# CISC/CMPE 422/835
# Formal Methods in Software Engineering

Juergen Dingel

Fall 2022

Course intro & overview

# Admin

- Marking scheme for CISC/CMPE422
  - Final exam: 40% of mark
  - 1 midterm: 20% of mark
  - Assignments (4, individual, weighted equally): 40% of mark
- Marking scheme for CISC835
  - Final: 35%, midterm: 15%, assignments: 30%, project: 20%
- Exams (closed book, 1 8.5”x11” datasheet):
  - Midterm: Week 8 (Thurs, Nov 3), in class
  - Final (in person): tba
- OnQ:
  - Syllabus, courseware, assignments
  - Discussion forum
- TAs:
  - Dimitrios Economou, Ella Morgan, Georgia Reed, December Stuart , Michaelah Wales

# Admin (Cont'd)

- JD will be away
  - Week 7 (Oct 25-27)

# About Me

**Small town Germany:**

Born, raised, etc

**Berlin**: Undergrad

**Pittsburgh**: PhD

**Kingston**: since 2000

A

B

D

E

# A Definition

## Formal methods

- Notations, techniques and tools to
  - capture relevant aspects of software unambiguously and precisely and
  - allow analysis
- Other suitable course titles
  - “Formal Modeling and Analysis”
  - “Formal Specifications for Requirements, Design, Implementation, and Testing”

# Overview

What are formal methods?

Why use formal methods?

‘Formal methods’ in other disciplines

Overview of course content

# Why Use Formal Methods?

**Statement 1:**

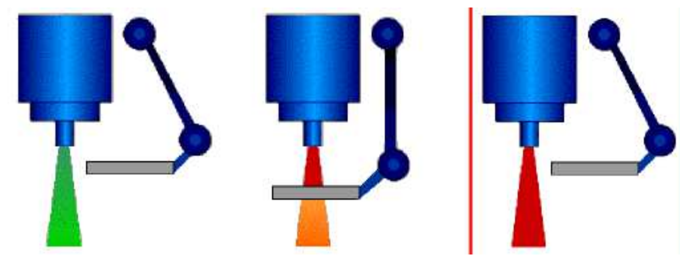
*"Sometimes, it is very important that certain software failures don't occur and that there is acceptable supporting evidence for this"*

# Statement 1: Examples

- ‘Safety-’ or ‘mission-critical’ software
  - Military, nuclear, medical, automotive, avionics, aerospace
- Infrastructure
  - Energy, telecom, avionics
- Economy
  - Financial

# Example 1: Therac-25 (1985-87)

- Radiotherapy machine with SW controller
- SW failed to maintain essential invariants:
  - To generate X-rays:
    - either use low-power electron beam, or
    - use high-power beam w/ intervening 'beam spreader plate'



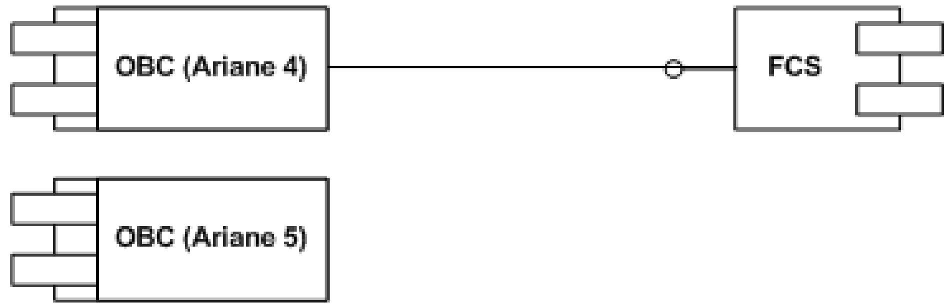
- Several deaths ($\geq$ 6) due to burning

# Example 2: ESA Ariane 5 (June 1996)

- On June 4, 1996, unmanned Ariane 5 launched by ESA explodes 40 seconds after lift-off
- One decade of development costing $7billion lost

- What went wrong?
  - Bad reuse of code from Ariane 4
  - Bad fault-tolerance mechanism
  - Bad coding practices

# Example 2: ESA Ariane 5 (June 1996) (Cont'd)




- Example of how not to do reuse:
  - Parts of FCS from Ariane 4
  - $v_x$ much greater for Ariane 5
  - Conversion operation in FCS fails
  - OBC interprets error code as flight data
  - …
  - Launcher self-destructs
- Example of how not to achieve fault-tolerance:
  - FCS and backup FCS identical, thus backup also failed
- Example of how not to code:
  - When code caused exception, it wasn't even needed anymore
- References:
  - [Gle96] and **www.ima.umn.edu/~arnold/disasters/ariane.html**

