ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Network Protocol Design and Evaluation

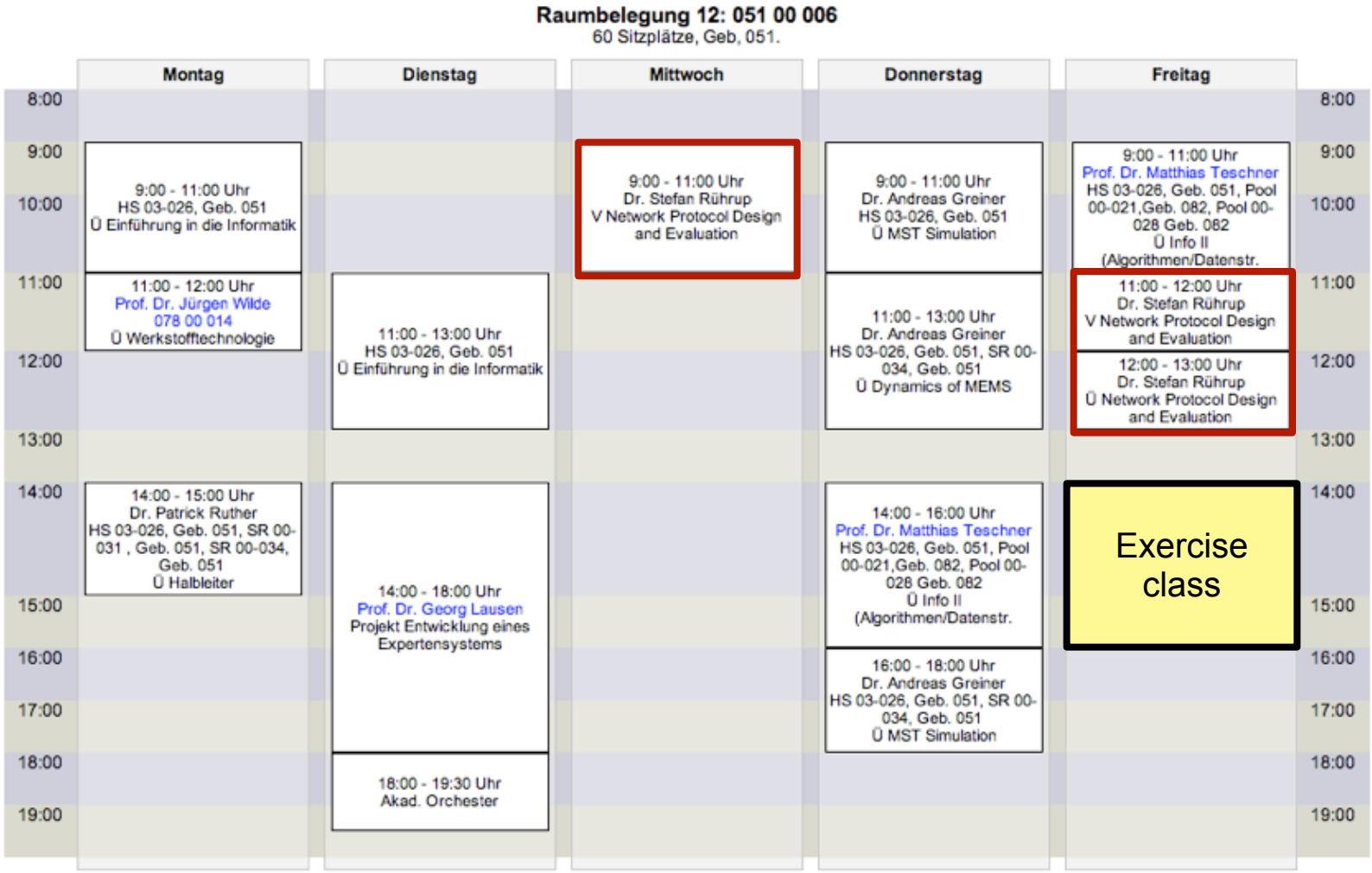
03 - The Design Process

Stefan Rührup

University of Freiburg
Computer Networks and Telematics

Summer 2009

Lecture Times



In the last lecture / Today

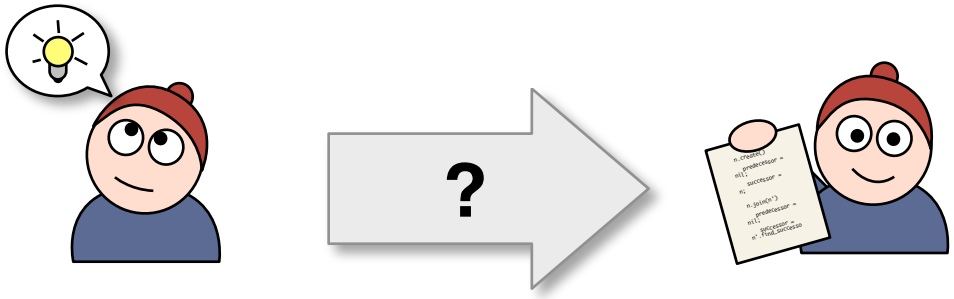
- **In the last lecture:**
  - Design Aspects and Guidelines
  - Internet Design Principles

- **Today:**
  - Development Process

How to develop protocols?

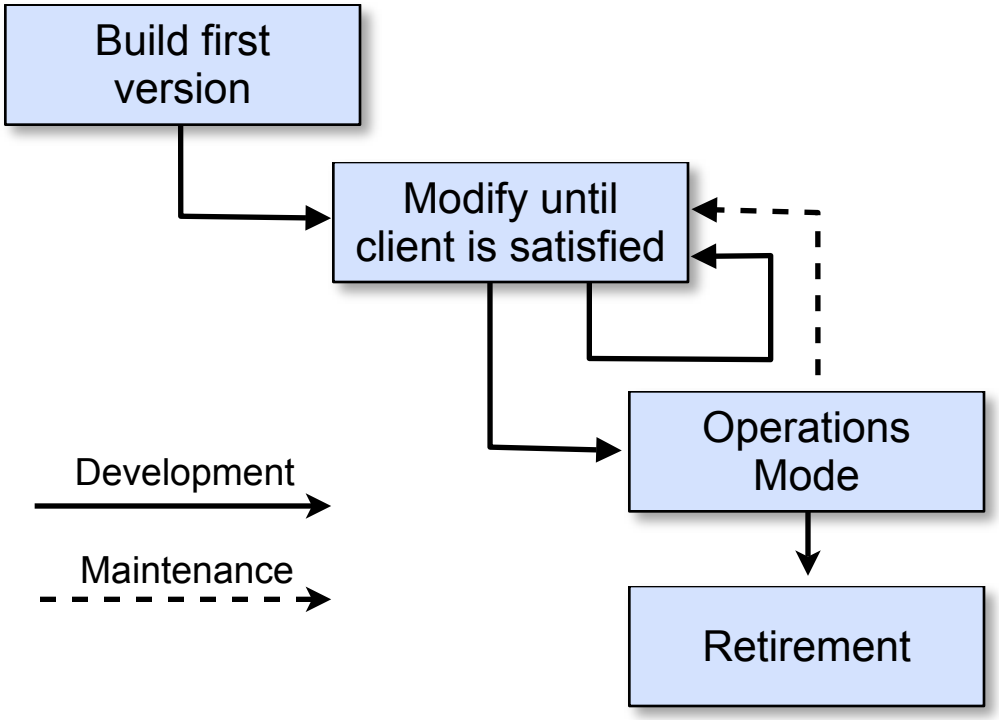
(How to develop software?)

