FM4All

FME Guidelines for Teaching Formal Methods

Luigia Petre
Åbo Akademi University, Finland
Formal Methods Europe

This session is 1/2

Åbo Akademi University

- Need to teach Formal Methods (FM)
  - Why?
- ACM Curriculum for Computer Science
  - The status quo
- Current FM representation in ACM Curriculum
  - What?
- Actions
  - How?

- Alternate between content and polls
  - We need your involvement

Lets test the polls

- Question
  - Where do you come from?

- Answer by
  - Either go to menti.com, use code 5516 4183
  - Or use QR code

Åbo Akademi University

Why teach Formal Methods?

A bit of history

Åbo Akademi University

- We started “FM4All” in December 2022, at a FME Business meeting
  - I presented the Teaching Committee status
  - Brijesh Dongol was part of the “Foundation of Programming Languages (FPL)” committee in the ACM Curriculum Task Force
  - He reported there was very little FM, only in two “Knowledge Areas”
    - In FPL
    - In Software Engineering (SE)
  - Cliff Jones (in the meeting) suggested we do something about it
  - We thought we could influence the ACM 2023 version of the curriculum

Why did we start FM4All?

Åbo Akademi University

- We contacted the ACM curriculum chairs and they asked these questions
  - Does every Computer Scientist need to know Formal Methods?
  - Aren't Formal Methods just for the safety-critical industry?
  - Isn't the current offer of Formal Methods education adequate?

- We want(ed) to make Formal Methods a core competency for Computer Scientists
  - Launched FAoC Special Issue call addressing the 3 questions
  - Summary of papers in December 2024 Inroads issue of the ACM
    - The Role of Formal Methods in Computer Science Education

FAoC Special Issue papers

- Does Every Computer Scientist Need to Know Formal Methods?
- Formal Methods in Industry
- On Formal Methods Thinking in Computer Science Education

Is there need?

- For systematically teach FM
- Lets check
  - The industry need
  - The education status

Industry using FM

Industry	What verified	Tools / Methods	Example
Aerospace	Flight control software,…	Astrée, Frama-C, CompCert	Airbus A380 flight control
Semiconductors	Processor correctness	Model checking, theorem proving	Intel / AMD CPU verification
Automotive	Autonomous, safety logic	Model checking, runtime verification	Autonomous driving systems
Medical Devices	Life-critical control	Model checking	Pacemakers, infusion pumps
Railways	Signalling & interlocking	B-Method	Metro/train control systems
Finance & Crypto	Security & protocols	Theorem proving, SMT	Ethereum smart contracts
Cloud / Distributed Systems	Consistency, fault tolerance, authorization	TLA+, Dafny, Z3	AWS S3, DynamoDB, AWS authorization engine
Operating Systems	Kernel correctness	Isabelle/HOL	seL4
Defense & Space	Mission-critical systems	Coq, SMT solvers	DARPA / Draper systems
Industrial Automation	Embedded control	Model checking	PLC / factory systems

Åbo Akademi University

Formal Methods in Industry

Åbo Akademi University

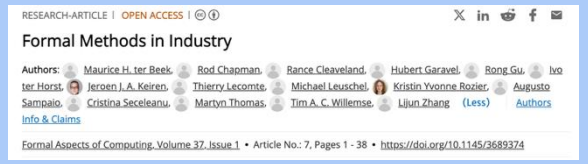
RESEARCH-ARTICLE | OPEN ACCESS |

Formal Methods in Industry

Authors: Maurice H. ter Beek, Rod Chapman, Rance Cleaveland, Hubert Garavel, Rong Gu, Ivo ter Horst, Jeroen J. A. Keiren, Thierry Lecomte, Michael Leuschel, Kristin Yvonne Rozier, Augusto Sampaio, Cristina Seceleanu, Martyn Thomas, Tim A. C. Willemse, Lijun Zhang (Less) Authors Info & Claims

Formal Aspects of Computing, Volume 37, Issue 1 • Article No.: 7, Pages 1 - 38 • https://doi.org/10.1145/3689374

From the paper


RESEARCH-ARTICLE | OPEN ACCESS

Formal Methods in Industry

Authors: Maurice H. ter Beek, Rod Chapman, Rance Cleaveland, Hubert Garavel, Rong Gu, Ivo ter Horst, Jeroen J. A. Keiren, Thierry Lecomte, Michael Leuschel, Kristin Yvonne Rozier, Augusto Sampaio, Cristina Seceleanu, Martyn Thomas, Tim A. C. Willemse, Lijun Zhang (Less) Authors Info & Claims

Formal Aspects of Computing, Volume 37, Issue 1 • Article No.: 7, Pages 1 - 38 • https://doi.org/10.1145/3689374

Åbo Akademi University

- Formal methods are widely applied in industry … elicitation of requirements and the early design phases all the way to the deployment, configuration, and runtime monitoring of actual systems.
- Formal methods allow one to **precisely specify the environment in which a system operates, the requirements and properties that the system should satisfy, the models of the system used during the various design steps, and the code embedded in the final implementation, as well as to express conformance relations between these specifications**.
- We present … well-known success stories from the safety-critical domain, like railways and other transportation systems, but also … lithography manufacturing and cloud security in e-commerce… testimonies from a number of representatives from industry …
- These persons are spread geographically, including **Europe, Asia, North and South America**, ….
- We thus make a case for … in particular of the capacity to abstract and mathematical reasoning that are taught as part of any formal methods course. **These are fundamental Computer Science skills that graduates should profit from when working as computer scientists in industry, as confirmed by industry representatives.**

Poll questions

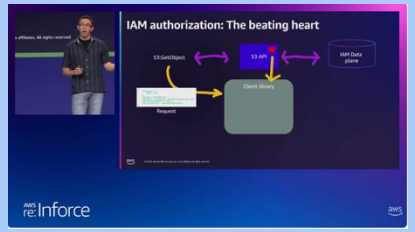
- Question
  - What industrial examples of using FM do you know about?
- Answer by
  - Either go to menti.com, use code 5516 4183
  - Or use QR code

Do we have a tipping point?


Formally Verified Cloud-Scale Authorization

Åbo Akademi University

- Amazon Web Services (AWS)
  - Verify their software
    - Summer 2024 → replaced IAM service with a verified one
      - without anyone noticing
    - Security policies proven correct
  - Formally Verified Cloud-Scale Authorization (ICSE, 2025)
  - Great talk @AWS re:Inforce 2025



- AWS → publicly presented formal verification as a factor driving customer adoption and workload migration toward AWS
  - Byron Cook, FLOC'2018 keynote
- **For the first time**
  - Industry companies not catching up to proof and verification → left behind!

Over a **period of four years**, we built a new authorization engine, one that **behaves functionally the same as its predecessor**, using the verification-aware programming language **Dafny**. We can now **confidently deploy enhancements and optimizations while maintaining the highest assurance of both correctness and backward compatibility**. We deployed the new engine in **2024** without incident and **customers immediately enjoyed a threefold performance improvement**

1) Rather than prove correct the existing engine, written in Java, we found it more effective to *write a new engine* in Dafny, a language built for *verification from the ground up*, and then compile the result to Java.
2) To ensure performance, debuggability, and to gain trust from stakeholders, we needed to generate readable, *idiomatic* Java code, essentially a transliteration of the source Dafny.

Åbo Akademi University

Formally Verified Cloud-Scale Authorization

Aleks Chakarov, Amazon, aleksach@
Jaco Geldenhuys, Amazon, jgeldenh@
Matthew Heck, Amazon, mcheck@
Michael Hicks, Amazon, mwhicks@
Sam Huang, Amazon, srhuang@
Georges-Axel Jaloyan, Amazon, gjaloyan@
Anjali Joshi, Amazon, anjalijs@
K. Rustan M. Leino, Amazon, leino@
Mikael Mayer, Amazon, mimayere@
Sean McLaughlin, Amazon, seanmcl@
Akhilesh Mritunjai, Amazon, amritun@
Clement Pit-Claudel, Amazon, pitclauc@amazon.ch
Sorawee Porncharoenwase, Amazon, soraweep@
Florian Rabe, Amazon, florrabe@
Marianna Rapoport, Amazon, rapopor@
Giles Reger, Amazon, reggiles@
Cody Roux, Amazon, codyroux@
Neha Rungta, Amazon, rungta@
Robin Salkeld, Amazon, salkeldr@
Matthias Schlaipfer, Amazon, schlaipf@
Daniel Schoepe, Amazon, schoeped@
Johanna Schwartzentruber, Amazon, jjsch@
Serdar Tasiran, Amazon, tasirans@
Aaron Tomb, Amazon, aarotomb@
Emina Torlak, Amazon, torlaket@
Jean-Baptiste Tristan, Amazon, trjohnb@
Lucas Wagner, Amazon, lgwagner@
Michael W. Whalen, Amazon, mnww@
Remy Willems, Amazon, rwillems@
Tongtong Xiang, Amazon, ttx@
Tae Joon Byun, Meta, taejoon@umn.edu
Joshua Cohen, Princeton University, jmc16@cs.princeton.edu
Ruijie Fang, University of Texas at Austin, rjf@abstractpredicates.org
Junyoung Jang, McGill University, junyoung.jang@mail.mcgill.ca
Jakob Rath, TU Wien, jakob.rath@tuwien.ac.at
Hira Taqdees Syeda, University of Melbourne, hira.syeda@unimelb.edu.au
Dominik Wagner, NTU Singapore, dominik.wagner@cs.ox.ac.uk
Yongwei Yuan, Purdue University, yuan311@purdue.edu