# Example 3: The Blackout Bug

- Aug 13, 2003: >50 Million people w/o electricity for hours, days
- Cause: Race condition in alarm system (10^6 Loc of C)
- Worst black out in North American history
- Cost: US$ 6 billion

## Tracking the blackout bug

*Kevin Poulsen*, SecurityFocus 2004-04-07

<snip>

languages. Eventually they were able to reproduce the Ohio alarm crash in GE Energy's Florida laboratory, says Unum. "It took us a considerable amount of time to go in and reconstruct the events." In the end, they had to slow down the system, injecting deliberate delays in the code while feeding alarm inputs to the program. About eight weeks after the blackout, the bug was unmasked as a particularly subtle incarnation of a common programming error called a "race condition," triggered on August 14th by a perfect storm of events and alarm conditions on the equipment being monitored. The bug had a window of opportunity measured in milliseconds. "There was a couple of processes that were in contention for a common data structure, and through a software coding error in one of the application processes, they were both able to get write access to a data structure at the same time," says Unum. "And that corruption led to the alarm event application getting into an infinite loop and spinning." **Testing**

<snip>

# Example 4: 2010 Toyota Prius

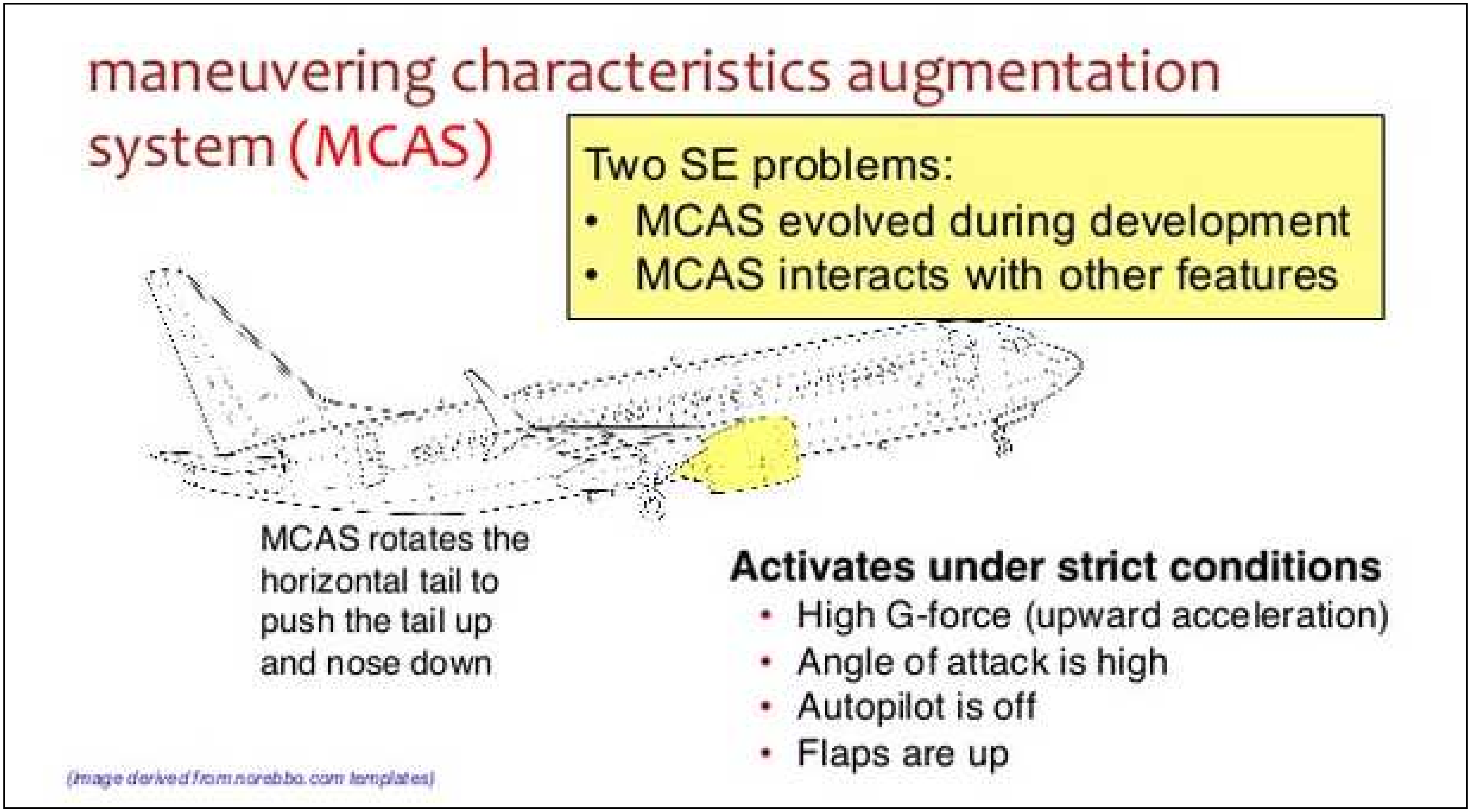
- Three systems
  - Hybrid brake system
    - Normal
    - Regenerative
  - Anti-lock brake system (ABS)
- Unintended interaction
  - Braking force reduced after ABS actuation

