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 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**.

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.

2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)

Catching up with Method and Process Practice: An Industry-Informed Baseline for Researchers

Jil Klünder*, Regina Hebig†, Paolo Tell‡, Marco Kuhrmann§, Joyce Nakatumba-Nabende¶, Rogardt Heldal†, Stephan Krusche**, Masud Fazal-Baqaie††, Michael Felderer‡‡, Marcela Fabiana Genero Bocco[x], Steffen Küpper‡, Sherlock A. Licorish[xi], Gustavo López[xii], Fergal McCaffery[xiii], Özden Özcan Top[xiii], Christian R. Prause[xiv], Rafael Prikladnicki[xv], Eray Tüzün[xvi], Dietmar Pfahl[xvii], Kurt Schneider* and Stephen G. MacDonell[xviii]

*Leibniz University Hannover, Germany, Email: {jil.kluender, kurt.schneider}@inf.uni-hannover.de; †Chalmers | University of Gothenburg, Sweden, Email: {regina.hebig, heldal}@cse.gu.se, ‡IT University Copenhagen, Denmark, Email: pate@itu.dk, §Clausthal University of Technology, Germany, Email: {marco.kuhrmann, steffen.kuepper}@tu-clausthal.de, ¶Makerere University, Uganda, Email: jnakatumba@cis.mak.ac.ug, **Technical University of Munich, Germany, Email: krusche@in.tum.de, ††Fraunhofer IEM, Germany, Email: masud.fazal-baqaie@iem.fraunhofer.de, ‡‡University of Innsbruck, Austria, Email: michael.felderer@uibk.ac.at, [x]University of Castilla-La Mancha, Spain, Email: marcela.genero@uclm.es, [xi]University of Otago, New Zealand, Email: sherlock.licorish@otago.ac.nz, [xii]University of Costa Rica, Costa Rica, Email: gustavo.lopez_h@ucr.ac.cr, [xiii]Dundalk Institute of Technology & Lero, Ireland, Email: {fergal.mccaffery, ozden.ozcantop}@dkit.ie, [xiv]DLR Space Administration, Germany, Email: christian.prause@dlr.de, [xv]Pontifícia Universidade Católica do Rio Grande do Sul, Brazil, Email: rafael.prikladnicki@pucrs.br, [xvi]Bilkent University, Turkey, Email: eraytuzun@cs.bilkent.edu.tr, [xvii]University of Tartu, Estonia, Email: dietmar.pfahl@ut.ee, [xviii]Auckland University of Technology, New Zealand, Email: smacdone@aut.ac.nz

***Abstract*—Software development methods are usually not applied by the book. Companies are under pressure to continuously deploy software products that meet market needs and stakeholders' requests. To implement efficient and effective development processes, companies utilize multiple frameworks, methods and practices, and combine these into hybrid methods. A common combination contains a rich management framework to organize and steer projects complemented with a number of smaller practices providing the development teams with tools to complete their tasks. In this paper, based on 732 data points collected through an international survey, we study the software development process use in practice. Our results show that 76.8% of the companies implement hybrid methods. Company size as well as the strategy in devising and evolving hybrid methods affect the suitability of the chosen process to reach company or project goals. Our findings show that companies that combine planned improvement programs with process evolution can increase their process' suitability by up to 5%.**

***Index Terms*—software development, software process, hybrid methods, survey research**

I. Introduction

For decades, software companies, teams, and even individual developers have sought approaches that enable efficient and effective software development. Since the 1970's, numerous processes have been proposed. The community started with the Waterfall model [1], then the Spiral model [2], followed by agile methods and lean development approaches [3]. Since the early 2000s, few innovative software development approaches were proposed, but several proposals for scaling agile methods, e.g., SAFe or LeSS, were published. Meanwhile, an increasing number of studies showing that modern software development is neither purely "traditional" nor "agile" can be found reflecting that companies use processes comprised of various development practices [4], [5].

Problem Statement: Research that focuses on agile methods and practices only cannot support practitioners who are faced with the reality of *hybrid development methods*. Similarly, the 100+ tailoring criteria [6], [7] for processes established in the last decade seem to have no relevance for practitioners who are devising hybrid methods and seeking immediate and practical solutions to solve short-term problems. Thus, process-related research has lost momentum as it is no longer aligned with the concerns of practice.

Objective: In response to the situation above, our objective is *to understand how and why practitioners devise hybrid development methods*. Our goal is to set a new baseline for the next decade of evidence-based research on software development approaches driven by practice.

Contribution: Based on an online survey comprising 732 data points we study the use of hybrid methods and the factors influencing the suitability of development approaches for reaching goals. According to our results, 3/4 of the companies use a hybrid method, and company size and strategies to devise hybrid methods influence the suitability of the approach to achieve defined goals.

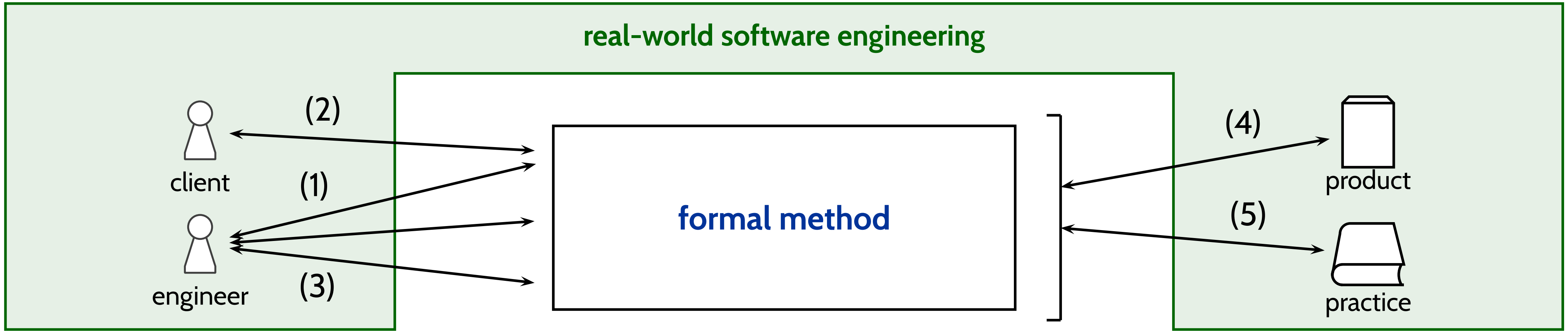
Context: This research is based on the HELENA study[1], which is a large-scale international survey in which 75 researchers and practitioners from 25 countries participated. We

[1] HELENA: Hybrid dEveLopmENt Approaches in software systems development, online: https://helenastudy.wordpress.com.

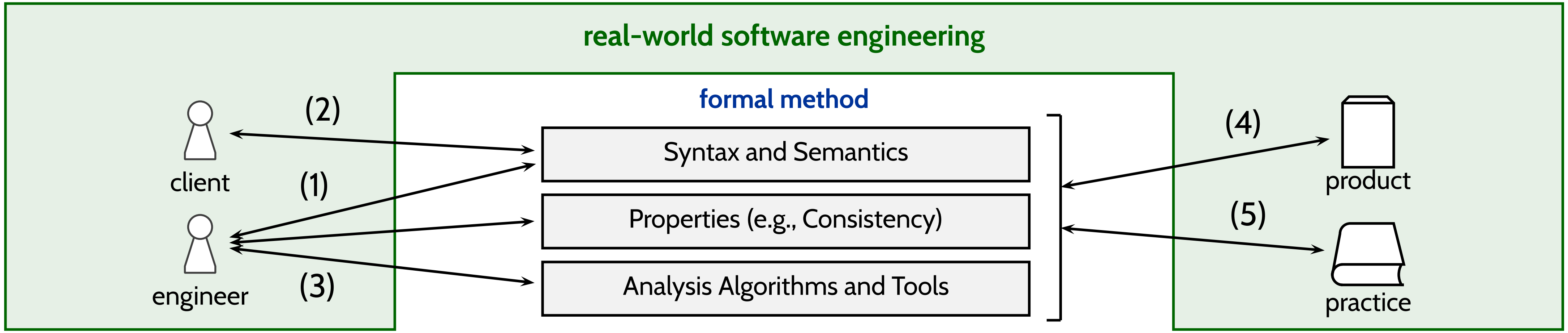
978-1-7281-1760-7/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSE-SEIP.2019.00036

(Klünder et al., 2019)

