# Formal Software Development Methods

Instructor: Madhusudan Parthasarathy (call me Madhu)

madhu@illinois.edu

Siebel Center 3226

https://madhu.cs.illinois.edu/

Fall 2026

# Course Objectives

- provide an overview of mathematical models, languages, and methods for software specification, development, and verification
- we will also cover some of the latest trends including using AI + formal methods for software, in particular:
  - Formal specifications in the realm of AI-based programming
  - Lean proofs using AI to quickly prove soundness of analysis, proof systems, etc.
- 3 undergraduate hours.
- 4 graduate hours.
- Prerequisites: CS 225; CS 374 or MATH 414.
- Useful: CS 421; CS 475

# Course Details

| Name | Office Hours | Email |
|---|---|---|
| Madhusudan (Madhu)<br>(Siebel 3226) | Tue 3pm-4pm starting next week | madhu@illinois.edu |
| Nausheen Mohammed<br>(most likely; half TA) | Fri 11am-12pm | |

**Course website**: https://courses.grainger.illinois.edu/cs477
**Gradescope code**: TBA (used for homework submissions + Github)

**Piazza discussion forum**: https://piazza.com/illinois/fall2026/cs477ece478, Access Code: hoare

**Lecture recordings**: TBD

# Lectures

- Most lectures will be in-person in Urbana-Champaign,

- A few lectures may be online (if traveling)

# Grading Scheme (subject to small changes)

| Task | % of the grade |
|---|---|
| Attendance | 10% |
| Homeworks, approx 6, could be more/less, no dropped assignments; subject to academic integrity (more on this later);<br>May be graded for attempt and for major feedback only, depending on staffing constraints; may be peer graded | 40% |
| Exams (midterm+final) | 50%<br>(likely 20%+30% ) |
| Students registered for 4 credits need to do a course project either individually or in groups of 2 | 25 percent points extra, averaged |

# Homeworks

- The assignments will both have theoretical components (e.g., proofs) and implementations (code). Some homeworks maybe more theoretical and others maybe more implementation heavy
- Between 1- and 2-week time will be given to solve the homework
  - **you can collaborate with *one* person at most on your homework, but final writing must be individual and each homework must be submitted individually**
  - homeworks must be submitted electronically via gradescope
  - scanned (or photographed) answers are OK as long as the handwriting is legible
- No late homeworks (ask in advance if you have emergencies/conference travel to get grace time)

# Semester Project (for 4 credits)

- Projects may be done individually or in groups of 2.
- An abstract (1 to 2 pages) describing the project is due before the October 15.
- You must talk and we must agree to the project before you begin!
- A list of projects will be posted; you can choose from this or propose your own, mid-semester
- Involves either
  - reading up a set of papers, and writing a report, or
  - research, theoretical on a particular topic, or
  - research/engineering a particular technique and submitting a write-up
- Towards the end of the semester:
  - may involve project presentation in the last two weeks of the semester
  - project report (8 pages) due by last day of course

# Tentative Grading Scale

| Grade | Score |
|---|---|
| A+ | [95,100] |
| A | [90,94] |
| A- | [85,89] |
| B+ | [80,84] |
| B | [75,79] |
| B- | [70,74] |
| C+ | [65,69] |
| C | [60,64] |
| C- | [55,59] |
| D | [50, 54] |
| F | [0,49] |

Grades may be curved if the spread of points is too high
The curve *may* be separate for undergrads and grads
(we may not use separate curves if undergrads do as well as grads!)

# Academic Integrity

- Any form of cheating/copying/paraphrasing will not be tolerated
- Unless when explicitly instructed, the use of Large Language Model (LLM) tools, including ChatGPT, is **strictly prohibited** in the completion of any coursework and assignments in this class
- Clearly give your sources and help (LLMs, content on web, etc.)
- Instructors/TAs may ask a student to explain their solutions to homework exercises at any time.

# Course References

## No required textbook

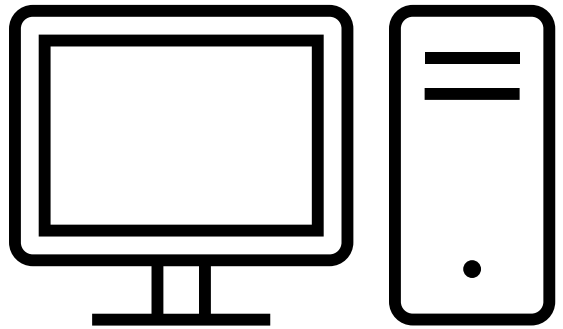
- But many existing resources available (+ online)

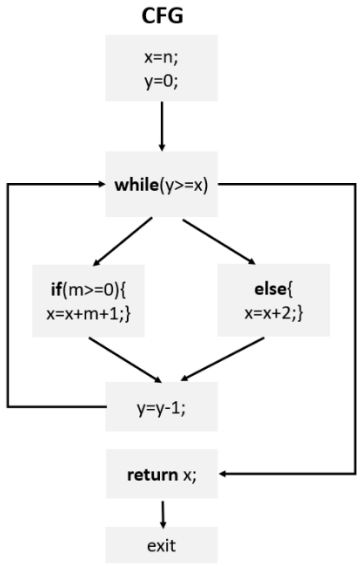
## Reference Books:

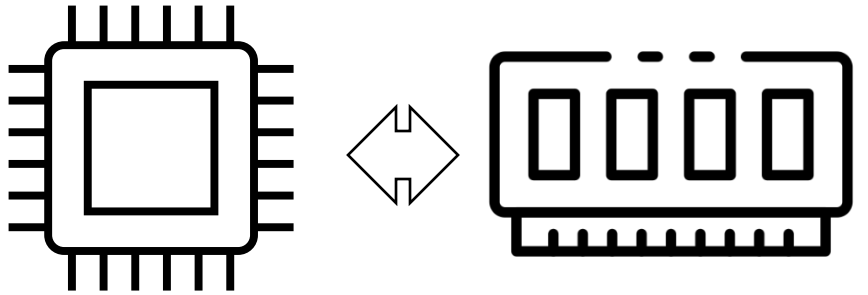
- Principles of Static Analysis: F. Nielson, H. Nielson, Hankin, Springer 2005
- The Calculus of Computation: Bradley and Manna, Springer 2007
- Introduction to Static Analysis, Rival and Yi, MIT Press 2021
- Principles of Model Checking: Baier and Katoen, MIT Press 2008
- Principles of Abstract Interpretation, Patrick Cousot, MIT Press, 2021

# From Real World to Formal Methods

```
int compute(int n, int m) {
   int x=n, y=0;
   while (y>=x){
        if(m>=0){
            x=x+m+1;
        }
        else{
            x=x+2;
        }
        y=y-1;
   }
   return x;
}
```

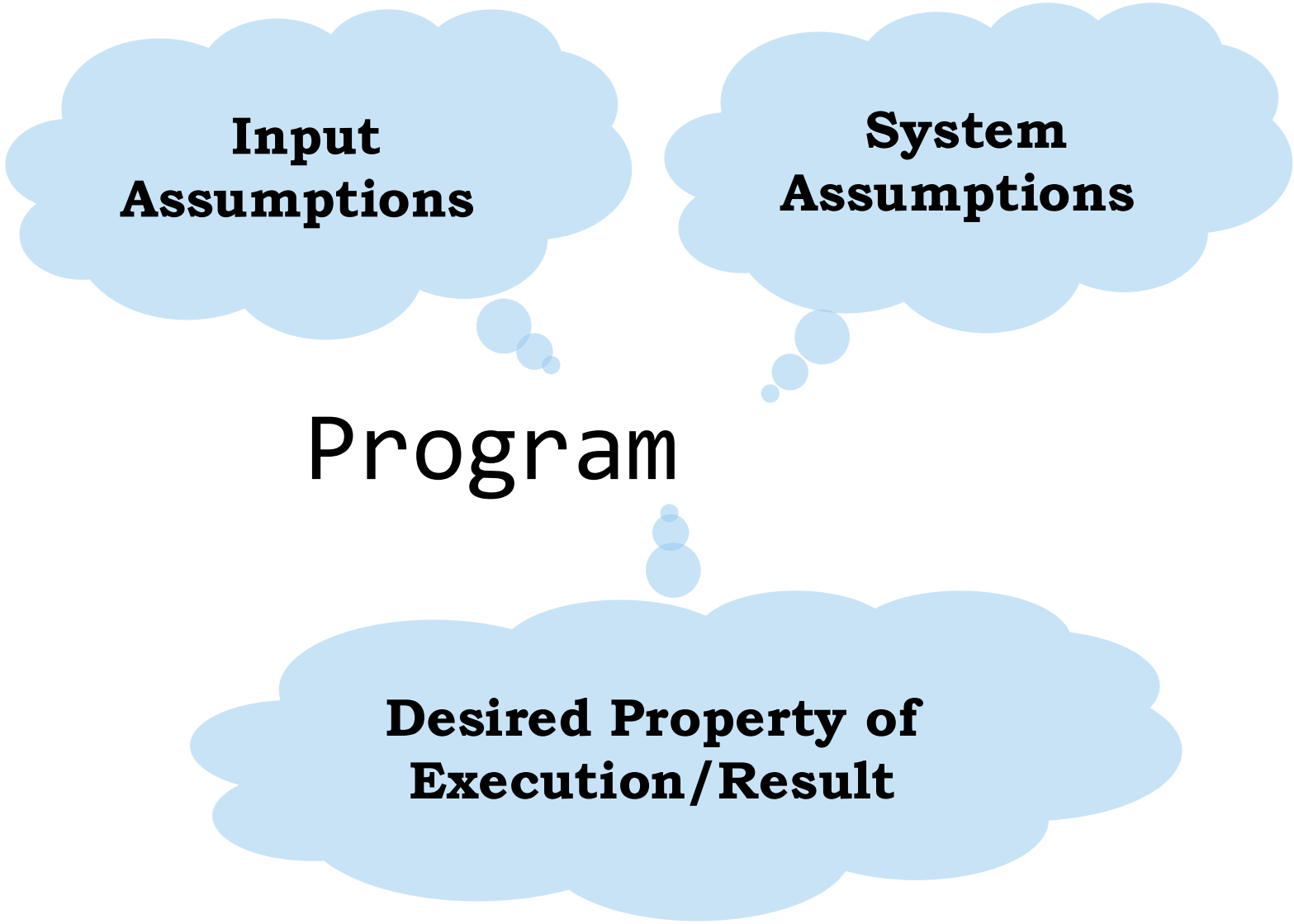




https://www.flaticon.com/free-icons/ram-memory

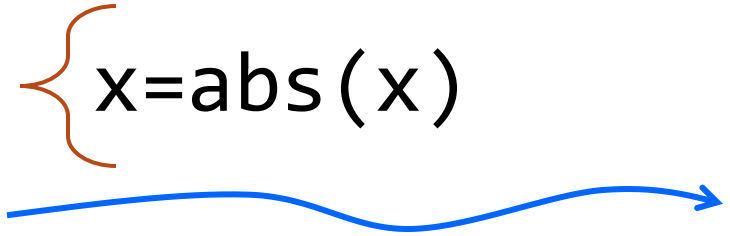
# What do we want to do?



