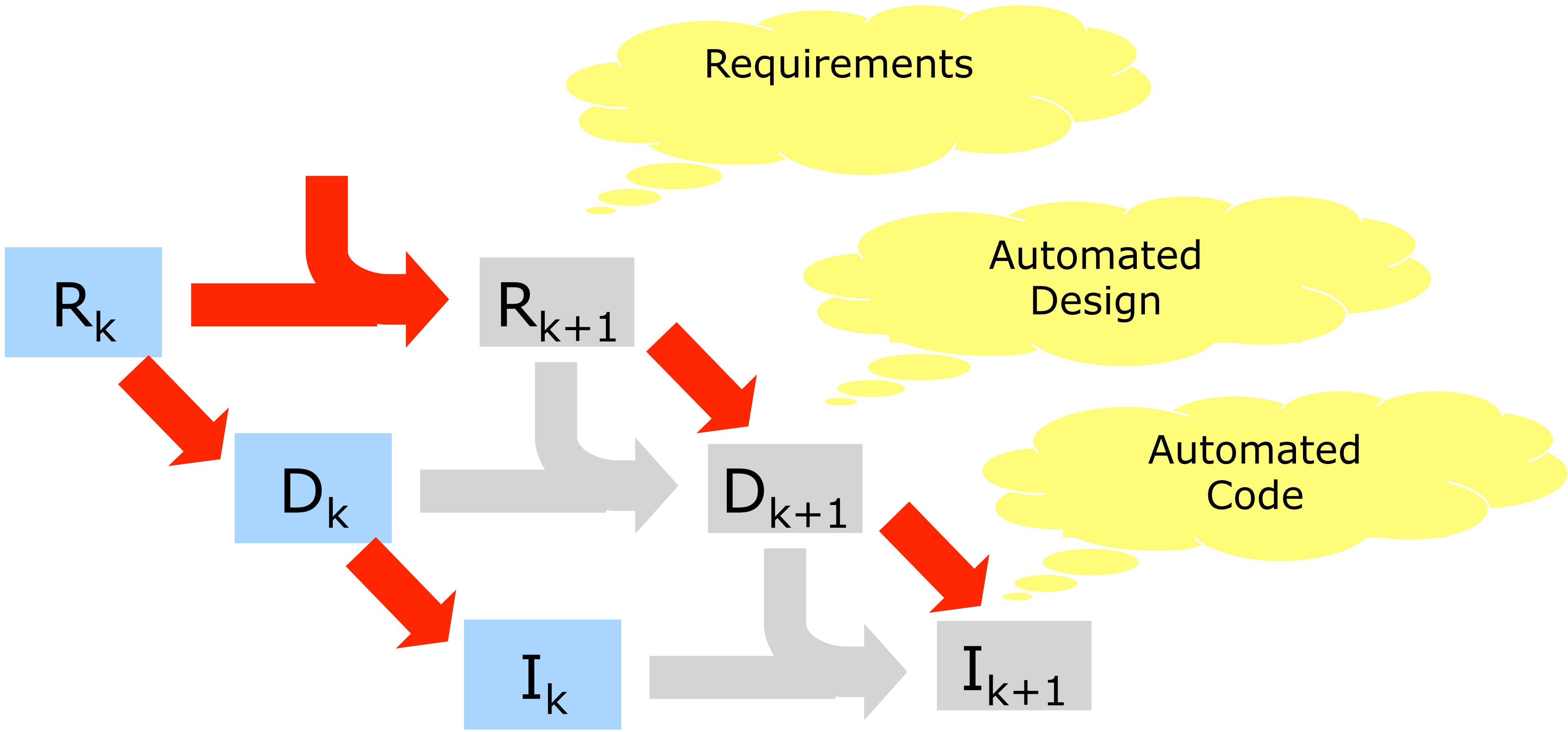
- Changed requirements require refactoring and extensions



