TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

Software and Systems Engineering for Embedded Systems

VO Embedded Systems Engineering

Armin Wasicek

WS 2010/11

Outline

- What is Software Engineering?
- Processes Models
- Standards and Reference Architectures
- Software Engineering Disciplines
  - Requirements
  - Design
  - Testing

Software Crises

"To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem." [Edsger Dijkstra]

Manifestation

- Projects running over-budget and over-time
- Software was of low quality and code difficult to maintain
- Unmanageable projects

The term *software engineering* was popularized after 1968,during the 1968 NATO Software Engineering Conference

Software Engineering is …

(1) the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, that is, the application of engineering to software," and

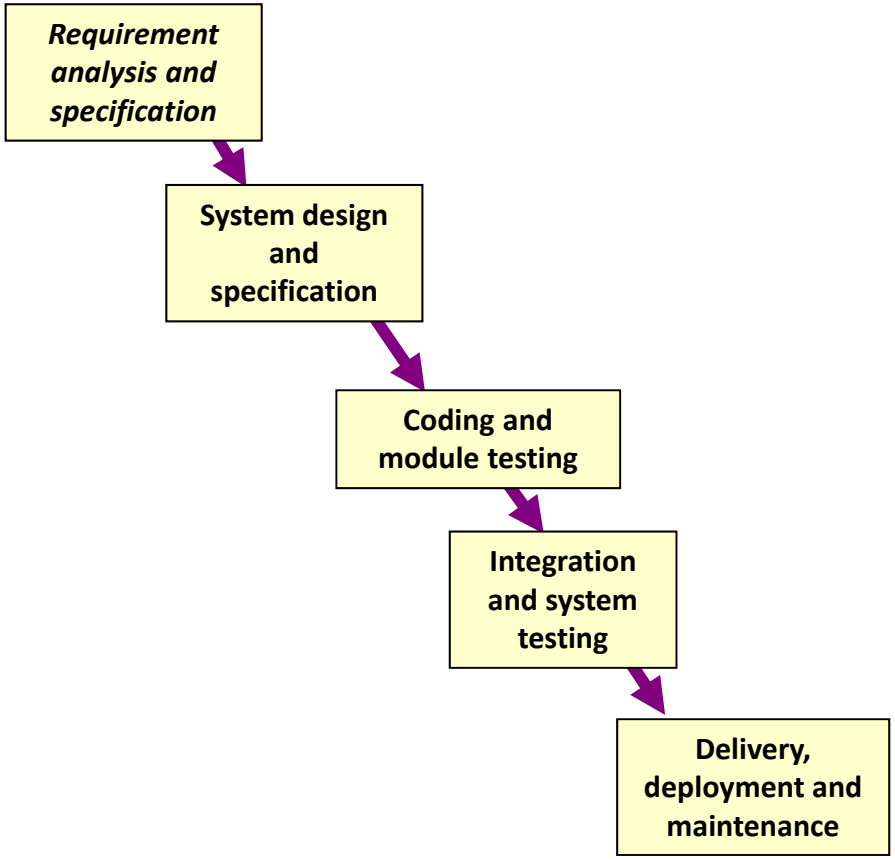
(2) the study of approaches as in (1).

