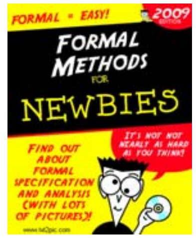
# CISC422/853: Formal Methods in Software Engineering: Computer-Aided Verification




## Topic 0: Intro, Motivation, Overview, Admin

**Juergen Dingel**
**Jan 5, 2009**



# About Me

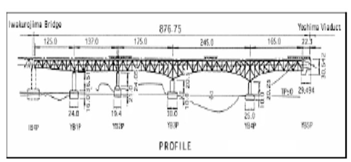
- Born and raised in Germany
- Undergrad in Berlin, Germany
- Grad school at CMU in Pittsburgh, PA
- At Queen's since January 1, 2000
- Research interests:
  - software development, programming languages
  - all things having to do with supporting software development through modeling and analysis: E.g.,
    - software model checking
    - foundations of UML and MDD
    - run-time monitoring, testing, etc



# About (some of) our research

- Foundations of Model-Driven Development (MDD)
  - **Main goal:** Develop notations, methods, tools to
    - **increase level of abstraction**
      - through use of models
    - **increase degree of automation**
      - e.g., through code generation from models

    in software development
  - *"Models, rather than code, form the primary artifact"*
  - *"Models are the new code"*
  - *"Put more `engineering' into software engineering"*
  - *"MDD = Computer-aided manufacturing for IT"*

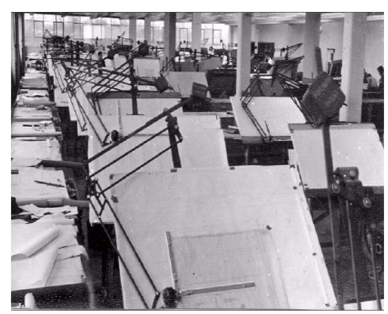


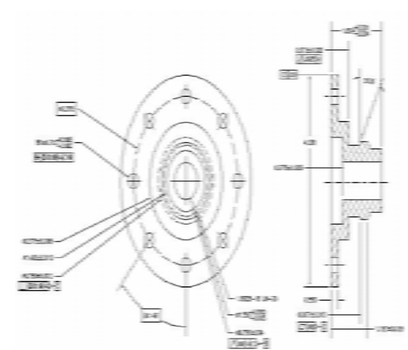



# MDD = computer-aided manufacturing for IT

- **Mechanical design** from 1800 to about 1980:
  1. Draftsmen create 3-view drawings
  2. Machinists create parts from drawings

  ⇒ laborious, error-prone, inefficient





## MDD = Computer-aided manufacturing for IT (Cont'd)

- **Concorde (1976 – 2003)**
  - > 100,000 drawings
  - in 2 languages, using both metric and imperial systems

  ⇒ worked, but 7x over budget



## MDD = Computer-aided manufacturing for IT (Cont'd)

- **Mechanical design** from about 1972: CAD/CAM
  1. Create drawings with computer (CAD)
  2. From drawing, computer automatically generates program to drive the milling and CNC machines (CAM)

  ⇒ much better analysis capabilities and productivity

  ⇒ CAD/CAM has revolutionized manufacturing
- **Most IT development today:**
  - models are still predominantly for communication
  - MDD suggests to
    - make computers "understand" the models, and
    - automatically generate code from models

This course is **not** about MDD, but it is about models and analysis

I am looking for grad students to help us make this vision a reality



## Next few lectures

- **Motivation**
  - Software development is hard
  - It won't get any easier
  - Need more powerful tools and techniques
- **Overview**
- **Admin stuff**



## Complexity of today's software

| Product | Lines of code |
|---|---|
| Microsoft Word in 1983 | 27,000 |
| Micro | |
| Mi | |
| Tax processing system for IRS | > 100 million |
| Pacemaker | > 100,000 |
| C | |
| Ce | |
| Ca | |
| Car in 2010 | **?** |

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

**But perhaps "Lines of code" is a poor measure of complexity?!**

[Source: "Why Software Fails". R.N. Charette. IEEE Spectrum, Sept 2005]

# Complexity of today's software (Cont'd)

- State of a program P
  - snapshot of execution of P
  - formally: mapping of variables in P to values
- State space of P
  - set of reachable states of P
- State spaces can be very large
  - quadrillion values
  - What about Windows XP?

