Introduction to software analysis

Software Analysis
Topic 1

Carlo A. Furia

USI – Università della Svizzera Italiana

Today's menu

Motivation

What is software analysis?

Correctness

Course outline

Motivation

Ariane 5 rocket

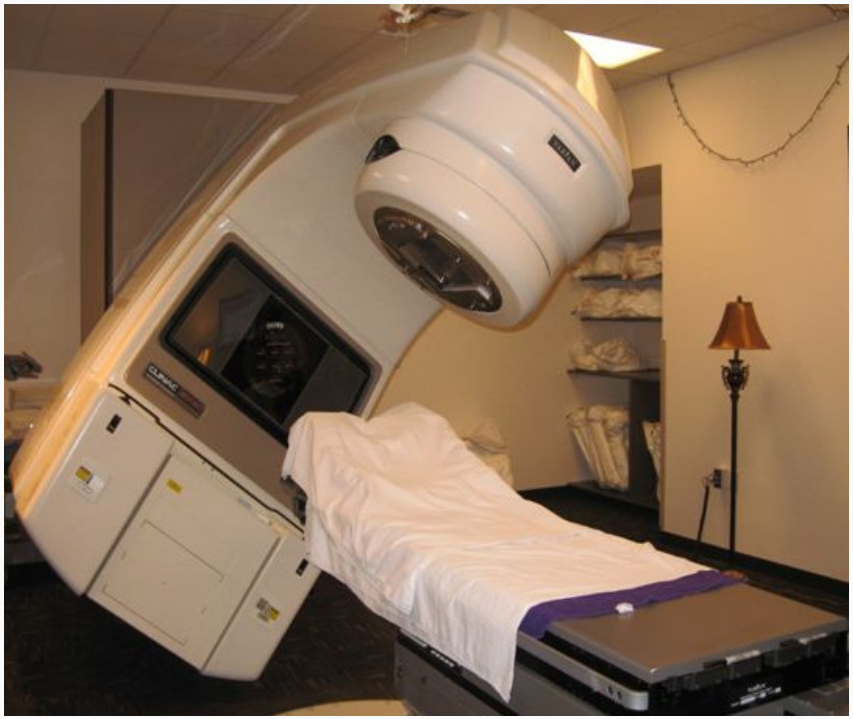
Ariane 5 rocket – Flight 501

Ariane 5 rocket

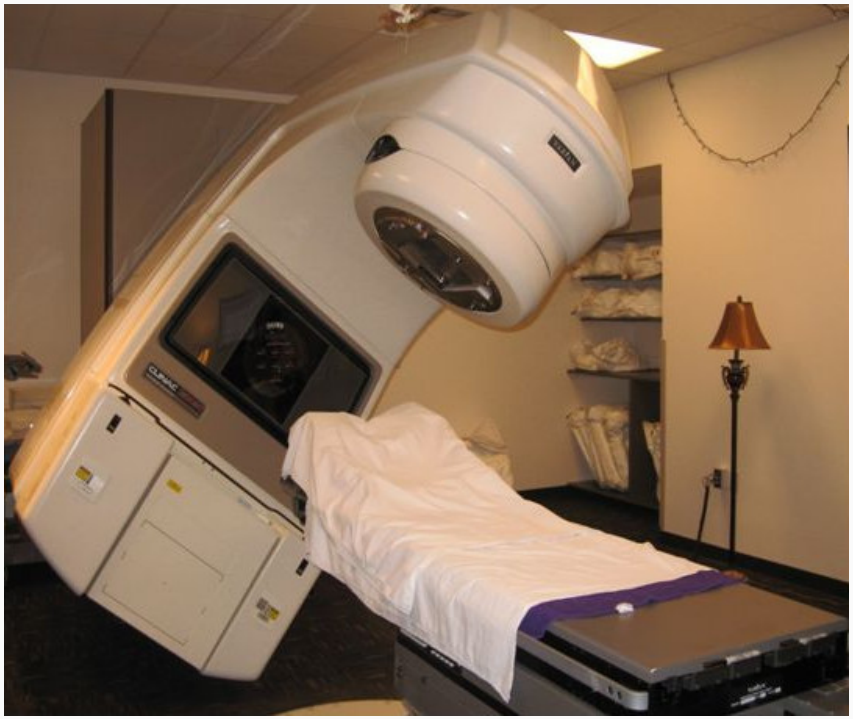
Ariane 5 rocket – Flight 501

- 1996: the launcher rocket disintegrated 39 seconds after take off
- Failure: an overflow, caused by a conversion from 64-bit to 16-bit floating point
- Mistake: reusing the inertial reference platform of the Ariane 4, where the overflow cannot happen due to different operational condition
- Cost: $500 million payload, $8 billion development program

Therac-25 radiation therapy machine



Therac-25 radiation therapy machine



- 1985–1987: the machine gave overdoses of radiations to 6 patients, killing 3
- Failure: a race condition (concurrency error)
- Mistake: poor design, poor software development practice (for example: hard to test)
- Cost: several human lives

NASA's Pathfinder mission

NASA's Pathfinder mission

- 1997: the rover Soujourner hung when collecting meteorological data on Mars
- Failure: a priority inversion (concurrency error)
- Mistake: the bug was known before mission but was given low priority
- Cost: the bug was patched after a 18-hour remote debugging session

A gruesome beginning

Software correctness is very important.

A gruesome beginning

Software correctness is very important.

Formal software analysis is our only hope to avoid havoc.


And now for
something
completely
different. . .

Software verification anyone?

The formal verification of programs is difficult to justify and manage.

Reports and Articles

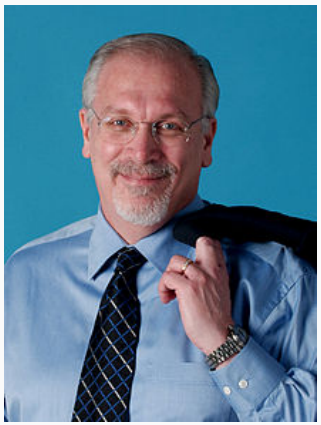
Social Processes and Proofs of Theorems and Programs

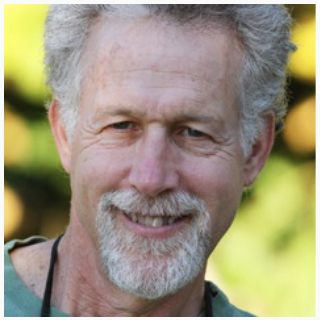
Richard A. De Millo
Georgia Institute of Technology

Richard J. Lipton and Alan J. Perlis
Yale University

Communications of the ACM

May 1979
Volume 22
Number 5





Formal methods anyone?

Many current formal methods tools and techniques more closely resemble the Wright Flyer than the Boeing 777.

To appear in the Proceedings of the 16th Digital Avionics Systems Conference, October 1997

WHY ENGINEERS SHOULD CONSIDER FORMAL METHODS

C. Michael Holloway

NASA Langley Research Center
Mail Stop 130 / 1 South Wright Street
Hampton, Virginia 23681-0001
E-mail: c.m.holloway@larc.nasa.gov

Formal specifications anyone?

Formally specifying real systems [...] continue[s] to be impossibly difficult.

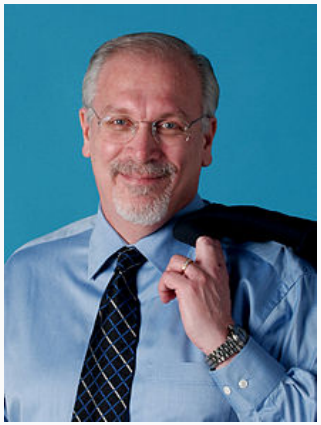
letters to the editor

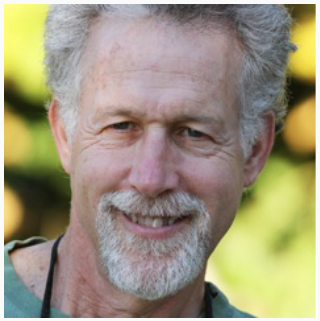
DOI:10.1145/1666420.1666422

Too Much Debate?