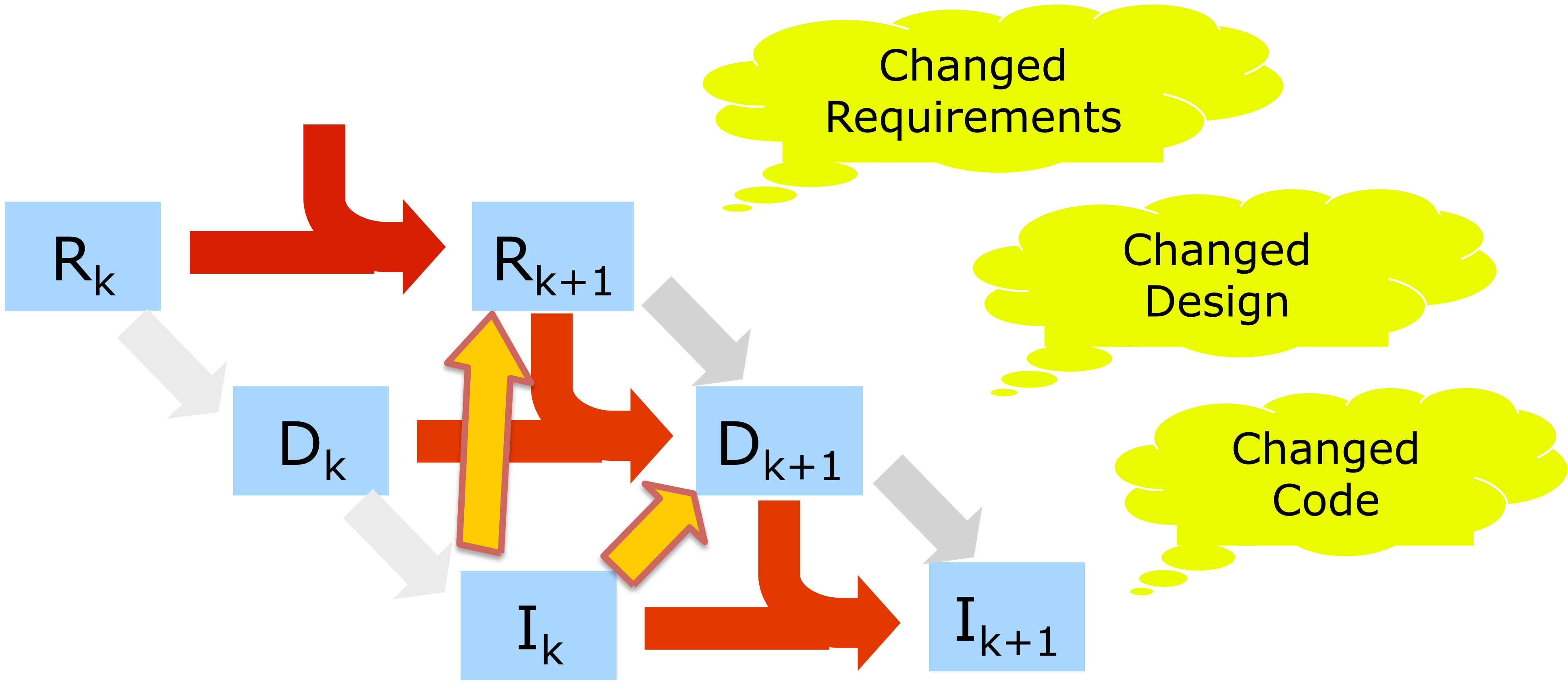
- **Reverse Engineering** attempts to recover design from code
- **Reengineering** uses the gained design for further forward engineering



