Chair for Circuit and System Design

Lecture
EDA Tools

Tools, Methods and Algorithms used for Designing Circuits and Systems

CHEMNITZ UNIVERSITY OF TECHNOLOGY

Chair for Circuit and System Design

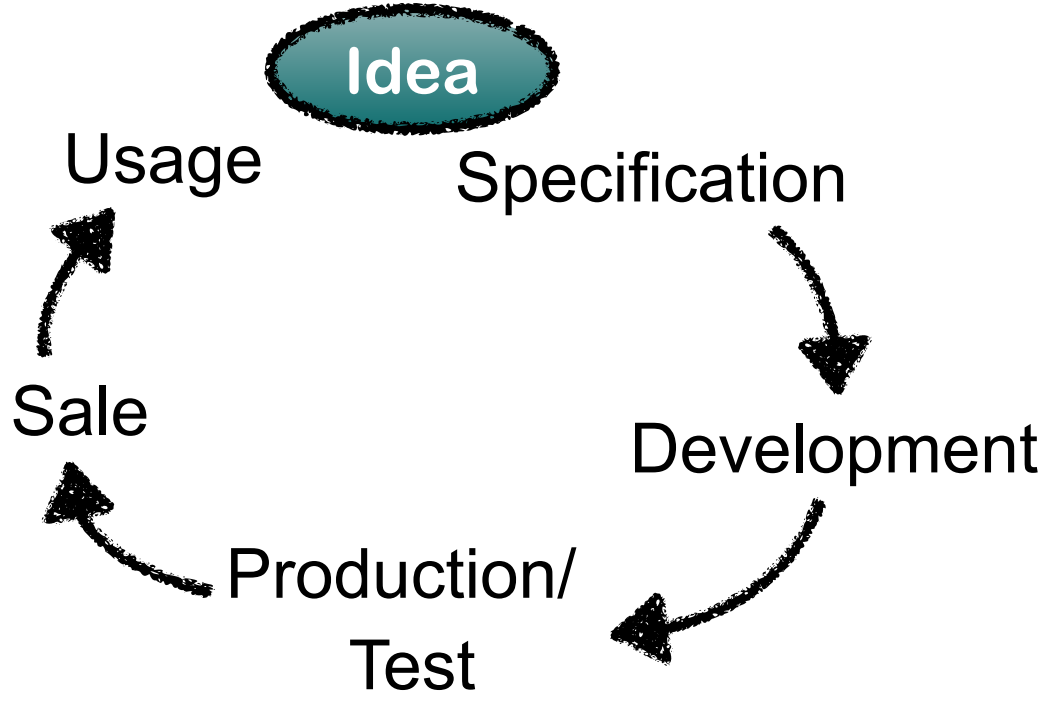
Specification

credits to:
Prof. Stefan Leue
Prof. Christian Haubelt

CHEMNITZ UNIVERSITY OF TECHNOLOGY

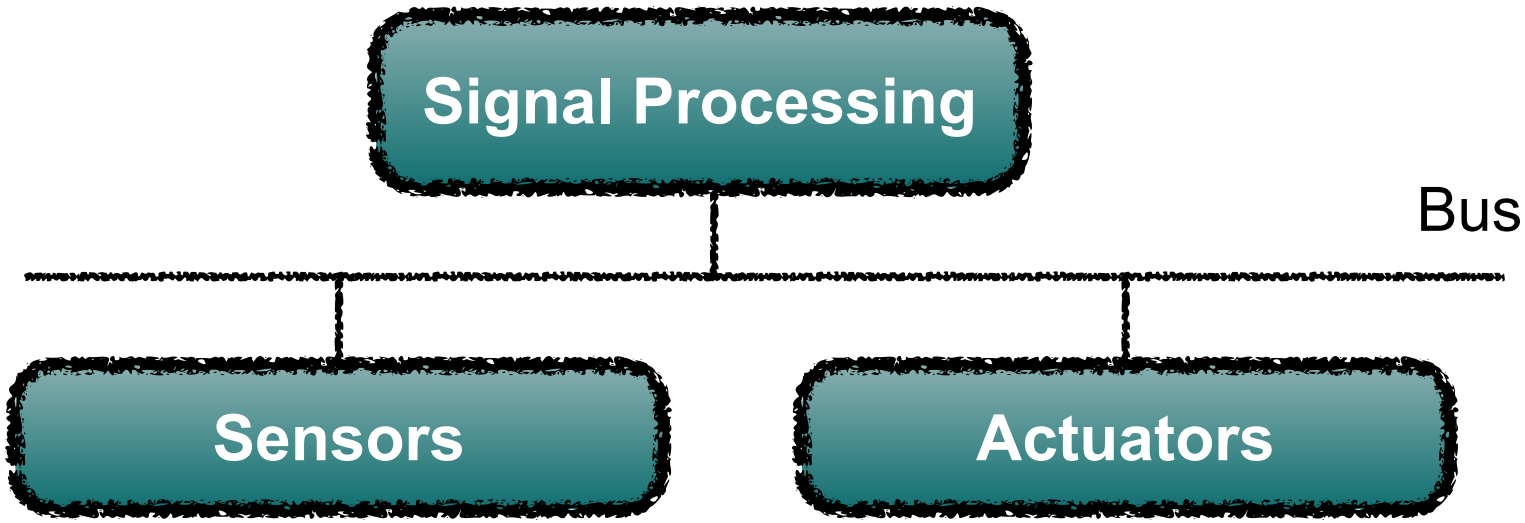
Terminology: System and lifecycle

- System (DIN/IEC 60050-251): Is a set of related elements that can be seen as a whole in a specific context and can be differentiated from its environment.



Terminology: Technical System

… contain components from the field of **Electronics** and **Computer Science** and its interacting environments. Usually these are **Sensors**, **Actuators** and **Signal Processing** elements.



System Design: **V-Modell**

The V-Modell is the approach to model the process of **Software Development**, that represents the design standard for IT systems of public authorities in Germany.