Richard A. DeMillo and Richard J. Lipton, Atlanta, GA

In his editor's Letter "More Debate, Please!" (Jan. 2010), Moshe Y. Vardi made a plea for controversial topics on these pages, citing a desire to "let truth emerge from vigorous debate." preliminary version was accepted by a highly selective conference program committee in 1976—predating by more than a year the article by Amir Pnueli that Vardi criticized us for not citing—and its presentation was attended by





Rigorous documentation anyone?

Software documentation is disliked by almost everyone.

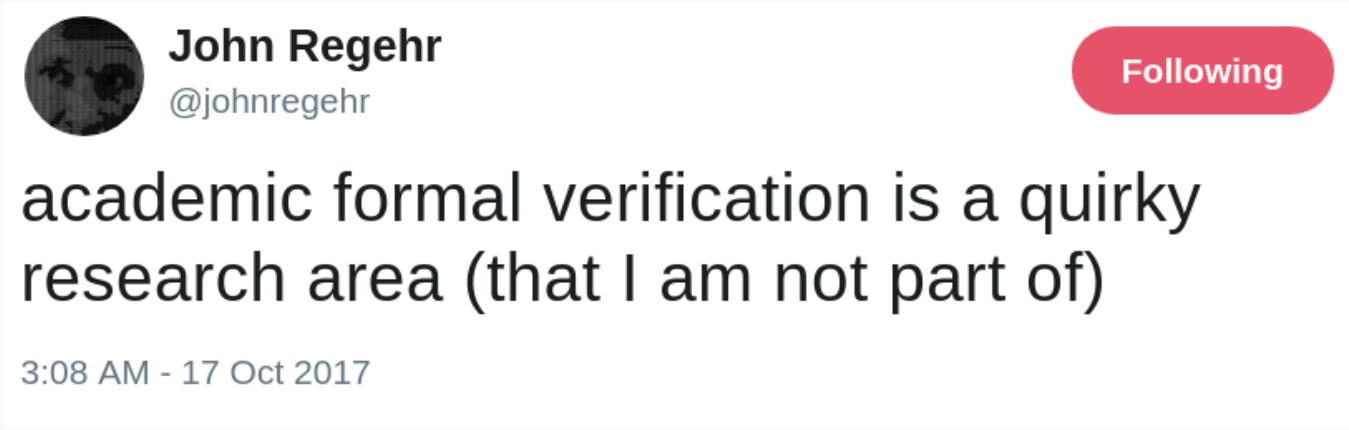
David Lorge Parnas

Precise Documentation: The Key to Better Software

S. Nanz (ed.), *The Future of Software Engineering*,
DOI 10.1007/978-3-642-15187-3_8, © Springer-Verlag Berlin Heidelberg 2011

Formal verification anyone?

Academic formal verification is a quirky research area recovering slowly from a long illness of practical irrelevance.



Microsoft uses assertions

There is systematic use of assertions in some Microsoft components; [. . .] more assertions and code verifications means fewer bugs.

Amazon uses formal methods

Formal methods are surprisingly feasible for mainstream software development and give good return on investment.

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

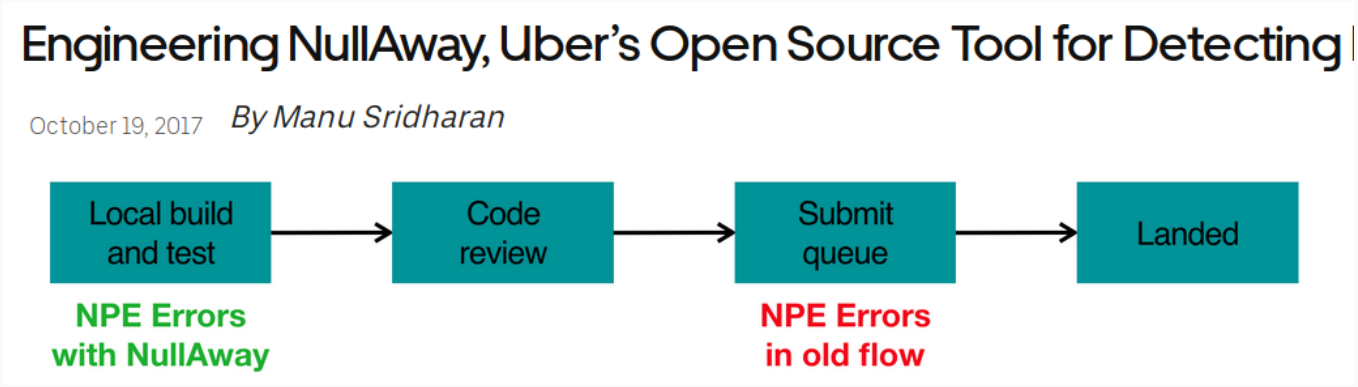
BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

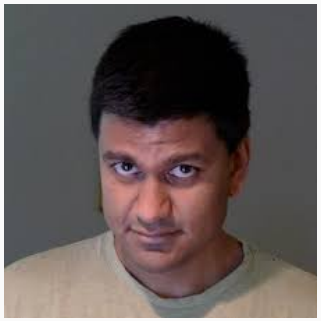
How Amazon Web Services Uses Formal Methods

COMMUNICATIONS OF THE ACM | APRIL 2015 | VOL. 58 | NO. 4

Uber uses static analysis

Our strategy at Uber has been
to use static code analysis tools
to prevent null pointer exception crashes.





Facebook uses static analysis

Each month, hundreds of potential bugs identified by Facebook Infer are fixed [. . .] before they are [. . .] deployed to people's phones.

theguardian

Facebook buys code-checking Silicon Roundabout startup Monoidics

MONOIDICS

facebook

Software analysis today

In the last decade (or so), formal software analysis techniques have gone from being theoretical research interests to delivering practical cost-effective tools.

Software analysis today

In the last decade (or so), formal software analysis techniques have gone from being theoretical research interests to delivering practical cost-effective tools.

What changed:

- software ever more complex
- verification tools ever more efficient
- combination of analysis techniques
- goals more focused, and promises less lofty

Software ever more complex

Without these characteristics – speed, scale and low friction – supporting [the conversion of News Feed to multi-threaded], where $>$ 1000 *issues were addressed before code was placed in a concurrent context, would not have been viable.*

Facebook Infer for race detection

Software ever more complex

The most important thing I have done as a programmer in recent years is to aggressively pursue static code analysis.

