On Complementing an Undergraduate Software Engineering Course with Formal Methods

CSEE&T 2020 — November, 9th–12th — Internet

Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
Structure

- **Working Definition** ‘Formal Methods’

- **Formal Methods** in the Context of Software Engineering (Towards Learning Objectives)

- The Challenge of **Complementation**

- Proposed **Didactical Approach**

- **Conclusion**
Definition.

[(Bjørner & Havelund, 2014)]

A method is called formal method if and only if its techniques and tools can be explained in mathematics.
Definition and Examples

Definition. [(Bjørner & Havelund, 2014)]
A method is called **formal method** if and only if its techniques and tools can be explained in **mathematics**.

Examples:
- Requirements Patterns
- Decision Tables
- Sequence Diagrams
- Class-/Object-Diagrams, OCL
- State Machines
- Pre- and Post-Conditions
- etc. etc.
**Definition and Examples**

**Definition.** [[Bjørner & Havelund, 2014]]
A method is called **formal method** if and only if its techniques and tools can be explained in **mathematics**.

**Examples:**
- **formal** Requirements Patterns
- **formal** Decision Tables
- **formal** Sequence Diagrams
- **formal** Class-/Object-Diagrams, OCL
- **formal** State Machines
- **formal** Pre- and Post-Conditions
- etc. etc.
Definition and Examples

**Definition.** [Bjørner & Havelund, 2014]
A method is called **formal method** if and only if its techniques and tools can be explained in **mathematics**.

**Examples:**
- **formal** Requirements Patterns → **automatic** consistency analyses
- **formal** Decision Tables → **automatic** test case generation
- **formal** Sequence Diagrams → **precise** acceptance test instructions
- **formal** Class-/Object-Diagrams, OCL → **unambiguous** documentation
- **formal** State Machines → **exhaustive** model checking
- **formal** Pre- and Post-Conditions → **automatic** static analyses
- etc. etc.
Definition and Examples

Definition. [(Bjørner & Havelund, 2014)]
A method is called **formal method** if and only if its techniques and tools can be explained in **mathematics**.

Examples:
- **formal** Requirements Patterns → **automatic** consistency analyses
- **formal** Decision Tables → **automatic** test case generation
- **formal** Sequence Diagrams → **precise** acceptance test instructions
- **formal** Class-/Object-Diagrams, OCL → **unambiguous** documentation
- **formal** State Machines → **exhaustive** model checking
- **formal** Pre- and Post-Conditions → **automatic** static analyses
- etc. etc.
Formal Methods in the Context of Software Engineering

real-world software engineering

(2) client

(1) engineer

(3)

formal method

(4) product

(5) practice
Formal Methods in the Context of Software Engineering

- Syntax and Semantics
- Properties (e.g., Consistency)
- Analysis Algorithms and Tools

real-world software engineering

(1) Syntax and Semantics
(2) Properties (e.g., Consistency)
(3) Analysis Algorithms and Tools
(4) Product
(5) Practice

client

engineer
(1) engineers need to **know syntax and semantics**
to formalise understandings of, e.g., a design idea
Formal Methods in the Context of Software Engineering

(1) engineers need to **know syntax and semantics** to formalise understandings of, e.g., a design idea

(2) engineers may need to **validate formalisations with clients** (is the formal description a valid model of, e.g., a requirement?)
(1) engineers need to **know syntax and semantics** to formalise understandings of, e.g., a design idea

(2) engineers may need to **validate formalisations with clients** (is the formal description a valid model of, e.g., a requirement?)

(3) engineers need to be able to **analyse for properties**
(1) engineers need to **know syntax and semantics** to formalise understandings of, e.g., a design idea

(2) engineers may need to **validate formalisations with clients** (is the formal description a valid model of, e.g., a requirement?)

(3) engineers need to be able to **analyse for properties**

(4) outcomes of formal analyses need to be **interpreted in context**, appropriate actions need to be taken
Formal Methods in the Context of Software Engineering

(1) engineers need to **know syntax and semantics** to formalise understandings of, e.g., a design idea

(2) engineers may need to **validate formalisations with clients** (is the formal description a valid model of, e.g., a requirement?)

(3) engineers need to be able to **analyse for properties**

(4) outcomes of formal analyses need to be **interpreted in context**, appropriate actions need to be taken

(5) techniques need to be **discussed in contemporary context**
Approach: Interpolative instead of Extrapolative

- Syntax and Semantics
- Properties (e.g., Consistency)
- Analysis Algorithms and Tools

Real-world software engineering

Client

Engineer

Formal method

Product

Practice
Approach: Interpolative instead of Extrapolative

Syntax and Semantics
Properties (e.g., Consistency)
Analysis Algorithms and Tools

real-world software engineering

client
engineer

informal
semi-formal
formal
Approach: Interpolative instead of Extrapolative

real-world software engineering

- Syntax and Semantics
- Properties (e.g., Consistency)
- Analysis Algorithms and Tools

formal method

(2) client
(1)
(3) engineer

(4) product
(5) practice

informal
semi-formal
formal

simple
complex
Complementing an ‘Ordinary’ Introduction to Software Engineering
Complementing an ‘Ordinary’ Introduction to Software Engineering

- Introduction
- Software Process Management
- Requirements Engineering
- Architecture & Design
- Software Quality Assurance

Informal, Semi-formal, Formal
Complementing an ‘Ordinary’ Introduction to Software Engineering

- Introduction
- Software Process Management
- Requirements Engineering
- Architecture & Design
- Software Quality Assurance

- Process modelling
- Use cases, scenarios
- Structural software modelling
- Testing
- Behavioural software modelling

- Informal
- Semi-formal
- Formal

- Simple
- Complex
Complementing an ‘Ordinary’ Introduction to Software Engineering

- **Introduction**
- **Software Process Management**
- **Requirements Engineering**
- **Architecture & Design**
- **Software Quality Assurance**

- **Process Modelling**
  - informal
  - semi-formal
  - formal

- **Use Cases, Scenarios**
  - informal
  - semi-formal
  - formal

- **Structural Software Modelling**
  - formal

- **Behavioural Software Modelling**
  - formal

- **Testing**
  - formal
Complementing an ‘Ordinary’ Introduction to Software Engineering

- **Introduction**
  - Software Process Management
- **Requirements Engineering**
  - Use cases, scenarios (formal)
  - Structural software modelling (formal)
  - Testing
- **Architecture & Design**
  - Behavioural software modelling (formal)
- **Software Quality Assurance**
  - Program verification (formal)

**Process Modelling**
- Informal
- Semi-formal
- Formal

**Complexity Levels**
- Simple
- Complex
Progression

semi-formal → concrete syntax
Progression

- semi-formal → concrete syntax

principles of formal methods
(formal semantics, formalisation, validation, formal analysis, interpretation of results)

<table>
<thead>
<tr>
<th>7: room ventilation</th>
<th>r1</th>
<th>r2</th>
<th>r3</th>
</tr>
</thead>
<tbody>
<tr>
<td>off button pressed?</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>off ventilation off?</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>on ventilation on?</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>go start ventilation</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>stop stop ventilation</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
Progression

- semi-formal → concrete syntax

- principles of formal methods (formal semantics, formalisation, validation, formal analysis, interpretation of results)

- complex concrete and abstract syntax; complex semantics

### Progression Diagram

- **7: Room Ventilation**
  - button pressed?
  - ventilation on?
  - start ventilation
  - stop ventilation

- **Customer’s Requirements**
  - complete
  - incomplete

- **DT (formally)**
  - true positive
  - false positive
  - true negative
  - false negative

- **LSC: Get Change**
  - true
  - invariant
  - permissive

### Diagram Elements

- User
- Vendor
- Change
- Stop
- On
- Off

### Graphical Representation

- Informal
- Semi-formal
- Formal
- Simple
- Complex
Progression

- **semi-formal** → **concrete syntax**

**principles of formal methods**
- formal semantics, formalisation, validation, formal analysis, interpretation of results

- **complex concrete and abstract syntax**; complex semantics

- **model; less complex syntax and semantics, focus on complex modelling**

---

<table>
<thead>
<tr>
<th>F: room ventilation</th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>button pressed?</td>
<td>☑</td>
<td></td>
</tr>
<tr>
<td>ventilation off?</td>
<td></td>
<td>☑</td>
</tr>
<tr>
<td>ventilation on?</td>
<td>☑</td>
<td></td>
</tr>
<tr>
<td>start ventilation</td>
<td></td>
<td>☑</td>
</tr>
<tr>
<td>stop ventilation</td>
<td>☑</td>
<td></td>
</tr>
</tbody>
</table>

- **customers' requirements**
  - complete
  - incomplete

- **false positive**
- **true positive**
- **false negative**
- **true negative**

- **customer's requirements**
  - complete
  - incomplete

- **DT (formally)**
  - incomplete
  - complete

- **INFORMAL**

- **semi-formal**

- **formal**

- **complex**

- **simple**
Progression

- **Principles of formal methods**
  - Formal semantics, formalisation, validation, formal analysis, interpretation of results

- **Complex concrete and abstract syntax**
  - Complex concrete and abstract syntax; complex semantics

- **Deductive program verification**
  - Formal model; less complex syntax and semantics, focus on complex modelling

<table>
<thead>
<tr>
<th>F: room ventilation</th>
<th>P</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>button pressed?</td>
<td>off</td>
<td>on</td>
<td>off</td>
</tr>
<tr>
<td>ventilation off?</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>ventilation on?</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>start ventilation</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>stop ventilation</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

- **Customer's requirements**
  - DT: False positive, true negative
  - CT: False negative, true positive

- **Logic verification**
  - \( \forall d_1 \in \text{allInstances}_D \)
  - \( c(d_1) = c(d_2) \implies a(d_1) = a(d_2) \)

- **Informal vs. formal**
  - Informal: semi-formal, complex
  - Semi-formal: complex concrete and abstract syntax, complex semantics
  - Formal: simple, complex

- **Example**
  - Deductive program verification
  - While \( b \geq y \) do
    - \( P \land b \geq y \)
    - \( b := b - y \)
    - \((a + 1) \cdot y + b = x \land b \geq 0 \)
    - \( a := a + 1 \)
    - \( a \cdot y + b = x \land b \geq 0 \)
  - End
  - \{ \( P \land \neg (b \geq y) \) \}

- **Buttons and Ventilation Model**
  - User presses button:
    - Ventilation off:
      - Start ventilation
    - Ventilation on:
      - Stop ventilation
• Motivated a need for Formal Methods in introductions to Software Engineering

• Presented Complementation Approach

• Proposed Progression

• In the paper:
  • Details of the motivation, related work.
  • Definition of learning objectives.
  • Details of the progression.
  • Experience from five seasons of teaching an implementation of this course design:
    No indications of student over-strain (neither time, nor level.)