Formal Methods in the Context of Software Engineering



Formal Methods in the Context of Software Engineering



real-world software engineering

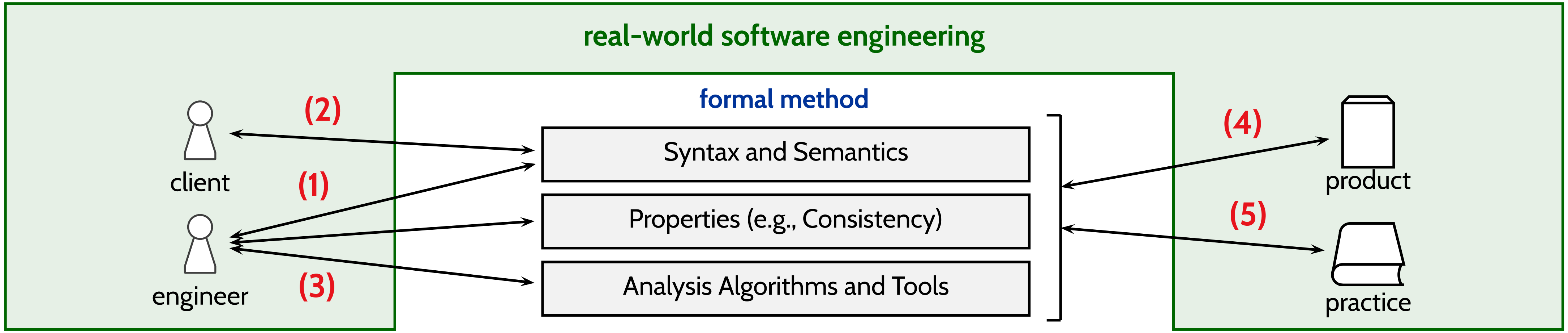
formal method

Syntax and Semantics

Properties (e.g., Consistency)

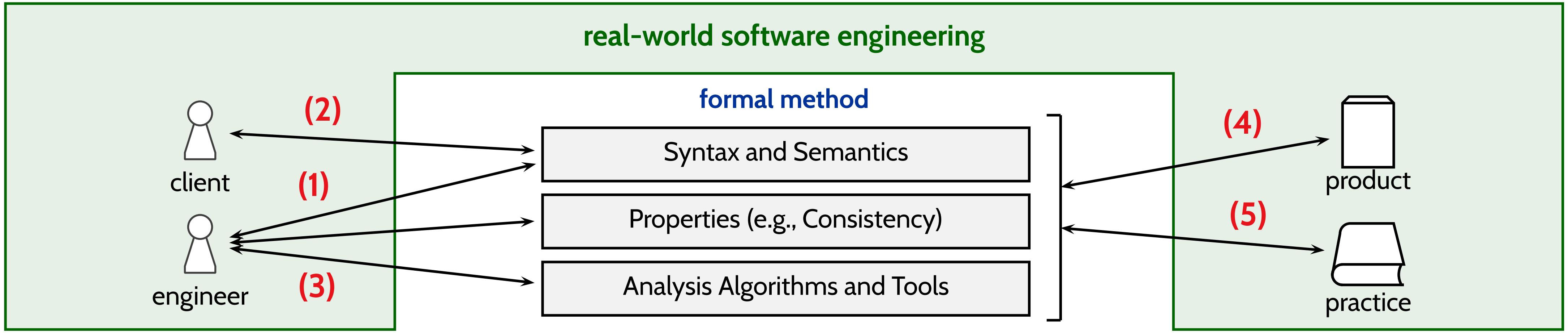
Analysis Algorithms and Tools

Formal Methods in the Context of Software Engineering


real-world software engineering
formal method
client
engineer
(1)
(2)
(3)
Syntax and Semantics
Properties (e.g., Consistency)
Analysis Algorithms and Tools
(4)
(5)
product
practice

(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

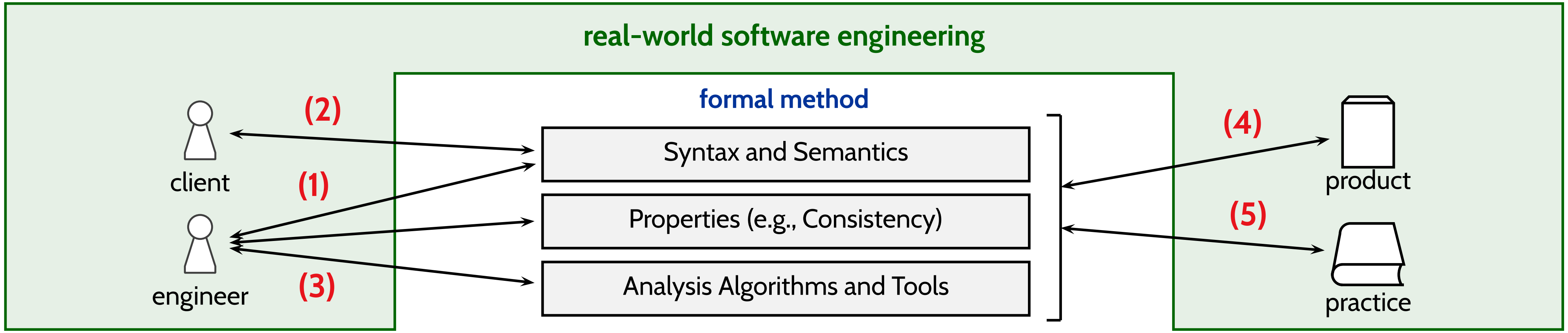


real-world software engineering
formal method
client
engineer
(1)
(2)
(3)
(4)
(5)
Syntax and Semantics
Properties (e.g., Consistency)
Analysis Algorithms and Tools
product
practice

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

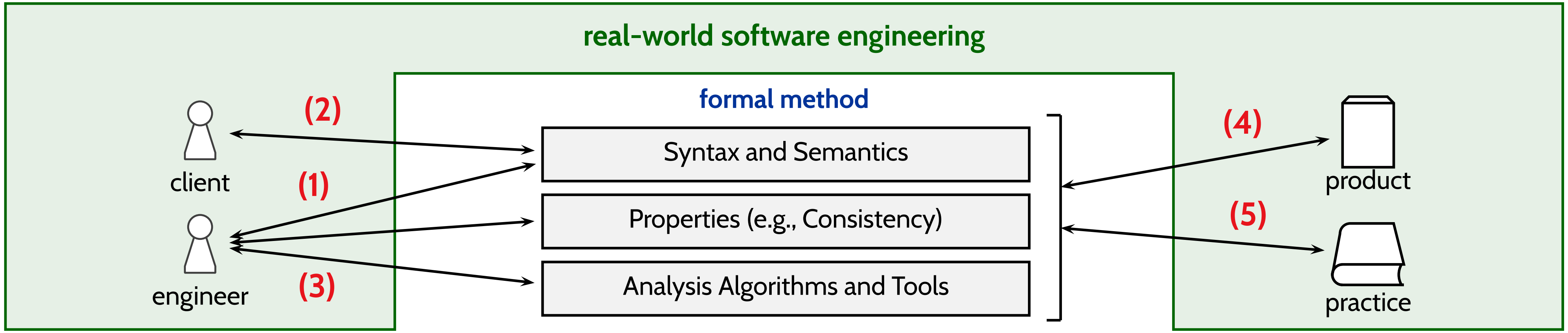
(2) engineers may need to **validate formalisations with clients**
(is the formal description a valid model of, e.g., a requirement?)

