https://www.menti.com/2h1p9xc4ad

Code: **2825 7782**

Software Testing

Manuel Oriol, Prof of Computer Science

sit.org

Introduction

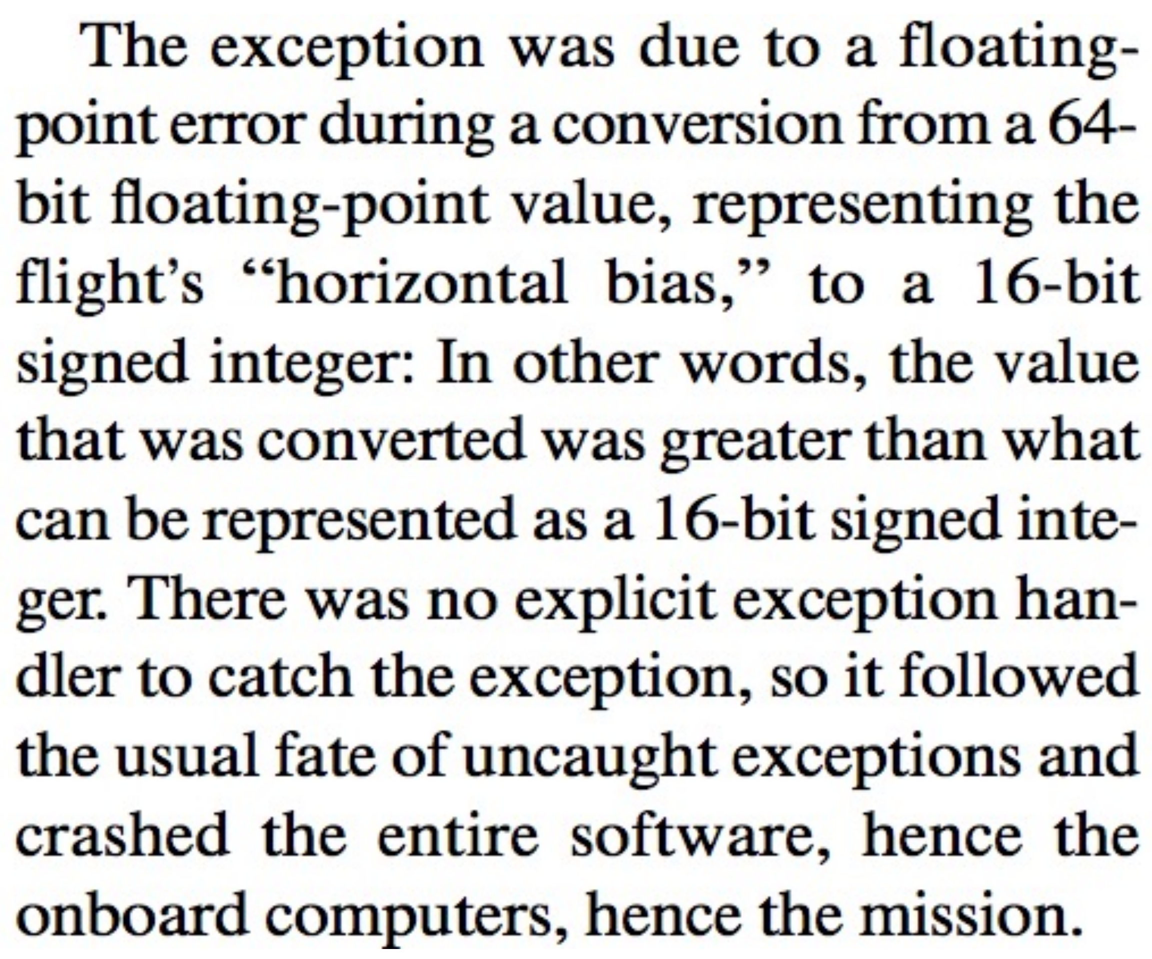
- Why do we test?

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

Ariane 5

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

Ariane 5



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.

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=562936

Quizz

https://www.menti.com/2h1p9xc4ad

Code: **2825 7782**

We have been trained to make assumptions

~~$x*x=x^2 \geq 0$~~	false for x= 46341 (and many more int)
~~$x+1>x$~~	**false for x=MAX_INT**
~~$(x * y) / x = y$~~	**false for x=0 or float x**
~~$(y / x) * x = y$~~	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
- Know how to interpret results?

increment(Integer.MAX_VALUE) ???

Remember this!

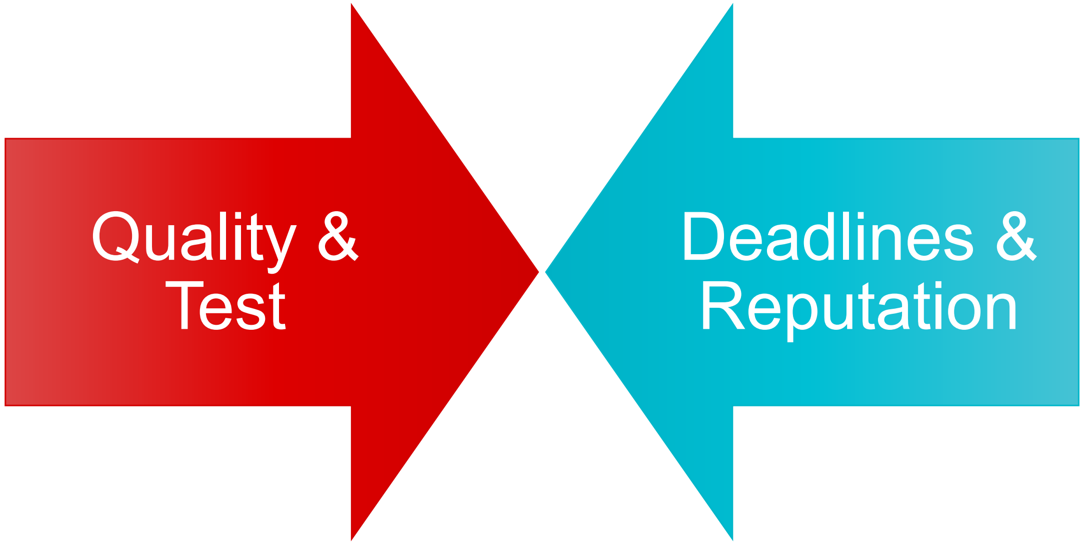
Program testing can be used to show the presence of bugs, but never to show their absence!

http://en.wikiquote.org/wiki/Edsger_W._Dijkstra
Referencing:
Notes On Structured Programming, 1972,
at the end of section 3,
On The Reliability of Mechanisms.

Edsger W. Dijkstra

Turing Award recipient, 1972

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

With system-level testing without unit testing	**With system-level testing and unit testing**
70 bugs	1 bug
16 weeks debugging time	50% less overall time
	5%-30% of the time writing tests 5%-20% running tests

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!

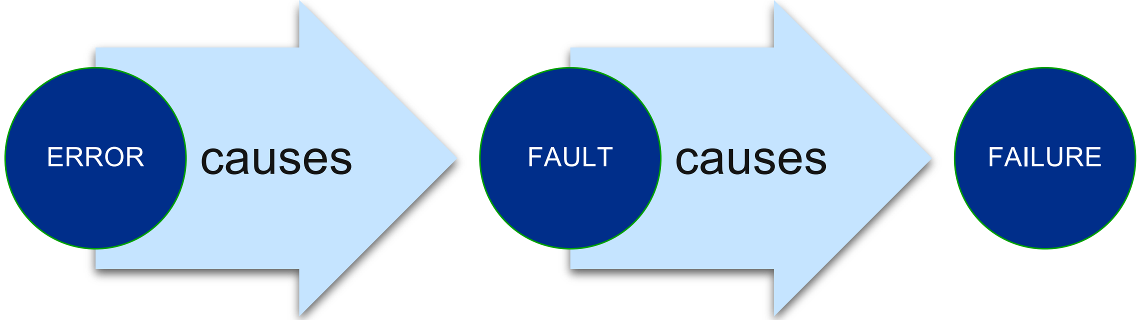
- Testing techniques are numerous and give a very large panel of possibilities

- 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

- 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

1. Types of testing
2. Testing scopes
3. Testing Processes
4. Testing Artifacts
5. 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