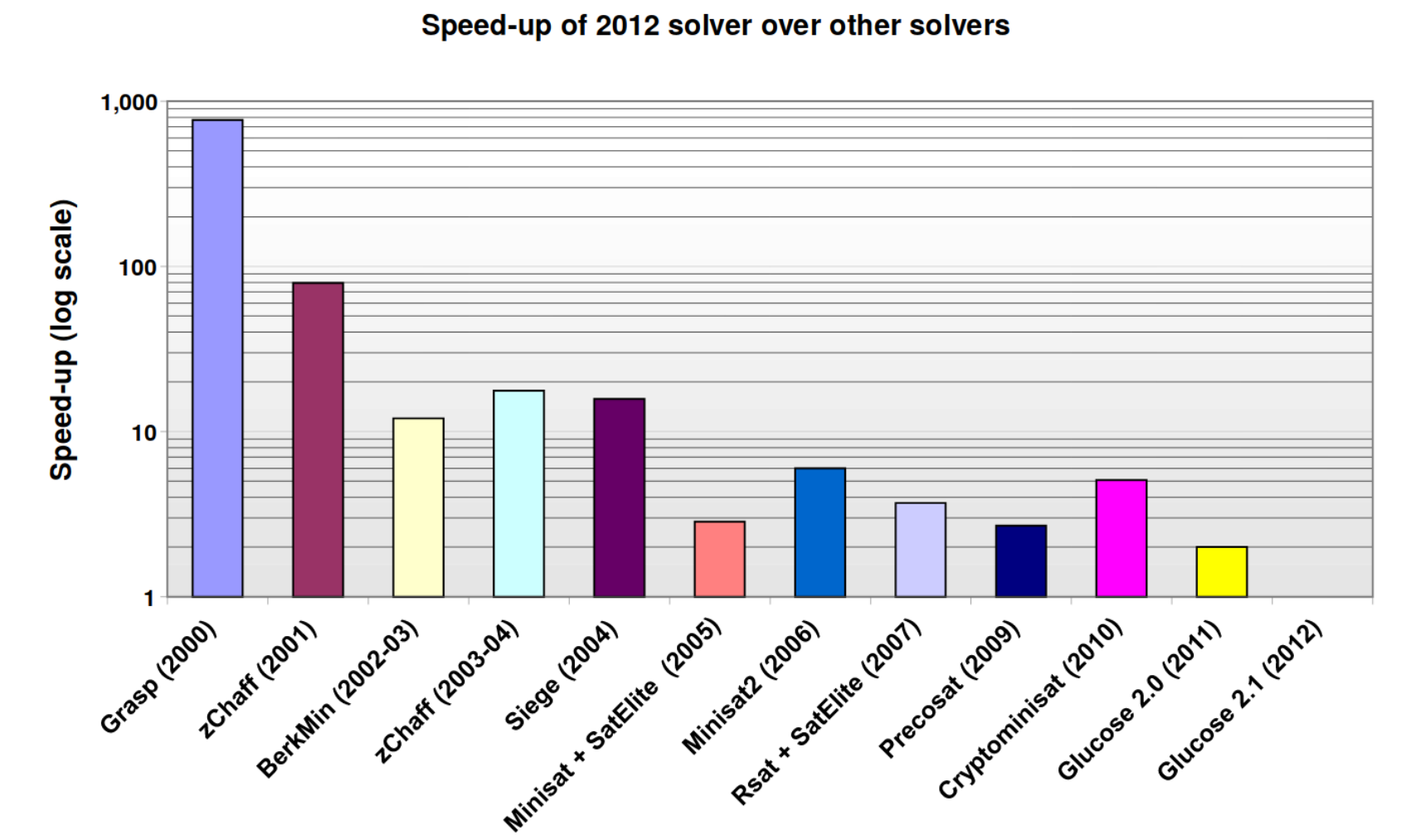
John Carmack

Software ever more complex

People care more about writing software that works.

Simon Peyton Jones

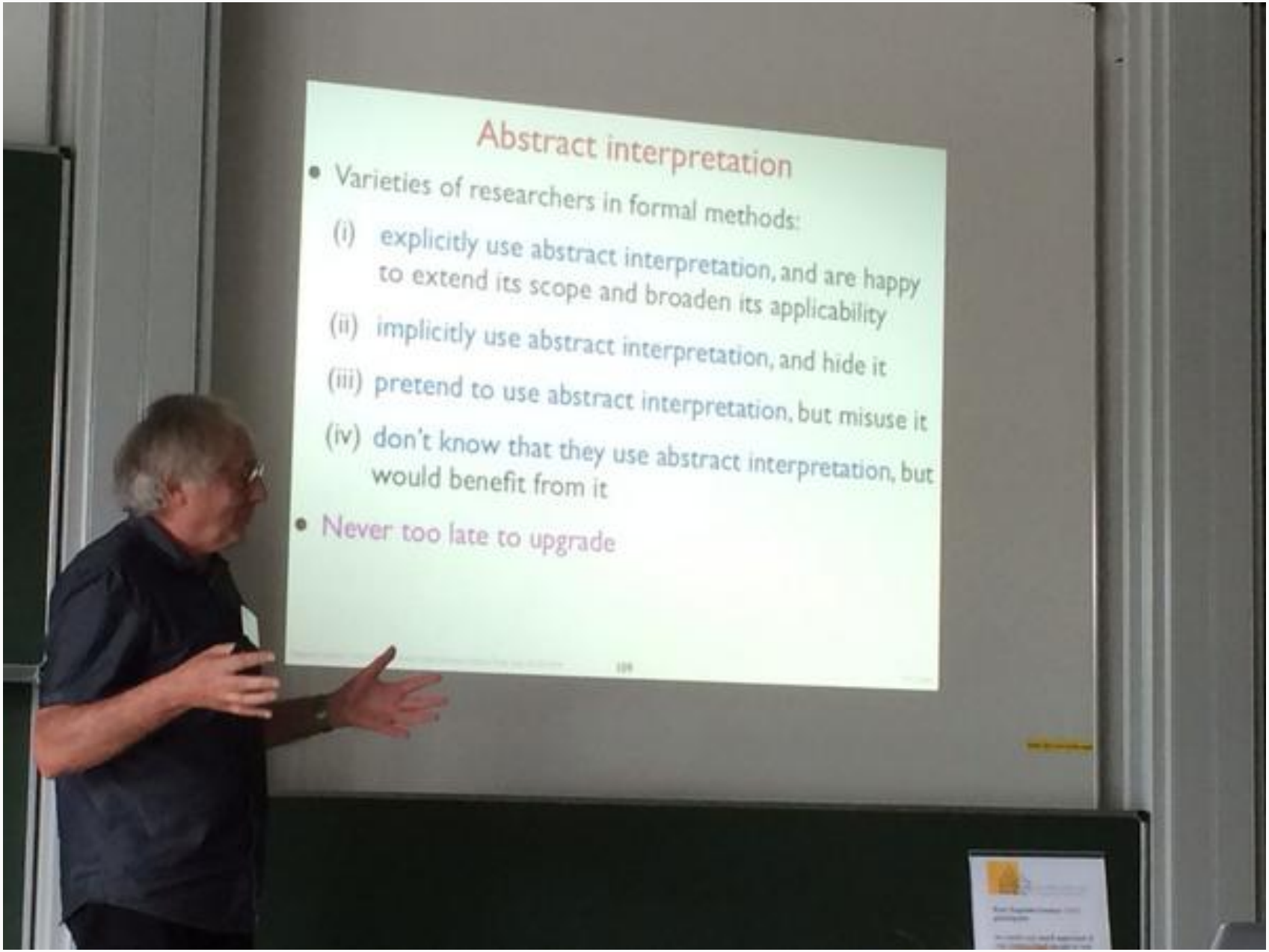
Tools ever more efficient



From Vardi's talk "The SAT revolution", 2015

Tools ever more efficient

Combination of analysis techniques



Combination of analysis techniques

Nowadays, it is common to combine heterogeneous techniques:

> *Modern tools simultaneously perform analyses traditionally classified as theorem proving, or model checking, or dataflow analysis.*
>
> *Jhala & Majumdar, 2009*

Goals more focused

From Dijkstra's programming = correctness proofs:

> *The programmer's task is not just to write down a program, but [their] main task is to give a formal proof that the program [they] propose meets the equally formal functional specification.*
>
> *[Programming] students [should be] protected from the temptation to test their programs.*
>
> *Dijkstra, 1988*

Goals more focused

To testing is verification too:

> *Formal methods include specification, verification, and testing techniques. [. . .]*
>
> *Understanding the limitations of formal methods is not less important than presenting their success stories. [. . .]*
>
> *Methods that attempt to guarantee correctness are treated with suspicion, while methods that are aimed at finding errors are preferred.*
>
> *Peled: "Software reliability methods", Springer, 2001*

Our motivation

Formal software analysis is not a panacea (silver bullet),
but does help improve software.