Formal Methods in the Context of Software Engineering


real-world software engineering
formal method
client
engineer
(2)
(1)
(3)
Syntax and Semantics
Properties (e.g., Consistency)
Analysis Algorithms and Tools
(4)
(5)
product
practice

- **(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**

Formal Methods in the Context of Software Engineering


real-world software engineering
formal method
Syntax and Semantics
Properties (e.g., Consistency)
Analysis Algorithms and Tools
client
engineer
product
practice
(1)
(2)
(3)
(4)
(5)

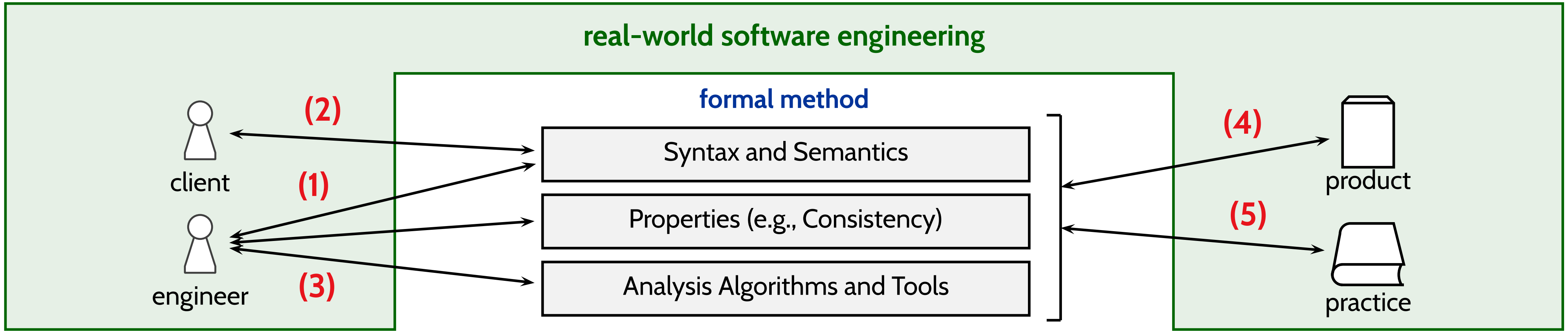
(1) engineers need to **know syntax and semantics** to formalise understandings of, e.g., a design idea

(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


real-world software engineering
formal method
client
engineer
(2)
(1)
(3)
Syntax and Semantics
Properties (e.g., Consistency)
Analysis Algorithms and Tools
(4)
(5)
product
practice

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

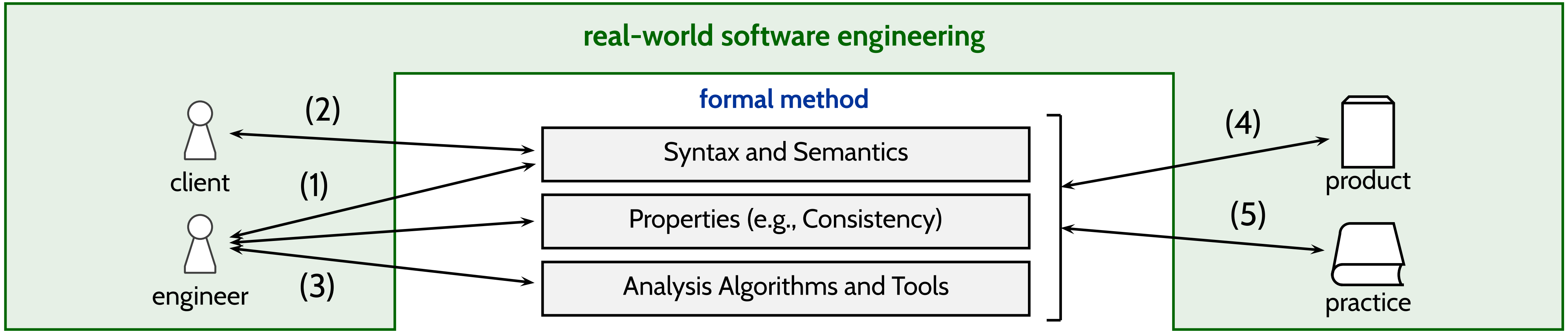
(2) engineers may need to **validate formalisations with clients** (is the formal description a valid model of, e.g., a requirement?)

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

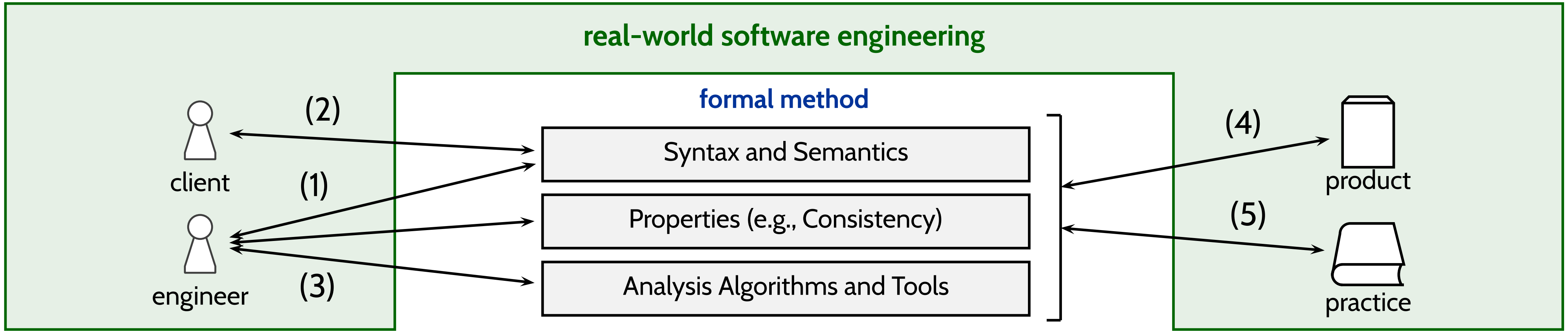
(4) outcomes of formal analyses need to be **interpreted in context**, appropriate actions need to be taken

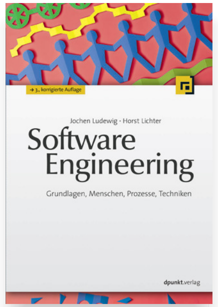
(5) techniques need to be **discussed in contemporary context**

Approach: Interpolative instead of Extrapolative

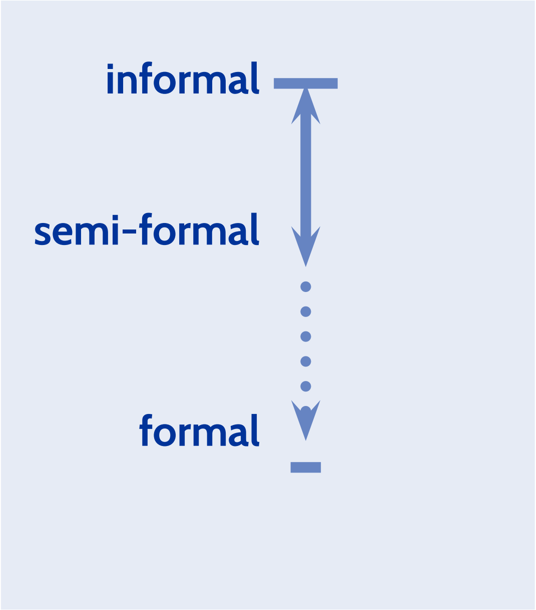

real-world software engineering
formal method
client
engineer
(1)
(2)
(3)
Syntax and Semantics
Properties (e.g., Consistency)
Analysis Algorithms and Tools
(4)
(5)
product
practice

Approach: Interpolative instead of Extrapolative

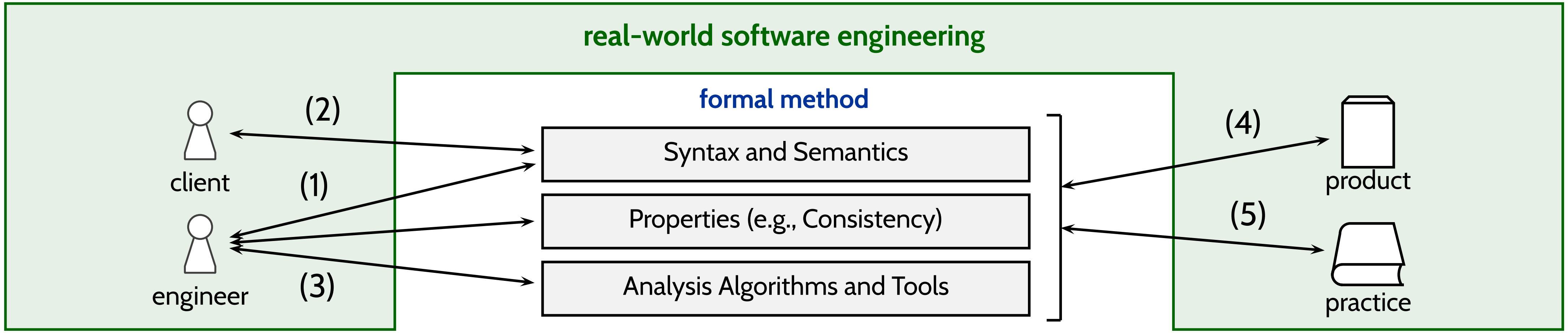

real-world software engineering
formal method
client
engineer
(1)
(2)
(3)
Syntax and Semantics
Properties (e.g., Consistency)
Analysis Algorithms and Tools
(4)
(5)
product
practice


3., korrigierte Auflage
Jochen Ludewig · Horst Lichter
Software Engineering
Grundlagen, Menschen, Prozesse, Techniken
dpunkt.verlag