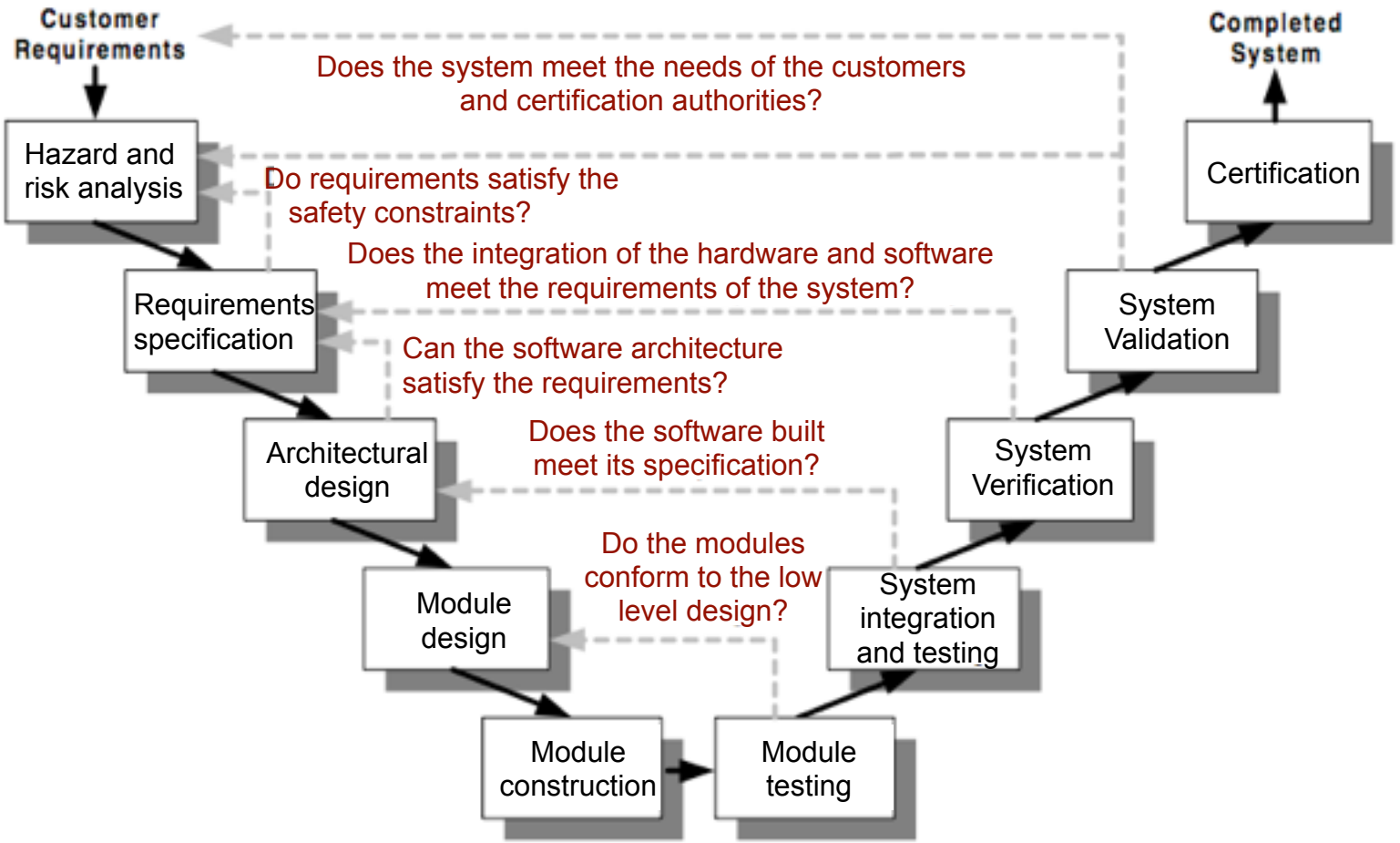
The modelling approach describes **activities** and **results**, that are to be conducted and compiled respectively during Software design (no chronology is defined).

In addition to military applications, the V-Modell is also mandatory for all federal authorities and some state administrations and is employed as in-house standard for Software design by many companies.

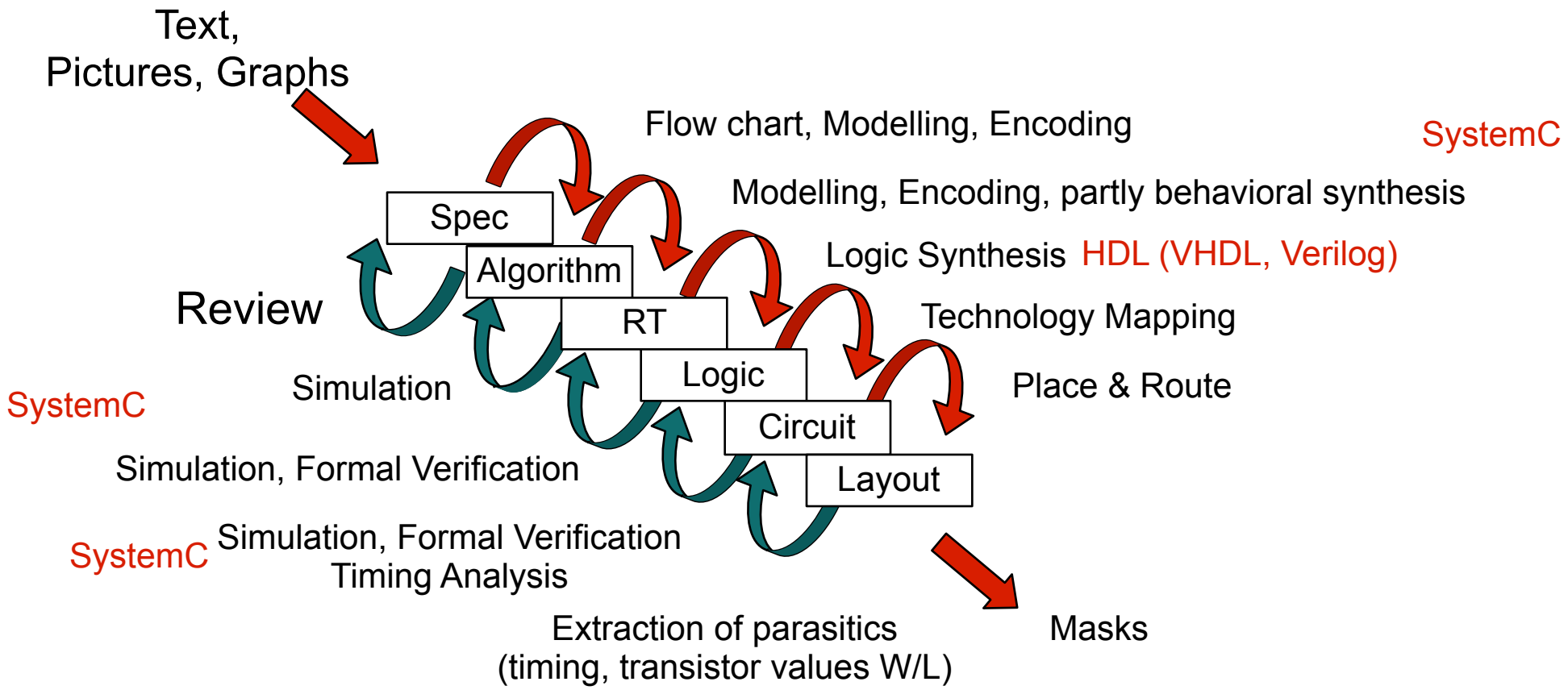
The V-Modell-97 was superseded by V-Modell XT in February 2005.
The model of approach is a process model, helping projects to be conducted in compliance with **ISO 9001 standard**.