Our motivation

Formal software analysis is not a panacea (silver bullet),
but does help improve software.

Not perfect, but better!

What is software analysis?

Software analysis: a definition

"Software analysis" denotes techniques, methods, and tools useful to establish that some software behaves according to some properties.

Software analysis: a definition

"Software analysis" denotes techniques, methods, and tools useful to establish that some software behaves according to some properties.

techniques: notations and algorithms
(first-order logic, dataflow analysis, deductive verification, . . .)

methods: principled ways of applying techniques
(design by contract, invariant methods, B method, . . .)

tools: implementing and supporting the application of techniques and methods
(model-checkers, theorem provers, . . .)

Software analysis: a definition

"Software analysis" denotes techniques, methods, and tools useful to establish that some software behaves according to some properties.

behavioral properties: properties of software at runtime (while it is running)

Properties we're interested in (Yay! 👍):

- method `m` always terminates
- if the input is positive number, the program returns its inverse
- variable `x` always stores value `0`
- this input runs into a race condition
- the program crashes with input `3`
- in 80% of the runs, this statement executes in ≥ 10 ms

Properties we're not interested in (Nay! 👎):

- there are no loops in method `m`
- the code is indented using tabs
- there are 3 lines of comments for every executable line
- each class has at least 3 subclasses
- this class only has visible attributes

Software analysis: a definition

"Software analysis" denotes techniques, methods, and tools useful to establish that some software behaves according to some properties.

establish: determine with certainty

The properties we establish may be weak:

- all variables may overflow
- only variable x may overflow (the others won't)

The properties we establish may depend on strong conditions:

- this sorting procedure works correctly on lists of up to 2 elements

Many names, many voices

This course spans several areas that are quite similar:

Program analysis: software analysis

Static analysis: the analysis is static (on source code) but the properties are dynamic

Formal analysis: establishes properties based on rigorous, mathematical semantics

Verification: establishing that a program behaves as expected (in general or with testing)

Validation: verification of semi-formal or high-level models

Formal methods: formal specification, analysis, development, verification, and synthesis

Correctness

Correctness as a property

"Software analysis" denotes techniques, methods, and tools useful to establish that some software behaves according to some properties.

Correctness as a property

"Software analysis" denotes techniques, methods, and tools useful to establish that some software behaves according to some properties.

Correctness is a key property that software should have.

Is this program correct?

```
int proc(int x)
{
  return x + 1;
}
```

Is this program correct?

```
int proc(int x)
{
  return x + 1;
}
```

- What programming language is `proc` written in?

Is this program correct?

```
int proc(int x)
{
  return x + 1;
}
```

- What programming language is `proc` written in?

  C: it is syntactically correct

Is this program correct?

```
int proc(int x)
{
  return x + 1;
}
```

- What programming language is `proc` written in?

  C: it is syntactically correct

  Java: it is syntactically correct if it is part of a class declaration

Is this program correct?

```
int proc(int x)
{
  return x + 1;
}
```

- What programming language is `proc` written in?

  C: it is syntactically correct

  Java: it is syntactically correct if it is part of a class declaration

  Lisp: it is syntactically incorrect

Is this program correct?

```
int proc(int x)
{
  return x + 1;
}
```

- What programming language is `proc` written in?

  C: it is syntactically correct

  Java: it is syntactically correct if it is part of a class declaration

  Lisp: it is syntactically incorrect

- What should `proc` do?

Is this program correct?

```
int proc(int x)
{
  return x + 1;
}
```

- What programming language is `proc` written in?

  C: it is syntactically correct

  Java: it is syntactically correct if it is part of a class declaration

  Lisp: it is syntactically incorrect

- What should `proc` do?
  - `proc` should not throw `FileNotFoundException`: correct

Is this program correct?

```
int proc(int x)
{
  return x + 1;
}
```

- What programming language is `proc` written in?

  C: it is syntactically correct

  Java: it is syntactically correct if it is part of a class declaration

  Lisp: it is syntactically incorrect

- What should `proc` do?
  - `proc` should not throw `FileNotFoundException`: correct
  - `proc` should return `0`: incorrect (except for $x = -1$)

Is this program correct?

```
int proc(int x)
{
  return x + 1;
}
```

- What programming language is `proc` written in?

  C: it is syntactically correct

  Java: it is syntactically correct if it is part of a class declaration

  Lisp: it is syntactically incorrect

- What should `proc` do?
  - `proc` should not throw `FileNotFoundException`: correct
  - `proc` should return `0`: incorrect (except for $x = -1$)
  - `proc` should return an integer: correct (but what about overflows?)

Is this program correct?

```
int proc(int x)
{
  return x + 1;
}
```

- What programming language is `proc` written in?

  C: it is syntactically correct

  Java: it is syntactically correct if it is part of a class declaration

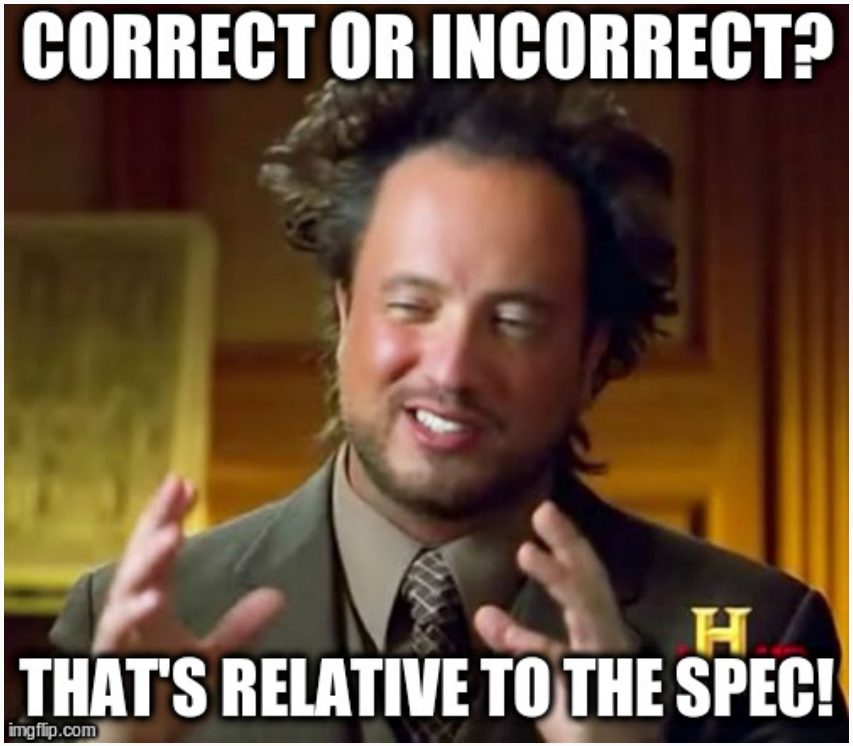
  Lisp: it is syntactically incorrect

- What should `proc` do?
  - `proc` should not throw `FileNotFoundException`: correct
  - `proc` should return `0`: incorrect (except for $x = -1$)
  - `proc` should return an integer: correct (but what about overflows?)
  - `proc` should return a positive number: correct if called with $0 \leq x < \texttt{Integer.MAX\_VALUE}$

Correctness is relative

Correctness is a relative property:

- a piece of software is correct relative to a specification of its intended behavior
- correctness = implementation and specification are consistent



Correctness is relative

Correctness is a relative property:

- a piece of software is correct relative to a specification of its intended behavior
- correctness = implementation and specification are consistent

The specification may be implicit or explicit:

```
int proc(int x) { return x + 1; }
```

Implicit:

- type correctness
- termination
- no overflows
- no memory leaks
- no race conditions

Explicit:

- restrictions on input
- guarantees on output
- effects on the state
- non-functional properties: timeliness, memory usage, ...