- From the first idea ... to the final solution



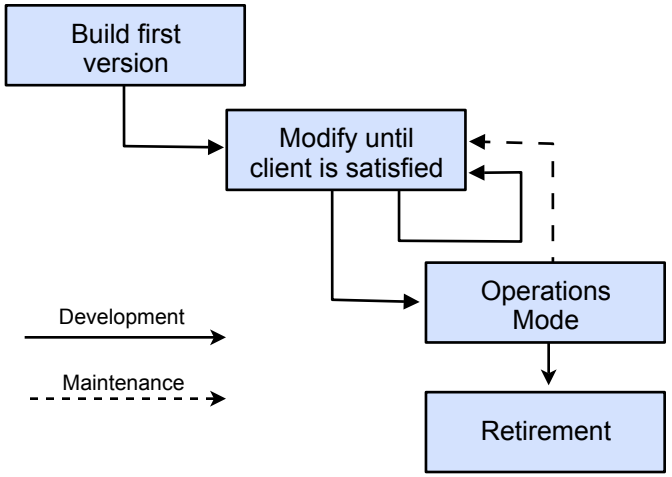
- A development **process** that can be structured
- Examples ...

Build and Fix



Build and Fix

- **Simple process model**
- used for small projects
- **Problems**
  - no specification phase
  - begin coding, think about requirements, design etc. later
  - higher effort for fixing errors in later phases



Structuring the Development Process

- **Process and lifecycle models**
  - structure the software development process into stages
  - Well-defined transitions

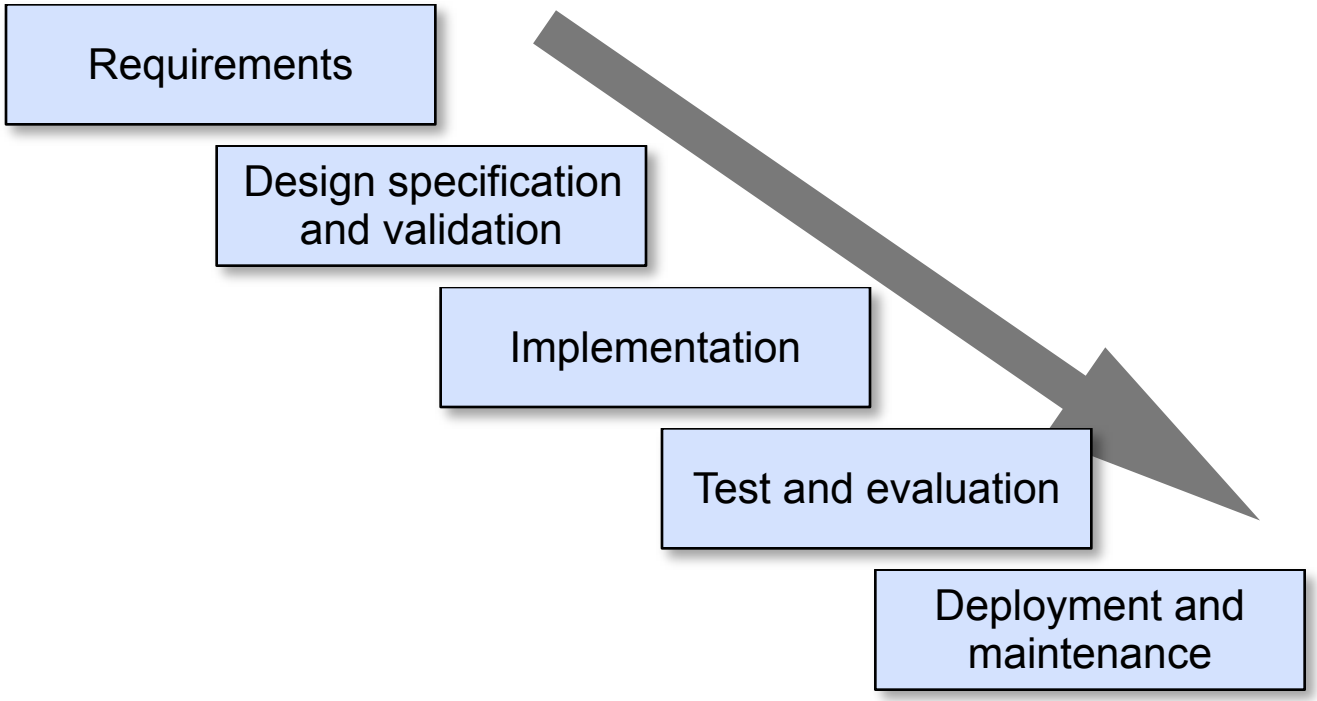
- **Examples:**
  - Build-and-Fix Model
  - Waterfall Model
  - Boehm's Spiral Model
  - etc.

Stages of the Development Process

- **Basic activities** (which appear in many process models)
  - Requirements analysis
  - Design specification
  - Validation
  - Implementation
  - Test and evaluation
  - Deployment
  - Maintenance

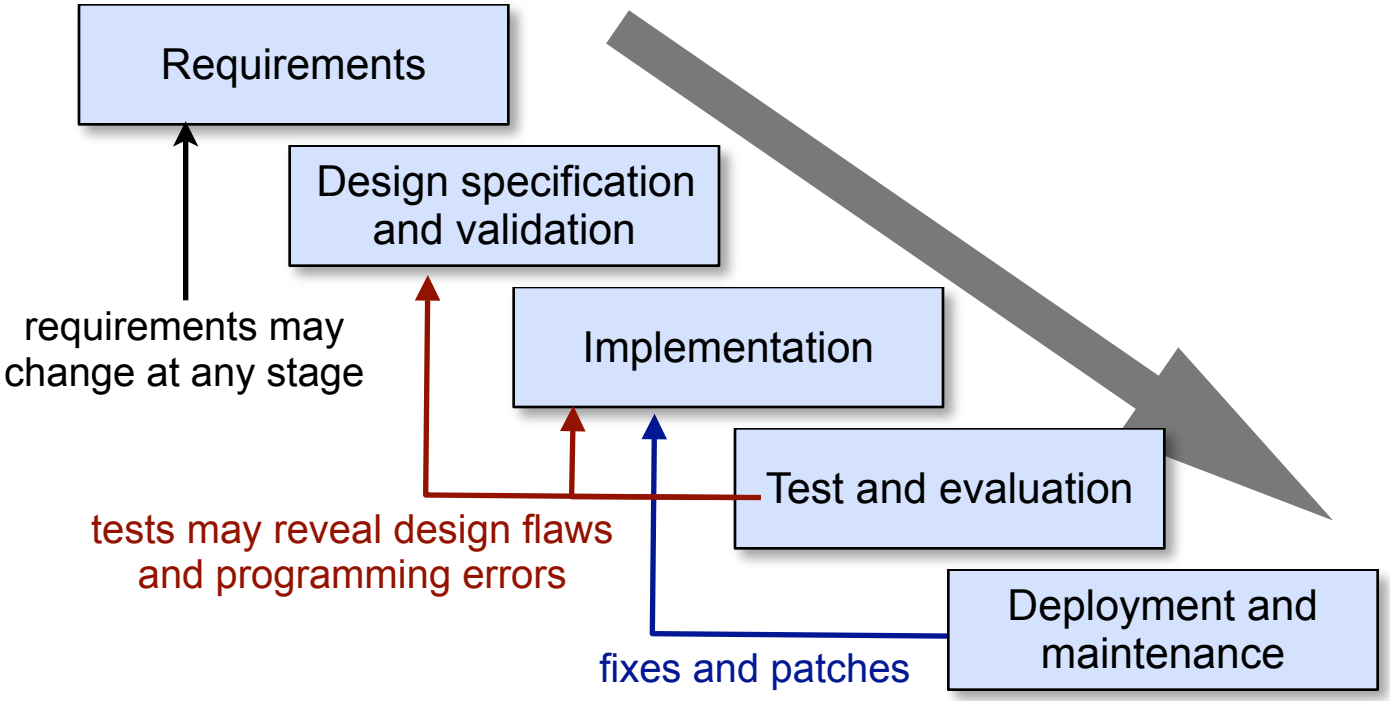
- Different opinions on *if* and *when* to use the stages

