

SFWRENG 3S03: Software Testing

Introduction to the Course

Who Am I?

- Instructor: Dr Richard Paige
- Office: ITB 159A (allegedly)
- Office Hours: by appointment (usually easy to find on Wednesdays and Fridays)
- Email: paigeri@mcmaster.ca
- Twitter: @richpaige
- My background in software testing...

Course Organization

- 3 lectures per week
- Tutorials starting week of 17 January, once every other week
- Evaluation:
 - 3 assignments worth 10% each
 - 50 minute midterm worth 20% (Avenue quiz)
 - Final exam worth 50% (Avenue quiz)
- All course materials on Avenue/Teams.
- Online until “no earlier than 7 February”.

Assignments

- First assignment uploaded to Avenue end of this week/early next week.
- Due dates around:
 - Assignment 1: February 11
 - Assignment 2: March 11
 - Assignment 3: April 8
- Short exercises and practical applications of material taught in lectures.
- Bonus scheme.

Course content

- A pragmatic introduction to software testing and measurement.
 - Minimal mathematical theory (that's for more advanced courses)
 - Emphasis on engineering practice
 - Emphasis on techniques that are used in industry
 - Plenty of anecdotes about the necessity and inadequacy of testing
 - Hopefully some insights about how testing complements other forms of verification.

Some topics

- Types of testing
- Exploratory, partition, random, functional testing
- Coverage measures
- Inspections
- Mutation testing
- Static analysis
- Unit testing/TDD
- Continuous integration
- Test planning and test strategies
- Test practices at FAANG etc
- Mobile application and security testing

Software Testing

Why You Shouldn't Test Your Code Too Well

Question: How much testing do you do?

```
import java.util.Scanner;

public final class HelloName{
public static void main(String[] args) {
    //declare scanner object
    Scanner sc = new Scanner(System.in);

    //prompt user
    System.out.print("Please enter your name:\t");

    //get input and print string
    System.out.println("Hello " + sc.nextLong() + "!");
}
}
```

What Could We Do?

- Compiler checks syntax
- Lint4j checks common errors
- Hire experienced test consultant to do (manual) exploratory testing
- Run with exhaustive input strings up to e.g. 128 chars e.g. “a” through to “~øfflΓR....”
- Boundary cases e.g. empty string, max length string
- Special inputs e.g. ctrl-C, ctrl-D, ctrl-Z
- Measure code coverage achieved by the above using EMMA or Cobertura

... even scarier ...

- 6-person multidisciplinary team allocated two days to do Fagan review
- Use Java Pathfinder, which exhaustively explores state space of possible executions
 - Hire NASA software engineers to get you up to speed
- Find an academic collaborator and apply for funds to develop a general theory of Hello World correctness
-

Please enter your name: Richard

Exception in thread "main"

java.util.InputMismatchException

at java.util.Scanner.throwFor(Unknown Source)

at java.util.Scanner.next(Unknown Source)

at java.util.Scanner.nextLong(Unknown Source)

at java.util.Scanner.nextLong(Unknown Source)

at HelloName.main(HelloName.java:12)

Clearly, “no testing at all” is bad

- ... but you can carry it to absurd extremes

The Message

“You should test software well
*given your context and
purposes*”

Exciting exercise – chat

- **“What do I most want to find out about in this course?”**
- Take about 60 seconds
- Post your thoughts, either in the Team chat, or in a private Team message to me.
 - I’ll collect them and mention some of them next time.

SECTION 1 – WHAT IS SOFTWARE TESTING, REALLY?

An obvious definition

Testing (1)

Testing consists of executing a program on some inputs in order to find errors

Problems with definition (1)

- Leaves questions unanswered
 - How do we choose the inputs?
 - How do we recognise errors?
- Doesn't cover all testing
 - Not all testing is at the program level
 - Possible to test the code without executing it
 - Can test for reasons other than fault discovery

A better definition

Testing (2)

Testing consists of evaluating software in order to derive an estimate of whether or not it meets some criteria