  $\Rightarrow$ Increased stopping distance

  $\Rightarrow$ 62 crashes, 12 injuries

US NHTSA,
https://www.nhtsa.gov/vehicle/2010/TOYOTA/PRIUS/4%252520DR#investigations

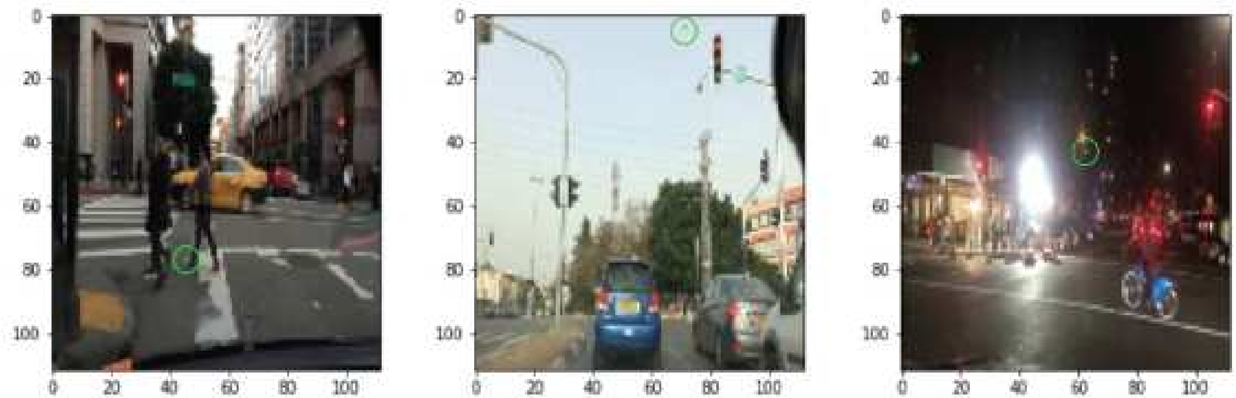
# Example 5: Boeing 737 Max




[Slide from Jo Atlee, Living with Feature Interactions, FSE’19 ]

# Example 6: Deep Neural Nets for Autonomous Driving

## Should we worry about safety?



(a) (b) (c)

Red light classified as green with (a) 68%, (b) 95%, (c) 78% confidence after one pixel change.

– TACAS 2018, https://arxiv.org/abs/1710.07859

Can we verify that such behaviour cannot occur?

[Marta Kwiatkowska, Safety and Robustness for Deep Learning with Provable Guarantees, FSE'19]

# Why Use Formal Methods? (Cont'd)

**Statement 1:**

*"Sometimes, it is very important that certain software failures don't occur and that there is acceptable supporting evidence for this"*

# Certification of Software in Medical Devices

*The FDA's analysis of 3140 medical device recalls conducted between 1992 and 1998 reveals that 242 of them (7.7%) are attributable to software failures*

*[…] any medical device software product developed after June 1, 1997 […]
is subject to applicable design control provisions. (See of 21 CFR §820.30.)
[…]
Design controls, such as planning, input, verification, and reviews, are required for medical device software. (See 21 CFR §820.30.)*

*The corresponding documented results from these activities can provide additional support for a conclusion that medical device software is validated.*

[FDA] U.S. Department of Health and Human Services, Food and Drug Administration, Center for Devices and Radiological Health, Center for Biologics Evaluation and Research. General Principles of Software Validation; Final Guidance for Industry and FDA Staff. Jan 2002, https://www.fda.gov/regulatory-information/search-fda-guidance-documents/general-principles-software-validation

# Certification of Avionics Software

## DO-178C

*"is an acceptable means, but not the only means, for showing compliance with the applicable airworthiness regulations for the software aspects of airborne systems and equipment certification"*

### Software levels

- From E (failure has no effect) to A (failure has catastrophic effect)

### Certification objectives

- the higher the level, the more objectives

### Examples of activities necessary to satisfy objectives

- Review of requirements, design, and code; testing; configuration management

[Radio Technical Commission for Aeronautics (RTCA). DO-178C: Software Considerations in Airborne Systems and Equipment Certification. Jan 2012] https://en.wikipedia.org/wiki/DO-178C