# What do we want to do?

What assumptions do we need to make about inputs?
What assumptions do we need to make about libraries/hardware?

```
read(x)
y=sqrt(x)
return y
```




```
double sqrt_newtonmethod(double x) {
    if (x==0) return x;
    double div = x, val = x, last;
    do {
       last = val;
       val = (val + div / val) * 0.5;
    } while (abs(val - last) > 1e-9);
    return val;
}
```

Is the program going to experience an error (crash/buf overflow/div by 0)?
Is the program going to always terminate?
Is the program going to produce a correct/accurate result/behavior?

# What do we wa

What assumptions do we nee
What assumptions do we nee

```
read(x)
y=sqrt(x)
return y
```

```
x=abs(
```

https://mudongliang.github.io/x86/

x86 Instruction Set Reference

FSQRT

Square Root

| Opcode | Mnemonic | Description |
|---|---|---|
| D9 FA | FSQRT | Computes square root of ST(0) and stores the result in ST(0) |

Description

Computes the square root of the source value in the ST(0) register and stores the result in ST(0).

The following table shows the results obtained when taking the square root of various classes of numbers, assuming that neither overflow nor underflow occurs.

FSQRT Results

| Source (ST(0)) | Destination (ST(0)) |
|---|---|
| -inf | * |
| -F | * |
| -0 | -0 |
| +0 | +0 |
| +F | +F |
| +inf | +inf |
| NaN | NaN |

F Means finite floating-point value

* Indicates floating-point invalid-arithmetic-operand (#IA) exception.

Operation

ST(0) ← SquareRoot(ST(0));

FPU flags affected

C1 Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise. C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.
#IS Stack underflow occurred.
#IA Source operand is an SNaN value or unsupported format. Source operand is a negative value (except for -0).
#D Source operand is a denormal value.

Protected Mode Exceptions

#NE EM or TS in CR0 is set.

Real-Address Mode Exceptions

#NE EM or TS in CR0 is set.

Virtual-8086 Mode Exceptions

#NE EM or TS in CR0 is set.

| Instruction | Latency | Throughput | Execution Unit |
|---|---|---|---|
| FPU3D | DF3n/DF2n | DF3n/DF2n | DF2n |
| FSQRT_SP | 30/23 | 30/23 | FP_DIV |
| FSQRT_DP | 40/38 | 40/38 | FP_DIV |
| FSQRT_EP | 44/43 | 44/43 | FP_DIV |

Is the program going to exper
Is the program going to termi
Is the program going to produce a correct/accurate result?

# Theoretical tools (math/logic/algorithms)

- Sets, Graphs, Trees
- Algorithms (for automation)
- Automata and formal models of computing
- Logic and Proof theory, Temporal logics
- Induction, especially structural induction and well-founded induction, inductive relations
- Lattices, Galois Connections, Fixed Points of Functions

# The Advent of AI/LLMs

Programming is becoming automated

How does this change the scope of formal methods in software development?
Active area of research! Think about this course in this light!

- AI can write bad code; can we try to verify it?
- Specifications will be in NL:
  Humans need to still specify *what they want*.
  What are the new modes of specification? How do we formalize them? Test them? Verify them?
- AI can write annotations for code that help prove code;
  (unlike humans who complain!).
  How do we tap into this?
- AI can even prove programs correct.
  How do we build proof systems for them to use?

# Primary kinds of analysis/proofs

- Algorithmic analysis (for automation)
  - Can a tool find prove your program correct with respect to specs automatically?
  - Can a tool find errors in your program automatically?
  - Undecidability barriers
  - Compilers/type-systems already do a lot of such checks
  - Framework of abstract interpretation that unifies all kinds of analysis (based on least fixpoints on lattices)

# Primary kinds of analysis/proofs

- Specifications and Proof systems
  - Specification languages
    - First-order logic
    - Separation logic / heap logics
    - Temporal logics for reactive behaviors
    - Contract languages: contracts/class invariants/loop invs
      JML, SpecSharp, etc.
  - Proof systems: Hoare logic;
    formal proofs of programs

# Primary kinds of analysis/proofs

- Logics: Automating reasoning (entire course on this! CS474)
  - Proofs need not be done by humans (or even AI)
    We can *reliably automate reasoning in several domains*
  - Do logical statements always have proofs?
    Are they always decidable?
  - Decidable Logics
    - Classes of decidable logics
    - SAT/SMT solvers
    - How to integrate automated reasoning into proofs of programs; verification conditions

# What Kinds of Tools?

- Abstract interpreters, e.g. ASTREE, ELINA, ERAN  (fully automated)
- Program verification that's semi-automated using SMT solvers
  e.g., Dafny, Civl
- SAT/SMT solvers, e.g., Z3, CVC and their utility in both proving programs as well as finding errors
- Interactive Theorem Provers, e.g., Lean, Isabelle, Coq

Lots more we won't look at:

- Model-checkers
- Runtime monitoring

# Aims of this Course

**Theoretical:** The fundamental mathematics for reasoning about software

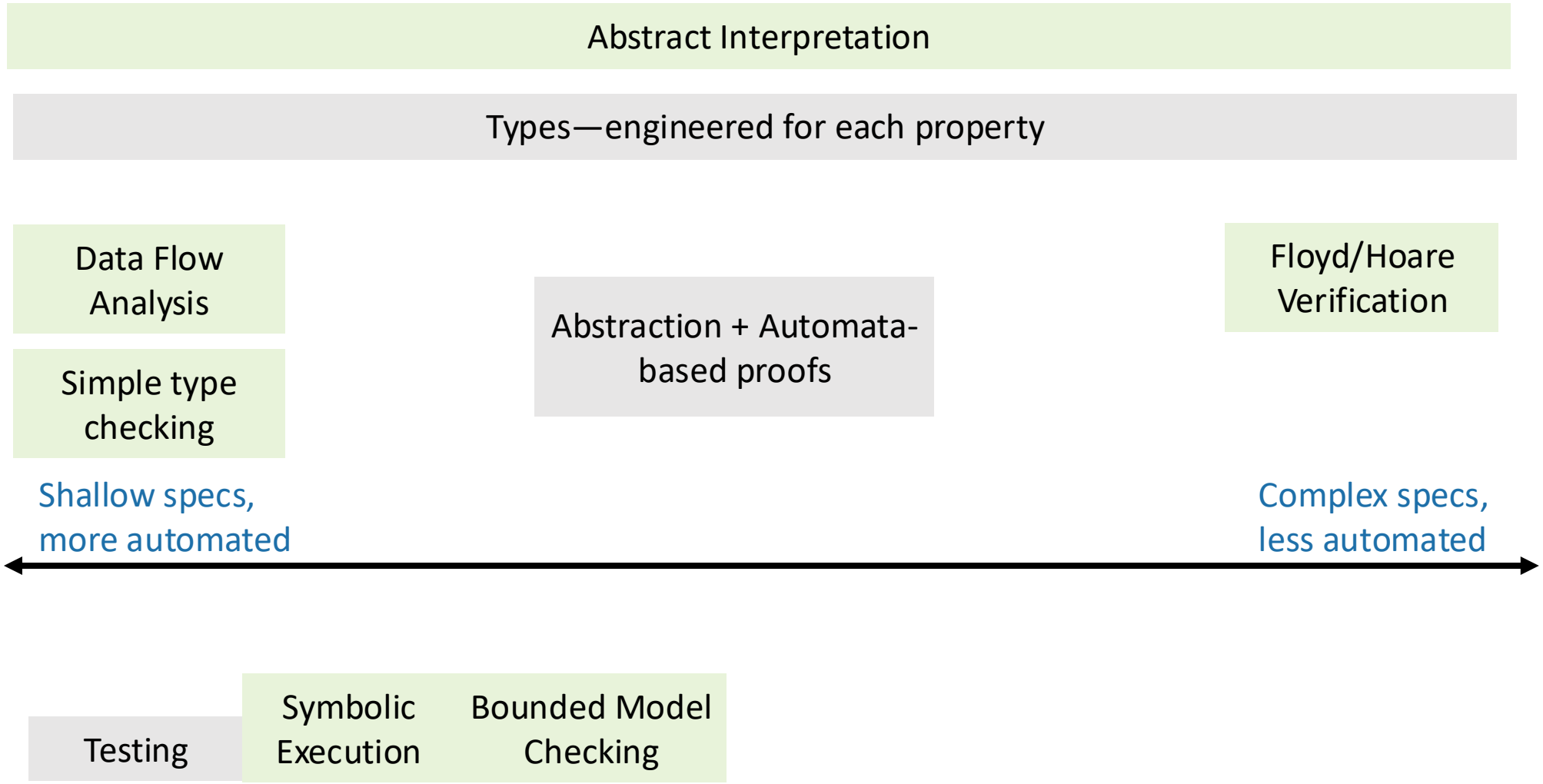
| |
|---|
| Formally specifications of software systems |
| Static analysis using abstraction; abstract interpretations. |
| Floyd-Hoare logic; contracts; pre/post conditions; inductive invariants verification conditions, strongest postcondition, weakest precondition |
| Contract-based programming; developing software using contracts. |
| Inherent limits of program verification; undecidability |
| Decidable and undecidable logics |

# Aims of this Course

**Practical:** Use existing tools or build your own, e.g.,

| |
|---|
| Build static analysis algorithms for analyzing software using abstract-interpretation tools |
| Proving small programs correct using a modern program verification tool (Floyd-style) |
| Use SMT solvers to solve logical constraints; understand how software verification can be done using these solvers |
| Learn contract-based programming using Dafny; use to generate unit tests and proofs |
| Undestand formal reasoning of software with advent of AI:<br>Specifications in precise NL that is formalizable; Tools that prove AI-written code correct; AI helping program verification using annotations and/or proofs in formal proof systems. |
| |

# Landscape of Formal Approaches



A problem has been detect and Windows has been shut down to prevent damage
to your computer.

THREAD_STUCK_IN_DEVICE_DRIVER

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restar
your computer, press F8 to select Advanced Startup options, and then
select Safe Mode.

Technical information:

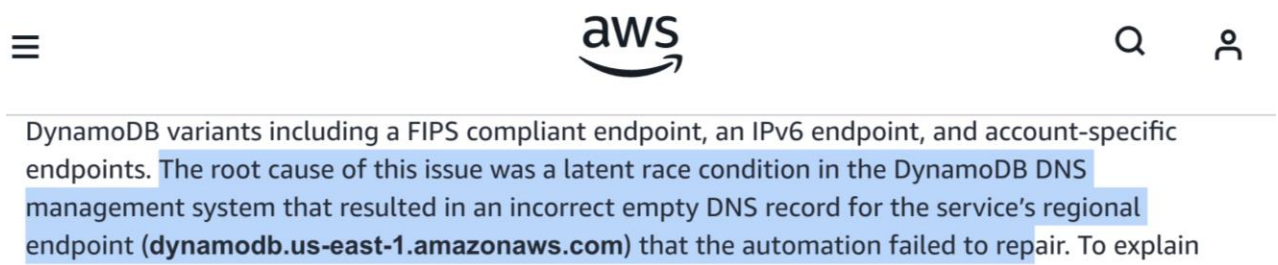
*** STOP: 0x0000000A (0x00000000, 0x00000000)