Verification vs. validation

In the context of software engineering processes there's often a distinction between:

verification is internal: are we building the software right (with respect to a specification)?

validation is external: are we building the right software (with respect to requirements)?

In this course we will generally use verification (and specification) to cover both.

Software quality assurance

Software analysis is an important part of the more general activities of software quality assurance:

- Define software quality for your project
- Define policies and processes to achieve quality
- Assess quality and find defects
- Improve quality

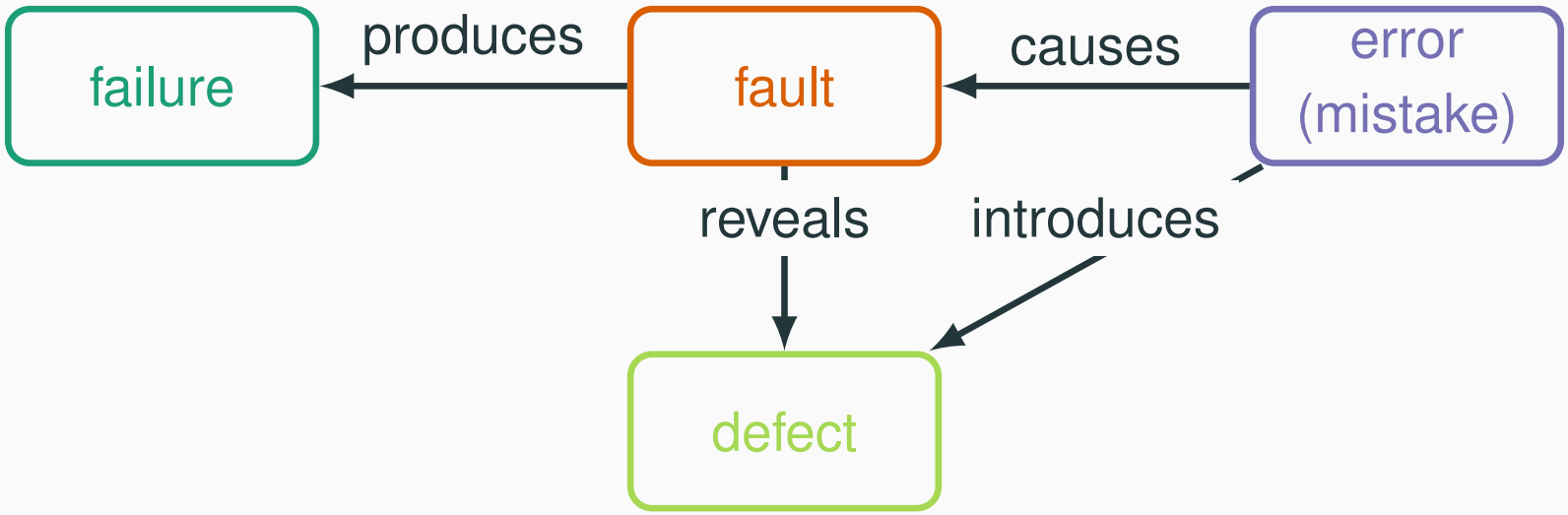
Bugs

Conversely, quality is the absence of bugs or defects.

Bugs

Conversely, quality is the absence of bugs or defects.

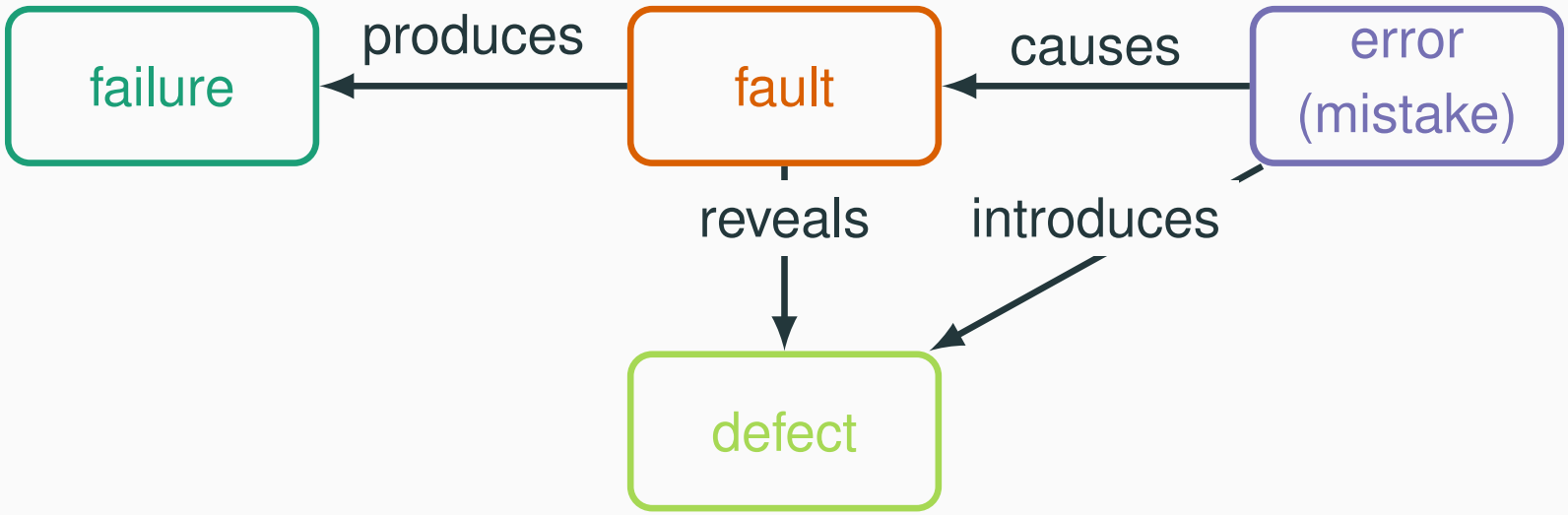
A more precise classification from the IEEE Computer Society:



Bugs

Conversely, quality is the absence of bugs or defects.

A more precise classification from the IEEE Computer Society:

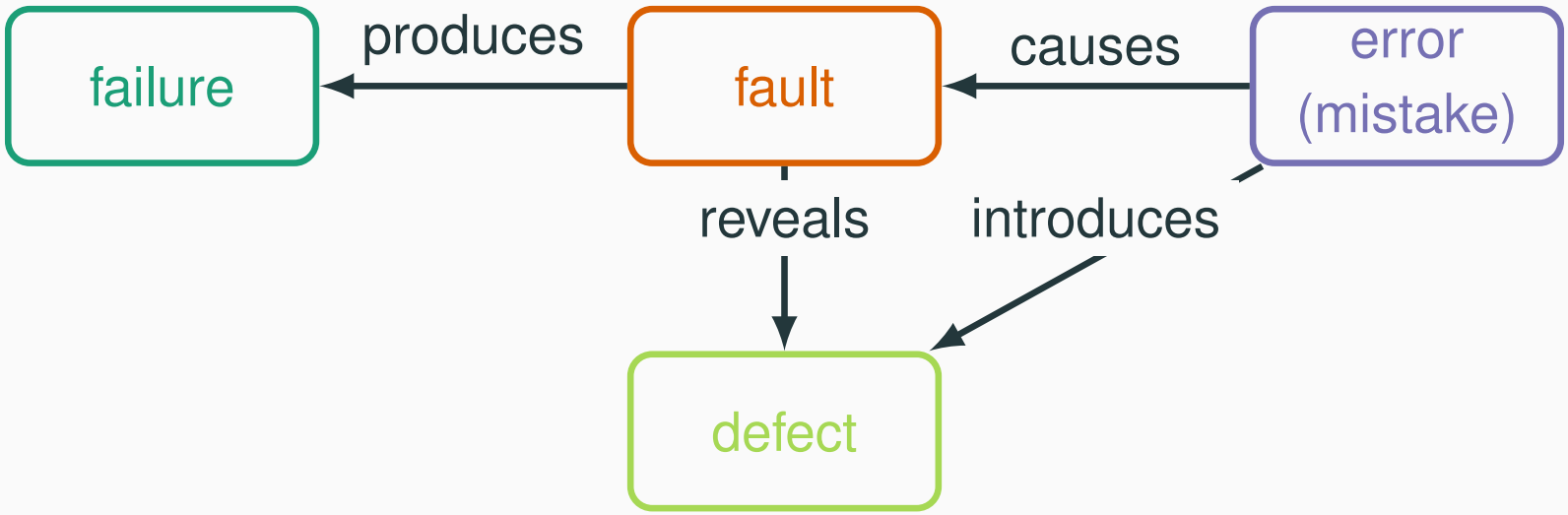


failure: event where program execution cannot continue
(“uncaught exception”, “division by zero”, . . .)

Bugs

Conversely, quality is the absence of bugs or defects.

A more precise classification from the IEEE Computer Society:

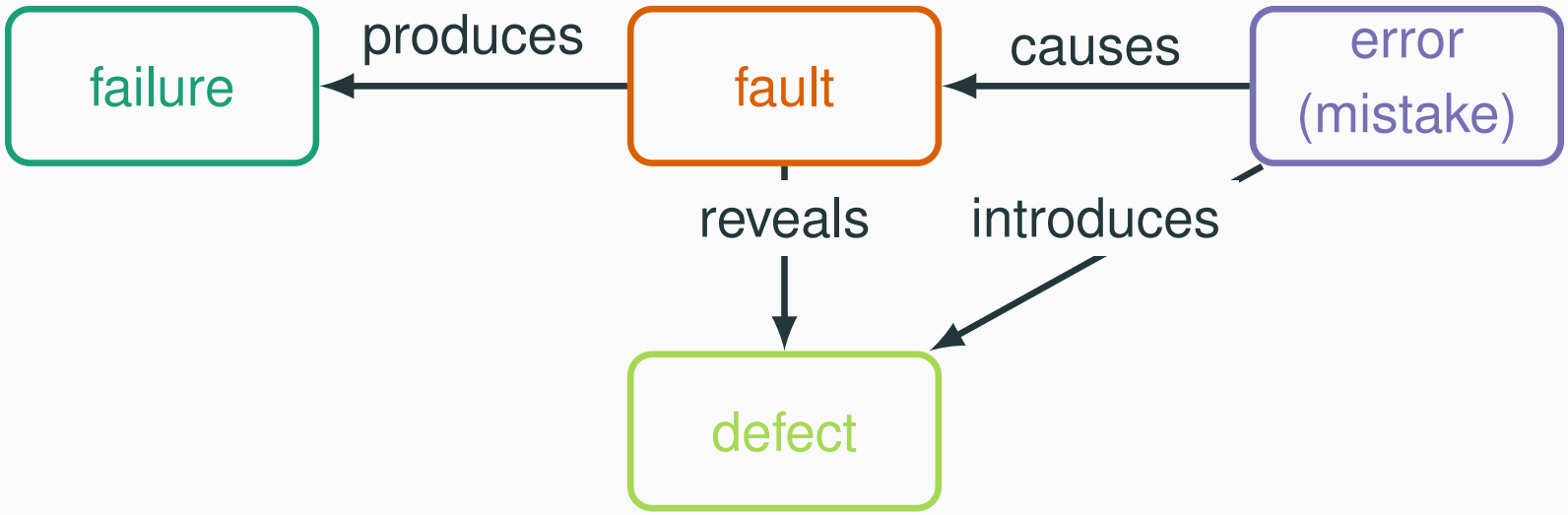


fault: manifestation of an error
(“the incorrect value is computed”, “a variable is not initialized”, . . .)

Bugs

Conversely, quality is the absence of bugs or defects.

A more precise classification from the IEEE Computer Society:

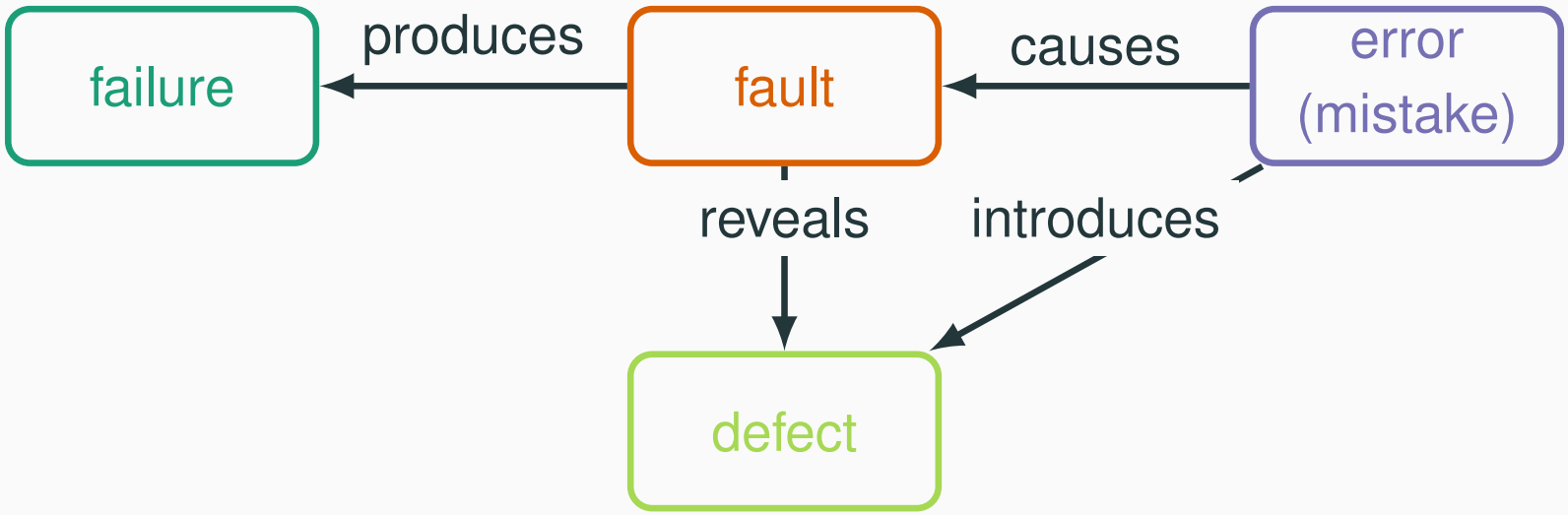


error: human action that introduced an incorrect result
(“any programming mistakes”)

Bugs

Conversely, quality is the absence of bugs or defects.