# ISO Standard for Automotive Software

## ISO 26262

- Covers functional safety aspects of the entire development process
- As for DO-178-C, classifies software
  - into levels (called Automotive Safety Integrity Levels, ASILs)
- to determine
  - risk and
  - requirements for validation and confirmation measures

[International Standards Organization (ISO). “Road vehicles – Functional safety (ISO 26262)”. 2011]
https://en.wikipedia.org/wiki/ISO_26262

# Why Use Formal Methods? (Cont'd)

**Statement 1:**

*"Sometimes, it is very important that certain software failures don't occur and that there is acceptable supporting evidence for this"*

**Statement 2:**

*"Sometimes, relevant aspects of the software (e.g., requirements, development context, operating conditions) are so complex that Statement 1 is impossible to achieve with 'standard' methods"*

# The Limits of Testing

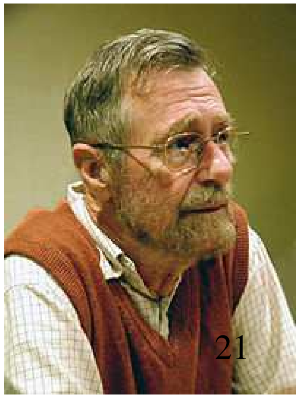
*"We test exhaustively, we test with third parties, and we had in excess of three million online operational hours in which nothing had ever exercised that bug. […] I'm not sure that more testing would have revealed that."*

Manager at GE,
maker of Energy Management System responsible for Blackout Bug in 2003
in 'Tracking the blackout bug'

*¨Typically, testing alone cannot fully verify that software is complete and correct. In addition to testing, other verification techniques and a structured and documented development process should be combined to ensure a comprehensive validation approach"*

In [FDA Guidelines]

*"Testing shows the presence, not the absence of bugs."*
Edsger W. Dijkstra



E.W.Dijkstra.
Turing Award 1972

# Software Complexity: In Lines of Code

- **Windows OSs**
  - NT 3.1 (1993): 0.5 million LoC
  - 95:
    - 11
      tes
  - 2000: 29 million LoC
  - XP (2001): 35 million LoC
  - Vista (2007): 50 million LoC
  - Windows 7: 40 million LoC
- **Windows**
  - Office (2001): 25 million LoC
  - Office (2013): 44 million LoC
  - Visual Studio (2012): 50 million LoC
- **Mac OS X "Tiger":** 85 million LoC

**Software is one of the most complex man-made artifacts!**

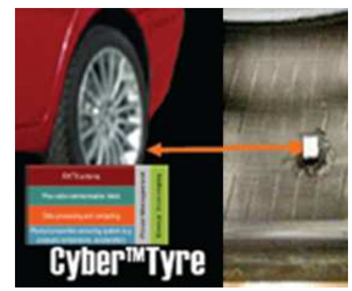
- **Average iPhone app:** 40,000 LoC
- **Pacemaker:** 100,000 LoC
- million LoC
- n LoC
- **Boeing 787:** 14 million LoC
- **F-35 fighter jet:** 24 million LoC
- **Large Hadron Collider:** 50 million LoC
- **Facebook:** 60 million LoC
- **Car**
  - 1981: 50,000 LoC
  - 2005: 10 million LoC
  - 2014: 100 million LoC

1M LoC = 18,000 pages of printed text
= stack 6 feet high

[Charette. "Why Software Fails". IEEE Spectrum, Sept 2005]
[McCandless, www.informationisbeautiful.net/visualizations/million-lines-of-code]

# It is Not Going to Get Easier

- **More complexity**
  - Less mechanical, more electronic & computerized
  - More
    - features & capabilities
    - integration
    - virtualization
    - distribution & concurrency






[from A. Sangiovanni-Vincentelli]