from the IEEE Standard 610.12

Incremental Process Models

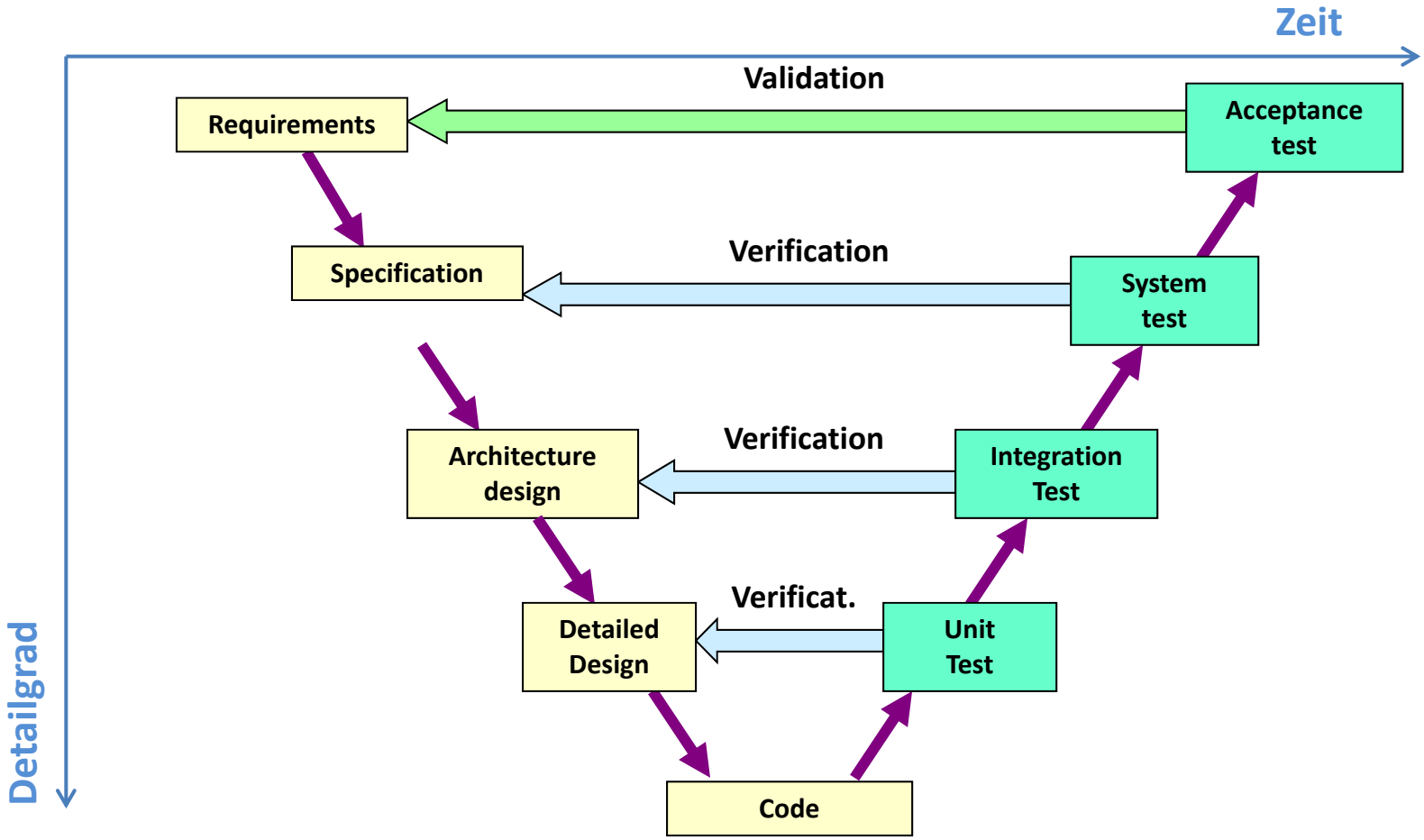
- Waterfall Model
- V-Model
- ROPES Spiral Lifecycle
- Rational Unified Process
- Hardware/Software Codesign

