CMSC 433
Programming Language Technologies and Paradigms

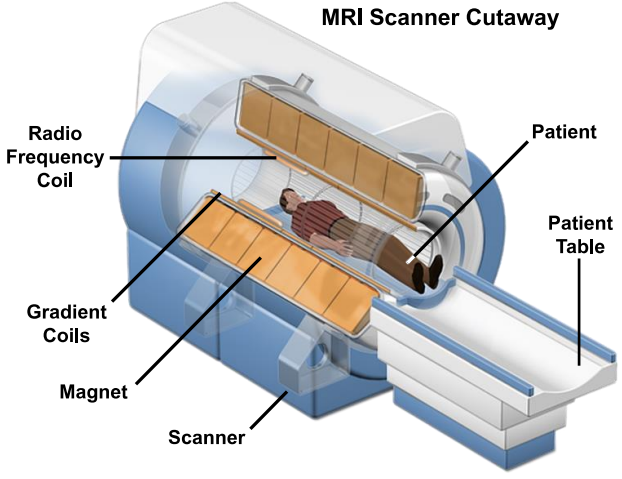
Introduction

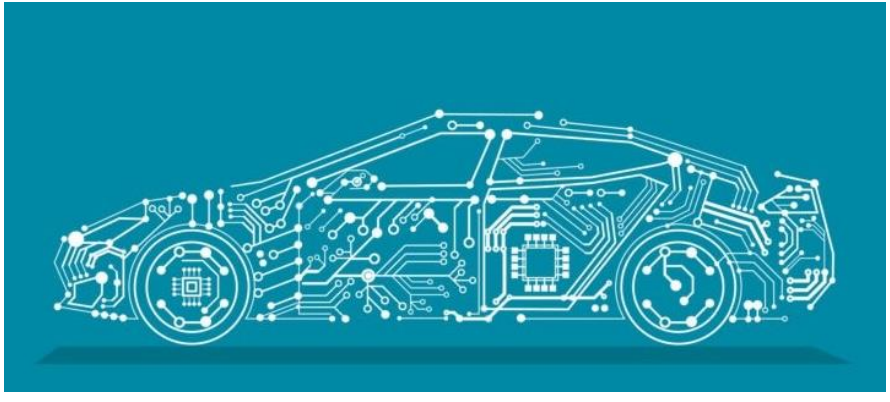
CMSC433 History

CMSC433 used to be a study of Concurrent programming.

Now it is on program proofs and verification.

Software is everywhere





Software has bugs

A software bug is a defect in a computer program or system that causes an undesired result.

Build Better Software

- We test software to check whether software satisfies expectations. Testing can show errors but not their absence.
- Software errors in critical systems can cause major disasters.
- This course is an introduction to techniques to get certainty that your program does what it is supposed to do.

Course logistics

- 6 assignments
- Midterm 10/23 (Wednesday)
- Final exam: Monday, December 15
- Several surveys and quizzes (on ELMS)

Grading

Assignments	50%
Quizzes & surveys	5%
Attendance	5%
Midterm	15%
Final	25%

Key resources

- Class web page (syllabus, assignments, course notes)
  - https://www.cs.umd.edu/class/fall2025/cmsc433/
- ELMS (announcements, grades)
- Piazza (communication, discussion)
- Gradescope (assignments, exams)
- Office Hours

Course Structure

- Topics:
  - Dafny
  - SAT solving and its applications
  - Solver aided programming
  - Computer aided (interactive) theorem proving
  - Testing
  - Supplementary reading

Dafny

- Dafny is both a programming language and a formal verification tool designed to help developers write programs that are mathematically proven to be correct.

- In Dafny, you can write preconditions, postconditions, and invariants. Dafny then uses an automated theorem prover (based on SMT solvers like Z3) to check whether your program meets its specification.

- This means Dafny doesn't just test code on some inputs—it proves properties hold for all possible executions.