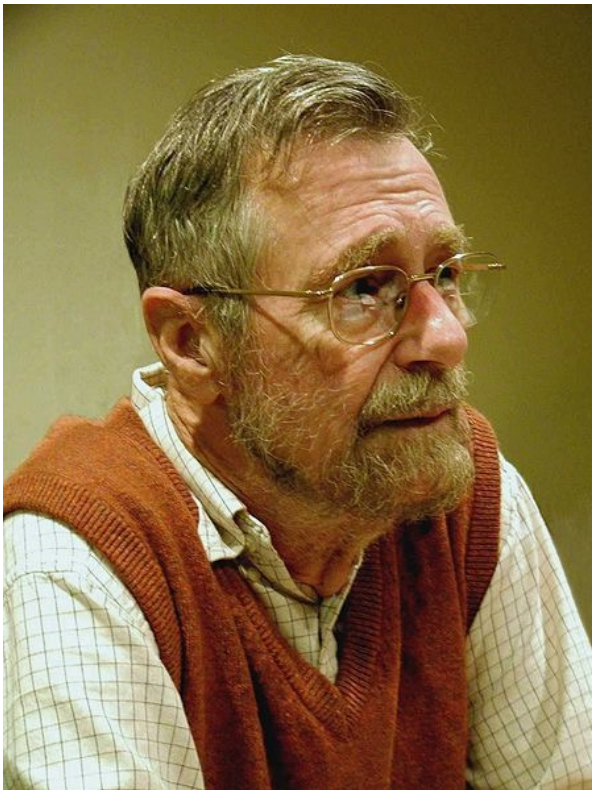
Still some questions unanswered

- What is evaluated?
- When is it evaluated?
- How is it evaluated?
- What criteria must it meet?
- How do we know if it meets the criteria?

The most famous quote on testing...

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

Edsger W. Dijkstra



[http://en.wikiquote.org/wiki/Edsger W. Dijkstra](http://en.wikiquote.org/wiki/Edsger_W._Dijkstra)

Reference:

Notes On Structured Programming, 1972,
at the end of section 3,
On The Reliability of Mechanisms.

What *can* testing do, then?

- Testing can show software works
 - i.e. that it sometimes does what we want
- Testing can show software doesn't work
 - i.e. that it sometimes does something bad
- Testing can reduce the chance that software doesn't work
 - i.e. it can build *confidence* that key properties hold

- Any questions?

SECTION 2 – ANALYZING SOFTWARE WELL CAN BRING LARGE BENEFITS

Do you have to test well?

- No
- Sometimes you're lucky
- Sometimes you can recover

Ariane 5



Ariane 5

- SRI = Inertial Reference System
- SRI needs to be calibrated before liftoff (up to T-9 seconds)
- Sometimes, launch is delayed, and SRI is needed after that
- Restarting SRI could take hours... so they let it keep running (until T+50)
 - i.e. it's still running 50s into the flight

Ariane 5

- Periodically, “horizontal bias” value is measured as 64-bit float and converted to 16 bit signed int
- Ariane 4, this is fine – bias value always fits in that int (at least during first 50 seconds)
- Ariane 5, trajectory is different
 - Value is measured that won’t fit in that int
 - Component throws exception
 - Nothing in place to catch that exception
 - SRI crashes
 - Ariane 5 computers all crash

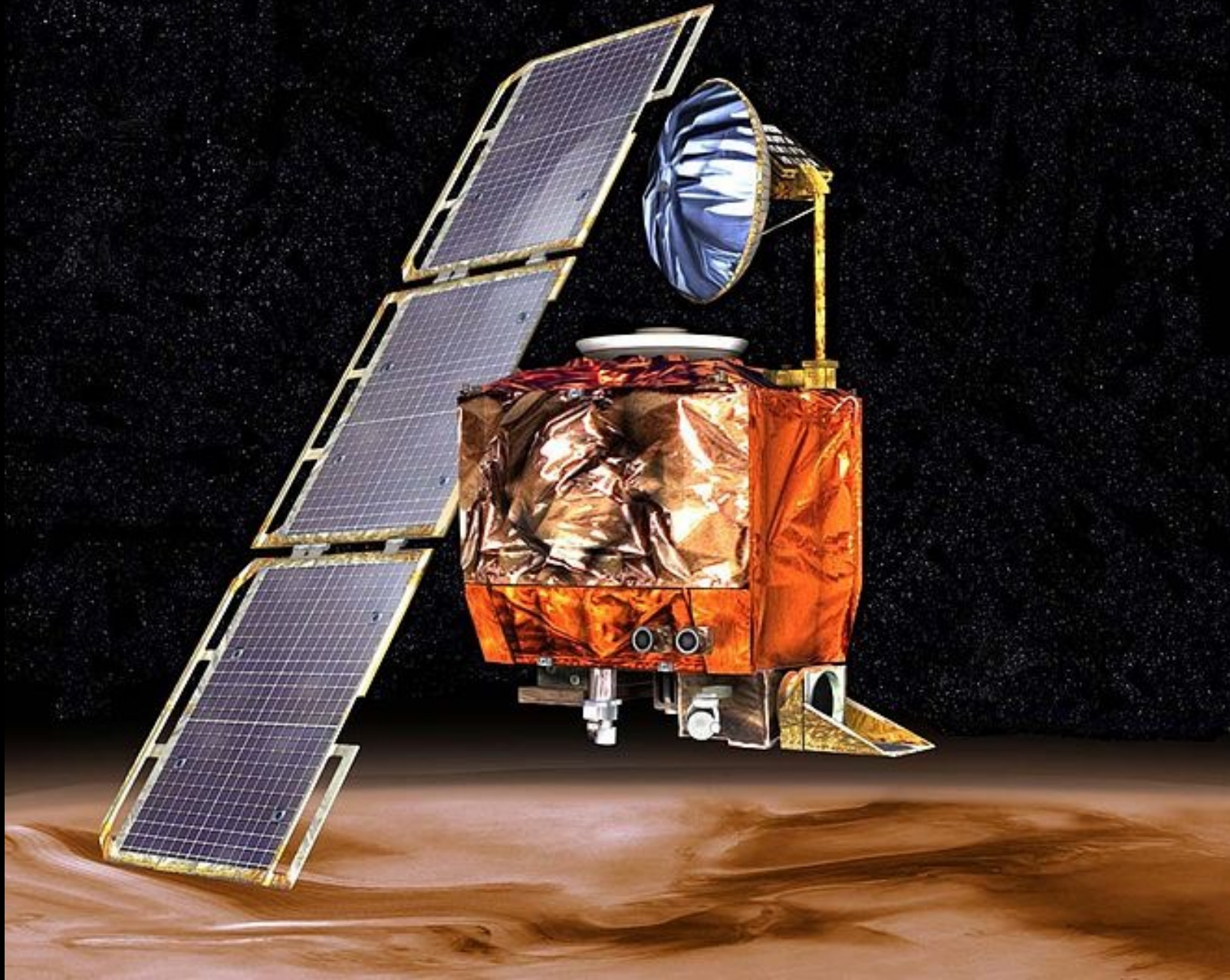


<http://www.youtube.com/watch?v=kYUrqdUyEpl>

Ariane 5

- Source for this account -
<http://dx.doi.org/10.1109/2.562936>
- Inquiry Board report -
<http://www.di.unito.it/~damiani/ariane5rep.html>

Mars Climate Orbiter



Mars Climate Orbiter

- Two teams, two components:
 - SM_FORCES (“Small Forces”) program
 - Produces “Angular Momentum Desaturation” data file (AMD)
 - Works in Imperial/English units (lb/sec)
 - Trajectory modelling team
 - Uses AMD file
 - Works in metric units (Newton/sec)
- Calculations are out by a factor of 4.45
 - (that’s newtons-per-pound)
- Orbiter flew too low and was destroyed
- Mishap Investigation Board report -
ftp://ftp.hq.nasa.gov/pub/pao/reports/1999/MCO_report.pdf

Mars Polar Lander



Mars Polar Lander

- Lander enters the Martian atmosphere
- Deploys parachute, fires slowdown rockets
- Deploys legs
 - Sensors in legs send spurious signals, consistent with weight on legs after landing
 - Control software should have been designed to ignore these signals while legs were deploying
 - It wasn't
- Control software shuts down rockets (too early)
- Lander hits ground too hard, and is wrecked

Mars Polar Lander

- More
 - <http://news.bbc.co.uk/1/hi/sci/tech/462264.stm>
 - <http://partners.nytimes.com/library/national/science/032900sci-nasa-mars.html>
 - <http://www.spaceref.com/news/viewnews.html?id=105>

Common themes from those examples

- Testing was inadequate
 - They could have found the fault before launch, but they didn't
- Failures can be emergent
 - Perfectly valid components, when put together, can lead to bad things
- Space travel is hard...
- ...but so is telecoms, enterprise systems, keeping a Tesla running over patches and more ...