Links: http://www.kbst.bund.de/ - http://www.v-modell.iabg.de

System Design: **V-Modell**



IC-Design: Waterfall Model



Terminology: Verification

Proving the correctness of a design by means of testing or simulation

Checking the result/functionality of a design step compared to the step‘s specifications (using different methods)

- **Simulation**: Verification of a Hardware description on a computer (e.g. HDL Simulator)

- **Emulation**: Verification of a netlist using „real“ components (prior synthesis required)
  - FPGA Board: quasi real time, if ASIC descriptions are emulated
  - HW Accelerator: real time, „real“ gates

Terminology: Formal Verification

Proving the correctness of a design step by (automatically) mathematically proving the equivalence of the result of this step and the step‘s specification (the result of a prior step), with regard to certain properties.

Examples:

- Comparing Layout with electrical circuit (LVS), comparing electrical circuit with logic description
- Logic with logic, e.g. after minimization, re-timing (equivalence check) **EDAT-II**
- Logic and behavioural description: still in research today
- Model Checking (proving the existence of specific properties - Searching for a counterexample to an assertion) **EDAT-II**

Terminology: Validation

Validation comes at the end of the design process

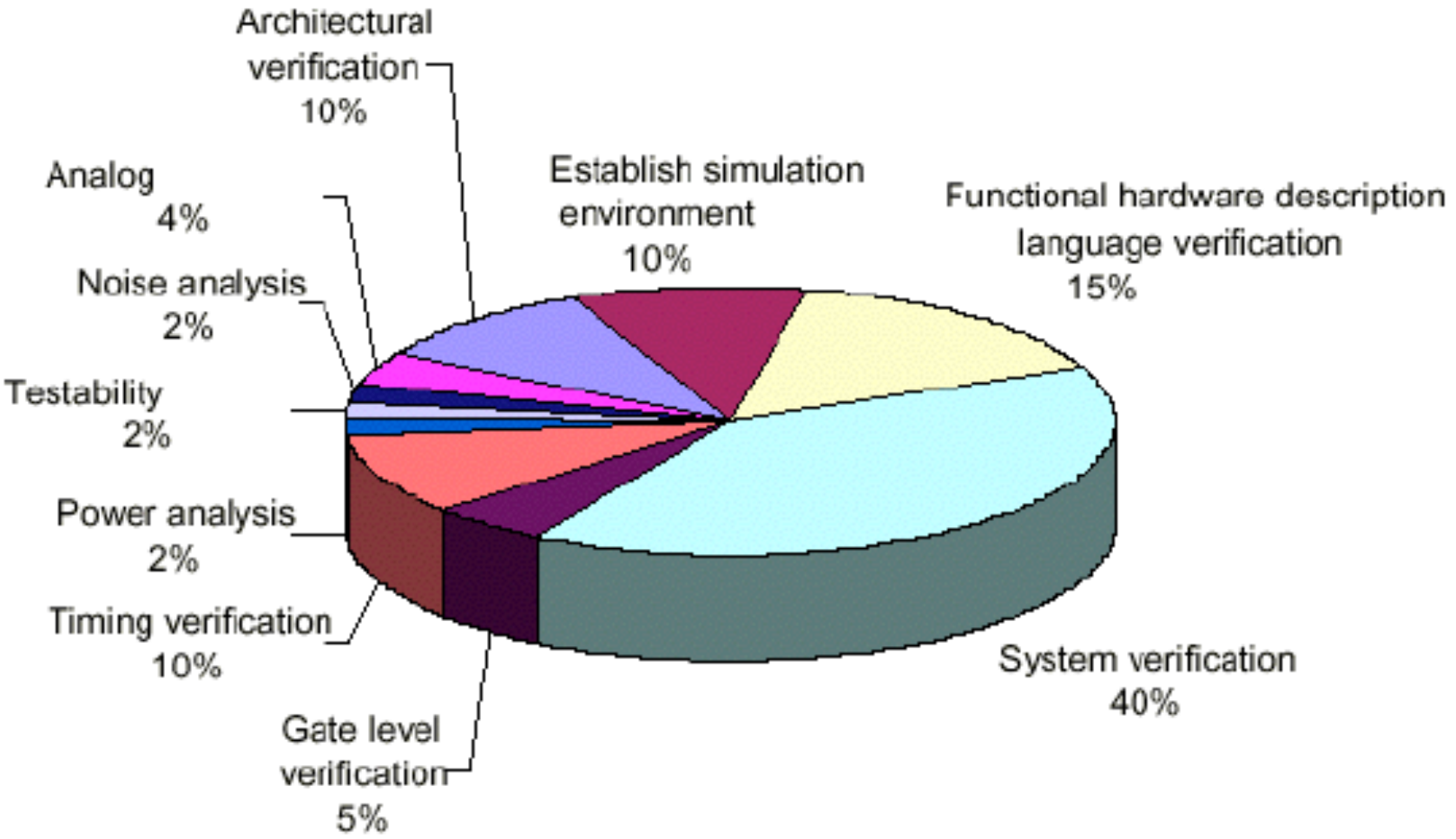
- Testing first prototype boards „in the field“ or in laboratories
- Time Pressure: every department involved in the design process wants to work with a prototype

  - System: interfaces to peripheral/other devices, system behaviour
  - Software: driver, new features, ...
  - Hardware: ASIC, FPGA, timing, ...
  - Board: Power, Heat, ...