Waterfall Model

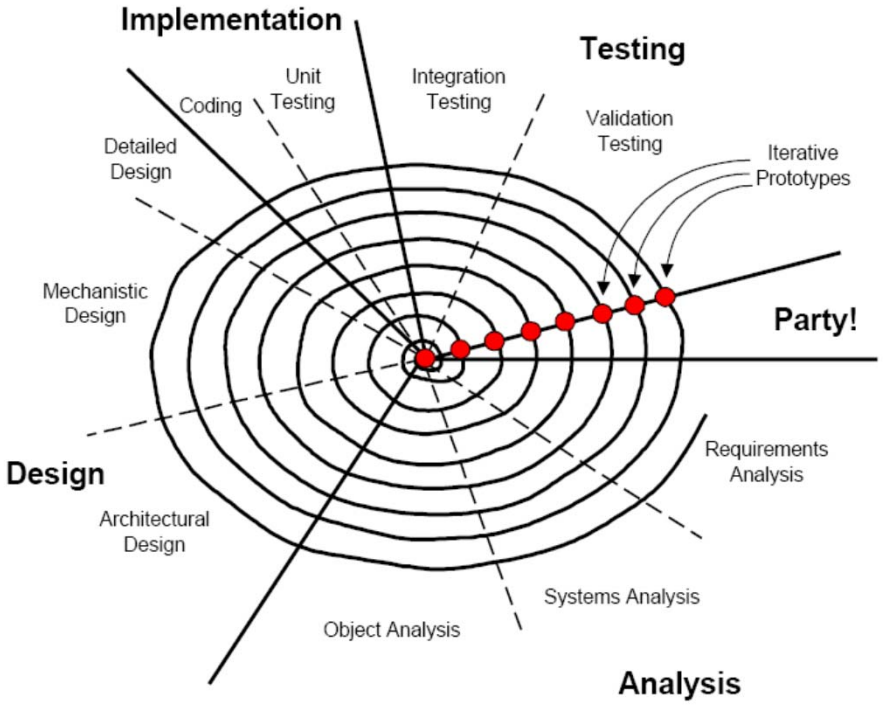


- Dates back to 70ies
- Separate and distinct phases of specification and development.
- Very disciplined approach
- Assumes sharp edges between phases
- Used by US-DoD and NASA
- Many people consider this assumption to be unrealistic

V Model

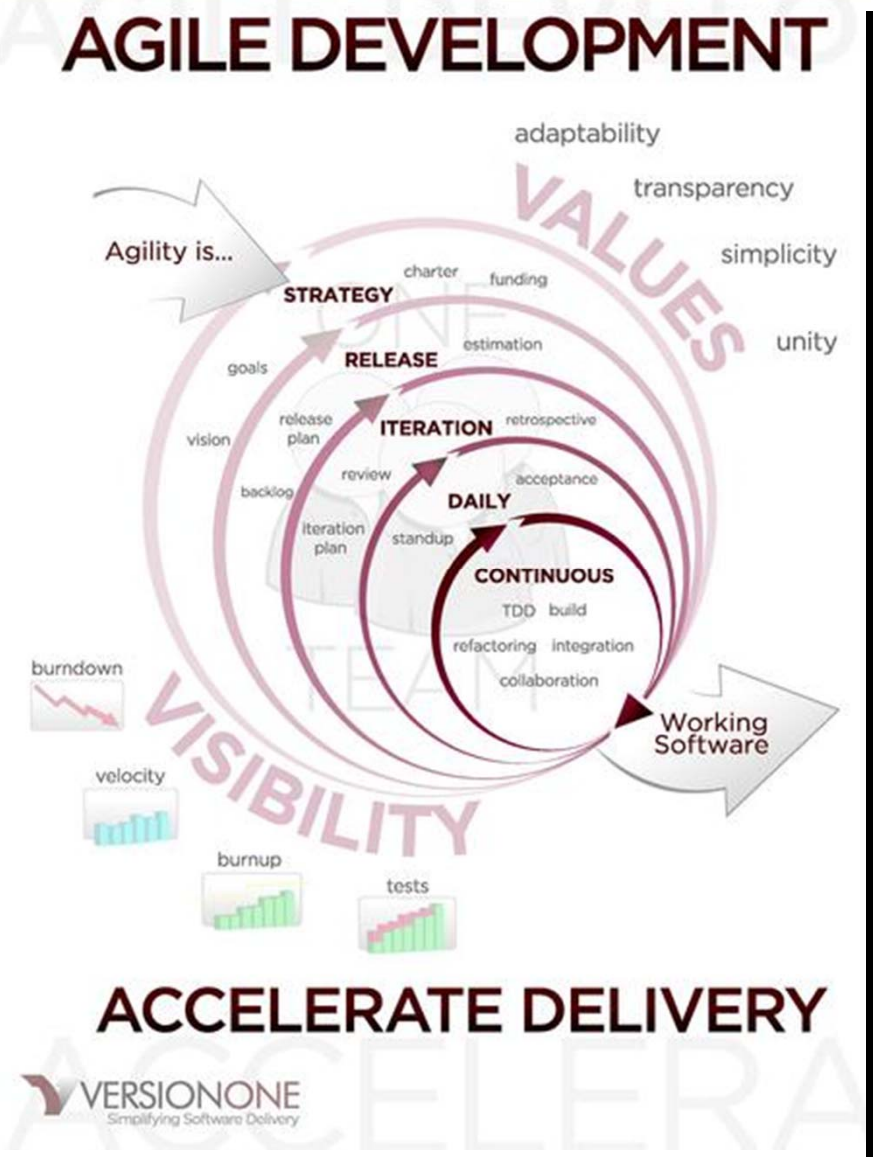

Zeit
Validation
Requirements
Acceptance test
Specification
Verification
System test
Architecture design
Verification
Integration Test
Detailed Design
Verificat.
Unit Test
Code
Detailgrad

