Are You Covered?

Keith D Gregory
Philly JUG

14 October 2009


Coverage
tools lie

What is Coverage?

Measurement of how well your tests exercise your code

Metric: percent coverage

Coverage tools modify bytecode, inserting counters, then report result

Bytecode Modification

```
public class Hello
{
    public static void main(String[] argv)
    throws Exception
    {
        System.out.println("Hello, World");
    }
}
```


```
public static void main(java.lang.String[])
throws java.lang.Exception;
  Code:
   0:   getstatic       #19; //Field
java/lang/System.out:Ljava/io/PrintStream;
   3:   ldc     #25; //String Hello, World
   5:   invokevirtual   #27; //Method
java/io/PrintStream.println:(Ljava/lang/String;)V
   8:   return
```

```
public static void main(java.lang.String[])
throws java.lang.Exception;
  Code:
   0:       getstatic   #41; //Field $VR4019:[[Z
   3:       iconst_1
   4:       aaload
   5:       astore_1
   6:       getstatic   #19; //Field
java/lang/System.out:Ljava/io/PrintStream;
   9:       ldc         #25; //String Hello,
World
   11:      invokevirtual           #27; //Method
java/io/PrintStream.println:(Ljava/lang/String;)V
   14:      aload_1
   15:      iconst_0
   16:      iconst_1
   17:      bastore
   18:      return
```

Levels of Coverage

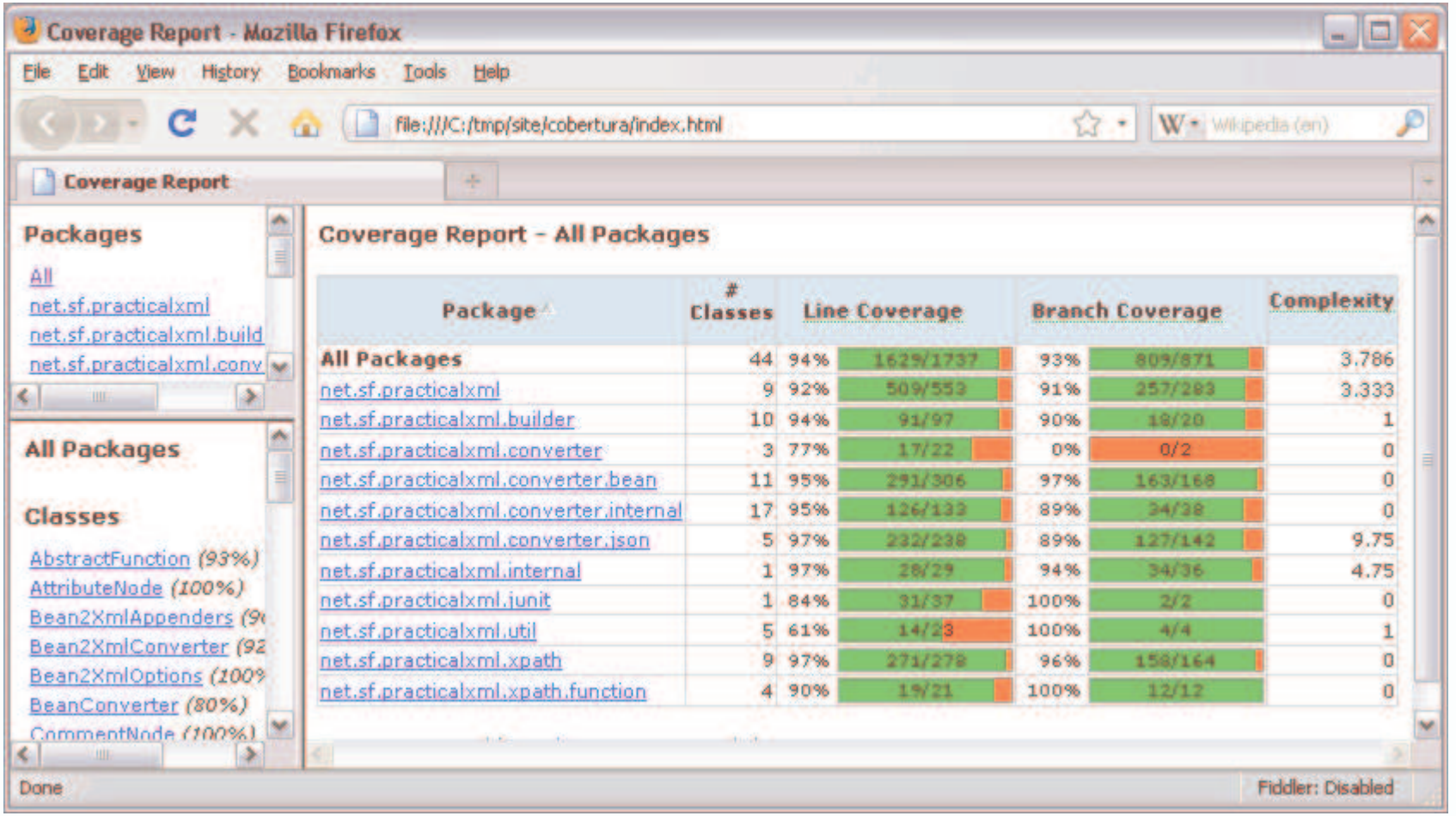
Class / Method / Line

- Obsolete

Basic Block

- Inserts counters after every branch
- Shows partial coverage of if / loop / ternary

