CSE 814
Formal Methods in Software Engineering
Software is complex

“One of the most complex man made artifacts”
-- M. Dwyer, J. Hatcliff, R. Howell (KSU CIS 771)

“What were the lessons I learned from so many years of intensive work on the practical problem of setting type by computer? One of the most important lessons, perhaps, is the fact that SOFTWARE IS HARD.... the writing of programs for TEX and for METAFONT proved to be much more difficult than all the other things I had done (like proving theorems or writing books). The creation of good software demands a significantly higher standard of accuracy than those other things do, and it requires a longer attention span than other intellectual tasks.”
Software is a key infrastructure

- Process control: oil, gas, water, power, …
- Transportation: traffic control, auto pilot, …
- Health care: patient monitoring, device control, e-records, …
- Finance: e-trading, on-line banking, security, automated teller, …
- Defense: weapons control, surveillance, intelligence, …
- Manufacturing: precision milling, assembly, CAD, …

Failing software costs money and lives!
Failing software costs money & lives

- A collection of well-know software failures, G. Tan (Lehigh University)
  http://www.cse.lehigh.edu/~gtan/bug/softwarebug.html

- Illustrative risks to the public in the use of computer systems and related technology
  P. Neumann (SRI)
  http://www.csl.sri.com/users/neumann/illustrative.html

- National Vulnerability Database:
  NVD includes databases of security checklists, security related software flaws, misconfigurations, product names, and impact metrics.
  http://nvd.nist.gov/
Therac - 25

- A software-controlled radiation machine
  - Responsible for at least 6 fatal accidents
  - Two modes of operation:
    - Low power electron beam
Therac - 25

- A software-controlled radiation machine
  - Responsible for at least 6 fatal accidents
  - Two modes of operation:
    - Low power electron beam
    - High power electron beam absorbed by metal target to produce an X-ray beam
Therac - 25

- A software-controlled radiation machine
  - Responsible for at least 6 fatal accidents
  - Two modes of operation:
    - Low power electron beam
    - High power electron beam absorbed by metal target to produce an X-ray beam
  - Race condition
    - Activate HP beam
A software-controlled radiation machine
- Responsible for at least 6 fatal accidents
- Two modes of operation:
  - Low power electron beam
  - High power electron beam absorbed by metal target to produce an X-ray beam
- Race condition
  - Activate HP beam
  - Within 8 seconds, activate LP beam

Some software engineering issues
- Over confidence in software: Lack of hardware interlocks
- Poor user interface w/ high incidence of false alarms
- Failures were infrequent and hard to reproduce
  - Occasional arithmetic overflow
  - Unexpected input sequence
Lost Radio Contact Leaves Pilots On Their Own

Communications error wreaks havoc in the Los Angeles air control system
By LINDA GEPPERT / NOVEMBER 2004

It was an air traffic controller's worst nightmare. Without warning, on Tuesday, 14 September, at about 5 p.m. Pacific daylight time, air traffic controllers lost voice contact with 400 airplanes they were tracking over the southwestern United States. Planes started to head toward one another, something that occurs routinely under careful control of the air traffic controllers, who keep airplanes safely apart. But now the controllers had no way to redirect the planes’ courses.

"You could see airplanes getting awfully close but you're powerless. You can do nothing about it," said Hamid Ghaffari, an air traffic controller at the Los Angeles Air Route Traffic Control Center in Palmdale, Calif., where the crisis occurred. The center is responsible for airplanes flying above 13 000 feet (4000 meters) in 460 000 square kilometers of airspace over Southern California and parts of Arizona, Nevada, and Utah, including the busy McCarran International Airport in Las Vegas, Nev.

The controllers lost contact with the planes when the main voice communications system shut down unexpectedly. To make matters worse, a backup system that was supposed to take over in such an event crashed within a minute after it was turned on. The outage disrupted about 800 flights across the country.

In at least five cases, according to reports in The New York Times and elsewhere, airplanes came within the
Lost Radio Contact Leaves Pilots On Their Own

Communications error wreaks havoc in the Los Angeles air control system
By LINDA GEPPERT / NOVEMBER 2004

So what went wrong on 14 September? In a statement issued the next day, the FAA laid the blame squarely on human error: "Our preliminary findings indicate that the outage was not the result of system reliability but rather an event that should've been avoided had strict FAA operating and maintenance procedures been followed."

Those procedures require that a technician reboot the voice switching system every 30 days.

Inside the control system unit is a countdown timer that ticks off time in milliseconds. The VCSU uses the timer as a pulse to send out periodic queries to the VSCS. It starts out at the highest possible number that the system's server and its software can handle--$2^{32}$. It's a number just over 4 billion milliseconds. When the counter reaches zero, the system runs out of ticks and can no longer time itself. So it shuts down.