Spiral Model (e.g., ROPES)



(Taken from Bruce Powel Douglas)

- The result of every spiral cycle is a prototype
- Each prototype should have a mission statement
- Iterative prototypes are constructed by modifying the previous prototype
- **Incremental development**



Manifesto:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

Identified Failure Causes in SW Projects

- Ad hoc requirements management
- Ambiguous and imprecise communication
- Brittle architecture
- Overwhelming complexity
- Undetected inconsistencies in requirements, designs, and implementations
- Insufficient testing
- Subjective assessment of project status
- Failure to attack risks
- Uncontrolled change propagation
- Insufficient automation

Principles and Best Practices

- Develop software iteratively
- Manage requirements
- Use component based architecture
- Visually model software
- Verify software quality
- Control changes to software

Hardware/Software Co-Design

Traditional Design

- A specific hardware platform is chosen
- Software is designed for that platform
- Hardware and software are optimized independently

HW/SW Co-Design

- The whole system is specified in an platform independent way
- Semi-automatic HW/SW partitioning w.r.t. system functions
- Semi-automatic synthesis and code generation
- HW and SW are optimized together
- Optimized cost/value ratio

Exemplary Standards in the Automotive domain

- OSEK/VDX Standard
- AUTOSAR
- MISRA C

OSEK/VDX - Standard

“Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug”

Founded in 1993 as a joint project in the German automotive industry

- Project partners: BMW, Bosch, DaimlerChrysler, Opel, Siemens, VW and University of Karlsruhe,
- In 1994 the two French companies (PSA and Renault) joined introducing their VDX-approach (Vehicle Distributed eXecutive)

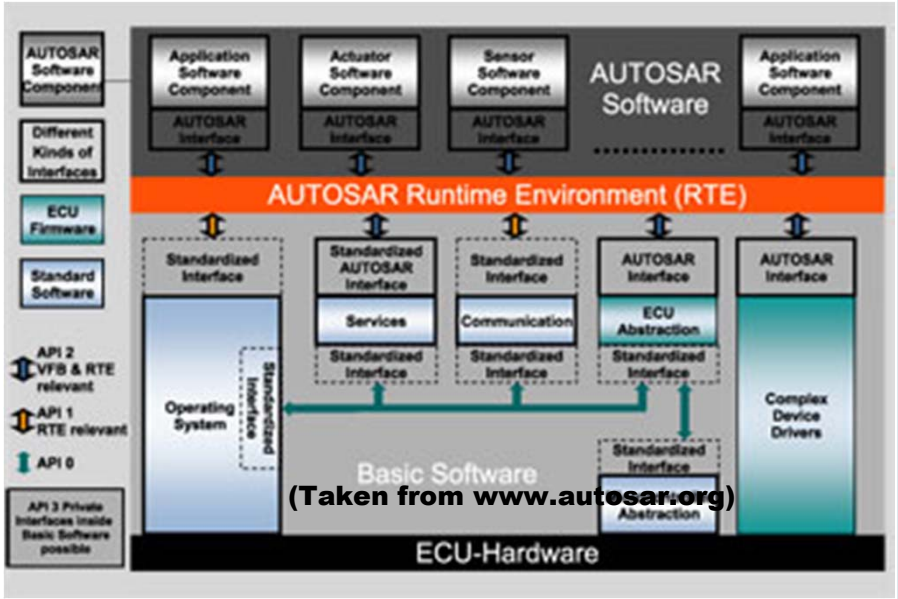
The open architecture comprises three areas:

- OSEK-COM (data exchange within and between control units)
- OSEK-OS (real-time execution of ECU software and base for the other OSEK/VDX modules)
- OSEK-NM (Network Management, Configuration determination and monitoring)

AUTOSAR - Technical Overview

"AUTomotive Open System Architecture"

- Open standard for automotive E/E architecture



- **Technical goals:**
  - Modularity
  - Scalability
  - Transferability
  - Re-Usability

- **Achieved by standardized interfaces**
  - AUTOSAR software components (encapsulate applications)
  - Virtual Functional Bus – VFB (sum of all communication mechanisms provided on an abstract, i.e. technology independent, level.)
  - Runtime Environment (implements the VFB functionality on a specific ECU)

Coding Guidelines - MISRA C

"Motor Industry Software Reliability Association"

- Guidelines for the use of the C language in critical systems
- Used in automotive, rail, aerospace, military and medical sector
- More than 100 rules for safe programming to avoid runtime errors and misunderstanding among programmers
- E.g.: `if (i = a) {  /* instruction */  }` is forbidden. Could be confused with
  `if(i == a) {/* instruction */ }`
- In the automotive sector these guidelines are generally mandatory

Software Engineering Disciplines

- **Requirements Engineering**
- **Design**
- Construction
- **Testing**
- Maintenance
- Quality Assessment
- Documentation

Requirements Engineering

- Establishes the **services** that are required from a system by the costumer and the **constraints** under which a system has to be developed
- The result of this process is a **requirement specification** document, which can serve as a contract between the costumer and the developer
- Often used as a **binding contract** between distributor and customer

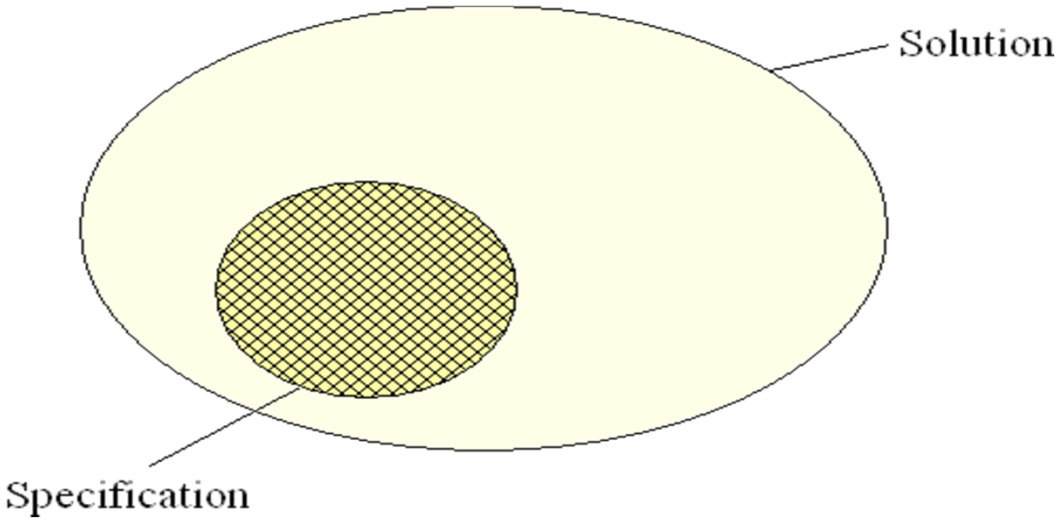
Functional vs. Non-Functional Requirement

- **Functional Requirement**
  - Specifies a specific behavior of a system
  - Based on use cases
- **Non-functional Requirement**
  - Specifies criteria that can be used to judge the operation of a system
  - Constraints on standards or the development process
- ***Are temporal requirements functional or non-functional in embedded systems?***

Good Requirements

Attribute	Description
Concise	The requirement addresses one and only one thing.
Unambiguous	The requirement is concisely stated without recourse to technical jargon, acronyms, or other esoteric verbiage.
Necessary	The absence of the Req. will result in a deficiency that cannot be tolerated.
Consistent	The requirement does not contradict any other requirement.
Complete	The Req. is fully stated in one place with no missing information.
Traceable	The Req. Can be traced throughout the entire development process.
Verifiable	Fulfillment of Req. can be vriefied by inspection, demonstration, test or analysis.

Requirement Specification vs. Solution



- Specify abstract enough to avoid predefined solutions
- Specify concrete enough to be able to reach a contract

Software Engineering Disciplines

- Requirements Engineering
- **Design**
- Construction
- Testing
- Maintenance
- Quality Assessment
- Documentation