SAT/SMT Solvers

- SAT/SMT solvers are computer programs which aim to solve the Boolean satisfiability problem.
- They are used in program verification (like Dafny, SPARK, Why3), theorem proving, and software/hardware correctness.
- Tools like Dafny rely on SMT solvers (e.g., Z3) to mathematically prove that your code always meets its specification.
- Modern SAT/SMT solvers (like Z3, CVC5, MiniSat) can solve *huge, real-world problems surprisingly fast.*

Computer Aided (interactive) Theorem Proving

- A software tool to assist with the development of formal proofs by human–machine collaboration.
  - Human guide the proof construction.
  - Machine checks the low-level details
- Examples:
  - Rocq (Coq), Isabelle HOL, F*, Lean4

Building Reliable Software

Cost of Software Errors

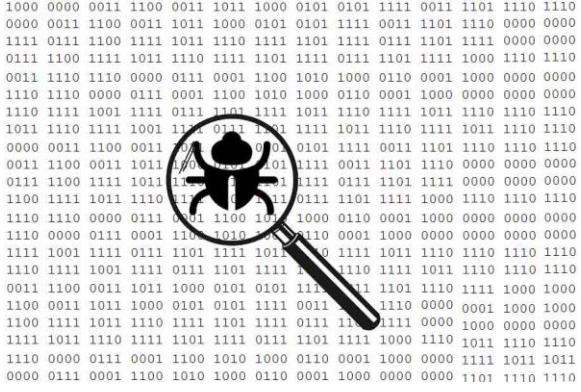
$2.8 Trillion in 2020 alone

Source: Forbes

https://www.forbes.com/councils/forbestechcouncil/2023/12/26/costly-code-the-price-of-software-errors/

Cost of Software Errors

Eestimated 50% of programmers time spent on finding and fixing bugs.

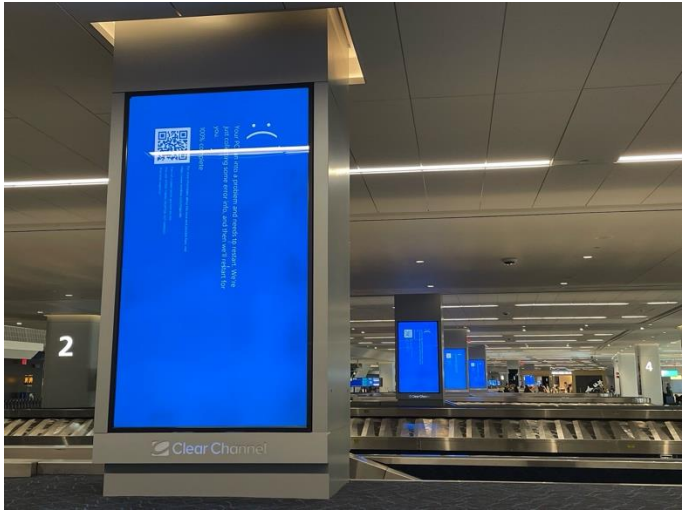


Software failure examples: 2024 CrowdStrike incident

On July 19th 2024, CrowdStrike distributed a faulty update to its Falcon Sensor security software that caused widespread problems with Microsoft Windows.

Roughly 8.5 million systems crashed and were unable to properly restart.

The worldwide financial damage has been estimated to be at least US $10 billion.



Nobody travels on 07/19

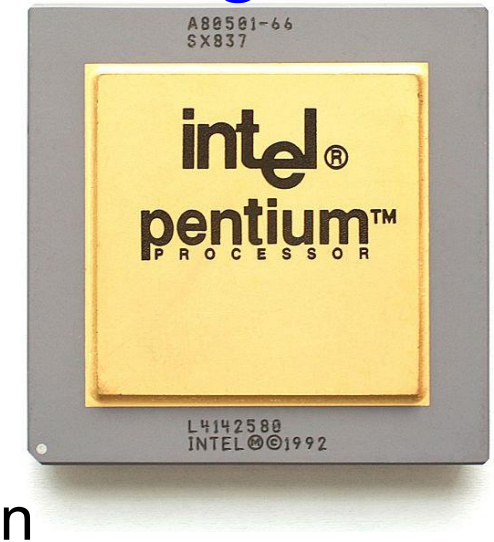
Software failure examples: **Ariane flight V88**

Ariane flight V88 (Ariane 5 rocket) exploded right after launch in 1996.