Counting down from $2^{32}$ to zero in milliseconds takes just under 50 days. The FAA procedure of having a technician reboot the VSCS every 30 days resets the timer to $2^{32}$ almost three weeks before it runs out of digits.
The jury said Jepperson was 17 percent responsible and Honeywell was 8 percent at fault. It said the two companies produced a defective product used by the airlines.

**Dividing Blame**

The nine-week trial before U.S. District Judge Ursula Ungaro-Benages was an attempt by American to force the two companies to pay a portion of the $300 million already paid out by the airline and its insurers to crash victims and families.
Priorities are changing

From: Bill Gates  
Sent: Tuesday, January 15, 2002 2:22 PM  
To: Microsoft and Subsidiaries: All FTE  
Subject: Trustworthy computing

Trustworthy Computing is computing that is as available, reliable and secure as electricity, water services and telephony.

In the past, we've made our software and services more compelling for users by adding new features and functionality

We've done a terrific job at that, but all those great features won't matter unless customers trust our software. So now, when we face a choice between adding features and resolving security issues, we need to choose security.

These principles should apply at every stage of the development cycle of every kind of software we create
Software engineering …

- A collection of techniques and methodologies for systematic development of complex software systems
- Essential for projects that
  - Involve large teams of developers
  - Are long-lived (supported, maintained, and evolved)
  - Are *safety critical* – failure could result in loss of life or other disaster
  - Required to be of *high integrity* – failure could result in unacceptable financial loss
Software lifecycle

http://www.samsvb.co.uk/images/development_life_cycle.gif
Software development phases

- Requirements analysis
  - Identify requirements, possibly also architecture
- Specification: *What* the system should do
  - Functional and non-functional properties
- Design: *How* system will meet the specs
  - Modules and their interfaces
- Coding …
- Testing
  - Unit testing, integration testing, system testing
- Maintenance
  - Debugging, support client, evolve system
Waterfall model: classic

- Reqt's
  - Specification
    - Design
      - Coding
        - Testing
          - Maintenance
V-shaped model
Incremental/iterative model

Spiral model: incremental model emphasizing risk analysis

Basis for agile software development
Necessary, but insufficient for safety-critical systems

- Testing
  - samples execution behavior, misses some
- Systematic Inspections
  - don’t scale very well, misses some
- Rigorous development processes
  - can help, but most organizations don’t apply them

*Formal methods* are increasingly being used to provide stronger assurances, especially in early phases.
Formal methods approach

“The formal methods approach to software construction is based on viewing a program and its execution as mathematical objects and applying mathematical and logical techniques to specify and analyze the properties and behaviors of these objects.”

-- R. Dewar and A. Pneuli

**Formal specification**: express properties that a system must satisfy

- Formal language: Syntax can be checked, unambiguous semantics
- Abstraction: Above the level of source code, “what” (not “how”)
Why use formal methods?

- Forced to think systematically about issues
  - Leads to better design
  - Earlier detection of inconsistencies and flaws
- Formal spec provides precise reference for
  - Checking that reqt’s are satisfied, conformance
  - Directing latter development phases
  - Documentation within a team of developers
  - Specification matching in support of reuse
  - . . .

“Formal Specification: A Roadmap,” A. van Lamsweerds
Future of Software Engineering (Limerick, Ireland),
The essence of software

“The essence of a software entity is a construct of interlocking concepts: data sets, relationships among data items, algorithms, and invocations of functions. This essence is abstract ... Nonetheless, it is highly precise and richly detailed.

I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation.”
Use formal specifications to

- Precisely document the essence of software
- Check syntactic well-formedness
- Generate test cases
- Check for/classify failures in test runs
- Provide reqt’s for next iteration
- Semantic analysis (design testing)

If my system satisfies these specs, can I infer that it satisfies some additional properties or invariants?

Focus of this course is on “strong formal methods”: formal methods with tool-supported semantic analysis
In this course

- Study representative formal spec languages
  - Predicate logic: Specification and automated proof
  - Alloy: Structural design models based on relations
  - SPARK Ada: Correct-by-construction using software contracts
  - Possibly: Finite State Processes (FSP) - Linear-time model of concurrent system behaviors
  - Possibly: Temporal logic – specification notation for concurrent systems
  - Possibly: Model checking – verification that a model satisfies a specification
In this course

- Learn about their associated methodologies and analysis techniques
- Use tools to check syntax, simulate behavior (early prototyping), verify semantic properties
Summary

- Software is increasingly complex, increasingly ubiquitous
- Cannot trust safety critical software to careful design, programming, and testing
- Formal methods are not a panacea, but can help, and are increasingly becoming necessary to achieve high assurance
- We will learn several formal methods—different development stages, different methodologies, different application domains
- We will emphasize strong formal methods
Acknowledgments

- Others borrow heavily from: