Journal of Theoretical and Applied Information Technology

www.jatit.org

MEASURING THE EFFECTIVENESS OF OPEN COVERAGE BASED TESTING TOOLS

MS. L.SHANMUGA PRIYA, MS.A.ASKARUNISA, DR. N.RAMARAJ
Thiagarajar College of Engineering, Affiliated to Anna University, Chennai.
E-Mail: nishanazer@yahoo.com

ABSTRACT

The levels of quality, maintainability, testability, and stability of software can be improved and measured through the use of automated testing tools throughout the software development process. Automated testing tools assist software engineers to gauge the quality of software by automating the mechanical aspects of the software-testing task. Automated testing tools vary in their underlying approach, quality, and ease-of-use, among other characteristics In Software testing; Software Metrics provide information to support a quantitative managerial decision-making for the test managers. Among the various metrics, Code coverage metric is considered as the most important metric often used in analysis of software projects in the industries for testing, Code coverage analysis also helps in the testing process by finding the areas of a program not exercised by a set of test cases, creating additional test cases to increase coverage, and determine the quantitative measure of the code, which is an indirect measure of quality. The test manager needs coverage metric in making decisions while selecting test cases for regression testing. In literature there are a large number of automated tools to find the coverage of test cases in Java. Choosing an appropriate tool for the application to be tested may be a complicated process for the test Manager. To ease the job of the Test manager in selecting an appropriate tool, we propose a suite of objective metrics for measuring tool characteristics as an aid in systematically evaluating and selecting automated testing tools.

Keywords
Software testing, Software Metrics, Code Coverage, automated tools.

1. INTRODUCTION

" Software Testing is a process to detect the defects and minimize the risk associated with the residual defects of a software".[3]. A test case tests the response of a single method to a particular set of inputs. Test case is a combination of inputs, executing function and expected output. A test suite is a collection of test cases. Automated testing tools assist software engineers to gauge the quality of software by automating the mechanical aspects of the software-testing task. Automated testing tools vary in their underlying approach, quality, and ease-of-use, among other characteristics. In addition, the selection of testing tools needs to be predicated on characteristics of the software component to be tested. But how does a project manager choose the best suite of testing tools for testing a particular software component?

In this paper we propose a suite of objective metrics for measuring tool characteristics, as an aid for systematically evaluating and selecting the automated testing tools that would be most appropriate for testing the system or component under test. Our suite of metrics is also intended to be used to monitor and gauge the effectiveness of specific combinations of testing tools during software development. In addition, the suite of test-tool metrics is to be used in conjunction with existing and future guidelines for conducting tool evaluation and selection. In December 1991, a working group of software developers and tool users completed the Reference Model for Computing System-Tool Interconnections (MCSTI), known as IEEE Standard 1175[15]. As an offshoot of their work, they also introduced a tool-evaluation system. The system implements a set of forms which systematically guide users in gathering, organizing, and analyzing information on testing and other types of tools for developing and maintaining software. The user can view tool-dependent factors such as performance, user friendliness, and reliability, in addition to environment-dependent factors such as the cost of the tool, the tool's effect on organizational policy and procedures, and tool interaction with existing hardware and software assets of an

organization. The data forms also facilitate the preference weighting, rating, and summarizing selection criteria. The process model underlying the MCSTI consists of five steps: analyzing user needs, establishing selection criteria, tool search, tool selection, and reevaluation.

Software metrics are quantitative standards of measurement for various aspects of software projects. These Metrics help, track aspects of an ongoing software project, such as changing requirements, rates of finding and fixing defects, and growth in size and complexity of code. From the testing point of view, the metrics typically focus on the quantity and quality of any software

In Section2 the importance of software metrics and types of metrics are discussed .Section3 describes the prior work on metrics. Section4 includes the suite of metrics for testing tools In section5 the importance of coverage metrics is discussed. In section6 various types of code coverage tools are discussed.Section7 provides the detailed description of the selected code coverage tools for our analysis. Section8 provides the experimental details of programs.Section9 provide the experimental details of the selected tools. In section10 the results are analyzed.

2. SOFTWARE METRICS

Software metrics is defined as the current state of art in the measurement of software products and process [9]. Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined unambiguous rules. Metrics strongly support software project management activities mainly test management. They relate to the four functions of management as follows:

i. Planning - Metrics serve as a basis of cost estimating, training planning, and resource planning, scheduling, and budgeting.
ii. Organizing - Size and schedule metrics influence a project's organization.
iii. Controlling - Metrics are used to status and track software development activities for compliance to plans.
iv. Improving - Metrics are used as a tool for process improvement and to identify where improvement efforts should be concentrated and measure the effects of process improvement efforts.

2.1 Importance of Metrics

Deriving metrics in every phase of the SDLC has a major importance through out the life cycle of Software for management, Managers, Developers and Customers[4].

2.1.1 Metrics in Project Management

1. Metrics make the project's status visible. Managers can measure progress to discover if a project is on schedule or not.
2. Metrics focus on activity. Workers respond to objectives, and metrics provide a direct objective for improvement.
3. Metrics help to set realistic expectations. By assisting in estimation of the time and resources required for a project, metrics help managers set achievable targets.
4. Metrics lay the foundations for long-term improvement. By keeping records of what happens on various projects, the beneficial activities can be identified and encouraged, while the detrimental ones are rejected.
5. Metrics also help the management in reducing various resources namely people, time and cost in every phase of SDLC.

2.1.2 Metrics in Decision Making

Metrics will not drive a return on investment unless managers use them for decision making. Some decisions in which software metrics can play a role include

Product readiness to ship/deploy,
Cost and schedule for a custom project,
How much contingency to include in cost and schedule estimates?
Where to invest for the biggest payback in process improvement, and
When to begin user training.

Managers should demand supporting metrics data before making decisions such as these. For example, they can use fault-arrival-and-close-rate data when deciding readiness to deploy. Knowing the Overall project risk through metrics can help managers decide how much contingency to include in cost and schedule estimates.