# And for everyone else…

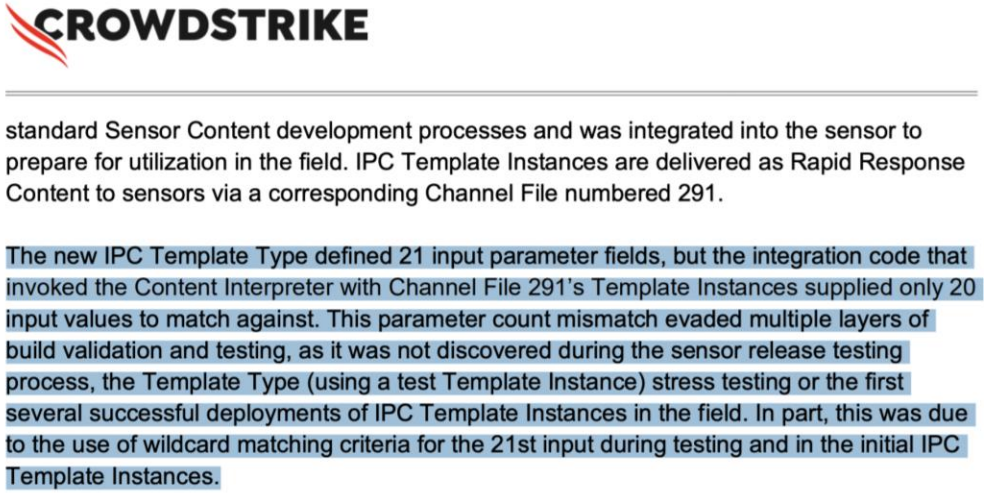
- "Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability." **Bill Gates, April 18, 2002. WinHec Keynote**
- https://www.microsoft.com/en-us/research/project/slam/

# The Software Challenge

- Software failures were estimated to cost the US economy about $2.4 trillion [CISQ]




AWS Outage October 2025




CrowdStrike incident July 2024

# The Software Challenge in the age of AI

- AI write lots and lots of code!
- Engineers need to deal with checking if this code is what they wanted, whether it is correct, secure, etc.
- AI has tendency to escape safety hatches!
- AI (e.g., Mythos) are getting much better at finding exploitable errors in code; nightmare scenarios predicted
- A new era of formal methods!
  - **Need:**
    AI written code *must be proven correct if we are to use it reliably*
  - **Methods:**
    AI can *help engineering correct code / write proofs*
  - **Changing landscape of problems:**
    New proof systems/mechanisms that are written for AI, not humans, e.g., more expressive type-systems that require a lot more annotations.

# Extra slides from previous editions of the course

# What are Formal Methods?

- Mathematical formalism for the specification, design, validation, and verification of software systems.
  - Consist of a large collection of techniques varying in complexity and usability.
- Not all methods are equally used.
  - Light-weight methods such as design by contract are common
- More rigorous methods used design and maintenance of safety critical systems.

# Motivation for Formal Methods

Introduce **rigor** to improve the software development process for

- Improved code structure
- Increased maintainability
- Reduced errors

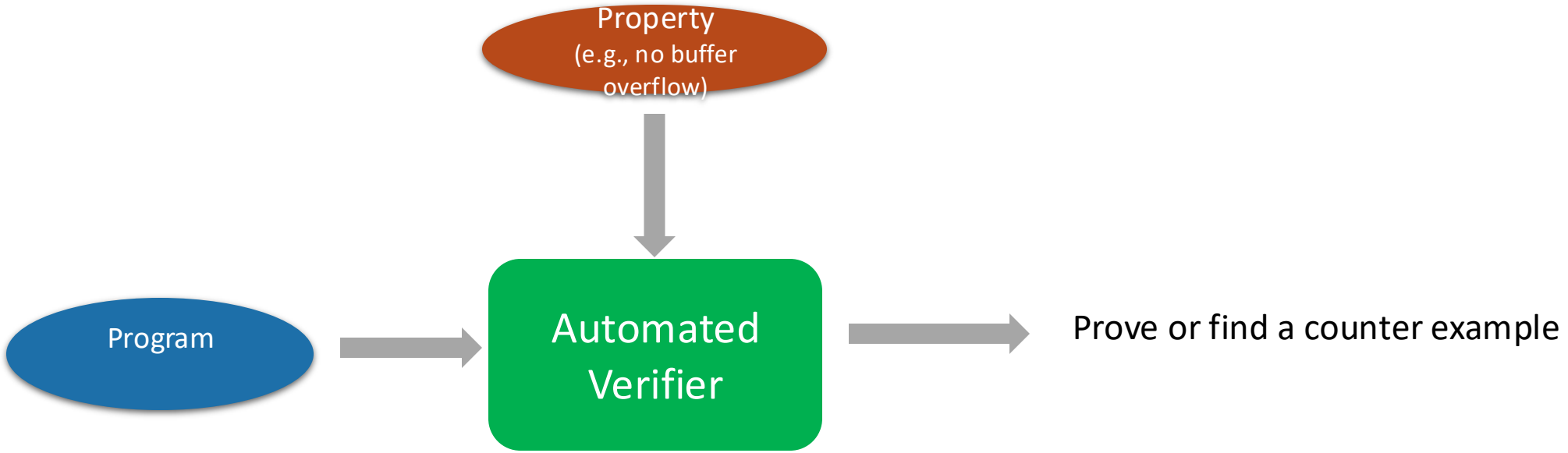
Usually formal methods are applied to only some part of a large software system (perhaps 10%).