Unified Modeling Language - UML

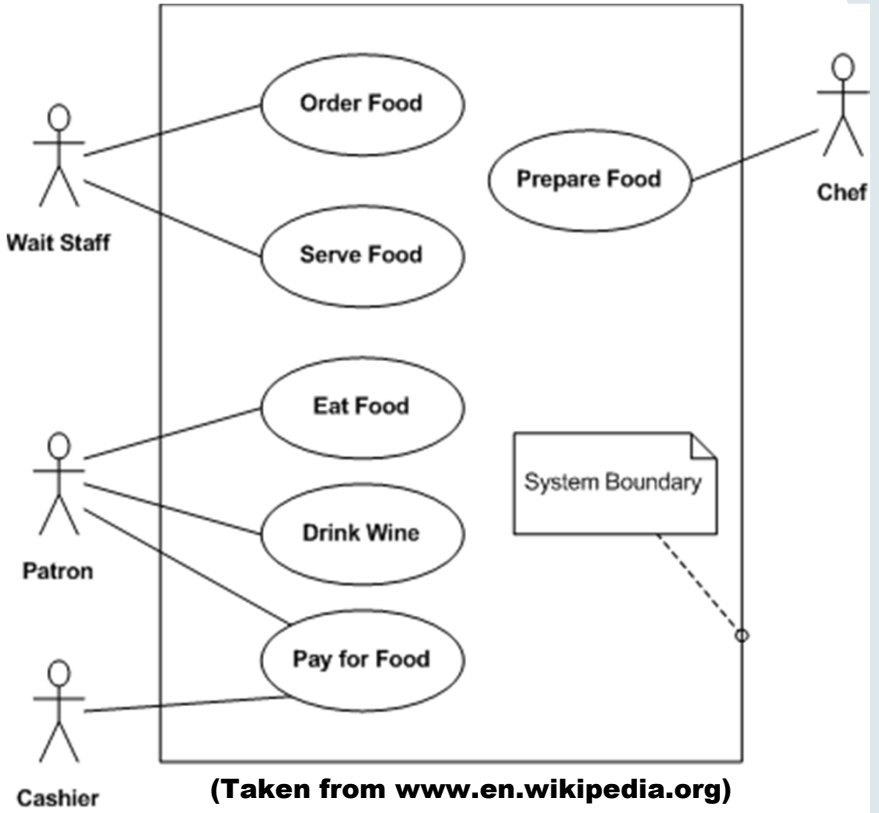
- Specification language for object modeling
- Officially defined at the Object Management Group (OMG)
- Non-proprietary
- General-purpose
- Standardized graphical notion
- Extendable

UML Model

- **Functional Model**
  - Functionality of the system from the user's Point of View
  - e.g. Use Case Diagrams
- **Object Model**
  - Structure and substructure of the system (objects, attributes, operations, associations)
  - e.g. Class Diagrams
- **Dynamic Model**
  - Internal behavior of the system
  - e.g Sequence Diagrams, Activity Diagrams, State Machine Diagrams

Functional Model: Use Case Diagram

- Focuses on the value provided by the system to external entities (human users or other systems)
- Promotes common understanding between the customer/owner/user and the development team

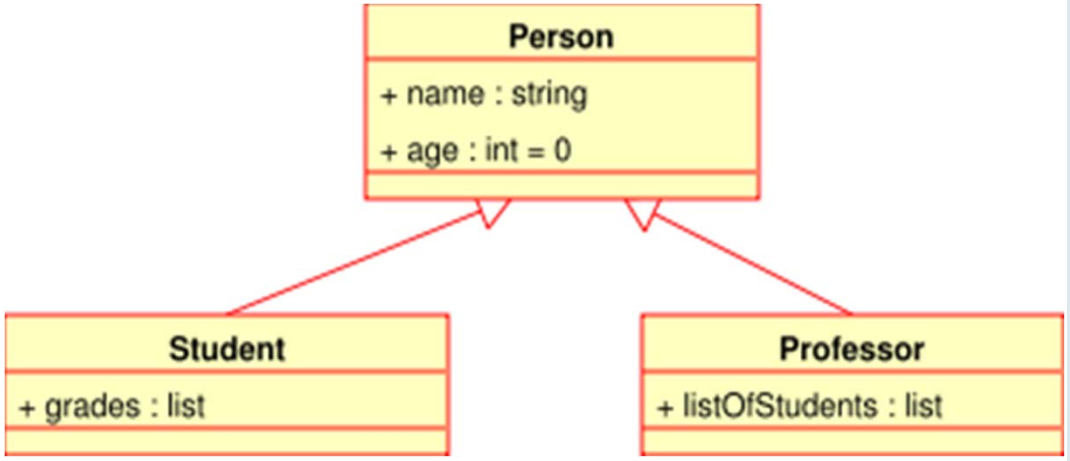


(Taken from www.en.wikipedia.org)