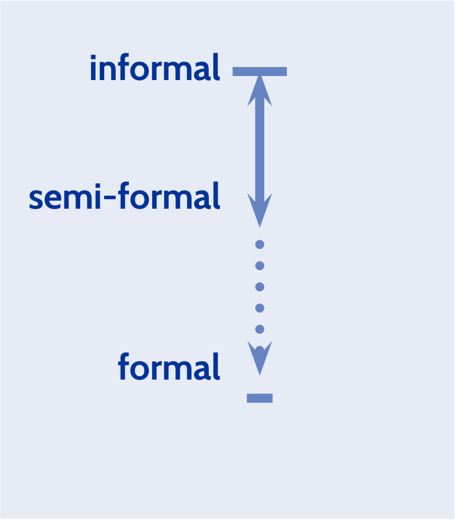
Helmut Balzert
Lehrbuch der Softwaretechnik
Basiskonzepte und Requirements Engineering
3. Auflage
Lehrbücher der Informatik
Spektrum

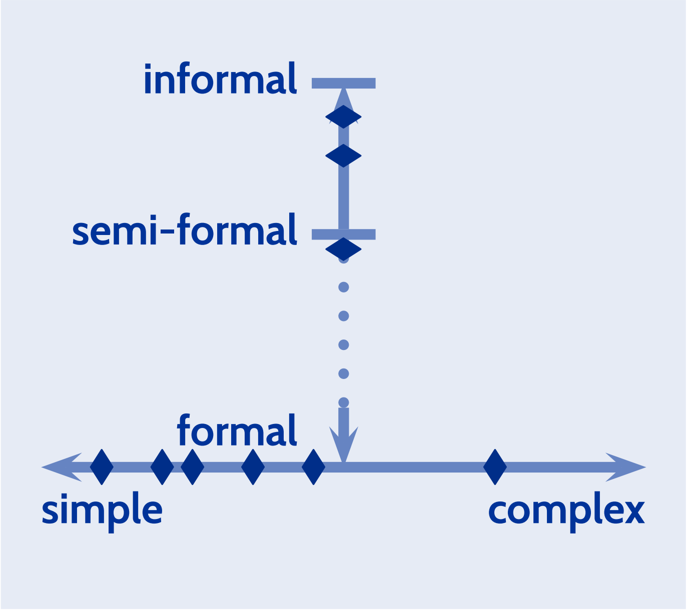


Approach: Interpolative instead of Extrapolative

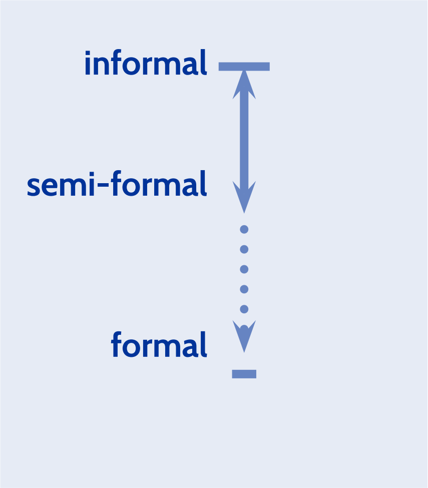

real-world software engineering
formal method
client
engineer
(1)
(2)
(3)
(4)
(5)
Syntax and Semantics
Properties (e.g., Consistency)
Analysis Algorithms and Tools
product
practice

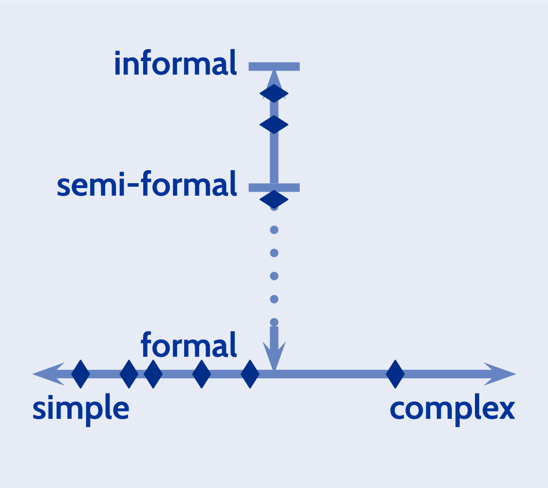
GLOBAL EDITION
Software Engineering
TENTH EDITION
Ian Sommerville
PEARSON
Jochen Ludewig · Horst Lichter
Software Engineering
Grundlagen, Menschen, Prozesse, Techniken
dpunkt.verlag
Helmut Balzert
Lehrbuch der Softwaretechnik
Basiskonzepte und Requirements Engineering
3. Auflage
Lehrbücher der Informatik
Spektrum



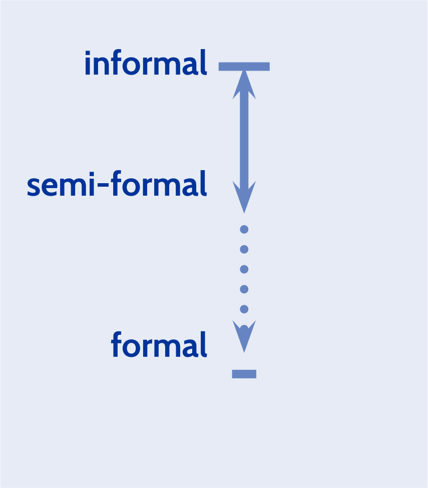


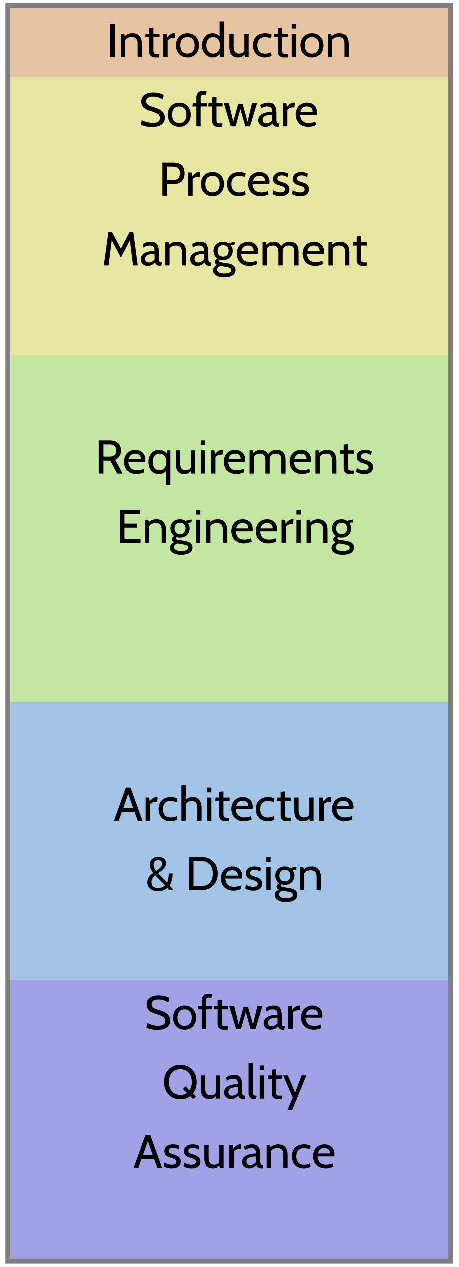
Complementing an 'Ordinary' Introduction to Software Engineering

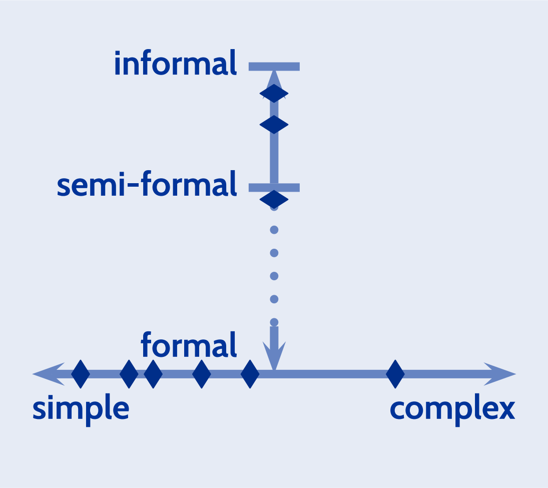




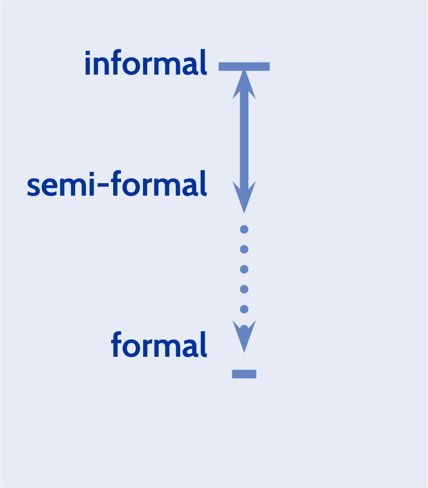
Complementing an ‘Ordinary’ Introduction to Software Engineering







