Welcome to Mission Critical Systems

EECS 4315

wiki.eecs.yorku.ca/course/4315/

Instructor

- Name: Franck van Breugel
- Email: franck@eecs.yorku.ca
- Office: Lassonde Building, room 3046
- Office hours: Monday and Wednesday, 10:30-11:30 or by appointment

Evaluation

- 3 quizzes (5% each)
- midterm (15%)
- final exam (30%)
- project (40%)

Evaluation

3 quizzes (5% each)

- January 11: programming (during lab)
- January 21: written (during lecture)
- February 8: programming (during lab)

Evaluation

Midterm (15%)

February 27: written (during lecture)

Evaluation

Final exam (30%)

- written (15%)
- programming (15%)

Exam period

Evaluation

Project (40%)

- January 25: install JPF (5%)
- February 15: draft proposal (2%)
- February 25: proposal (3%)
- March 8: first progress report (5%)
- March 22: second progress report (5%)
- Exam period: deliverables (20%)

Labs

- Lassonde Building, room 1004
- Friday, 10:00-11:00
- Two quizzes will be held during the labs.

Quizzes and midterm

- Students with a documented reason for missing a quiz or the midterm, such as illness, compassionate grounds, etc., will have the weight of the missed quizzes and midterm shifted to an extra exam. This extra exam will cover all the material covered in the course.
- During quizzes, the midterm and the final exam, students are expected to do their own work. Looking at someone else's work during a test, talking during a test, using aids not permitted (such as a phone) during a test, and impersonation are all examples of academically dishonest behaviour.

Drop deadline

March 8

Until this date you can drop the course without getting a grade for it.

`https://registrar.yorku.ca/enrol/dates/fw18` contains important dates.

Academic honesty

"If you put your name on something, then it is your work, unless you explicitly say that it is not."

Examples of academic dishonesty include

- copying code,
- looking at someone else's work during a quiz,
- talking during a quiz,
- using aids not permitted (such as a phone) during a quiz,
- impersonation.

Read `http://secretariat-policies.info.yorku.ca/policies/academic-honesty-senate-policy-on/` for more details.

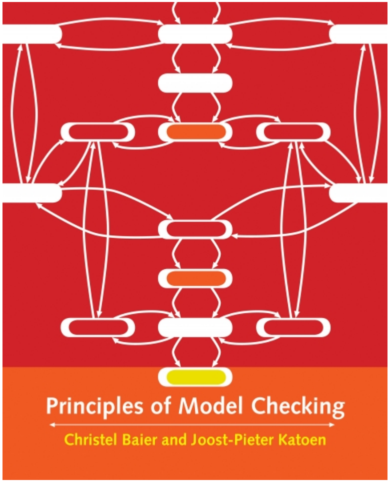
Academic honesty

Academic honest behaviour of students increases the value of your degree.

- The instructors will do their best to design quizzes and policies that promote honest behaviour.
- The students are expected to behave honestly.

Textbook

Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.


Principles of Model Checking
Christel Baier and Joost-Pieter Katoen

Source: mitpress.mit.edu

Textbook

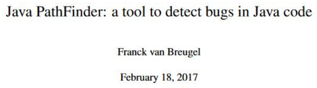
The textbook is required for this course.

Studying only the slides and your lecture notes may not be sufficient. There may be questions on quizzes, midterm and final exam about material that is not covered in class. Therefore, you should study the textbook.

Although you need to memorize some material, most of the material you have to understand.

Notes

Franck van Breugel. *Java PathFinder: a tool to detect bugs in Java code*. 2019.


Java PathFinder: a tool to detect bugs in Java code

Franck van Breugel

February 18, 2017

Other reading material

Will be posted on the course wiki.

Students are expected to . . .

- attend the lectures (3 hours per week)
- attend the lab (1 hour per week)
- prepare for the lab (2 hours per week)
- study the textbook and other reading material, and work on the project (3 hours per week)

Expected learning outcomes

- The ability to explain the importance of safety-, mission-, business-, and security-critical systems.
- Demonstrated knowledge of the importance of good software engineering practices for critical systems.
- The ability to use rigorous software engineering methods to develop dependable software applications that are accompanied by certification evidence for their safety and correctness.
- Knowledge of the method and tools using deductive approaches (such as theorem proving).
- Knowledge of methods and tools for algorithmic approaches (such as model checking, bounded satisfiability) etc.
- Knowledge of the theory underlying deductive and algorithmic approaches.
- The ability to use industrial strength tools associated with the methods on large systems.

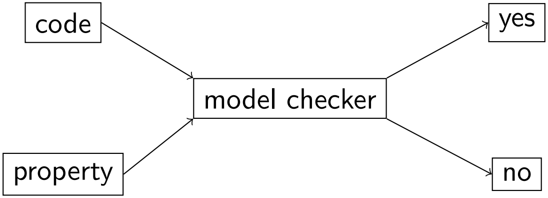
Expected learning outcomes

- The ability to explain the importance of safety-, mission-, business-, and security-critical systems.
- Demonstrated knowledge of the importance of good software engineering practices for critical systems.
- The ability to use rigorous software engineering methods to develop dependable software applications that are accompanied by certification evidence for their safety and correctness.
- Knowledge of the method and tools using deductive approaches (such as theorem proving).
- Knowledge of methods and tools for algorithmic approaches (such as model checking, bounded satisfiability) etc.
- Knowledge of the theory underlying deductive and algorithmic approaches.
- The ability to use industrial strength tools associated with the methods on large systems.

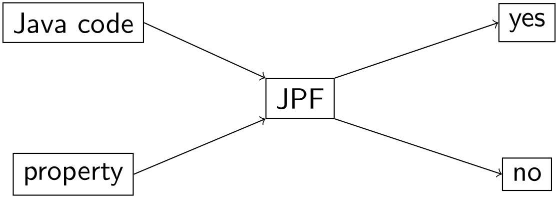
Main topics

- Model checking.
- Java PathFinder.
- Concurrent programming in Java.

Model checking



Java PathFinder (JPF)



Properties: uncaught exceptions, deadlocks, data races, . . .

Concurrent programming in Java

In Operating System Fundamentals (EECS 3221), the following topics are covered:

- Threads (multi-thread programming, multi-core systems, thread libraries and implementations).
- Process Synchronization (critical section problem, deadlocks, software and hardware solutions, mutex locks, semaphores, monitors, classic problems).

We will review this material in the context of Java (EECS 3221 uses C), consider other concurrency primitives, and apply model checking to concurrent Java code.

System Specification and Refinement (EECS 3342)

In EECS 3342, you

- construct high level, abstract mathematical models of a system (consisting of both the system and its environment) amenable to formal reasoning.

In this course, you

- work with models that are automatically generated from Java bytecode.

System Specification and Refinement (EECS 3342)

In EECS 3342, you

- apply set theory and predicate logic to express properties.

In this course, you

- implement listeners in Java to check properties.

System Specification and Refinement (EECS 3342)

In EECS 3342, you

- use practical tools for constructing and reasoning about the models.

In this course, you

- use, modify and extend practical tools for checking properties of the models.

System Specification and Refinement (EECS 3342)

In EECS 3342, you

- use a theorem prover (which often needs input from you).

In this course, you

- use a model checker (which needs no input from you).

System Specification and Refinement (EECS 3342)

In EECS 3342, you

- focus on designing code that is correct by construction.

In this course, you

- focus on finding bugs in code.

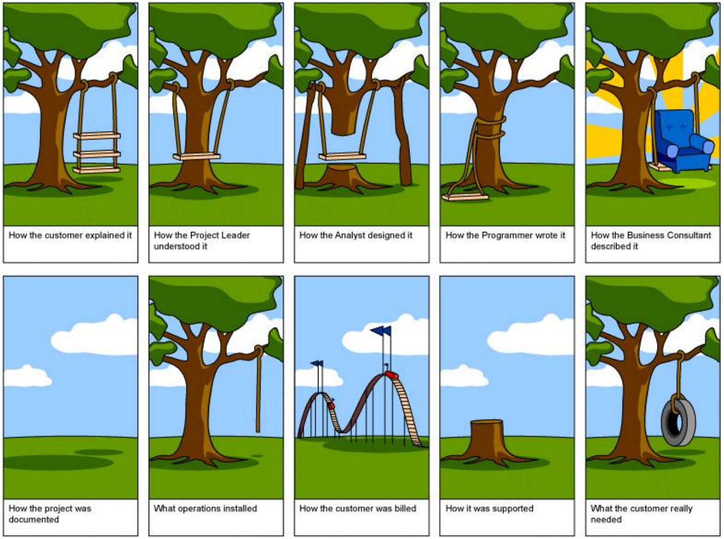
Bugs are everywhere

EECS 4315

www.eecs.yorku.ca/course/4315/

What is verification?

"Have you made what you were trying to make?"


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Source: Paragon Innovations

What is Verification?

”Have you made what you were trying to make?”

”Does the code satisfy (all the properties of) its specification?”

How the Programmer wrote it

How the customer explained it

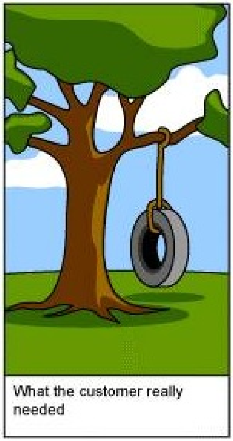
Source: Paragon Innovations

In contrast to . . .

"Have you made the right thing?"

Is the specification of the system correct?

which is also known as validation.



How the customer explained it

Source: Paragon Innovations

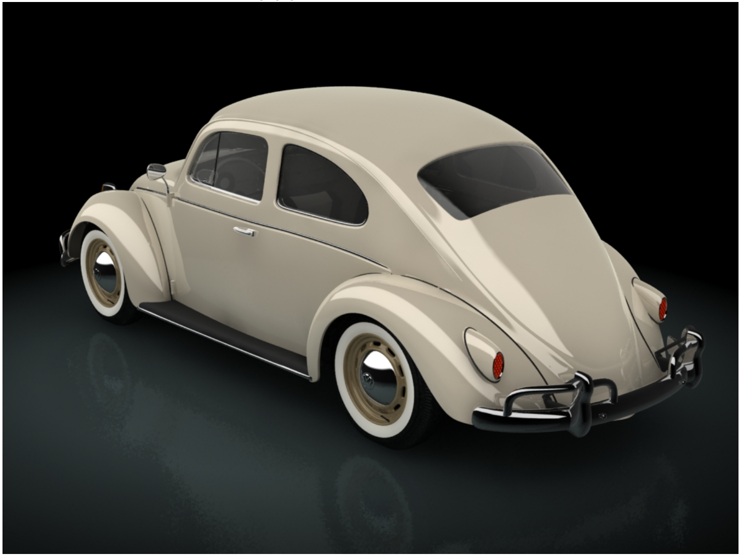
Why do we verify?

Bugs are everywhere.

Source: Bruce Campbell

Classic bug

1968 Brazilian Beetle



Source: Dan Palatnik

Classic bug

"A clear example of the risks of poor programming and verification techniques is the tragic story of the Therac-25 — one in a series of radiation therapy machines developed and sold over a number of years by Atomic Energy Canada Limited (AECL). As a direct result of inadequate programming techniques and verification techniques, at least six patients received massive radiation overdoses which caused great pain and suffering and from which three died."

Peter Roosen-Runge. Software Verification Tools.

Source: unknown

Classic bug

A computer malfunction at Bank of New York brought the Treasury bond market's deliveries and payments systems to a near standstill for almost 28 hours . . . it seems that the primary error occurred in a messaging system which buffered messages going in and out of the bank. The actual error was an overflow in a counter which was only 16 bits wide, instead of the usual 32. This caused a message database to become corrupted. The programmers and operators, working under tremendous pressure to solve the problem quickly, accidentally copied the corrupt copy of the database over the backup, instead of the other way around."

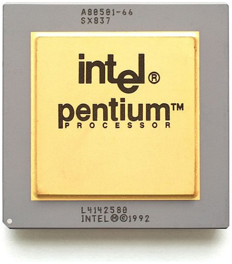
Wall Street Journal, November 25, 1985.

Source: unknown

Classic bug

”To correct an anomaly that caused inaccurate results on some high-precision calculations, Intel Corp. last week confirmed that it had updated the floating-point unit (FPU) in the Pentium microprocessor. The company said that the glitch was discovered midyear and was fixed with a mask change in recent silicon. “This was a very rare condition that happened once every 9 to 10 billion operand pairs” said Steve Smith, a Pentium engineering manager at Intel.”

EE Times, November 7, 1994.

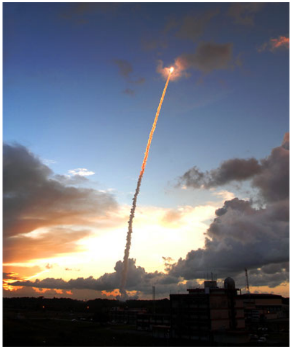


Source: Konstantin Lanzet

Classic bug

"On 4 June 1996, the maiden flight of the Ariane 5 launcher ended in a failure. Only about 40 seconds after initiation of the flight sequence, at an altitude of about 3700 meters, the launcher veered off its flight path, broke up and exploded. . . . The reason why the active SRI 2 did not send correct attitude data was that the unit had declared a failure due to a software exception. . . . . The data conversion instructions (in Ada code) were not protected from causing an operand error, although other conversions of comparable variables in the same place in the code were protected."

Report of the Ariane Inquiry Board



Source: unknown

Bug of the 21st century

2012 Beetle

Source: unknown

Bug of the 21st century

The Toronto skyline

Source: unknown

Bug of the 21st century

The Toronto skyline on August 14, 2003

Source: unknown

Bug of the 21st century

The first known death caused by a self-driving car was disclosed by Tesla Motors on Thursday, a development that is sure to cause consumers to second-guess the trust they put in the booming autonomous vehicle industry. . . . Against a bright spring sky, the car's sensors system failed to distinguish a large white 18-wheel truck and trailer crossing the highway, Tesla said. The car attempted to drive full speed under the trailer, with the bottom of the trailer impacting the windshield of the Model S, Tesla said."

Danny Yadron and Dan Tynan, The Guardian, July 1, 2016

Source: Daily Mail

Bug of the 21st century

" "A Model 787 airplane that has been powered continuously for 248 days can lose all alternating current electrical power due to the generator control units simultaneously going into failsafe mode," the FAA said in a statement warning of the flaw. . . . Most importantly, the company's already working on an update that will patch the software vulnerability."

engadget.com, 2015.

Source: engadget.com

Bug of the 21st century

"The Knight Capital Group announced on Thursday that it lost $440 million when it sold all the stocks it accidentally bought Wednesday morning because a computer glitch. . . . The company said the problems happened because of new trading software that had been installed. The event was the latest to draw attention to the potentially destabilizing effect of the computerized trading that has increasingly dominated the nation's stock markets."

Nathaniel Popper, The New York Times, August 2, 2012.

Source: Brendan McDermid

Bug of the 21st century

https://www.youtube.com/watch?v=FZ1st1Vw2kY

Why do we verify?

Ask Jessie J!

Source: unknown

It's all about the money money money.

What’s the price tag?

Bank of New York bug: $ 5 million
Pentium bug: $ 475 million
Ariane bug: $ 500 million
Blackout bug: $ 6 billion
Knight bug: $ 440 million

”The cost of software bugs to the U.S. economy is estimated at $ 60 billion per year.’

National Institute of Standards and Technology, 2002

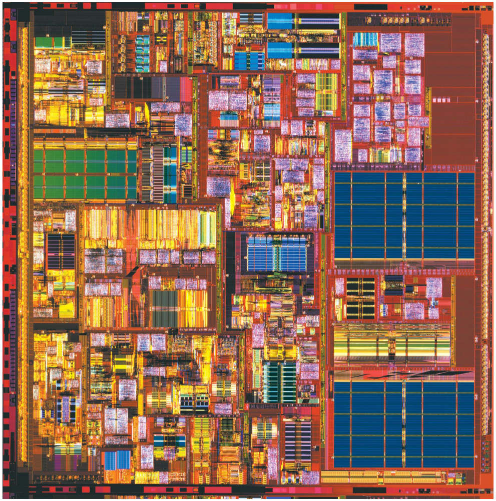
”Wages-only estimated cost of debugging: US $312 billion per year.”

Tom Britton, Lisa Jeng, Graham Carver, Paul Cheak and Tomer Katzenellenbogen, 2013

Why are bugs introduced?

Hardware and software systems are among the most complex artifacts ever produced by humans.

Pentium 4 microprocessor


Source: unknown

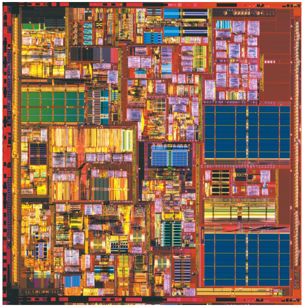
- transistors:
  55 million
- area:
  146 mm^2

If . . .

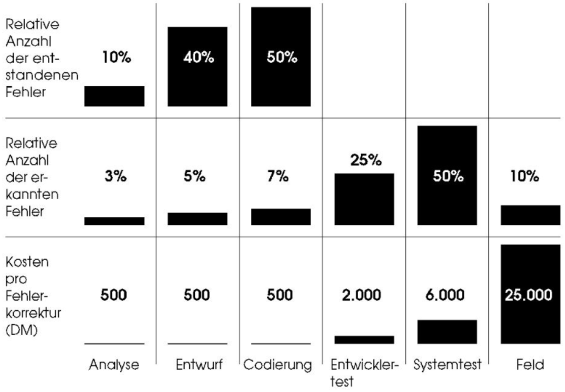
. . . the connections on a microprocessor were roads in the GTA, . . .

Area of microprocessor: 146 mm^2
Area of GTA: 7,124 km^2
Scale: 12 mm / 84 km $\approx$ 1 / 70,000,000

. . . then, since each connection is 0.13 μm wide, the roads in the GTA would be 3 feet wide, 3 feet apart and eight layers deep!

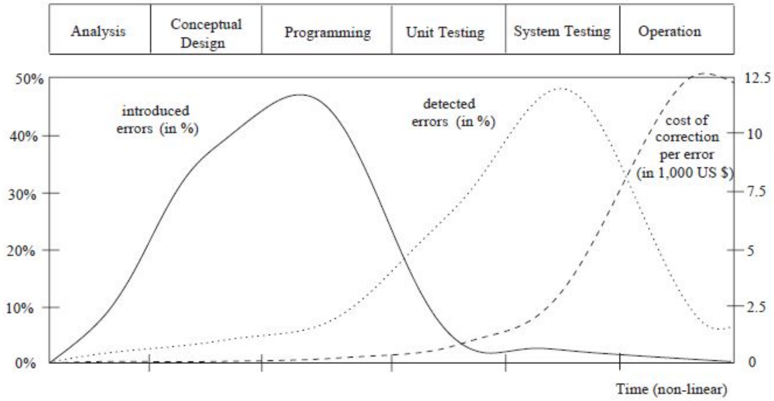


When are bugs introduced and detected?



Peter Liggesmeyer, Martin Rothfelder, Michael Rettelbach, and Thomas Ackermann. Qualitätssicherung Software-basiertertechnischer Systeme – Problembereiche und Lösungsansätze. *Informatik-Spektrum*, 21(5):249–258, October 1998

When are bugs introduced and detected?



Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press. 2008.

How are bugs detected?

- Peer review
- Simulation
- Testing
- Verification

Limitations of peer review

- Catches on average only 60% of the bugs.
- Is labour intensive (250 lines per hour).

Limitations of simulation

How long does it take to simulate a 128-bit multiplier on a 3 GHz machine?

Limitations of simulation

How long does it take to simulate a 128-bit multiplier on a 3 GHz machine?

1. How many cases need to be checked?

Limitations of simulation

How long does it take to simulate a 128-bit multiplier on a 3 GHz machine?

1. How many cases need to be checked?
   $2^{128} \times 2^{128} = 2^{256} \approx 1.2 \times 10^{77}$

Limitations of simulation

How long does it take to simulate a 128-bit multiplier on a 3 GHz machine?

1. How many cases need to be checked?
   $2^{128} \times 2^{128} = 2^{256} \approx 1.2 \times 10^{77}$
2. How many can we check in one second?

Limitations of simulation

How long does it take to simulate a 128-bit multiplier on a 3 GHz machine?

1. How many cases need to be checked?
   $2^{128} \times 2^{128} = 2^{256} \approx 1.2 \times 10^{77}$
2. How many can we check in one second?
   3×10^{9}

Limitations of simulation

How long does it take to simulate a 128-bit multiplier on a 3 GHz machine?

1. How many cases need to be checked?
   $2^{128} \times 2^{128} = 2^{256} \approx 1.2 \times 10^{77}$
2. How many can we check in one second?
   3×10^{9}
3. How many seconds does it take?

Limitations of simulation

How long does it take to simulate a 128-bit multiplier on a 3 GHz machine?

1. How many cases need to be checked?
   $2^{128} \times 2^{128} = 2^{256} \approx 1.2 \times 10^{77}$
2. How many can we check in one second?
   3×10^9
3. How many seconds does it take?
   $1.2 \times 10^{77} / 3 \times 10^9 = 4 \times 10^{67}$

Limitations of simulation

How long does it take to simulate a 128-bit multiplier on a 3 GHz machine?

1. How many cases need to be checked?
   $2^{128} \times 2^{128} = 2^{256} \approx 1.2 \times 10^{77}$
2. How many can we check in one second?
   3×10^{9}
3. How many seconds does it take?
   $1.2 \times 10^{77} / 3 \times 10^{9} = 4 \times 10^{67}$
4. How many years is that?

Limitations of simulation

How long does it take to simulate a 128-bit multiplier on a 3 GHz machine?

1. How many cases need to be checked?
   $2^{128} \times 2^{128} = 2^{256} \approx 1.2 \times 10^{77}$
2. How many can we check in one second?
   3×10^{9}
3. How many seconds does it take?
   $1.2 \times 10^{77} / 3 \times 10^{9} = 4 \times 10^{67}$
4. How many years is that?
   2×10^{59}

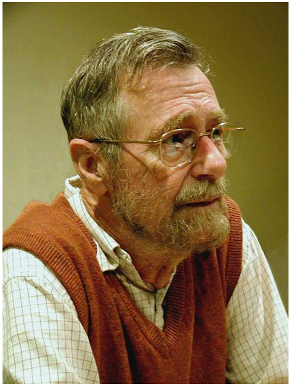
Limitations of testing

“Program testing can be used to show the presence of bugs, but never to show their absence!”

Edsger W. Dijkstra. Notes on structured programming. Report 70-WSK-03, Technological University Eindhoven, April 1970.

Edsger Wybe Dijkstra (1930–2002)

- Member of the Royal Netherlands Academy of Arts and Sciences (1971)
- Distinguished Fellow of the British Computer Society (1971)
- Recipient of the Turing Award (1972)
- Foreign Honorary Member of the American Academy of Arts and Sciences (1975)
- My scientific uncle (the supervisor of my supervisor was also Dijkstra's supervisor)


Source: Hamilton Richard