Waterfall Model

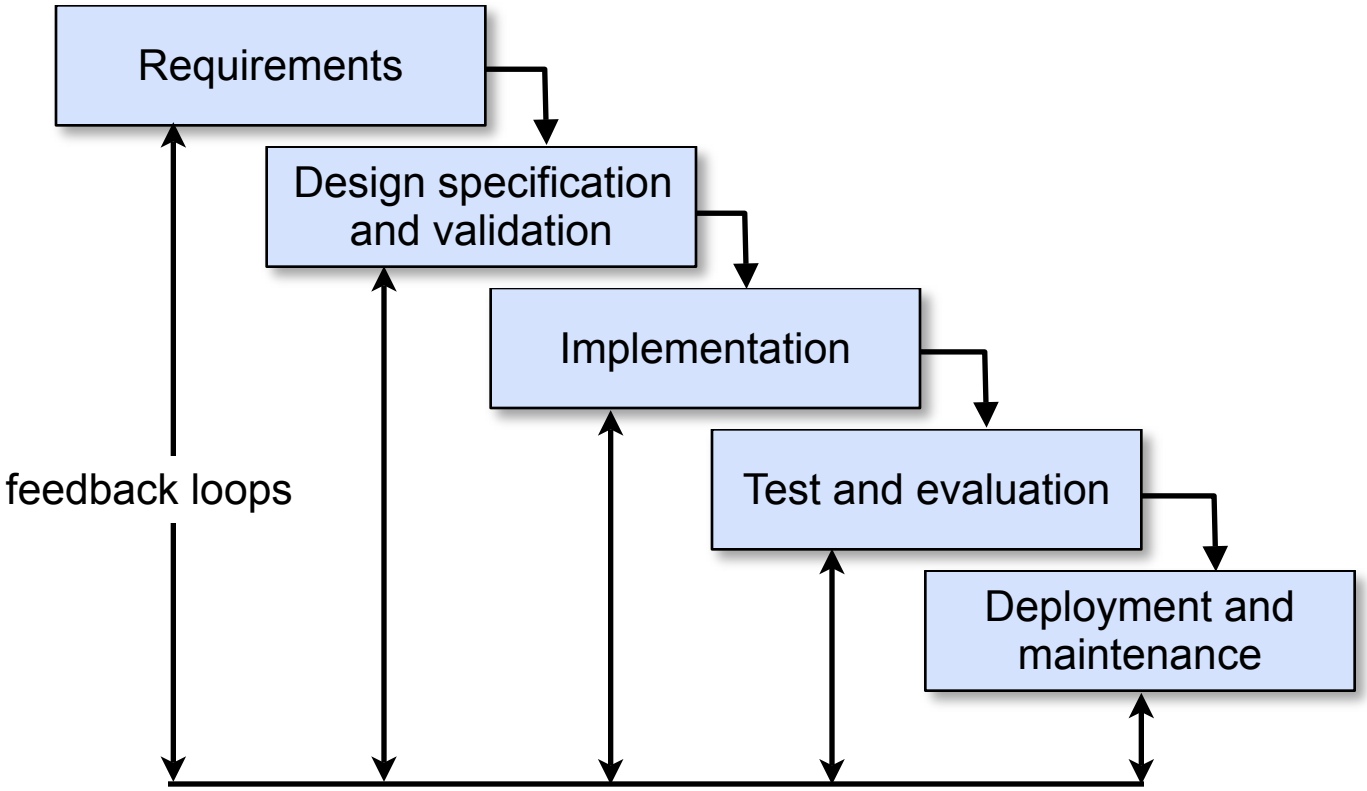


Waterfall Model

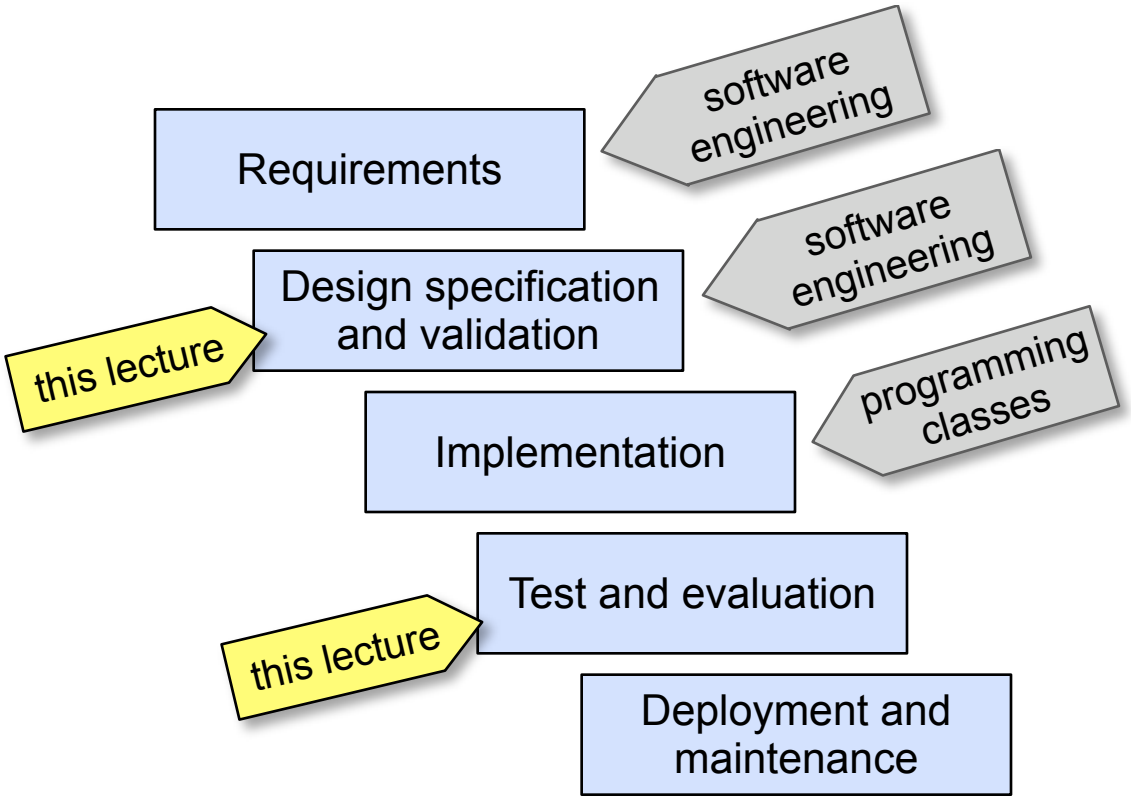
Problems of the pure waterfall model:



Modified Waterfall Model

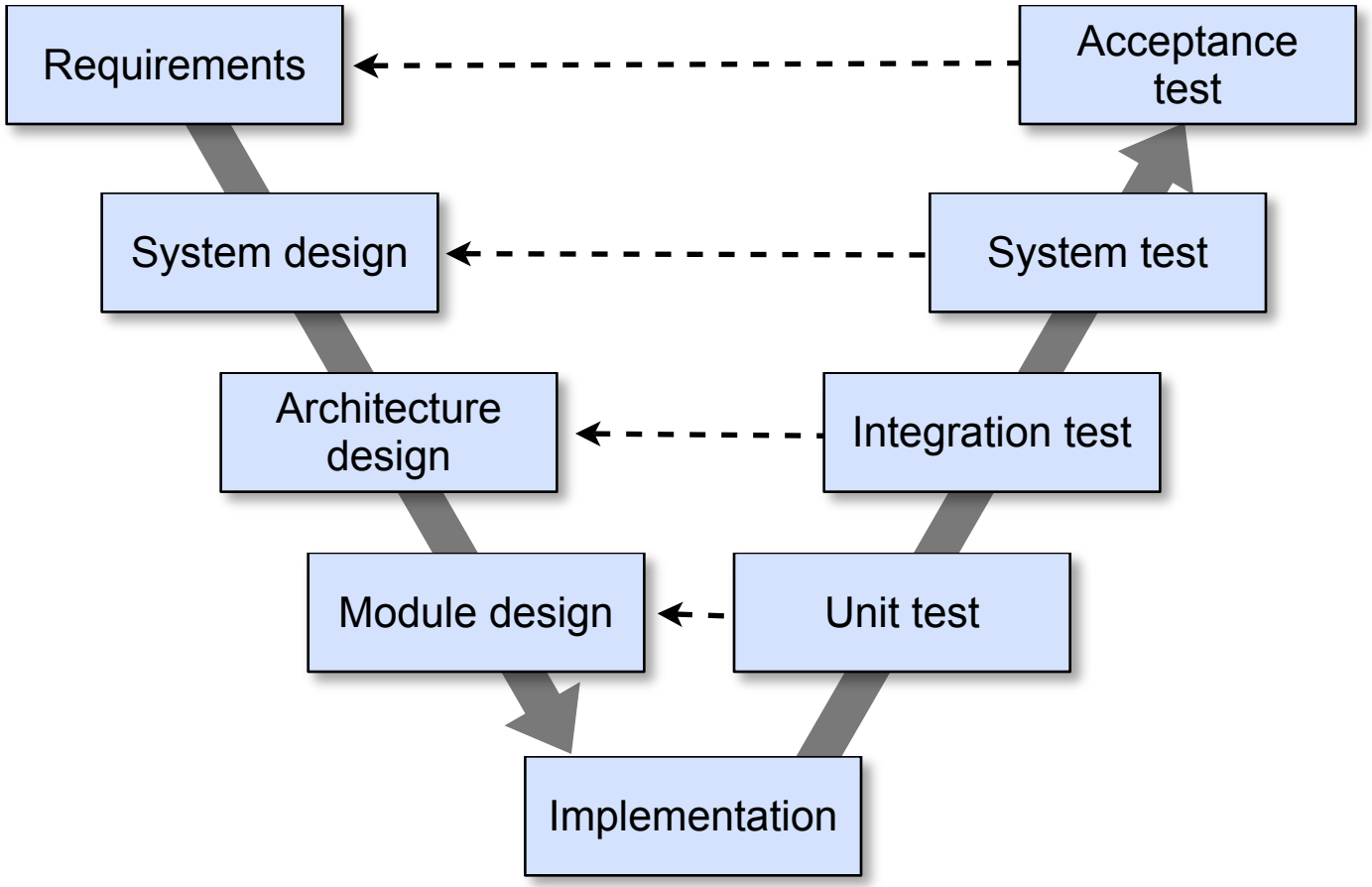


Btw... The focus of this lecture

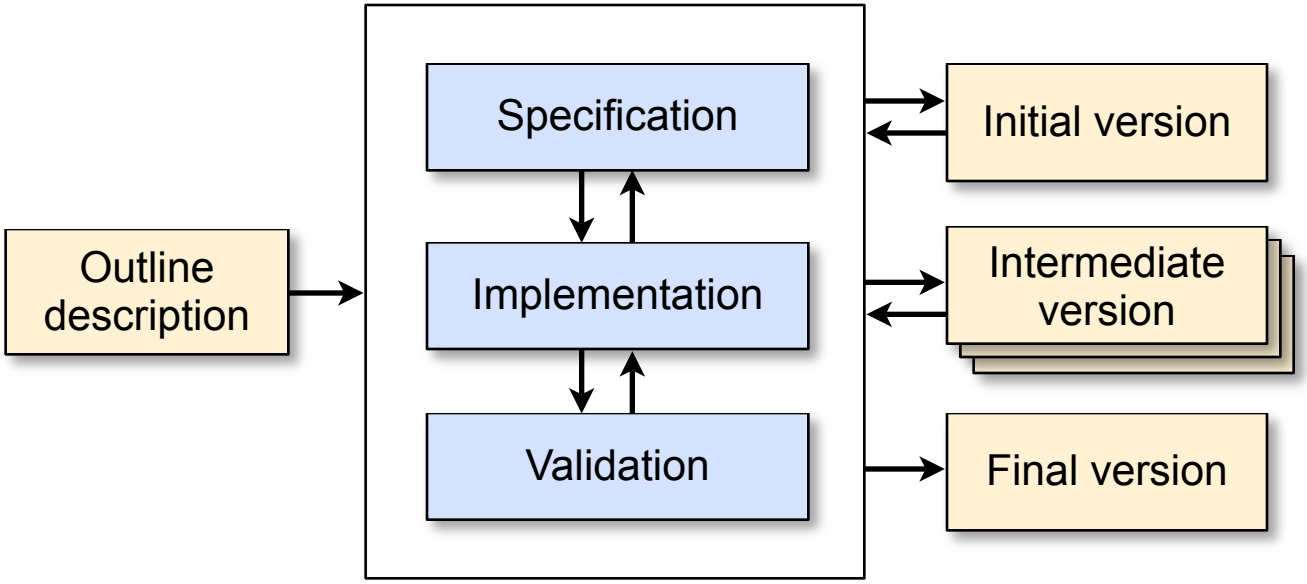


The V-Model

(for software development)



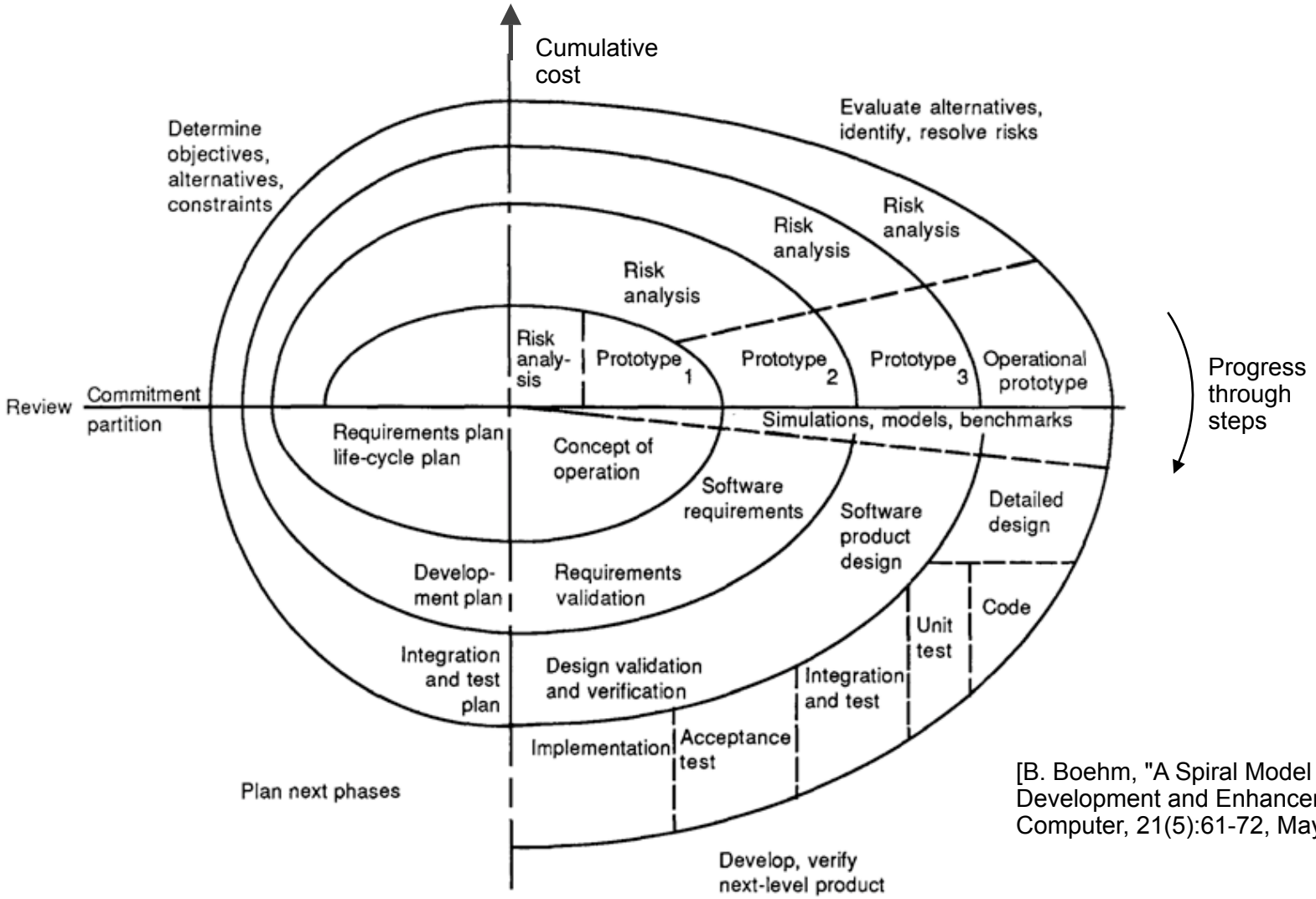
Evolutionary development



- **Problems**: Process is not visible, continuing changes
- Can be used in the prototyping phase of a larger process

[I. Somerville: Software Engineering, 5/e, 1995]

Boehm's Spiral Model



[B. Boehm, "A Spiral Model of Software Development and Enhancement", IEEE Computer, 21(5):61-72, May 1988]

Agile methods

- Adaptive process
- Iterative development cycles
- Emphasis on working functional units
- Short time frames instead of long-term planning
- Communication instead of detailed documentation

- “opposite” of the waterfall model

- Flexibility vs. difficulty to make changes

Process Models

- **Waterfall model**
  - often-cited with known problems
  - well-defined phases, requires a disciplined approach
- **V-Model**
  - extends the waterfall model, considers modular design
- **Boehm's Spiral Model**
  - considers an **iterative development process**
  - suitable for large and complex projects
- **Agile methods**
  - iterative process; flexible, but changes are difficult

Process Models

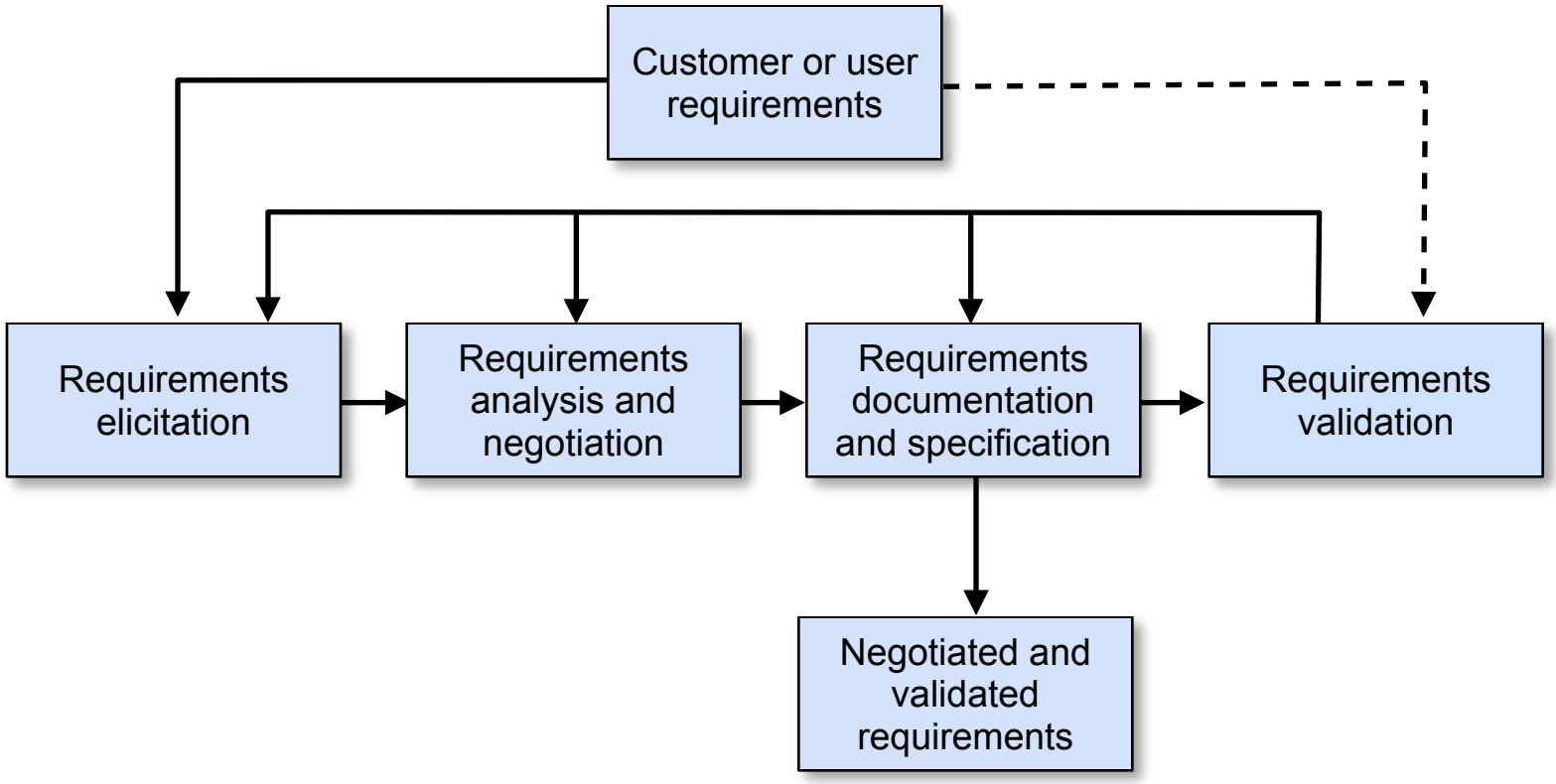
- **The bottom line:**
  - Choice of a process models depends on the type and complexity of the project
  - Network protocols often need revisions and extensions due to changing requirements or problems
  - **Protocol design is an iterative process**