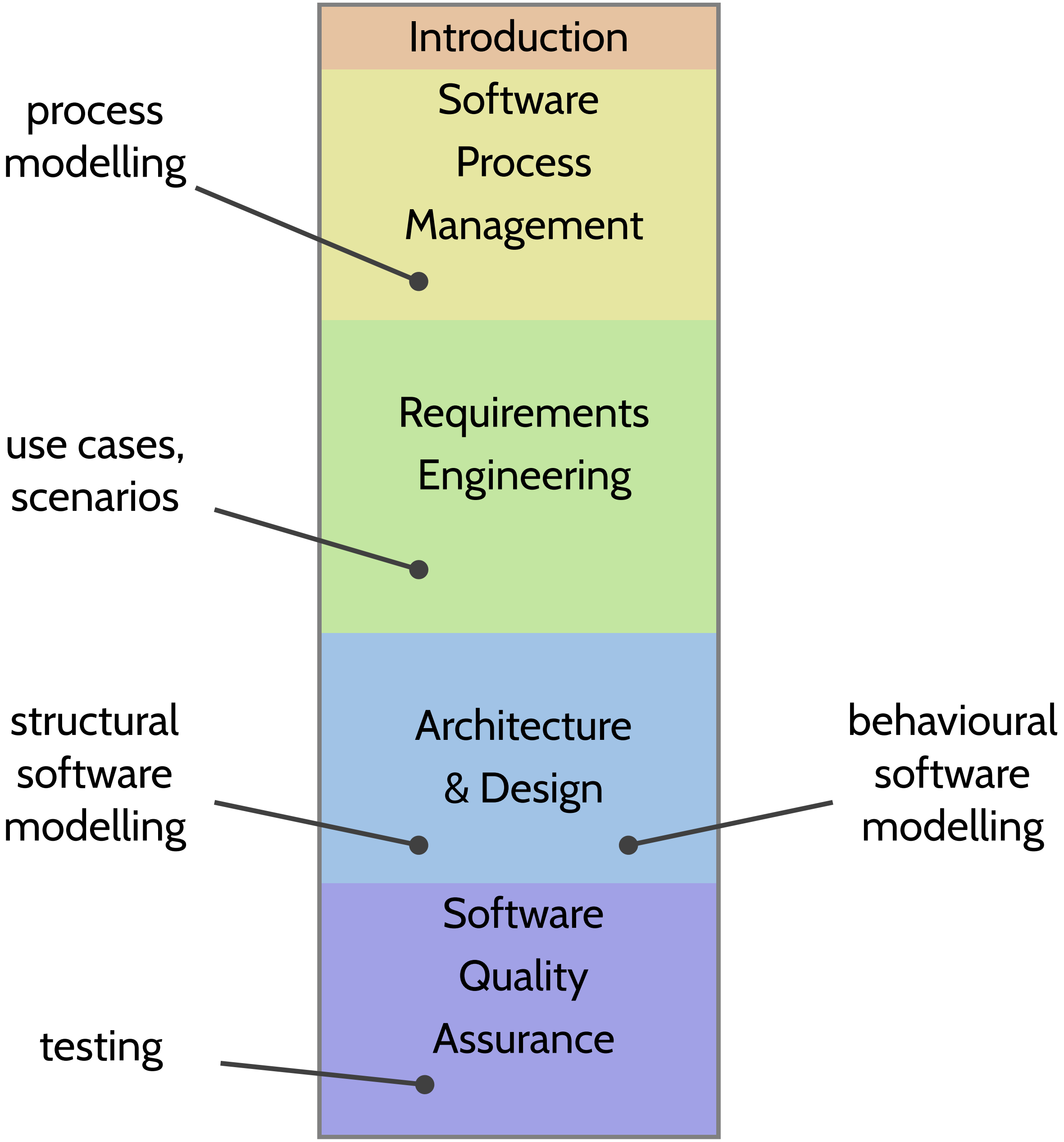
Complementing an ‘Ordinary’ Introduction to Software Engineering