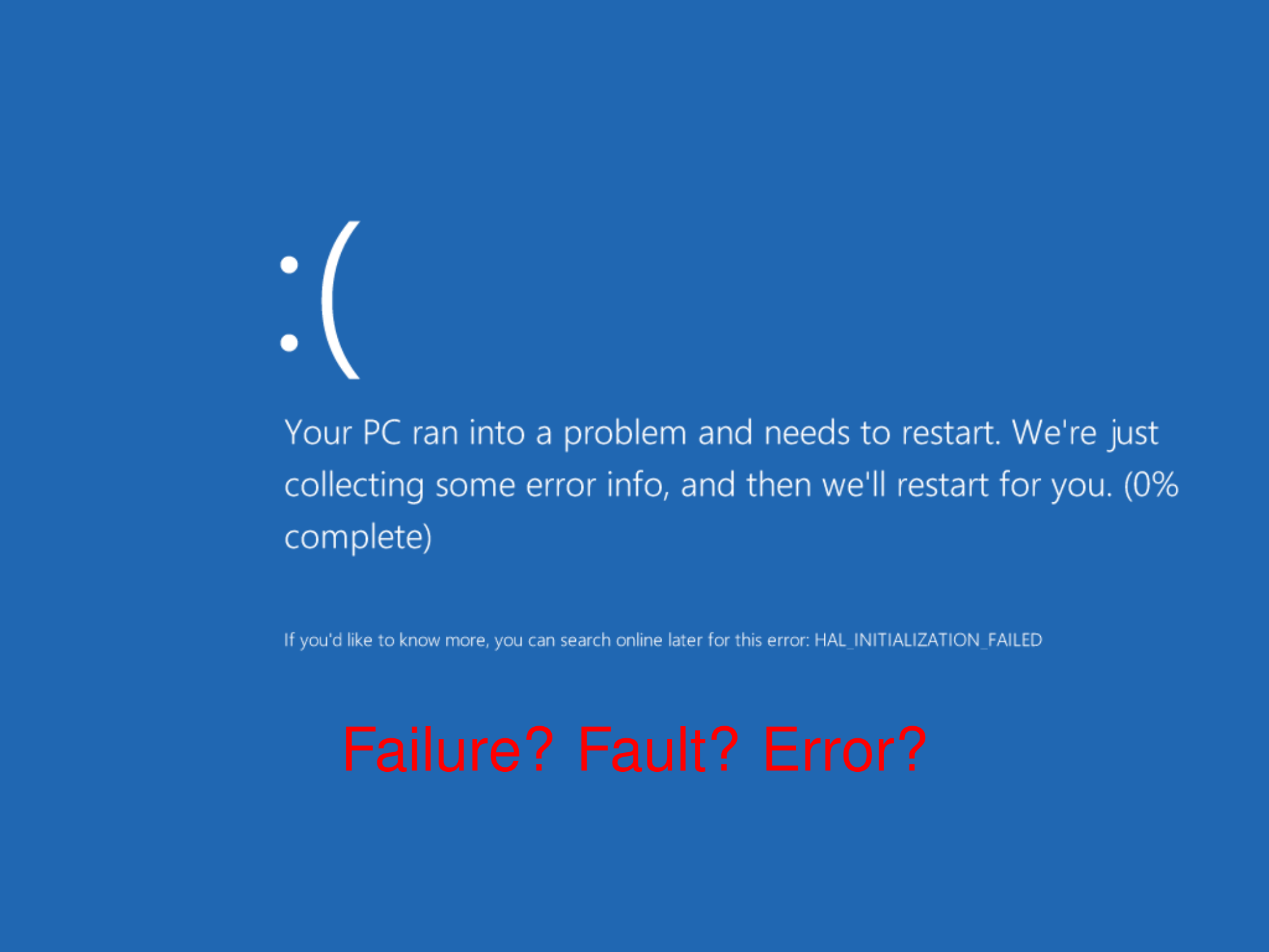
A more precise classification from the IEEE Computer Society:



defect: an imperfection or deficiency in a program
("this function should always return a positive value, but returns a negative value in this case")

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you. (0% complete)

If you'd like to know more, you can search online later for this error: HAL_INITIALIZATION_FAILED


:(
Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you. (0% complete)
If you'd like to know more, you can search online later for this error: HAL_INITIALIZATION_FAILED
Failure? Fault? Error?

Course outline

Software analysis at a glance

1. Introduction (this lecture)
2. Concepts of logic and computation
3. Software analysis: the very idea
4. Deductive verification
5. Static analysis
6. Model-checking & predicate abstraction
7. Symbolic execution
8. Dynamic analysis

Concepts of logic and computation

- Logic
  - Propositional logic: syntax, semantics, and complexity
  - Predicate logic: syntax, semantics, and complexity
  - Logic theories: decidability and complexity
- Computation
  - Decidability
  - Computational complexity
  - Complexity classes: P, NP, EXPTIME, PSPACE, ...

Software analysis: the very idea

- Soundness and completeness of analysis
- Trade-offs: soundness, expressiveness, and automation
- Helium: a small imperative language and its semantics

Deductive verification

- Hoare triples and logic
- Preconditions and postconditions
- Predicate transformers
- Termination analysis
- Arrays and aliasing
- Procedures and modular reasoning
- Reasoning about objects
- Separation logic

Static analysis

- Data-flow analysis
- Abstract interpretation
- Type systems

Model checking

- Finite-state automata and temporal logic
- Automata-based model checking
- Predicate abstraction & software model checking
- Real-time verification with model checking

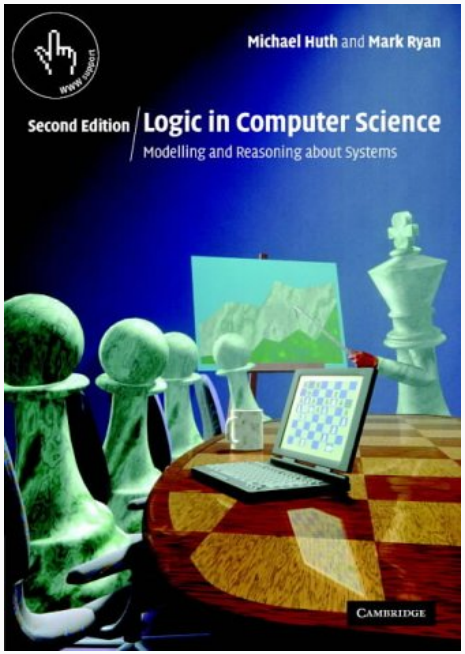
Symbolic execution

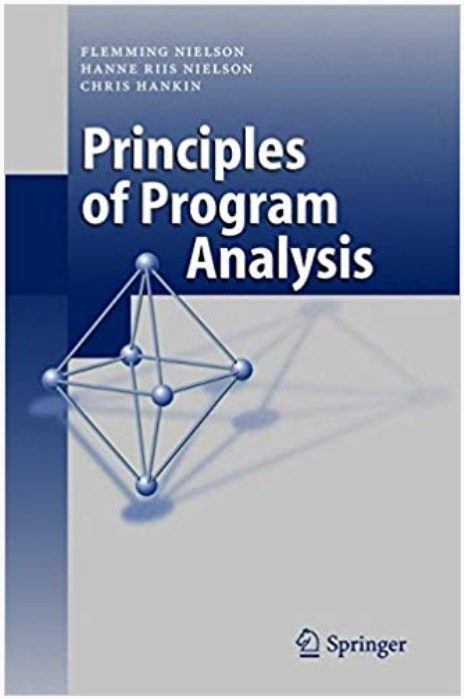
- Classic symbolic execution
- Dynamic symbolic execution

Dynamic analysis

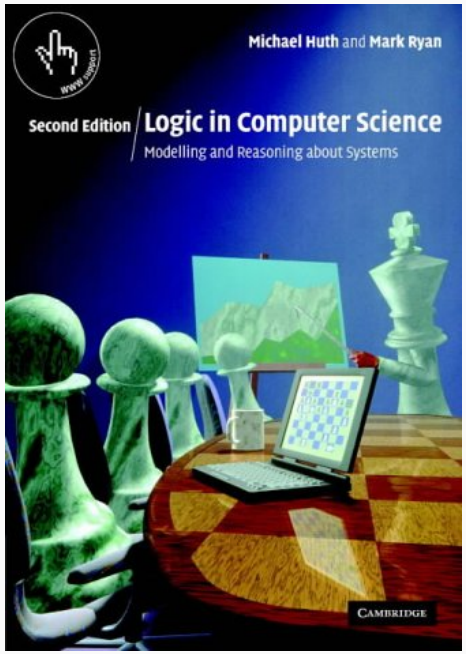
- Runtime assertion checking
- Dynamic invariant mining
- Dynamic debugging techniques