# Why Formal Methods?

- To **catch bugs**
- To **eliminate whole classes of errors**
- Testing vs formal methods:

| Testing | Formal Methods |
|---|---|
| Can find system errors | Can find system errors |
| Generally works on actual code or sometimes simulated environment | Generally work on abstract model of code and environment |
| Cannot show that errors do not exist | Can show certain types of errors cannot exist |

Formal methods are the mathematics of software engineering




The general problem is undecidable, due to Rice's theorem

**While not all programs can be analyzed, many programs that developers care about can!**

# Limitations of Different Formal Method Techniques

- *Static Methods*: may result in falsely identifying errors.
- *Test generation*: generally incomplete.
- *Model checking*: limitations on how large a state space can be handled.
- *Theorem proving*: generally requires human expert intervention (latest approaches use LLMs).

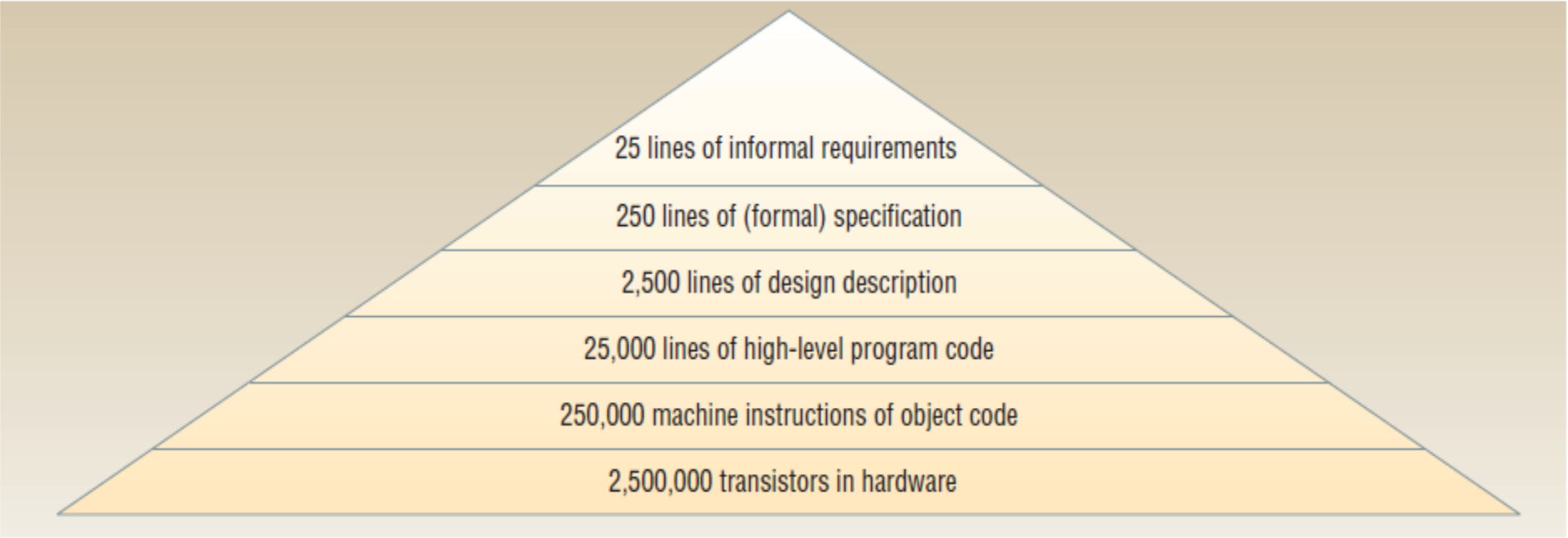
# Formal Methods “Levels”

| Level | Name | Involves |
|---|---|---|
| 0 | Formal specification | Using formal notation to specify requirements only; no analysis or proof |
| 1 | Formal development/verification | Proving properties and applying refinement calculus |
| 2 | Machine-checked proofs | Using a theorem prover or checker to prove consistency and integrity |

*Source: Bownen and Hinchey , “Ten Commandments of Formal Methods .. Ten Years Later, “ IEEE Computer 2006*

# Getting the Design Right

The size explodes as we proceed down the development implementation hierarchy (hypothetical numbers):