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



# Consequences of this complexity

- **Computers still "under-utilized"**

*"It is widely agreed that the main obstacle to "help computers help us more" and relegate to these helpful partners even more complex and sensitive tasks is not inadequate speed and unsatisfactory raw computing power in the existing machines, but our limited ability to design and implement complex systems with a sufficiently high degree of confidence in their correctness under all circumstances"*

*Amir Pnueli, Turing Award Winner*
*in foreword to [CGP99]*



# Consequences of this complexity (Cont'd)

- **Failing software**
  - money
    - Examples: ESA Ariane 5, Mars Climate Orbiter, US telephone system, …
    - Cost of errors in software in US in 2001:

US$ 60B

[**Source**: US National Institute of Standards and Technology]

  - lives
    - Therac 25, …

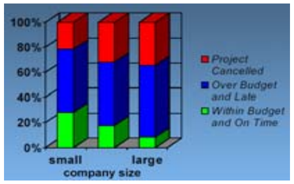
## More details

- Peter Neumann's `www.risks.org`
- Ivars Peterson. Fatal Defect: Chasing Killer Computer Bugs. Vintage Books, New York, 1996.



# Consequences of this complexity (Cont'd)

- **Failing software development**
  - **According to the 1995 Standish report**
    - 94 of 100 projects have to be restarted
    - 31% of all projects are cancelled
    - Of the ones not cancelled
      - 23% have cost overruns of > 50%
      - 67% have time overruns of > 50%




  - **Most costly activity in SW development**:
    - Quality assurance
  - **Examples:**
    - Luggage Handling system at Denver airport, Canadian Gun Registry, US FAA Advanced Automation System, German Tax Processing system, …

# Example: Therac-25 (1985-87)

- Radiotherapy machine with SW controller
- Several deaths due to burning
- **Problems:**
  - "poor SWE practices",
  - error messages cryptic/undocumented,
  - false error messages,
  - user interface w/o safety checks
- **References:**
  - N.G. Leveson and C.S. Turner. An Investigation of the Therac-25 accidents. Computer, 26(7):18-41, July 1993.



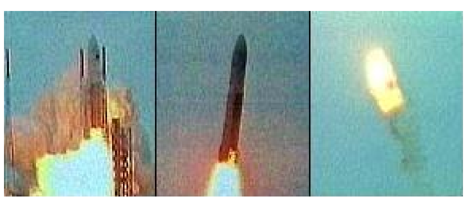
# Example: "Browser War" (MS vs NS)

- **In a nutshell:**
  - From 1995 to 1997 NS concentrated on features at the expense of good design
  - MS hurried to get IE going, but took time to restructure IE3.0 (NT built from scratch, shared components in Office)
  - By 1997, NS C4.0 had 130 developers, 3M loc
  - Two months not enough to rearchitect NS C4.0
  - NS decides to start from scratch with C6.0
  - C6.0 never finished, developers reassigned to C4.0
  - C5.0 open source, but nobody wants to work on it
  - MS wins Browser War, AOL buys NS
- NS C4.0 still contains 1.2M loc
- **Reference:**
  - [CY98]



# Example: ESA Ariane 5 (June 1996)

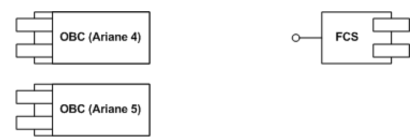
- On June 4, 1996, unmanned Ariane 5 launched by ESA explodes 40 seconds after lift-off
- One decade of development costing $7billion lost
- Rocket and cargo valued at $500million destroyed



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



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




- Example of how not to do reuse:
  - Parts of Flight Control System (FCS) taken from Ariane 4
  - Horizontal velocity much greater for Ariane 5
  - Unprotected conversion operation in FCS causes error
  - On-board computer (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: NASA Mars Climate Orbiter (1999)

