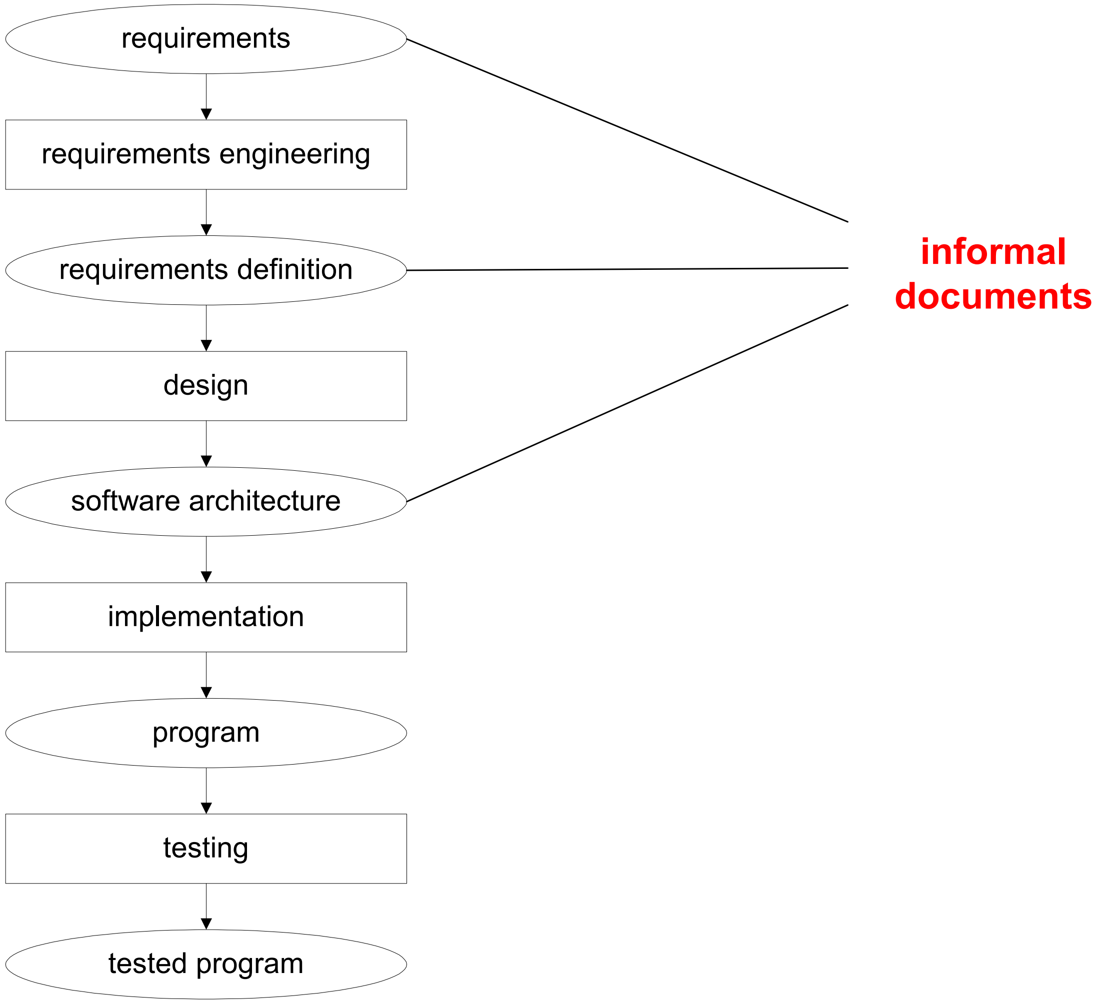
# Introduction

*Bernhard Westfechtel*

*Lehrstuhl für Informatik III, RWTH Aachen*

# Motivation

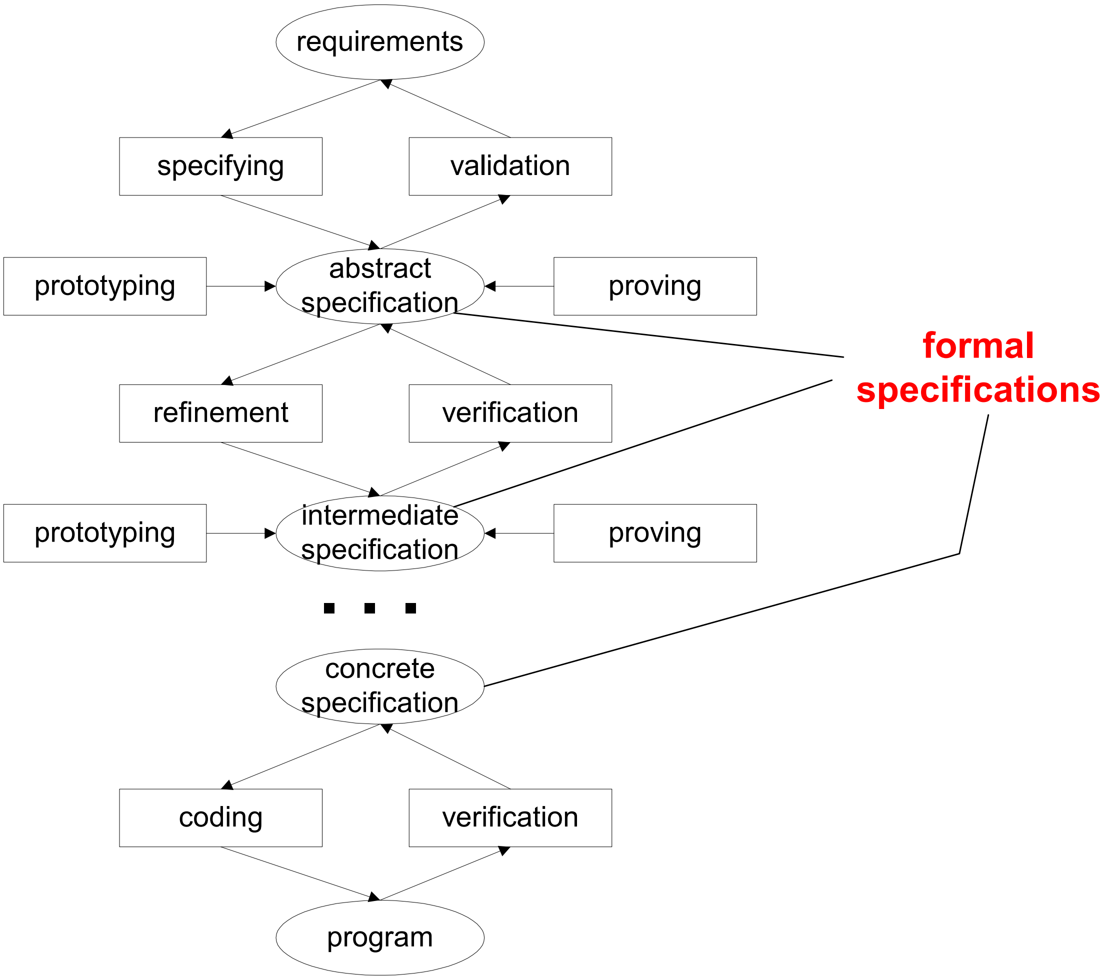
# Software engineering in industrial praxis



# 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



# 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 $\Rightarrow$ 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

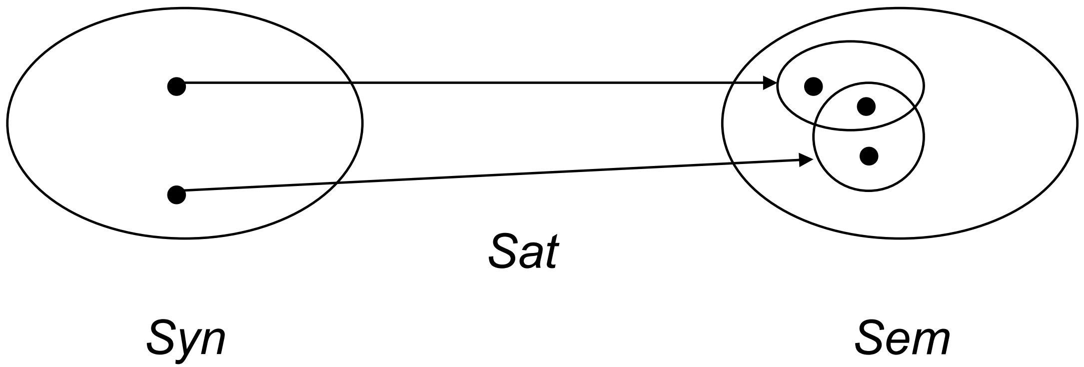
- **Specification language**

  *SL* = < *Syn*, *Sem*, *Sat* >

  » *Syn* syntactic domain

  » *Sem* semantic domain

  » *Sat* $\subseteq$ *Syn* x *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 | Model**

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

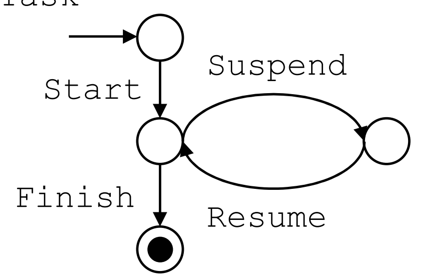
```
data type Stack =
    representation s : sequence[Elem];
    operations
        Push(s,e) = e s
        Pop (e s) = s
        ...
end;
```

