Introduction
Motivation
Software engineering in industrial praxis

- requirements
  - requirements engineering
    - requirements definition
      - design
        - software architecture
          - implementation
            - program
              - testing
                - tested program

informal documents
Problems

- Requirements definition ambiguous ⇒
  » Interpreted in different ways by client and contractor
  » Difficult to assess whether design and implemented program conforms to the requirements

- Architecture described informally ⇒
  » Wrong interpretation by programmer
  » No precise description of the properties of implemented modules

- Validation is performed too late (requires implemented program)

- Tests can only reveal errors but cannot prove correctness
Software engineering with formal specifications

1. Requirements
2. Specifying
3. Validation
4. Prototyping
5. Abstract specification
6. Refinement
7. Verification
8. Intermediate specification
9. Refinement
10. Verification
11. Intermediate specification
12. Refinement
13. Verification
14. Concrete specification
15. Coding
16. Verification
17. Program
Advantages

- Requirements are formally defined
  - Client and contractor have a safe reference point
  - Well defined input for design
- Documentation of a software system on a high level of abstraction
- Formal definition of architecture and interfaces
  - Well defined input for implementation
- Formal proofs of system properties
- Rapid Prototyping (execution of specifications)
- Automatic code generation
- Stepwise refinement from abstract to concrete specifications
- Derivation of test data from formal specifications
Problems and limitations of formal specifications

- Formal proof that requirements of the client are satisfied cannot be conducted (validation instead of verification)
- Cryptic formalism make communication with the client difficult
- Not all problems may be formalized mathematically in an elegant way
- Construction of formal specifications is laborious and justified only for critical parts of a software system (lack of scalability)
- Use of specification methods and languages requires highly qualified developers
- Even formal specifications and proofs may contain errors
- Many researchers are primarily interested in theory and neglect applications
- Limited tool support
Basic Notions and Classification
Notions

- **Specification**
  - Description of (parts of) a software system which determines "what" but not "how"

- **Formal specification**
  - Specification with formally defined syntax and semantics

- **Implementation**
  - Realization of a specification (determines "how" to realize "what")
Notions

- **Specification language**
  \[ SL = \langle \text{Syn}, \text{Sem}, \text{Sat} \rangle \]
  - Syn syntactic domain
  - Sem semantic domain
  - \( \text{Sat} \subseteq \text{Syn} \times \text{Sem} \) "Satisfies" relation
Notions

- **Specifying**
  - Construction of a formal specification from informal requirements

- **Validation**
  - Check whether the specification "correctly" represents the informal requirements

- **Prototyping**
  - Execution of a specification for early validation

- **Proving**
  - Conducting formal proofs of the properties of a specification

- **Refinement**
  - Transition from an abstract to a concrete specification

- **Verification**
  - Formal proof that an implementation satisfies the specification

- **Coding**
  - Creation of a program satisfying the specification
Classification of specification approaches

- Transition from a specification to a program
  - Evolutionary: step-wise transformation in a broadband language
  - Discrete: different languages on different levels of abstractions

- Executability of a specification
  - Operational: specification can be executed

- Object of specification
  - Data-oriented: specification of data structures
  - Processes: specification of processes

- Way of specification
  - Behavioral: externally observable behavior ("black box")
  - Model-oriented: description with the help of an abstract model ("white box")
Illustration of the classification

### Behavior

```
abstract data type Stack =
  operations
    Push : Stack x Elem -> Stack;
    Pop : Stack -> Stack;
  axioms
    Pop(Push(s,e)) = s
    ...
end;
```

### Model

```
data type Stack =
  representation s : sequence[Elem];
  operations
    Push(s,e) = e s
    Pop (e s) = s
    ...
end;
```

### Data

### Processes

```
process Task =
  events Start, Suspend, Resume, Finish;
  behavior
    Start (Suspend Resume)* Finish
end;
```

```
process Task =
  events Start, Suspend, Resume, Finish;
  behavior
    Start (Suspend Resume)* Finish
end;
```
Classification of specification languages

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Processes</td>
</tr>
<tr>
<td>algebraic specifications (Act One, Clear, OBJ2)</td>
<td>VDM</td>
</tr>
<tr>
<td>Z</td>
<td>PROGRES</td>
</tr>
<tr>
<td>Lotos</td>
<td>SDL</td>
</tr>
<tr>
<td>CCS</td>
<td>statecharts</td>
</tr>
<tr>
<td>temporal Logic</td>
<td>Petri nets</td>
</tr>
</tbody>
</table>
Contents and Goals of the Lecture
General information

Goals:
- Provide a survey of specification languages
- Application of specification languages in software engineering
- Comparison and evaluation of specification methods and languages

Study information:
- Master of science Software Systems Engineering
- Focus: software engineering
- 4 ECTS credit points
- Knowledge in programming is assumed
- Basic knowledge in software engineering would be helpful
Literature

  *One of only a few text books which cover multiple specification languages (Larch, VDM, Z, and algebraic specifications)*

  *Collection of papers on specification and design*

  *Short survey on formal specifications*

  *Frequently cited survey on formal methods and specifications*