Reporting


Coverage Report - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/tmp/site/cobertura/index.html
Coverage Report
Packages
All
net.sf.practicalxml
net.sf.practicalxml.build
net.sf.practicalxml.conv
All Packages
Classes
AbstractFunction (93%)
AttributeNode (100%)
Bean2XmlAppenders (9
Bean2XmlConverter (92
Bean2XmlOptions (100
BeanConverter (80%)
CommentNode (100%)
Coverage Report - All Packages
Package
Classes
Line Coverage
Branch Coverage
Complexity
All Packages 44 94% 1629/1737 93% 809/871 3.786
net.sf.practicalxml 9 92% 509/553 91% 257/283 3.333
net.sf.practicalxml.builder 10 94% 91/97 90% 18/20 1
net.sf.practicalxml.converter 3 77% 17/22 0% 0/2 0
net.sf.practicalxml.converter.bean 11 95% 291/306 97% 163/168 0
net.sf.practicalxml.converter.internal 17 95% 126/133 89% 34/38 0
net.sf.practicalxml.converter.json 5 97% 232/238 89% 127/142 9.75
net.sf.practicalxml.internal 1 97% 28/29 94% 34/36 4.75
net.sf.practicalxml.junit 1 84% 31/37 100% 2/2 0
net.sf.practicalxml.util 5 61% 14/23 100% 4/4 1
net.sf.practicalxml.xpath 9 97% 271/278 96% 158/164 0
net.sf.practicalxml.xpath.function 4 90% 19/21 100% 12/12 0
Done
Fiddler: Disabled

Tools

Cobertura

- http://cobertura.sourceforge.net/
- Plugins for Ant, Maven 2
- Licensed under GPL

Emma

- http://emma.sourceforge.net/
- Plugins for Ant, Maven 1, Eclipse
- Can instrument on-the-fly via classloader
- Licensed under IBM CPL

When to Run?

- Daily or after Significant Change
  - Every build is too often
  - “Just before release” is not enough
  - Run first thing in morning, or after lunch
- IDE integration encourages usage
- Useful habit: follow with FindBugs™

Coverage Beyond the Unit Test

- Interactive (desktop) applications
  - Instrumentation adds minimal overhead
  - Goal: identify action invocations (class-level coverage sufficient)
- In-container testing of web-apps
  - Integration tests
  - Check on manual QA

The Problem

Coverage tools can only tell you whether your tests exercise your code

They won't tell you about missing features or missed specifications

The report can be misleading

You can get 100% coverage without fully testing your code

Misleading Reports

It's easy to put counters in bytecode

It's harder to tie back to Java

- Particularly if Hotspot modifies that code

Different tools may report different coverage for same code

- Example: switch statement

Independent Paths

How many paths through this code?

```
public int myFunc(int a, int b, int c)
{
    if (a > 5)
        c += a;
    if (b > 5)
        c += b;
    return c;
}
```

How many tests for 100% coverage?

```
assertEquals(2, myFunc(2, 2, 2));
assertEquals(21, myFunc(7, 7, 7));
```

Independent Paths

Every non-trivial program has independent paths

Number of paths increases geometrically with number of “ifs”

Multi-threaded programs add time dimension — infinite number of paths

Independent Path Identification

Truth Table

- Can become unmanageable
- Useful as checklist for high-level code

Cyclomatic Complexity

- Generated as part of Cobertura report
- Bob Martin's "C.R.A.P." index

Independent Path Resolution

Refactor Mercilessly

Goal: one decision per method

High-level tests simply verify that lower-level methods were called

Exceptions

Most tests verify “happy path”

Need tests for likely exceptions

Understand boundary conditions

Not all exceptions can/should be tested

- Configuration exceptions from third-party libraries
- Exceptions that are rethrown or logged

Missing Features

"That test, I do not think it tests what you think it tests"

Tests should be tied to specifications

- Are any specifications sufficiently detailed?

What about TDD?

Formal Test Driven Development:

- Write minimal failing unit test
- Write minimal mainline code to pass this test
- Refactor and repeat

This guarantees 100% coverage!

Beware when removing duplicated code

- Coverage tools useful to find dead code
- Sometimes, different code does different stuff

What about Getters & Setters?

- Should be tested as part of normal use
  - Don’t write tests just to validate they work
- If not, why are they there?
  - Is your code accessing the underlying members?
  - Or is the data never used?

Code That Can't Be Tested

- Unreachable code
  - Example: default switch clause for enum
- Exceptions that require drastic effort to induce
  - Example: configuration exception for JDK XML parser
- Should such code even exist?

The Bottom Line

- Coverage tools are valuable
  - They clearly highlight code that isn't tested / used
- 100% coverage is unreasonable
  - Defensive coding may create unreachable code
- 100% coverage is not enough
  - Tests need to validate, not simply exercise

The Real Bottom Line

There's no substitute for a dedicated, thoughtful, test-infected programmer

For More Information

http://www.kdgregory.com/index.php?page=junit.coverage