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

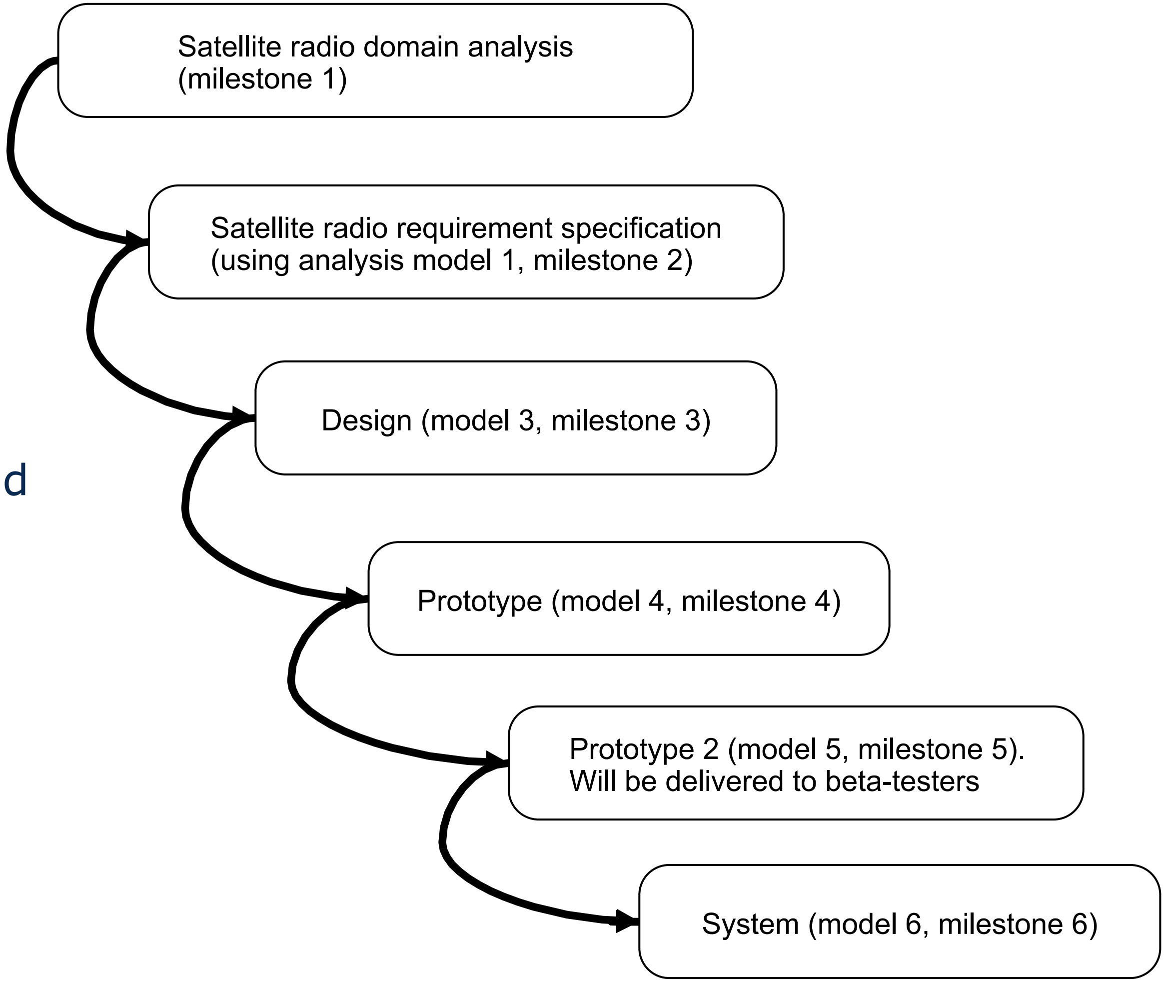


- **Round-trip engineering** combines forward and reverse engineering
  - It allows for editing on all levels, keeping all artefacts consistent



2.2 A RUN THROUGH AN ENGINEERING CYCLE

- 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)



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*
    - *To **change, manage, and control** that reality*
- 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
- A descriptive model allows to understand a reality
- A prescriptive model allows to change, manage, and control a reality