# Why Use Formal Methods (Cont'd)

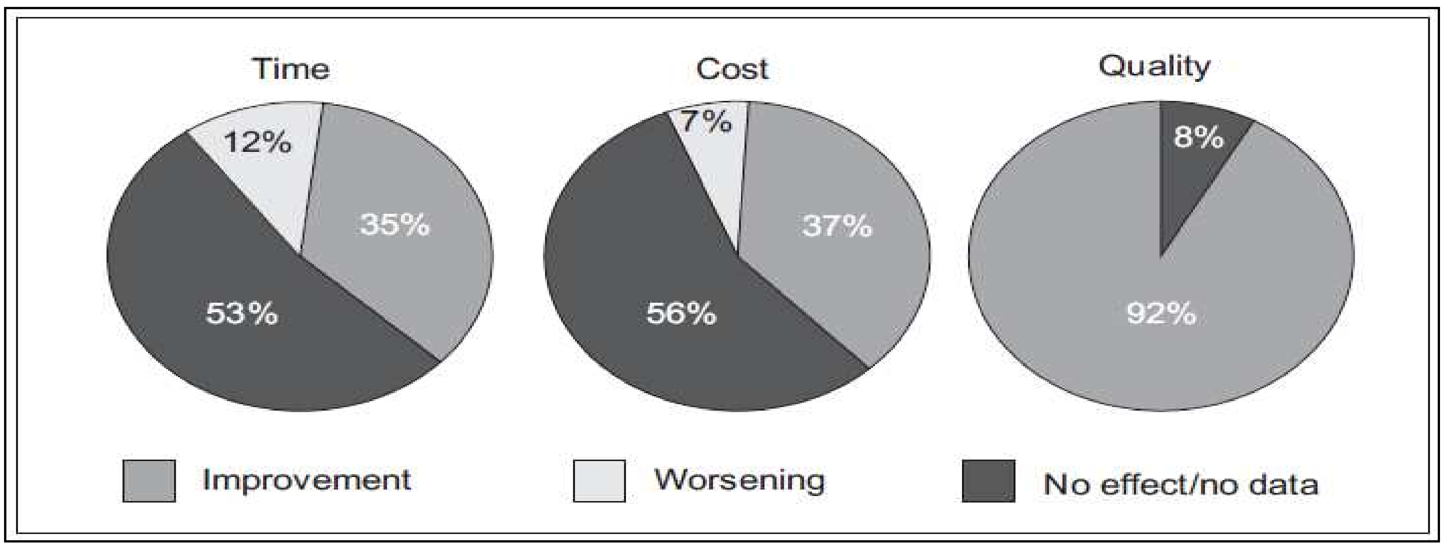
**Statement 1:**
*"Sometimes, it is very important that certain software failures don't occur and that there is acceptable supporting evidence for this"*

**Statement 2:**
*"Sometimes, relevant aspects of the software (e.g., requirements, development context, operating conditions) are so complex that Claim 1 is impossible to achieve with 'standard' methods"*

**Statement 3:**
*"In these cases, formal methods can help by allowing the construction of unambiguous artifacts modeling relevant aspects of the system such that it can be analyzed w.r.t. desirable properties"*

# Examples of Uses of Formal Methods

- DO-178C for avionics software allows formal methods to complement testing
- Survey of 62 int'l FM projects
  - **Domains:** Real-time, distributed & parallel, transaction processing, high-data volume, control, services




[Radio Technical Commission for Aeronautics (RTCA). DO-333: Formal Methods Supplement to DO-178C and DO-278A.

[Woodcock et al. Formal Methods: Practice and Experience. ACM Computing Surveys 41(4). 2009]

# A Definition

## Formal methods

- Notations, techniques and tools to
  - capture aspects of software unambiguously and precisely and
  - allow analysis
  - make software engineering more rigorous

# Overview

What are formal methods?

Why use formal methods?

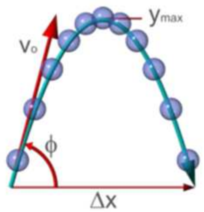
‘Formal methods’ in other disciplines

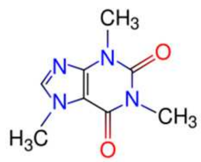
Overview of course content

# ‘Formal Methods’ in Other Disciplines

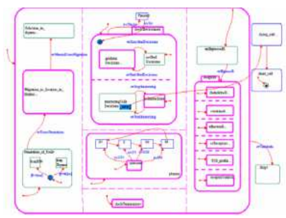
## Natural sciences

- Understanding, predicting existing phenomena (c.f., “Backwards Engineering”)





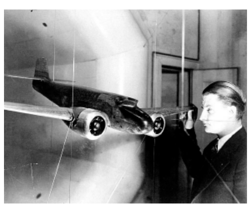
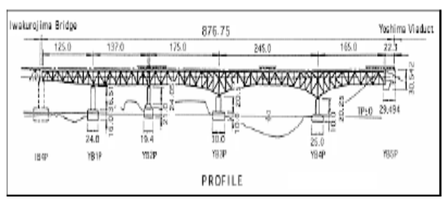






## Engineering

- Building artifacts with certain properties (c.f., “Forwards Engineering”)

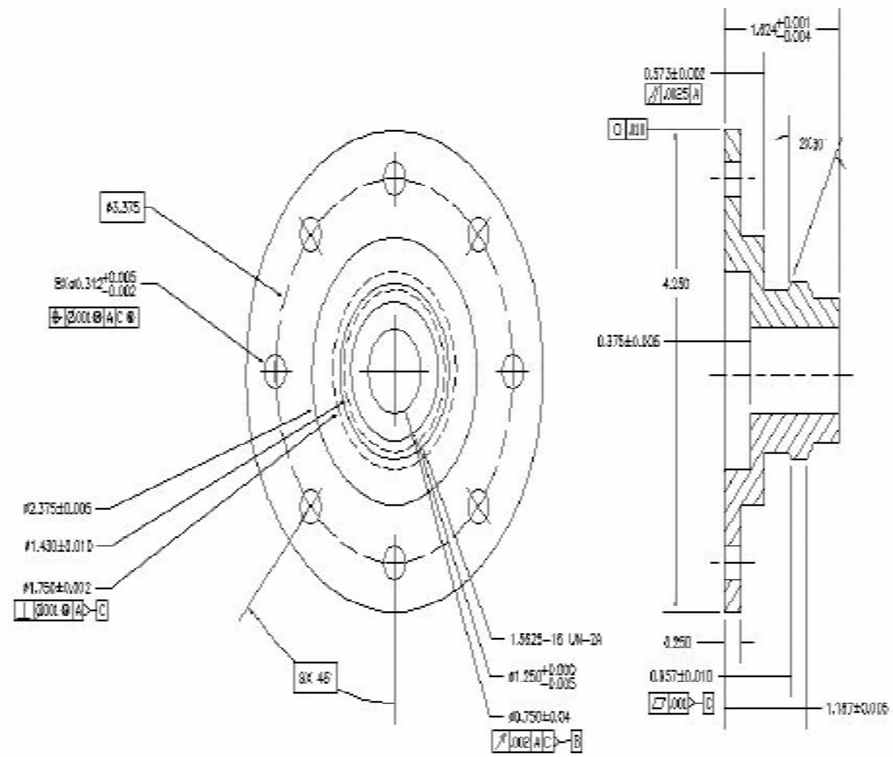

> Modeling is central, except in SW Eng

# Formal Modeling in Manufacturing

Mechanical design from 1800 to about 1980:

1. Draftsmen create 3-view drawings
2. Machinists create parts from drawings

$\Rightarrow$ laborious, error-prone, inefficient



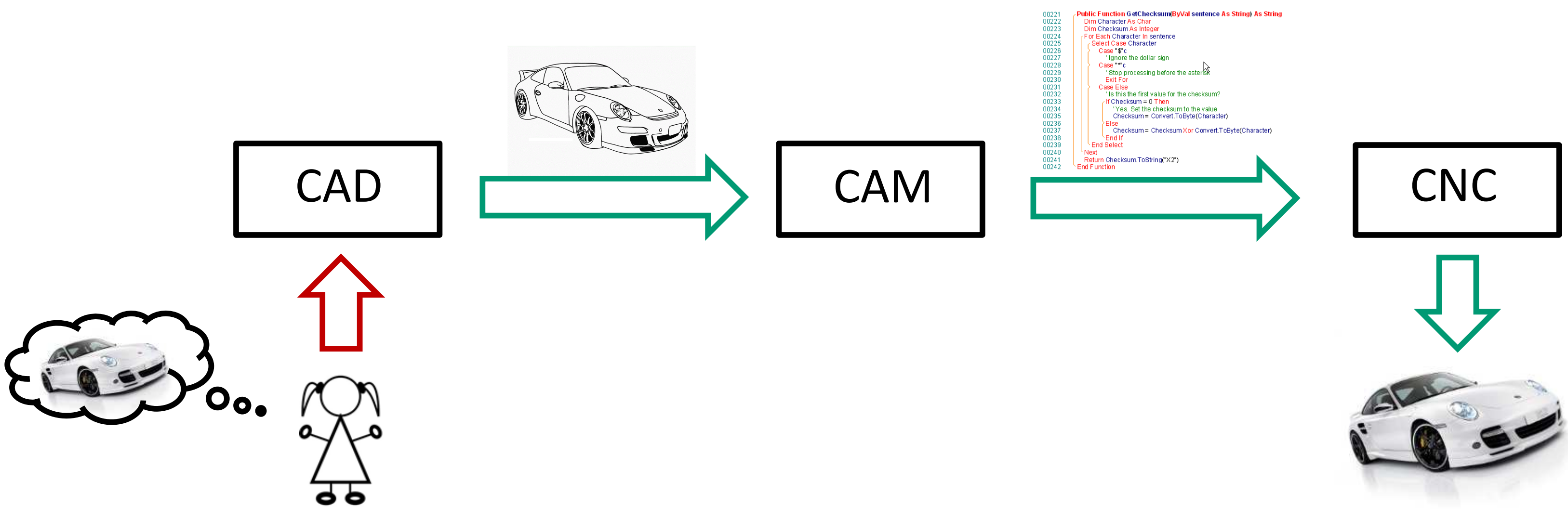
# Formal Modeling in Manufacturing (Cont'd)

- **Example: Concorde (1976 – 2003)**
  - > 100,000 drawings
  - in 2 languages, using both metric and imperial systems
  - $\Rightarrow$ worked, but 7x over budget

# Formal Modeling in Manufacturing (Cont'd)

Mechanical design from about 1972: CAD/CAM

1. Create drawings w/ computer (CAD)
2. From drawing, computer automatically generates program to drive milling and CNC machines (CAM)




⇒ much better analysis capabilities and productivity

⇒ **CAD/CAM has revolutionized manufacturing**

# What is Software Engineering?

**Engineering:**

*"The application of scientific and mathematical principles to practical ends such as the design, manufacture, and operation of efficient and economical structures, machines, processes, and systems"*

American Heritage Dictionary

**Software Engineering:**

*"The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, that is, the application of engineering to software"*

IEEE Standard 610.12

Hmm...

# What is Software Engineering (Cont'd)

## Engineering

1. build (mathematical) models
2. analyze models rigorously
3. refine models
4. build artifact
5. little testing

**Characteristics**

- Very rigorous
- "front-loaded"
- **Main QA technique:**
  Modeling & analysis

## Software Engineering

1. some (informal) modeling
2. build artifact
3. some (informal) reuse
4. lots of testing

**Characteristics**

- Mostly informal
- "back-loaded"
- **Main QA technique:**
  Testing (often >50% of total development effort)

⇒ Software engineering currently isn't really like engineering!

⇒ Formal Methods can help to make software engineering more rigorous

# Overview

What are formal methods?

Why use formal methods?

'Formal methods' in other disciplines

Overview of course content

# CISC422/835: Overview (Cont'd)

| Artifact | Notation | Analysis | Tool | |
|---|---|---|---|---|
| Requirements | | | | Weeks 2-3 |
| Class models | | | | Weeks 4-6 |
| Code | | | | Weeks 7-8 |
| Finite state machines | | | | Weeks 11-12 |

## Things you are going to learn

- Details about notations, analysis techniques, and tools
- Formalization (i.e., precise thinking, expression, …)

# Part 1: FM for Requirements

Bugs often creep in early in development…

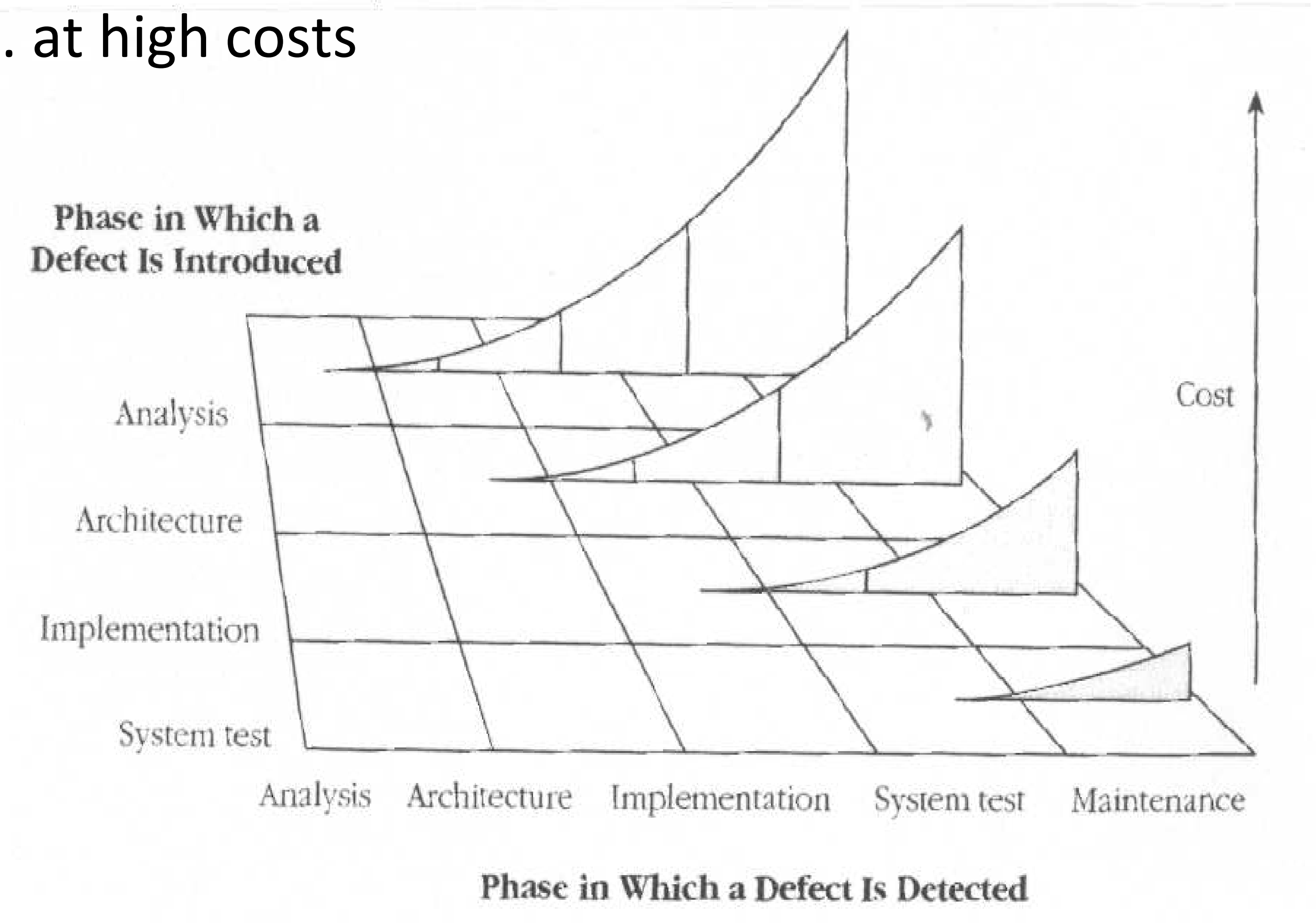
1. *"70% of errors in embedded safety-critical software are introduced in the requirements (35%) and architecture design phases (35%)"*
2. *"80% of all errors are not discovered until system integration or later*

[Feiler, Goodenough, Gurfinkel, Weinstock, Wrage. Four Pillars for Improving the Quality of Safety-Critical Software Reliant Systems. White Paper. SEI. 2013]

What's the relationship b/w time bug is discovered and costs?

# Part 1: FM for Requirements (Cont'd)

… at high costs




Informal relationship b/w time bug is discovered and costs
[S. McConnell. Code Complete. 1993]

FM techniques can help

# Part 2: FM for Class Models

- UML class diagrams
  - most widely used diagram type
  - very useful for describing classes/concepts and their relationships
- We'll look at a class-modeling language (Alloy), that is
  - Reminiscent of UML class diagrams
  - Has precise semantics
  - Comes with usable, powerful, automatic analysis tool

# Part 3: FM for Testing

- Testing typically is ‘example/instance-based’
- Use property-based testing to
  - capture properties that output is expected to satisfy
  - automate test input generation
  - facilitate effective, high-volume testing

# Part 4: FM for FSMs

- Even small pieces of code can be very intricate
- Example: Tie-breaker protocol for mutual exclusion

```
P1 =
while true do
  f1 := true;
  last := 1;
  await (!f2 or last!=1);
  criticalSection1;
  f1 := false
end
```

```
P2 =
while true do
  f2 := true;
  last := 2;
  await (!f1 or last!=2);
  criticalSection2;
  f2 := false
end
```

What if

`f1:=true; last:=1`

is replaced by

`last:=1; f1:=true`

in **`P1`** and similarly for **`P2`**?

Embedded code often is concurrent

# Part 4: FM for FSMs (Cont'd)

- Resulting version of Tie-breaker protocol is incorrect

```
P1 =
while true do
  last := 1;
  f1 := true;
  await (!f2 or last!=1);
  criticalSection1;
  f1 := false
od
```

```
P2 =
while true do
  last := 2;
  f2 := true;
  await (!f1 or last!=2);
  criticalSection2;
  f2 := false
od
```

| | | | | | | |
|---|---|---|---|---|---|---|
| *f1=false* | $\xrightarrow{\textbf{P2}}$ | *f1=false* | $\xrightarrow{\textbf{P1}}$ | *f1=true* | $\xrightarrow{\textbf{P2}}$ | *f1=true* |
| *f2=false* | | *f2=false* | | *f2=false* | | *f2=true* |
| *last=** | | *last=2* | | *last=1* | | *last=1* |
| | | | | */* P1 in CS */* | | */* P2 in CS */* |

# Part 4: FM for FSMs (Cont'd)

- State space exploration (a.k.a., model checking)
  - Perfect for these kinds of problems
  - Analysis technique for finite state machines and protocols based on exhaustive state space exploration and temporal logic

# Summary

- Software
  - is becoming more pervasive & complex
  - making it hard to certify
- Formal modeling and analysis can help
- CISC 422/835 offers a comparative study of different formal modeling notations and analysis techniques for different artifacts
  - Requirements
    - Propositional/predicate logic & theorem proving
  - Class models & constraint solving
  - Code & testing
  - Finite state machines & model checking