Formale Methoden in der Softwaretechnik-Vorlesung

SEUH 2019 — 21./22. Februar — Bremerhaven

Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany

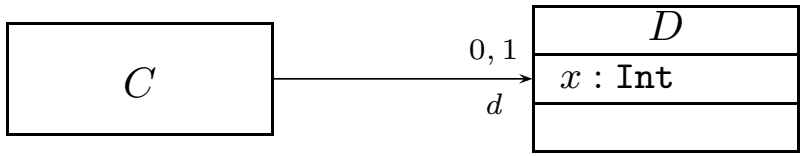
Formal Methods: Definition

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

Definition.
We call a software description language **formal** if and only if it has

- a (possibly graphical) **concrete syntax**,
- a precisely defined **abstract syntax**, and each syntactically correct concrete representation has a well-defined abstract syntax,
- a precisely defined **semantics**.
  that maps each abstract syntax representation to its meaning.

Example: Class Diagrams as Formal Description Language



concrete syntax

abstract syntax

basic types *classes* *attributes* *attribute mapping*

(Object System) **Signature** $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$

Example: $\mathscr{S} = (\{\texttt{Int}\}, \{C, D\}, \{x : \texttt{Int}, d : D_{0,1}\}, C \mapsto \{d\}, D \mapsto \{x\})$

semantics

object identities *attributes* *basic type values* *links (sets of identities)*

System state of $\mathscr{S}$ (wrt. structure $\mathscr{D}$)

$$\sigma : \mathscr{D}(\mathscr{C}) \nrightarrow (V \nrightarrow (\mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}_*))).$$

Example: $\sigma = \{ \quad 1_C \mapsto \{d \mapsto \{27_D\}\}, \quad 27_D \mapsto \{x \mapsto 13\} \quad \}$

OMG (2006)

Motivation ("School")

Lehmann and Buth (2015): (as example for incoherent constructive alignment)

> "'**Es darf nicht erwartet werden**, dass die Studierenden UML Modelle erstellen können,
> wenn in der VL **nur die einzelnen Symbole durchgearbeitet werden**
> und dann auf dieser Basis in der Prüfung **Analyseaufgaben** auf einem gegebenen Diagramm gestellt werden.'"

Task. Is there a system state such that the OCL contraint

$$\text{context } C \text{ inv} : d \,.\, x = 13$$

evaluates to $\bot$? Give a proof for your answer.

With **formal** syntax/semantics, we can **meaningfully** state such **exercise/exam tasks**.

Motivation ("Life")

For about 20 years now,
there is an **increasing** industry **interest** and **use** (!) of formal methods:

- $\sim$ 20 years ago: '"**the usual suspects**"'
  - Airbus, ESA, …
- $\sim$ 5–10 years ago: **large- and medium-sized**
  - Daimler, Audi, BOSCH, GE, Microsoft, facebook, Amazon, …
  - SICK AG (software part: medium to large)
- $\sim$ 10–0 years ago: **small-sized**
  - Arenis et al. (2016): SeCa (small)

$\rightarrow$ **particular formal** methods solve **particular**, practical problems.

- **Tomorrow's software people** may be confronted with some formal methods.
  (Claim: if not, no harm is done.)

  To **use formal methods effectively**, they should know the concepts behind it
  and how to interpret the outcome of analyses.

Outline of the Talk

- Definition Formal Methods ✔
- Motivation ✔
- Teaching Goals
  - Overall Course Goals
  - Formal Methods-specific Goals
- Implementation
  - Course and Students Situation
  - Course Construction
  - Topic Area Structure
  - Example: Requirements Engineering
- Evaluation
- Conclusion

Teaching Goals

Overall, colloquially put,

- "Be able to **effectively talk to** Software Engineering people."
- "No (big) surprises in the **established textbooks**."

  (Sommerville (2010); Balzert (2009); Ludewig and Lichter (2013) etc.)

