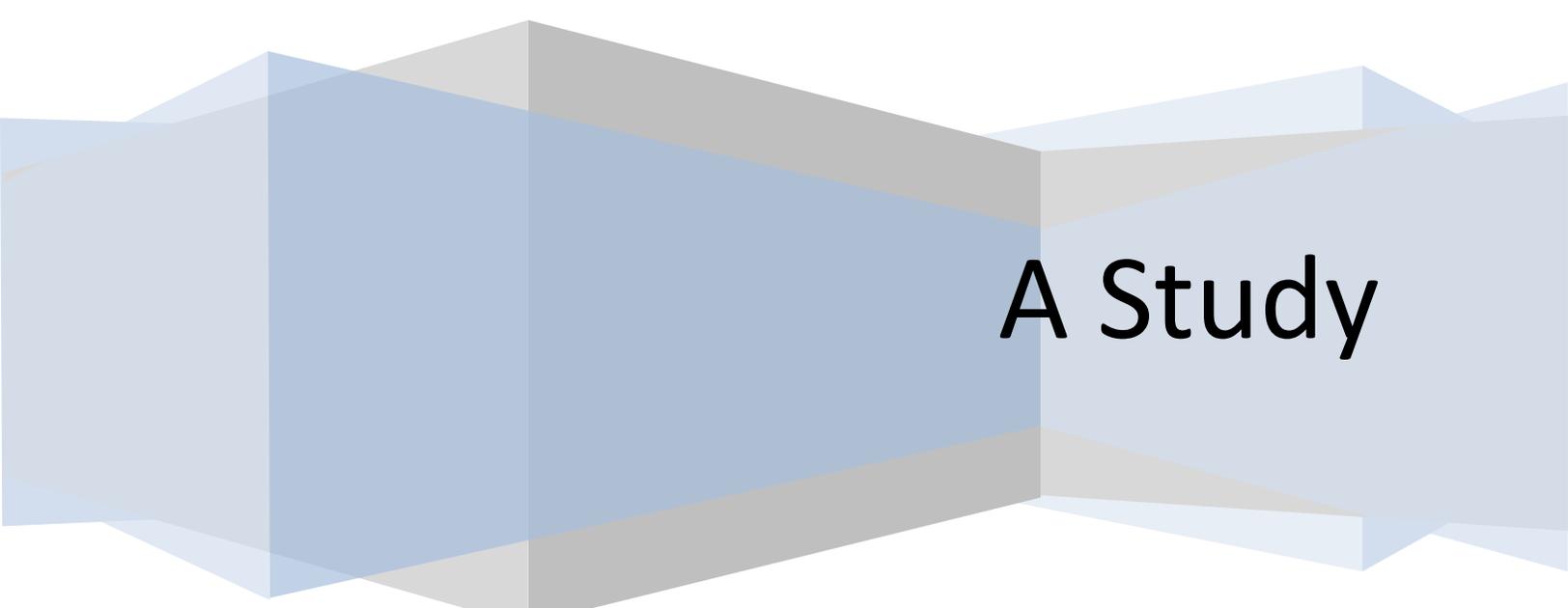


A Code Coverage Study

Code Coverage – Ensuring Quality

Benefits . Tooling Evaluations . Case Study.
Strategy

Vijayan Reddy, Nithya Jayachandran



A Study

Table of Contents

Table of Contents

1.0.	Introduction	3
2.0.	Why Code Coverage.....	3
3.0.	Benefits of Code Coverage.....	4
4.0.	Code Coverage Terminologies	4
4.1.	Instrumentation	4
4.2.	Merge.....	4
4.3.	Coverage Types	4
5.0.	Code Coverage Analysis	5
6.0.	What Code Coverage is and is Not.....	6
7.0.	Tooling Infrastructure	7
8.0.	Tool Deployment	8
8.1.1.	Cobertura : Integrated Instrumentation with the build process	9
8.1.2.	Cobertura : Deployment of Code Coverage instrumented application.....	9
8.1.3.	Cobertura : Auto-Collection of Coverage data during testing	10
8.1.4.	Cobertura : Merge & Final Report Generation	10
8.2.	NCover Deployment for .NET based Applications	12
9.0.	Tools Evaluation.....	12
10.0.	Some Popular Tools Reference.....	14
11.0.	Appendix.....	14

Table of Figures

<i>Figure 1 : Global Summary Report.....</i>	<i>5</i>
<i>Figure 2 : Package Summary Report.....</i>	<i>5</i>
<i>Figure 3 : Class Summary Report</i>	<i>5</i>
<i>Figure 4 : Class Detail Report.....</i>	<i>6</i>
<i>Figure 5 : Tool Deployment Workflow</i>	<i>7</i>
<i>Figure 7 : Tool Evaluation Parameters.....</i>	<i>13</i>
<i>Figure 8 : Popular Tool References</i>	<i>14</i>

1.0. Introduction

Code Coverage is an important measurement in Software Quality Engineering. While Software testing ensures correctness of the applications, a metric is required to track the completeness and effectiveness of the testing undertaken. Code Coverage helps achieve reliable quality through identifying untested areas of the application.

It is still a challenge to identify the right Code Coverage tooling solution. The next challenge lies in formulating the strategy for deployment of the tool and the process. This paper discusses Code Coverage tooling solutions and deployment strategy for quick benefits.

2.0. Why Code Coverage

Software testing is a challenging function. The testers need to ensure complete functional and non-functional correctness of the product. Considering the complex workflows and use cases of modern day applications, the number of unique cases that the software can be used often run into millions, which is not feasible to be covered under testing exercise. The testers thus need to

- While Planning Tests
 - o Ensure covering all workflows in terms of decision trees in the code
 - o Ensure covering all data values – by identifying patterns rather covering millions of values
- While testing
 - o Ensuring the testing is completely exercising the whole application with planned and exploratory tests.

At the end of testing, the decision to stop testing and release the product still remains subjective, based on the presence or absence of bugs, inflow of new bugs, success rate of each test cycle, confidence rating of the testers or users, etc. Whereas the definitive metric of quantifying how much of the application was really tested, is missed.

Code Coverage is measured as quantification of application code exercised by the testing activities. Code Coverage can be measured at various levels – in terms of programming language constructs – Packages, Classes, Methods, Branches or in terms of physical artifacts - Folders, Files and Lines.

For Eg. A Line Coverage metric of 67% means the testing exercised 67% of all executable statements of the application. A Code Coverage metric usually is accompanied by Code Coverage Analysis Report – which helps identify the un-tested part of the application code, thereby giving the testers early inputs for complete testing.

3.0. Benefits of Code Coverage

- Objective Indicator of Test Coverage of application code
 - Pointers to uncovered Packages / Classes / Methods / Branches
 - Pointers to uncovered Folders / Files / Lines
 - Drill down to untested part of source code and devise new tests
- Early Indicator for Testing Quality and Fixing it by adding new tests.
- Remove redundancy in testing
- Increased Confidence for Releases

4.0. Code Coverage Terminologies

4.1. Instrumentation

Instrumentation is the process to add code to the application to be able to output Code Coverage data. Instrumentation can be done at Source Level or at Intermediate language level, and rarely at run-time as information collection.

- **Source Level Instrumentation:** Prior to compilation, instrumentation code are inserted by the code coverage tool to the application. This will be compiled into the application code.
- **Object level Instrumentation:** Post compilation of the Application, executable lines to collect Coverage data are injected into the Object code.

4.2. Merge

Ability to run tests in batches, or in different environments, yet consolidate the overall coverage reports. Most of good coverage tools support offline merge feature.

4.3. Coverage Types

- **Folder Coverage**
- **File Coverage**
- **Lines Coverage**
- **Package Coverage**
- **Class Coverage**
- **Method Coverage**
- **Branch Coverage**

5.0. Code Coverage Analysis

Code Coverage Metrics in isolation are of limited help. But Code Coverage Report helps in analyzing the uncovered areas of code.

The following report tracks the code Coverage at Package level for a Java based application – The tool used below is Cobertura, and the Code Coverage report was on Cobertura’s testing.

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

Report generated by Cobertura 1.9 on 6/9/07 12:37 AM.

Figure 1 : Global Summary Report

Attached below is Code Coverage report at Class Level. The following report lists all the classes in a particular package, and tracks Branches & Lines Covered.

Coverage Report - net.sourceforge.cobertura.instrument

Package	# Classes	Line Coverage	Branch Coverage	Complexity
net.sourceforge.cobertura.instrument	10	90% 460/510	75% 123/164	1.854

Classes in this Package	Line Coverage	Branch Coverage	Complexity
JumpHolder	100% 6/6	N/A N/A	1
FirstPassMethodInstrumenter	100% 58/58	75% 15/20	1.353
Archive	100% 11/11	N/A N/A	1
ClassPattern	100% 24/24	79% 11/14	2
SwitchHolder	100% 5/5	N/A N/A	1
SecondPassMethodInstrumenter	98% 144/147	88% 23/26	1.346
ClassInstrumenter	94% 33/35	75% 12/16	1.875
NewLocalVariableMethodAdapter	93% 14/15	90% 9/10	1.167
Main	79% 157/198	69% 50/72	6.111
CoberturaFile	73% 8/11	50% 3/6	1.333

Report generated by Cobertura 1.9 on 6/9/07 12:37 AM.

Figure 2 : Package Summary Report

Detailed Class level report – Header below: This report states the percentage of lines covered -79% (75 of 95 lines) and branches covered.

Coverage Report - net.sourceforge.cobertura.ant.InstrumentTask

Classes in this File	Line Coverage	Branch Coverage	Complexity
InstrumentTask	79% 75/95	62% 20/32	2.357

Figure 3 : Class Summary Report

The following is an example of detailed code coverage report. The tool links to the source code – and reports the untested lines. The color scheme used is GREEN for hit code and RED for unhit code. The second column registers the number of times the code is visited.

The tester thus gets two sets of valuable inputs:

- 1) Track the un-tested code and create a new test case that exercises the code effectively.
- 2) Identify duplicate tests that visit the same part of code redundantly – thereby reducing the tests without compromising quality and reducing the test cycle times.

```

150     public void execute() throws BuildException
151     {
152 4         CommandLineBuilder builder = null;
153     try {
154 4             builder = new CommandLineBuilder();
155 4             if (dataFile != null)
156 4                 builder.addArg("--datafile", dataFile);
157 4             if (toDir != null)
158 4                 builder.addArg("--destination", toDir.getAbsolutePath());
159
160 5             for (int i = 0; i < ignoreRegexs.size(); i++) {
161 1                 Ignore ignoreRegex = (Ignore)ignoreRegexs.get(i);
162 1                 builder.addArg("--ignore", ignoreRegex.getRegex());
163             }
164
165 4             for (int i = 0; i < ignoreBranchesRegexs.size(); i++) {
166 0                 IgnoreBranches ignoreBranchesRegex = (IgnoreBranches)ignoreBranchesRegexs.get(i);
167 0                 builder.addArg("--ignoreBranches", ignoreBranchesRegex.getRegex());
168             }
169
170 6             for (int i = 0; i < includeClassesRegexs.size(); i++) {
171 2                 IncludeClasses includeClassesRegex = (IncludeClasses)includeClassesRegexs.get(i);
172 2                 builder.addArg("--includeClasses", includeClassesRegex.getRegex());
173             }
174
175 8             for (int i = 0; i < excludeClassesRegexs.size(); i++) {
176 4                 ExcludeClasses excludeClassesRegex = (ExcludeClasses)excludeClassesRegexs.get(i);
177 4                 builder.addArg("--excludeClasses", excludeClassesRegex.getRegex());
178             }
179
180 4             if (instrumentationClasspath != null) {
181 1                 processInstrumentationClasspath();
182             }
183 4             createArgumentsForFilesets(builder);
184
185 4             builder.saveArgs();
186 0         } catch (IOException ioe) {
187 0             getProject().log("Error creating commands file.", Project.MSG_ERR);
188 0             throw new BuildException("Unable to create the commands file.", ioe);
189 4         }
190
191         // Execute GPL licensed code in separate virtual machine
192 4         getJava().createArg().setValue("--commandfile");
193 4         getJava().createArg().setValue(builder.getCommandLineFile());
194 4         if (forkedJVMDebugPort != null && forkedJVMDebugPort.intValue() > 0) {
195 0             getJava().createJvarg().setValue("-Xdebug");
196 0             getJava().createJvarg().setValue("-Xrunjwp:transport=dt_socket,address=" + forkedJVMDebugPort + ",server=y,suspend=y");
197         }
198 4         AntUtil.transferCoberturaDataFileProperty(getJava());
199 4         if (getJava().executeJava() != 0) {
200 0             throw new BuildException(
201                 "Error instrumenting classes. See messages above.");

```

Figure 4 : Class Detail Report

6.0. What Code Coverage is and is Not

- 100% Code Coverage Does Not say the product is bug free, it ensures product is 100% tested. If the product was tested wrongly, Code Coverage can't help.
- Code Coverage testing does not require any special test methods – On the instrumented build – any testing that is carried out will generate coverage data. Regular using of the instrumented application will generate coverage information as well.

- Code Coverage is not about Whitebox testing. Code coverage is generated when you test the application in Any way. To analyse the code, the review happens at Code Level. It is not Whitebox testing.
- Code Coverage is not only through Unit Testing or Automated Testing.
- Code Coverage does not require extra testing efforts
- Code Coverage is not an end game activity, the earlier the better. It gives scope to improve tests and cover more code, thereby ensuring higher quality.
- Code Coverage instrumented builds can't be used for Performance Testing. It will add performance overhead.

7.0. Tooling Infrastructure

For Code Coverage analysis the ideal infrastructure would be

- Integrated Instrumentation with the build process
- Deployment of Code Coverage instrumented application
- Auto-Collection of Coverage data during testing
- Merge & Final Report automation

Code Coverage reports can be generated on every test cycle if tests.

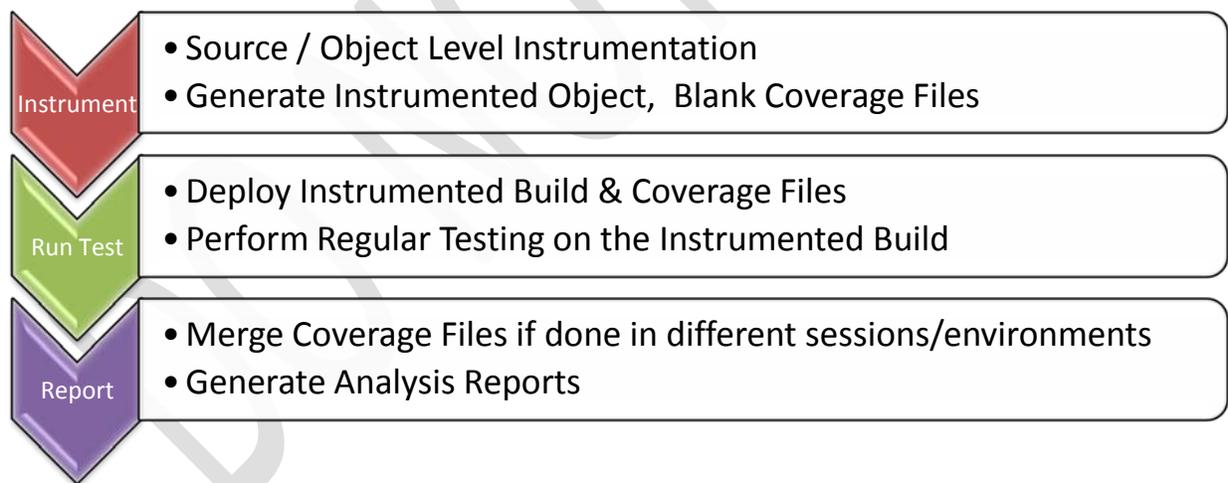


Figure 5 : Tool Deployment Workflow

8.0. Tool Deployment

To illustrate the usage of Code Coverage tools, two popular tools that are rich in features, ease of integration with existing build & test processes and documentation. These two tools are picked, one each from a leading enterprise application development technology.

- Cobertura for Java/J2EE
- NCover for .NET

The following section is not meant as User manual for any tool, but for illustrating the ease of use and integration with the build process, the usage is showcased. Please refer the product manuals for more details.

8.1. Cobertura Deployment for Java based Applications

We will study the Code Coverage Deployment using Cobertura for Java/J2EE applications. Cobertura offers excellent integration with build processes with Command line based executables, or with Cobertura ANT tasks. We will discuss both briefly.

In order to use Cobertura's command line executables, please add the Cobertura installation directory to the System path.

The ANT Snippets are used from Cobertura's online documentation for consistency for reference.

In order to use Cobertura's ANT tasks, we must do

- Add Cobertura.jar that ships with Cobertura tool, to the ANT's lib directories – or Add it in the class path reference variable in the build script.

Eg.

```
<property name="cobertura.dir" value="<<<SPECIFY COBERTURA INSTALL DIR>>>" />
<path id="cobertura.classpath">
  <fileset dir="{cobertura.dir}">
    <include name="cobertura.jar" />
    <include name="lib/**/*.*jar" />
  </fileset>
</path>
```

To be able to use the ANT Tasks, the following code snippet would help:

```
<taskdef classpathref="cobertura.classpath" resource="tasks.properties" />
```

8.1.1. Cobertura : Integrated Instrumentation with the build process

ANT Usage:

Cobertura supports a task “Cobertura-instrument” for the instrumentation process. The parameters supported are,

- datafile (Optional – defaults to ‘Cobertura.ser’ in the current directory. It is advised to set this explicitly to a convenient place that can be used to store and be picked for reporting)
- maxmemory (Optional – Good to set larger JVM max memory, if the instrumentation covers large number of classes)
- todir (Optional – To avoid the not instrumented classes being overwritten with the instrumented classes, it is good practice to specify output directory)

Eg for cobertura-instrument usage

Please note that it is a good practice to delete the Instrumentation serialized file (“Cobertura.ser” in the snippet below).

```
<delete file="cobertura.ser" />
<cobertura-instrument todir="${instrumented.dir}">
  <ignore regex="org.apache.log4j.*" />
  <fileset dir="${classes.dir}">
    <include name="**/*.class" />
    <exclude name="**/*Test.class" />
  </fileset>
  <fileset dir="${guiclasses.dir}">
    <include name="**/*.class" />
    <exclude name="**/*Test.class" />
  </fileset>
  <fileset dir="${jars.dir}">
    <include name="my-simple-plugin.jar" />
  </fileset>
</cobertura-instrument>
```

Commandline Usage:

The command-line usage for Cobertura is :

```
cobertura-instrument.bat [--basedir dir] [--datafile file] [--destination dir] [--ignore regex] classes [...]
```

Eg:

```
cobertura-instrument.bat --destination C:\MyProject\build\instrumented
C:\MyProject\build\classes
```

8.1.2. Cobertura : Deployment of Code Coverage instrumented application

After generation of the instrumented classes, pointed to by

- \${instrumented.dir} in the ANT task or
- --destination dir in the Commandline usage

Please deploy the 'instrumented application'. If the build process creates a JAR, WAR or a EAR file as part of the build process, or any other processing, the usage of originally compiled classes be replaced with the instrumented classes.

Rest of build and deployment process remains unchanged for either ANT or manual methods.

8.1.3. Cobertura : Auto-Collection of Coverage data during testing

While testing, please ensure the Cobertura classpath is set –

- If the tests are run through an ANT script, then add Cobertura classpath to the test task's classpath (Explained below)
- If the tests are run through batch file, command-line or any other mode, ensure that the current system path, and any classpath passed to the JVM as arg, has Cobertura classpath included, and the instrumented classes are in classpath before the un-instrumented classpath.

ANT Usage:

In case of ANT, the following snippets can be used in any task that runs the tests, the referred

- `fork="yes"` arg to be passed for ANT
- `<sysproperty key="net.sourceforge.cobertura.datafile" file="cobertura.ser" />`
(Please ensure the file reference is right, or provide the right path to this file)
- `<classpath location="{instrumented.dir}" />`
(The variable `instrumented.dir` is referred the way it is used in example snippets above)
- `<classpath refid="cobertura_classpath" />`
(The variable `Cobertura.classpath` is referred the way it is used in example snippets above)

8.1.4. Cobertura : Merge & Final Report Generation

Once the tests are completed, we may need to

- Merge the separate .SER files,
 - if we used different instrument tasks with separate .SER files for the application and we need to integrate them to a single report
 - If we used different copies of the same .SER file for different environments and need to merge for overall report
 - If we used different copies for the same .SER files for different kinds of testing and need to merge for the overall report
- Create the Report in HTML / XML

ANT Usage:

Cobertura-merge task is used to merge the different .SER files. It majorly requires two inputs –

- List of input .SER files, the folder and file name for each of them

- Where the output datafile to be created (defaults to “Cobertura.ser”, as illustrated in the example below where no datafile has been specified).

Eg.

```
<cobertura-merge>
  <fileset dir="{test.execution.dir}">
    <include name="server/cobertura.ser" />
    <include name="client/cobertura.ser" />
  </fileset>
</cobertura-merge>
```

Cobertura-report task is used for report generation. It accepts parameters such as,

- Datafile (Location & Name of the .SER file from which the coverage data is read)
- Dstidir (Where the generated report will be written to)
- Format (defaults to HTML)
- Srcdir (Where the source code for the instrumented classes are located – this is important for source linking from the coverage reports)
- Maxmemory (JVM param, optional, but recommended to set for larger value for large code bases).

Eg.

```
<cobertura-report format="html" destdir="{coveragereport.dir}" >
  <fileset dir="{src.dir}">
    <include name="**/*.java" />
    <exclude name="**/*Stub.java" />
  </fileset>
  <fileset dir="{guisrc.dir}">
    <include name="**/*.java" />
    <exclude name="**/*RB.java" />
  </fileset>
</cobertura-report>
```

Command-line Usage:

`cobertura-merge.bat [--datafile file] datafile [...]`

Eg.

```
cobertura-merge.bat --datafile C:\MyProject\build\cobertura.ser
C:\MyProject\testrundir\server\cobertura.ser
C:\MyProject\testrundir\client\cobertura.ser
```

`cobertura-report.bat [--datafile file] [--destination dir] [--format (html|xml)] source code directory [...] [-basedir dir file underneath basedir...]`

Eg.

```
cobertura-report.bat --format html --datafile C:\MyProject\build\cobertura.ser --
destination C:\MyProject\reports\coverage C:\MyProject\src
```

8.2. NCover Deployment for .NET based Applications

NCover supports integration with build process through MS Build & NAnt. The tasks are supported by the DLLs provided as part of NCover.

For using NCover NAnt Build tasks, the DLL should be included like,

```
<loadtasks assembly="NCoverExplorer.NAntTasks.dll"/>
```

For using NCover MSBuild Tasks, the inclusion should be done like,

```
UsingTask TaskName="NCoverExplorer.MSBuildTasks.NCoverExplorer"
  AssemblyFile =
  "C:\Program Files\NCover\Build Task Plugins\NCoverExplorer.MSBuildTasks.dll"/>
<UsingTask TaskName="NCoverExplorer.MSBuildTasks.NCover"
  AssemblyFile =
  "C:\Program Files\NCover\Build Task Plugins\NCoverExplorer.MSBuildTasks.dll"/>
<UsingTask TaskName="NCoverExplorer.MSBuildTasks.NUnitProject"
  AssemblyFile =
  "C:\Program Files\NCover\Build Task Plugins\NCoverExplorer.MSBuildTasks.dll"/>
```

More information along the usage and deployment of NCover using MS Build and NAnt is available in the Online documentation for NCover.

9.0. Tools Evaluation

To evaluate good Code Coverage tools, one would require listing the evaluation parameters and rating the tools on these. With that in mind, this paper lists major requirements for Code Coverage tools. These need to be diligently checked and applied as relevant.

Parameter	Weightage	Notes
Coverage Levels		
Package /Namespace Level Coverage	2	Java – Package, .NET –NameSpace
Class Level Coverage	2	
Method level Coverage	3	
Block Coverage	3	
Line Coverage	3	
File Level Coverage	2	
Report Clarity		
Source Linking	3	Ability to link Coverage report and Source files.
Hit Count	3	No. of times the statement/code block has been hit.
Exclusion Management		
Being able to exclude certain areas of code from reporting		
Exclusion Provision	3	
Namespace/Package level Exclusion	2	
Class Level Exclusion	2	
Method Level Exclusion	3	
Line Level Exclusion	2	

Exclusion Patterns Definition	1	
Source File Exclusion	2	
Exclusion Reporting & Validation	2	
Advanced Reporting		Exportable Reports
HTML reports	2	
Spreadsheet Export	2	
Baselining & Versioning	2	To track coverage for only the newly added code
Incremental Reporting	2	
Platform & Coverage		
GUI Support	3	
Command Line / Batch Mode Support	1	
Windows Vista Support	2	
Windows - 64 bit support	2	
Support for Build Script	2	
Child Process Coverage	3	
Stand Alone .NET apps Support	3	
IIS Hosted ASP .NET apps	1	
Windows Service Apps	3	
Licensing & Logistics		
Detailed Documentation	3	
Open Source	2	
Commercial	2	
Paid Support	3	
Community based Free Support	3	
Technical Aspects		
Source Level Instrumentation	3	
Object Level Instrumentation	2	
Auto Saving Frequency	3	
Performance Increase	1	
Compilation Time Increase	3	
Merging	3	
Standalone Instrumentation independent of build	2	
Code Size Increase	1	

Figure 6 : Tool Evaluation Parameters

10.0. Some Popular Tools Reference

S.No	Technology	Tool Names
1	Java	Emma, Cobertura
2	.NET	NCover, PartCover
3	C/C++	BullsEye, CoverageMeter
4	.NET & C++	DevPartner, Rational PureCoverage etc.
5	Flex	FlexCover

Figure 7 : Popular Tool References

11.0. Appendix

1. http://en.wikipedia.org/wiki/Code_coverage : Has very limited information – has list of some tooling solutions
2. <http://www.bullseye.com/coverage.html>
3. <http://www.cenqua.com/clover/doc/coverage/intro.html>
4. <http://cobertura.sourceforge.net/anttaskreference.html>
5. <http://www.ncover.com/documentation/buildtasks>