



# A path for quality engineering in the age of AI

Bertrand Meyer

Eiffel Software & Recognize AI

ICEIS, Benidorm, 22 May 2026

# About these slides

---



This is a PDF of the slides presented during the keynote at ICEIS. Note that some slides had animations and may not be fully understandable without them.

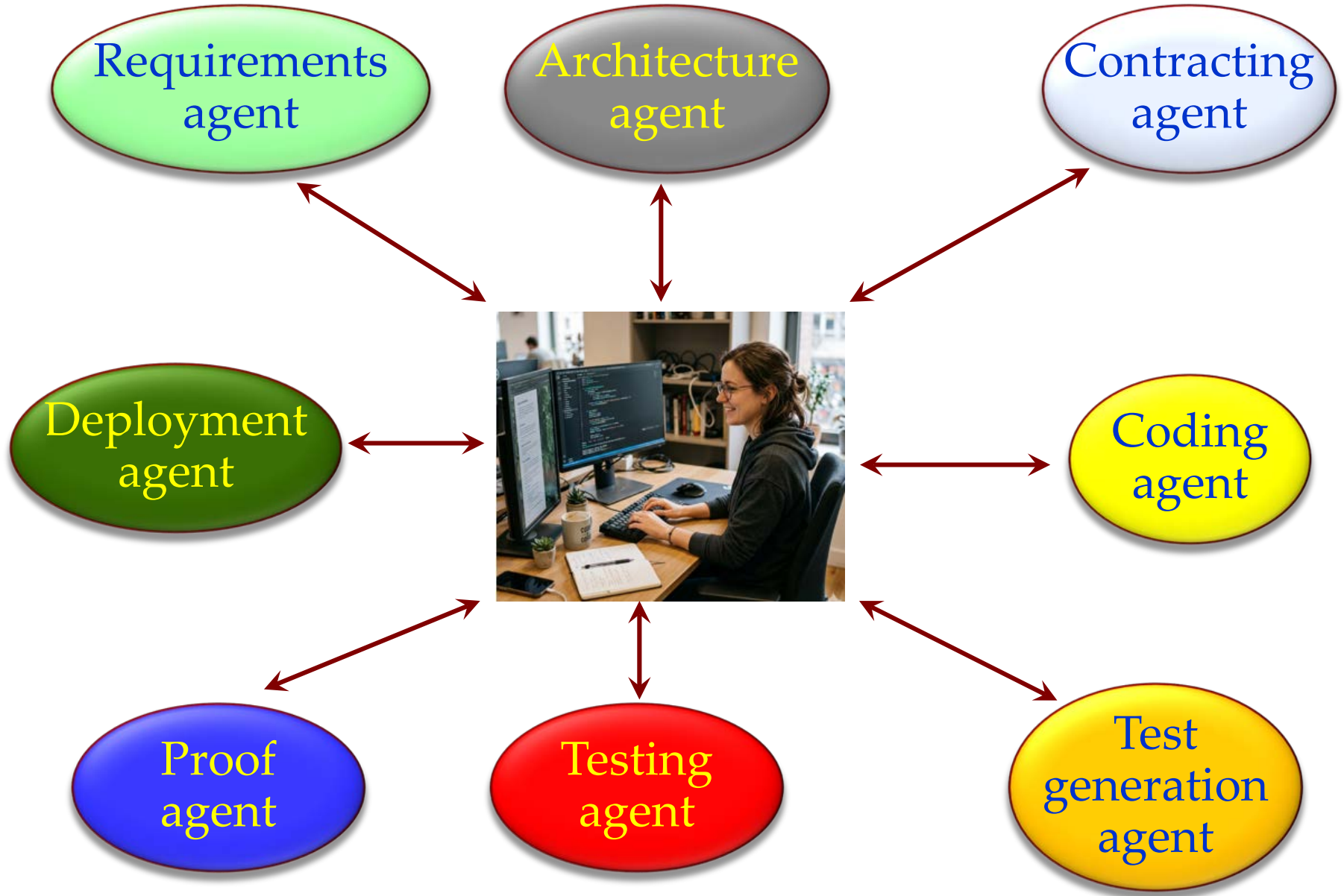
This document is not meant as a stand-alone document but as support material for the participants in the seminar. It is provided for the benefit of these participants and should not be circulated further.

For more structured presentations see:

- The “*Probable to Provable*” article, June 2026 Communications of the ACM.
- My forthcoming book “Understanding Artificial Intelligence: The Triumph of Empiricism” (to appear in June/July, there is a discount code for the ICEIS participants on the reference page)

For references to these and other cited works, with QR codes, see [The References page \(slide 50\)](#).

# Spilling the beans...



# The viral tweet (one year ago)



**Andrej Karpathy** ✓

@karpathy



There's a new kind of coding I call "vibe coding", where you fully give in to the vibes, embrace exponentials, and forget that the code even exists. It's possible because the LLMs (e.g. Cursor Composer w Sonnet) are getting too good. Also I just talk to Composer with SuperWhisper so I barely even touch the keyboard. I ask for the dumbest things like "decrease the padding on the sidebar by half" because I'm too lazy to find it. I "Accept All" always, I don't read the diffs anymore. When I get error messages I just copy paste them in with no comment, usually that fixes it. The code grows beyond my usual comprehension, I'd have to really read through it for a while. Sometimes the LLMs can't fix a bug so I just work around it or ask for random changes until it goes away. It's not too bad for throwaway weekend projects, but still quite amusing. I'm building a project or webapp, but it's not really coding - I just see stuff, say stuff, run stuff, and copy paste stuff, and it mostly works.

7:17 AM · Feb 3, 2025 · 7.1M Views

# The prospect of AI-based coding



# Technology breakthroughs: type L versus type E



Skill-leveling:

Type L

- AutoCAD
- GPS navigation
- Point-and-shoot photography
- Tax-preparation software



Skill-enhancing:

Type E

- Printing press
- Bloomberg terminals
- Professional photography equipment
- Object-oriented programming
- GPUs

# B.A.D. (Bullshit-Avoidance Discipline)

---



- Base our deductions  
(especially negative ones)  
on **latest** and **professional** tool versions  
(Things changed significantly in 2026)
- Base our deductions on measured (not anecdotal) evidence
- Distinguish between early hiccups and long-term properties

(At least, attempted)

# Original CNET iPhone review (30 June 2007)



**THE GOOD** The Apple iPhone has a stunning display, a sleek design and an innovative multitouch user interface. Its Safari browser makes for a superb web surfing experience, and it offers easy-to-use apps. As an iPod, it shines.