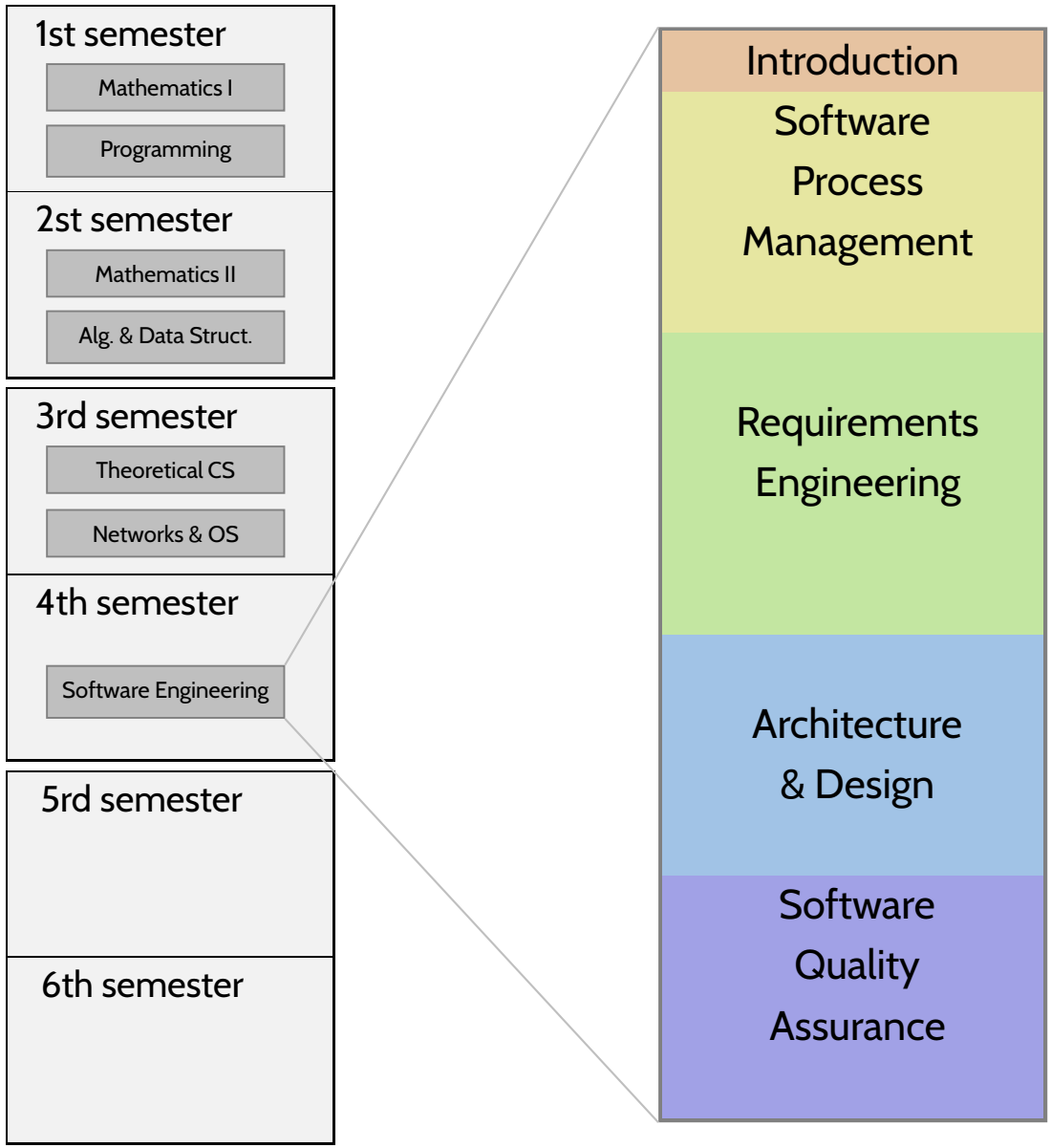
In particular **awareness** for and **understanding** of problems and difficulties in principle, such as **misunderstandings**, **discovering errors late**, etc.