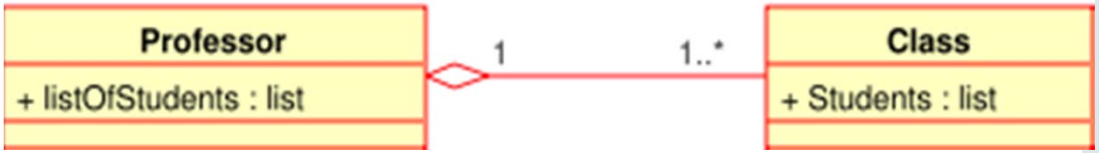
Object Model: Class Diagram

Describes classes, attributes and relationships

- *e.g. generalization*



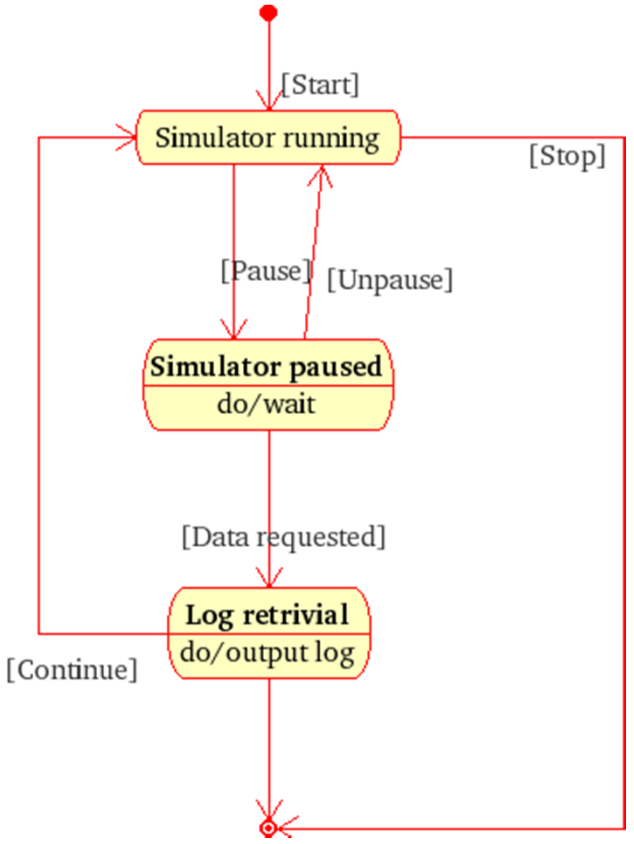
- *e.g aggregation*



(Taken from www.en.wikipedia.org)

Dynamic Model: State Diagram

- Filled circle, denoting start
- Hollow circle, denoting stop
- Rectangle, denoting state
  - Name
  - Activity
- Arrow, denoting transition
  - [ ] denoting condition



(Taken from www.en.wikipedia.org)

Definition of SW Testing

Testing is the process of executing a program or system with the intent of finding errors.