***Abstract*—All critical systems must evolve to meet the needs of a growing and diversifying user base. But supporting that evolution is challenging at increasing scale: Maintainers must find a way to ensure that each change does only what is intended, and will not inadvertently change behavior for existing users. This paper presents how we addressed this challenge for the Amazon Web Services (AWS) authorization engine, invoked 1 billion times per second, by using formal verification. Over a period of four years, we built a new authorization engine, one that behaves functionally the same as its predecessor, using the verification-aware programming language Dafny. We can now confidently deploy enhancements and optimizations while maintaining the highest assurance of both correctness and backward compatibility. We deployed the new engine in 2024 without incident and customers immediately enjoyed a threefold performance improvement. The methodology we followed to build this new engine was not an off-the-shelf application of an existing verification tool, and this paper presents several key insights: 1) Rather than prove correct the existing engine, written in Java, we found it more effective to *write a new engine* in Dafny, a language built for *verification from the ground up*, and then compile the result to Java. 2) To ensure performance, debuggability, and to gain trust from stakeholders, we needed to generate readable, *idiomatic* Java code, essentially a transliteration of the source Dafny. 3) To ensure that the specification matches the system's actual behavior, we performed *extensive differential and shadow testing* throughout the development process, ultimately comparing against 10^{15} production samples prior to deployment.**

Our approach demonstrates how formal verification can be effectively applied to evolve critical legacy software at scale.

The email addresses of Amazon-affiliated authors end with amazon.com. The work by non-Amazon-affiliated authors was done while they were at Amazon.

I. Introduction

To control access to their data and resources, AWS customers write policies that express fine-grained permissions. A cloud-hosted *authorization engine* evaluates incoming requests against those policies to either grant or deny access. It must ensure that decisions are both *fast*, to not slow down normal processing too much, and *correct*, by adhering to public documentation and meeting customer expectations. This authorization engine is used to secure access to over 200 fully featured services, invoked over 1 billion times per second.

Today, any change to the authorization engine undergoes a rigorous evaluation of both design and implementation,

This session is 2/2

- *Need to teach Formal Methods (FM)*
  - *Why?*
    - *Industry*
    - Universities
- ACM Curriculum for Computer Science
  - The status quo
- Current FM representation in ACM Curriculum
  - What?
- Actions
  - How?

- Alternate between content and polls
  - We need your involvement

Universities

- How many universities in the world?
  - 50K? → https://en.uhomes.com/blog/how-many-universities-are-there-worldwide?
  - UNESCO
    - ~15K in 78 countries → https://www.sciencedirect.com/science/article/pii/S0272775718300414?
  - WHED (World Higher Education Database)
    - 21K → https://whed.net/home.php?
  - 264 million students in 2023 → https://unesdoc.unesco.org/ark:/48223/pf0000394112

How many universities teach FM?

Åbo Akademi University

- How many universities teach CS?
  - About 75% → EdTechChronicle, Times Higher Education
- How many universities teach FM?
  - ~15–25%
  - FM appears as
    - Optional courses
    - Modules inside Software Engineering, Programming Languages areas
    - Graduate-only offerings
  - Top-tier / research-intensive universities: **~60–80%**
  - Mid-tier: **~20–40%**
  - Many institutions: **0% explicit FM**

How many Software Engineers know FM?

Åbo Akademi University

- How many Software Engineers in the world?
  - > 30 millions!
    - Developer Nation / SlashData estimates ~27M developers (2021) and ~47M (2025)
    - Evans Data estimates ~26–29M professional developers
- How many Software Engineers are capable of applying FM?
  - Well below 300k!
  - <1%

- Are there 300k people who can:
  - write invariants
  - use Dafny / Coq / TLA+ / model checkers
  - reason formally about correctness?

Gap

Metric	Estimate
Software engineers worldwide	>30 million
FM-capable engineers	<300000
FM workforce share	<1%
Universities teaching FM systematically	very small minority

Industry adoption growing
- AWS (TLA+)
- Intel (hardware FM)
- Airbus (avionics) …

▲
GAP
(skills shortage)
▼

SUPPLY limited
- Few FM courses
- Limited hands-on training
- Fragmented curricula

Åbo Akademi University

We have a gap


Rooting Formal Methods Within Higher Education Curricula for Computer Science and Software Engineering — A White Paper —


Formal Methods in Dependable Systems Engineering: A Survey of Professionals from Europe and North America

Åbo Akademi University

- "Lack of graduates qualified to apply formal methods"
  → Rooting Formal Methods within Higher Education Curricula for Computer Science and Software Engineering -- A White Paper (2021)
- Skills & education = major adoption barriers
  → Formal Methods in Dependable Systems Engineering: A Survey of Professionals from Europe and North America (2020)
- Academia–industry gap
  → The 2020 Expert Survey on Formal Methods (2020)


The 2020 Expert Survey on Formal Methods

The bottleneck for formal methods adoption

- is not tools/demand
- **is the shortage of people trained to use them**

Why do we teach so little FM?

Åbo Akademi University

- Misconceptions
  - Perceived mathematical difficulty for learning Formal Methods
    - Rooting Formal Methods within Higher Education Curricula for Computer Science and Software Engineering -- A White Paper (2021)
  - Software engineers dislike math
    - How Important are Formal Methods and Formal Logic for Software Engineering Education? (2025)
- Companies hire Computer Science students before they graduate
  - Bypass of complete formal education
  - Graduates
    - Slow career growth due to incomplete education
  - Companies
    - Lack of specialists
  - The Role of Formal Methods in Computer Science Education (2024)


Rooting Formal Methods Within Higher Education Curricula for Computer Science and Software Engineering — A White Paper —


How Important are Formal Methods and Formal Logic for Software Engineering Education?

Poll Questions

- Questions
  - Does your university teach FM?
  - If yes, how many courses?
  - If yes, at what level?
  - What FM courses?

- Answer by
  - Either go to menti.com, use code 5516 4183
  - Or use QR code

ACM Curriculum

CS-2023

Knowledge Areas

Knowledge Areas of CS2023:

- Algorithmic Foundations (AL)
- Architecture and Organization (AR)
- Artificial Intelligence (AI)
- Data Management (DM)
- Foundations of Programming Languages (FPL)
- Graphics and Interactive Techniques (GIT)
- Human-Computer Interaction (HCI)
- Mathematical and Statistical Foundations (MSF)
- Networking and Communication (NC)
- Operating Systems (OS)
- Parallel and Distributed Computing (PDC)
- Security (SEC)
- Society, Ethics, and the Profession (SEP)
- Software Development Fundamentals (SDF)
- Software Engineering (SE)
- Specialized Platform Development (SPD)
- Systems Fundamentals (SF)

ACM

IEEE-CS

AAAI

Åbo Akademi University

What is “the ACM Curriculum”?

Åbo Akademi University

- CS-2023
- Produced by ACM, IEEE-CS, AAAI
- Body of Knowledge for CS → made of 17 Knowledge Areas (KAs)
  - Topics every CS graduate must know → CS-Core
  - Topics for in-depth study → KA-Core
  - Every curriculum → focus on some KAs of interest
    - These KAs become the curriculum’s competency areas
- SEP (Society, Ethics, Profession) KA → newly introduced
- Role of Math ↗
- Need for Professional Dispositions → identified for each KA

History

- 1968 → classification of subject areas and course (ACM)
- 1978 → core and elective courses (ACM)
- 1991 → revision, together with IEEE-CS
- 2001 → focus on CS
  - Computer Engineering, Software Engineering → their own curricular guidelines
- 2008 → interim revision of ‘01 version
- 2013 → most recent before the present ‘23
- 2023 → AAAI joins ACM and IEEE-CS

Knowledge vs Competency

- Knowledge model
  - What is taught
    - Focus on content
  - First 5 editions only this
- Competency model
  - What is learnt
    - Focus on outcomes
  - Since 2013

Knowledge model

A **knowledge model** of a curriculum is structured as a set of knowledge areas:

$$\text{Knowledge model} = \{ \text{Knowledge areas} \}$$

A **knowledge area** is a set of related knowledge units:

$$\text{Knowledge area} = \{ \text{Knowledge units} \}$$

A **knowledge unit** is a set of related topics and a set of learning outcomes for those topics:

$$\text{Knowledge unit} = \{ \text{Topics} \} + \{ \text{Learning outcomes} \}$$

Learning outcomes are used for assessment. Each topic in a knowledge unit is categorized as either core or elective:

$$\text{Topic} \in \{ \text{core} \} \cup \{ \text{elective} \}$$

Core topics are topics that *every* graduate **must** know. Every curriculum is typically expected to cover all the core topics.

Elective topics are those that are not required of every graduate. Nevertheless, they complement the coverage of core topics. In addition to covering all the core topics, a curriculum is expected to cover a considerable percentage of elective topics.

Poll Questions

- Questions
  - Are you familiar with ACM Curriculum?
  - Is your university following the ACM Curriculum?

- Answer by
  - Either go to menti.com, use code 5516 4183
  - Or use QR code

Core topics

Åbo Akademi University

- CS-Core
  - Every CS graduate must know these
  - Every CS curriculum expected to cover all CS-Core topics
  - Included in > 50% of all KUs
- KA-Core
  - Topics recommended for in-depth studies
  - Included in > 75% of all KUs
  - A curriculum may choose to focus on KA-topics of some KAs in more depth/breadth than others

Competency areas

Åbo Akademi University

- Competency area ⊆ Knowledge model
- Competency area of a curriculum
  - Subset of KAs on which said curriculum is focused
    - When chosen coherently
- For guidelines → CS-2023 includes 3 competency areas deemed “representative”
  A. **Software development** → programmer
  B. **Systems development** → student able to provide essential services, including non-functional requirements
  C. **Applications development** → student prepped with problem-specific/solution-specific knowledge
  - **A.** prerequisite for **B.** and **C.**
  - SEP, MSF → part of all competency areas

Poll Questions

- Questions
  - Does your university have a defined competence area? E.g., scientific computing, security, …
- Answer by
  - Either go to menti.com, use code 5516 4183
  - Or use QR code

Sections in a Knowledge Area

Åbo Akademi University

- **Preamble** → describes KA, includes
  - Changes in KA since CS-2013
  - Optional overview of KUs in the KA
- Table with **CS-Core** and **KA-Core hours** assigned to KA
- **List of KUs**, each of which having
  - CS-Core, KA-Core, Non-core topics → enumerated
  - CS-Core and KA-Core illustrative learning outcomes → also enumerated
- **Professional dispositions** most relevant to the KA
- **Math requirements** for the KA → both required and desired
- **Suggestions for packaging courses** from the KA
- The committee members who reviewed and revised the KA recommendations

ACM Curriculum Context

Åbo Akademi University

- ACM curriculum
  - Curricular guidelines with worldwide recognition
  - Reference point for CS curricula worldwide
- Most recent → CS2023
- Some other undergraduate CS curricula
  - “All India Council for Technical Education”
  - ”101 Plan” → Ministry of Education in China, 2024
  - Parallel and Distributed Computing 2020
  - Computing Curricula, 2020, ACM+IEEE
    - Holistic view, relations between CS and SW Eng, SW Security, Data Science

Body of Knowledge

	Knowledge Area	# Knowledge Units	CS Core Hours	KA Core Hours
AI	Artificial Intelligence	12	12	18
AL	Algorithmic Foundations	5	32	32
AR	Architecture and Organization	11	9	16
DM	Data Management	13	10	26
FPL	Foundations of Programming Languages	22	21	19
GIT	Graphics and Interactive Techniques	12	4	70
HCI	Human-Computer Interaction	6	8	16
MSF	Mathematical and Statistical Foundations	5	55	145
NC	Networking and Communication	8	7	24
OS	Operating Systems	14	8	13
PDC	Parallel and Distributed Computing	5	9	26
SDF	Software Development Fundamentals	5	43	
SE	Software Engineering	9	6	21
SEC	Security	7	6	35
SEP	Society, Ethics, and the Profession	11	18	14
SF	Systems Fundamentals	9	18	8
SPD	Specialized Platform Development	8	4	
	Total	162	270	N/A

Åbo Akademi University

17 Knowledge Areas

Poll questions

- Question
  - Out of the 17 KAs, how many are represented with at least one course at your university?

- Answer by
  - Either go to menti.com, use code 5516 4183
  - Or use QR code

AI	Artificial Intelligence
AL	Algorithmic Foundations
AR	Architecture and Organization
DM	Data Management
FPL	Foundations of Programming Languages
GIT	Graphics and Interactive Techniques
HCI	Human-Computer Interaction
MSF	Mathematical and Statistical Foundations
NC	Networking and Communication
OS	Operating Systems
PDC	Parallel and Distributed Computing
SDF	Software Development Fundamentals
SE	Software Engineering
SEC	Security
SEP	Society, Ethics, and the Profession
SF	Systems Fundamentals
SPD	Specialized Platform Development

Åbo Akademi University

Åbo Akademi University

Current FM representation in CS-2023

Body of Knowledge

	Knowledge Area	# Knowledge Units	CS Core Hours	KA Core Hours
AI	Artificial Intelligence	12	12	18
AL	Algorithmic Foundations	5	32	32
AR	Architecture and Organization	11	9	16
DM	Data Management	13	10	26
FPL	Foundations of Programming Languages	22	21	19
GIT	Graphics and Interactive Techniques	12	4	70
HCI	Human-Computer Interaction	6	8	16
MSF	Mathematical and Statistical Foundations	5	55	145
NC	Networking and Communication	8	7	24
OS	Operating Systems	14	8	13
PDC	Parallel and Distributed Computing	5	9	26
SDF	Software Development Fundamentals	5	43	
SE	Software Engineering	9	6	21
SEC	Security	7	6	35
SEP	Society, Ethics, and the Profession	11	18	14
SF	Systems Fundamentals	9	18	8
SPD	Specialized Platform Development	8	4	
	Total	162	270	N/A

Åbo Akademi University

FM in 2 Knowledge Areas

Foundations of Programming Languages (FPL)

- CS-2023

Knowledge Unit	CS Core	KA Core
Object-Oriented Programming	4 + 1 (SDF)	1
Functional Programming	4	3
Logic Programming		2 + 1 (MSF)
Shell Scripting	2	
Event-Driven and Reactive Programming	2	2
Parallel and Distributed Computing	2 + 1 (PDC)	2
Aspect-Oriented Programming		
Type Systems	3	3

67

Systems Execution and Memory Model	2 + 1 (AR and OS)	
Language Translation and Execution	2	3
Program Abstraction and Representation		3
Syntax Analysis		
Compiler Semantic Analysis		
Program Analysis and Analyzers		
Code Generation		
Runtime Behavior and Systems		
Advanced Programming Constructs		
Language Pragmatics		
Formal Semantics		
Formal Development Methodologies		
Design Principles of Programming Languages		
Society, Ethics, and the Profession	Included in SEP hours	
Total	**21**	**19**

Åbo Akademi University

2 FM Knowledge Units (KU) in FPL

Åbo Akademi

FPL-Methodologies: Formal Development Methodologies

1. Formal specification languages and methodologies
2. Theorem provers, proof assistants, and logics
3. Constraint checkers (See also: FPL-Formalism)
4. Dependent types (universal quantification as dependent function, existential quantification as dependent product) (See also: FPL-Types, FPL-Formalism)
5. Specification and proof discharge for fully verified software systems using pre/post conditions, refinement types, etc.
6. Formal modeling and manual refinement/implementation of software systems.
7. Use of symbolic testing and fuzzing in software development.
8. Model checking.
9. Understanding of situations where formal methods can be effectively applied and how to structure development to maximize their value.

Illustrative learning outcomes:

Non-core:

1. Use formal modeling techniques to develop and validate architectures.
2. Use proof assisted programming languages to develop fully specified and verified software artifacts.
3. Use verifier and specification support in programming languages to formally validate system properties.
4. Integrate symbolic validation tooling into a programming workflow.
5. Discuss when and how formal methods can be effectively used in the development process.

FPL-Formalism: Formal Semantics

Non-core:

1. Syntax vs semantics
2. Approaches to semantics: axiomatic, operational, denotational, type-based
3. Axiomatic semantics of abstract constructs such as assignment, selection, iteration using pre-condition, post-conditions, and loop invariant
4. Operational semantics analysis of abstract constructs and sequence of such as assignment, expression evaluation, selection, iteration using environment and store
   a. Symbolic execution
   b. Constraint checkers
5. Denotational semantics
   a. Lambda Calculus. (See also: AL-Models, FPL-Functional)
6. Proofs by induction over language semantics
7. Formal definitions and proofs for type systems (See also: FPL-Types)
   a. Propositions as types (implication as a function, conjunction as a product, disjunction as a sum)
   b. Dependent types (universal quantification as dependent function, existential quantification as dependent product)
   c. Parametricity

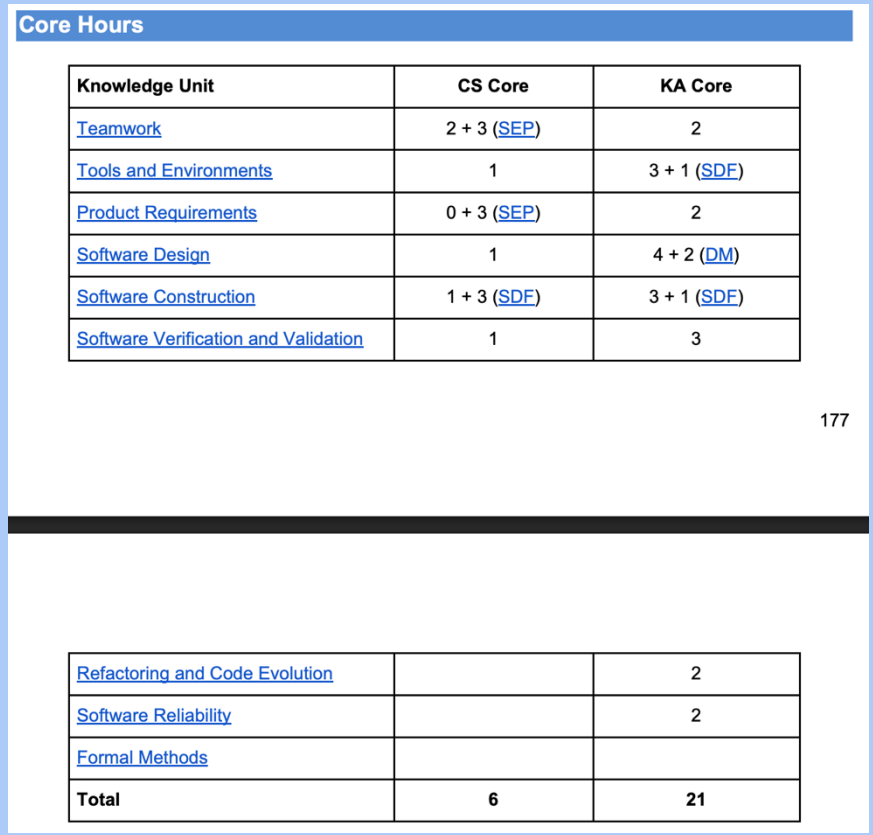
Illustrative learning outcomes:

Non-core:

1. Construct formal semantics for a small language.
2. Write a lambda-calculus program and show its evaluation to a normal form.
3. Discuss the different approaches of operational, denotational, and axiomatic semantics.
4. Use induction to prove properties of all programs in a language.
5. Use induction to prove properties of all programs in a language that is well-typed according to a formally defined type system.
6. Use parametricity to establish the behavior of code given only its type.

Software Engineering (SE)

- CS-2023



Core Hours

Knowledge Unit	CS Core	KA Core
Teamwork	2 + 3 (SEP)	2
Tools and Environments	1	3 + 1 (SDF)
Product Requirements	0 + 3 (SEP)	2
Software Design	1	4 + 2 (DM)
Software Construction	1 + 3 (SDF)	3 + 1 (SDF)
Software Verification and Validation	1	3

177

Refactoring and Code Evolution		2
Software Reliability		2
Formal Methods		
Total	**6**	**21**

1 KU in SE

Åbo Akademi University

SE-Formal: Formal Methods

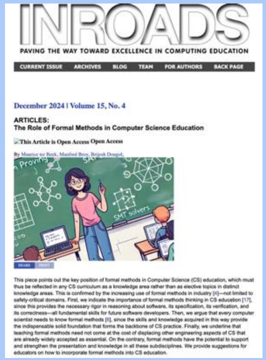
Non-Core:

1. Formal specification of interfaces
   a. Specification of pre- and post- conditions
   b. Formal languages for writing and analyzing pre- and post-conditions.
2. Problem areas well served by formal methods
   a. Lock-free programming, data races
   b. Asynchronous and distributed systems, deadlock, livelock, etc.
3. Comparison to other tools and techniques for defect detection
   a. Testing
   b. Fuzzing
4. Formal approaches to software modeling and analysis
   a. Model checkers
   b. Model finders

Illustrative Learning Outcomes:

1. Describe the role formal specification and analysis techniques can play in the development of complex software and compare their use as validation and verification techniques with testing.
2. Apply formal specification and analysis techniques to software designs and programs with low complexity.
3. Explain the potential benefits and drawbacks of using formal specification languages.

We tried to add FM in CS-2023



INROADS

PAVING THE WAY TOWARD EXCELLENCE IN COMPUTING EDUCATION

CURRENT ISSUE | ARCHIVES | BLOG | TEAM | FOR AUTHORS | BACK PAGE

December 2024 | Volume 15, No. 4

ARTICLES:
The Role of Formal Methods in Computer Science Education

This Article is Open Access Open Access

By Maurice ter Beek, Manfred Broy, Brijesh Dongol,

This piece points out the key position of formal methods in Computer Science (CS) education, which must thus be reflected in any CS curriculum as a knowledge area rather than as elective topics in distinct knowledge areas. This is confirmed by the increasing use of formal methods in industry [4]—not limited to safety-critical domains. First, we indicate the importance of formal methods thinking in CS education [17], since this provides the necessary rigor in reasoning about software, its specification, its verification, and its correctness—all fundamental skills for future software developers. Then, we argue that every computer scientist needs to know formal methods [6], since the skills and knowledge acquired in this way provide the indispensable solid foundation that forms the backbone of CS practice. Finally, we underline that teaching formal methods need not come at the cost of displacing other engineering aspects of CS that are already widely accepted as essential. On the contrary, formal methods have the potential to support and strengthen the presentation and knowledge in all these subdisciplines. We provide suggestions for educators on how to incorporate formal methods into CS education.

- The Role of Formal Methods in Computer Science Education (2024)
  - Importance of formal methods thinking in CS education
  - Every computer scientist needs to know formal methods
  - Teaching formal methods
    - need not come at the cost of displacing other engineering aspects of CS
  - Formal methods
    - potential to support and strengthen the presentation and knowledge in all these subdisciplines
    - suggestions for educators on how to incorporate formal methods into CS education.

Poll Questions

- Questions
  - What are the most important FM courses that you think should be taught by universities?
- Answer by
  - Either go to menti.com, use code 5516 4183
  - Or use QR code

Actions

Proposed Actions

Åbo Akademi University

1. **Guidelines for teaching Formal Methods**
   - Supported by FME
2. **Formal Methods “Knowledge Area” (FM-KA)**
   - For inclusion in the **ACM Computer Science Curriculum**
   - Define knowledge units, topics, propose courses
   - Link Formal Methods topics with existing computer science KUs in
     - Databases, Automata, Programming, …
   - Seek worldwide academic and industry support

Community effort

Åbo Akademi University

- Adoption to ACM Curriculum
  - If and when community supports innitiative of teaching FM in universities
  - Remember their questions
    - Does every Computer Scientist REALLY need to know Formal Methods?
    - AreN'T Formal Methods just for the safety-critical industry?
    - IsN'T the current offer of Formal Methods education adequate?
- FME Guidelines
  - FME's goal → represent the community of FM people

FM4All people

Åbo Akademi University

- Layers of involvement
  - Core committee → most active group
    - Produce initial draft
    - Incorporate community revisions
    - Produce public version for presentation to ACM
  - Advisory Committee → people in key roles who can support our work
    - Advice on major drafts
    - Promotion of inclusion to ACM curriculum

Core Committee

Åbo Akademi University

- Luigia Petre (Finland)
- Brijesh Dongol (UK)
- Stefan Hallerstaede (Denmark)
- Thierry Lecomte (France, Industry)
- Elvinia Riccobene (Italy)
- Peter-Michael Osera (USA)
- Graeme Smith (Australia)
- Kristine Yvonne Rozier (USA)
- Natarajan Shankar (USA)
- Cliff Jones (UK)
- Catherine Dubois (France)

Plan for feedback processing

Åbo Akademi University

Sections	Feedback Opens	Feedback Closes	Feedback Integration
Knowledge Units	April 2026	May 2026	June 2026
Topics	July 2026	August 2026	September 2026
Core/Elective	October 2026	November 2026	December 2026
Learning Objectives	January 2027	February 2027	March 2027
Professional dispositions	April 2027	May 2027	June 2027
Math requirements	July 2027	August 2027	September 2027
Courses	October 2027	November 2027	December 2027
Preamble	January 2028	February 2028	March 2028

Action Plan Timeline

- 2025
  - Create initial content
- 2026-2027/Early 2028
  - Present work progress biannually at major conferences
    - At least
  - Online presentations, held per KA section
  - Get feedback from community
  - Integrate feedback
- 2028
  - Publish research, expand industry collaboration

FM-KA

Structure

- **Preamble** → describes KA, includes
  - Changes in KA since CS-2013
  - Optional overview of KUs in the KA
- Table with **CS-Core** and **KA-Core hours** assigned to KA
- **List of KUs**, each of which having
  - CS-Core, KA-Core, Non-core topics → enumerated
  - CS-Core and KA-Core illustrative learning outcomes → also enumerated
- **Professional dispositions** most relevant to the KA
- **Math requirements** for the KA → both required and desired
- **Suggestions for packaging courses** from the KA
- The committee members who reviewed and revised the KA recommendations

Start from KU or LO?

- KU = Knowledge Unit
- LO = Learning Outcome/Objective

- We started with 6 LOs
  - LO1: Explain system correctness (as conformance to specified expected behavior)
  - LO2: Explain the benefits of specifying systems before their development
  - LO3: Develop rigorous specifications/models of system components
  - LO4: Evaluate through rigorous proof whether a system satisfies a specification
  - LO5: Develop rigorous reasoning and/or formal proofs
  - LO6: Apply tools to assist with program reasoning

LOs correspond to KUs

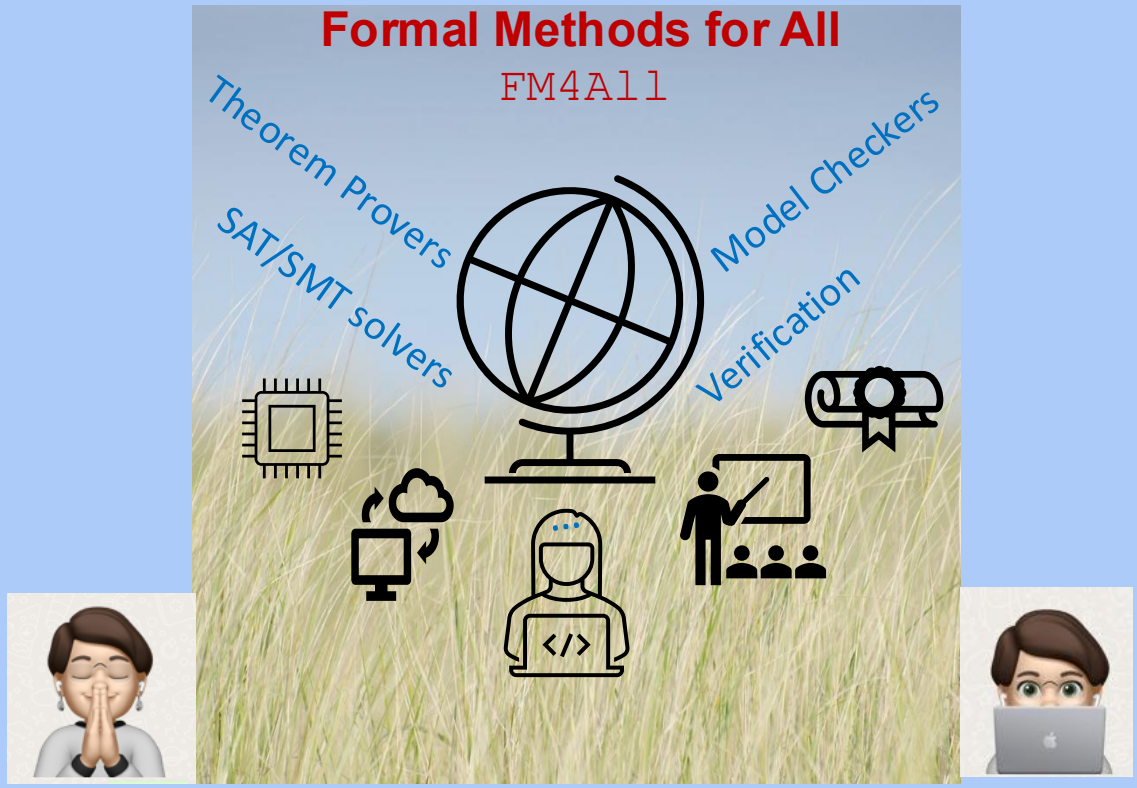
- Correctness
- Why specification
- Specification
- Modelling
- Analysis
- Proof
- Tools
- Applications

Poll Questions

- Questions
  - Do you see the need for other Knowledge Units (= FM relevant clusters of topics) to address?
- Answer by
  - Either go to menti.com, use code 5516 4183
  - Or use QR code

Open for feedback

- Here you can find the first community feedback form:
  - https://forms.gle/7vmMs8UpmA3xYMrL8
- Deadline May 31, 2026, AoE


Formal Methods for All
FM4All
Theorem Provers
SAT/SMT solvers
Model Checkers
Verification