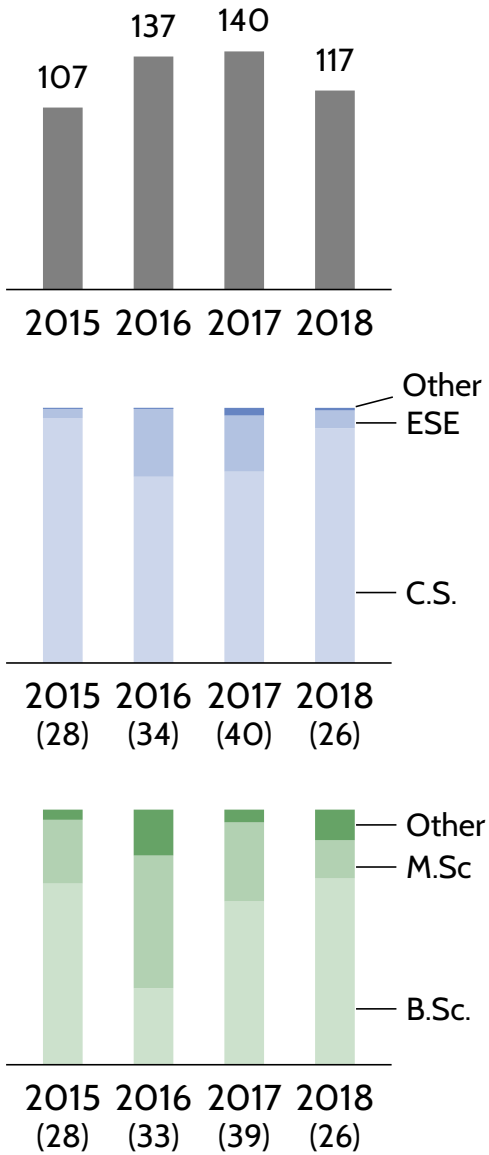
Formal Methods-specific, be able to

- solve SWE tasks **using the discussed formalisms**,
- correctly **interpret** of fomal analysis **outcomes** and take appropriate actions,
- be able to **assess the costs and benefits** of different formal techniques.

Implementation of the Course

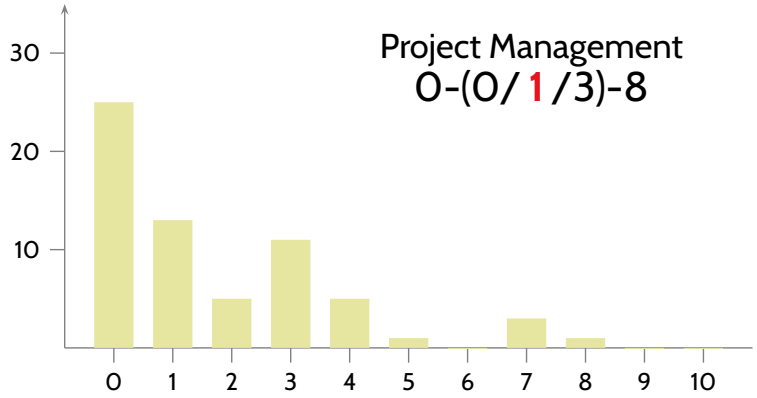
Course Situation

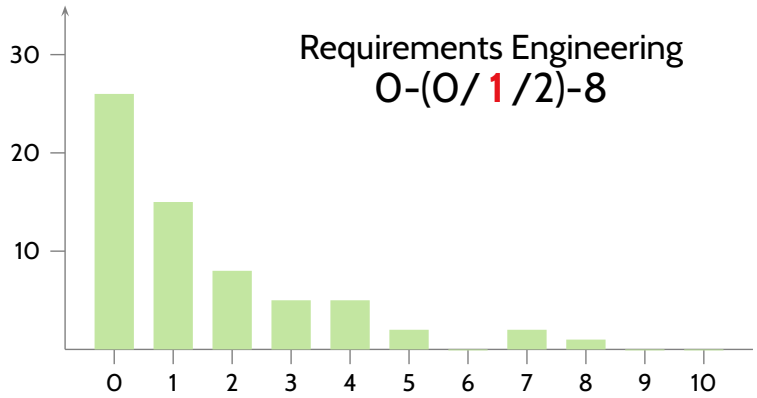


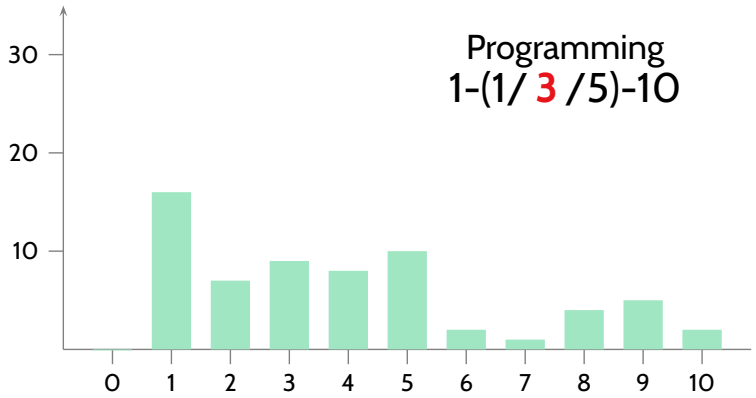


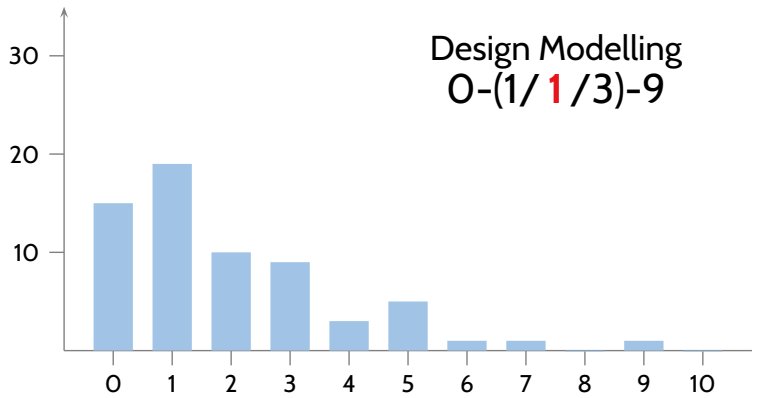
Students' Situation (2018)

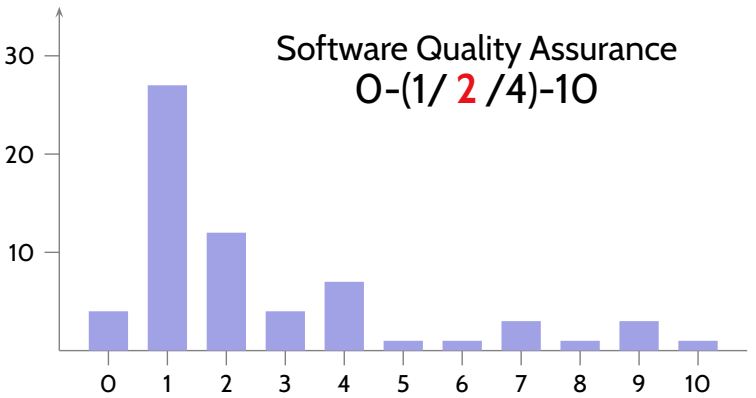
0: No experience whatsoever; not taken any related subjects.
1: Only performed the activity in the context of a lecture or course.
10: Responsibly performed activity in (100+ users) / (36+ PM) / commercial purpose project.