Requirements

- **Requirements**
  - describe desired behaviour of a protocol
  - independent of the later design and implementation (describe *what* the protocol does, not *how*)

- **Requirements Analysis/Engineering has its own process**

Requirements engineering



[S. Leue, Design of Reactive Systems, Lecture Notes, 2001]

Types of Requirements

- **Functional requirements** or **use cases**
  - System behaviour and data format
  - Here: procedure rules and message format
- **Non-functional** or **quality requirements**
  - e.g. reliability, performance
- **Design constraints**
  - environment, interfaces

[S. Pfleeger, J. Atlee, “Software Engineering - Theory and Practice”, 3/e, Prentice Hall]

Requirements documents

- **Requirements definition**
  - abstract description of the system's services and functions (external behaviour)
  - mostly written in natural language
  - for developers and users

- **Requirements specification**
  - precise description of the system's functions
  - may be written in a formal language

[I. Somerville: Software Engineering, 5/e, 1995]

Requirement documents

Contents of a **Software Requirements Specification (SRS)**

(according to IEEE Standard 830-1998)

1. Introduction (Purpose, Scope, Acronyms, References, Outline)
2. General Description (Context, Functions, Constraints, Assumptions)
3. Specific Requirements
    - 3.1. External Interface Requirements
    - 3.2. Functional Requirements
    - 3.3. Performance Requirements
    - 3.4. Design Constraints
    - 3.5. Quality Requirements
    - 3.6. Other Requirements
4. Appendices

Characteristics of Requirements

- **Requirements should be ...**
  - correct (developer's understanding = stakeholder's needs)
  - consistent (no conflicting goals)
  - unambiguous (formal specification)
  - complete (no under-specification)
  - relevant, design-independent (no over-specification)
  - feasible (possibility to meet all requirements)
  - verifiable/testable (quantifiable statements)
  - traceable (references to the specification)

[S. Pfleeger, J. Atlee, "Software Engineering - Theory and Practice", 3/e, Prentice Hall]

Requirements validation

- **In general:** checking that the specification matches the user's requirements

- **Ambiguity - Natural language or formal notation?**
  - easily understandable vs. precise and unambiguous

- **Making requirements consistent**
  - Resolving conflicts, e.g. prioritization into essential, desirable and optional goals (quality requirements)

[S. Pfleeger, J. Atlee, "Software Engineering - Theory and Practice", 3/e, Prentice Hall]

Requirements validation

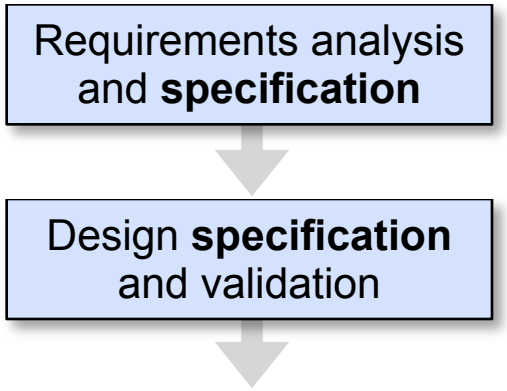
- **Testability: Requirements should be quantified** (holds also for the specification)
  - *the server is expected to respond immediately*
    better: *the server has to respond within 5ms.*
  - *the packet is dropped after some unsuccessful retries*
    better: *the packet is dropped after 3 unsuccessful retries. Each retry is triggered after a timeout of 2 x RTT.*

Validation and Verification

- **Lots of techniques for requirements validation**
  - interviews, reviews
  - prototypes, simulations

- **... and verification**
  - cross-referencing
  - model checking
  - mathematical proofs

From Requirements to the Design



- **Requirements analysis leads to a specification**
  - Specification states which requirements should be fulfilled
  - Modeling languages and tools are used in both phases

Design documents

Contents of a **Software Design Description (SDD)**

(according to IEEE Standard 1016-1998)

1. Introduction (Design Overview, Requirements Traceability Matrix)
2. Architectural Design
    - Chosen System Architecture
    - Discussion of Alternative Designs
    - System Interface Description
3. Detailed Description of Components
4. User Interface Design
    - Description of the User Interface
    - Objects and Actions
5. Additional Material

Modeling and specification

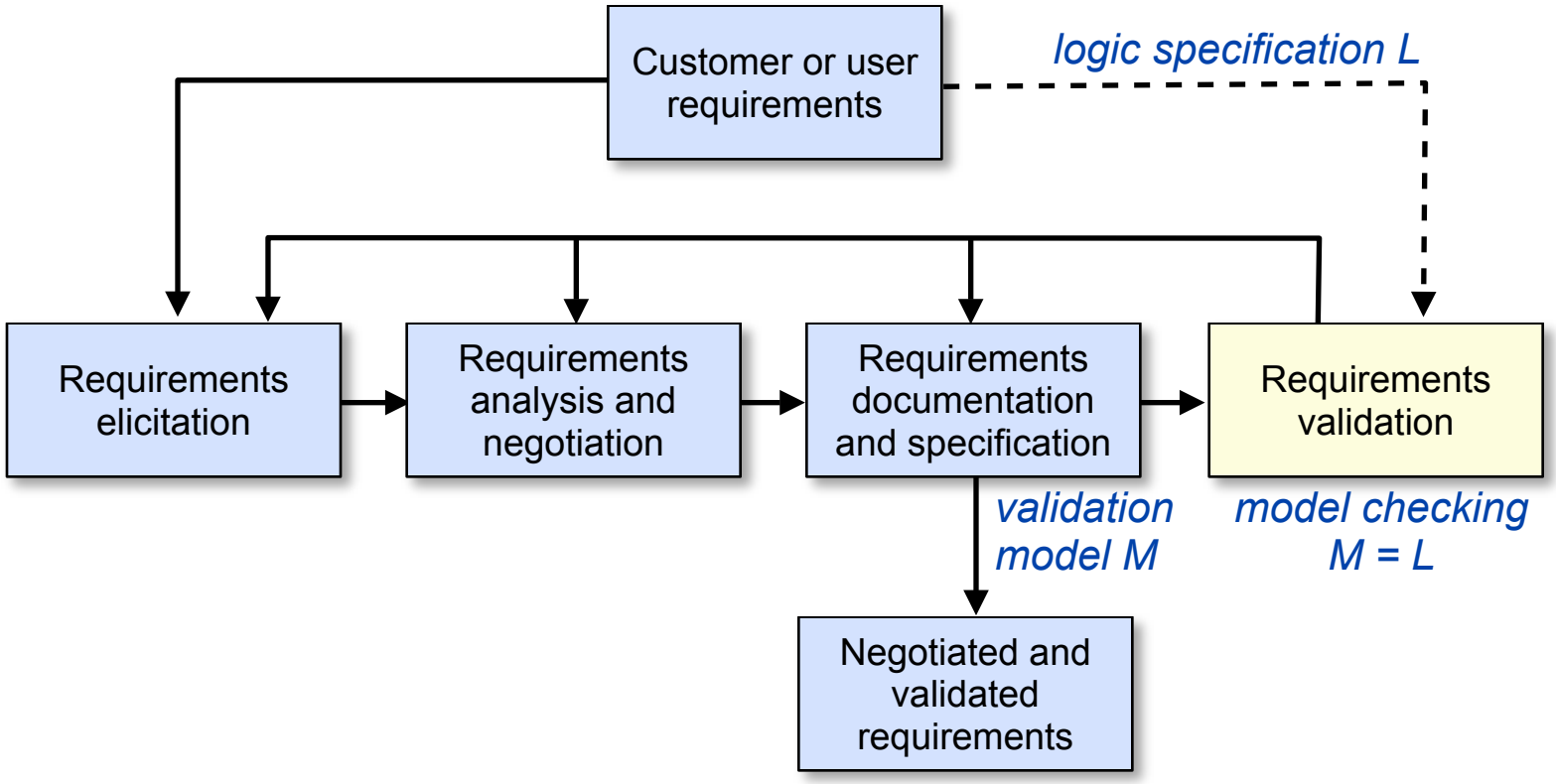
- **Formal notation and languages:**
  - State machines
  - Unified Modeling Language (UML), e.g.
    - UML state charts
    - UML protocol state machines
    - UML sequence charts
    - UML use case diagrams
  - Specification and Description Language (SDL) [ITU-T Z.100], e.g. process diagrams
  - Message Sequence Charts [ITU-T Z.120]

Validation and Model Checking

- **Validation models** for protocols:
  - Description of procedure rules (partial description)
  - Finite state model

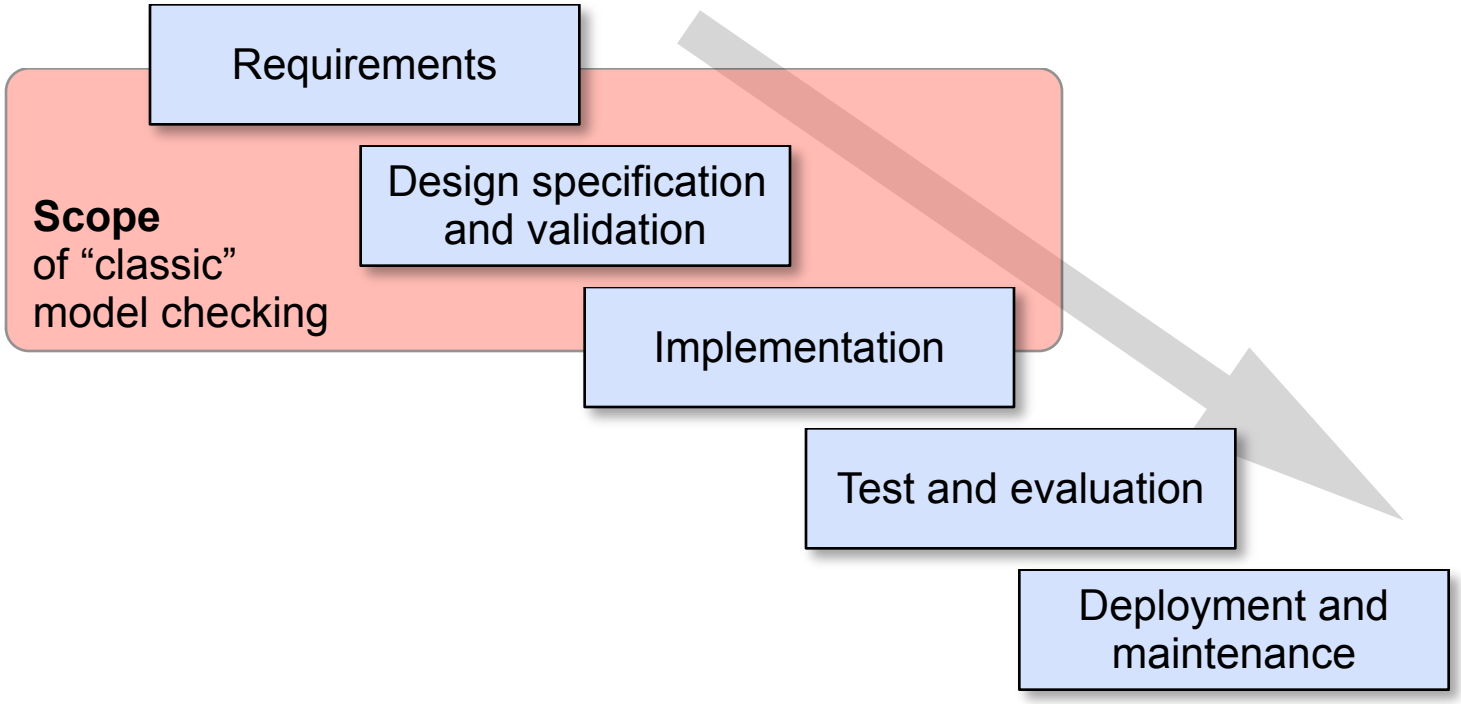
- **Model checking**
  - Automated verification technique
  - Does a protocol satisfy some predefined logical properties?