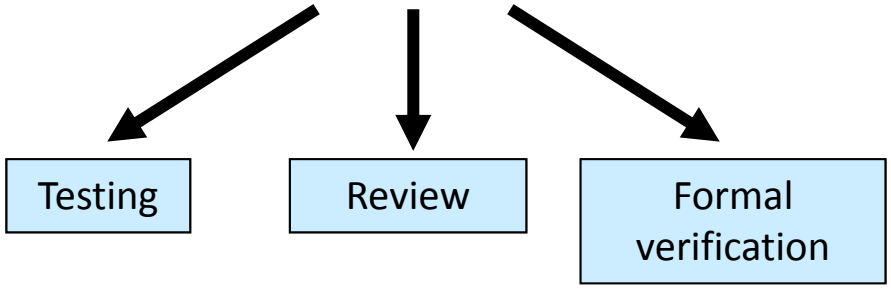
[Myers 79] The art of software testing

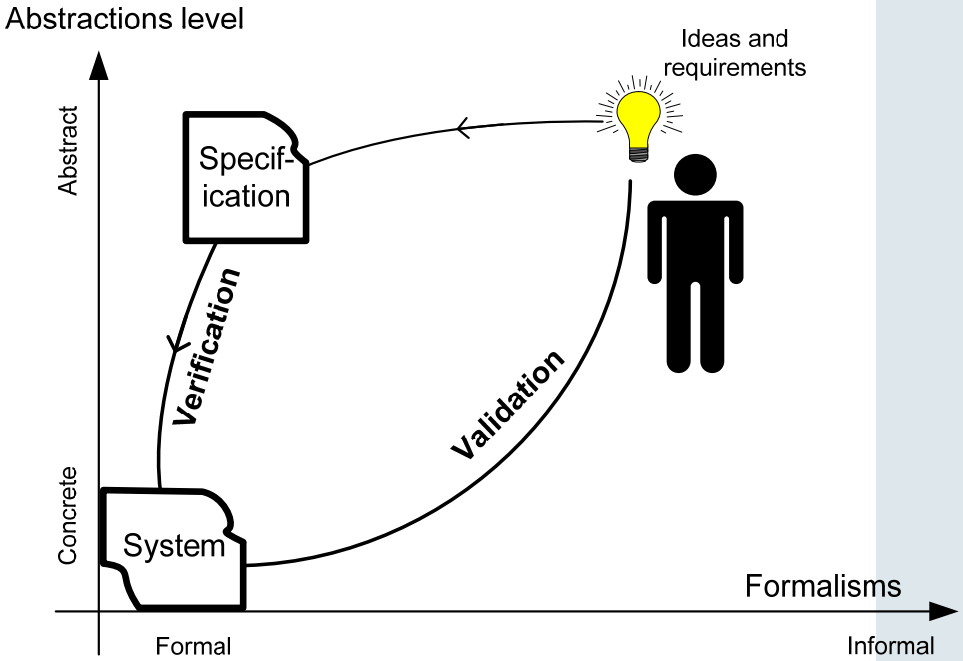
Testing can show the presence of bugs, but never their absence.

[Dijkstra (1972)]

Validation vs. Verification

- **Validation** – are we building the right system?
- **Verification** – are we building the system right?
  - Does the system/software meets its specification





Black-Box testing

- Functional focus
  - Test data are derived by the functional specification
- Specification driven, examines system from the services it provides via its interface - **"Exhaustive input testing"**
  - Tester considers the SW/system as a black box, visible are only: Inputs, Outputs and Specification
- No implementation details can be used to build test-cases (they are not known)
- Remaining problem (is the Specification correct?)
  - Approx. 30% of bugs due to incorrect Specification

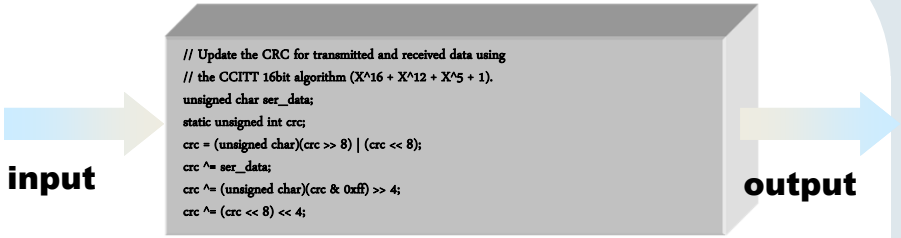
Black-Box / Functional testing

- **Boundary values**
  - use inputs that represent “boundary” values within a particular range

  `e.g., for unsigned char, use 0 and 256`

- **Exception testing**
  - Test what should trigger a failure mode or an exception mode
  - Error guessing: test based on prior experience

White-Box testing



- Structural focus - unit and system testing
  - Test data are driven based on knowledge of program design/implementation
- Performed in parallel with code development
  - Can be started the first day of development
- **Goal:** exercise paths and states within the object
  - **„Exhaustive path testing“**

White-Box testing

- **Code coverage**
  - Each code line shall be executed at least once during the testing
- **Branch coverage**
  - DC - Decision coverage
    - All possible decisions
  - MCDC - modified condition decision coverage
    - All possible variants of the decision

```
if (A || B) {dosomething1} else {dosomething2}
```

When do we stop testing?

Is a trade-off between budget/time and quality

Pessimistic:

- whenever some, or any of the allocated resources - time, budget are exhausted (unfortunately most often used).
- test cases are exhausted

Optimistic:

- when either reliability meets the requirement, or the benefit from continuing testing cannot justify the testing cost.

ENDE

Danke für die Aufmerksamkeit!