informal

semi-formal

formal



Introduction

Software Process Management

Requirements Engineering

Architecture & Design

Software Quality Assurance

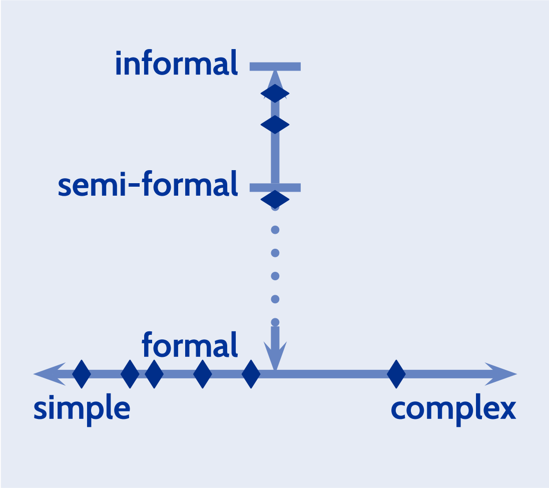
process modelling

use cases, scenarios

structural software modelling

behavioural software modelling

testing



informal

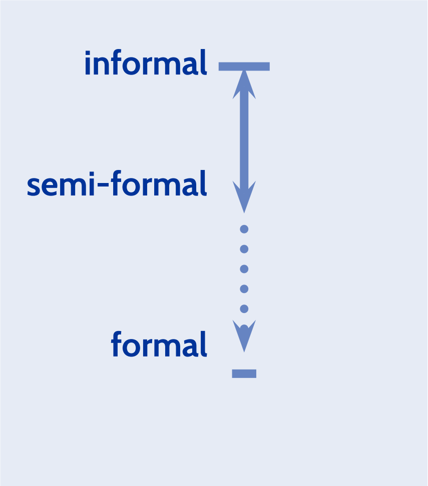
semi-formal

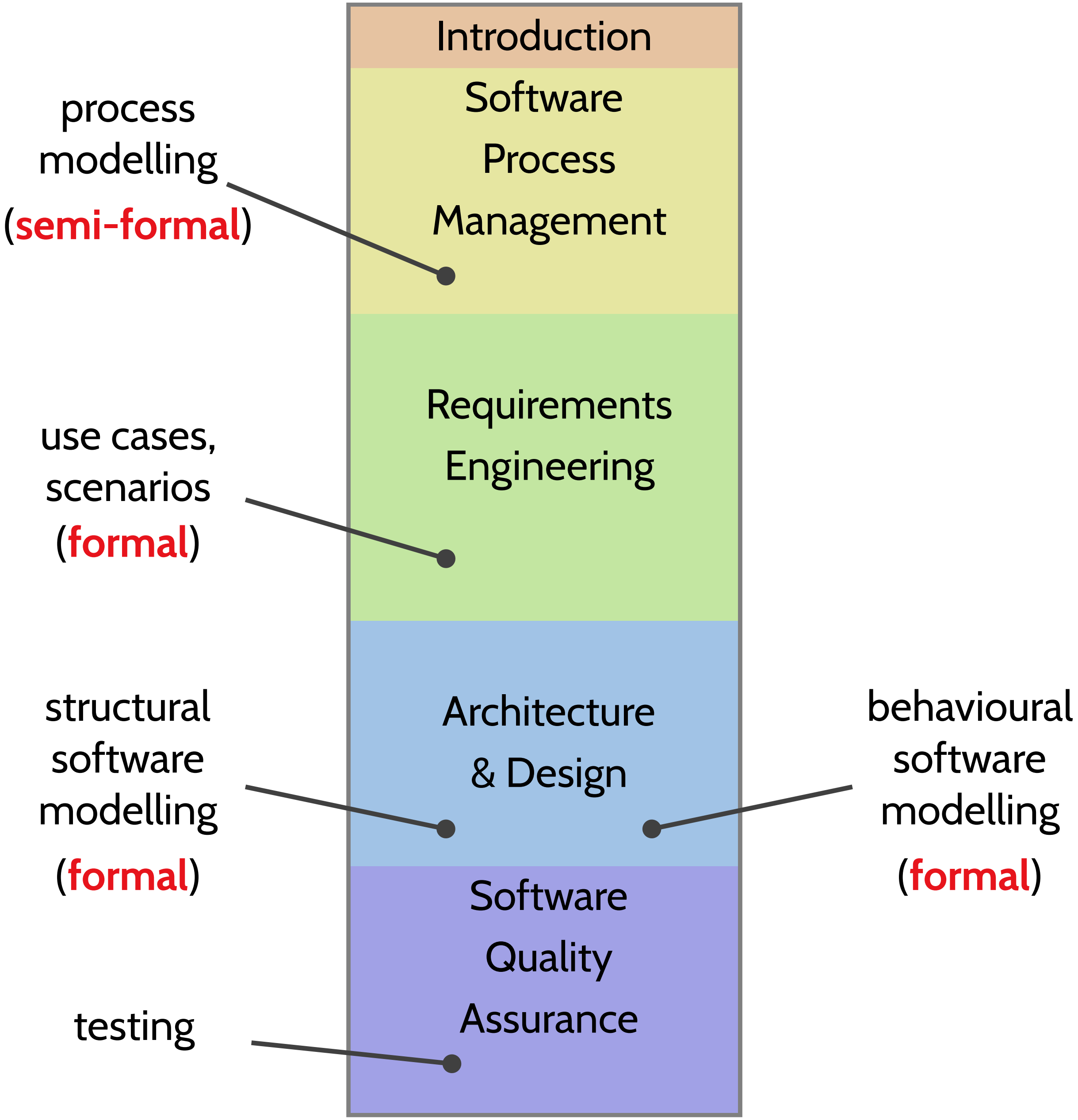
formal

simple

complex

Complementing an ‘Ordinary’ Introduction to Software Engineering




Introduction

Software Process Management

process modelling (semi-formal)

Requirements Engineering

use cases, scenarios (formal)

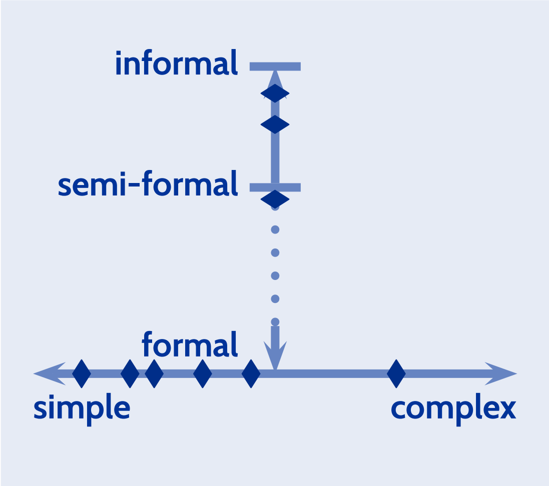
Architecture & Design

structural software modelling (formal)

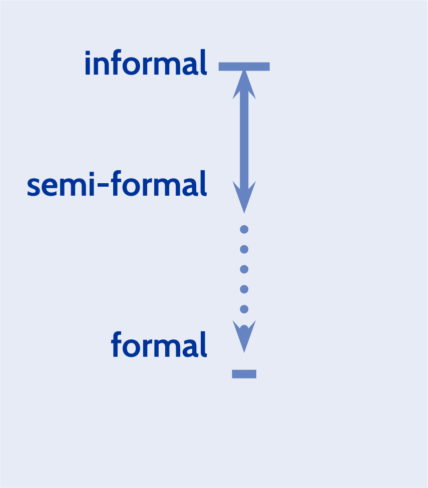
behavioural software modelling (formal)

Software Quality Assurance

testing



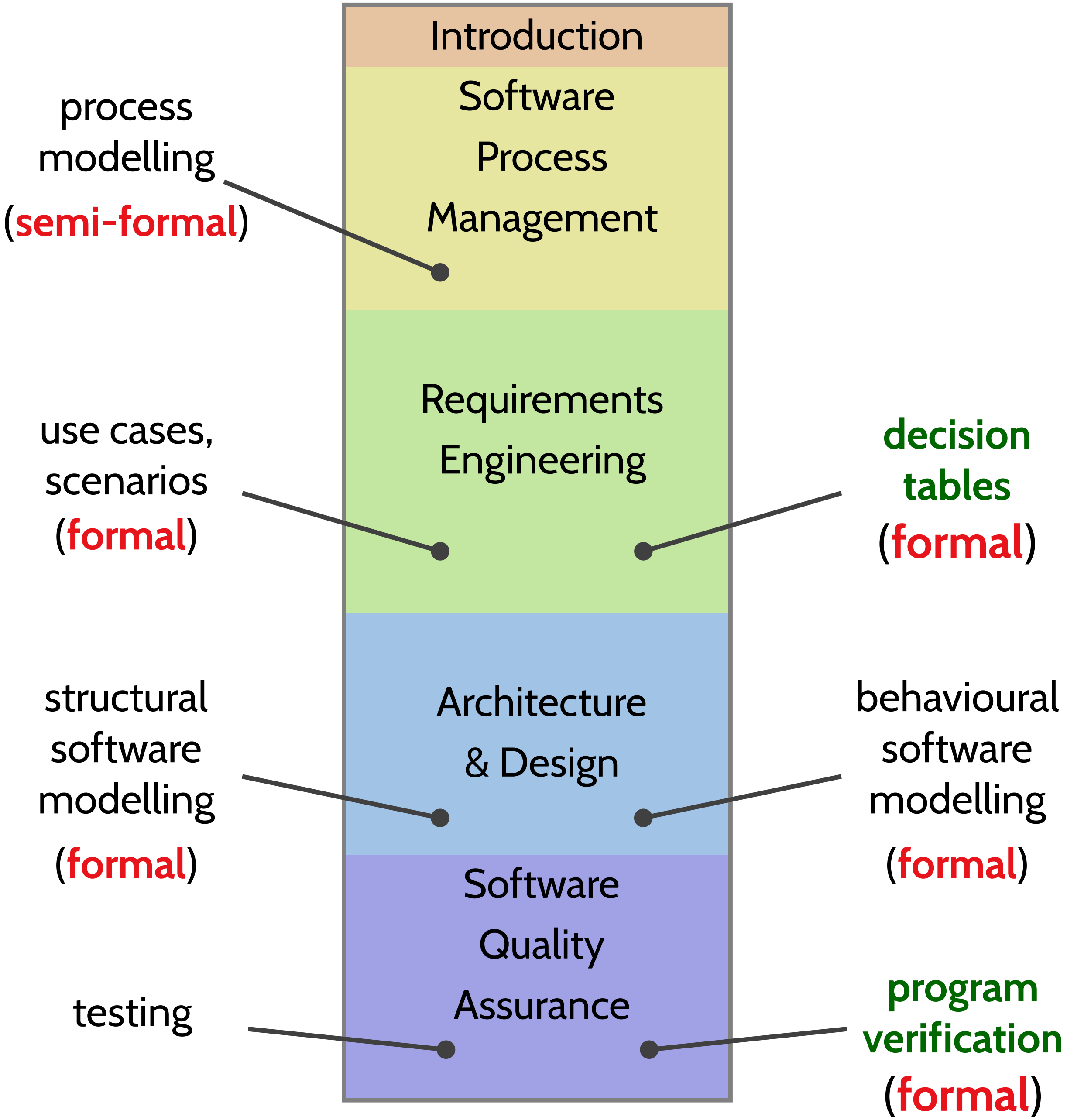
Complementing an ‘Ordinary’ Introduction to Software Engineering



informal

semi-formal

formal



Introduction

Software Process Management

process modelling (semi-formal)

Requirements Engineering

use cases, scenarios (formal)

decision tables (formal)

Architecture & Design

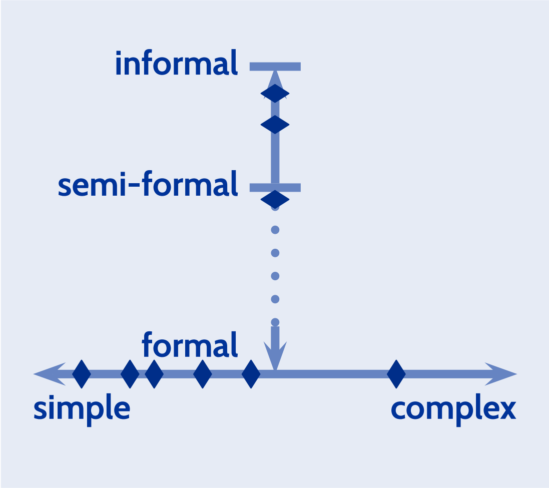
structural software modelling (formal)

behavioural software modelling (formal)