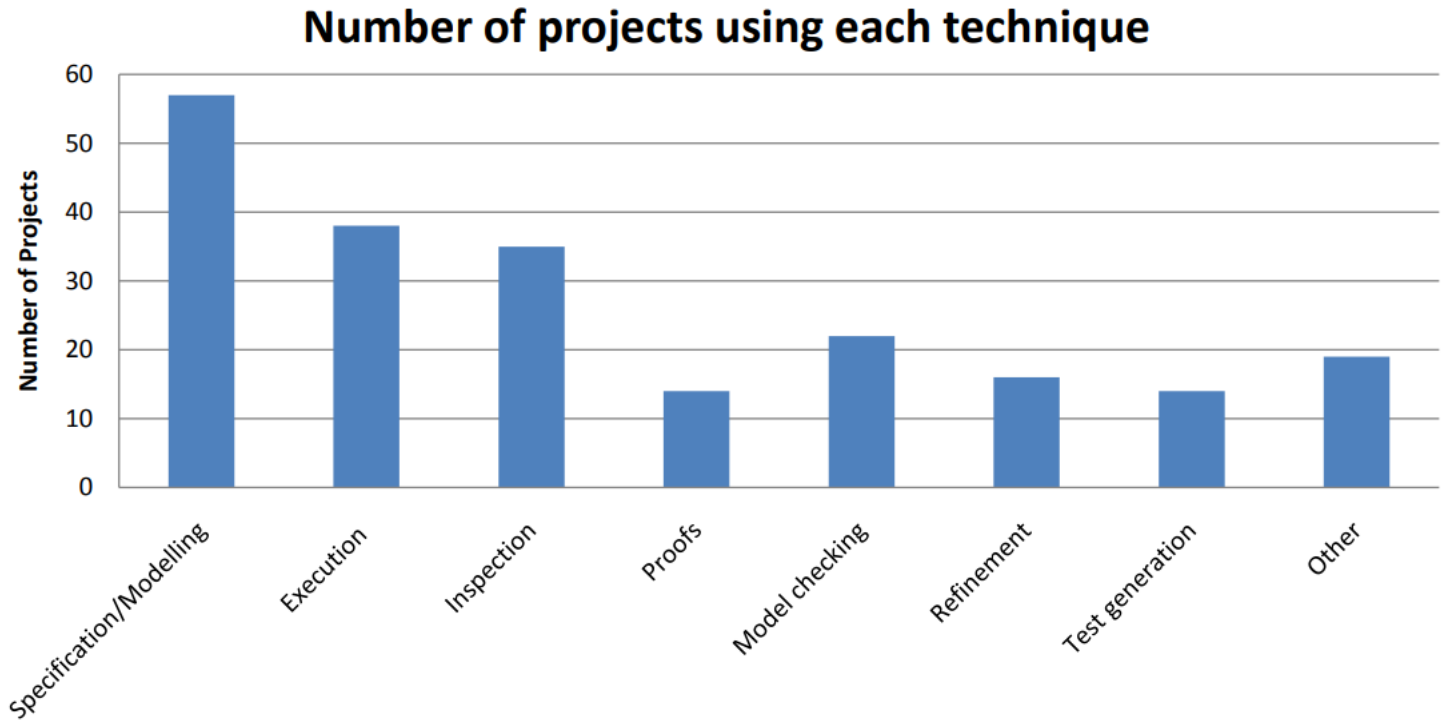



*Source: Bownen and Hinchey , "Ten Commandments of Formal Methods .. Ten Years Later, " IEEE Computer 2006*

# Notes on Kinds of Formal Methods Used

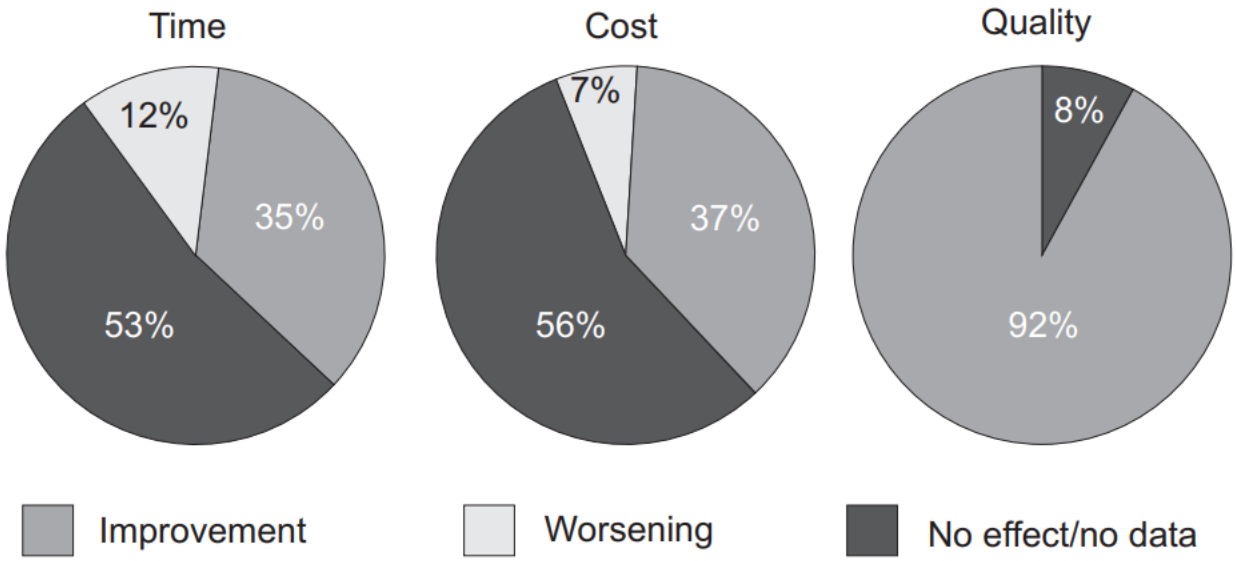
- Many projects use more than one technique.
- Different techniques applied to different parts of the code.
- Formal methods often applied only to some parts of a large code (e.g. judged safety-critical).

# Types of Formal Methods Used



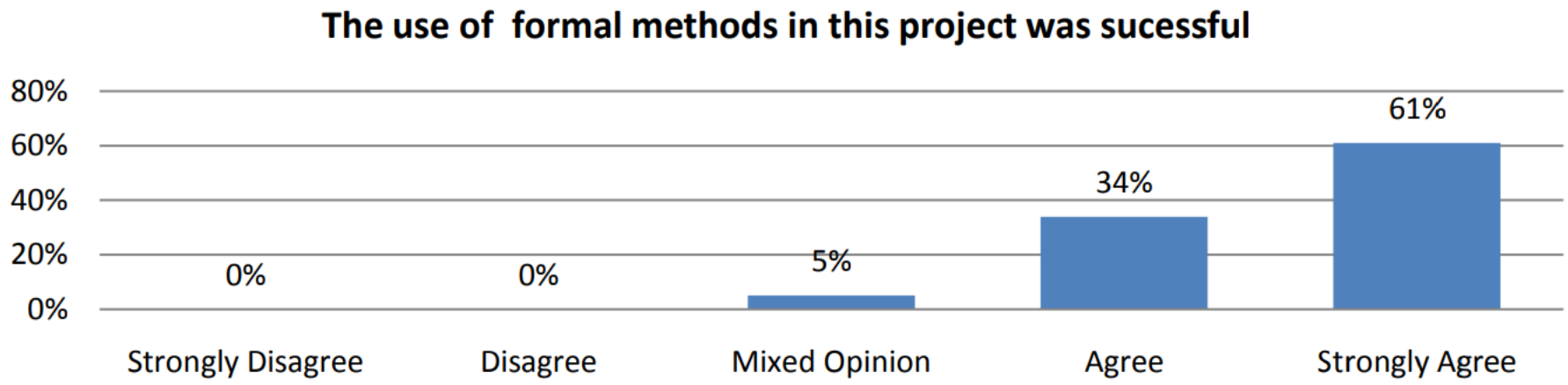

*Source: Woodcock, et al. "Formal Methods: Practice and Experience", ACM Computing Surveys, 2009*

