

Software testing

(with emphasis on JUnit testing)

Martin Senger

m.senger@cgiar.org

IRRI (International Rice Research Institute),
Philippines

Let's start from the end...

- What we want to achieve
 - To have software that behaves well
 - To be able to maintain this software at relatively low cost
- Reasonably well known path to this goal is software testing
 - But less known (or agreed on) is how to test, what to test, when to test and who tests

A bit of theory (just one slide)

- Software testing (ideally) consists of
 - Unit testing
 - test small(est) program components (often methods)
by programmers
 - Integration testing
 - software modules are integrated and tested together
 - here (usually) belong any tests that require
 - access to databases
 - network communication
 - System testing
 - test of a complete system, including hardware (and end-users patience)

Testing is not only a quality check

- Testing has a **documentary value**
 - it shows how to use your code
 - it stays close to the code (as with JavaDoc)
- Development with testing is a **design technique**
 - if it not easy to write unit tests for your software, it may indicate that the whole design is faulty

Testing is not a fun, and it costs









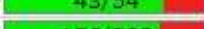

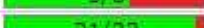
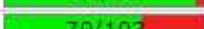
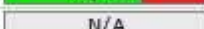

- (My) Four moods of software development
 - Designing a component is a **fun**
 - Implementing it is a **work**
 - Writing tests is the **dark side** of my job
 - Documenting it is a **nightmare**
- Nobody knows how much it costs
 - but it is probably about 25% of your coding time
 - that's why you should test when it is worth to
 - there are some estimates indicating that only about 20% of developers uses unit testing

How we do it in GCP Java projects

- Every project can be used **both** from command-line Ant, and from Eclipse
- We encourage to use **JUnit 4** testing
 - easier to write tests for protected methods
 - class-scope setup and clean up methods
 - much easier (and better) testing of exceptions
 - testing performance and timeouts are possible
 - still backward compatible with JUnit 3
- So far, we have not measured test coverage
 - e.g. using Cobertura tool; **should we?**

Cobertura report example

Coverage Report - All Packages

Package /	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	55	75%  1625/2179	64%  472/738	2.319
net.sourceforge.cobertura.ant	11	52%  170/330	43%  40/94	1.848
net.sourceforge.cobertura.check	3	0%  0/150	0%  0/76	2.429
net.sourceforge.cobertura.coveragedata	13	N/A  N/A	N/A  N/A	2.277
net.sourceforge.cobertura.instrument	10	90%  460/510	75%  123/164	1.854
net.sourceforge.cobertura.merge	1	86%  30/35	88%  14/16	5.5
net.sourceforge.cobertura.reporting	3	87%  116/134	80%  43/54	2.882
net.sourceforge.cobertura.reporting.html	4	91%  475/523	77%  156/202	4.444
net.sourceforge.cobertura.reporting.html.files	1	87%  39/45	62%  5/8	4.5
net.sourceforge.cobertura.reporting.xml	1	100%  155/155	95%  21/22	1.524
net.sourceforge.cobertura.util	9	60%  175/291	69%  70/102	2.892
someotherpackage	1	83%  5/6	N/A  N/A	1.2

<http://cobertura.sourceforge.net/>

To be more concrete...

- Each project has `xmls/junit.xml` Ant's file
- There is an Ant's task `ant test-junit` that:
 - checks the presence of the JUnit library
 - compiles tests
 - runs tests
- Testing code is outside the main code tree
 - in `src/test/java` and `src/test/junit-resources`
- Optionally: each test class has code that allows running tests outside of Ant or from a JUnit 3 tools

Example of a test class

```
@Test
public void matchProperties() {
    assertTrue (Config.addConfigPropertyFile (TEST_CONFIG_PROPS));
    Properties props =
        Config.getMatchingProperties ("grid.env", "org.classic.HelloWorld");
    assertTrue ("Not a correct number of the matching properties.",
        props.size() == 3);
    assertEquals ("Matching properties mismatch.",
        "ein", props.getProperty ("One"));
}
```

```
@Test
public void getStrings() {
    assertTrue (Config.addConfigPropertyFile (TEST_CONFIG_PROPS));
    String[] elems = Config.getStrings ("element", null, null);
    assertFalse ("Returned array should not be null.", elems == null);
    assertEquals ("Wrong size of the returned array.", 5, elems.length);
}
```

Running the tests - example

```
C:\Users\martin\Desktop\Pantheon Config>ant test-junit
Buildfile: build.xml
checkmaven:
initmaven:
init:
junit-init:
junit-present:
initeclipse:
config:
compile:
compile-tests:
do-junit:
[junit] Running org.build.LogTest
[junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.023 sec
[junit] Running org.generationcp.core.config.BasicUsageTest
[junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.022 sec
[junit] Running org.generationcp.core.config.ConfigTest
[junit] Tests run: 14, Failures: 0, Errors: 0, Time elapsed: 0.23 sec
[junit] Running org.generationcp.core.utils.RefResolverTest
[junit] Tests run: 2, Failures: 0, Errors: 0, Time elapsed: 0.108 sec

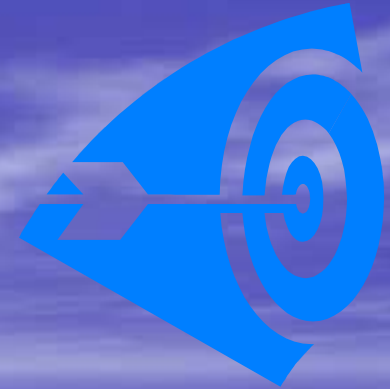
test-junit:

BUILD SUCCESSFUL
Total time: 3 seconds
```

(The) Frequently Asked Question

- How can I test the GUI?
 - Easy answer: It is not simple. Sometimes even impossible. It is definitely not a unit testing.
 - But you can make it easier by:
 - Using **better design** of your application
 - Model-View-Controller pattern
 - Okay, but what else and what next?
 - Well, I do not know, actually...

Thank you...



- When you are completely exhausted by writing more and more test code because your boss insists and insists... tell him/her:
 - A bus station is where a bus stops.
A train station is where a train stops.
On my desk I have a work station...

[copied from an Internet forum]