- Any questions?

SECTION 3 – TYPES OF TESTING

Software can, clearly, go wrong

- So let's test it
- Complication – not every kind of testing can reveal every kind of fault

Definition

Dynamic Testing

Testing code by executing it.

Definition

Static Analysis

Testing code without executing it.

Examples – Static/Dynamic

- **Come up** with an example of each type of testing for...
- 1) Apache web-server
- 2) a smartphone app
- 3) the code of a satellite

Some possible answers

- Apache
 - **Dynamic:** make a request generator that hammers a standard web site
 - **Static:** extract an arch model and calculate load on components under various circumstances – where is the bottleneck?
- Phone app
 - **Dynamic:** give to target users to try out – monitor their confusion and the app's crashes
 - **Static:** use tools to search code for patterns commonly associated with security vulnerabilities (e.g. buffer overflows)
- Satellite
 - **Dynamic:** put whole code in simulated environment, and try with expected operating cycles (at least one orbit, but multiple is better);
 - **Static:** rigorous structured inspection of whole code base – after all, you can't retrieve satellite once it's up there

So far, I've assumed visibility of code...

Definition

Grey-Box Testing

Testing that exploits knowledge of the system to design tests for it.

Sometimes, you won't have source code

- E.g. closed-source COTS product
- E.g. you're building tests for many possible implementations and you don't have sight of them all
- E.g. you have an API but not implementations
- E.g. you are working in Matlab Simulink and what you have is a model
-

Definition

Black-Box Testing

Testing without any knowledge of the software's implementation.

If you *do* have code visibility, you can systematically exploit that

Definition

White-Box Testing

Testing that *systematically* exploits knowledge of the system to design tests for it.

Examples – Black/Grey/White -box

- **Come up** with an example of each type for testing...
- 1) Apache web server
- 2) a smartphone app
- 3) the code of a satellite

Some possible answers

- Apache
 - **B**: external request-generator (perhaps based on data we've collected about what kinds of requests are typically received)
 - **G**: request-generator designed to target things we know are expensive or think might be bottlenecks
 - **W**: run request-generator with a coverage tool which tells you which parts of the code are actually being run
- Phone app
 - **B**: users test through user interface how they'd normally use the phone
 - **G**: users directed to focus on new features, or features with many fault reports against them
 - **W**: static analysis for security (as earlier slide)

Some possible answers (2)

- Satellite
 - **B**: simulation testing, where dev team know nothing about the specific code of the satellite, only its expected missions and environment
 - **G**: simulation testing, team have access to the code and software designs, though they don't necessarily use it
 - **W**: simulation testing with code coverage assessment

Does feature X work?

- E.g.
 - “Can we print the current document?”
 - “Does `calculate(“2 * 2 + 1”)` return the correct answer?”
 - “If we start it with a bootable USB stick plugged in, does it boot from that?”
 - Does our app allow login after we’ve switched from AWS to “our own bare metal infrastructure, LOL”.

Definition

Functional Testing

Testing that checks whether the program provides some specific functionality.

Q - What else might we care about?

- Performance when the database is large
- Error rate when identifying faces in photos
- Security against hacking attempts over the network
- ...

Definition

Non-Functional Testing

Testing that assesses the program on something other than whether it provides some function.

Examples – Functional/Non-Functional

- **Come up** with an example of each type for testing...
- 1) Apache web server
- 2) a smartphone app
- 3) the code of a satellite

Some possible answers

- Apache
 - **F** --- does it implement all HTTP 2.0 or HTTP 3.0 requirements?
 - **NF** --- how many typical pages can it serve per second on typical hardware?
- App
 - **F** --- does our app link to Facebook?
 - **NF** --- are there any ways that our app could print personal data (e.g. address, text message list) to Facebook when the user doesn't want it to?
- Satellite
 - **F** --- can the satellite code save multispectral (infrared + visible light + UV) images as single false-color pictures?
 - **NF** --- can the satellite manage power so as to provide full imaging functionality for the longest expected time it will spend in shadow of the earth?