**THE BAD** The Apple iPhone has variable call quality and lacks some basic features found in many cellphones, including stereo Bluetooth support and a faster data network. Integrated memory is stingy for an iPod, and you have to sync the iPhone to manage music content.

**THE BOTTOM LINE** Despite some important missing features, a slow data network and call quality that doesn't always deliver, the Apple iPhone sets a new benchmark for an integrated cellphone and MP3 player.



# The claim: no need to write programs anymore

---



*Just* state what you need

and the AI will do the rest for you!

# Historical perspective




*“No need to program anymore!  
No need for programmers anymore!  
Users will write the programs!”*

1960s: **COBOL**

1970s: **4GL**

1980s: “Fifth generation”, CASE tools

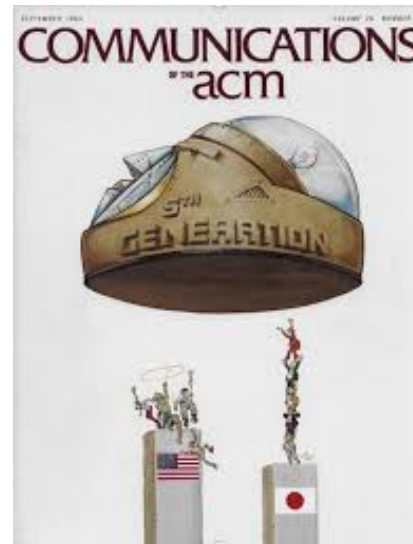
1990s: Component-Based Development

2000s: 

2010s: low-code, no-code



**NOCODE**  
DEVELOPMENT



Each time:  
elevated level of abstraction  
Is it the same this time?



- Most programmers in high-level approaches do not look at the generated code
  
- Most programmers using generative AI to produce the code still need to be able to look at it

# What are we talking about (1)?



A: Acute

B: Business

C: Casual

## Bertrand Meyer's technology+ blog

Software engineering, programming methodology, languages, verification, general technology, publication culture, and more

[Home](#)

[Bio](#)

[Earlier columns](#)

### The ABC of software engineering

25 March 2013

Blog, 2014



# Jensen Huang, CEO of NVIDIA



2024: *It is our job to create computing technology such that nobody has to program, and that the programming language is human. Everybody in the world is now a programmer.*

2024: *There's a new programming language. **It's called English...** You now have a computer that will do what you tell it to do.*

January 2026: *Everyone thought that software programming was the ultimate smart profession. Look what is the first thing that AI is solving. Software programming. And so it turns out that the definition of smart is very different than what most people think.*

March 2026: *I want our engineers to spend **exactly zero percent** of their time writing code... I want them to be directors of AI agents".*

# On the other hand...



## Senior Software Engineer, ASIC Design and Verification Tools

Apply

US, CA, Santa Clara

Full time

Posted Today

JR2012173

## Senior Software Engineer Aerial – Wireless RAN Algorithms Development

Apply

Sweden, Lund  
Sweden, Remote

Full time

Posted Today

JR2018617

## Manager, Software Engineering - Security Firmware

Apply

US, CA, Santa Clara  
US, Remote

Full time

Posted Today

JR2015868

## Senior System Software Engineer, Networking - DGX Cloud

Apply

Israel, Tel Aviv

Full time

Posted Yesterday

JR2011687

We are looking for a 100% hands-on Network Software engineer to join the block storage group. You will be a member of a team that builds the next generation block storage capabilities. You will work closely with a variety of teams and architects including the networking team, and external customers. You will take part in defining the software architecture and implementation of the most advanced storage services! Services that will need to meet extreme performance and scalability demands! We have crafted a team of extraordinary people stretching around the globe, whose mission is to push the frontiers of what is possible today and define the platform of tomorrow.

## Senior DevOps Engineer

Apply

India, Pune

Full time

Posted Yesterday

JR2017356

NVIDIA is looking for an excellent engineer to join its Software Infrastructure team. The position will be part of a dynamic crew that develops sophisticated software tools to optimize development workflow and increase overall efficiency. NVIDIA is crafting a

# Documented Gen-AI-linked mishaps

2 March 2026: amazon.com disruption,  
120,000 orders lost, 1.6 M site errors,  
*“novel GenAI usage for which best practices  
and safeguards not yet fully established”*  
→ 90-day “code safety reset”

Georgia Tech tracking: 35 new CVEs  
in single month (March 2026) directly  
attributable to AI coding tools...  
20% of AI-generated code references  
packages that do not exist

\$1.78M exploit hit  
Moonwell, a DeFi protocol,  
with the vulnerable code  
listing Claude as co-author

Lightrun, 2026 report:  
43% of AI-generated code  
changes require manual  
debugging even after  
passing QA tests. Not a  
single respondent said  
they could verify an AI-  
suggested fix with just  
one redeploy cycle; 88%  
reported needing 2 or 3

Sherlock Forensic (March 2026):  
Comprehensive audit of AI-  
generated codebases found that 92%  
contained at least one critical  
vulnerability, with an average of 8.3  
exploitable findings per application

# What are we talking about (2)?



If **coding** can be handled by AI... then what?

Software engineering includes:

- Requirements engineering
- Architecture, design
- **Coding (10 to 20%)**
- Validation and verification (V & V), including
  - Testing
  - Debugging
- Maintenance, extension, reuse
- Project management, measurement, process improvement...