# Perceived Impact




*Source: Woodcock, et al. "Formal Methods: Practice and Experience", ACM Computing Surveys, 2009*

# Use of Formal Methods Judged Successful?




*Source: Woodcock, et al. "Formal Methods: Practice and Experience", ACM Computing Surveys, 2009*

*Caveat: These projects reported using Formal Methods*

You may be less likely to report use if you thought it was a failure.

# Some Case Studies

- *Woodcock et al. 2009* highlights a number of industrial projects using formal methods. Software projects include:
  - Railway signaling and train control
  - Smart card based electronic cash system
  - Microcode for the AAMP5 microprocessor
  - Airbus controllers: Flight Control Secondary Computer and the Electric Load Management Unit.
  - Maeslant Kering Storm Surge Barrier

# Railway signaling and train control

- RATP 1998 (Parisian regional rail system): computerized signaling system
  - Increase network traffic by 25%
  - Preserve safety levels
- SACEM system with embedded hardware and software controls the speed of trains on RER Line A

http://www.flickr.com/photos/bindonlane/5802050248/

# RER Line A

- 21,000 lines of Modula-2 code:
  - 63% regarded as safety critical
  - Subjected to formal specification and verification
- Specification written in B and proofs done interactively using automatically generated verification conditions.
- Validation took 100 person years.
- Method used again on other lines..
- Unit tests replaced by global tests, all successful.

# Paris Metro Railway Signaling

| | **Paris Métro Line 14** | **Roissy Shuttle** |
|---|---|---|
| Line length (km) | 8.5 | 3.3 |
| Number of stops | 8 | 5 |
| Inter-train time (s) | 115 | 105 |
| Speed (km/hr) | 40 | 26 |
| Number of trains | 17 | 14 |
| Passengers/day | 350,000 | 40,000 |
| Number of lines of Ada | 86,000 | 158,000 |
| Number of lines of B | 115,000 | 183,000 |
| Number of proofs | 27,800 | 43,610 |
| Interactive proof percentage | 8.1 | 3.3 |
| Interactive proof effort (person-months) | 7.1 | 4.6 |

# Astrée

Static code analyzer that proves the absence of runtime errors in software written in C or C++

| |
|---|
| Division by Zero |
| Out-of-bounds array indexing |
| Null pointer errors |
| Integer and floating-point arithmetic overflow |
| Read access to uninitialized variables |
| Violations of user-defined assertions |

# Astrée

## Who uses Astrée?

Since 2003, Airbus France has been using Astrée in the development of safety-critical software for various aircraft series, including the A380.

In 2018, Bosch Automotive Steering replaced their legacy tools with Astrée and RuleChecker, resulting in significant savings thanks to faster analyses, higher accuracy, and optimized licensing and support costs.

Framatome employs Astrée for verification of their safety-critical TELEPERM XS platform that is used for engineering, testing, commissioning, operating and troubleshooting nuclear reactors.

The global automotive supplier Helbako in Germany is using Astrée to guarantee that no runtime errors can occur in their electronic control software and to demonstrate MISRA compliance of the code.

In 2008, Astrée proved the absence of any runtime errors in a C version of the automatic docking software of the Jules Verne Automated Transfer Vehicle, enabling ESA to transport payloads to the International Space Station.

A world leader in motors and ventilators for air-conditioning and refrigeration systems, ebm-papst is using Astrée for fully automatic continuous verification of safety-critical interrupt-driven control software for commutating high-efficiency EC motors for ventilator systems.

MTU Friedrichshafen uses Astrée to demonstrate the correctness of control software for emergency power generators in power plants. Together with its qualification package, Astrée is part of the IEC 60880 certification process.

https://www.absint.com/astree/index.htm

A problem has been detect and Windows has been shut down to prevent damage
to your computer.

THREAD_STUCK_IN_DEVICE_DRIVER

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restar
your computer, press F8 to select Advanced Startup options, and then
select Safe Mode.

Technical information:

*** STOP: 0x0000000A (0x00000000, 0x00000000)

# And for everyone else…

- "Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability." **Bill Gates, April 18, 2002. WinHec Keynote**
- https://www.microsoft.com/en-us/research/project/slam/

# More Reports from the Trenches

- **A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World**
  Dawson Engler (Stanford/Coverity) et al. CACM 2010:
  https://cacm.acm.org/magazines/2010/2/69354-a-few-billion-lines-of-code-later/fulltext

- **Continuous Reasoning: Scaling the Impact of Formal Methods**
  Peter O'Hearn (UCLondon/Facebook), 2018:
  https://research.fb.com/publications/continuous-reasoning-scaling-the-impact-of-formal-methods/

- **Formally verified cloud-scale authorization for AWS**
  Chakarov et al., ICSE 2025:
  https://www.amazon.science/publications/formally-verified-cloud-scale-authorization

- **Why Aren't There More Programming Languages Startups?**
  Jean Yang (ex CMU/Akita), blog 2021:
  https://www.akitasoftware.com/blog-posts/why-arent-there-more-programming-languages-startups