2.2 Procedural (Traditional) Software Metrics

Support decision making by management and enhance return on the IT

investment [5]. Once business goals have been identified, the next step is to select metrics that support them. Various types of Metrics [10] found in literature are:

Lines of Code:

- Total Lines of Count (TLOC)
- Executable Lines of Count (ELOC)
- Comment Lines of Count (CLOC).

Hallstead's Metrics:

- Program Length (N)
- Program Volume(V)
- Effort to Implement(E)
- Time to Implement (T)
- Number of Delivered Bugs (B).

Function point (FP) [16] is a metric that may be applied independent of a specific programming language, in fact, it can be determined in the design stage prior to the commencement of writing the program. To determine *FP*, an Unadjusted Function Point Count (UFC) is calculated. *UFC* is found by counting the number of external inputs (user input), external outputs (program output), external inquiries (interactive inputs requiring a response), external files (inter-system interface), and internal files (system logical master files). Each member of the above five groups is analyzed as having either simple, average or complex complexity, and a weight is associated with that member based upon a table of *FP* complexity weights. *UFC* is then calculated

$UFC = \sum 1\text{->}15$ (number of items of variety i) x (weight of i) (1)

Next, a Technical Complexity Factor (TCF) is determined by analyzing fourteen contributing factors. Each factor is assigned a score from zero to five based on its criticality to the system being built. The *TCF* is then found through the equation:

$$TCF = 0.65 + 0.01 \sum_{1\text{->}14} F_i \qquad (2)$$

where *FP* is the product of *UFC* and *TCF*. *FP* has been criticized due to its reliance upon subjective ratings and its foundation on early design characteristics that are likely to change as the development process progresses.

2.3 Object Oriented Metrics:

The most commonly cited software metrics to be computed for software with an object-oriented design are those proposed by Chidamber and Kemerer [16],[17]. Their suite of metrics consists of the following metrics: weighted methods per class, depth of inheritance tree, number of children, coupling between object classes, response for a class, and lack of cohesion in methods.

- Weighted Methods Per Class (WMC)
- Response For a class(RFC)
- Lack Of Cohesion Of Methods (LCOM)
- Coupling Between Object Methods (CBO)
- Depth Of Inheritance Tree (DIT)
- Number Of Children (NOC)

McCabe's Metrics:

- Cyclomatic Complexity
- Essential Complexity
- Actual Complexity.

Lorenz and Kidd [18] proposed another set of object-oriented software quality metrics. Their suite includes the following:

- Number of scenarios scripts (use cases) (NSS)
- Number of key classes (NKC)
- Number of support classes
- Average number of support classes per key class (ANSC)
- Number of subsystems (NSUB)
- Class size (CS)
- Total number of operations + number of attributes
- Both include inherited features
- Number of operations overridden by subclass (NOO)
- Number of operations added by a subclass (NOA)
- Specialization index (SI)
- SI = [NOO x level] / [Total class method]
- Average method size
- Average number of methods
- Average number of instance variables
- Class hierarchy nesting level

All the above metrics can be easily calculated once the code is available. It just gives a measure that is not completely used to find the effectiveness of testing but can be useful in other phases of the SDLC. The metrics that are mostly helpful for the test managers or during testing are Code Coverage Metric[16], Test Case Execution

Time metric, Test Case Fault detection Capability metric etc.

2.4 Coverage Metrics:

To measure how well the program is exercised by a test suite, coverage metrics are used. [4, 5] There exists a number of coverage metrics in literature. Following are descriptions of some types of coverage metrics.

Statement Coverage

This metric is defined as"The percentage of executable statements in a component that have been exercised by a test case suite."

Branch Coverage

This metric is defined as"The percentage of branches in a component that have been exercised by a test suite."

Loop Coverage

This metric is defined as"The percentage of loops in a component that have been exercised by a test suite."

Decision Coverage

This metric is defined as"The percentage of Boolean expressions in a component that have been exercised by a test suite." [6]

Condition Coverage

This metric is defined as"The percentage of decisions in a component that have been exercised by a test suite." [7]

Function Coverage

This metric is defined as"The percentage of functions in a component that have been exercised by a test suite."

Path Coverage

This metric is defined as"The percentage of paths in a component that have been exercised by a test suite." [8]

Entry/Exit Coverage

This metric is defined as"The percentage of call and return of the function in a component that have been exercised by a test suite."

Requirements Coverage

This metric is defined as "The percentage of requirements in a component that have been covered by a test case suite."

3. PRIOR WORK ON METRICS FOR SOFTWARE-TESTING TOOLS

The Institute for Defense Analyses (IDA) published two survey reports on tools for testing software [19],[20]. Although the tool descriptions contained in those reports are dated, the analyses provide a historical frame of reference for the recent advances in testing tools and identify a large number of measurements that may be used in assessing testing tools. For each tool, the report details different types of analysis conducted, the capabilities within those analysis categories, operating environment requirements, tool-interaction features, along with generic tool information such as price, graphical support, and the number of users.

The research conducted at IDA was intended to provide guidance to the U.S. Department of Defense on how to evaluate and select software-testing tools. The major conclusions of the study were that:

- Test management tools offer critical support for planning tests and monitoring test progress.
- Problem reporting tools offered support for test management by providing insight software products' status and development progress.
- Available static analysis tools of the time were limited to facilitating program understanding and assessing characteristics of software quality.
- Static analysis tools provided only minimal support for guiding dynamic testing.
- Many needed dynamic analysis capabilities were not commonly available.
- Tools were available that offered considerable support for dynamic testing to increase confidence in correct software operation.
- Most importantly, they determined that the range of capabilities of the tools and the tools' immaturity required careful analysis prior to selection and adoption of a specific tool.

The Software Technology Support Center (STSC) at Hill AFB works with Air Force software organizations to identify, evaluate and adopt technologies to improve product quality, increase production efficiency and schedule prediction ability [21]. Section four of their

report discusses several issues that should be addressed when evaluating testing tools and provides a sample tool-scoring matrix. Current product critiques and tool-evaluation metrics and other information can be obtained by contacting them through their website at http://www.stsc.hill.af.mil/SWTesting/.

4. PROPOSED SUITE OF METRICS FOR EVALUATING AND SELECTING SOFTWARE-TESTING TOOLS

Weyuker identified nine properties that complexity measures should possess [13]. Several of these properties can be applied to other metrics too; these characteristics were considered in our formulation of metrics for evaluating and selecting software testing tools.
Our suite of metrics for evaluating and selecting software testing tools has the following properties: the metrics exhibit non-coarseness in that they provide different values when applied to different testing tools; the metrics are finite in that there are a finite number of tools for which the metrics' results in an equal value, yet they are non-unique in that a metric may provide the same value when applied to different tools; and the metrics are designed to have an objective means of assessment rather than being based on subjective opinions of the evaluator.

4.1. Metrics for Tools that Support Testing of Procedural Software

These metrics are applied to the testing tool in its entirety via a specific function performed by the tool.

4.1.1. Human Interface Design (HID). All automated testing tools require the tester to set configurations prior to the commencement of testing. Tools with well-designed human interfaces enable easy, efficient, and accurate setting of tool configuration.
Factors that lead to difficult, inefficient, and inaccurate
human input include multiple switching between keyboard and mouse input, requiring large amount of keyboard input overall, and individual input fields that require long strings of input. *HID* also accounts for easy recognition of the functionality of provided shortcut buttons.

$$HID = KMS + IFPF + ALIF + (100 - BR) \quad (3)$$

where *KMS* is the average number of keyboard to mouse switches per function, *IFPF* is the average number of input fields per function, *ALIF* is the average string length of input fields, *BR* is the percentage of buttons whose functions were identified via inspection. A large *HID* indicates the level of difficulty to learn the tool's procedures on purchase and the likelihood of errors in using the tool over a long period of time. *HID* can be reduced by designing input functions to take advantage of current configurations as well as using input to recent fields as default in applicable follow on input fields. For example, if a tool requires several directories to be identified, subsequent directory path input fields could be automatically completed with previously used paths. This would require the tester to only modify the final subfolder as required than reentering lengthy directory paths multiple times.

4.1.2. Maturity & Customer Base (MCB). There are several providers of automated testing tools for the business of software testers. These providers have a wide range of experience in developing software-testing tools. Tools that have achieved considerable maturity typically do so as a result of customer satisfaction in the tool's ability to adequately test their software. This satisfaction leads to referrals to other users of testing tools and an increase in the tool's customer base.

$$MCB = M + CB + P \quad (4)$$

where *M* (maturity) is the number of years tool (and its previous versions) have been applied in real world applications, *CB* (customer base) is the number of customers who have more than one year of experience applying the tool, and *P* (projects) is the number of previous projects of similar size that used the tool Care must be taken in evaluating maturity to ensure the tool's current version does not depart too far from the vendor's previous successful path. Customer base and projects are difficult to evaluate without relying upon information from a vendor who has a vested interest in the outcome of the measurement.

4.1.3. Tool Management (TM). As software projects become larger and more complex, large teams are used to design, encode, and test the software. Automated testing tools should provide for several users to access the information while ensuring proper management of the information.

Possible methods may include automated generation of reports to inform other testers on outcome of current tests, and different levels of access (e.g., read results, add test cases, and modify/remove test cases).

$$TM = AL + ICM \quad (5)$$

where *AL* (access levels) is the number of different access levels to tool information, and *ICM* (information control methods) is the sum of the different methods of controlling tool and test information.

4.1.4. Ease of Use (EU). A testing tool must be easy to use to ensure timely, adequate, and continual integration into the software development process. Ease of use accounts for the following: learning time of first-time users, retainability of procedural knowledge for frequent and casual users, and operational time of frequent and casual users.

$$EU = LTFU + RFU + RCU + OTFU + OFCU \quad (6)$$

where *LTFU* is the learning time for first users, *RFU* is the retainability of procedure knowledge for frequent users, *RCU* is the retainability of procedure knowledge for casual users, *OTFU* is the average operational time for frequent users, and *OTCU* is the average operational time for casual users.

4.1.5. User Control (UC). Automated testing tools that provide users expansive control over tool operations enable testers to effectively and efficiently test those portions of the program that are considered to have a higher level of criticality, have insufficient coverage, or meet other criteria determined by the tester. *UC* is defined as the summation of the different portions and combinations of portions that can be tested. A tool that tests only an entire executable program would receive a low *UC* value. Tools that permit the tester to identify which portions of the executable will be evaluated by tester-specified test scenarios would earn a higher *UC* value. Tools that will be implemented by testing teams conducting a significant amount of regression testing should have a high *UC* value to avoid retesting of unchanged portions of code.

4.1.6 Test Case Generation (TCG). The ability to automatically generate and readily modify test cases is desirable. Testing tools which can automatically generate test cases based on parsing the software under test are much more desirable that tools that require testers to generate their own test cases or provide significant input for tool generation of test cases. Availability of functions to create new test cases based on modification to automatically generated test cases greatly increases the tester's ability to observe program behavior under different operating conditions.

$$TCG = ATG + TRF \quad (7)$$

where *ATG* is the level of automated test case generation as defined by:

10: fully automated generation of test cases
8: tester provides tool with parameter names & types via user-friendly methods (i.e., pull down menus)
6: tester provides tool with parameter names & types
4: tester must provide tool with parameter names, types and range of values via user-friendly methods
2: tester must provide tool with parameter names, types and range of values
0: tester must generate test cases by hand

and *TRF* is the level of test case reuse functionality:

10: test cases may be modified by user friendly methods (i.e. pull down menus on each test case parameter) and saved as a new test case
8: test cases may be modified and saved as a new test case
6: test cases may be modified by user friendly methods but cannot be saved as new test cases
4: test cases may be modified but cannot be saved as new test cases
0: test cases cannot be modified

4.1.7. Tool Support (TS). The level of tool support is important to ensure efficient implementation of the testing tool, but it is difficult to objectively measure. Technical support should be available to testers at all times testing is being conducted, including outside traditional weekday working hours. This is especially important for the extensive amount of testing frequently
conducted just prior to product release. Technical support includes help desks available telephonically or via email, and on-line users' groups monitored by vendor technical support staff. Additionally, the availability of tool

documentation that is well organized, indexed, and searchable is of great benefit to users.

$$TS = ART + ARTAH + ATSD - DI \quad (8)$$

where *ART* is the average response time during scheduled testing schedule, *ARTAH* is the average response time outside scheduled testing schedule, *ATSD* is the average time to search documentation for desired information, and *DI* is the documentation inadequacy measured as the number of unsuccessful searches of documentation.

4.1.8. Estimated Return on Investment (EROI). A study conducted by the Quality Assurance Institute involving 1,750 test cases and 700 errors has shown that automated testing can reduce time requirements for nearly every testing stage and reduces overall testing time by approximately 75% [14]. Vendors may also be able to provide similar statistics for their customers
currently using their tools.

$$EROI = (EPG \times ETT \times ACTH) + EII - ETIC + (EQC \times EHCS \times ACCS) \quad (9)$$

where *EPG* is the Estimated Productivity Gain, *ETT* is the Estimated Testing Time without tool, *ACTH* is the Average Cost of One Testing Hour, *EII* is the Estimated Income Increase, *ETIC* is the Estimated Tool Implementation Cost, *EQC* is the Estimated Quality Gain, *EHCS* is the Estimated Hours of Customer Support per Project, and *ACCS* is the Average Cost of One Hour of Customer Support.

4.1.9. Reliability (Rel). Tool reliability is defined as the average mean time between failures.

4.1.10. Maximum Number of Classes (MNC). Maximum number of classes that may be included in a tool's testing project.

4.1.11. Maximum Number of Parameters (MNP).
Maximum number of parameters that may be included in a tool's testing project.

4.1.12. Response Time (RT). Amount of time used to apply test case on specified size of software. RT is difficult to measure due to the varying complexity of different programs of the same size.

4.1.13. Features Support (FS). Count of the following features:

- Extendable: tester can write functions that expand provided functions
- Database available: open database for use by testers
- Integrates with software development tools
- Provides summary reports of findings.

5. IMPORTANCE OF COVERAGE METRICS FOR TESTING

There are a large number of metrics for software applications with respect to testing viz. quality metrics, reliability metrics etc.. Among all the metric's, coverage metric plays a vital role in selecting best test cases that reduces most of the resources required for software regression testing. Coverage metric describes a fraction of lines of code "covered" by a test case. Testing with measurement and tracking of test coverage can motivate development of additional test cases that will typically drive test coverage to 90% or more of the code. As a result, more of the faults in the code will be discovered by testers rather than by users. Fixing these faults prior to deployment can dramatically improve the quality of installed software and reduce software support costs.

5.1 Code Coverage Analysis

Code coverage metrics are amongst the first techniques invented for systematic software testing. Code coverage analysis is the process of

1. Finding areas of a program not exercised by a set of test cases,
2. Creating additional test cases to increase coverage, and
3. Determining a quantitative measure of code coverage, an indirect measure of quality.
4. Identifying redundant test cases that do not increase coverage.

Coverage analysis is used to assure quality to a set of tests, not the quality of the actual product. Coverage analysis requires access to test program source code and often requires recompiling it with a special command [1][2]. Establish a minimum percentage of coverage, to determine when to stop analyzing coverage.

6. SURVEY OF VARIOUS COVERAGE TOOLS

In literature, variety of tools are available to perform code coverage analysis. Following are a description of some Java-based coverage testing tools.

JCover

JCover [11] is a free code-coverage tool. For Java programmers that allow to measure the effectiveness of their Java tests and how much of a software program's code has been tested.

Code Cover

Code Cover [16] is a free code-coverage tool for Java programmers that provide several ways to increase test quality. It shows the quality of our test suite and helps to develop new test cases and rearrange test cases to save some of them.

Cobertura

Cobertura [11] is a free Java tool that calculates the percentage of code accessed by tests. It can be used to identify which parts of our Java program are lacking test coverage.

JBlanket

JBlanket [11] is a method coverage tool for stand-alone and client-server Java programs.

Quilt

Quilt [11] is a Java software development tool that measures coverage, the extent to which unit testing exercises the software under test.

Emma

Emma [14] is an open-source toolkit for measuring and reporting Java code coverage. Emma is so lightweight; developers can use it during the process of writing tests instead of waiting for a "test build".

NoUnit

NoUnit [11] allows us to see how good our JUnit tests are.

InsECT

InsECT [11] which stands for Instrumentation Execution Coverage Tool, is a system developed in Java to obtain coverage information for Java programs.

Hansel

Hansel [11] decorates a JUnit Test class and instruments one or more classes under test to verify 100% branch coverage of the tested classes by the Test class.

GroboCodeCoverage

GroboCodeCoverage [11] is a 100% Pure Java implementation of a Code Coverage tool. It uses Jakarta's BCEL platform to post-compile class files to add logging statements for tracking coverage.

Jester

Jester [11] finds code that is not covered by tests. Jester makes some change to the code, runs tests, and if the tests pass Jester displays a message saying what it changed.

DJUnit

DJUnit [11] is a JUnit test runner, which generates coverage report and allows virtual mock objects. It integrates with Eclipse and Ant.

Gretel

Gretel [13] is a test coverage monitoring tool for Java programs. The current version provides statement coverage monitoring.

Clover

Clover [11] is a low cost code coverage tool for Java. It is tightly integrated with the popular Ant build tool.

Koalog Code Coverage

Koalog Code Coverage [11] is a code coverage computation application written in the Java programming language.

7. FOUR TOOLS SELECTED FOR USE IN VALIDATING THE PROPOSED SUITE OF METRICS

To validate our proposed suite of metrics for evaluating and selecting software testing tools, we have selected four software Code Coverage tools against which to apply the proposed metrics. We describe and discuss the setup of each tool for validation purposes and discuss problems encountered in exercising the tools. We have considered four different java based code coverage tools viz. JCover, Emma, Gretel and Code Cover.