# FISEE 3

Third International Workshop on  
**Frontiers In  
Software Engineering Education**



**Artificial Intelligence, Software Engineering  
and Teaching: *Recipes for Success***



September 27 - 30, 2026

📍 Villebrumier, France (near Toulouse)

Deadline 10 July – Springer LNCS proceedings

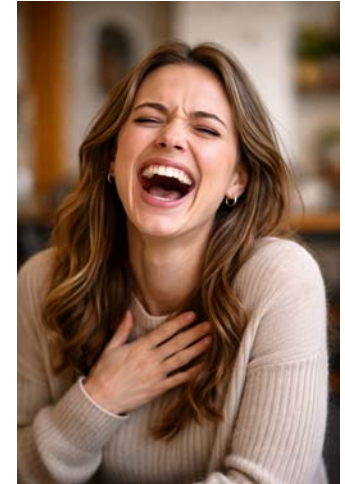
<https://www.laser-foundation.org/fisee/2026/>



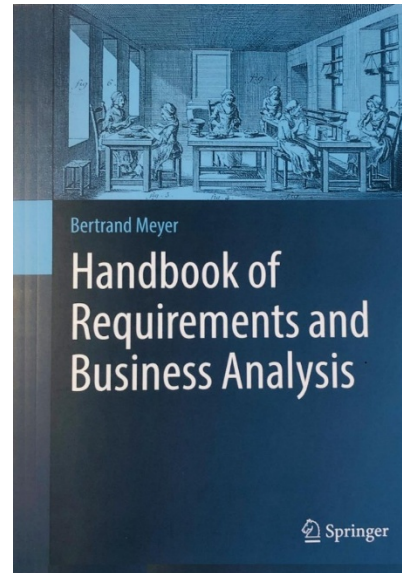
# You “just” state what you need



“*Just* stating what you need”  
is the task of requirements engineering...



... one of the most  
difficult parts of  
software engineering



[se.ethz.ch/requirements/](https://se.ethz.ch/requirements/)



Springer, 2022



# Hallucinations

Honoré Daumier,  
*Hallucination*



## *Do AI models help produce verified bug fixes?*

Li Huang, Ilgiz Mustafin, Marco Piccioni, Alessandro Schena, Reto Weber, Bertrand Meyer

To appear in VERIFAI-2026, Springer LNCS

Available as arXiv: [2507.15822](https://arxiv.org/abs/2507.15822)





Probability- and statistics-based

*“Stochastic parrots”*



Machine learning, neural networks, transformer architecture

The triumph of empiricism:

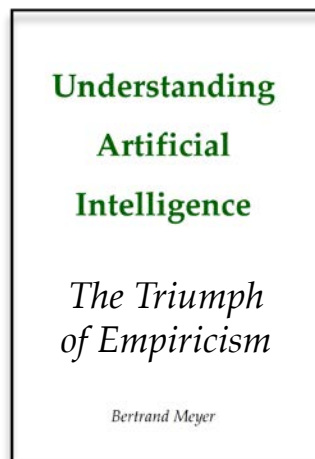
*“Do not design what you can learn”*

Made possible by

- Progress in hardware (GPUs)
- The Web: enormous corpus including large subset of human knowledge

Forthcoming book (June)

(ICEIS discount)



# Size of training base for code (estimated)

---



60 million public GitHub repositories  
(**100-500 billion LOC**)

60 million Stack Overflow posts

Wikipedia articles containing code

Books containing code

# How does this work for software engineering?



BY BERTRAND MEYER

## Artificial Intelligence for Software Engineering: From Probable to Provable

*Combining the creativity of artificial intelligence with the rigor of formal specification methods and the power of formal program verification, supported by modern proof tools.*



Here we go again: no programmers will be needed anymore! AI will generate the code!

If you have been around for a while, you may feel a sense of déjà vu. That same line advertised COBOL in the sixties, 4GLs in the seventies, CASE tools in the eighties, Component-Based Development in the nineties, Model-Driven Architecture in the aughts, low-code/no-code in the tens. Some of these approaches did improve programming, but they did not *replace* programming, let alone programmers. They simply introduced higher levels of abstraction or new tools, sometimes taking advantage of a restricted application domain. Is it the same this time, or do AI and vibe coding upend the game? More generally, can AI and software engineering enter into a successful marriage?

*Warning and spoiler-alert:* even though the following discussion starts out by examining limitations of AI for software construction, do not just expect a critique. Its aim is positive, in *support* of AI-supported software engineering. Its core thesis (here I am really spilling the beans) is that a successful solution requires combining AI with *formal* verification. (End of spoiler.)

**“Prompt engineering” is requirements engineering**

Are we about to witness the end of programming? The typical grand pronouncement is something like: “You will just state what you need: AI will generate the code for you”. The adverb, “just”, is epic. Ha-ha-ha. Stating what we need (“just” what we need) is Requirements Engineering (RE), among the hardest parts of software engineering (I devoted a recent book to it [7]). Anyone who has practiced RE knows that it does not differ that much from programming. It is not exactly the same thing, since it ignores algorithms and the implementation of data structures, but shares many challenges and techniques with programming, particularly concerns of abstraction, structuring, componentization and refinement. Tellingly, many specification languages resemble programming languages, sans the implementation part.

To appear (June 2026) in  
*Communications of the ACM*

Preprint: arXiv:2511.23159



Final URL (not yet operational):  
[doi.org/10.1145/3773295](https://doi.org/10.1145/3773295)





ESPRIT project

## Design of a Provably Secure Operating System

b



*C.A.R. Hoare*



*Brian Randell*

# “Almost correct”

Assume modules are  
(independently) correct with  
probability **99.9%**

Probability that their composition is correct:

1000 modules: 37%

**5000 modules: < 1%**

Based on a remark by Edsger Dijkstra  
in *Structured Programming*, 1970



# SE married to AI?



# SE married to AI?



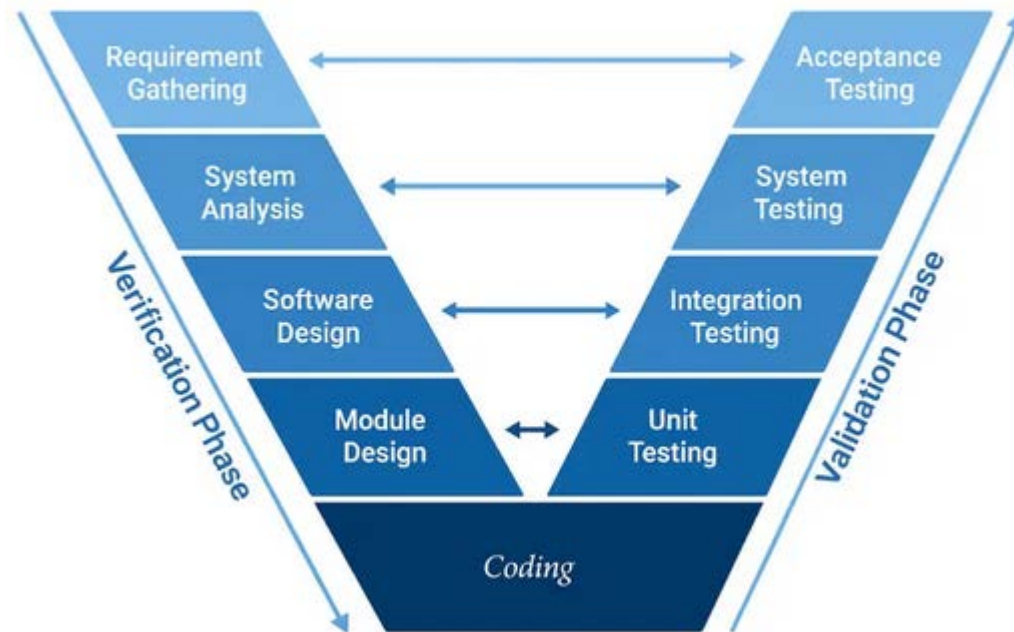
# Two sides of software development

---



*Anton Kaulbach:  
"Faust and Mephisto"*

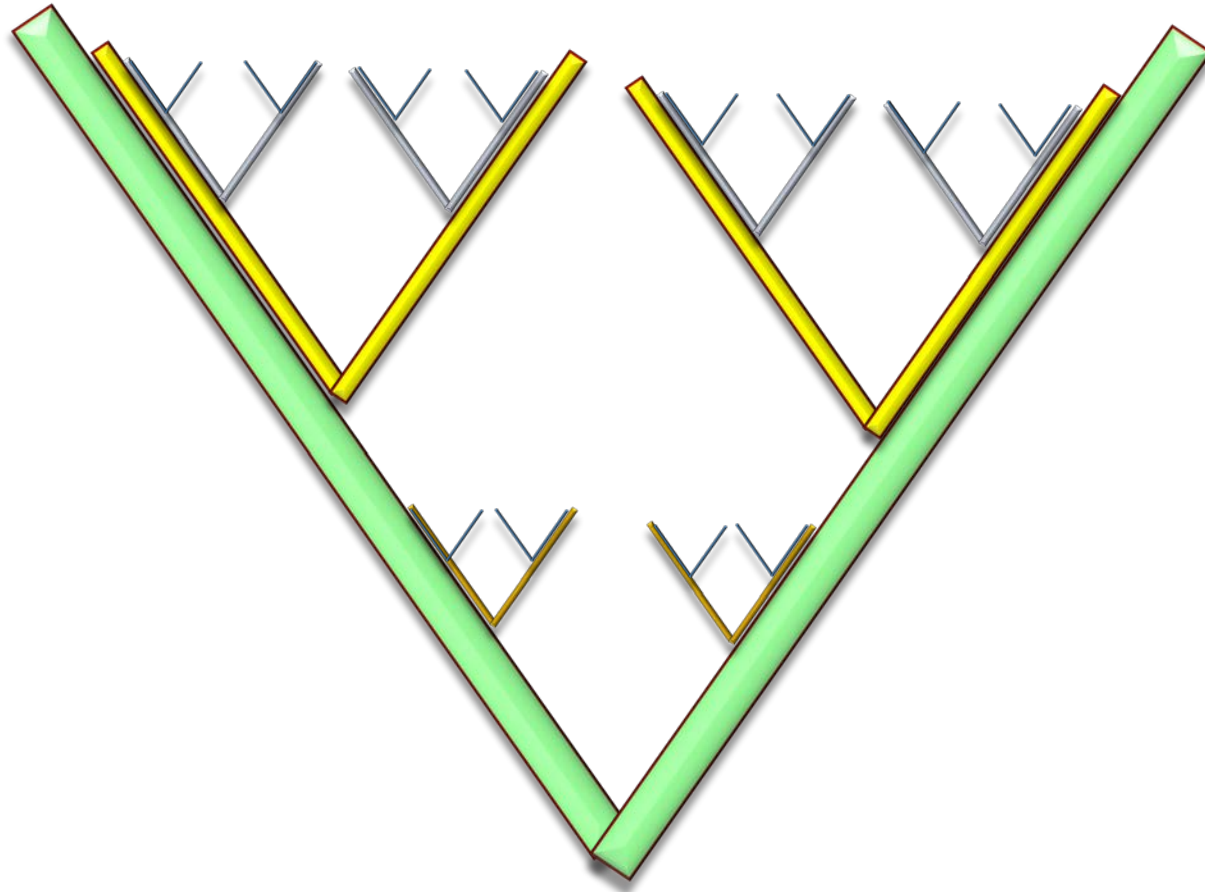
# The V model of software development\*



\*Picture by Abhay Talreja, [teachingagile.com/de/sdlc/modelle/v-model](https://teachingagile.com/de/sdlc/modelle/v-model)