Software Quality Assurance

testing

program verification (formal)



informal

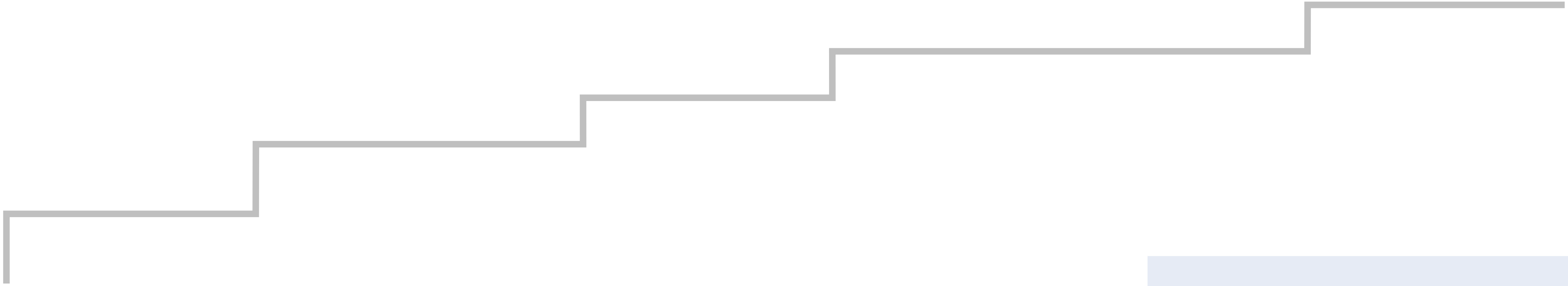
semi-formal

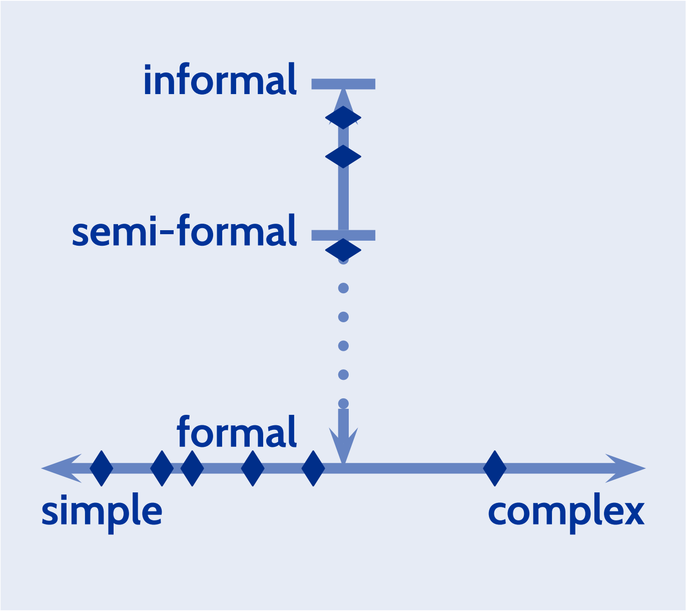
formal

simple

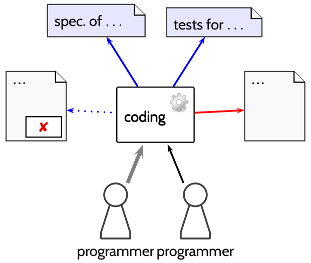
complex

Progression

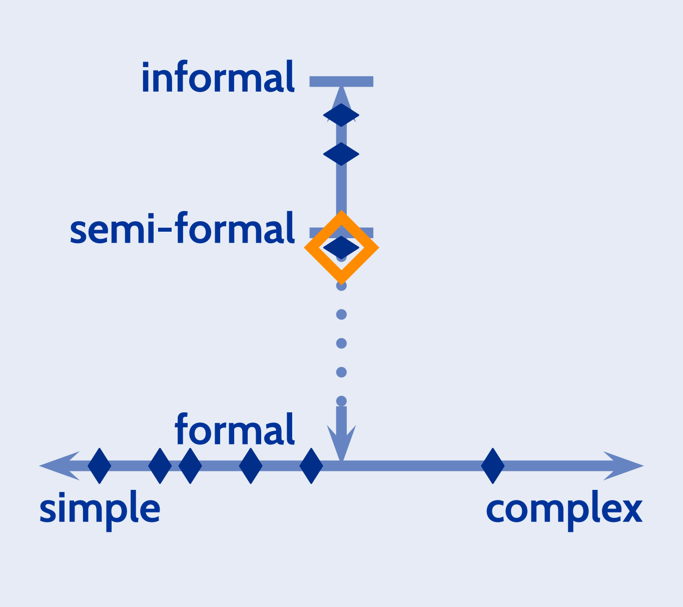




Progression

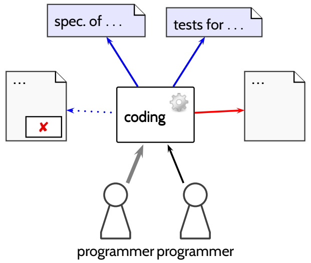


semi-formal
→ concrete syntax



Progression

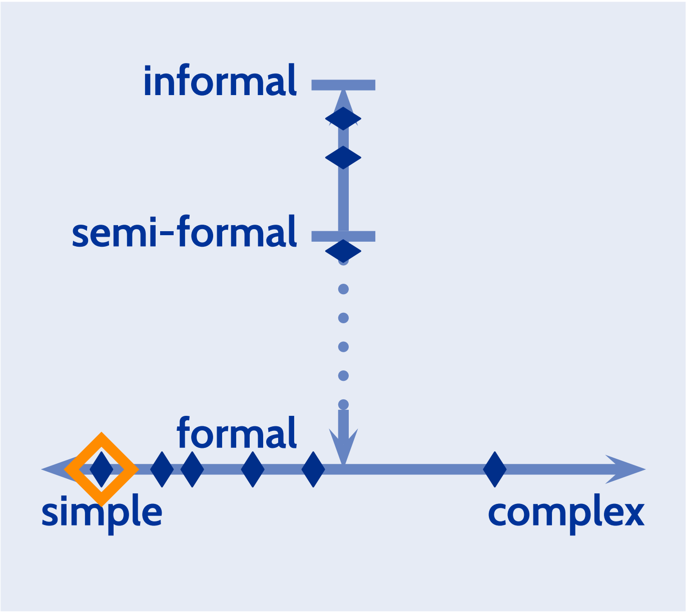
T: room ventilation		r_1	r_2	r_3
b	button pressed?	×	×	–
off	ventilation off?	×	–	*
on	ventilation on?	–	×	*
go	start ventilation	×	–	–
$stop$	stop ventilation	–	×	–



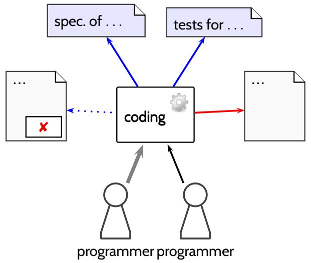
	customer's requirements	
	complete	incomplete
DT (formally) incomplete	false positive	true positive
DT (formally) complete	true negative	false negative

semi-formal
→ concrete syntax

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

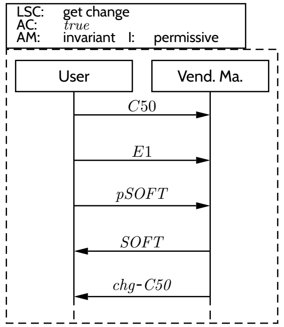


Progression



T: room ventilation		r_1	r_2	r_3
b	button pressed?	×	×	–
off	ventilation off?	×	–	*
on	ventilation on?	–	×	*
go	start ventilation	×	–	–
stop	stop ventilation	–	×	–