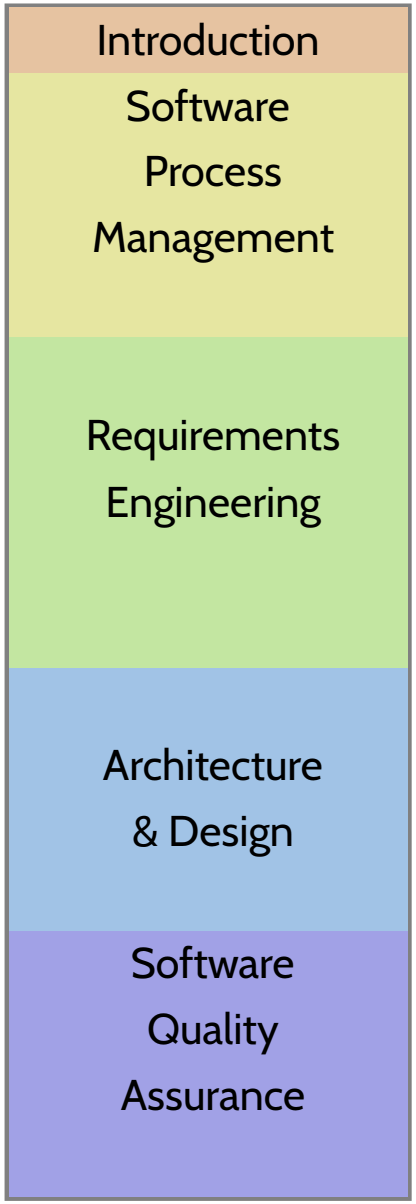








Course Construction

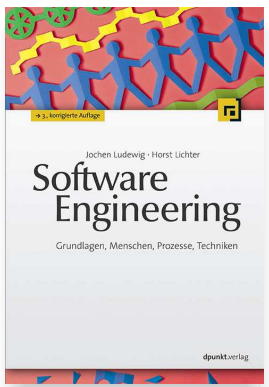


Lect.	Topic Area	Content Overview
1	Intro.	Software; Software Engineering; Successful software development; Construction of the course.
2 – 5	Proj. Mgmt.	Software metrics; Cost estimation; Project, process modelling (**semi-formal**); Procedure & Process models.
6 – 10	Req. Eng.	Requirements, Requirements Analysis; Properties (completeness, consistency, …); Requirements specification languages: • Natural language patterns, • Decision Tables (**formal**), • Use Cases and Diagrams (**semi-formal**), • LSCs (**formal**).
11 – 14	Arch. & Des.	Model; Views; Modelling Structure: • Class & Object Diagrams (**formal**), • Proto-OCL (**formal**), Principles of Design; Design Patterns; Modelling software behaviour: • Communicating Finite Automata (**formal**), • Query Language (**formal**), an outlook on UML state machines.
15 – 18	QA	Test case (**formal**), test execution; true and false positive/negative outcomes; Limits of software testing; Program verification (**formal**); Code Review.

Course Construction

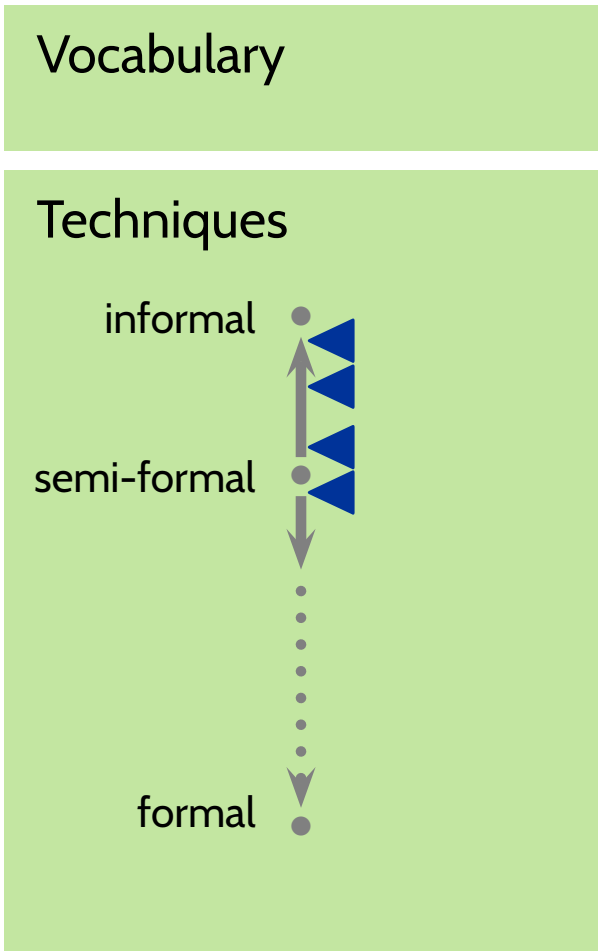
- Introduction
- Software Process Management
- Requirements Engineering
- Architecture & Design
- Software Quality Assurance

Lect.	Topic Area	Content Overview
1	Intro.	Software; Software Engineering; Successful software development; Construction of the course.
2 – 5	Proj. Mgmt.	Software metrics; Cost estimation; Project, process modelling (**semi-formal**); Procedure & Process models.
6 – 10	Req. Eng.	Requirements, Requirements Analysis; Properties (completeness, consistency, …); Requirements specification languages: • Natural language patterns, • Decision Tables (**formal**), • Use Cases and Diagrams (**semi-formal**), • LSCs (**formal**).
11 – 14	Arch. & Des.	Model; Views; Modelling Structure: • Class & Object Diagrams (**formal**), • Proto-OCL (**formal**), Principles of Design; Design Patterns; Modelling software behaviour: • Communicating Finite Automata (**formal**), • Query Language (**formal**), an outlook on UML state machines.
15 – 18	QA	Test case (**formal**), test execution; true and false positive/negative outcomes; Limits of software testing; Program verification (**formal**); Code Review.