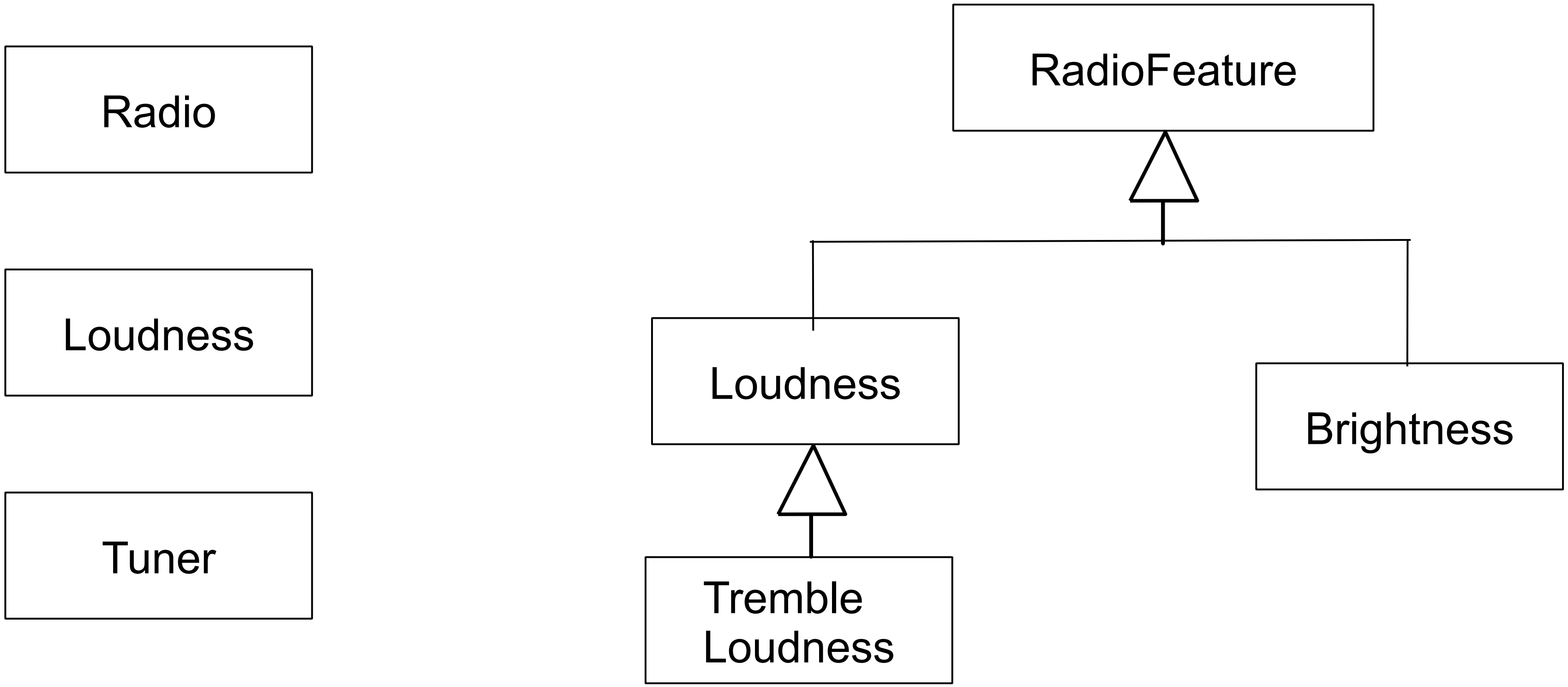
- Question: what does this mean for the Satellite radio?

- Software construction uses two kinds of models

The World	Software Systems
Problem Domain	*System Domain*
Problem Analysis	*System Design*
What is the problem?	What is the solution?
Problem model (Analysis model)	System model (Design model)
Models the *problem reality*	Models the *system reality*
Descriptive (analytic) models	Prescriptive models (specifications)