- Some programs worked in English units, some metric units
- Conversion from English to metric forgotten
- Instead of 65 miles probe attempted to orbit 65 km

  (40 miles) above Mars
- $327M lost
- References:
  - **`http://mars.jpl.nasa.gov/msp98/orbiter/`**



# Example: FAA Advanced Automation System (2001)

*"FAA's major modernization project, the Advanced Automation System (AAS), was originally estimated to cost $2.5 billion with a completion date of 1996. The program, however, experienced numerous delays and cost overruns, which were blamed on both FAA and the primary contractor, IBM. In 1994, FAA cancelled part of the program and split the remaining systems into three phases, and in several cases, re-bid the contracts. […] According to the General Accounting Office, almost $1.5 billion of the $2.6 spent on AAS was completely wasted."*

Reference:
**`www.house.gov/transportation/press/press2001/release15.html`**



# Example: Intel's Pentium FDIV Bug

- In summer 1994, Prof Thomas Nicely of Lynchburg College first identified a problem with the floating point processor of Intel Pentium chips
- The result of entering

  **`(4195835/3145727) * 3145727 - 4195835`**

  into the Windows calculator was **`512`**, not **`0`**
- Intel's PR disaster:
  - Nov 1994: Intel disputes the severity of the problem
  - Intel offers to replace chip based on need
  - Intel stock price falls
  - Dec 1994, Intel offers to replace all chips
- Total cost of bug to Intel estimated at: **$475million**

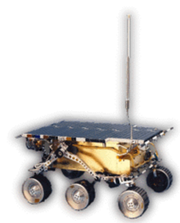


# Example: NASA Mars PathFinder

- Launched December 4, 1996
- A few days after landing on Mars, the Sojourner rover tasks began missing their deadlines causing **total system resets**
- Problem: **priority inversion** is the scenario where a low priority task holds a shared resource that is required by a high priority task
- Reference:

  **`http://research.microsoft.com/en-us/um/people/mbj/mars_pathfinder/Authoritative_Account.html`**

Mars Pathfinder launched on 4 December 1996



# Example: Skype

The New York Times
nytimes.com

PRINTER-FRIENDLY FORMAT SPONSORED BY

August 17, 2007

## Error in Skype's Software Shuts Down Phone Service

By BRAD STONE

SAN FRANCISCO, Aug. 16 — The online telephone service Skype was not working for much of the day on Thursday, leaving its 220 million users, some of them small businesses that had given up their landlines, without a way to call colleagues, customers and friends.

Executives at Skype, a division of eBay that is based in Luxembourg, said its engineers worked throughout the day to bring the service back online. But they said that while they had pinpointed the source of the problem, they still did not know why it had resulted in a network failure, and they could not ensure that the service would be running smoothly again by Friday.

"There is a chance this could go on beyond tomorrow, but it's our hope that it's going to be resolved," Kurt Sauer, Skype's chief security officer, said. "What happened today was caused by a unique set of events, the genesis of which is not entirely understood."



# Example: The Blackout Bug

- 50 Million people w/o electricity
- Worst black out in North American history
- Cause: Race condition in alarm system (10^6Loc of C)

## 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>



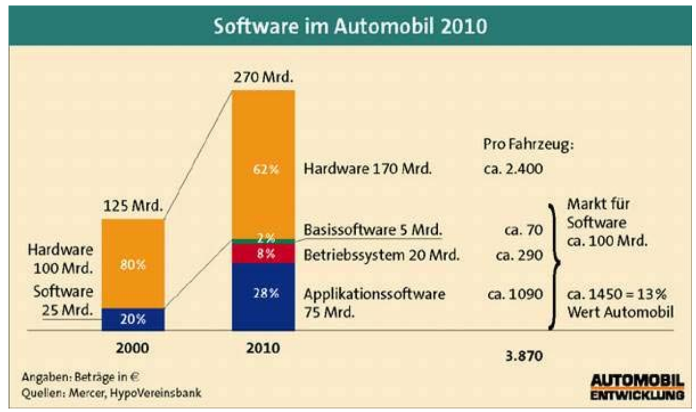
# In the future …

- **Our dependency on SW will grow**
  - More software in almost everything
    - health care
      - computer-aided surgery
      - tele-medicine
      - HL7 standards (`www.hl7.org`)
        - for exchange, management and integration of electronic healthcare information
      - networked watches, appliances, …
    - cars
      - "drive by wire"
    - infrastructure
      - intelligent highways
    - Clothes
      - "smart" diapers

**The "smart" diaper moisture detection system. Siden, J.; Koptioug, A.; Gulliksson, M. Microwave Symposium Digest, 2004 IEEE MTT-S International 2, June 2004 Page(s): 659 - 662**



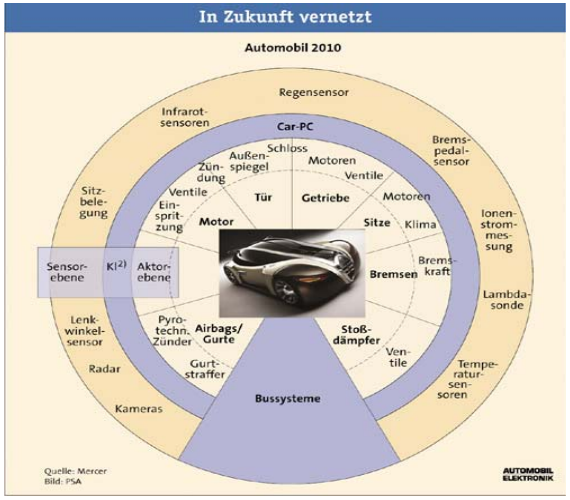
# For Example: In Cars




[source: www.automagazin.de]

In English:

- In 2010, software will make up 13% of a car's overall value
- Compared to 2000, the market for automotive software will quadruple to 100 Billion Euro




## For Example: In Cars (Cont'd)

[source: www.automagazin.de]

In English:

- There are up to 80 separate electronic systems and components in a car. In 2010, all of these could be networked. Their functionality will then be solely driven by software.



## In the future … (Cont'd)

- **SW will get more and more complex**
  - Because it will …
    - … be even larger
    - … carry out more complex tasks
    - … be more concurrent
      - *"In the future, applications will need to be **concurrent** to fully exploit CPU throughput gains"* [Sut05]
    - … therefore potentially be more buggy
      - *"I conjecture that most multithreaded-general purpose applications are so full of concurrency bugs that - as multicore architectures become commonplace – these bugs will begin to show up as system failures"* [Lee06]
    - … have to function in more complex environments



## In the future … (Cont'd)

| Product | Lines of code |
|---|---|
| Microsoft Word in 1983 | 27,000 |
| Microsoft Word in 2005 | > 1 million |
| Microsoft XP | > 45 million |
| Tax processing system for IRS | > 100 million |
| Pacemaker | > 100,000 |
| Cellphone in 2005 | 2 million |
| Cellphone in 2010 | 20 million |
| Car in 2005 (BMW) | 7.5 million |
| Car in 2010 (GM) | 100 million |

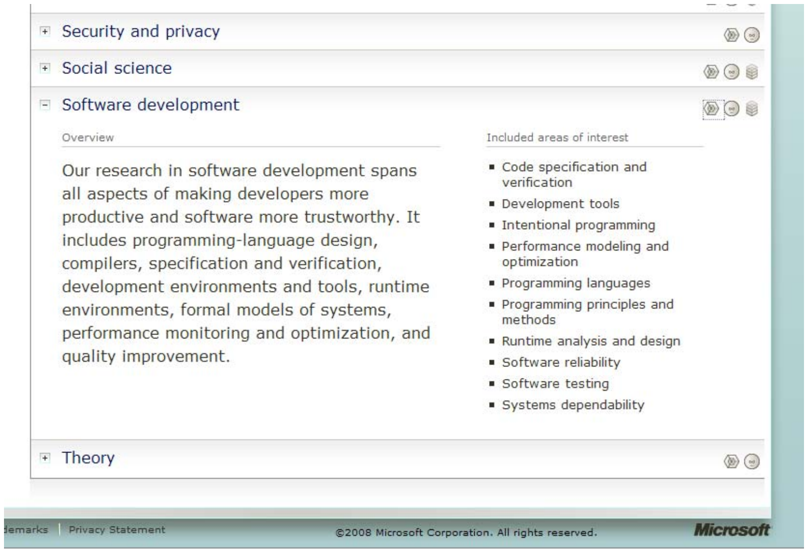
[Source: "Why Software Fails". R.N. Charette. IEEE Spectrum, Sept 2005]



## In the future: Conclusion

- **Potential costs of SW failure will grow while likelihood of failure will increase**
  - Most vulnerable:
    - Safety critical systems
    - Concurrent, distributed, and embedded systems
- **We will need**
  - better ways to deal with complexity
  - more powerful QA techniques
    - achieving acceptable levels of quality in, e.g., large concurrent or embedded systems with standard techniques is very hard if not impossible } More on this later…
  - see, for instance,
    - 1999 PITAC-report (`www.nitrd.gov/pitac/report/`)
    - research at MSR

## http://research.microsoft.com/apps/dp/areas.aspx






# What can we do?

## Ways to control complexity

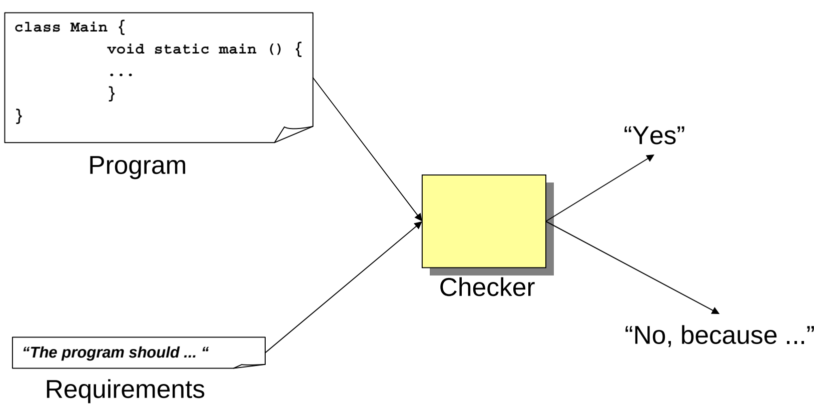
- Reuse, decomposition (e.g., modularity, divide & conquer)
- Improve abstraction mechanisms
  - e.g., through use of models such as **finite state machines**
- Improve analysis
  - e.g., through **model checking**
    - on models
    - directly on software

**And this is what this course is about!**

**Key ingredients for "Model-Driven Development"**



# Software Verification: The Dream

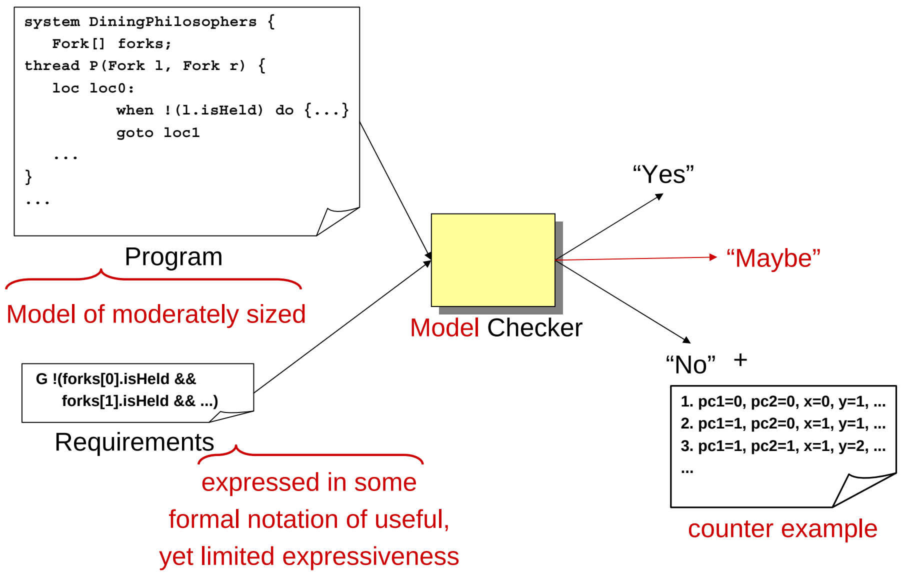





# SW verification: Fundamental limitations

- **Some assumptions are always necessary**
  - Correct execution of a program **relies on many things** (e.g., editor, compiler, libraries, optimizer, hardware)
  - ⇒ correct workings of some things will have to be **assumed**
- **Some formality is necessary**
  - Must express requirements in **precise, unambiguous terms**
  - E.g., propositional logic, predicate logic, temporal **logic**
- **Precision/scalability tradeoff**
  - The more complex the analysis, the less likely it will scale
  - ⇒ have to find **happy medium**
- **Undecidability**
  - Some properties of programs are **undecidable**
  - ⇒ must be careful we **don't ask for something impossible**

# Software Verification: The State of the Art






# Model Checking

- **Typically:**
  Automatic technique based on exhaustive state space exploration to decide if a finite state machine satisfies a temporal logic specification
- Developed in early 1980s; has been tremendously successful for hardware and protocol verification
  - All large chip manufacturers (e.g., Intel, Motorola, Cadence) use model checking
- **Keys to success**
  - full automation (allows to hide complexity)
  - counter examples (allow developers to see precisely where things go wrong)
  - optimization techniques (e.g., abstraction, Partial Order Reduction, Binary Decision Diagrams)



# Model Checking (Cont'd)

- **Challenges**
  - state space explosion through
    - large number of variables
    - large number of values variables can take on
    - high degree of non-determinism (e.g., through large number of unsynchronized parallel processes)
- **Successes**
  - new optimization techniques (e.g., Boolean programs)
  - lots of publicly available tools (e.g., Bandera, VeriSoft, JPF)
  - already some industrial success stories (e.g., SLAM at MSR)
  - 2008 Turing Award for Clarke, Emerson, and Sifakis



# This Course

- **Introduction to fundamental concepts, techniques, tools, and research questions in model checking**
- Other forms of software verification that we will **not** consider:
  - proofs of correctness
    - e.g., Hoare logic, weakest preconditions
    - because it doesn't scale
  - theorem proving
    - because it doesn't scale

  (However, both areas of research have been very influential and we will use some of their results

  E.g., MSR's Spec# `http://research.microsoft.com/en-us/projects/specsharp/`)

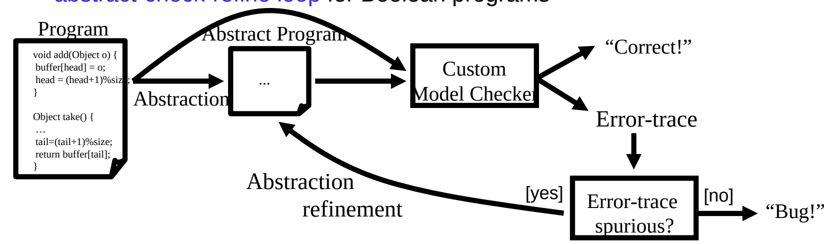
# Success Story 1: SLAM Project at MSR

- Started in 2000, hired lots of “formal people”
- **SLAM starting points:**
  - Buggy third-party device drivers are big headache for MS
    - more than 5,000 device drivers for Windows in the field
    - Windows Kernel interface provides more than 800 functions
    - MS provides Driver Development toolkit to facilitate development
  - Device drivers good domain for formal analysis, because
    - relatively small (typically less than 100,000 lines of C code)
    - interface rules mostly control oriented
- **SLAM goal:**
  - use model checking to check rigorously that code obeys “interface usage rules”



# Success Story 1: SLAM Project at MSR

- **SLAM main ingredients:**
  - Boolean programs
    - subset of C
    - conservative abstraction of original C program
    - many difficult problems (e.g., Halting problem) are decidable
  - abstract-check-refine loop for Boolean programs




  - innovative use of established formal analysis techniques, e.g.,
    - model checking
    - theorem proving
    - static analysis

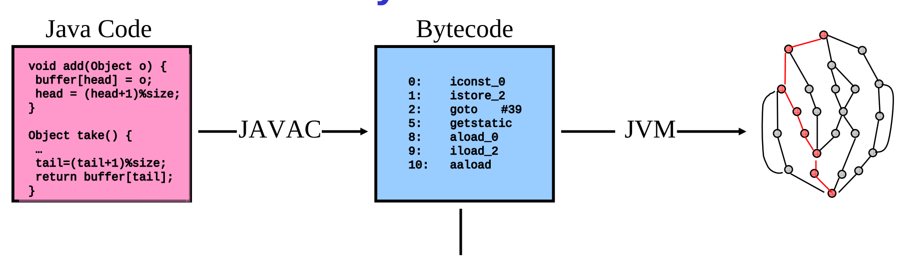


# Success Story 1: SLAM Project at MSR

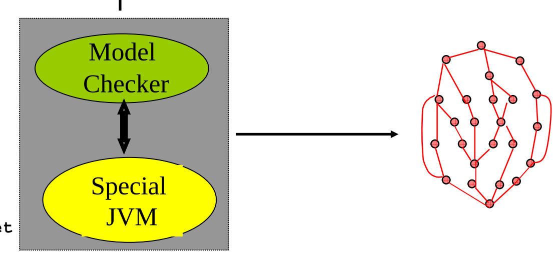
- **SLAM mile stones:**
  - 2001: SLAM finds its first bug
  - March 2002: demo to Bill Gates
  - August 2002: Driver Quality Team formed to
    - gradually hand over project to Windows development group
    - extend SLAM to a user-friendly tool SDV (Static Driver Verifier)
  - April 2003: decision made to turn SDV into a product
  - Nov 2003: SDV presented at Driver Developer Conference
  - Aug 2005: beta-version of SDV released
- **References:**
  - [BCLR04]: Th.Ball, B.Cook, V.Levin, S.Rajamani: SLAM and Static Driver Verifier: Technology Transfer of Formal Methods inside Microsoft. MSR-TR-2004-08.
  - **`www.research.microsoft.com/slam`**
  - **`www.microsoft.com/whdc/devtools/tools/sdv.mspx`**



# Success Story 2: Java PathFinder




- Developed at NASA AMES
- Helped find bugs in spacecraft software
- Now open source on SourceForge at `javapathfinder.sourceforge.net`
- Possibly more on this later



# CISC422/853: Contents

1. **A few words on concurrency**
2. **Modeling: How to describe behaviour of a software system?**
   - finite automata
3. **Intro to 2 software model checkers**
   - Bogor (Santos group at Kansas State University)
   - Spin (G. Holzmann at JPL)
4. **Model checking I**
   - algorithms for basic exploration
5. **Specifying: How to express properties of a software system?**
   - assertions, invariants, safety and liveness properties
   - Linear temporal logic (LTL) and Buechi automata
   - Computation Tree Logic (CTL)
6. **Model checking II**
   - algorithms for checking properties

Assignment 1 (Bogor)

Assignment 2 (Spin)

Assignment 3 (Theory)



# CISC422/853: Contents (Cont'd)

8. **Optimizations**
   - Partial order reduction
   - Static analysis and slicing
9. **Overview of software model checking tools**

Assignment 4 (slicing)

**Final exam**

- Covering the theoretical parts and some of the practical

**Projects (for grad students)**

- 2 possibilities
  - practical: experimentation with a tool
  - theoretical: look at some details of the theory
- I will provide list of suggestions
- In both cases, I expect project proposal, presentation & summary paper



# CISC422/853: Goals

- Provide introduction to fundamental
  - concepts,
  - techniques,
  - tools and
  - research questions

  in model checking
- Give you some ideas for your own research
- Have fun!



# CISC422/853: Expected Background

- Programming
  - concurrent
  - object-oriented
- Discrete maths
  - sets, functions, relations, automata
- Logic
  - propositional and predicate logic

## CISC422/853: Evaluation

- For undergrads
  - 4 assignments 60%
    - In groups of 1-2 students
  - Final exam 40%
- For grads
  - 4 assignments 50%
    - In groups of 1-2 students
  - Final exam 20%
  - project-related work 30%
    - In groups of 1-2 students
    - Proposal, presentation, summary paper



## CISC422/853: Evaluation

- Assignments
  - A1 using Bogor
  - A2 using Spin
  - A4 using Java
  - A3 using pencil and paper

Tutorials will be given to introduce these tools; Details tba



## CISC422/853: Material

- Lecture slides
  - will be posted
- Spin book
  - Gerard Holzmann. The Spin Model Checker: Primer and Reference Manual. Addison Wesley. 2004. ($80)
  - You are encouraged to purchase it, but don't have to
  - At least 3 copies will be available in Douglas library
- Course notes and papers
  - distributed by instructor
- Online information (code and documentation)
  - `www.cs.queensu.ca/~cisc853` **with link to WebCT forum**
  - `www.spinroot.com` **// Spin website**
  - `bogor.projects.cis.ksu.edu` **// Bogor website**



## CISC422/853: Material (Cont'd)

- Lectures
  - I highly recommend coming to lectures
  - Text book doesn't cover everything (it's mostly for the Spin part)
  - Slides "supersede" text book in case of "conflict"
- Tutorials
  - Every practical assignment will be preceded by a tutorial providing a short introduction to the tool/software the assignment asks you to use
  - Led by TA Scott
  - Dates and times: tba

# References


**Books:**

- [CGP99]: E.Clarke, O.Grumberg, D.Peled. *Model Checking*. MIT Press. 1999.
- [Pet96]: I. Peterson. *Fatal Defect: Chasing Killer Computer Bugs*. Vintage Books, New York. 1996.
- [CY98]: M.A. Cusumano, D.B. Yoffie. *Competing on Internet Time: Lessons from Netscape and Its Battle with Microsoft*. Free Press. 1998.

**Articles:**

- [Gle96]: J. Gleick. *A Bug and a Crash: Sometimes a Bug Is More Than a Nuisance*. 1996. Available at www.around.com.
- [LT93]: N.G. Leveson and C.S. Turner. An Investigation of the Therac-25 accidents. Computer, 26(7):18-41, July 1993.
- [Man02]: C. Mann. *Why Software Is So Bad.* Technology Review. July/August 2002.
- [Eco03]: *Building a better bug-trap*. The Economist, June 19, 2003.
- [BCLR04]: Th.Ball, B.Cook, V.Levin, S.Rajamani: SLAM and Static Driver Verifier: Technology Transfer of Formal Methods inside Microsoft. MSR-TR-2004-08.
- [Sut05]: H. Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3), March 2005
- [Lee06]: Edward A. Lee. The problem with threads. Computer, 39(5):33– 42, May 2006.
- [Pou04]: K. Poulsen, "Tracking the blackout bug", SecurityFocus, http://www.securityfocus.com, Apr. 2004




# References (Cont'd)


**Web Pages:**

- [Neu04]: P. Neumann. *The Risk Digest*. Available at **www.risks.org.**
- **www.research.microsoft.com/slam**
- **www.microsoft.com/whdc/devtools/tools/sdv.mspx**
- **www.house.gov/transportation/press/press2001/release15.html**
- **mars.jpl.nasa.gov/msp98/orbiter/**
- **www.ima.umn.edu/~arnold/disasters/ariane.html**





# Acknowledgements


- Course designed following
  - CIS842: Specification and Verification of Reactive Systems at Kansas State University
  - G. Holzmann. The Spin Model Checker: Primer and Reference Manual. Addison Wesley. 2004.
- Thanks to John Hatcliff, Matt Dwyer, Robby, and Gerard Holzmann for letting me use some of their slides