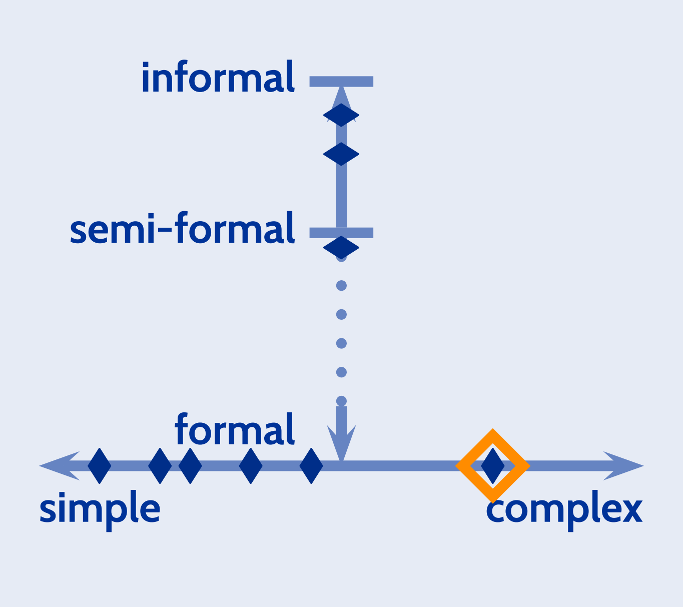
	customer's requirements	
	complete	incomplete
DT (formally) incomplete	false positive	true positive
DT (formally) complete	true negative	false negative



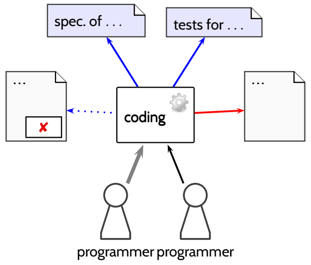
semi-formal
→ concrete syntax

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

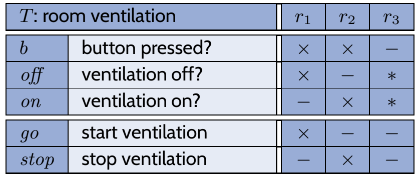
complex concrete and abstract syntax; complex semantics



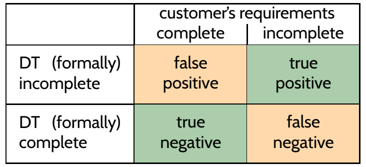
Progression



semi-formal
→ concrete syntax

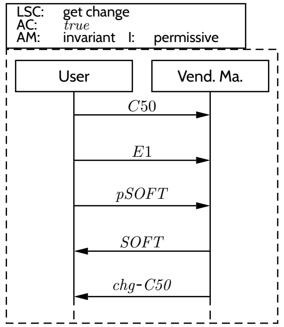


T: room ventilation		r_1	r_2	r_3
b	button pressed?	×	×	–
off	ventilation off?	×	–	*
on	ventilation on?	–	×	*
go	start ventilation	×	–	–
$stop$	stop ventilation	–	×	–

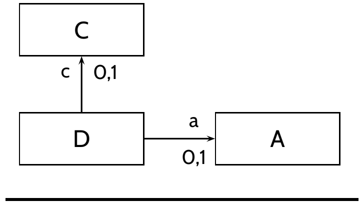


	customer's requirements complete	customer's requirements incomplete
DT (formally) incomplete	false positive	true positive
DT (formally) complete	true negative	false negative

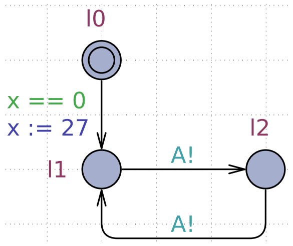
principles of formal methods (formal semantics, formalisation, validation, formal analysis, interpretation of results)



complex concrete and abstract syntax; complex semantics

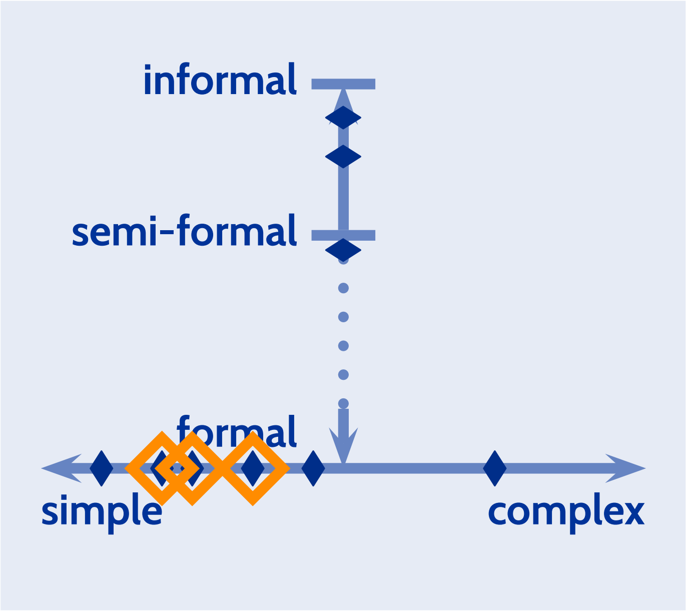


$$\forall d_1 \in allInstances_D \bullet$$
$$\forall d_2 \in allInstances_D \bullet$$
$$c(d_1) = c(d_2) \implies a(d_1) = a(d_2)$$

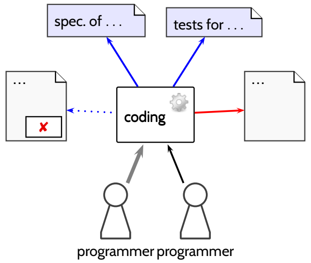


E<> x == 1

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



Progression

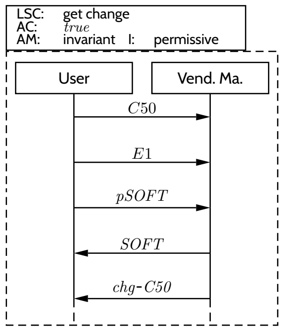


semi-formal
→ concrete syntax

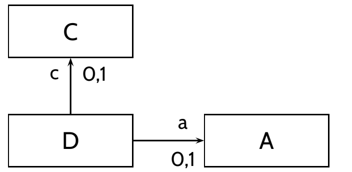
T: room ventilation		r_1	r_2	r_3
b	button pressed?	×	×	–
off	ventilation off?	×	–	∗
on	ventilation on?	–	×	∗
go	start ventilation	×	–	–
$stop$	stop ventilation	–	×	–

	customer's requirements	
	complete	incomplete
DT (formally) incomplete	false positive	true positive
DT (formally) complete	true negative	false negative

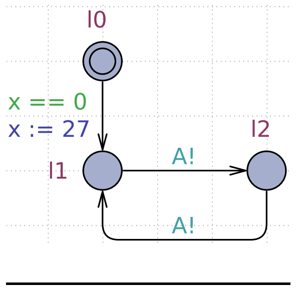
principles of formal methods (formal semantics, formalisation, validation, formal analysis, interpretation of results)



complex concrete and abstract syntax; complex semantics



$\forall d_1 \in allInstances_D \bullet$
$\forall d_2 \in allInstances_D \bullet$
$c(d_1) = c(d_2) \implies a(d_1) = a(d_2)$

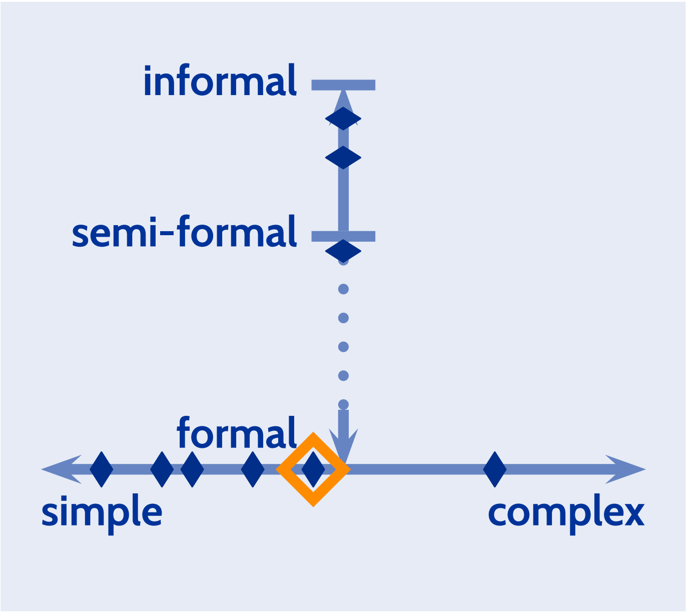


`E<> x == 1`

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

$\{P\}$ **while** $b \geq y$ **do**
$\{P \wedge b \geq y\}$
$b := b - y;$
$\{(a+1) \cdot y + b = x \wedge b \geq 0\}$
$a := a + 1$
$\{a \cdot y + b = x \wedge b \geq 0\}$
od $\{P \wedge \neg(b \geq y)\}$

deductive program verification



Conclusion

- 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 seaons of teaching an implementation of this course design:

    No indications of student over-strain (neither time, nor level.)