Fault, Error, Failure

- Fault
 - A stable property of the program that can cause a failure to occur
 - E.g. if the program is started with a configuration file that doesn't specify the main data file, it sets the data file name to null, and carries on
- Error
 - A bad state of a *running* program that can lead to a failure
 - E.g. the string containing the program's data file name is Null
- Failure
 - The software does something observably bad
 - E.g. crashes with NullPointerException
- Source – Avizienis et al at <http://dx.doi.org/10.1109/TDSC.2004.2>
- Amman & Offutt use these definitions.

Definitions List

- Testing
- Static / dynamic
- Black / white / grey -box
- Functional / non-functional
- Fault, Error and Failure

- Any questions?

SECTION 4 – WHAT AND WHEN SHOULD YOU TEST?

So, you've got a software system, and you want to test it

- But where do you start?
- And what artefacts do you test?
- Post thoughts in the chat!
- Make it concrete if you like: suppose you were asked to test Mosaic, or Parler, or the control software for a 737Max!

One option is to start with the smallest meaningful unit

Method:

```
int increment(int i){  
    return i+1;  
}
```



Test

Test:

```
@Test  
public void test_1(){  
    int j = 0;  
    assertEquals(1, increment(j));  
}
```

This is a **unit test**

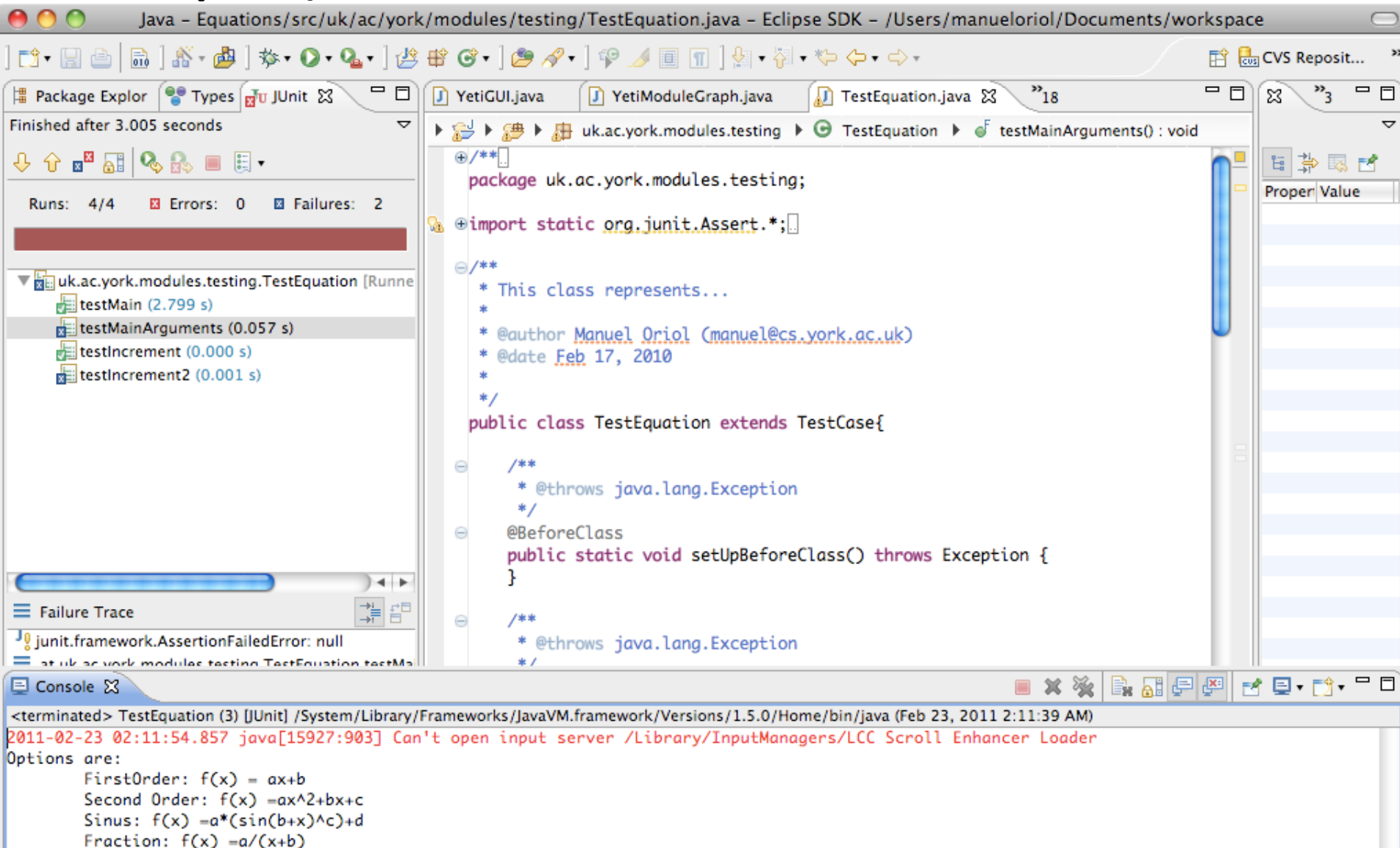
Strengths of unit testing

- Can locate a low-level fault very precisely
 - Typically to a single method
- Can easily be conducted by the programmer who's working on that code right now
 - I.e. the person who right now is the best in the world at understanding that code
 - They're the best-placed person to debug a failing test

What is the right “unit”?

- “Unit” definition can vary by team, project, language...
- In Java, typically method and class level
- But what’s the right “unit” for a Matlab Simulink program, or a TLA+ specification, or even a C++ program (not a purely OO language)?

Unit testing tool: JUnit (run from Eclipse)