Model checking

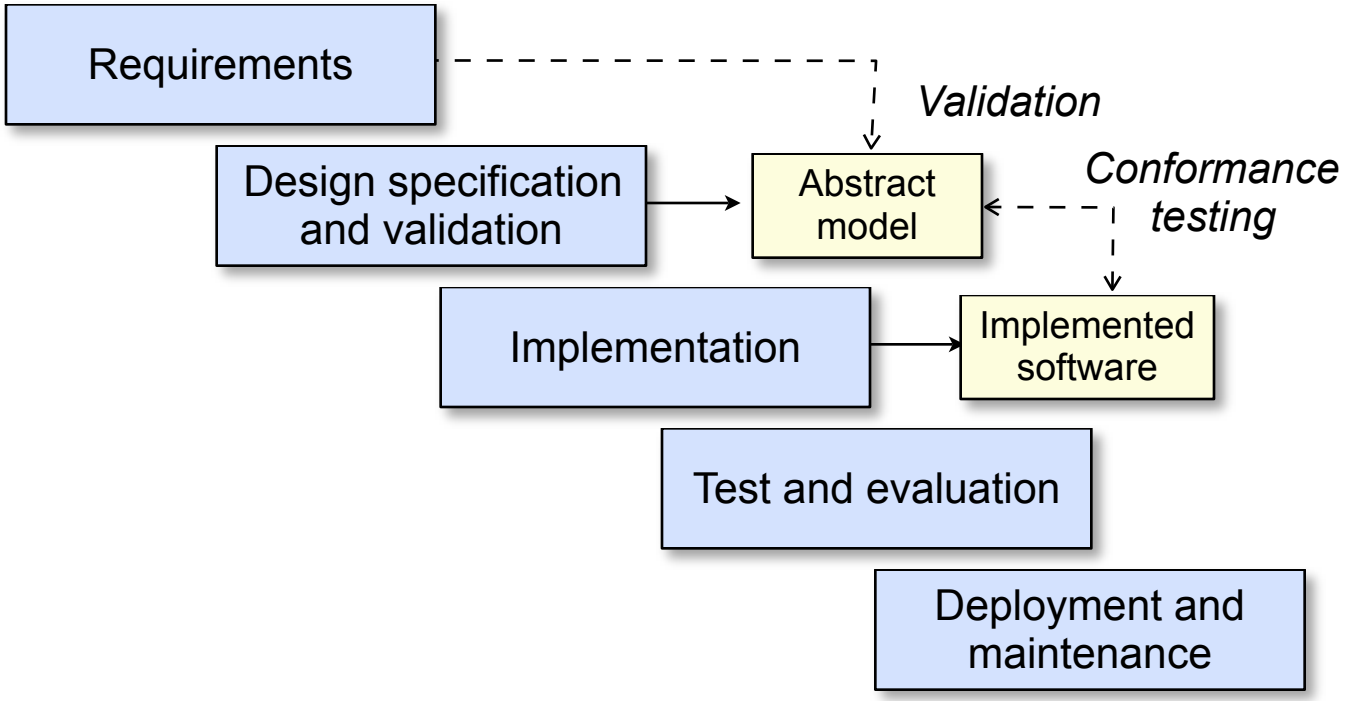


[S. Leue, Design of Reactive Systems, Lecture Notes, 2001]

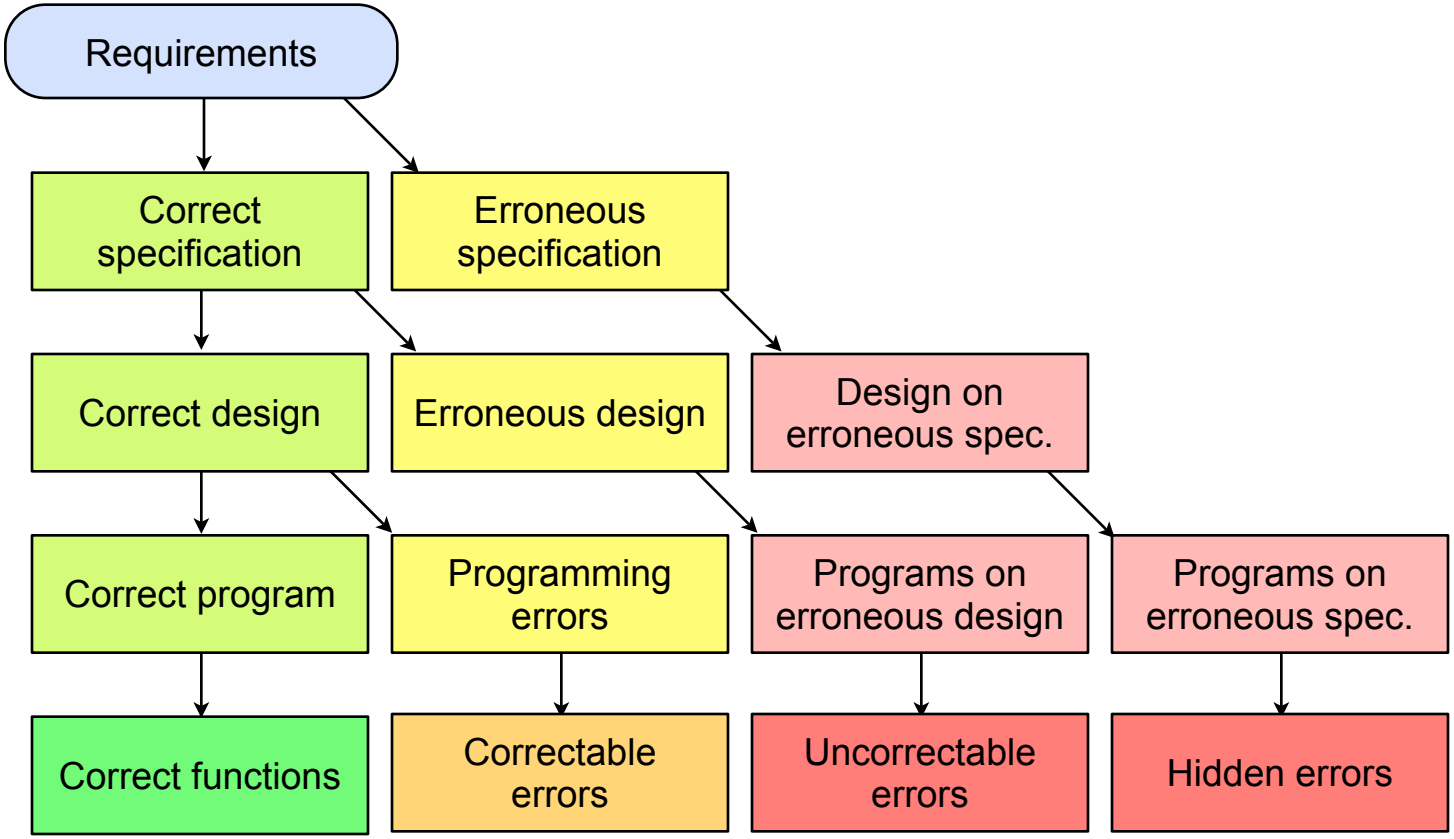
Model Checking



Model Checking

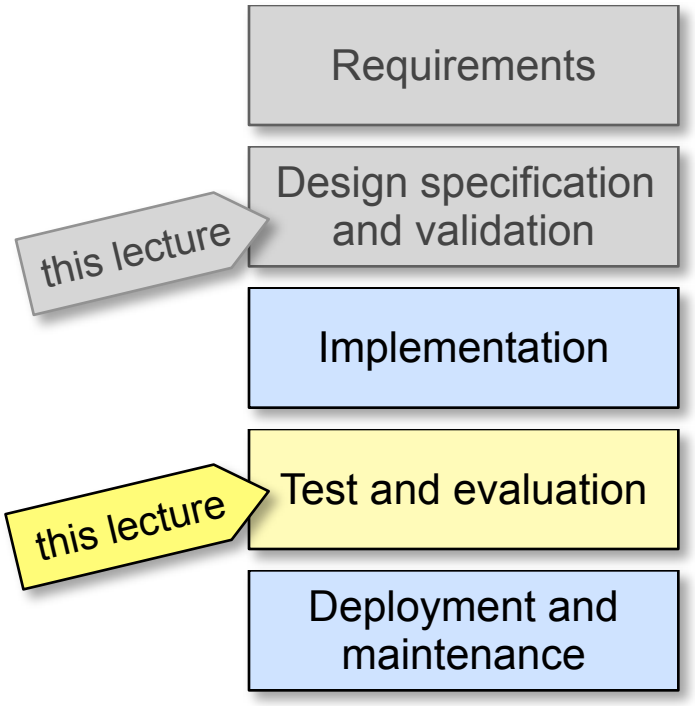


Importance of early stages



[Y. Mizuno, “Software Quality Improvement”, IEEE Computer, 1983]

The later stages



- **Testing**
  - Unit tests
  - Integration tests
  - Conformance testing
- **Evaluation methods**
  - (Analysis)
  - Simulation
  - Experiments

Lessons learned

- **Don’t skip the early stages** of the development process ... even in small projects

- Errors in requirements and specification phase are hard to fix later