Java JUnit

for Unit Testing

TEST COVERAGE AND SUMMARY OF WHITE BOX TESTING

ERIC Y. CHOU, PH.D. IEEE SENIOR MEMBER

JUnit

Java Code Coverage Tools

https://en.wikipedia.org/wiki/Java_Code_Coverage_Tools

Jcov, JaCoCo, Clover, Cobertura, EMMA, Serenity

Eclipse + JaCoCo + EclEmma is a good combination for Eclipse.

JUnit

What is code coverage

Code coverage, in short, is all about how thoroughly your tests exercise your code base. The intent of tests, of course, is to verify that your code does what it's expected to, but also to document what the code is expected to do. Taken further, code coverage can be considered as an indirect measure of quality -- indirect because we're talking about the degree to what our tests cover our code, or simply, the quality of tests. In other words, **code coverage** is not about verifying the end product's quality.

JUnit

Measures - Statement coverage

Statement coverage, also known as line coverage, is a measure which indicates the degree to which individual statements are getting executed during test execution. The statement coverage measure is possibly the easiest to implement considering that it can be applied over bytecode, which is a lot simpler to parse than source code having a touch of human handiwork in it. Statement coverage is also probably the most used by developers because it is easy to associate with source code lines -- "ah, that setter really doesn't get executed when my test calls it likes this!" However, the simplicity comes with a cost: statement coverage is unable to tell too much about how well you have covered your logic -- only whether you've executed each statement at least once.

Measures -Decision coverage

(also known as branch coverage)

Decision coverage (also known as branch coverage) is a measure based on whether decision points, such as if and while statements, evaluate to both true and false during test execution, thus causing both execution paths to be exercised. Decision coverage is also relatively simple, which is both its pro and its con. The downside is that the measure doesn't take into consideration how the boolean value was gotten -- whether a logical OR was short-circuited or not, for example, leaving whatever code was in the latter part of the statement unexecuted.

JUnit

JUnit

Measure – Variants for Decision Coverage

condition coverage, path coverage

This deficit of the decision coverage measure is tackled to some degree by **condition coverage**, which extends the boolean evaluation of decision coverage into the sub-expressions (separated by logical ANDs and ORs) as well, making sure each of them is evaluated to both true and false.

Moving on, **path coverage** represents yet another interesting measure. **Path coverage measures whether each possible path from start (method entry) to finish (return statement, thrown exception) is covered.**

JUnit

Function coverage

Function coverage is a measure for verifying that each function (method) is invoked during test execution. In all its simplicity, function coverage is a very easy way to spot the biggest gaps in your code coverage.

Brute Force Monitor

// Access monitor
p[5] = printOnce(5, p[5]);

```
// Point-visited record
boolean[] p = {false, false, false, false, false,
               false, false, false, false, false
              };
// Information Collection method
public static boolean printOnce(int i, boolean visited){
      if (!visited)
       System.out.print(i+"->");
     return true;
   }
```

JUnit

Information Collected at Monitors

Node ID Number

Time Stamp

Tried (number of visits)

Brute-force method can only be used for debug mode, not even integration mode. For integration mode and release mode, tools are suggested.

Eclipse+JUnit+JaCoCo+EclEmma

http://eclemma.org/installation.html

Coverage

TestAllPackages (Feb 13, 2012 9:52:22 AM)

Element	Coverage	Covered Lines	Missed Lines	Total Lines
commons-collections	80.7 %	11092	2646	13738
src	80.7 %	11092	2646	13738
org.apache.commons.collections	77.1 %	3991	1188	5179
org.apache.commons.collections.bag	66.9 %	234	116	350
org.apache.commons.collections.bidimap	91.2 %	964	93	1057
AbstractBidiMapDecorator.java	85.7 %	6	1	7
AbstractBidiMapDecorator	85.7 %	6	1	7
AbstractBidiMapDecorator(BidiMap)	100.0 %	2	0	2
getBidiMap()	100.0 %	1	0	1
getKey(Object)	100.0 %	1	0	1
inverseBidiMap()	0.0 %	0	1	1

JUnit

Coverage Report

The Coverage view shows all analyzed Java elements within the common Java hierarchy. Individual columns contain the following numbers for the active session, always summarizing the child elements of the respective Java element:

- **Coverage ratio**
- **Items covered**
- **Items not covered**
- **Total items**

JUnit

Installation of EclEmma

Go Eclipse!!!