+ **Formal Methods**

Structure of Topic Areas: Example RE

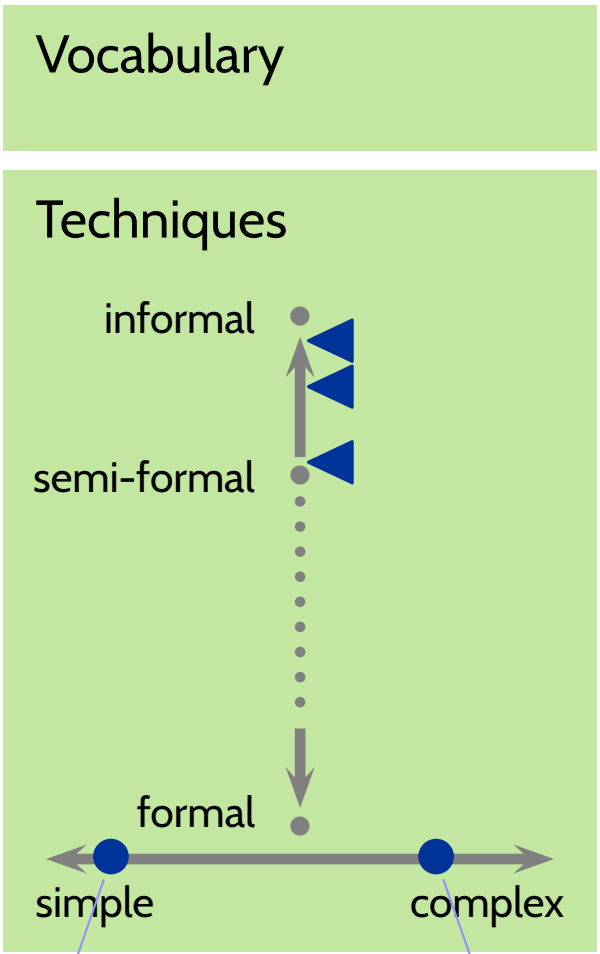


e.g. Natural Language Patterns

e.g. Use Cases

e.g. Use Case Diagrams

e.g. Decision Tables



Sommerville (2010),
Balzert (2009),
Ludewig and Lichter (2013)
etc.

e.g. Decision Tables (**fomal**)

e.g. Live Sequence Charts (proper subset, **fomal**)

Example:

Requirements Engineering / Decision Tables

Decision Table Semantics: Example

$$\mathcal{F}(r) := F(v_1, c_1) \wedge \cdots \wedge F(v_m, c_m) \wedge F(v_1, a_1) \wedge \cdots \wedge F(v_k, a_k)$$

$$F(v, x) = \begin{cases} x & \text{, if } v = \times \\ \neg x & \text{, if } v = - \\ \textit{true} & \text{, if } v = * \end{cases}$$

T		r_1	r_2	r_3
c_1		$\times$	$\times$	$-$
c_2		$\times$	$-$	$*$
c_3		$-$	$\times$	$*$
a_1		$\times$	$-$	$-$
a_2		$-$	$\times$	$-$

- $\mathcal{F}(r_1) = F(\times, c_1) \wedge F(\times, c_2) \wedge F(-, c_3) \wedge F(\times, a_1) \wedge F(-, a_2)$
  $= c_1 \wedge c_2 \wedge \neg c_3 \wedge a_1 \wedge \neg a_2$
- $\mathcal{F}(r_2) = c_1 \wedge \neg c_2 \wedge c_3 \wedge \neg a_1 \wedge a_2$
- $\mathcal{F}(r_3) = \neg c_1 \wedge \textit{true} \wedge \textit{true} \wedge \neg a_1 \wedge \neg a_2$

Property: (Formal) Completeness

Completeness

Definition. [*Completeness*] A decision table T is called **complete** if and only if the disjunction of all rules' premises is a tautology, i.e. if

$$\models \bigvee_{r \in T} \mathcal{F}_{pre}(r).$$

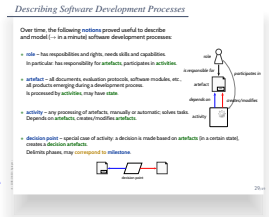
	Incomplete	Complete
(Formally) Incomplete	true positive	false positive
(Formally) Complete	false negative	true negative

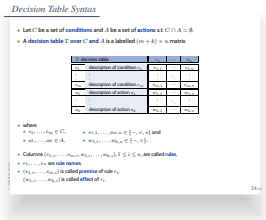
All issues, except for "RE engineer's DT is erroneously complete" will be detected and can be resolved. → Risk mitigation.

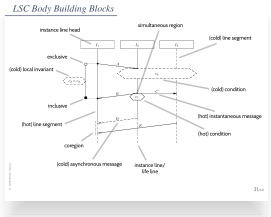
18/48

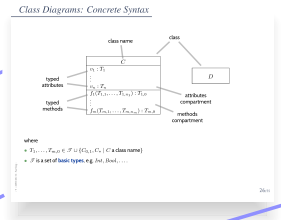
Formal Content in the Course

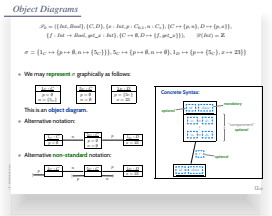
Introduction		L 1
Scales, Metrics, Costs,		L 2
		L 3
		T 1
Development Process	●	L 4
		L 5
Requirements Engineering		L 6
	●	L 7
		T 2
		L 8
	●	L 9
		T 3
		L 10
Arch. & Design, Software-	●	L 11
	●	L 12
		T 4
Modelling, Patterns	●	L 13
	●	L 14
QA	●	L 15
		T 5
(Testing, Formal Verification)	●	L 16
		L 17
Wrap-Up		L 18
		T 6

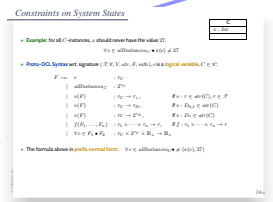


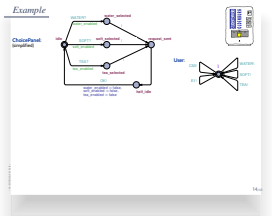


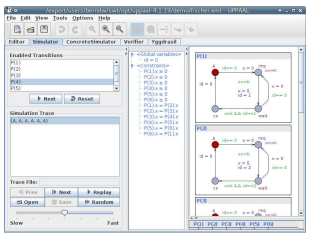




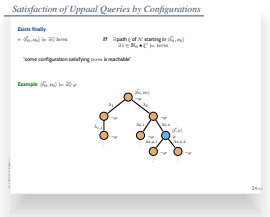


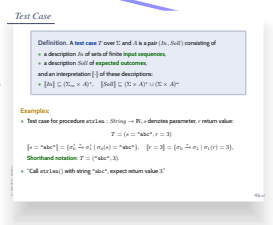


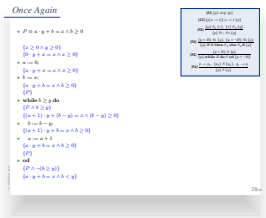


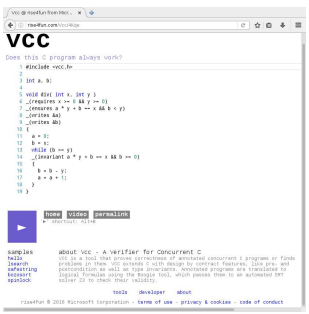


Tool: Uppaal









Tool: VCC

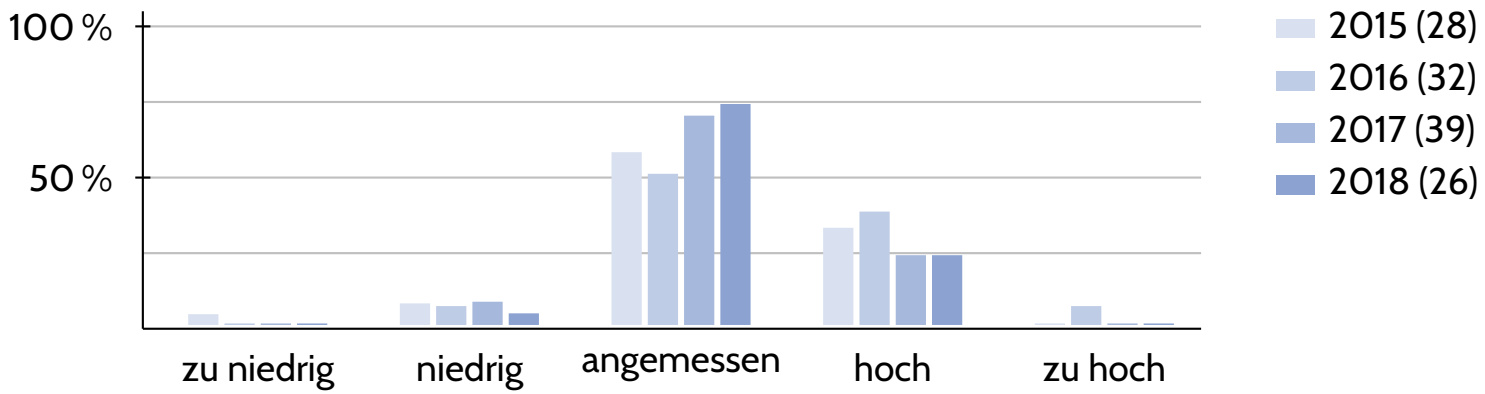
Evaluation

Evaluation: Exam Results (Requirements Engineering)

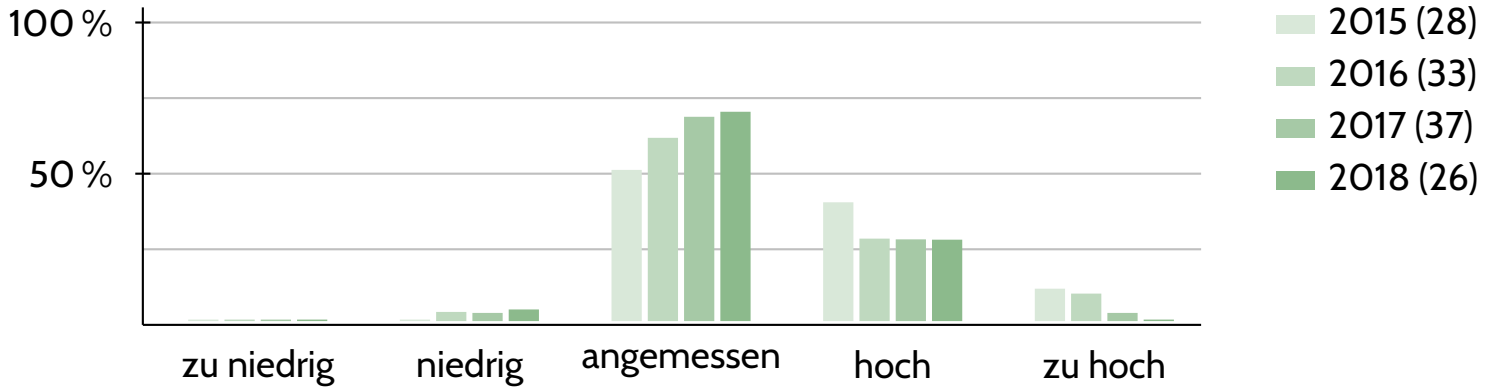
Lehrevaluation: Niveau and Workload

Lehrevaluation: Niveau and Workload

Das inhaltliche Niveau der Veranstaltung ist:



Verglichen mit den vergebenen Leistungspunkten
ist mein tatsächlicher Arbeitsaufwand für diese Lehrveranstaltung:



(Uttl et al. (2017): Student evaluation vs. learning)

Conclusion

Conclusion

Our approach to extend the SWE course by formal methods:

- for the fundamental concepts:
  one as-small-as-possible formal description language
- reduce CD, OCL, Statecharts, Formal Verification etc.
  to **essential cores**.
- for each formalism and technique, show the **problem domain** they are **good at** (real world-inspired examples).

works – with

- very decent exam results,
- not overstraining students.

Experienced Benefits:

- Crystal clear **technical** exercise and exam tasks.
  $\rightarrow$ Free cognitive capacities for the **creative** work.
- Common ground for, e.g., preliminary thesis discussion.
- **Future work**: Automatic generation of tasks.

References

Arenis, S. F., Westphal, B., et al. (2016). Ready for testing: ensuring conformance to industrial standards through formal verification. *FAoC*, 28(3):499–527.

Balzert, H. (2009). *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. Spektrum, 3rd edition.

Bjørner, D. and Havelund, K. (2014). 40 years of formal methods. talk.

Lehmann, T. and Buth, B. (2015). Lecture engineering. In Schmolitzky, A. et al., editors, *SEUH*, volume 1332, pages 103–109. CEUR-WS.

Ludewig, J. and Lichter, H. (2013). *Software Engineering*. dpunkt, 3rd edition.

OMG (2006). Object Constraint Language, version 2.0. Technical Report formal/06-05-01.

Sommerville, I. (2010). *Software Engineering*. Pearson, 9th edition.

Uttl, B. et al. (2017). Meta-analysis of faculty's teaching effectiveness: Student evaluation of teaching ratings and student learning are not related. *Stud. Educat. Eval.*, 54:22 – 42.