Conversion of 64-bit float to 16-bit integer caused an exception (made it crash)

European space agency spent 10 years and $7 billion to produce Ariane 5

Software failure examples: Pentium Floating Point (FDIV) Bug



A hardware bug affecting the floating-point unit (FPU) of the early Intel Pentium processors in 1994.

- Incorrect result through floating point division
- Rarely encountered in practice
- 1 in 9 billion floating point divides with random parameters would produce inaccurate results (Byte magazine)
- 475 million dollars, reputation of Intel.

Zune Leap Year Freeze

At midnight of December 31, 2008, all the millions of Zune 30 that Microsoft sold froze.

```
BOOL ConvertDays(UINT32 days, SYSTEMTIME* lpTime){
  ...
  year = ORIGINYEAR; /* = 1980 */
  while (days > 365){
    if (IsLeapYear(year)){
      if (days > 366){
        days -= 366;
        year += 1;
      }
    }else{
      days -= 365;
      year += 1;
    }
    }
  ...
}
```

Not just economic loss: **Toyota Unintended Acceleration**

- Bugs in electronic throttle control system (2009).
- Car kept accelerating on its own.
- May have caused up to 89 deaths in accidents.
- Recalls of 10 million vehicles.

Not just economic loss, **Therac-25**

- a computer-controlled radiation therapy machine (1985-1987)
- some patients were given massive overdoses of radiation.
- Killed four and left two others with lifelong injuries.

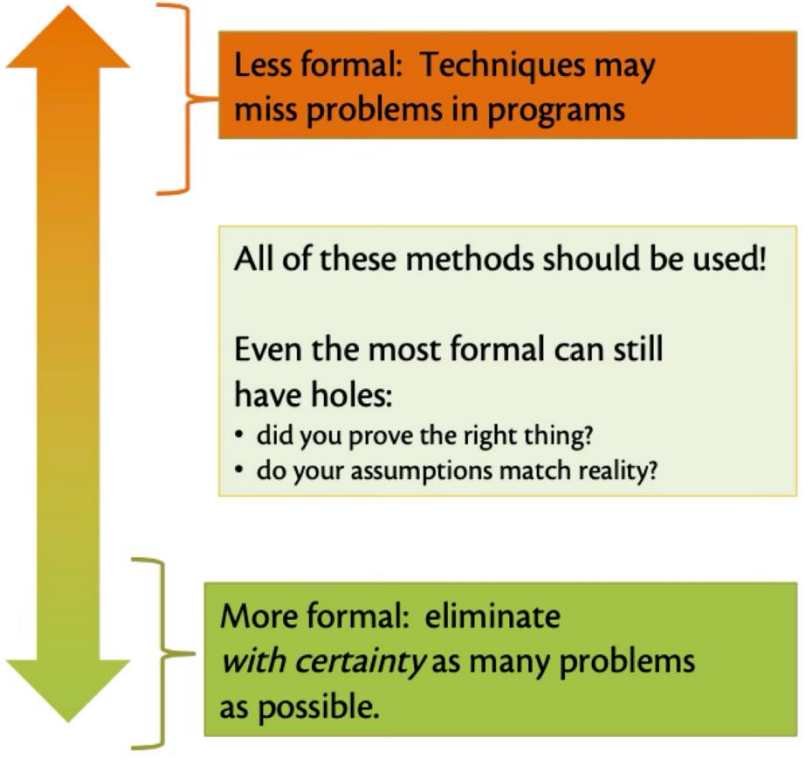
Problem Source

- Requirements: Incomplete, inconsistent, …
- Design: Flaws in design
- Implementation: Programming errors,…
- Tools: Defects in support systems and tools used

How can you get some assurance that a program does what you want it to do?

Approaches to Validation

- Social
  - Code reviews
  - Extreme/pair programming
- Methodological
  - Design patterns
  - Test-driven development
  - Version control
  - Bug Tracking
- Technological
  - Static analysis
  - Fuzzers
- Mathematical
  - Sound Type Systems
  - Formal verification



Testing

- Evaluating software by observing its execution

- Execute program with the intent of finding failures (try out inputs, see if outputs are correct)

How do we test?

- Test: try out inputs, see if outputs are correct

- Testing means to execute a program with the intent of detecting failure

- terminology, testing levels, unit testing, black box vs white box, principles of test-set construction/coverage, automated and repeatable testing (JUnit)

Formal verification

- Determine whether a piece of software fulfils a set of formal requirements in every execution
  - Formally prove method correct (find evidence of absence of failure)

Formal verification

- Scaled to 10s of lines of code in 1970s
- Now, research projects scale to real software:
  - CompCert: A verified C compiler
  - seL4: verified microkernel OS
  - Ynot: verified DBMS, web services
- In another 50 years?

Some failures are obvious

- obviously wrong output/behavior
  - non-termination
  - crash
  - freeze
- . . . but most are not!
  In general, what constitutes a failure, is defined by: a specification!

Specification

- Specification: An unambiguous description of what a program should do.
- Bug: Failure to meet specification.

- Unclear Specification leads to failure

Specification: Example

`Sort(src: Integer Array) -> Integer Array`

- Specification:
  - **Requires**: src is an array of integer
  - **Ensures**: returns a sorted array

Is this a good specification?

Specification: Example

```
Sort(src: Integer Array) -> Integer Array
```

- Specification:
  - **Requires**: src is an array of integer
  - **Ensures**: returns a sorted array

```
Sort([3,1,4,5]) == [ ]
Sort([3,1,4,5]) == [1,2,3]
```

Specification: Example

```
Sort(src: Integer Array) -> Integer Array
```

- Specification:
  - **Requires**: src is an array of integer
  - **Ensures**: returns a sorted array with only elements from the input

```
Sort([3,1,4,5]) == [1,1,4 ]
Sort([3,1,4,5]) == [1,3,3,5]
```

Specification: Example

`Sort(src: Integer Array) -> Integer Array`

- Specification:
  - **Requires**: src is an array of integer
  - **Ensures**: returns a permutation of src that is sorted

```
Sort([ ]) == ?
Sort(null) == ?
Permutation?
```

Specification: Example

`Sort(src: Integer Array) -> Integer Array`

- Specification:
  - **Requires**: src is a non-null array of integer
  - Ensures: returns a permutation of src that is sorted

Specification of a method

```
method m( )
   Requires: Precondition
   Ensures: Postcondition
```

Means:

- If a caller of `m()` fulfills the required Precondition, then the callee `m()` ensures that the Postcondition holds after `m()` finishes.
- Garbage in, garbage out

Failure vs Correctness

- What constitutes a failure

  A method fails when it is called in a state fulfilling the required precondition of its contract and it does not terminate in a state fulfilling the postcondition to be ensured.

Failure vs Correctness

- A method is correct means:
  - whenever it is started in a state fulfilling the required precondition, then it terminates in a state fulfilling the postcondition to be ensured.

- Correctness amounts to proving absence of failures! A correct method cannot fail!

Verification

- Testing cannot guarantee correctness, i.e., absence of failures
- Verification: Mathematically prove method correct
  - Goal: find evidence for absence of failures

- This course: Formal verification (logics, tool support)

Summary

- CS433 introduces techniques for ensuring program correctness—that is, proving that a program does what it is intended to do. The course provides a mathematical foundation for the rigorous analysis of real-world software systems.

- The skills developed in this course are in high demand, as nearly all human interactions are now mediated by software—and it is especially critical to guarantee the correctness of AI-generated programs.