About me...
Introduction

- Why do we test?

- Did you have to deal with testing in the past?
Ariane 5

http://www.youtube.com/watch?v=kYUrgdUyEpI
The exception was due to a floating-point error during a conversion from a 64-bit floating-point value, representing the flight’s “horizontal bias,” to a 16-bit signed integer: In other words, the value that was converted was greater than what can be represented as a 16-bit signed integer. There was no explicit exception handler to catch the exception, so it followed the usual fate of uncaught exceptions and crashed the entire software, hence the onboard computers, hence the mission.
We have been trained to make assumptions

\[ x \times x = x^2 \geq 0 \]  
false for \( x = 46341 \)  
(and many more int)

\[ x + 1 > x \]  
false for \( x = \text{MAX\_INT} \)

\[ (x \times y) / x = y \]  
false for \( x = 0 \) or float \( x \)

\[ (y / x) \times x = y \]  
false for \( x = 0 \) or int, float \( x \)
Typically impossible to...

- Test all values (see model-checking)
- Know what to omit when testing
- Know how to interpret results
An example

/*
 * A simple method that increments an integer value
 ***/

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

• Testing all values?
• What not to test?
• How to interpret results?
In this case...

- Test all values? It is possible!

- Know what to omit when testing? e.g. [http://en.wikipedia.org/wiki/Pentium_FDIV_bug](http://en.wikipedia.org/wiki/Pentium_FDIV_bug)

- Know how to interpret results? increment(Integer.MAX_VALUE) ??
The famous quote on testing...

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

Edsger W. Dijkstra


Referencing:
Notes On Structured Programming, 1972, at the end of section 3,
On The Reliability of Mechanisms.
The usual trade-off

Quality & Test

Deadlines & Reputation
Natural tendencies

- Testing is in the way to make deadlines
- Testing finds bugs that do not matter
- I have no time planned for the testing
- “Come on, our code is good!”
- “The code I write is throw-away”
So why do we really test?

We try to find bugs...

... to fix them ...

... to improve the quality of the code!
Testing saves time and finds bugs early

<table>
<thead>
<tr>
<th>With system-level testing</th>
<th>With system-level testing and unit testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>70 bugs</td>
<td>1 bug</td>
</tr>
<tr>
<td>16 weeks debugging time</td>
<td>50% less overall time</td>
</tr>
<tr>
<td></td>
<td>5%-30% of the time writing tests</td>
</tr>
<tr>
<td></td>
<td>5%-20% running tests</td>
</tr>
</tbody>
</table>

Gail C. Murphy, Paul Townsend, and Pok Sze Wong. 1994. Experiences with cluster and class testing. *Commun. ACM* 37, 9, 39-47
So, should we just test, test, test?

- This would not solve the problem if testing is not planned and strategically applied!

- Testing techniques are numerous and give a very large panel of possibilities

- A software test engineer (or software tester) will know how to apply most and be able to discover/adapt them to the software at hand.
What makes a good tester?

- The will to spend time crashing programs
- A strong commitment to drive the code to the best level of compliance with specifications
- The will to drive quality of the code up
- The will to understand how a program works to find its limitations
- The will to use tools and techniques that test programs
Some conventions

- **IEEE terminology:**
  - When a program exhibits an unexpected behaviour, it is a FAILURE
  - A failure is caused by a FAULT in the program
  - A defect is caused by an ERROR or a MISTAKE made by a programmer

Source: IEEE standard 610.1
Outline

- Types of testing
- Testing scopes
- Testing Processes
- Testing Artifacts
- Testing Metrics
Part I: Types of Testing
Categories of Testing

- Black-box/white-box/grey-box
- Static/dynamic testing
- Functional/non-functional
Black-box/White-Box/Grey-Box

- **Black-Box testing**: does not consider implementation details, only interfaces.

- **White-Box testing** (glass-box, clear-box, transparent, structural): uses the actual implementation of the program to devise tests.

- **Grey-Box**: Mixes both of them... If the test engineer know some of the internal of the program, it uses those to design some of the tests, the rest uses black-box.
Dynamic/Static Testing

- Dynamic testing is when the environment executes code, for example:
  - Automated testing
  - Unit tests

- Static testing does not require to execute the program, for example:
  - Walkthrough
  - Reviews
  - Inspections
  - Static analysis
Example of static Analysis tool: findbugs

http://findbugs.sourceforge.net
Other examples: Structure 101, Understand, Klocwork
Functional/non-functional Testing

- **Functional testing** tests that the program provides a functionality (e.g. calculates a result, doing something...)

- **Non-Functional testing** tests non-functional properties (scalability, security, “-ilities” in general)
Examples

- Stress-testing the Apache web-server
- Testing code that has been outsourced
- Testing the code of a satellite
- Testing the code running a cell phone
- Testing Microsoft Word
Part II: Testing Scope
Testing Scopes

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing
- Regression Testing
Unit Testing

- Testing small parts of the programs
- Typically the unit tests have an initialisation part and an assertion for testing the value that should be returned

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

Test:
```java
@Test
public void test_1(){
    int j = 0;
    assertTrue(increment(j)==1);
}
```
Integration Testing

- Typically grouping together all 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.
System Testing

- Tests integrated systems
- Tests functional and non-functional requirements
- Trying to understand even expected non-explicit requirements
- Typically black-box testing
Acceptance Testing

- Runs based on script

- Designed by domain experts (subject matter expert), 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
Example of acceptance testing: A/B Testing