Simulate -> (Emulate) -> Validate

PC -> (FPGA Prototype) -> IC/Board

Distribution of verification effort



System Specification - Definition

- Describes system components (system, support system or segment)
- Describes all functional and non-functional requirements
- Hierarchy: deducing requirements from the specifications of superordinate system components or the overall system specification, respectively
- Requirement resource for design and decomposition of system architecture
- Modification of a system component during a design step always requires to change the system specification
- Test specification defines tests to proof compliance of interfaces and requirements

System Specification

System Specification results

- Description of requirements concerning a system component
- Definition of Interfaces
- Description of refinement of requirements and interfaces into subordinate system components
- Description of assignment of requirements and interfaces to subordinate system components
- Created hand in hand with architecture design of the system

Role of System architect:

- Ensures consistency between specification and architecture

Requirement traceability

- Ensuring, that all requirements of a design component are taken into account when refining to another level of hierarchy.

Specification: Customer contracts Supplier

Product requirements document (PRD)

Functional Specification Document (FSD)

Product Requirements Document (PRD)

Used for:

- Customer determines and documents all relevant requirements of the system
- Contains all information necessary for the supplier to design the system
- Mandatory requirements for the system to be designed
- Basis of call for offers and drafting of contracts
- Reference for preparing an offer
- Attachment of the contract between customer and supplier
- Requirement set of general conditions for design
- Basis for target specification, overall system specification for supplier

Product Requirements Document (PRD)

Contains:

- Functional and non-functional requirements for a system
  - Basic doc of requirement tracing and change management
- Draft of overall system design
  - Takes into account environment and infrastructure in which the system will be operated
- Contains guidelines for technology decisions
- Identifies lifecycle stages of the design
  - i.e. logistics requirements, storage and shipping details
- Terms of delivery and acceptance criteria

Requirements formatted in a way that supports

- Traceability
- Change-Management / Version-Control
- during whole life cycle

Functional Specification Document (FSD)

also known as:

- Traget specification
- Overall System Specification

Equivalent part of customer‘s requirements (PRD) on supplier‘s side

Prepared by supplier in cooperation with customer

Initial document for system designer

Requirements are inherited from PRD and fittingly formatted to suppliers internal needs

http://h90761.serverkompetenz.net/v-modell-xt/Release-1.1/Dokumentation/html/

Functional Specification Document

FSD process results in:

- Functional and non-functional requirements for the overall system to be designed
- Initial draft of system architecture
- Overview of interfaces
- Support systems and external components are identified
- Mapping to all requirements of the PRD
- Detailed requirements for logistics
- Terms of delivery and acceptance criteria specified in details

Requirement tracing: ensuring all requirements are considered

Role of Requirements analyst: key person in this process

- requires knowledge in System design, System safety, Ergonomics, Logistics,…
- depends on intensive support from specialists of all the different disciplines

Why is Specification so difficult?

No comparable system engineered so far

- Problem never solved before
- Solution not yet known
- Assumptions regarding the system based on pure speculation
- Hard to estimate the required time and human resources needed to finish the project

Requirements not understood correctly

Requirements change during lifecycle (Development cycles)

- “When I saw the product, I knew it was not what I really wanted”

Complex interactions between services / components

- Feature Interaction: call forwarding / call screening
- Reverse thrust vs. accidental activation

Why is Specification so difficult?

e.g. Software: Reasons for failure of software projects: (standish group)

- incomplete requirements: 13,1%
- lacking inclusion of user: 12,4%
- lack of resources: 10,6%
- unrealistic expectations: 9,9%
- lack of management support: 9,3%
- changing requirements and specifications: 8,7%
- lack of planning: 8,1%
- system no longer needed: 7,5%

Identifying, Understanding, Documentation and Specification of requirements are the most common causes for the failures of software projects as listed above.

Relevance of early design stages

Wrong requirements are usually caused by incorrect facts, omissions, inconsistencies or ambiguities

- Analysis of Navy A-7E Specification, 77% of faults are not typing errors
  - 49% incorrect facts
  - 31% omissions **-> remarkable!!!, barely noticeable**
  - 13% inconsistencies
  - 5% ambiguities

Wrong requirements can be detected

- More effective inspections (manual or with automated analysis/verification tool)
- Substantial case studies using automated analysis tools, like SMV, SPIN, bounded model checking, etc.

EDAT-II

Terminology: Requirements

Requirement specification (PRD/FSD) often forms the foundation of contractual agreement between HW/SW supplier and customer.

ANSI/IEEE Standard 729-1983 “Glossary of Software Engineering Terminology”

- **Requirement**: “… (2) A condition or capability that must be met or professed by a system component to satisfy a contract, standard, specification or other formally imposed document.”
- **Requirement Specification**: “A specification that sets forth the requirements for a system of system component; … typically included are functional requirements, performance requirements, interface requirements, design requirements and development standards.”

“A requirements specification is a document containing a complete specification of **what** the system will do **without** saying **how** it will do that.”