Java - Equations/src/uk/ac/york/modules/testing/TestEquation.java - Eclipse SDK - /Users/manueloriol/Documents/workspace

Package Explorer | Types | JUnit

Finished after 3.005 seconds

Runs: 4/4 | Errors: 0 | Failures: 2

uk.ac.york.modules.testing.TestEquation [Runne]

- testMain (2.799 s)
- testMainArguments (0.057 s)
- testIncrement (0.000 s)
- testIncrement2 (0.001 s)

Failure Trace

junit.framework.AssertionFailedError: null

at uk.ac.york.modules.testing.TestEquation.testMain

```
package uk.ac.york.modules.testing;

import static org.junit.Assert.*;

/**
 * This class represents...
 *
 * @author Manuel Oriol (manuel@cs.york.ac.uk)
 * @date Feb 17, 2010
 */
public class TestEquation extends TestCase{

    /**
     * @throws java.lang.Exception
     */
    @BeforeClass
    public static void setUpBeforeClass() throws Exception {

    /**
     * @throws java.lang.Exception
     */
}
```

<terminated> TestEquation (3) [JUnit] /System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/Home/bin/java (Feb 23, 2011 2:11:39 AM)

2011-02-23 02:11:54.857 java[15927:903] Can't open input server /Library/InputManagers/LCC Scroll Enhancer Loader

Options are:

- FirstOrder: $f(x) = ax+b$
- Second Order: $f(x) = ax^2+bx+c$
- Sinus: $f(x) = a*(\sin(b*x)^c)+d$
- Fraction: $f(x) = a/(x+b)$

Unit tests can't find all problems

- Many faults don't manifest at the unit level at all
 - Indeed, often don't *exist* at that level, definitively
 - they arise from interactions between units
 - E.g. remember the Mars Climate Orbiter with its mismatched measurements?
 - Or the Ariane 5 with its unexpected exception?
 - Or <http://www.instagram.com/p/BQQasrzAt41>

Solution – we can test the whole system

- This is **system testing**
- Mars Climate Orbiter — A system test, in simulation, could have revealed that its trajectory made no sense
- Likewise Ariane 5 — any whole-system launch simulation would likely have revealed fault
- Typically black-box testing
 - Structural metrics and instrumentation etc much less mature on this scale