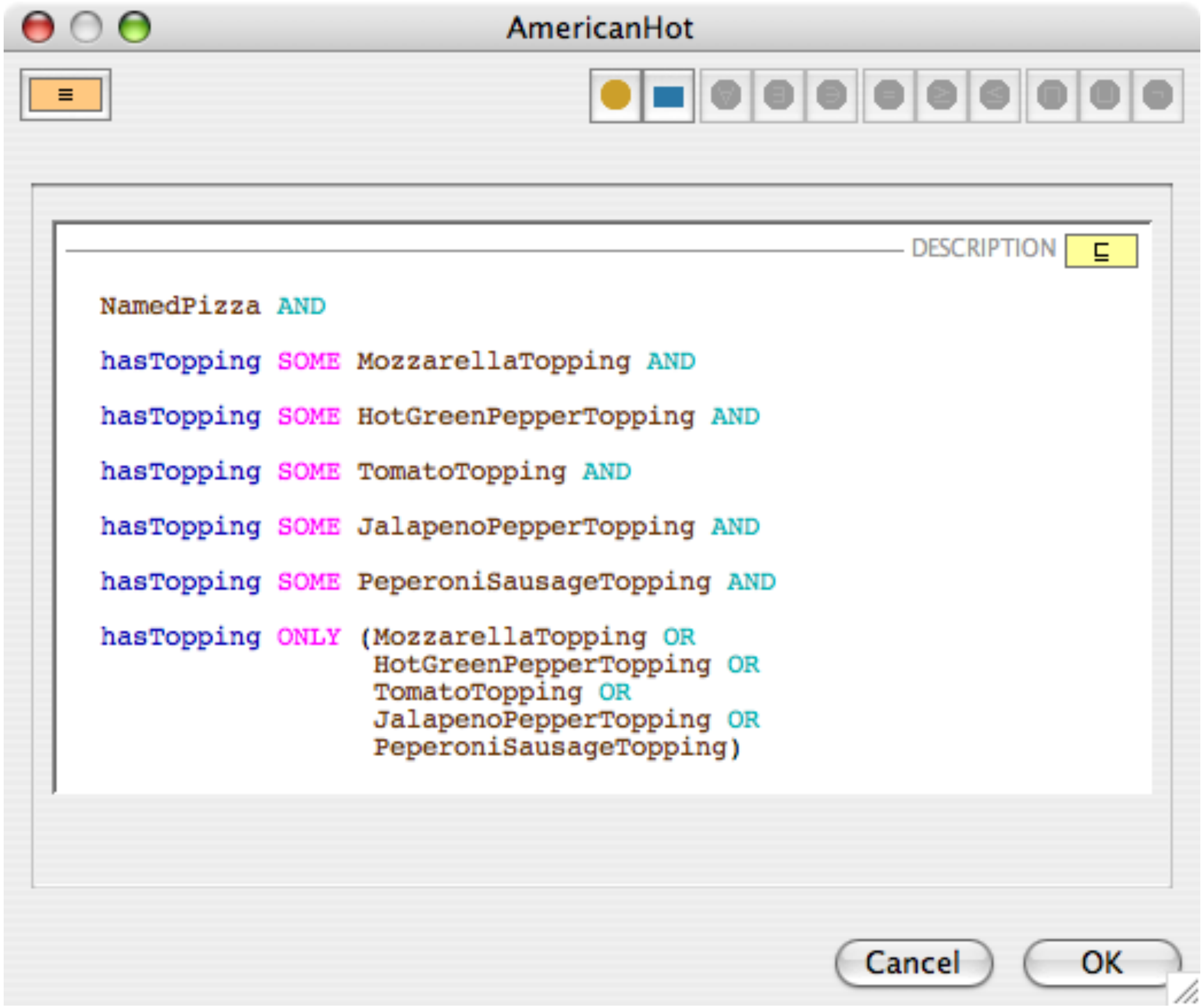
The World	Software Systems
Problem Domain	*System Domain*
Problem Analysis	*System Design*
No FM in USA	Satellite Radio
Digital radio quality required everywhere	Software-controlled embedded system

- A **glossary** is a set of explained terms
- A **classification** is a grouping of the concepts of a domain into classes
- A **taxonomy** superimposes a hierarchical or acyclic is-a relationship
  - Analyse similarity (commonality-variability analysis)



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)


AmericanHot
DESCRIPTION
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)
Cancel
OK

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
- However, often, the word specification and model are used interchangeably (which is not precise)

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?