- To compare two alternatives of a product and decide on the one to pick using a metric of success
- 50% of the traffic is version A and 50% on version B.
- Example:

  ![Diagram](https://vwo.com/ab-testing/)
Regression Testing

- The goal is to check that what used to work still does

- For example, test suites will be automatically executed to check that scenarios are working

- The scope itself can vary
Example (1/2)

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

@Test
public void test_1(){
    int j = 0;
    assertTrue(increment(j)==1);
}
Example (2/2)

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

@Test
public void test_1()
{
  int j = 0;
  assertTrue(increment(j)==1);
}
Regression Testing Tool: Junit (from Eclipse)
Part III: Testing Processes
Original waterfall model

Winston Royce, 1970, source: wikipedia, waterfall model
V-Model

Source: V-model, wikipedia
Test-Driven Development

K. Beck (2003),
Source: Test-driven_development, wikipedia
Extreme Programming

http://www.extremeprogramming.org/introduction.html
Integration testing or how to “trust but verify”

- Tests are going to be run on each commit (preferred) or nightly and reported to users
Artifacts

- Requirements Analysis
  - Requirements

- Test Planning
  - Test Strategy
  - Testbed
  - Test Plan/Procedures
  - Traceability Matrix

- Test development
  - Test procedures
  - Test cases
  - Test scenarios

- Test execution
  - Bug reports

- Test reporting
  - Test report

- Test Result Analysis
  - Faults prioritization
Part IV: Testing Artifacts
Test Case

- A test script that generally consists of a single step to test a program.
- Typically a test case will have a test oracle to decide whether is passes or fails
- Test cases generally include the following indications:
  - Id
  - Description
  - Related requirements
  - Category
  - Author
  - Status (pass/fail)
Example

- **Id:** test_1
- **Description:** a test to decide that checks “increment” with “0”
- **Related Requirement:** “increment” documentation
- **Category:** Functional, Unit
- **Author:** Manuel
- **Status:** Pass

```java
@Test
public void test_1(){
    int j = 0;
    assertTrue(increment(j)==1);
}
```
Test Oracle

- Typically a way of deciding whether a test case passes or fail

- Includes:
  - Documentation
  - Requirements
  - Assertions
  - Other means of calculating the result
Test Suite

- A test suite is a (potentially large) collection of test cases
- Typically test cases can be grouped in categories
- The goal of a test suite is to permit be used for checking that a new functionality does not break the code, or that it provides what is needed
- Large test suites might not be testable all the time (needed to test only a subset)
- Test suites quality is difficult to define (e.g. see mutation testing)
Test Data

- Values used during testing to test some functionality
- Typically stored in separate files
- Difficult to generate a good set of test data: it is often reused
Part V: Testing Metrics
Coverage

The coverage is a measure of a percentage of a structure or a domain that a program, a test case, a test suite exercises.
Coverages

- Function coverage
- Statement coverage
- Branch coverage
  (also known as: Decision coverage)
- Path coverage
- Condition coverage
- MCDC
Function Coverage

- The percentage of functions that were called by the test case

Typically function coverage should be 100%
Statement coverage

Percentage of statements that were executed
Example

Coverage of the red path:
86%
(6/7 statements)
Decision Coverage

- Each time a program has a branching instruction (if, for, while...) this creates two branches.

- Decision coverage is the percentage of these branches that were executed by a test suite.
Example

Decision Coverage of the red branches: 50%

```
a:=1
b:=a+2
If (b>3)
  Result:=b
  b:=a
else
  Result:=a
```
Branch Coverage

- Each time one has a branching instruction (if, for, while...) this create two branches.

- Branch coverage is the percentage of the branches that were executed by a test suite.
Example

Coverage of the red branches:
75%
(3/4 branches)

\begin{align*}
a & := 1 \\
b & := a + 2 \\
\text{If } (b > 3) \\
\text{true} \\
\text{false} \\
\text{Result} & := b \\
\text{Result} & := a \\
b & := a \\
b & := a
\end{align*}
Path Coverage

The percentage of different paths exercised by the tests (put in relation with cyclomatic complexity)
Example

Coverage of the red path:
50%
(1/2 paths)
Condition Coverage

- Each time one has a branching instruction (if, for, while...) that contains one or several conditions, each condition’s outcome (True or False) is a possibility.

- Condition coverage is the percentage of these possibilities that were executed by a test suite.

\[
\text{if (a || b)}
\]

100% obtained with (a,b) = (true, false) and (false,true).
Modified Condition/Decision Coverage

- Consists of:
  - 100% branch coverage
  - 100% condition coverage
  - Each entry/exit point is exercised
  - Each condition affects the behaviour independently

if (a || b)

DO-178B, Software Considerations in Airborne Systems and Equipment Certification
Tools to calculate the coverage: Cobertura

http://cobertura.sourceforge.net/
My advice…

- Write tests as a part of the coding activity
  - Not at the end, not at the beginning, rather per unit
- Write unit tests
  - Use unit testing frameworks like JUnit
    - Monitor decision coverage and try to get it close to 100%
- Write integration tests
  - use scripts and specific tools like Selenium
- Run your tests continuously
  - Use a continuous integration server like Jenkins
- Fix the bugs you find
Conclusions

- Software testing is at the core of any quality assurance mechanism currently used

- This presentation only gives a high level understanding of the techniques used in testing there is far more to learn
Some terms used in software testing
Power and productivity for a better world™