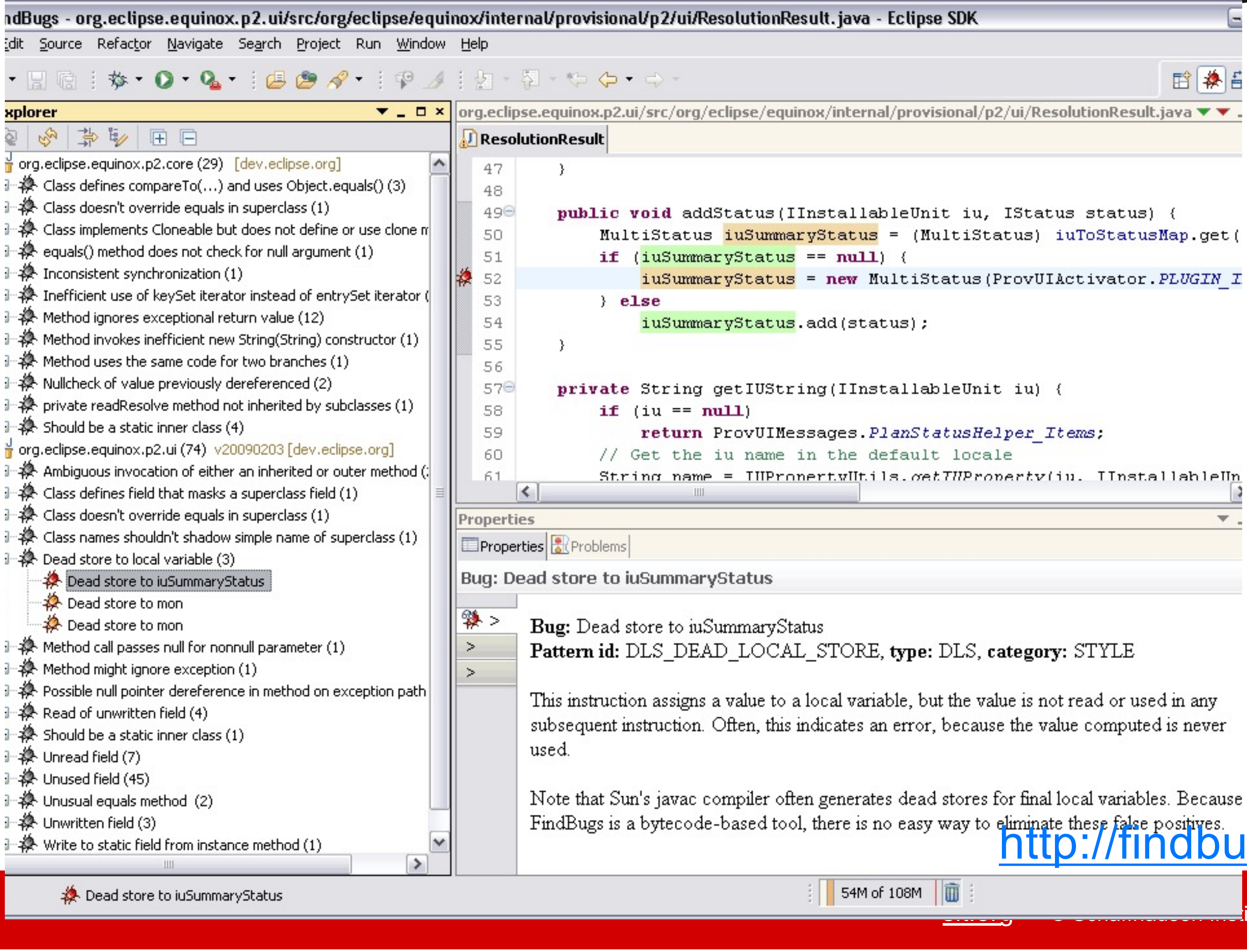
- Grey-Box: Mixes both of them… If the test engineer know some of the internal of the program, it uses those to design some of the tests, the rest uses black-box

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


ndBugs - org.eclipse.equinox.p2.ui/src/org/eclipse/equinox/internal/provisional/p2/ui/ResolutionResult.java - Eclipse SDK
Edit Source Refactor Navigate Search Project Run Window Help
xplorer
org.eclipse.equinox.p2.ui/src/org/eclipse/equinox/internal/provisional/p2/ui/ResolutionResult.java
ResolutionResult
Properties
Properties Problems
54M of 108M

org.eclipse.equinox.p2.core (29) [dev.eclipse.org]
- Class defines compareTo(...) and uses Object.equals() (3)
- Class doesn't override equals in superclass (1)
- Class implements Cloneable but does not define or use clone m
- equals() method does not check for null argument (1)
- Inconsistent synchronization (1)
- Inefficient use of keySet iterator instead of entrySet iterator (
- Method ignores exceptional return value (12)
- Method invokes inefficient new String(String) constructor (1)
- Method uses the same code for two branches (1)
- Nullcheck of value previously dereferenced (2)
- private readResolve method not inherited by subclasses (1)
- Should be a static inner class (4)

org.eclipse.equinox.p2.ui (74) v20090203 [dev.eclipse.org]
- Ambiguous invocation of either an inherited or outer method (
- Class defines field that masks a superclass field (1)
- Class doesn't override equals in superclass (1)
- Class names shouldn't shadow simple name of superclass (1)
- Dead store to local variable (3)
  - Dead store to iuSummaryStatus
  - Dead store to mon
  - Dead store to mon
- Method call passes null for nonnull parameter (1)
- Method might ignore exception (1)
- Possible null pointer dereference in method on exception path
- Read of unwritten field (4)
- Should be a static inner class (1)
- Unread field (7)
- Unused field (45)
- Unusual equals method (2)
- Unwritten field (3)
- Write to static field from instance method (1)

```
    }

    public void addStatus(IInstallableUnit iu, IStatus status) {
        MultiStatus iuSummaryStatus = (MultiStatus) iuToStatusMap.get(
        if (iuSummaryStatus == null) {
            iuSummaryStatus = new MultiStatus(ProvUIActivator.PLUGIN_I
        } else
            iuSummaryStatus.add(status);
    }

    private String getIUString(IInstallableUnit iu) {
        if (iu == null)
            return ProvUIMessages.PlanStatusHelper_Items;
        // Get the iu name in the default locale
        String name = IUPropertyUtils.getIUProperty(iu, IInstallableUn
```

Bug: Dead store to iuSummaryStatus

Bug: Dead store to iuSummaryStatus
Pattern id: DLS_DEAD_LOCAL_STORE, **type:** DLS, **category:** STYLE

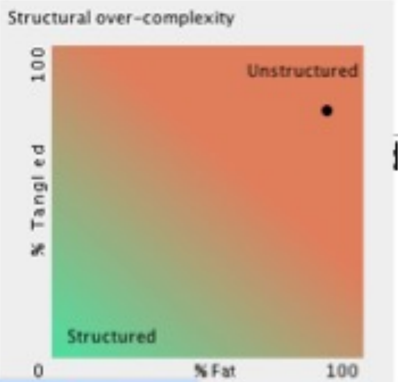
This instruction assigns a value to a local variable, but the value is not read or used in any subsequent instruction. Often, this indicates an error, because the value computed is never used.

Note that Sun's javac compiler often generates dead stores for final local variables. Because FindBugs is a bytecode-based tool, there is no easy way to eliminate these false positives.

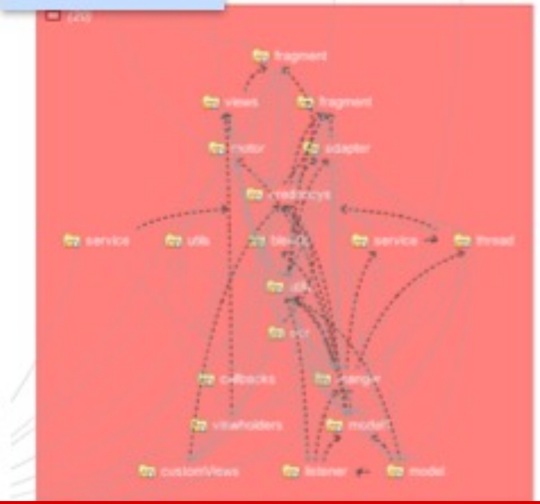
Dead store to iuSummaryStatus

http://findbugs.sourceforge.net

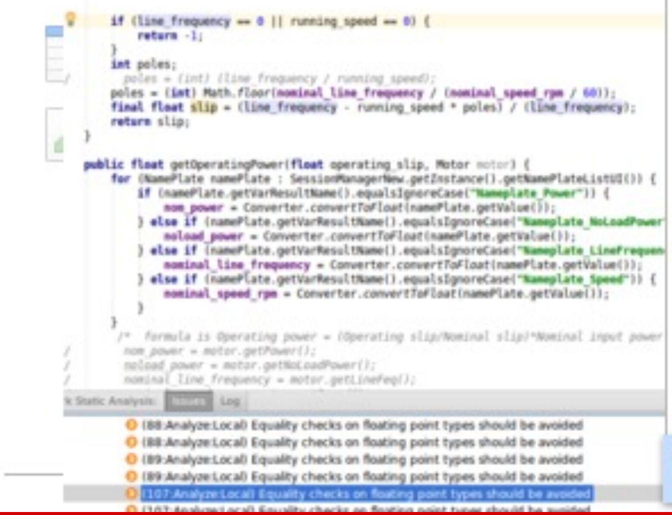
Other examples: Structure 101, Understand, Klocwork



Structure 101



Understand



```
        if (line_frequency == 0 || running_speed == 0) {
            return -1;
        }
        int poles;
//        poles = (int) (line_frequency / running_speed);
        poles = (int) Math.floor(nominal_line_frequency / (nominal_speed_rpm / 60));
        final float slip = (line_frequency - running_speed * poles) / (line_frequency);
        return slip;
    }

    public float getOperatingPower(float operating_slip, Motor motor) {
        for (NamePlate namePlate : SessionManagerNew.getInstance().getNamePlateListUI()) {
            if (namePlate.getVarResultName().equalsIgnoreCase("Nameplate_Power")) {
                nom_power = Converter.convertToFloat(namePlate.getValue());
            } else if (namePlate.getVarResultName().equalsIgnoreCase("Nameplate_NoLoadPower
                noload_power = Converter.convertToFloat(namePlate.getValue());
            } else if (namePlate.getVarResultName().equalsIgnoreCase("Nameplate_LineFrequen
                nominal_line_frequency = Converter.convertToFloat(namePlate.getValue());
            } else if (namePlate.getVarResultName().equalsIgnoreCase("Nameplate_Speed")) {
                nominal_speed_rpm = Converter.convertToFloat(namePlate.getValue());
            }
        }
        /*  formula is Operating power = (Operating slip/Nominal slip)*Nominal input power
/         nom_power = motor.getPower();
/         noload_power = motor.getNoLoadPower();
/         nominal_line_frequency = motor.getLineFeq();
```

```
(88:Analyze:Local) Equality checks on floating point types should be avoided
(88:Analyze:Local) Equality checks on floating point types should be avoided
(89:Analyze:Local) Equality checks on floating point types should be avoided
(89:Analyze:Local) Equality checks on floating point types should be avoided
(107:Analyze:Local) Equality checks on floating point types should be avoided
(107:Analyze:Local) Equality checks on floating point types should be avoided
```

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:

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

Test

Test:

```
@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

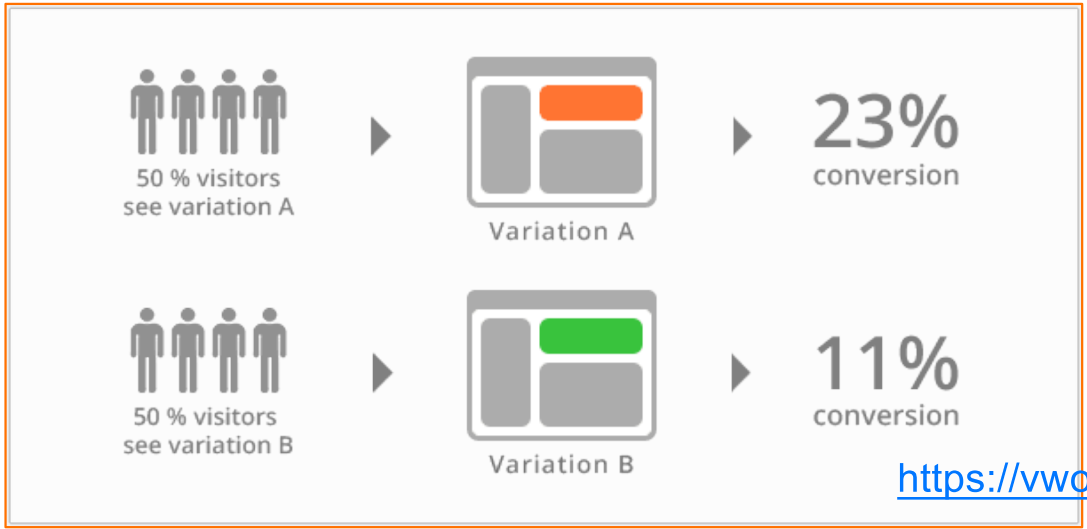
- Runs based on script

- 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:



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

```
@Test
public void test_1(){
    int j = 0;

assertTrue(increment(j)==1);
}
```

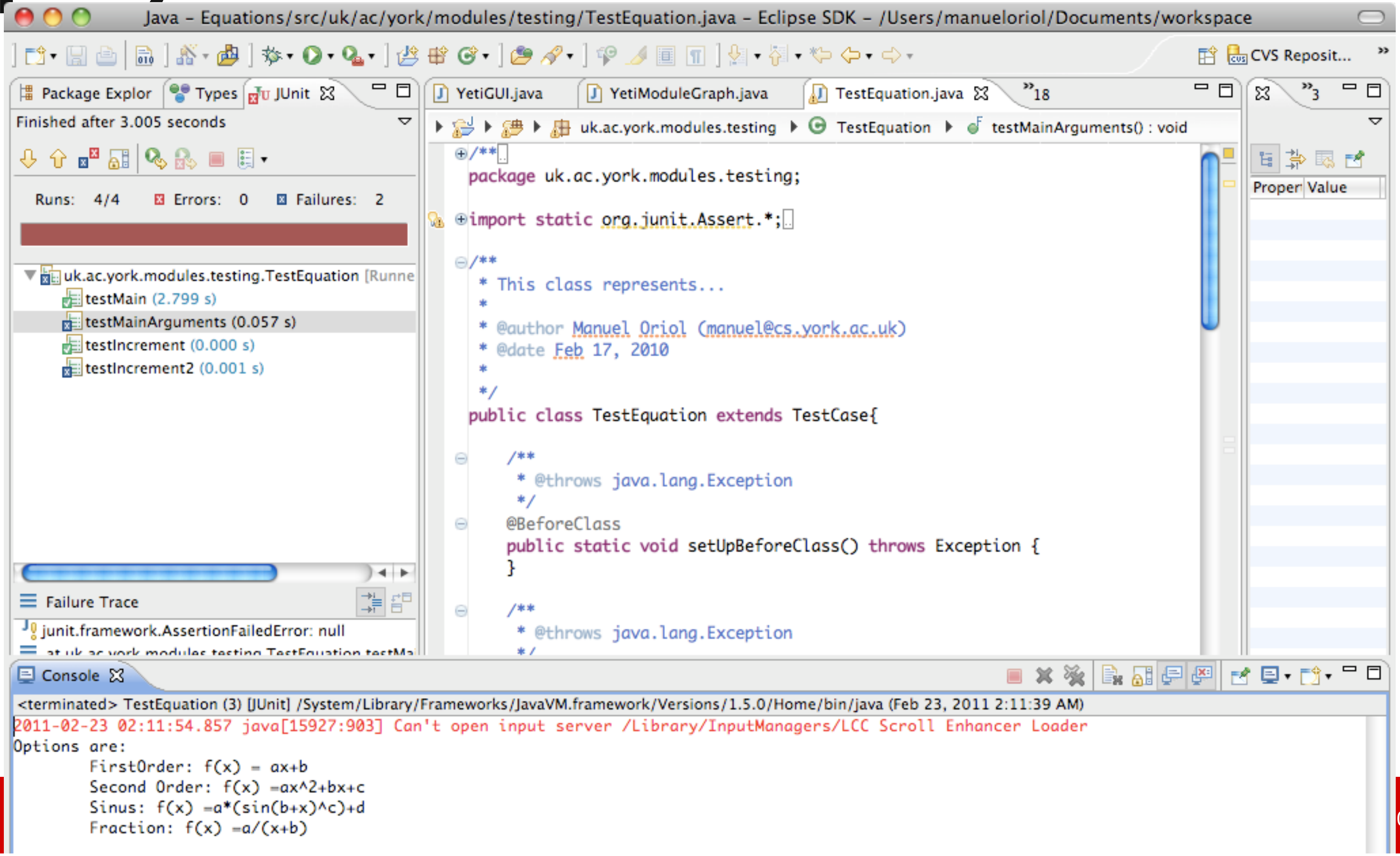
Example (2/2)

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

Test

```
@Test
public void test_1(){
    int j = 0;

assertTrue(increment(j)==1);
}
```

Regression Testing Tool: Junit (from Eclipse)



Java – Equations/src/uk/ac/york/modules/testing/TestEquation.java – Eclipse SDK – /Users/manueloriol/Documents/workspace
CVS Reposit...
Package Explor
Types
JUnit
YetiGUI.java
YetiModuleGraph.java
TestEquation.java
18
3
uk.ac.york.modules.testing
TestEquation
testMainArguments() : void
Proper
Value
Failure Trace
Console

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

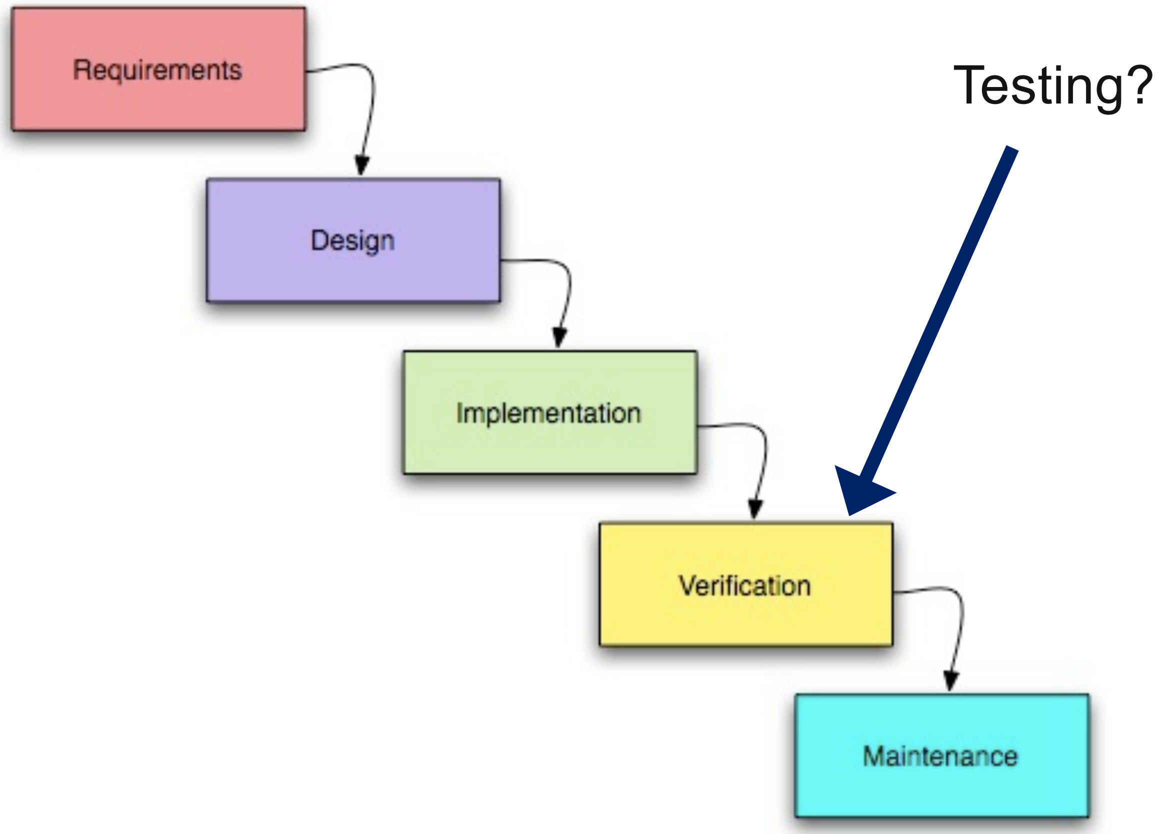
at uk.ac.york.modules.testing.TestEquation.testMa

```
/**
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)
```

Part III: Testing Processes

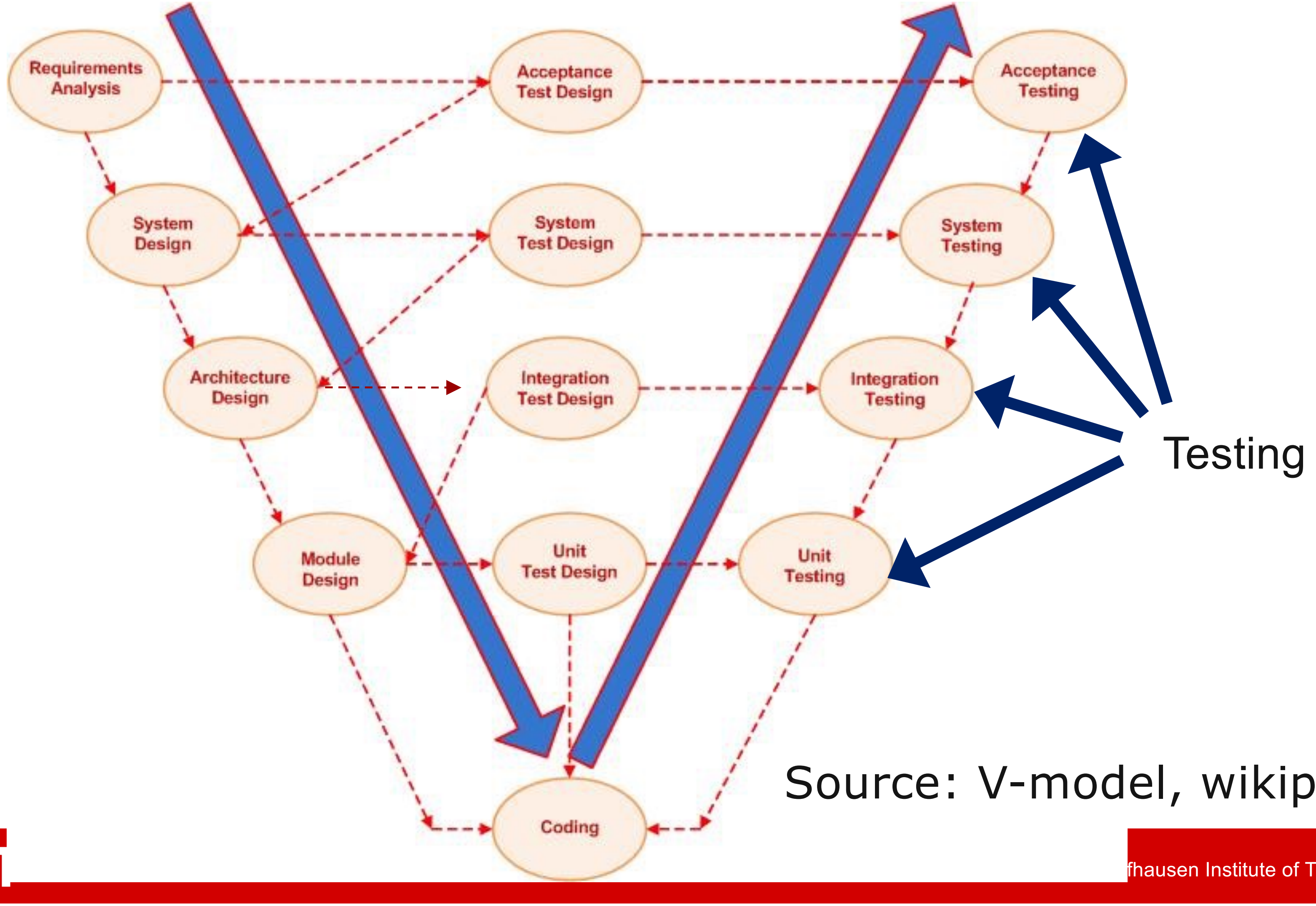
Original waterfall model



Testing?

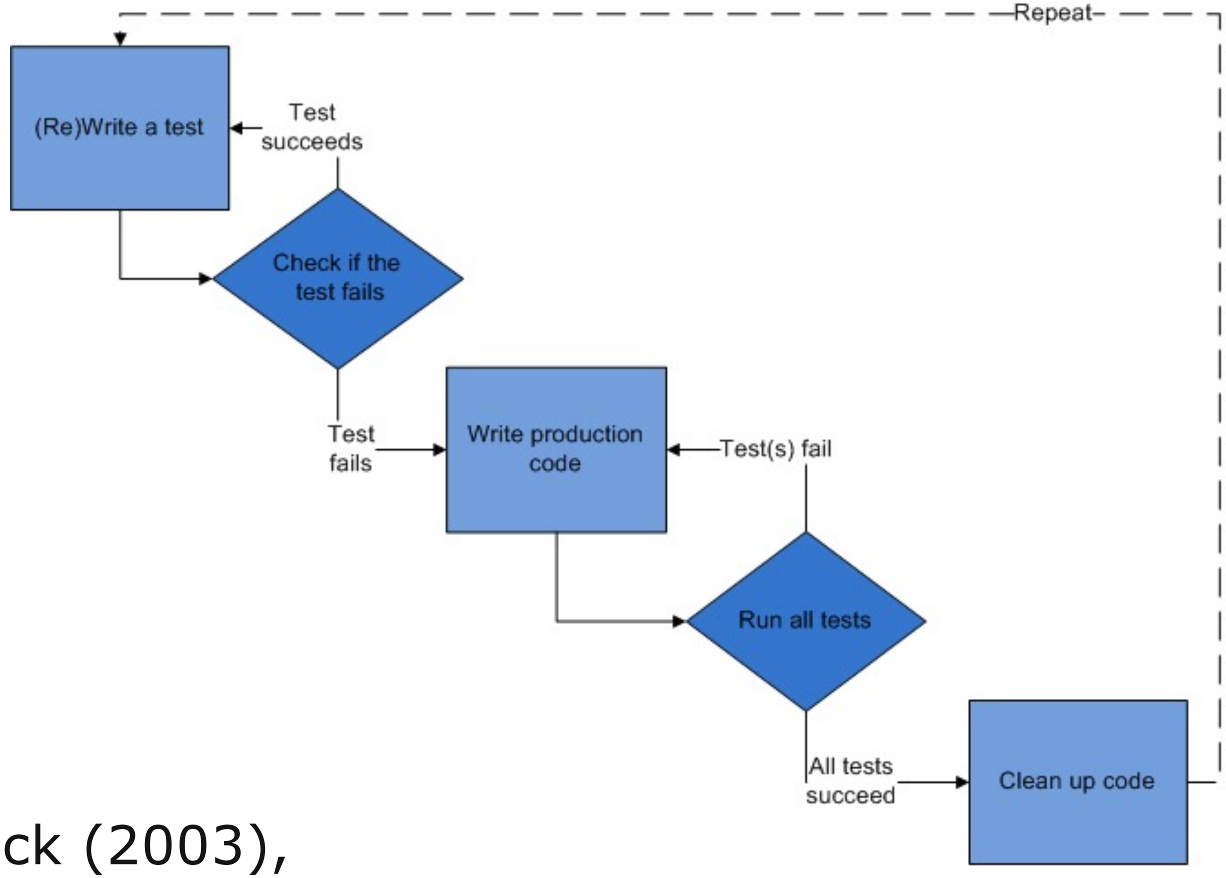
Winston Royce, 1970, source: wikipedia, waterfall model

V-Model



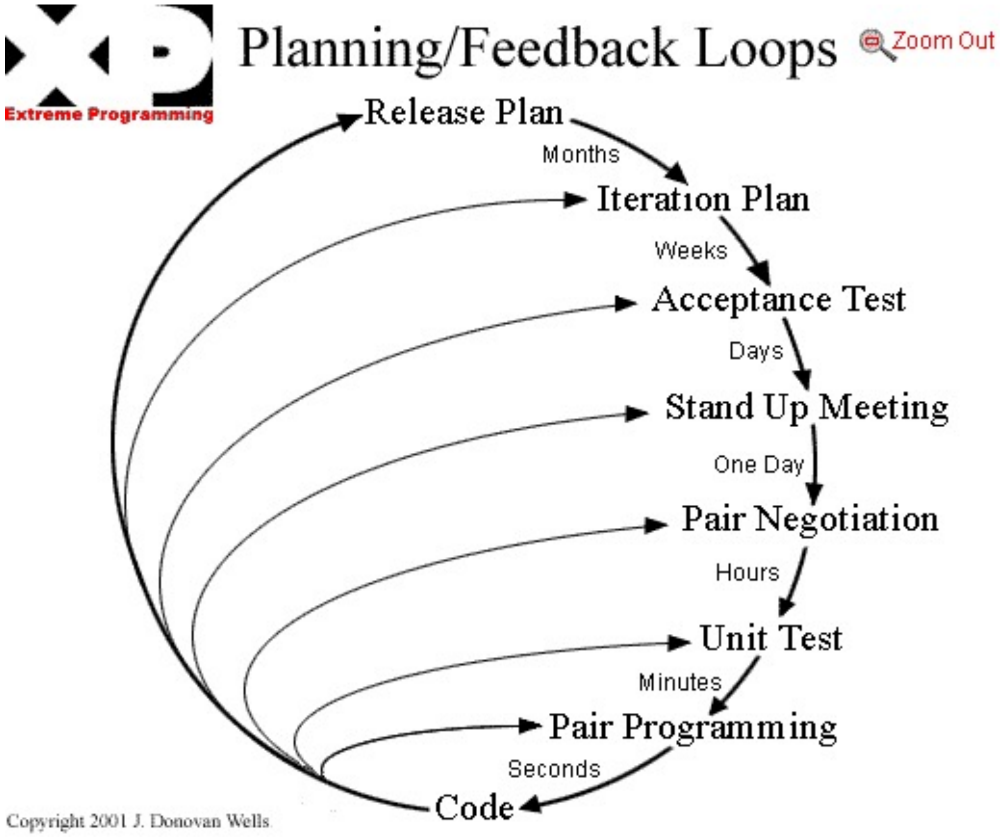
Source: V-model, wikipedia

Test-Driven Development



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

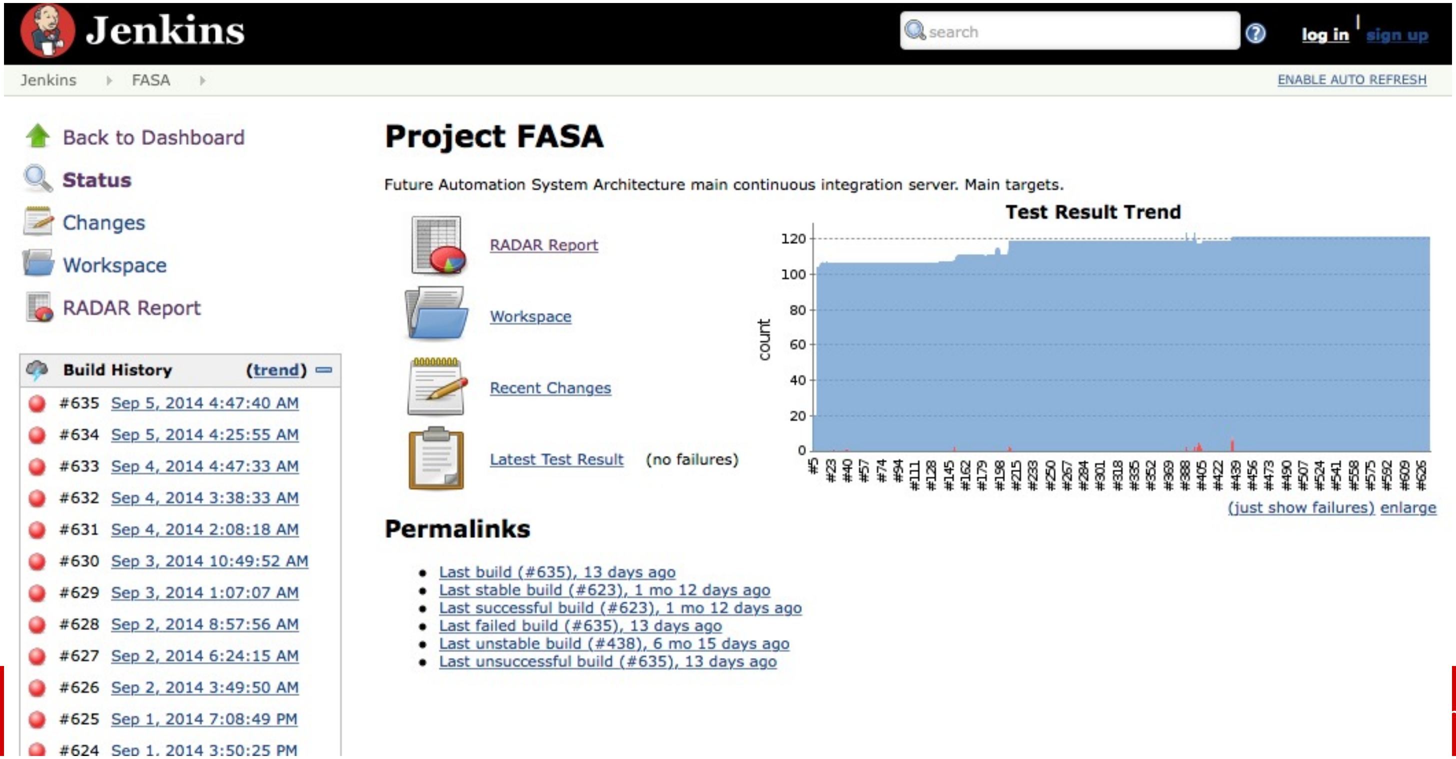
Extreme Programming



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



Project FASA

Future Automation System Architecture main continuous integration server. Main targets.

RADAR Report

Workspace

Recent Changes

Latest Test Result (no failures)

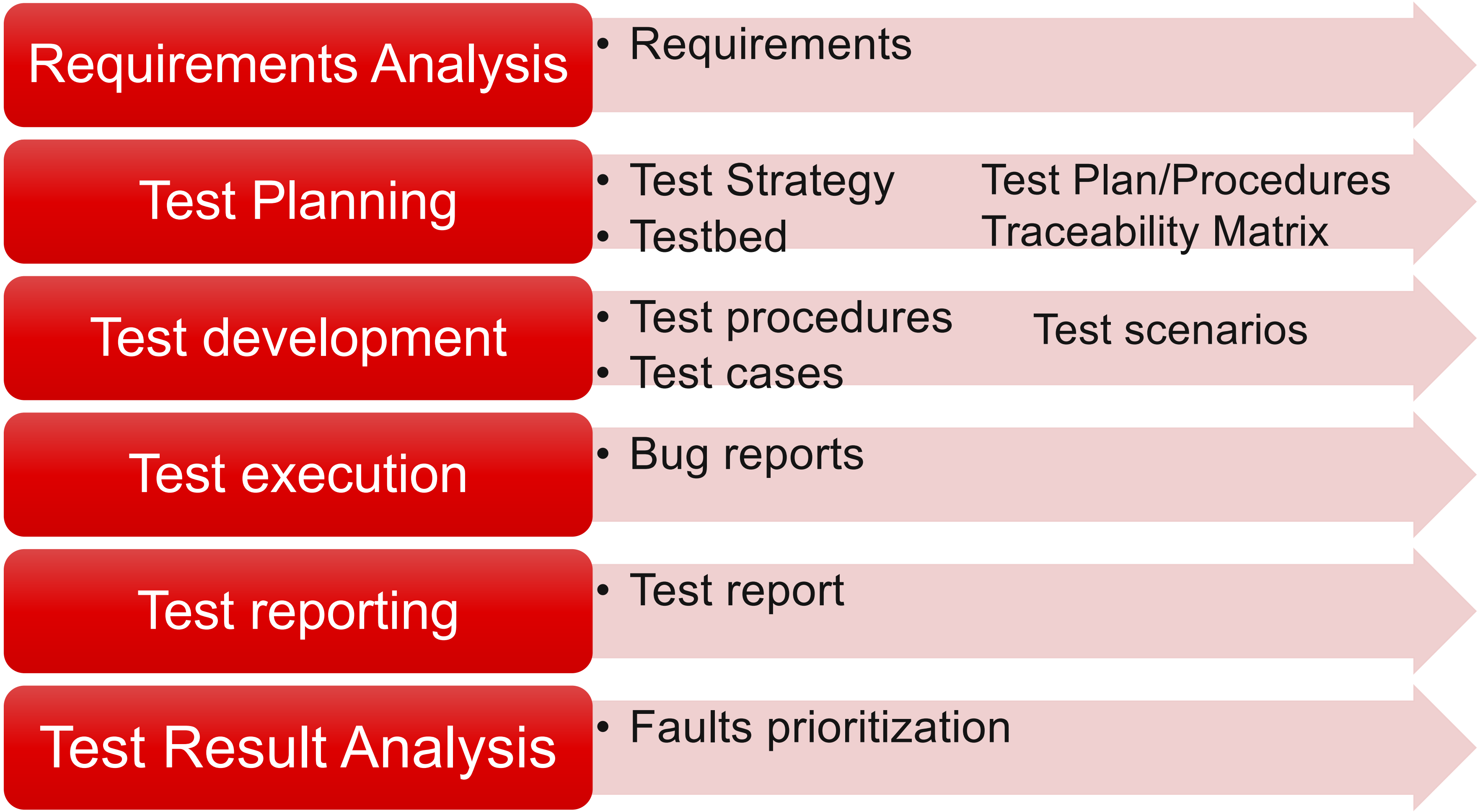
Build History (trend)

#635	Sep 5, 2014 4:47:40 AM
#634	Sep 5, 2014 4:25:55 AM
#633	Sep 4, 2014 4:47:33 AM
#632	Sep 4, 2014 3:38:33 AM
#631	Sep 4, 2014 2:08:18 AM
#630	Sep 3, 2014 10:49:52 AM
#629	Sep 3, 2014 1:07:07 AM
#628	Sep 2, 2014 8:57:56 AM
#627	Sep 2, 2014 6:24:15 AM
#626	Sep 2, 2014 3:49:50 AM
#625	Sep 1, 2014 7:08:49 PM
#624	Sep 1, 2014 3:50:25 PM

Permalinks

- Last build (#635), 13 days ago
- Last stable build (#623), 1 mo 12 days ago
- Last successful build (#623), 1 mo 12 days ago
- Last failed build (#635), 13 days ago
- Last unstable build (#438), 6 mo 15 days ago
- Last unsuccessful build (#635), 13 days ago

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

```
@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

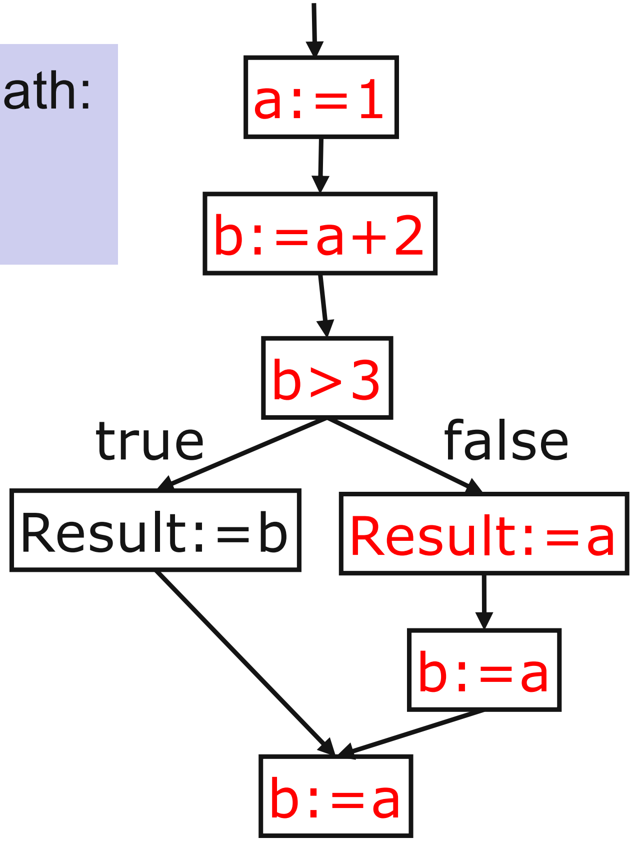
Typically function coverage should be 100%

Statement coverage

Percentage of statements that were executed

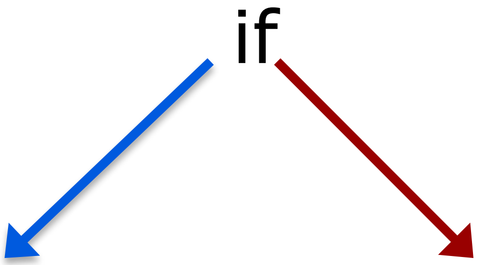
Example

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



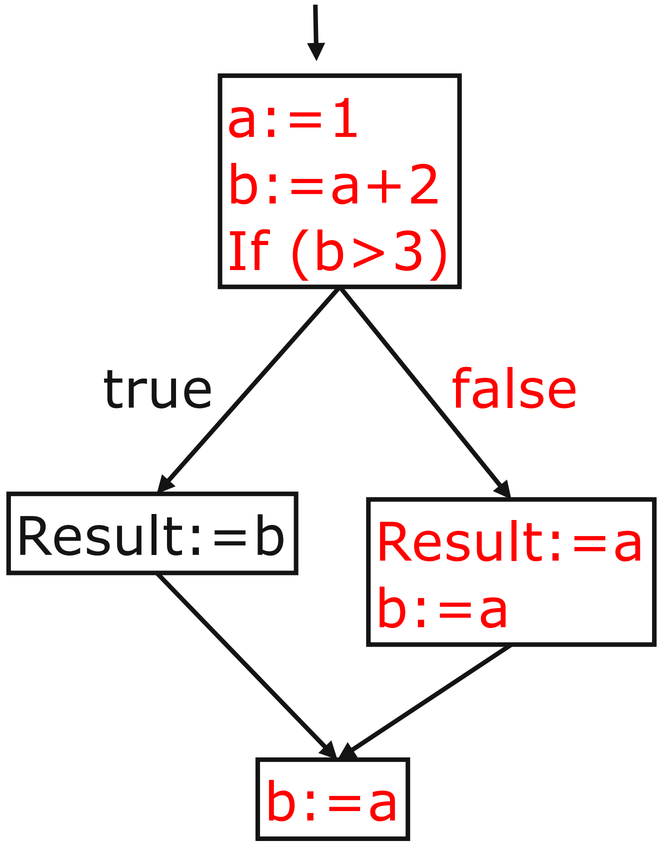
Decision coverage

- Each time a program has a branching instruction (if, for, while…) this create two branches.
- Decision coverage is the percentage of these branches that were executed by a test suite.



Example

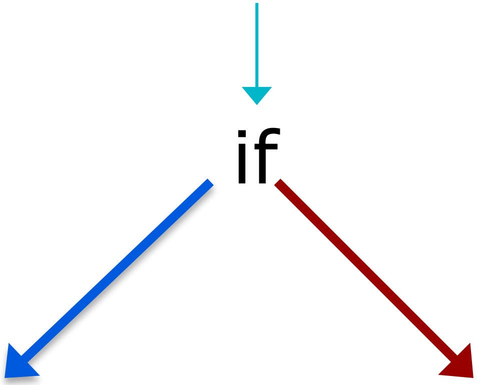
Decision Coverage of the red branches: 50%



Branch coverage

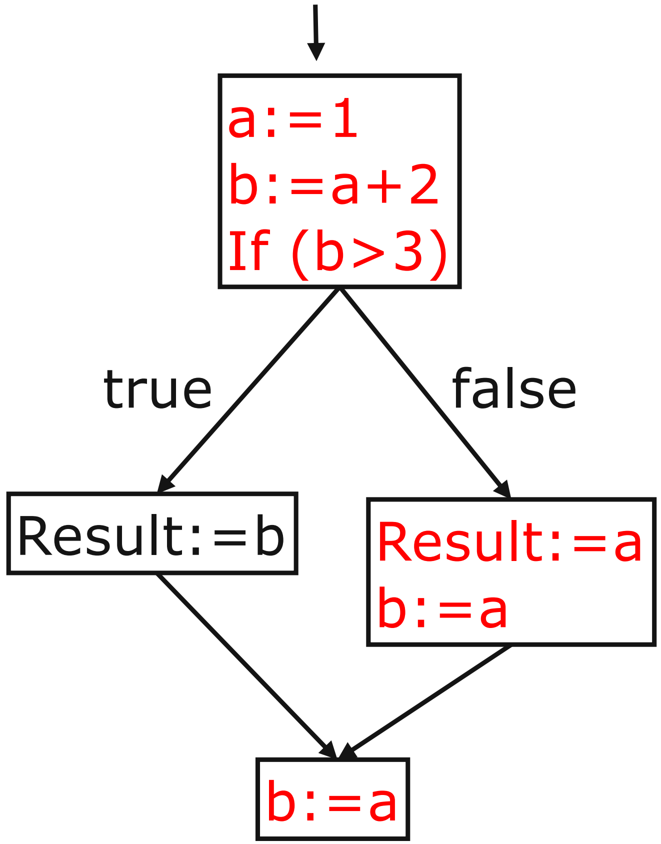
- Each time one has a branching instruction (if, for, while…) this create two branches.

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



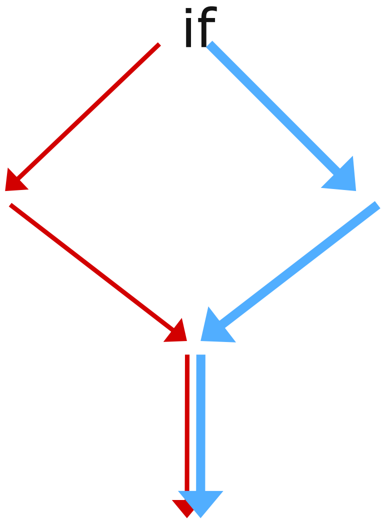
Example

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



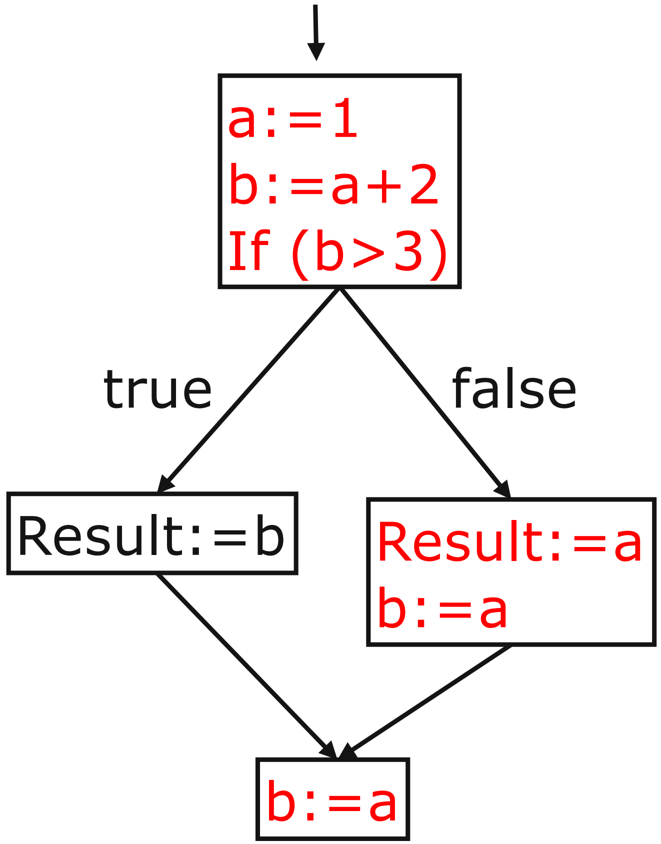
Path Coverage

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



Example

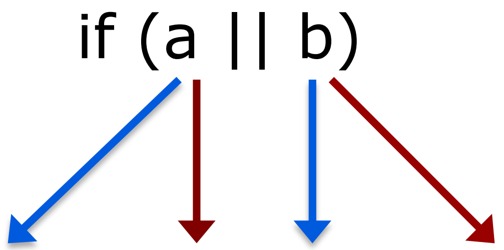
Coverage of the red red path:
50%
(1/2 paths)



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.

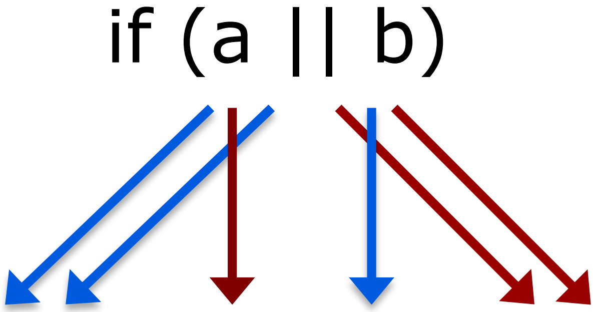


if (a || b)

100% obtained with (a,b) = (true, false) and (false,true)

Modified Condition/Decision Coverage (MCDC)

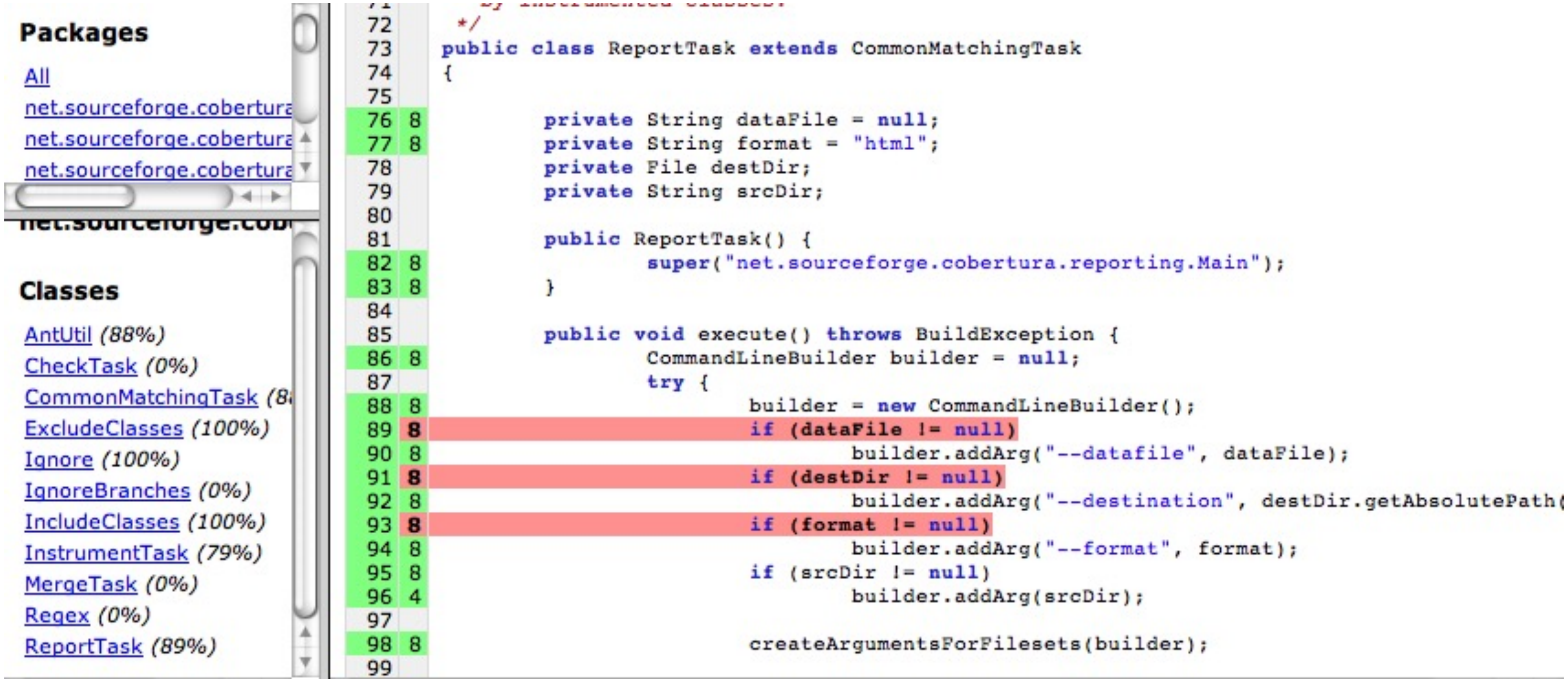
- Consists of:
  - 100% branch coverage
  - 100% condition coverage
  - Each entry/exit point is exercised
  - Each condition affects the behaviour independently



```
if (a || b)
```

DO-178B, Software Considerations in Airborne Systems and Equipment Certification

Tools to calculate the coverage: Cobertura



Packages

All
net.sourceforge.cobertura
net.sourceforge.cobertura
net.sourceforge.cobertura

net.sourceforge.cob

Classes

AntUtil	(88%)
CheckTask	(0%)
CommonMatchingTask	(8
ExcludeClasses	(100%)
Ignore	(100%)
IgnoreBranches	(0%)
IncludeClasses	(100%)
InstrumentTask	(79%)
MergeTask	(0%)
Regex	(0%)
ReportTask	(89%)

```
71        by instrumented classes.
72       */
73      public class ReportTask extends CommonMatchingTask
74      {
75
76  8           private String dataFile = null;
77  8           private String format = "html";
78              private File destDir;
79              private String srcDir;
80
81              public ReportTask() {
82  8                   super("net.sourceforge.cobertura.reporting.Main");
83  8           }
84
85              public void execute() throws BuildException {
86  8                   CommandLineBuilder builder = null;
87                      try {
88  8                           builder = new CommandLineBuilder();
89  8                           if (dataFile != null)
90  8                                   builder.addArg("--datafile", dataFile);
91  8                           if (destDir != null)
92  8                                   builder.addArg("--destination", destDir.getAbsolutePath(
93  8                           if (format != null)
94  8                                   builder.addArg("--format", format);
95  8                           if (srcDir != null)
96  4                                   builder.addArg(srcDir);
97
98  8                           createArgumentsForFilesets(builder);
99
```

http://cobertura.sourceforge.net/

My recommendations

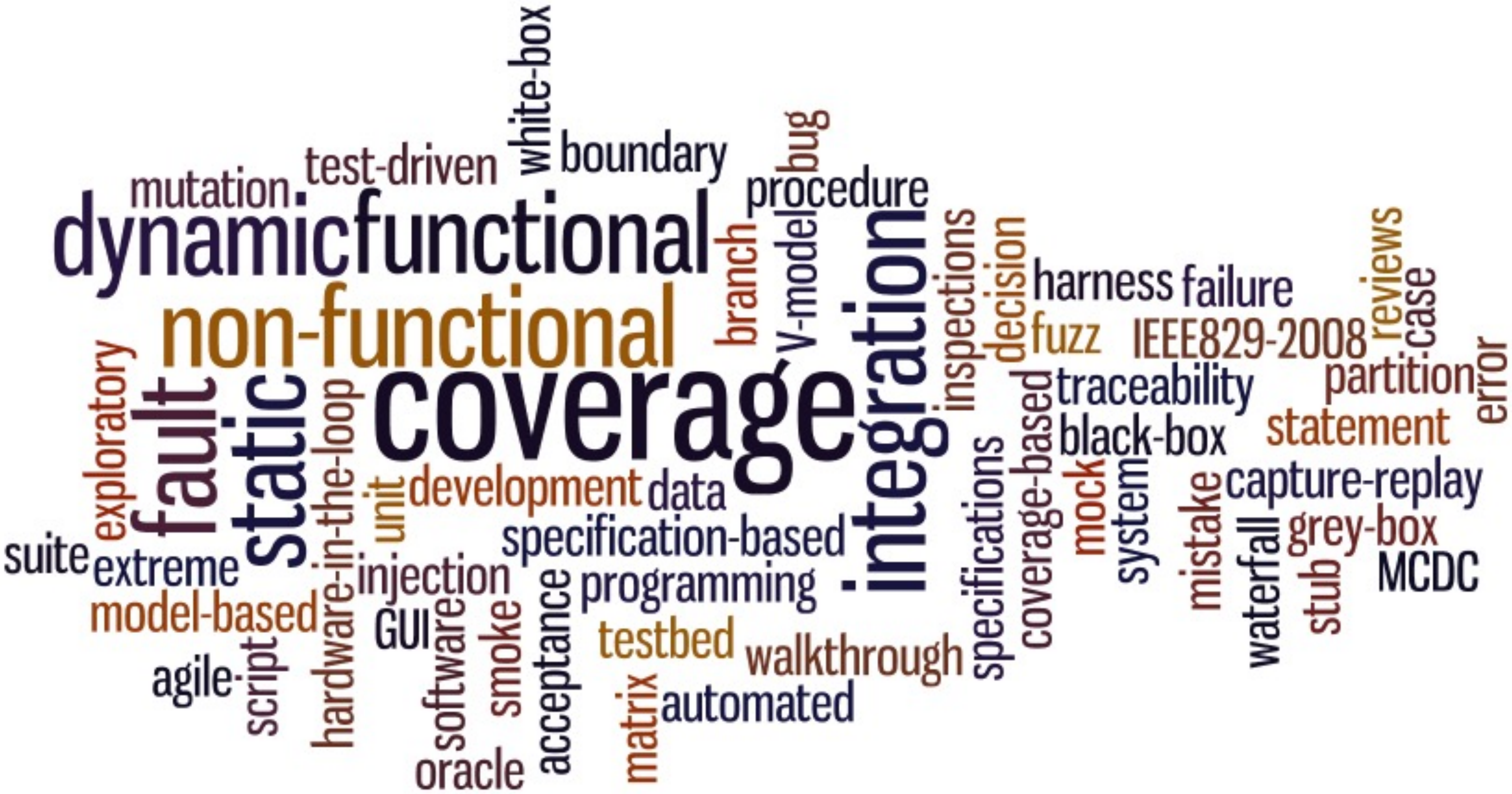
- Write tests as a part of the coding activity
  - Not at the end, not at the beginning, rather per unit
- Write unit tests
  - Use unit testing frameworks like JUnit
  - Monitor decision coverage and try to get it close to 100%
- Write integration tests
  - use scripts and specific tools like Selenium
- 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

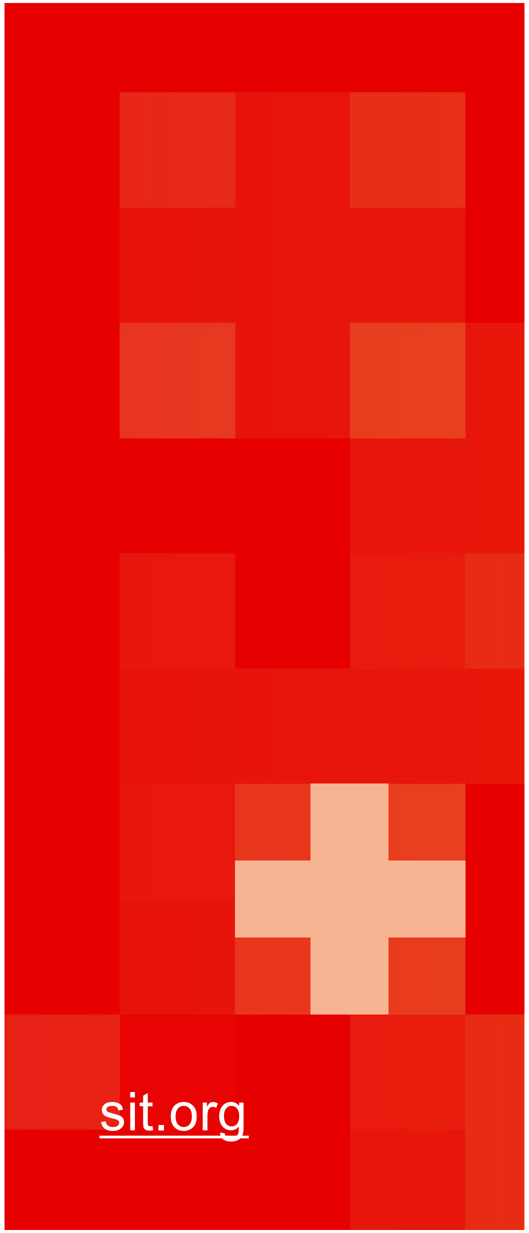
- This presentation only gives a high level understanding of the techniques used in testing there is far more to learn

Some terms used in software testing

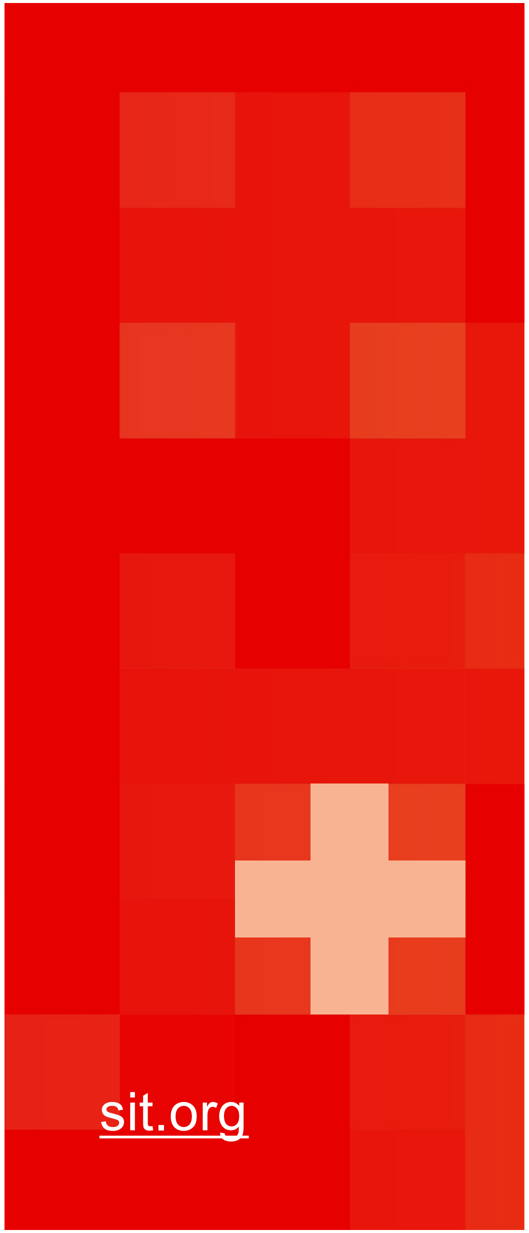


S⠶T

Thank you!



sit.org



sit.org