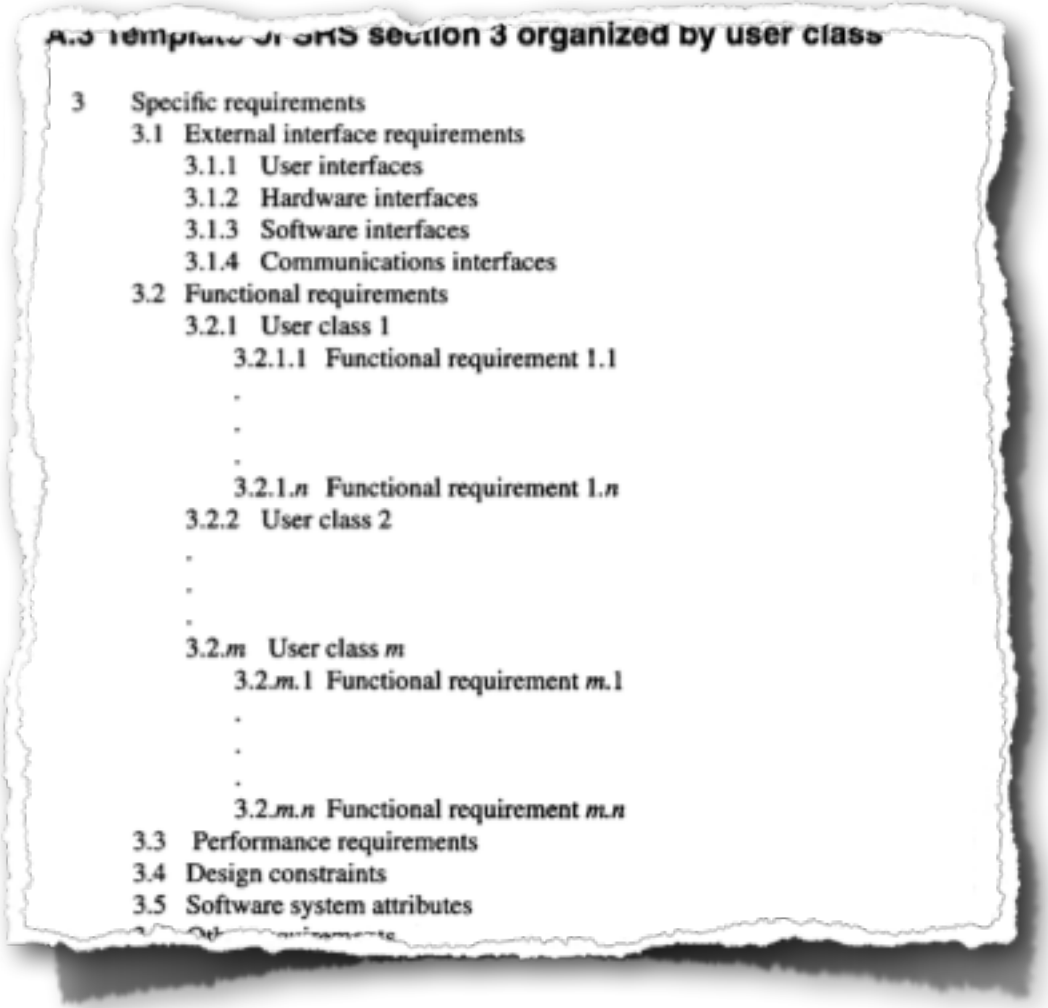
[Davis] A. M. Davis, Software Requirements - Objects, Functions and States, Prentice-Hall, 1993

Format of Requirement Specifications

Different, partly standardised templates

Example: ANSI/IEEE STD-830-1993 Standard

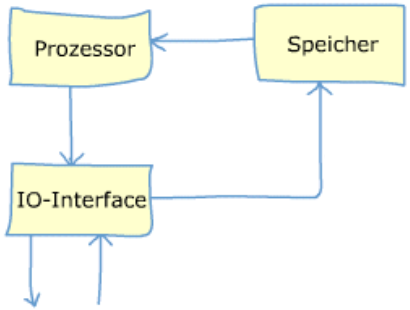
- Introduction (Purpose, Definitions, References, Abstract)
- General Description (Product functions, User characteristics, General conditions, Assumptions and Dependencies)
- Specific Requirements
  - Functional requirements (Input, Process, Output)
  - Requirements concerning external interfaces
  - Performance requirements
  - Design requirements
  - Attributes (Availability, Safety, Maintainability)
  - Other requirements



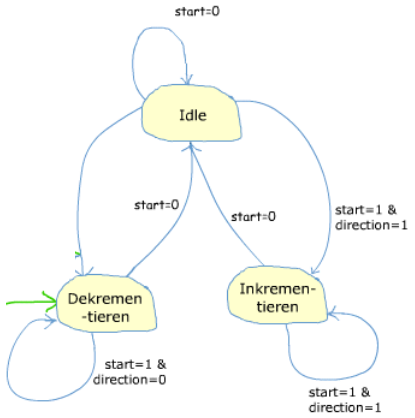
A.3 Template of SRS section 3 organized by user class

3 Specific requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 Functional requirements
  - 3.2.1 User class 1
    - 3.2.1.1 Functional requirement 1.1
    - .
    - .
    - .
    - 3.2.1.*n* Functional requirement 1.*n*
  - 3.2.2 User class 2
  - .
  - .
  - .
  - 3.2.*m* User class *m*
    - 3.2.*m*.1 Functional requirement *m*.1
    - .
    - .
    - .
    - 3.2.*m*.*n* Functional requirement *m*.*n*
- 3.3 Performance requirements
- 3.4 Design constraints
- 3.5 Software system attributes

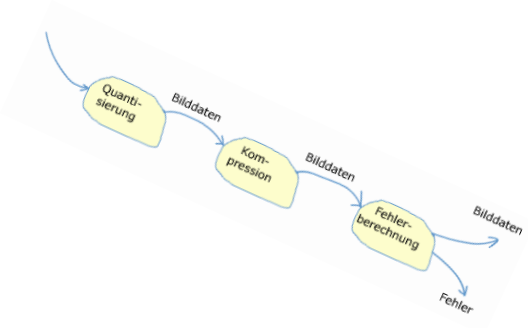
Formal Specification Methods



Block Diagram



State Charts



Task Flow Graph

```
#include "systemc.h"
SC_MODULE(nand2)
// declare nand2
sc_module
{
  sc_in<bool>
...
```

System-Level-Design-Language

Describe the structure of a system

Used to describe functionality, special focus on control flow.

Describe data flow

Languages designed to express system descriptions.

Languages for Requirement Specifications

Native vs. formal notations

– native language

"*if the telephone earphone is picked up, then a dial tone sounds*"

+ expressive

+ understood by everybody involved in the design process (??)

- ambiguities

– formal notations and languages (using mathematical semantics)

$$(\forall t_1, t_2 \mid t_1 \leq t_2)(ringtone(t_1) \Rightarrow dialtone(t_2))$$

+ unambiguous

+ automatically analyzable to a large extent

- inexpressive (especially if automatically analyzable)

- only mastered by some of the people involved

Languages for Requirement Specifications

Checklist for Specification Languages

- Usability
- Implementability (code generation)
- Support for testing and simulation
- Verifiability (e.g. readable for professionals)
- Modularity
- Maintenance
- Level of Abstraction and Expressiveness
  (How good can objects in the specification describe the objects in the application world?)
- Logic completeness
  (Are there **formal semantics** of the language to discover inconsistencies?)
- Runtime safety

nach Ardis (siehe [Pfleeger], 4.11)

Languages for Requirement Specifications

Checklist for Specification Languages (contd.)

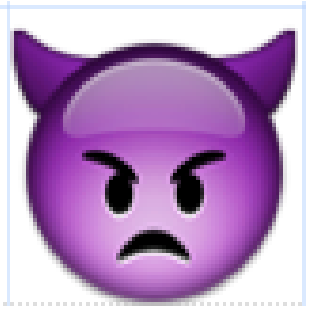
- Tool stability/history
- Looseness (Are incompletions and non-determinisms allowed?)
- **Learning curve**
- Technical maturity (Certification and Standardization)
- Data modeling (Representation of data and relationships)
- Discipline (Does the language force the user to write well structured, comprehensible and well behaving specifications?)

Specification languages are domain and problem specific.

Examples of Languages used for Specification

MS Word:

- Entry of text and figures
- no EDA support
- not formal
- interpretable
- no automated link to Design flow



Examples of Languages used for Specification

Petri nets:

PhD thesis of Carl Adam Petri 1962 → foundation of net theories

A Petri net is a mathematical model of concurrent systems

Method for the description and analysis of the information and control data flow of information processing systems

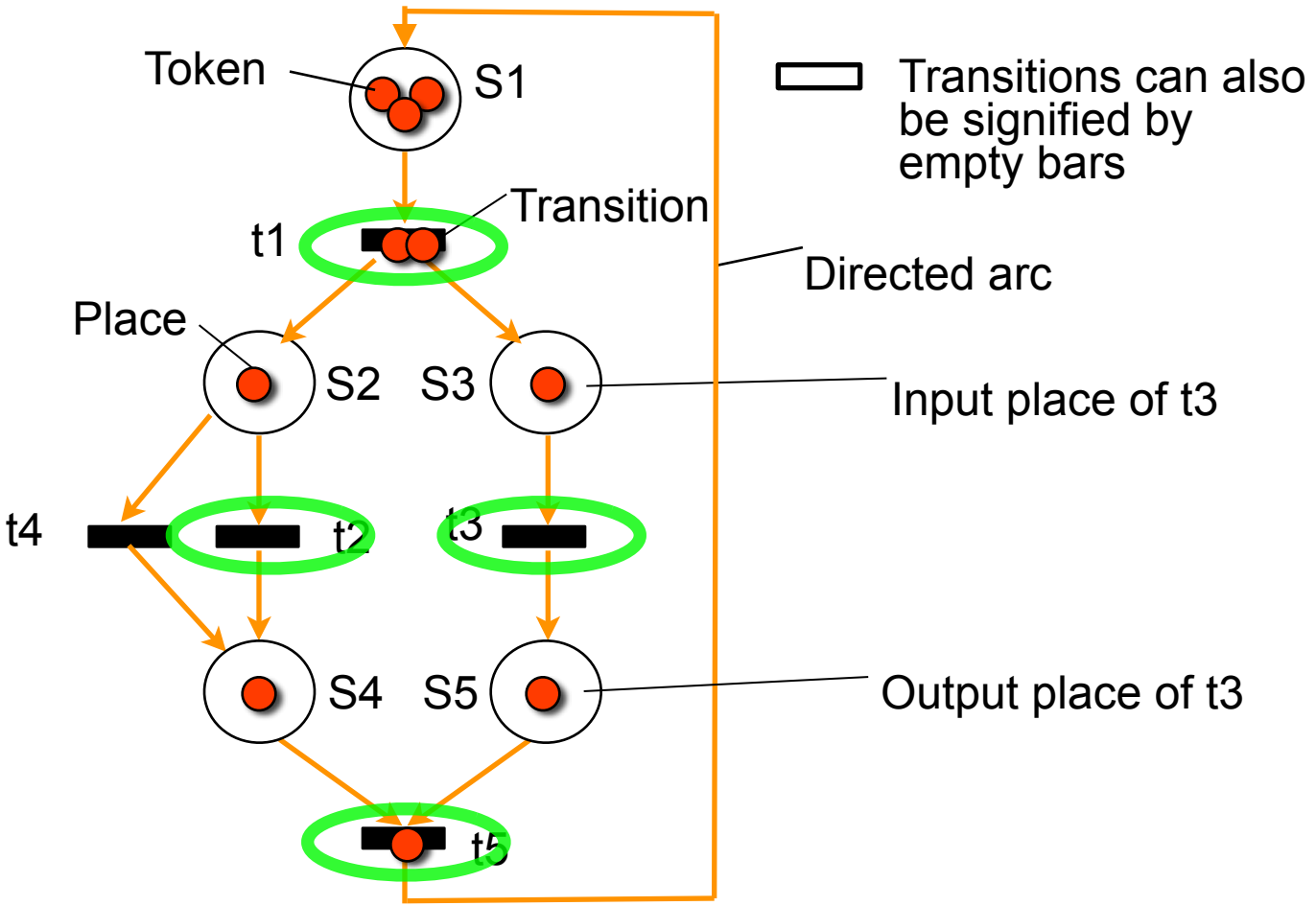
Abstract models of information and control data flow

Description of systems and processes on different levels of abstraction → different degrees of details

→ PN is similar to finite state machines, but more powerful.

- every FSM can be represented by a PN
- some PNs cannot be represented as FSM

Petri nets


Token
S1
Transition
t1
Place
S2
S3
t4
t2
t3
S4
S5
t5
Transitions can also be signified by empty bars
Directed arc
Input place of t3
Output place of t3

Examples of Languages used for Specification

UML:

a **U**nified **M**odeling **L**anguage for:

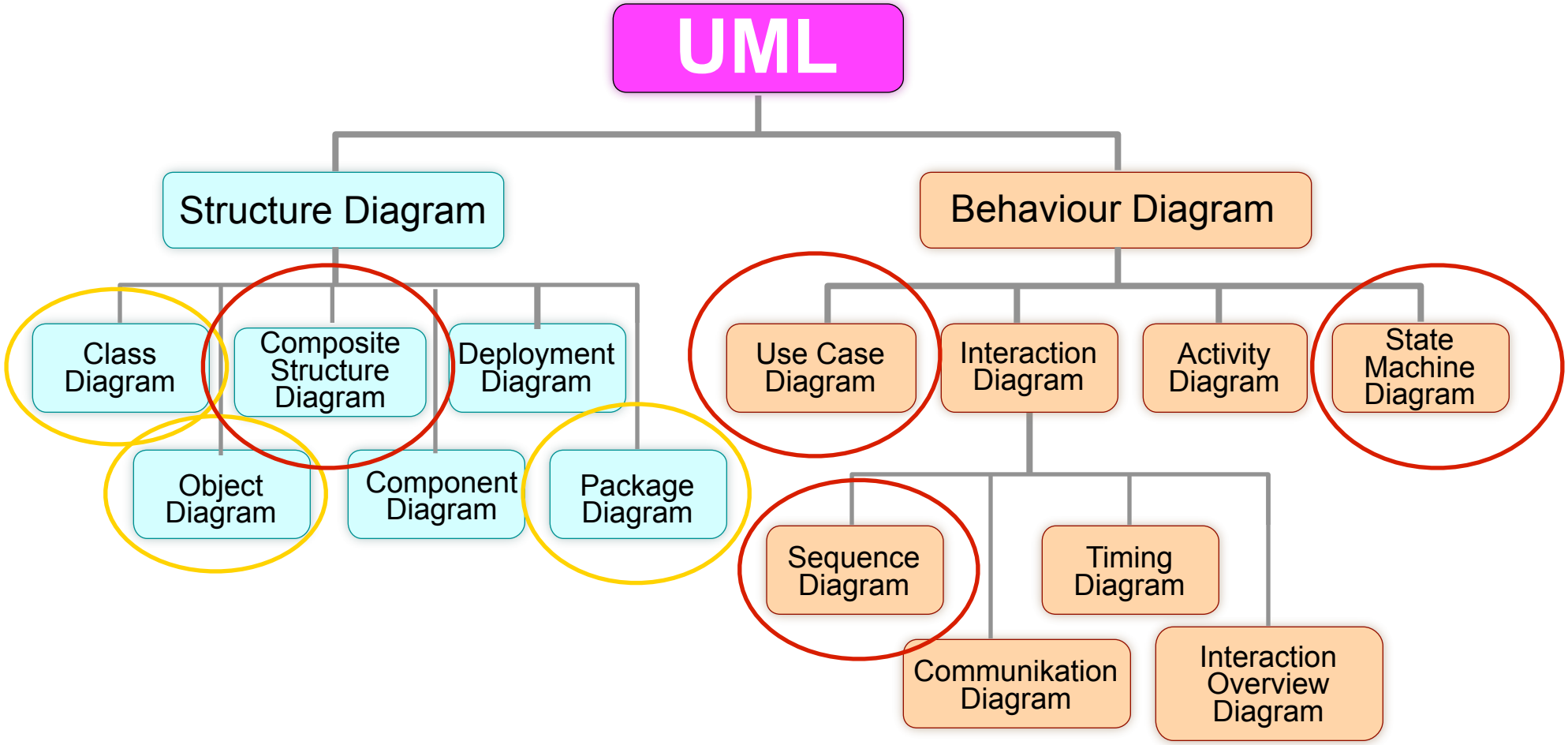
- Specification
- Visualization
- Construction
- Documentation

graphically specifying systems

Areas of application:

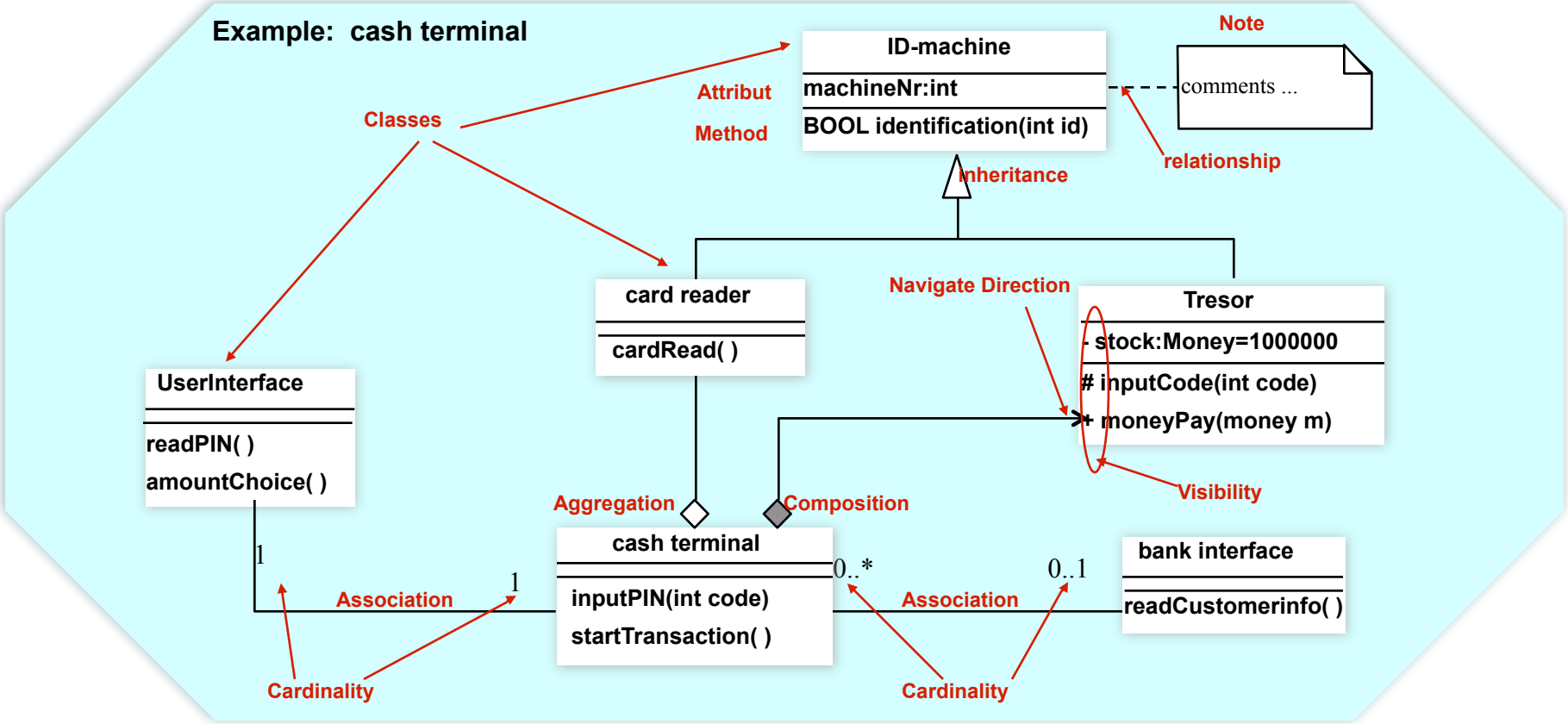
- Software systems
- Business models
- other non-Software systems

Unified Modeling Language - Types of diagrams



Unified Modeling Language - Types of diagrams

Class diagrams in an object orientet model are defining the classes and the relationships to each other



Examples of Languages used for Specification

SystemC:

SystemC permits the generation of a formal, and thus executable, System specification (in contrast to informal specifications)

Advantages:

- Generation of consistent, error free and complete specifications
- Unambiguous interpretation of the specification
- Validation of system functionality is possible before implementation
- Generation of performance models and validation of system performance
- Testbench of executable Spec can be used as reference environment for simulations of system implementation(s) (requires more detailed stimuli data)

SystemC

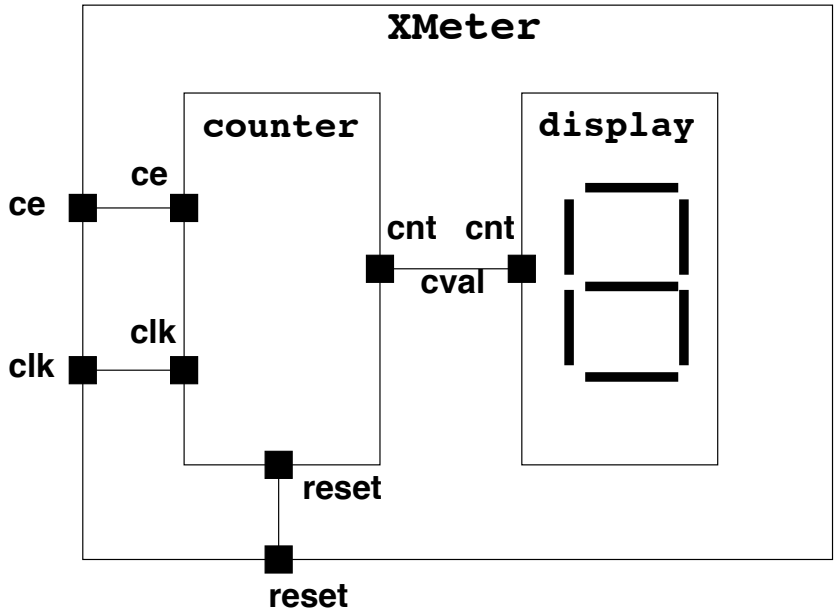
```
SC_MODULE(XMeter) {

   sc_in_clk clk;
   sc_in<bool> ce, reset;

   counter* C1;
   display* D1;
   sc_signal<sc_uint<4> > cval;

   SC_CTOR(XMeter){
      C1 = new counter(„C1“);
      D1 = new display(„D1“);

      C1->clk(clk);
      C1->reset(reset);
      C1->ce(ce);
      C1->cnt(cval);
      D1->cnt(cval);
   }
};
```



Inside the module constructor

- instantiated submodules are allocated
- static binding of ports to signals/ports

Characteristics of Requirement Specifications

Correct

- All facts mentioned in the requirement specification correspond to a property required by the system to be designed.
- Counter example:
  - The specification states that the callee hears a dial tone when the caller hangs up, while system requirements demand the callee hears a busy tone in this case.

Characteristics of Requirement Specifications

Unambiguous

- There is only one single interpretation of all facts mentioned in the requirement specification
- Natural language description often contain sources for ambiguity
  - At speeds up to 60 km/h cruise control shall remain deactivated
  - For up to 12 airplanes on the control screen the small representation size shall be used, otherwise the large representation size
    - To ensure clarity it is not substantial if this is interpreted as < 12 or ≤ 12
    - However, a problem arises, if two different design teams are responsible for the two cases, and there is no agreement on how the case =12 is handled -> a black screen can potentially appear in this case

Characteristics of Requirement Specifications

Completeness

Definition 1: Everything the system shall be able to do is expressed in the specification.

- Does this mean that a specification also has to express how the system shall **not** behave?
  - might be hard to express due to the number of possible system behaviors
  - formal specifications can help to reach this goal, e.g.

$$(\forall A, B)(Call(A, B) \wedge Idle(B) \Rightarrow (Ring(B) \wedge (\forall X \neq B)(\neg Ring(X))))$$

... but there might be good reasons for other phones to ring as well
... and it is hard to understand for people not knowing the formal expressions...

  - State machine and logic specifications achieve this requests due to their semantic models (“closed world assumption”)

Characteristics of Requirement Specifications

Completeness (cont.)

Definition 2: The response of the HW/SW system on all kinds of possible input values are defined in the specification.

- Important: Completeness of a requirement specification is different from the completeness of a specification language (e.g. logic)

Verifiability

- There exists an effective procedure to check - either manually or supported by automatic processing - if a product meets the properties required by the specification.
- Examples: formal verification, model checking, testing
- Many requirements are not verifiable:
  - "The operating system shall return control to the user after each command entered."
  - "The program shall never enter into an infinite loop."
  - "The user interface shall be easy to use."

Characteristics of Requirement Specifications

Consistency

- no requirement is inconsistent with another requirement
- possible inconsistencies:
  - conflicting behavior

    If the telephone earphone is picked up, then a dial tone sounds.

    If the telephone earphone is picked up, then a ring tone sounds.
  - inconsistent expressions
  - inconsistent properties
  - timing inconsistencies

Input event on a leads to output b at the same time.

$T_b = T_a + 0$ sec

Event b may never be observed within 15 seconds after event a.

$T_b > T_a + 15$ sec

$T_b \geq T_a + 15$ sec

?

Characteristics of Requirement Specifications

Understood/Traced

- Origin and cause for each requirement are obvious.
  - e. g. documentation, why certain values (e.g. for timing constraints) have been chosen

Traceable

- Requirement specification is written, such that each single requirement can be named isolated in a simple way
  - Usually: well-defined numbering scheme
  - Important for Software testing and Hardware verification

Design independent

- Requirement specification is not bound to a particular SW/HW architecture and specifies no algorithmic implementation
- Otherwise, it is over specified

Short summary

System design costs rise rapidly

- "Soft" Standards: Trend towards reconfigurability is noticeable
- Embedded solutions: HW: FPGA, SW: Network processing units

Future systems more and more complex

- Specification and Verification gets more expensive
- State of the art tools do not scale to industrial specifications
- Specification is still textual, and thus ambiguous and non-formal
- Specification is checked too late
  - "Tough" errors are not discovered until simulation

Understanding todays simulation constraints **EADT-I**

What is possible to design, what to simulate and how to do it

Goal: formal methods (starting in specification) **EADT-II**