# Rigorous Software Development
# CSCI-GA 3033-009

Instructor: Thomas Wies

Spring 2013

Lecture 1

# Important Facts

- Office Hours: Thu 3-4pm, or by appointment
- Office: CIWW 407
- Course web site: http://cs.nyu.edu/wies/teaching/rsd-13
- Mailing list: csci_ga_3033_009_sp13@cs.nyu.edu

# Prerequisites

- Discrete structures: sets, relations, functions
- Basic algorithms (undergraduate level)
- Mathematical maturity: comfort with notation, understand and write proofs
- Familiarity with Java-like languages
- (Moderate) programming experience

# Grading

- Weekly Assignments: 30% (starting 02/03/12)
- Term Project: 30%
- Final Exam: 40%

# Course Material

- *Software Abstractions: Logic, Language, and Analysis (revised edition)*. Daniel Jackson, The MIT Press, 2006
- *The Formal Semantics of Programming Languages*. Glynn Winskell, The MIT Press, 1993
- Additional material on the course web site

# Software Reliability

Driving force to use computer systems:

Increase the safety of technological artifacts

But can we really build safe computer systems?

“Software and cathedrals are much the same. First we build them, then we pray.”

*Sam Redwine*

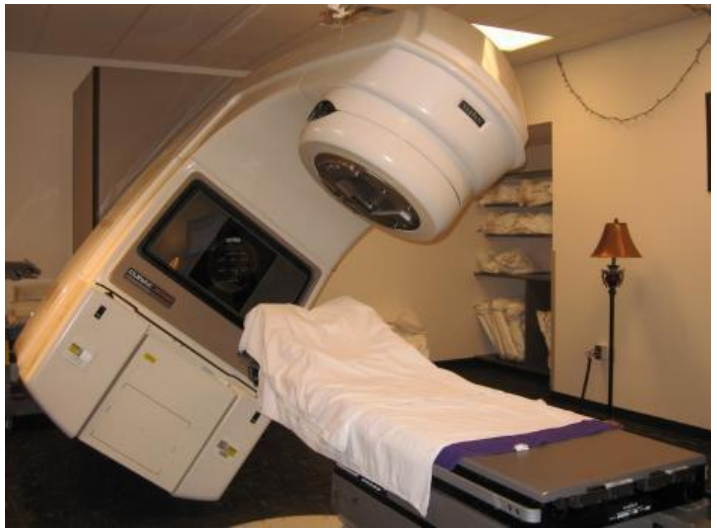
# Maiden Flight of Ariane 5 Rocket

- Ariane 5 exploded on its first test flight in 1996
- Cause: failure of flight-control software due to overflow in floating point to integer conversion

- Financial loss: $500,000,000 (including indirect costs: $2,000,000,000)

# Therac-25

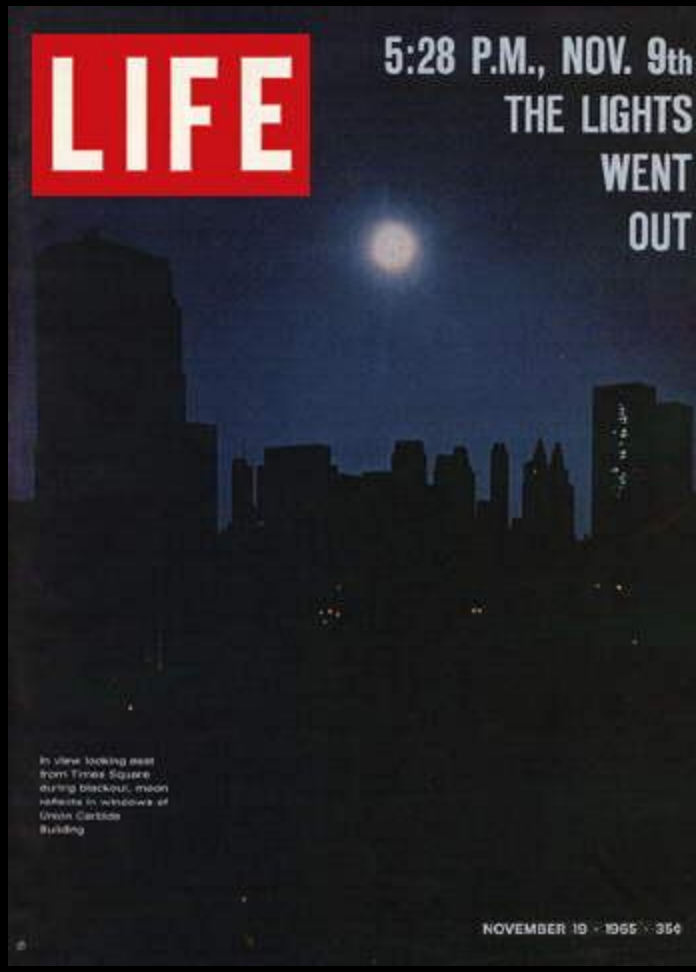
- Radiation therapy machine
- Two modes:
  - X-ray
  - electron-beam