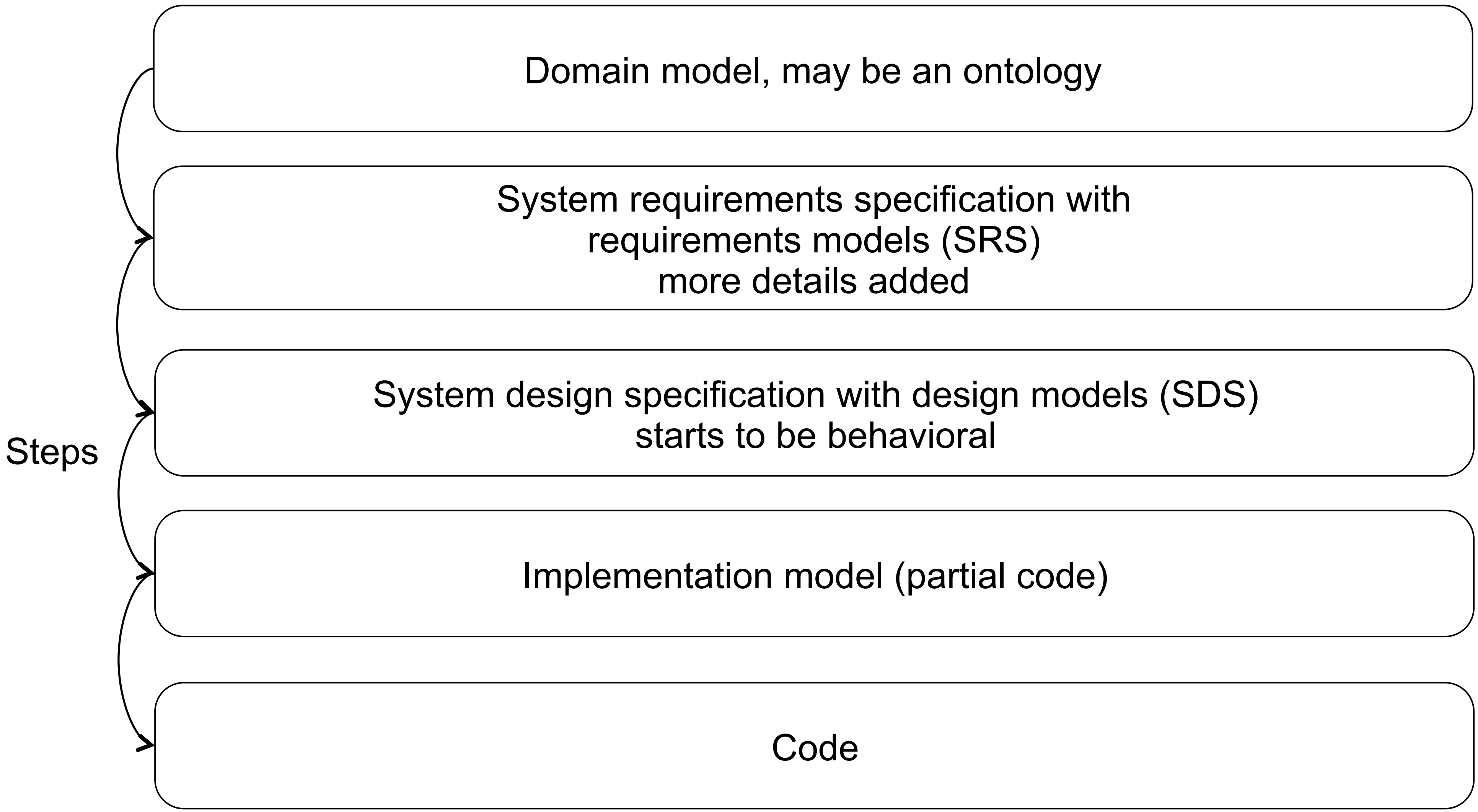
- System models (specifications)
  - From the analysis models, we derive the system models.
- Requirements specification (SRS):
  - the specification what the system should deliver.
  - Functional requirement model: system functions
  - Non-functional requirement model: system qualities
- Design models:
  - 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

- From declarative to behavioral models



- 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

How to come to the next model?

2.2.3 THIRD STEP: CONSTRUCTION

- 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 elaboration steps:
  - **Elaboration:** Elaborate more details – enrich with more semantics
  - **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 development

Questions for the Methods of Development

- Elaboration: Elaborate more details
  - Which Elaboration steps exist?
  - How do I know in which direction to elaborate?
- Pointwise Refinements (concretizations): detailing an abstract concept
  - With and without correctness proofs that the semantics of the abstract concept is provided by the refinement
- Rotations: Apply a semantics-preserving change
  - **Rotate:** Symmetry operations (semantics-preserving operations)
  - **Restructure** (**refactor)** (more structure, but keep requirements and delivery, i.e., semantics)
    - Which restructuring? (when is a specification too complex?)
  - **Transform Domains** (change representation, but keep semantics)
    - Which representation change? (which representations are appropriate for which purpose?)

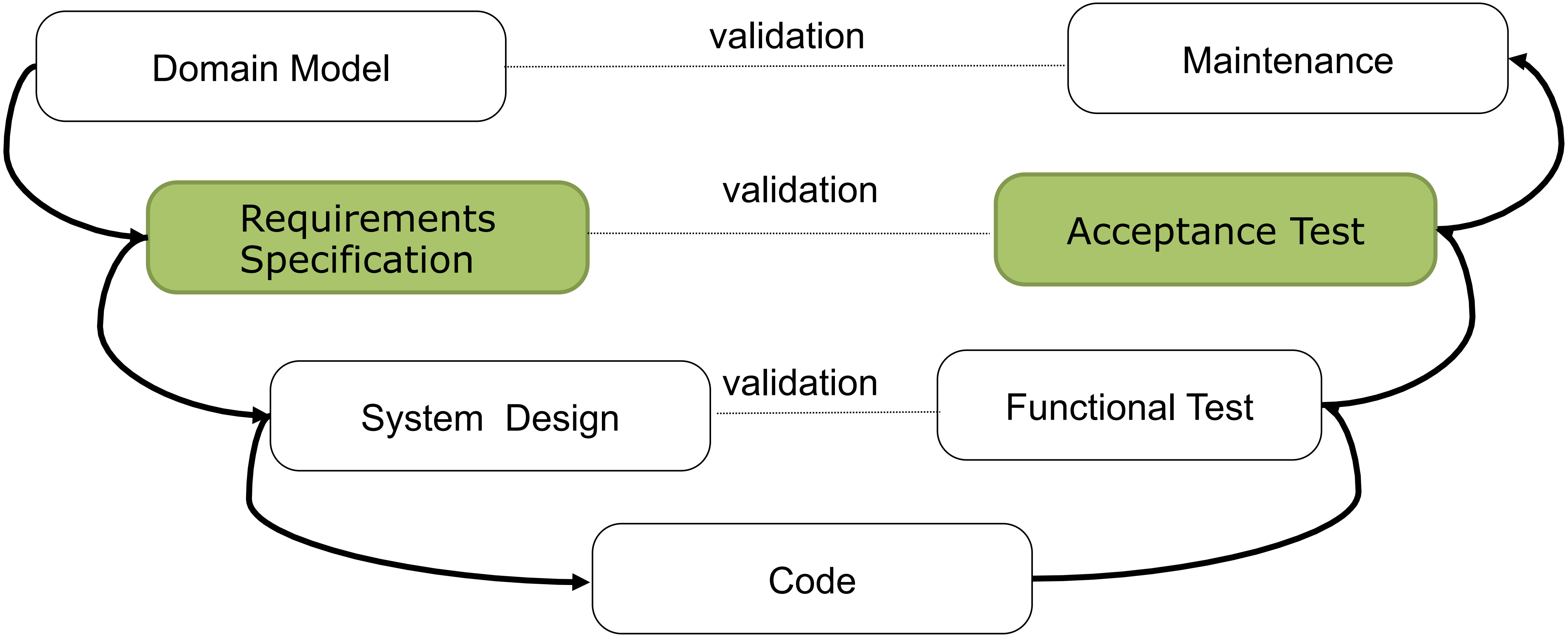
Reuse of Models and Code in Construction

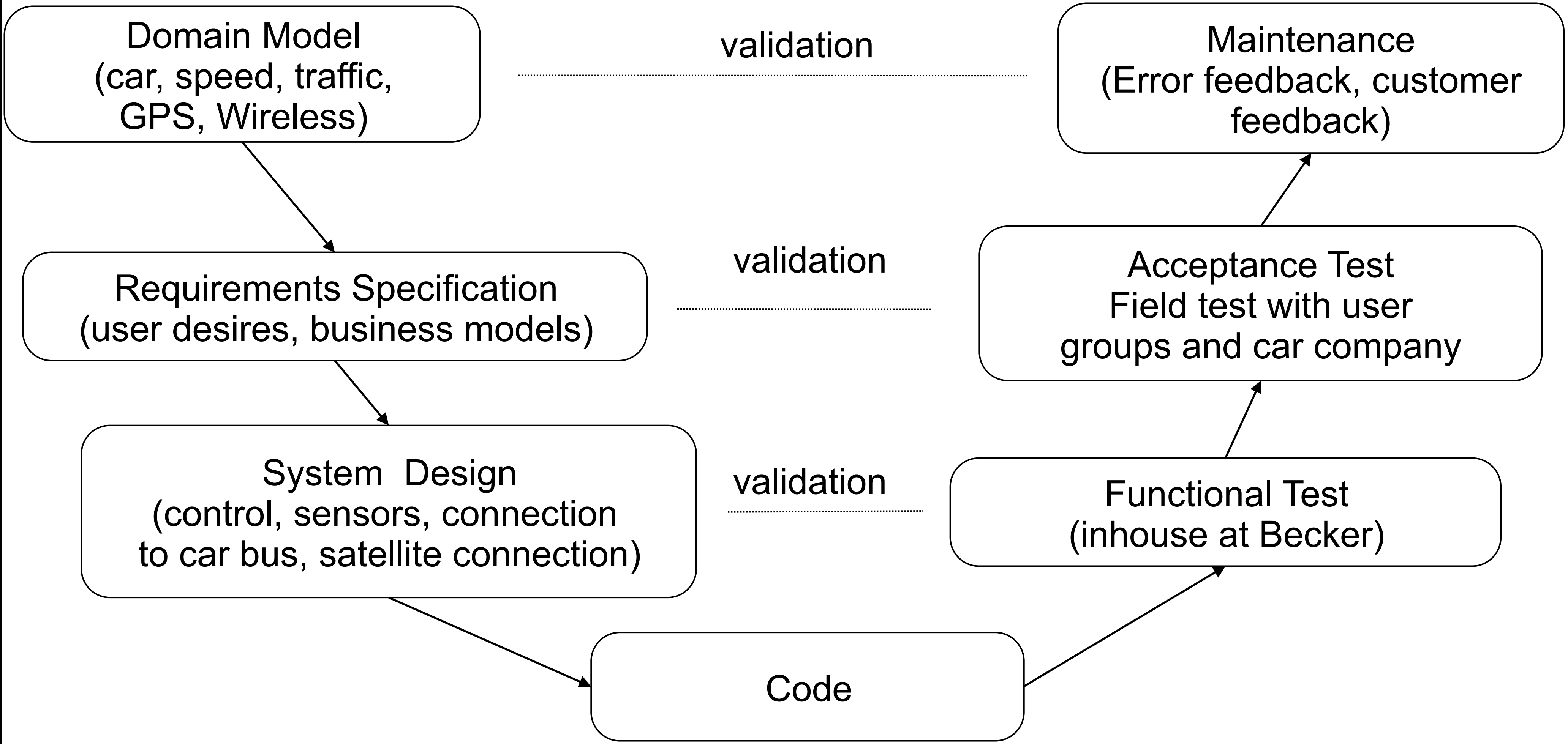
- Engineers try to reuse well-established solutions
  - Components (CBSE)
  - Design patterns
  - Models (model-driven architecture)
  - Best practives
- To simplify system construction
  - To save costs
  - To reduce testing effort

2.2.4. 4TH STEP: VALIDATION

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




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

2.2.5 5TH STEP: IMPROVEMENT

5th Step: 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

Some aspects in section "Earning Money with Software".

2.2.6 6TH STEP: 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

??

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
- 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)”.