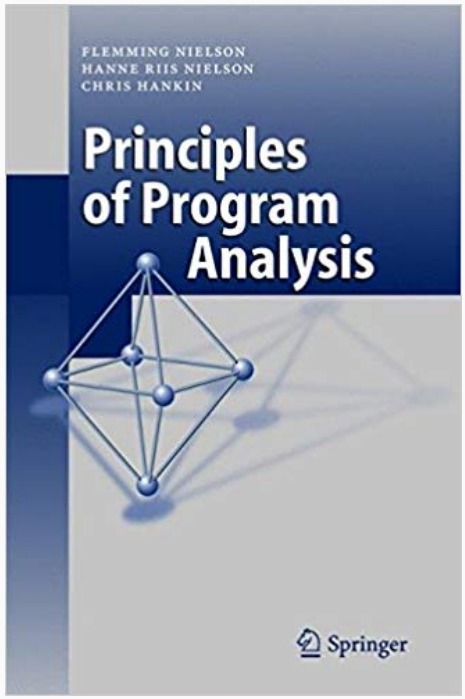
Textbooks

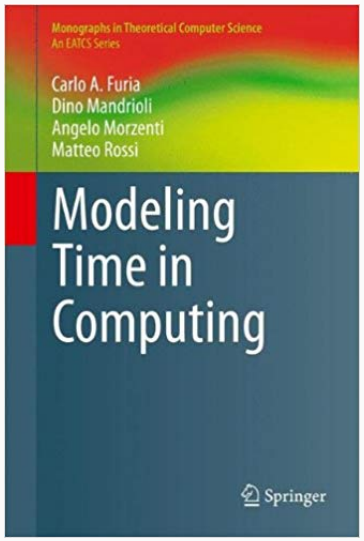

Michael Huth and Mark Ryan
Second Edition
Logic in Computer Science
Modelling and Reasoning about Systems
CAMBRIDGE

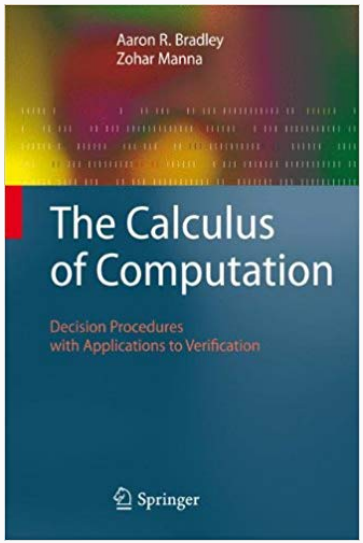

FLEMMING NIELSON
HANNE RIIS NIELSON
CHRIS HANKIN
Principles of Program Analysis
Springer

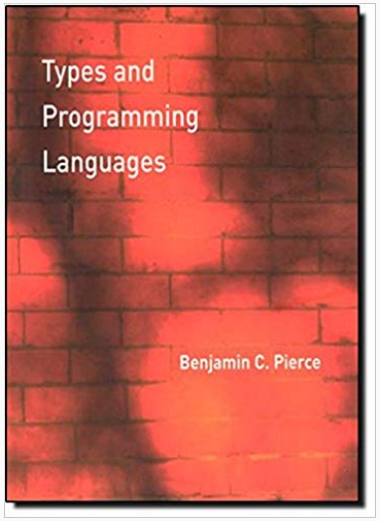
Textbooks

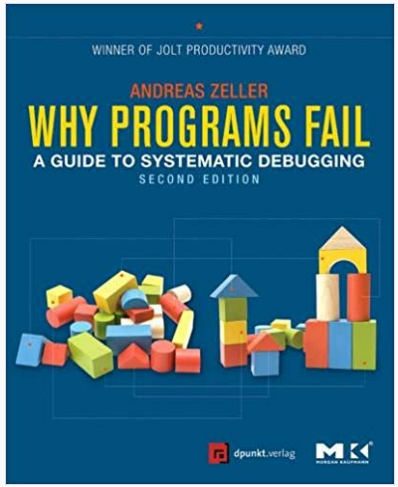

Michael Huth and Mark Ryan
Second Edition
Logic in Computer Science
Modelling and Reasoning about Systems
CAMBRIDGE


FLEMMING NIELSON
HANNE RIIS NIELSON
CHRIS HANKIN
Principles of Program Analysis
Springer


Monographs in Theoretical Computer Science
An EATCS Series
Carlo A. Furia
Dino Mandrioli
Angelo Morzenti
Matteo Rossi
Modeling Time in Computing
Springer

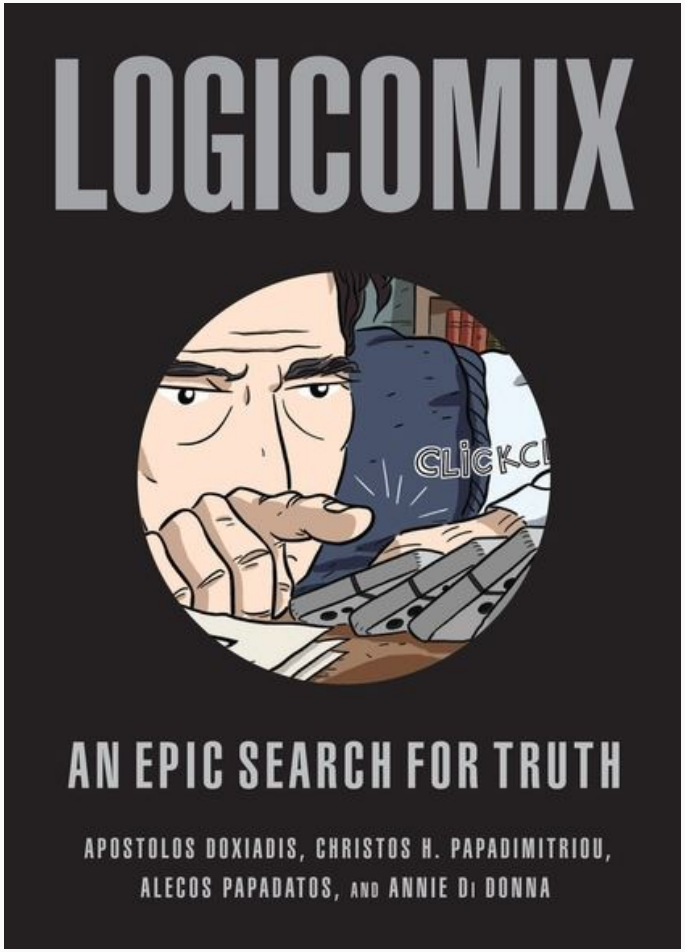

Aaron R. Bradley
Zohar Manna
The Calculus of Computation
Decision Procedures with Applications to Verification
Springer


Types and Programming Languages
Benjamin C. Pierce


WINNER OF JOLT PRODUCTIVITY AWARD
ANDREAS ZELLER
WHY PROGRAMS FAIL
A GUIDE TO SYSTEMATIC DEBUGGING
SECOND EDITION
dpunkt.verlag

For inspiration

Vardi's talk "And Logic Begat Computer Science"



Summary

Summary

In the last decade, rigorous software analysis has gone from theoretical results to practical, cost effective techniques and tools.

The main goal of software analysis is checking properties of programs – in particular, correctness of an implementation with respect to a specification.

These slides' license