# A more realistic picture

---

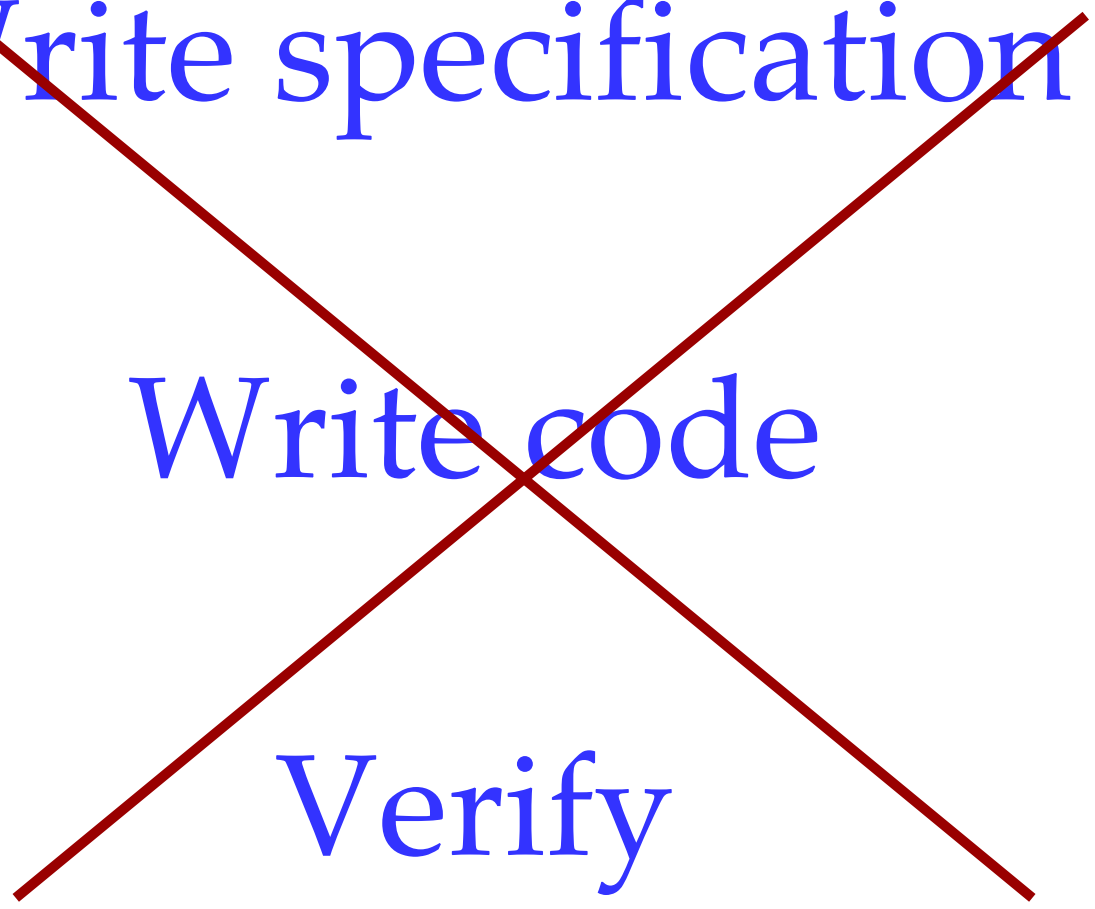




Write specification

Write code

Verify



# What formal methods work has (largely) missed

---

Brilliant techniques such as B implicitly assume that the requirements are known

Fascinating example:  
Compute 8<sup>th</sup> Fibonacci number



# Eiffel contracts: preconditions, postconditions, invariants



```
PERSON V_SET ANY ASSIGNMENT PAPER ✖
end

remove_author (p: PERSON)
  -- Remove `p` from authors.
  require
    p_attached: p /= Void
    present: authors.has (p)
    lock_wrapped: authors_impl.lock.is_wrapped
    is_locked: authors_impl.lock.locked [p]
  do
    authors_impl.remove (p)
    authors := authors_impl.set
  ensure
    removed: not authors.has (p)
  end

invariant
  model_representation_coupled:
    authors = authors_impl.set
```

Integrated with the program

Used for: testing, test generation,  
debugging, maintenance,  
project management...



[eiffel.com](http://eiffel.com)



# What formal methods work has (largely) missed



Brilliant techniques such as B implicitly assume that the requirements are known

Fascinating example:  
Compute 8<sup>th</sup> Fibonacci number

Largely ignores:

- Object-oriented programming
- Agile methods
- DevOps
- Benefits of tests (especially failed ones)
- Difficulty of getting specification right

# A proof environment: AutoProof

---



Eiffel classes with  
contracts

MML

AutoProof

Boogie (prover)

<http://autoproof.org>



# The reality of program verification (AutoProof example)



AutoProof

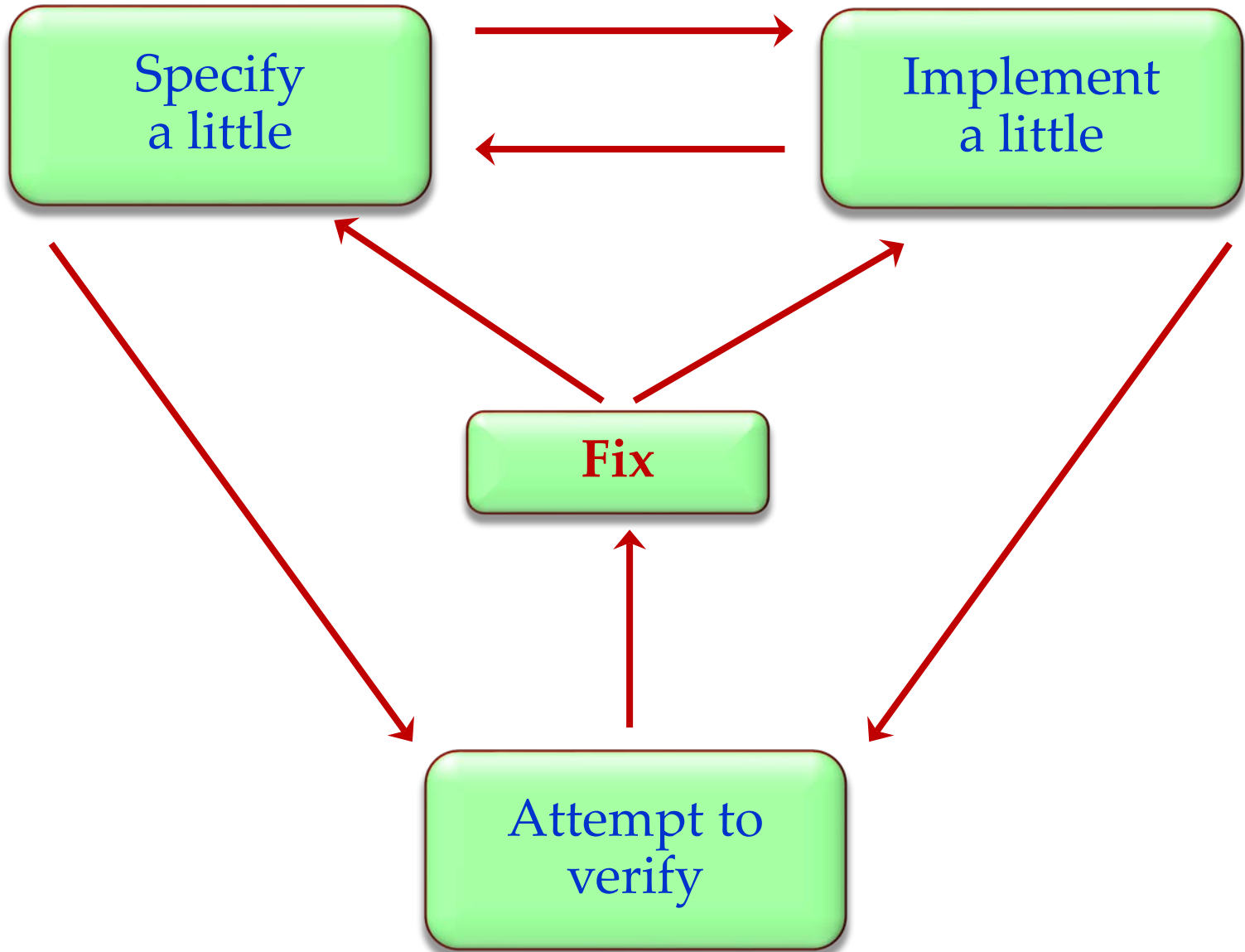
Verify \* 3 Successful 3 Failed 0 Errors Filter: [ ]

	Class	Feature	Information
✓	ACCOUNT	make (creator)	Verification successful.
✓	ACCOUNT	invariant admissibility	Verification successful.
✓	ACCOUNT	available_amount	Verification successful.
✗	ACCOUNT	deposit	Postcondition <code>balance_increased</code> may be violated.
✗	ACCOUNT	withdraw	Postcondition <code>balance_set</code> may be violated.
✗	ACCOUNT	transfer	Postcondition <code>deposit_made</code> may be violated. Postcondition <code>withdrawal_made</code> may be violated.

*A failed proof can yield a useful test, ICTSS 2023 (also SN-CS)*



# Towards a process





I want to produce a conference management system, a kind of mini-EasyChair, with the following two constraints: (1) it is written in Eiffel (at least for every part that can be written in Eiffel, the rest if any will be handled through "external" calls to C); and (2) it is entirely formally proved.

This means that I will need your help to produce both the specification (Eiffel contracts) and the code.

For the proofs, unless you can access an instance of AutoProof, I will perform them myself outside of this dialog.

I want to go step by step because my goal is not actually to produce a realistic, deployable alternative to EasyChair (which has been developed over several decades and has thousands of features and users) but to study a possible process for developing formally verified software.

Before you actually produce anything, is the general setting clear?



Generated:

- Code
- Architecture
- Glossary
- Project plan

## This step: complete P1 — Glossary (verification-oriented)

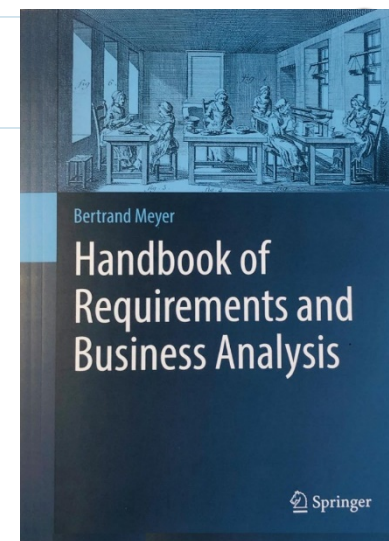
### 1) Actors & roles

- **1.1 PERSON:** An individual with identity `id`. May hold multiple roles in a given conference.
- **1.2 ROLE:** `{Author, Reviewer, Chair, PC_Member}` (extensible).
- **1.3 ROLE\_ASSIGNMENT:** Relation between `PERSON` and a `CONFERENCE` indicating roles active in that conference.
- **1.4 CONFLICT:** Symmetric relation on `PERSON × PERSON` scoped to a conference; includes self-conflict and declared/derived conflicts (e.g., coauthorship, institution rules).

### 2) Artifacts

- **2.1 PAPER:** An immutable sequence of **2.2 VERSION** objects (content hash, metadata, timestamp abstraction). Latest version is the submission of record unless withdrawn.
- **2.2 VERSION:** Content (opaque for now), authors list (subset of `PERSON`), title, abstract, tracks/topics.
- **2.3 REVIEW:** Evaluation by a reviewer for a specific paper and version; contains scores, text, and a recommendation.
- **2.4 ASSIGNMENT:** Relation between `PAPER` and `PERSON` (as reviewer) authorized by the chair/PC; must be conflict-free.
- **2.5 DECISION:** Element of `{Accept, Reject, Borderline, Shepherd}` with justification trace to reviews.

### 3) Structures



# Fair: proposed overall scheme



Given your constraints (Eiffel-first, C only via `externa1`, fully specified with contracts, you running AutoProof), I propose we proceed with a lean, verification-friendly process:

## 1. Guardrails

- Single-threaded, deterministic core (no time, no randomness).
- Persistence, I/O, email, and web UI hidden behind abstract interfaces (can later be `externa1 "C"` stubs).
- Use AutoProof-friendly idioms: MML `SET/SEQ/MAP/REL`, ghost/model attributes, semantic-collaboration invariants, pure vs. impure routines explicit, loop variants, selective export, void-safety.

## 2. Minimal initial scope ("mini-EasyChair v0")

- Entities: `CONFERENCE`, `PERSON` (roles: author/reviewer/chair), `PAPER` (with versions), `REVIEW`, `ASSIGNMENT`, `DECISION`.
- Core workflows: submission → assignment (respecting conflicts) → reviewing → decision.
- Key properties to verify early: no self-review, no conflicted assignments, every decision justified by available reviews, immutability of submitted versions.

## 3. Step-by-step deliverables

- Step A: Glossary + submission lifecycle state machine (verification-oriented).
- Step B: Deferred Eiffel skeletons with mathematical models and class invariants (no implementations yet).
- Step C: Effective implementations for the smallest vertical slice (submit, assign, review, decide), each routine fully contracted (pre/post/modify/frame, variants) and AutoProof-friendly.

If that matches your intent, I'll start with Step A (glossary + lifecycle) then immediately provide Step B (deferred class skeletons with models and invariants).



- Numerous errors
- Rules not enforced, e.g. code should always compile
- Hard to get provable code

# (After many steps) We can prove something!



The screenshot shows the Eclipse IDE interface. The main editor displays the following feature code for the `PERSON` class:

```
feature -- Commands (to be used later)

add_author (p: PERSON)
  -- Add `p` to authors.
  require
    p_attached: p /= Void
    not_present: not authors.has (p)
    lock_wrapped: authors_impl.lock.is_wrapped
    is_locked: authors_impl.lock.locked [p]
  do
    authors_impl.extend (p)
    authors := authors_impl.set
  ensure
    added: authors.has (p)
  end

remove_author (p: PERSON)
  -- Remove `p` from authors.
```

The right-hand pane shows a project tree with the following structure:

- EASAI\_CHAIR
  - core
    - ASSIGNMENT
    - CONFERENCE
    - CONFLICT\_POLICY
    - DECISION
    - PAPER
    - PERSON
    - REVIEW
    - TOPIC
    - VERSION
  - types
  - workflow
    - APPLICATION
- Libraries
  - base
  - testing

The bottom pane shows the AutoProof verification results:

Class	Feature	Information	P...	Ti...
PERSON	invariant admissibility	Verification successful.		0.40
PERSON	make (creator)	Verification successful.		0.00
PERSON	make	Verification successful.		0.00
PERSON	hash_code	Verification successful.		0.04
PERSON	hash_code_	Verification successful.		0.01
PERSON	is_model_equal	Verification successful.		0.00
PERSON	lemma_transitive	Precondition may be violated on invocation of (PERSON).is_model_equal.	760	0.04

# Invariant cannot be proved

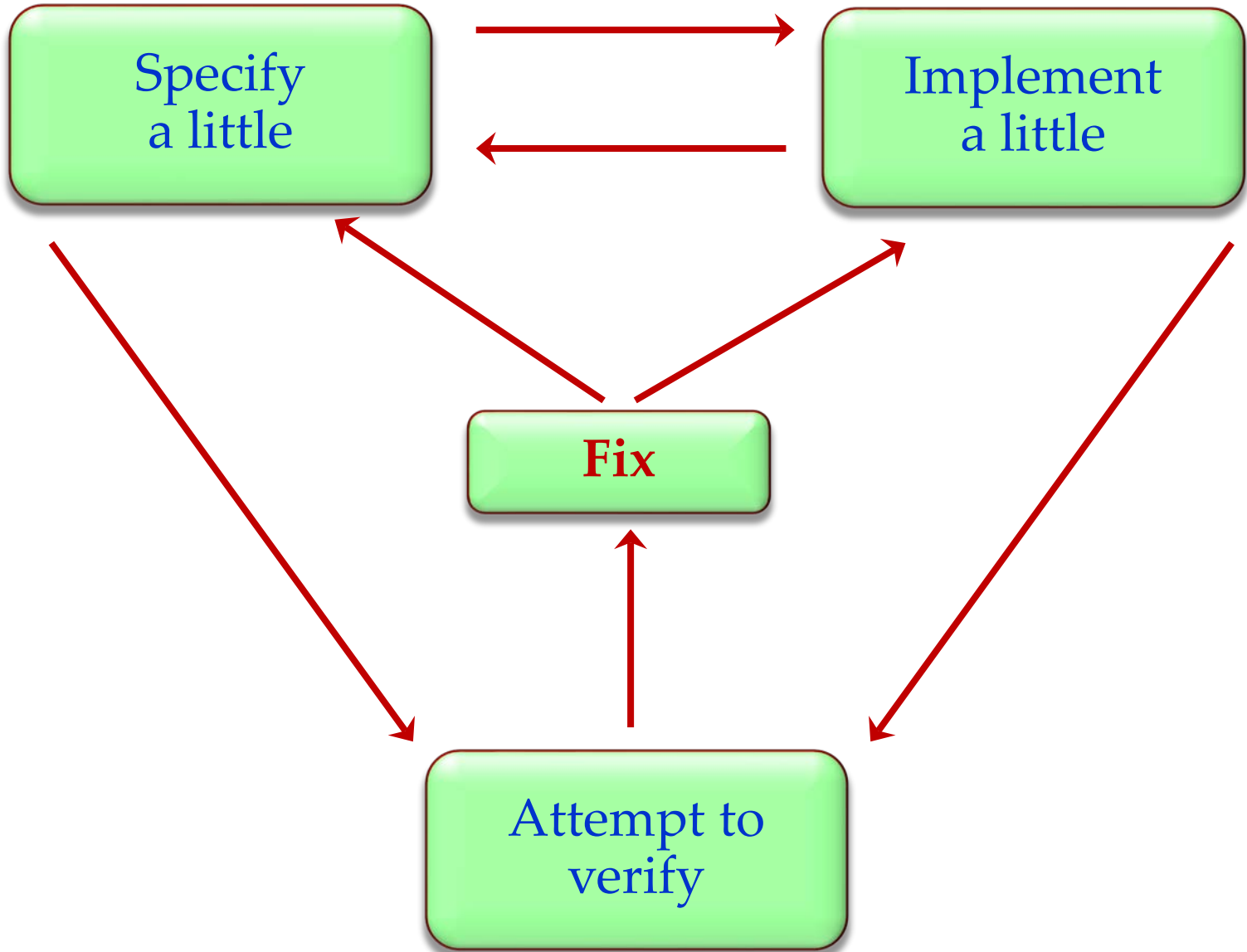


```
-- Remove 'p' from authors.
require
  p_attached: p /= Void
  present: authors.has (p)
  lock_wrapped: authors_impl.lock.is_wrapped
  is_locked: authors_impl.lock.locked [p]
do
  authors_impl.remove (p)
  authors := authors_impl.set
ensure
  removed: not authors.has (p)
end

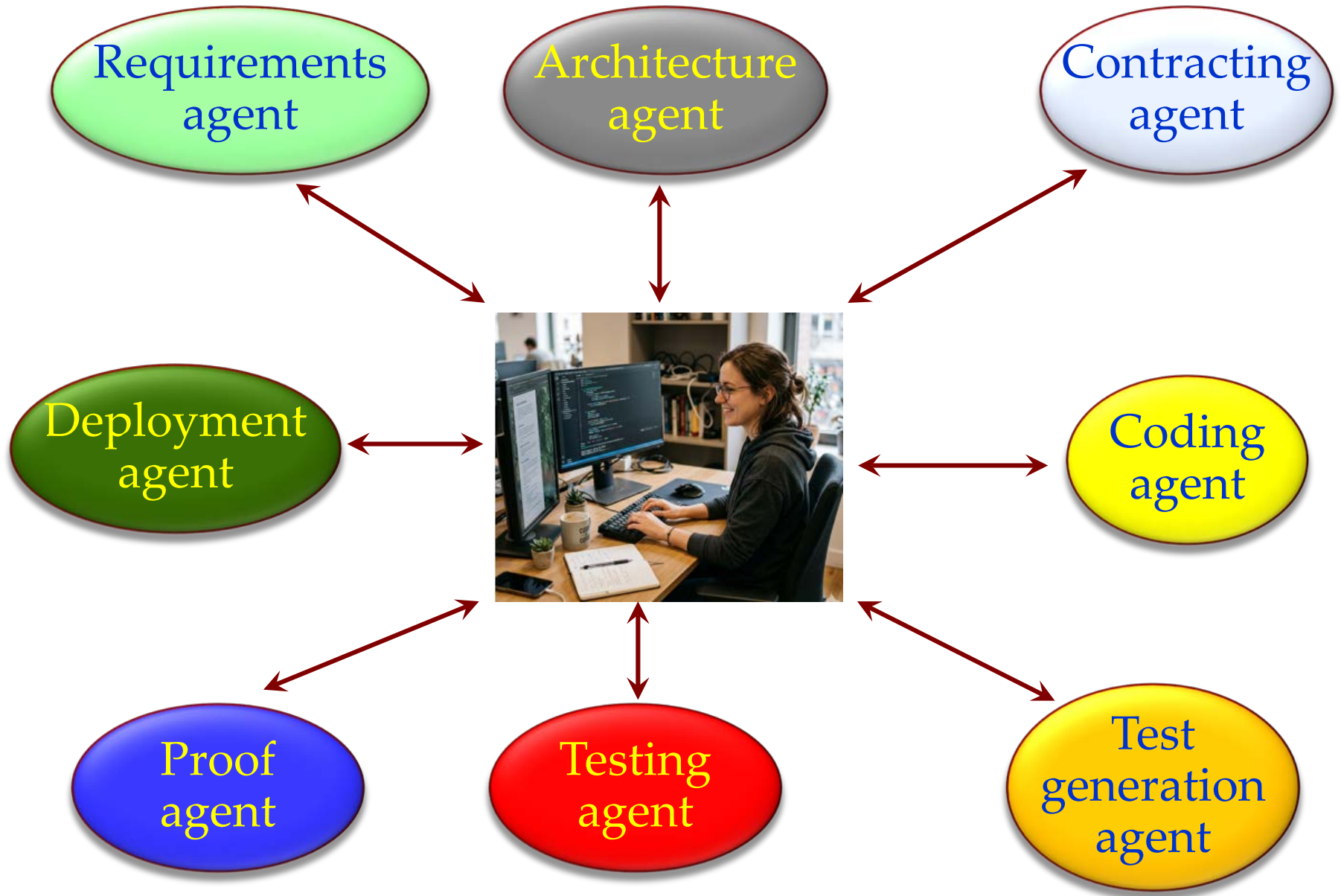
invariant
  model_representation_coupled:
    authors = authors_impl.set
end
```