Strengths of System Testing

- Can potentially find any fault wrt your design intent
- Can test functional and non-functional requirements
 - Most non-func requirements are emergent properties of components
 - can't pin down to unit level at all
 - E.g. performance – can run with application server, local database, remote web services, client network
 - ... then hammer it with simulated clients and measure overall throughput
- Can help you understand non-explicit requirements
 - Some system behaviour will come to seem obviously wrong
- Can often farm out to a specialist team who work only at this level

Some parts-interacting faults are hard to find from whole-system level

- E.g. imagine if Ariane 5 case only happened for a narrow subset of trajectories
- You *might* find that in system testing, but you might not
 - ... unless you spend an enormous effort on it

Testing interacting classes, modules or subsystems

- This is **integration testing**
- Consider the Climate Orbiter again
 - SM_FORCES (“Small Forces”) program in Imperial/English units (lb/sec)
 - Trajectory modelling team working in metric units (Newton/sec)

Could have tested some resulting calculations against answers determined independently

Integration Testing

- Typically, grouping together some units and testing them together using a black-box approach
- Three main approaches:
 - Big Bang: Put everything together then test
 - Top-down: Modules tested from the entry points and integrated progressively
 - Bottom-up: Modules are progressively integrated and tested from the most elementary ones.

Strengths of integration testing

- Can test interactions that unit tests can't
- Much more tractable than system test
- Can be performed by engineers who are quite close to the components being integrated

So, say we've got some pretty good tests

- Unit, integration and system all covered
- Pretty confident now of our system's quality
- Why should the customer trust us on that quality claim? How could we convince them?

Well, we could show them our test sets

- But our test sets are huge
- They're probably expressed in software-engineer-centric terms
 - Not necessarily domain or end-user terms
- We may only have done *verification*
 - May be consistent with written spec
 - May have built what we thought we were asked to build
 - ...but is it what the customer actually wants?

Solution – work with customer to build some tests meaningful to them

- This is **acceptance testing**
- Designed by domain experts (subject matter expert)
- Embodied in a paper script (probably supplemented by automated scripts)
- Performed by potential users
- Main intent is not to discover failing scenarios, it is to check that the product will work (and how well) in a production environment

Strengths of Acceptance Testing

- Very much a validation activity — did you build the right thing?
- Can help build trust relationship with customer
- Can help you understand customer, domain and market better

So, we've passed the acceptance tests...

- ... and we put out a first release of the system
- What do we do with all those tests we made?

We can keep our tests for re-use in the future

- (the automated ones, at least)
- They become a *regression test suite*, which we can use for **Regression Testing**
 - The goal is to check that what used to work still does
 - Can apply at any scope (unit, integ, sys ...)

Example (1/2)

```
//Version 0  
int increment(int i){  
    return i+1;  
}
```



Test

```
@Test  
public void test_1(){  
    int j = 0;  
    assertEquals(1, increment(j));  
}
```

Example (2/2)

```
//Version 1
int increment(int i) throws Exception{
    if (i<Integer.MAX_VALUE)
        return i+1;
    else
        throw new ArithmeticException();
}
```



Test

```
@Test
public void test_1(){
    int j = 0;
    assertEquals(1, increment(j));
}
```

- Any questions?

SECTION 5 – CHOOSING HOW TO TEST

Q – We've got lots techniques and approaches and types of testing. How do we know what to do?

- Read
 - Books, articles, blogs
- Talk
 - To testers, programmers, end users
- Practice
 - Testing, programming, using faulty software
- Watch
 - Others testing/failing to test

Testing is a Skill (-set)

- (Good) testing is hard
- (Good) testing isn't a mechanical process
- That's why there are "Test Engineers" and "Test Managers"
 - ... and people paid to talk about testing
 - Salary ranges \$50-\$100K for test engineer, could be more for EE/embedded testing with experience
 - Test manager likely more.
 - Test consultant, architect, automation engineer all similar.

Constraints on Your Choices

- ...are legion. One example:
- Regulated industries
 - Avionics software
 - Standard DO-178B
 - Must achieve 100% MC/DC test coverage

- Any questions?

SECTION 6 – WHY SHOULD I CARE ABOUT TESTING?