- Race condition in software caused use of electron-beam instead of X-ray
- six cases of radiation poisoning between 1985 and 1987, three of them fatal

# Northeast Blackout in 2003

- Cascading failure of power grid due to lightning stroke
- Race condition in energy control system stalled alarm in control room for > 1 hour
- Multiple server failures due to unprocessed events
- 55 million people affected



# Economics of Software Errors

Estimated annual costs of software errors in the US (2002)

$60 billion (0.6% of GDP)

Estimated size of the US software industry (2002)

$240 billion (50% development)

Estimated

50%

of each software project is spent on testing

# Testing

Software validation the “old-fashioned” way:

- Create a test suite (set of test cases)
- Run the test suite
- Fix the software if test suite fails
- Ship the software if test suite passes

“Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence.”

*Edsger W. Dijkstra*

Very hard to test the portion inside the “if" statement!

```
input x
if (hash(x) == 10) {
    ...
}
```

# Verification

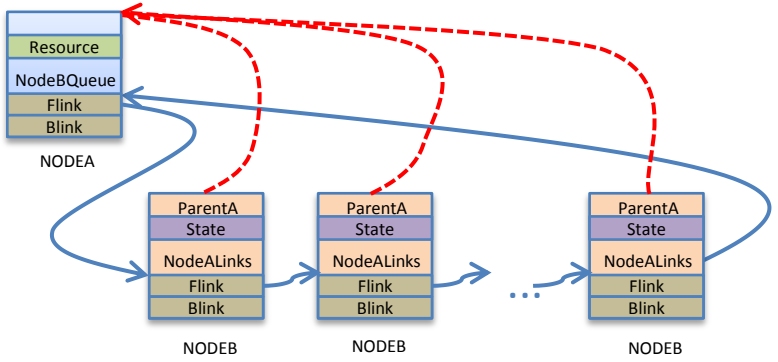
- **Verification:** formally prove that a computing system satisfies its specifications
  - **Rigor**: well established mathematical foundations
  - **Exhaustiveness**: considers **all** possible behaviors of the system, i.e., finds **all** errors
  - **Automation**: uses computers to build reliable computers
- **Formal Methods:** general area of research related to program specification and verification.

# Success Stories of Formal Methods

- Astrée Static Analyzer
  - Developed by Patrick Cousot's group and others
  - Verify absence of runtime errors in C code for Embedded Systems
  - Industrial applications include verification of Airbus fly-by-wire software


# Success Stories of Formal Methods

- Static Driver Verifier, HAVOC, and VCC
  - Developed at Microsoft Research
  - Verification of OS code
  - Applications:
    - Windows Device Drivers
    - Windows File System
    - Windows Hypervisor




```
#define FIRST_CHILD(x) x->NodeBQueue.Flink
#define NEXT_NODE(x) x->NodeAlinks.Flink

__type_invariant(PNODEA x){
    ENCL_NODEA(FIRST_CHILD(x)) != x ==>
      ENCL_NODEB(FIRST_CHILD(x))->ParentA == x
}

__type_invariant(PNODEB y){
    NEXT_NODE(y) != &(y->ParentA->NodeBQueue) ==>
      y->ParentA == ENCL_NODEB(NEXT_NODE(y))->ParentA
}
```

“Beware of bugs in the above code; I have only proved it correct, not tried it.”

*Donald Knuth*

You can only verify what you have specified.

Testing is still important, but can we make it less impromptu?

# Rigorous Software Development

Driving force to use computer systems:
Increase the safety of technological artifacts

But can we really build safe computer systems?

In this course you will learn

- how to use software to build reliable software
  - automated testing
  - automated program verification
- how formal methods tools work under their hoods

# Overview

## Topics

- Modeling Software
- Design by Contract
- Runtime Assertion Checking
- Automated Test Case Generation
- Extended Static Checking
- Semantics of Programming Languages
- Formal Verification
- Static Analysis

## Tools

- Alloy
- JML
- JMLUnit
- Korat
- Dafny

# Dafny Demo

# The Alloy Tool

http://alloy.mit.edu

Daniel Jackson

# Alloy

- Analyzes micro models of software
- Helps to
  - identify key properties of a software design
  - find conceptual errors (**not** implementation errors)
- **Small scope hypothesis**: many properties that do not hold have small counterexamples
- Exhaustively search for errors in all instances of bounded size

# Example Applications

- Security protocols
- Train controllers
- File systems
- Databases
- Network protocols
- Software design/testing/repair/sketching

Many examples are shipped with Alloy. More can be found on the Alloy website.

# Alloy in a Nutshell:
# A Simple Address Book