Class	Feature	Information	P...	Ti...
PAPER	invariant admissibility	Attribute [V_HASH_SET].set might not be readable.	76	
PAPER	make (creator)	Verification successful.		0.50
PAPER	make	Verification successful.		0.24
PAPER	has_author	Verification successful.		0.38
PAPER	add_author	Verification successful.		0.24
PAPER	remove_author	Verification successful.		0.09

# Towards a process



# AI agents to support the process



# Summary



A: acute  
B: Business  
C: Casual

- AI support for SE **is here to stay**
- “C”-class development can be **essentially automated**

For A and B:

- AI is **not yet ready** for a full SE process
- There is a **culture clash** between AI and SE
- **Hallucinations** remain a major problem
- The only sure way is **formal verification**
- We need a clear **process**
- Real software development is **iterative**
- We are **only at the beginning**

# Individual strategies



L: leveling  
E: enhancing

- The irruption of AI into SE is both:
  - Type-L for C-class developments (Casual)
  - **Type-E for the rest (Business and Acute)**
- Unclear prospects for low-skill outsourcing
- Excellent prospects for companies to reclaim ownership
- Every software engineer must master Modern-AI
- **You are still responsible for the code!**
- Quality is key, particularly correctness, robustness, extendibility, reusability, “maintainability”
- Lessons of software engineering still apply – more than ever

# A happy marriage?

---



# References

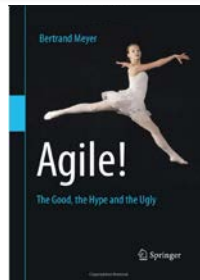
From probable to provable  
[arxiv.org/abs/2511.23159](https://arxiv.org/abs/2511.23159)



(Future: [doi.org/10.1145/3773295](https://doi.org/10.1145/3773295))



Agile book  
[agile.ethz.ch/](https://agile.ethz.ch/)



Tests and proofs

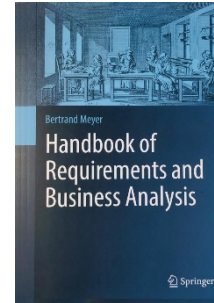
[arxiv.org/abs/2601.16239](https://arxiv.org/abs/2601.16239)



Eiffel  
[eiffel.com](https://eiffel.com)



# Requirements book



[se.ethz.ch/requirements/](https://se.ethz.ch/requirements/)



FISEE  
workshop: AI  
in SE education



Intro programming  
textbook

[touch.ethz.ch](https://touch.ethz.ch)



Autoproof prover  
[autoproof.org](https://autoproof.org)



Receive  
announcement  
+ discount for  
new AI book



Understanding  
Artificial  
Intelligence

*The Triumph  
of Empiricism*  
Bertrand Meyer

# The specter and appeal of AI-based ~~coding~~ software engineering