7.1 Why JCover, Emma, Gretel and Code Cover?

These tools have been chosen as a code coverage tool of choice because:

1.These tools are 100% open-source Java tools.

2.These tools have a large market share compared with the other open source coverage tools.

3.These tools are user friendly.

4.These have multiple report type formats.

5.These tools are for both open-source and commercial development projects.

7.2 JCOVER Tool

JCOVER [11] identifies how many times each line of code in the application has been executed and can see which parts of the software remain untested. After instrumenting the code and running the tests, a report is generated allowing to view information coverage figures from a project level right down to the individual line of code. JCover works by modifying Java classes at the byte code level. When the modified classes are executed, during a test-run for instance, data is collected that identifies how many times each line of code has been executed.

Features

- Measures the coverage percentage of code that has been tested.
- Run the tests; create reports get all information about problems in the system.
- Set-up test in isolation using mock-object technologies to reduce reliance on extensive end-to-end per system tests.
- A commercial JCover plug-in is available for Eclipse 3.[15]

7.3 EMMA Tool

EMMA [14] is an open-source toolkit for measuring and reporting Java code coverage. Emma distinguishes itself from other tools by going after a unique feature combination: support for large-scale enterprise software development while keeping individual developer's work fast and iterative. Emma is a tool for measuring coverage of Java software. Such a tool is essential for detecting dead code and verifying which parts of an application are actually exercised by the test suite and interactive use. .Emma differs from other coverage tools in its extreme orientation towards fast iterative develop-test style of writing software. Following are the steps involved in starting the Emma Tool

- Adding Emma command line tools in the class path.
- Instrumenting the java classes
- Execution of java classes

Features

- Emma can instrument classes for coverage either offline (before they are loaded) or on the fly (using an instrumenting application class loader).
- Supported coverage types: class, method, line, basic block. EMMA can detect when a single source code line is covered only partially.
- Output report types: plain text, HTML, XML.

7.4 GRETEL Tool

GRETEL [13] is an Open-Source Residual Test Coverage Tool. Gretel is a test coverage monitoring tool for Java programs. The current version provides statement coverage monitoring (identifying which lines of Java have been executed, and which have not been touched by testing). The primary difference between Gretel and other coverage monitoring tools is that Gretel implements residual test coverage monitoring. After one run a program that has been instrumented with Gretel, Gretel can re-instrument the program and remove instrumentation for those parts that have already been executed [13].

The main steps in using GRETEL are:

1. **Initial instrumentation** - Prepares a program to record which parts have been executed.
2. **Runs an application** - Instrumented application stores some information at the conclusion of each run.
3. **Interpreting results** - Gretel provides visualization of source code, indicating which parts have been executed.
4. **Reinstrumentation** - This step is optional, but it is the main way that Gretel differs from other test instrumentation packages.
5. **Remove all instrumentation** - This step is also optional.

7.5 CODECOVER Tool

Codec Cover [12] is an extensible open source code coverage tool. Code cover provides several ways to increase test quality. It shows the quality of test suite and helps to develop new test cases

and rearrange test cases to save some of them. So we get a higher quality and a better test productivity.

Features

- Supports statement coverage, branch coverage, loop coverage and strict condition coverage
- Performs source instrumentation for the most accurate coverage measurement.
- CLI interface, for easy use from the command line.
- Ant interface, for easy integration into an existing build process.
- Correlation Matrix to find redundant test cases and optimize your test suite.
- The source code is highlighted according to the measured data.

Steps for execution of CODE COVER

To generate the coverage report using Code cover tool the following three steps can be followed.

1. Selecting the files to instrument
2. Enabling Code Cover for a Java project
3. Running a Java project with Code Cover

Coverage can be viewed in the following ways using code cover

Test Sessions View

This view displays the test sessions and test cases in a test session container.

Coverage View

In this view we can see the coverage of individual parts of the SUT. Every metric has its own column.

Correlation View

This view is used to compare test cases with each other.

8. EXPERIMENTAL PROGRAMS USED FOR VALIDATION PROCESS

The validation experiments conducted were performed on java programs which computes various types of sorting algorithms like Bubble Sort, Quick Sort, insertion Sort, Heap Sort, Merge sort Selection Sort, Payroll calculation, Calculator application and Code of Java compiler(only arrays were considered). The details of the above programs are shown in table 1 as follows.

programs	LOC	NOC	NOM	CC	No. of Test cases
Bubble	51	1	3	4	7
Selection	52	1	3	4	7
Insertion	54	1	3	4	8
Heap	75	1	8	12	16
Merge	68	1	5	11	14
Quick	70	1	5	11	15
payroll	320	2	10	27	29
calculator	536	1	15	55	58
Arrays of Javac	1360	1	79	250	255

Table 1Experimental program details
LOC- Lines of code, NOM-No. of Methods NOC-No.of classes, CC-Cyclomatic complexity

We have developed 67 test cases and added 10 more test cases to test all the sorting programs, 29 test cases
for payroll program, 58 test cases for calculator program and 255 test cases for Arrays program.

9. EXERCISING THE SOFTWARE TESTING TOOLS:

The selected tools are analyzed as shown in Table 2 , based on Coverage measures that the tool supports like, Memory space, Graphical Representation, User Interface, Residual coverage Monitoring and reports in HTML format. While considering Coverage measures supported feature, Statement coverage is obtained by all the four tools.

Among the four tools, the File Coverage (FC) can be computed only by the JCover tool, the Block Coverage (BLC) could be found only by Emma Tool and the Condition Coverage (CC) and Loop Coverage (LC) found only by the Code Cover tool.

The Branch Coverage (BC), the Method Coverage (MC) and the Class Coverage (CLC) are found by JCover- Code Cover, JCover-Emma and JCover-Emma respectively. It is only the Statement Coverage (SC) that is found by all the selected tools.

9.1 Computation of Metrics

During the application of the four testing-tool suites on the various software programs, measurements were taken to calculate the testing-tool metrics

9.1.1 Human-Interface Design. To calculate the human-interface design (HID) metric, measurements were taken during three

operations: establishing test project, conducting test project, and viewing testing results. While conducting the operations with the JCover tool, there were six occasions that required the user to transfer from the keyboard to the mouse or vice versa. Dividing this number by the number of operations (three) results in an average of two keyboard-to-mouse switches (KMS). There were five input fields resulting in five average input fields per functions (IFPF).

Three of the input fields required only mouse clicks and one required entry of strings totaling twenty two characters. The average length of input fields (ALIF) was calculated by dividing the sum of these inputs by the number of input fields resulting in an ALIF of six. In attempting to identify the functions of sixteen buttons, eleven were identified correctly. The percentage of 68.75 was subtracted from 100, divided by ten, and rounded to the nearest integer to arrive at a button recognition factor (BR) of three. The sum of KMS, IFPF, ALIF, and BR earns LDRA a HID score of sixteen as shown in table 3

The same operations were performed with the Emma, Gretel and Code cover and the following results were obtained.

	JCover	Emma	Gretel	Code Cover
KMS	2	3	5	4
IFPF	5	4	4	3
ALIF	6	5	4	4
BR	3	2	2	2
HID	16	14	15	13

Table 3 : Calculation of HID

Features	JCover	Emma	Gretel	Code cover
Coverage measures supported	Statement, Branch, Method, File and Class Coverage	Statement, Block Method and Class Coverage	Statement coverage	Statement, Branch, loop and condition Coverage
Memory space	8 MB	460 KB	501 KB	3.63MB
Graphical Representation	✓	X	X	✓
User Interface	✓	X	✓	✓
Residual coverage Monitoring	X	X	✓	X
HTML format	✓	✓	X	✓
Reports	Text file, HTML format, Graph	Text file, HTML format	Line table, Hit table	Coverage, correlation ,Boolean analyzer views & HTML format
Integrated with JUnit	✓	✓	X	✓

Table2: Comparison of the various features of coverage tools

9.1.2. Maturity and Customer Base (MCB)

All the tools we have identified are open source tools. These tools may have achieved considerable maturity as a result of customer satisfaction. As per literature these tools are widely used by most of the academic institutions that come under Anna University for their students lab sessions in software testing. Though the maturity cannot be calculated correctly all the tools may have more or less the same customer base (CB) and projects (P) done. This may be approximately 50%.

9.1.3 Tool Management: None of the four testing tool suites provide different access levels or other information control methods. Tool management must be controlled via computer policies implemented in the operating system and other applications outside of the suite of testing tools.

9.1.4 Ease of Use (EU):

All the four tools are easy to use and ensure timely, adequate and continual integration into the software development process. For all the tools , learning time of

first time users , retainability of procedural knowledge and operational time for frequent and casual use not very high which concludes that these tools are Easy to use .

9.1.5. Reporting Features. The Reporting Features (RF) metric is determined by one point for automatically generating summary reports and one point for producing reports in a format (e.g., HTML or ASCII text documents) that are viewable outside the application. Code Cover

has various ways of reporting the results automatically and generates summary reports formatted in HTML earning a RF measure of two for each vendor. JCover and Gretel also automatically produce summary reports, but they must be viewed within the testing application. Therefore, these tools RF measures are one. Emma tool has only plain text, HTML, XML report formats and has a RF measure of 0.5.

9.1.6. Maximum Number of Classes. No tool reported a limit on the number of classes it could support when testing object-oriented programs. Even so, this metric should remain within the testing tool metric. It could be detrimental to a software development project's success if a tool were selected and implemented only to discover it could not support the number of classes contained in the project.

9.1.7. Response Time. Each tool performed well with regards to response time. JCover averaged twenty-Nine minutes in performing its coverage. Emma averaged approximately twenty minutes.Emma averaged to twenty two minutes and Code Cover averaged to forty-two minutes.

9.1.8. User Control. All tools offered extensive user control of which portions of the code would be tested by a specified test case. Each allowed the user to specify a function, class, or project, or any combination of the three, to be tested.

9.1.9. Other Testing Tool Metrics. The remaining testing tool metrics require execution of extensive experiments or input from tool vendors. The scope of our research prevents conducting detailed experiments. Along with insufficient input from the vendors, this prevents analysis of the remaining metrics.

10. ANALYSIS OF RESULTS

The tools were analyzed both for Coverage metrics and Tool metrics.

10.1 Coverage Metrics analysis

The four suites of testing tools provided interesting results on the relative quality of the software under test. The results of the selected tools were analyzed as shown in Table 2 , based on Coverage measures that the

Sorting programs	JCoverage					Emma				Gretel	Code cover			
	SC	BC	MC	FC	CLC	SC	BLC	MC	CLC	SC	SC	BC	LC	CC
Bubble	89	76	95	89	100	86	86	95	100	83	**92**	**89**	**78**	**86**
Selection	88	76	96	86	100	85	85	94	100	82	**93**	**87**	**79**	**87**
Insertion	88	77	94	85	100	86	86	95	100	81	**92**	**87**	**77**	**86**
Heap	89	78	95	87	100	87	87	95	100	80	**93**	**88**	**80**	**88**
Merge	88	78	95	86	100	85	85	95	100	82	**92**	**87**	**80**	**86**
Quick	89	77	94	85	100	87	87	94	100	83	**93**	**89**	**81**	**87**
Payroll	86	75	95	85	100	87	85	93	100	83	**96**	**100**	**100**	**86**
Calculator	88	76	94	86	100	85	86	95	100	80	**90**	**99**	**I00**	**91**
Arrays of javac	87	75	94	85	100	85	87	94	100	81	**95**	**88**	**77**	**86**

Table4 Analysis & Implementation of JCover, Emma, Gretel and Code Cover Using Various Sort Programs

SC-Statement Coverage, BLC-Block Coverage,BC- Branch Coverage , LC-Loop Coverage,MC-Method Coverage, CC-Condition Coverage,FC-File Coverage,CLC-Class Coverage

tool supports like, Memory space, Graphical User Interface, Residual coverage Monitoring and reports in HTML format. While considering Coverage measures supported feature, Statement coverage is obtained by all the four tools.

Branch coverage is obtained by JCover and Code cover tools. Method and class coverage are obtained by JCover and Emma tools. File coverage is obtained by JCover. Block coverage is obtained by Emma. Loop and Condition coverage are obtained by Code Cover tool. Coverage measure provided by JCover and Code Cover tools can be viewed as graphical representation. Except Emma other three tools are good in user interface. While considering Residual coverage Monitoring, Gretel tool provides Residual coverage. While considering report generation JCover generates report in the form of Text file, HTML format and Graph. Emma generates report in the form of Text file and HTML format. Gretel generates report in the form of Line table and Hit table. Gretel generates report in the form of Coverage view, correlation view, Boolean analyzer view and HTML format. While considering JUnit integration, except Gretel other three tools are integrated with JUnit.

This paper, analyses of various coverage tools
namely, JCover, Emma, Gretel and Code Cover tools are performed. We have considered the sorting programs like Bubble Sort, Quick Sort, Insertion Sort, Heap Sort, Merge sort, Selection Sort, Payroll calculation, Calculator application and Code of Java compiler(only arrays were considered). The details of the programs are given in Table 1. In the table, Cyclomatic complexity which provides an upper bound for the number of test cases that are to be written for complete testing of the application is calculated. In total 67 test cases for Sorting programs ,29 test cases for payroll program,58 test cases for calculator program and 255 test cases for Code of Java compiler(only arrays were considered), were written and executed using JUnit framework.[17]

JUnit [17] is a framework for executing unit test cases that contain java classes with one or more test methods Coverage percentage using the various tools were measured for the sorting programs and the results presented in Table.4. While considering Statement Coverage for Bubble sort JCover tool provides 89%, Emma tool provides 86%, Gretel tool provides 83% and Code cover tool provides 92%. While considering Statement Coverage for Calculator application JCover tool provides 88%, Emma tool provides 85%, Gretel tool provides 80% and Code cover tool provides 90%.From Table4 it is clear that Code Cover tool provides more coverage than the other three tools.

Coverage view of Code cover tool for Calculator program is shown in Figure 1.

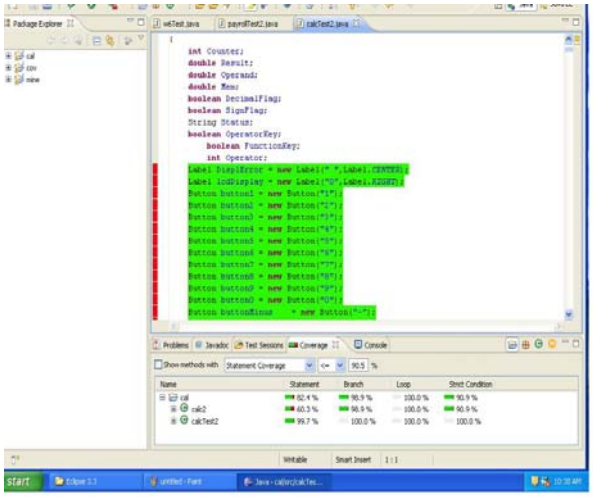

Figure1 Coverage view of Code Cover tool for Calculator program

For Heap sort Code Cover tool provides 93% statement coverage, 88%Branch coverage, 80%Loop Coverage and 86%Condition Coverage. Correlation view of Code Cover tool for Heap sort program is shown in Figure2

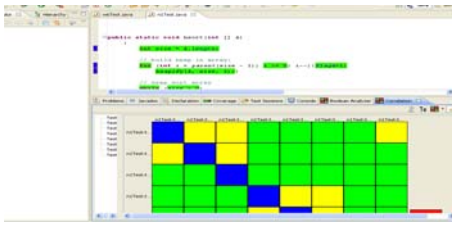

Figure2 Correlation view of Code Cover tool for Heap sort program

Graphical representation of Total coverage provided by JCover, Emma, Gretel and Code Cover tools for Java sorting programs is shown in Figure3.

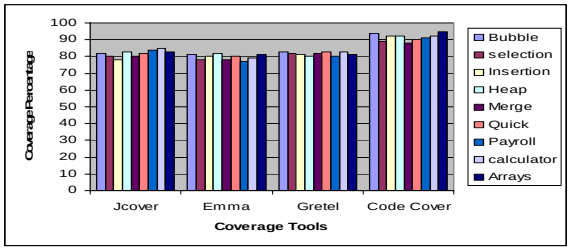

Figure 3.Graphical Representation of Total Coverage

From Figure 3., it is shown that Code cover tool provides more coverage percentage than JCover, Emma and Gretel. As a total, Code Cover tool provides 94% for Bubble sort, 89% for Selection sort, 92% for Insertion sort,92% for Heap sort ,88%for Merge sort 90% for Quick sort,91% for Payroll calculation,92% for calculator application and 95% for Arrays class of java compiler. According to our analysis, the Test Manager can select Code Cover tool to calculate the coverage metrics and to select the best test cases for regression testing for all kind of Java programs. Graphical representation of Statement Coverage, Branch Coverage, Loop Coverage and Condition Coverage provided by Code Cover tool is shown in Figure 4.

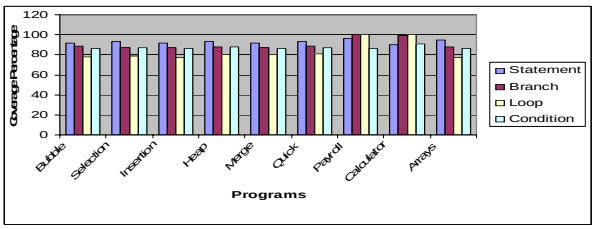

Fig4.Graphical Representation of Coverage provided by Code Cover tool.

10.2 Tool Metrics analysis

Success was also achieved in applying several of the metrics including HID, TCG, TM,EU,EC,TS and RF. HID measurements were calculated for each testing tool based on the sub-metrics of average KMS, IFPF, ALIF, and BR when applicable. The sub-metrics demonstrated non-coarseness (different values were measured), finiteness (no metric was the same for all tools), and non-uniqueness (some equal values were obtained). The HID measurements were all unique, indicating that the measurement could be useful in comparing tools during the evaluation and selection process.

Tools	HID	MCB	EU	RF	RT
JCover	16	50%	83%	84	22min
Emma	14	50%	82%	78	23min
Gretel	15	50%	81%	81	24min
Code Cover	**13**	**50%**	**86%**	**90**	**20min**

Table5.Analysis of Tool Metrics

RF measurements were also successful. It is simple to determine whether a tool automatically generates summary reports (SR) that are viewable without the tool application running (e.g., HTML document) (ER). The RF metric is non coarse, finite, and non-unique. However, because each tool earned a SR score of one, additional testing should be conducted to determine SR's level of non-uniqueness.

Tools	Coverage Metrics(%)								Tool Metrics				
	Bubble	Selection	Insertion	Heap	Merge	Quick	Payroll	Calculator	HID	MCB(%)	EU	RF	RT(secs)
JCover	82	80	78	83	80	82	84	85	16	50	83	84	22
Emma	81	78	80	82	78	80	77	79	14	50	82	78	23
Gretel	83	82	81	80	82	83	80	83	15	50	81	81	24
Code Cover	**94**	**89**	**92**	**88**	**90**	**91**	**92**	**95**	**13**	**50**	**86**	**90**	**20**

Table 6 Complete Analysis

The Maturity & Customer Base, Tool Support, Estimated Return on Investment, Reliability, and Maximum Number of Parameters metrics were not completed. In order to do so would involve conducting more experiments or obtaining tool-vendor input, the latter of which is not readily available.The details of the other metrics for the four tools are as shown in table 5.

From table 5 we infer that the Code Cover Tool is easy to use, has a very good response time for every command given, has very good reporting features as mentioned in section 7.5 and Table 2.

10.3 Combined Analysis

The combined analysis of Coverage and Tool metrics for the four tools we have selected namely JCover, Emma, Gretel and Code Cover is detailed in Table 6. From our analysis and from the table 6 it is clear that the Code Cover tool satisfies most of the features we have considered for the coverage tool evaluation

11. CONCLUSION AND FUTURE ENHANCEMENT

Well-designed metrics with documented objectives can help an organization obtain the information it needs to continue to improve its softwareproducts, processes, and services while maintaining a focus on what is important.

Our metrics captured differences in the various suites of software-testing tools, relative to the software system under test; the software-testing tools vary in their underlying approach, quality, and ease-of-use, among other characteristics. However, confirming evidence is needed to support our theories about the effectiveness of the tool metrics for improving the evaluation and selection of software-testing tools

Most test coverage analyzers help in evaluating the effectiveness of testing by providing data on various coverage metrics achieved during testing. If made available, the coverage information can be very useful for many other related activities, like, regression testing, test case prioritization, test-suite augmentation, test-suite minimization, etc. In this paper, an analysis of the various Java Code Coverage tools were presented and from the study we have suggested that Code Cover tool is comparatively better in calculating the coverage metrics and helps in the selection of best test cases based on coverage for regression testing of Java programs.

As a future enhancement, future research is to conduct more intensive testing with the candidate tools by creating additional test cases and modifying default test settings to improve test coverage and conducting regression testing. One could also compare the testing tools under various operating system configurations and tool settings, or measure a tool's capability and efficiency in both measuring and improving testing.

REFERENCES

[1] Boris Beizer: "Software Testing Techniques", Second Edition, International Thomson Computer Press, 1990, ISBN 1-85032-880-3.

[2] Glenford J.Myers, "The Art of Software Testing", John Wiley, 1979

[3] Prasad K.V.K.K "Software testing tools ", 2006 edition

[4] Qaiser Durrani, " Role of Software Metrics in Software Engineering and Requirements Analysis ", 2005 IEEE.

[5] Grady, R.B, Practical Software Metrics for ProjectManagement and Process Improvement, Prentice-Hall, 1992.

[6] John Joseph Chilenski and Steven P. Miller, "Applicability of Modified Condition/Decision Coverage to Software Testing", Software Engineering Journal, September 1994, Vol. 9, No. 5, pp.193-2000.

[7] Howden, "Weak Mutation Testing and Completeness of Test Sets", IEEE Trans. Software Eng., Vol.SE-8, No.4, July 1982, pp.371-379.

[8] Woodward, M.R., Hedley, D. and Hennell, M.A., "Experience with Path Analysis and Testing of Programs", IEEE Transactions on Software Engineering, Vol. SE-6, No. 3, pp. 278-286, May 1980.

[9] Norman E Fenton, Martin Neil, "Software Metrics: Roadmap".

[10] Roger S. Pressman, Software Engineering: A Practitioner's Approach

[11] http://www.codecoverage.com

[12]http://www.codecoveragetools.com/code_coverage_java.html

[13]http://www.cs.uoregon.edu/research/perpetual/dasada/Software/Gretel

[14] http://Emma.sourceforge.net

[15] Poston, R. M. and Sexton, M. P. Evaluating and selecting testing tools. *IEEE Software* 9, 3 (May 1992), 33-42.

[16] Dekkers, C. Demystifying function points: Let's understand some terminology. *IT Metrics Strategies*, Oct. 1998.

[17] Chidamber, S. R. and Kemerer, R. F. A metrics suite for object-oriented design. *IEEE Trans. Software Eng.* 20, 6 (June 1994), 476-493.

[18] Lorenz, M. and Kidd, J. *Object-Oriented Software Metrics*. Englewood Cliffs, N.J.: Prentice Hall, 1994.

[19] Youngblut, C. and Brykczynski B. An examination of selected software testing tools: 1992. IDA Paper P-2769,Inst. for Defense Analyses, Alexandria, Va., Dec. 1992.

[20] Youngblut, C. and Brykczynski, B. An examination of selected software testing tools: 1993 Supp. IDA Paper P- 2925, Inst. for Defense Analyses, Alexandria, Va., Oct. 1993.

[21] Daich, G. T., Price, G., Ragland, B., and Dawood, M. Software test technologies report. Software Technology Support Center, Hill AFB, Utah, Aug. 1994.