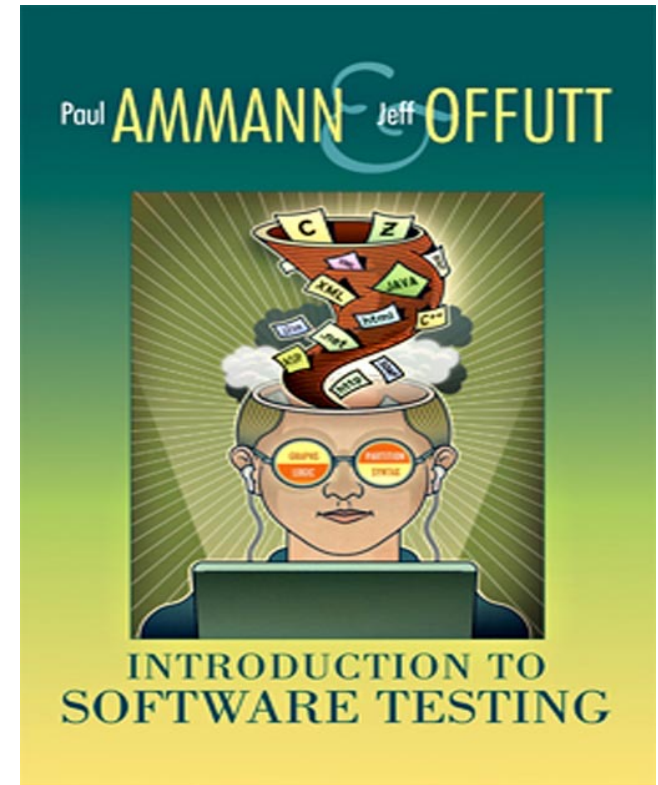
“I’m not planning to be a test engineer...”

- You might need to work with them
- You might end up in a small team or company
-

SECTION 7 – WHERE CAN I GET MORE INFORMATION?

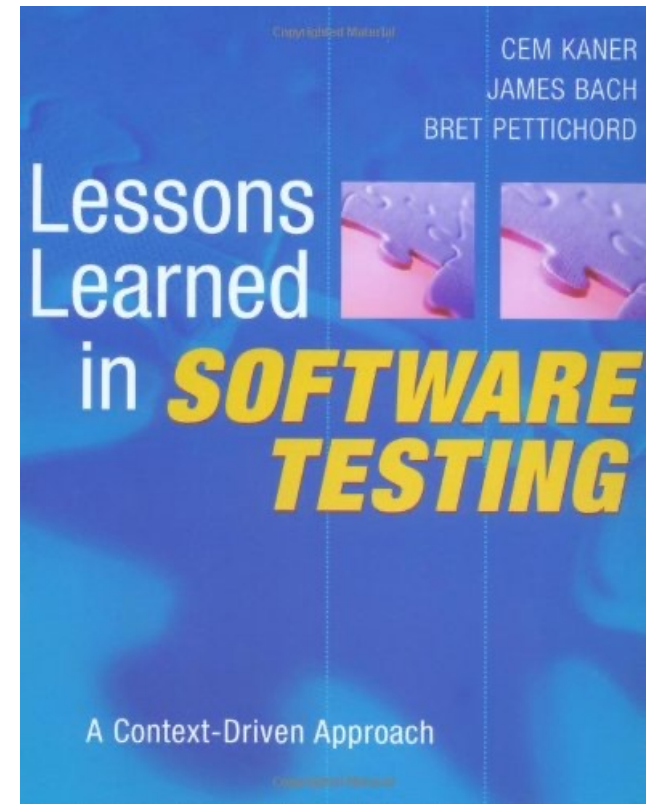
Textbook 1

- *Introduction to Software Testing*
Paul Ammann and Jeff Offutt
 - Precise, thorough, academic
 - Systematic testing techniques
 - Coverage criteria



Textbook 2

- *Lessons Learned in Software Testing: A Context Driven Approach*
Cem Kaner, James Bach and Bret Pettichord
 - Very practical discussion of testing in the real world.



Exciting Exercise

- **“What was the main point I learned in this set of lectures?”**
- **“What was the main point left unanswered in this set of lectures?”**
- Please share on the Team chat or in a private Team message to me.
- I’ll collect and share some of them out next time.

- Any questions?

Now

By email

(I'll answer by email, or in next
lecture)