**Data**

**Processes**

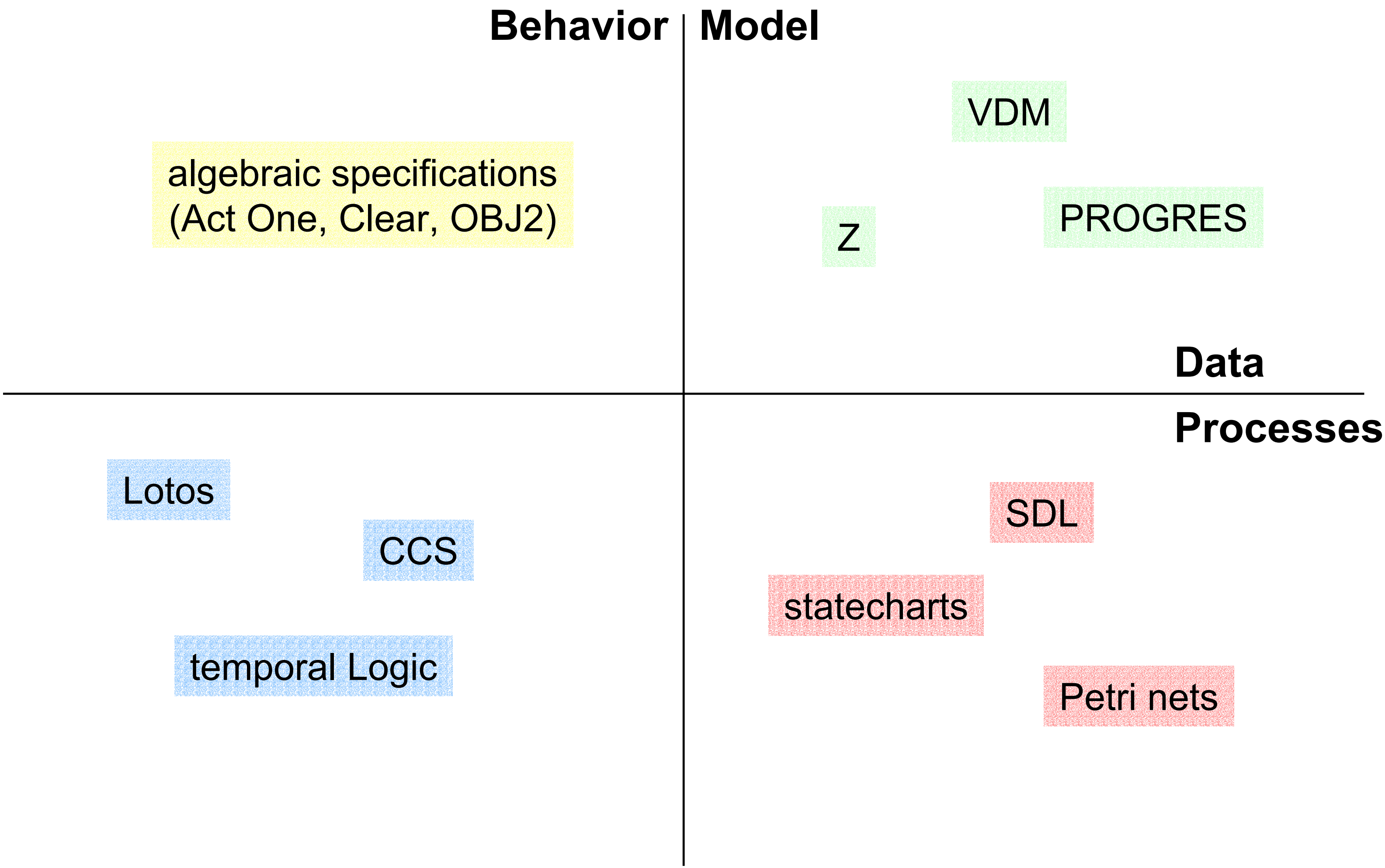
```
process Task =
    events Start, Suspend, Resume,
        Finish;
    behavior
        Start (Suspend Resume)* Finish
end;
```

```
process Task =
```




```
end;
```

# Classification of specification languages



# 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

- V.S. Alagar, K. Periyasamy: **Specification of Software Systems**, Graduate Texts in Computer Science, Springer-Verlag, 422 S. (1998)
  *One of only a few text books which cover multiple specification languages (Larch, VDM, Z, and algebraic specifications)*
- J.P. Bowen, M.G. Hinchey: **High-Integrity System Specification and Design**, Formal Approaches to Computing and Information Technology (FACIT), Springer-Verlag, 701 S. (1999)
  *Collection of papers on specification and design*
- M.-C. Gaudel, G. Bernot: **The Role of Formal Specifications**, in: E. Astesiano, H.-J. Kreowski, B. Krieg-Brückner (Hrsg.): Algebraic Foundations of Systems Specification, IFIP State-of-the-Art-Report, Springer-Verlag, S. 1-12 (1999)
  *Short survey on formal specifications*
- J. Wing: **A Specifier´s Introduction to Formal Methods**, in: Bowen et al., S. 167-200 (1990)
  *Frequently cited survey on formal methods and specifications*