

TABLE OF CONTENTS

S.No.	Contents	Page No.
1	Lab Rules	2
2	Instructions	4
3	Zero Lab	5
4	Syllabus	7
5	Marks Scheme	10
6	Lab Plan	11
7	Introduction to the subject	12
8	Lab objective	13
9	Procedure	14
10	List of lab exercises	15
11	Experiments	16
12	Resources	

LAB RULES

Responsibilities of Users

Users are expected to follow some fairly obvious rules of conduct:



Always:

- Enter the lab on time and leave at proper time.
- Wait for the previous class to leave before the next class enters.
- Keep the bag outside in the respective racks.
- Utilize lab hours in the corresponding.
- Turn off the machine before leaving the lab unless a member of lab staff has specifically told you not to do so.
- Leave the labs at least as nice as you found them.
- If you notice a problem with a piece of equipment (e.g. a computer doesn't respond) or the room in general (e.g. cooling, heating, lighting) please report it to lab staff immediately. Do not attempt to fix the problem yourself.



Never:

- Don't abuse the equipment.
- Do not adjust the heat or air conditioners. If you feel the temperature is not properly set, inform lab staff; we will attempt to maintain a balance that is healthy for people and machines.
- Do not attempt to reboot a computer. Report problems to lab staff.
- Do not remove or modify any software or file without permission.
- Do not remove printers and machines from the network without being explicitly told to do so by lab staff.

- Don't monopolize equipment. If you're going to be away from your machine for more than 10 or 15 minutes, log out before leaving. This is both for the security of your account, and to ensure that others are able to use the lab resources while you are not.
- Don't use internet, internet chat of any kind in your regular lab schedule.
- Do not download or upload of MP3, JPG or MPEG files.
- No games are allowed in the lab sessions.
- No hardware including USB drives can be connected or disconnected in the labs without prior permission of the lab in-charge.
- No food or drink is allowed in the lab or near any of the equipment. Aside from the fact that it leaves a mess and attracts pests, spilling anything on a keyboard or other piece of computer equipment could cause permanent, irreparable, and costly damage. (and in fact *has*) If you need to eat or drink, take a break and do so in the canteen.
- Don't bring any external material in the lab, except your lab record, copy and books.
- Don't bring the mobile phones in the lab. If necessary then keep them in silence mode.
- Please be considerate of those around you, especially in terms of noise level. While labs are a natural place for conversations of all types, kindly keep the volume turned down.

If you are having problems or questions, please go to either the faculty, lab in-charge or the lab supporting staff. They will help you. We need your full support and cooperation for smooth functioning of the lab.

INSTRUCTIONS

Before entering in the lab

All the students are supposed to prepare the theory regarding the next experiment.

Students are supposed to bring the practical file and the lab copy.

Previous programs should be written in the practical file.

All the students must follow the instructions, failing which he/she may not be allowed in the lab.

While working in the lab

Adhere to experimental schedule as instructed by the lab in-charge.

Get the previously executed program signed by the instructor.

Get the output of the current program checked by the instructor in the lab copy.

Each student should work on his/her assigned computer at each turn of the lab.

Take responsibility of valuable accessories.

Concentrate on the assigned practical and do not play games

If anyone caught red handed carrying any equipment of the lab, then he/she will have to face serious consequences.

**Swami Keshvanand Institute of Technology, Management &
Gramothan (SKIT Jaipur)
Software Testing and Validation Lab (8CS4-22)**

Zero Lab

Prepared by: Priyanka

1) Name of the lab with code: Software Testing and Validation Lab (8CS4-22)

2) Self-Introduction:

- a) **Name** :
- b) **Qualification** :
- c) **Designation** :
- d) **E-mail ID** :

e) Introduction of Students:

An interactive session will be held with the students wherein they will be asked to introduce themselves, covering the following points:-

- 1. Academics Merit/Weak
- 2. Co-curricular Activity
- 3. Day Scholar/ Hosteller
- 4. Medium Hindi/English
- 5. Family Background Urban/Rural
- 6. Learning Style seeing/ hearing/ doing

4). Introduction to Lab:-

a) Relation with other labs:

In this current semester they have other lab which is directly or indirectly related with this lab as core part of anything in the area of Computer Science is the development of good testing scenarios. Thus this lab definitely helps the student to develop their concepts in the software testing and therefore implement their program more efficiently.

Connection with previous year and next year:

Students have studied about software engineering in 4th semester. The concept and techniques of software testing they have learnt in those labs will definitely help them.

b) Lab schedule per week:

Two hours per batch per week

5) University Examination System:-

Sr. No.	Name of the Exam	Max. Marks	% of passing marks	Syllabus coverage (in %)
1	I Mid Term Exam	40	40	50%
2	II Mid Term Exam	40	40	50%
3	University (End) Term Exam	40	40	100%

Place:SKIT, Jaipur

Date : April 24 2021

SYLLABUS



RAJASTHAN TECHNICAL UNIVERSITY, KOTA

Scheme & Syllabus

IV Year- VII Semester: B. Tech. (Computer Science & Engineering)

8CS4-22: Software Testing and Validation Lab

Credit: 1

Max. Marks:50 (IA:30, ETE:20)

OL+OT+2P

End Term Exam: 2 Hours

SN	List of Experiments																	
1	<p>a) Write a program that calculates the area and perimeter of the circle. And find the Coverage & Test Cases of that program using JaButi Tool.</p> <p>b) Write a program which read the first name and last name from console and matching with expected result by using JaBuTi.</p> <p>c) Write a program that takes three double numbers from the java console representing , respectively, the three coefficients a,b, and c of a quadratic equation.</p> <p>d) Write a program that reads commercial website URL from a url from file .you should expect that the URL starts with www and ends with .com. retrieve the name of the site and output it. For instance, if the user inputs www.yahoo.com, you should output yahoo. After that find the test cases and coverage using JaButi.</p> <p>e) Write a program for a calculator and find the test case and coverage and Def-use-graph.</p> <p>f) Write a program that reads two words representing passwords from the java console and outputs the number of character in the smaller of the two. For example, if the words are open and sesame, then the output should be 4, the length of the shorter word, open. And test this program using JaButi</p>																	
2	<p>Analyse the performance of following website using JMeter.</p> <table><tr><td>Site</td><td>Website</td><td>Type</td></tr><tr><td>Amazon</td><td>Amazon.com</td><td>shopping</td></tr><tr><td>Flip kart</td><td>Flipkart.com</td><td>shopping</td></tr><tr><td>Railway reservation</td><td>Irctc.co.in</td><td>Ticket booking site</td></tr><tr><td>Train searching</td><td>Erail.in</td><td>Train searching</td></tr></table>			Site	Website	Type	Amazon	Amazon.com	shopping	Flip kart	Flipkart.com	shopping	Railway reservation	Irctc.co.in	Ticket booking site	Train searching	Erail.in	Train searching
Site	Website	Type																
Amazon	Amazon.com	shopping																
Flip kart	Flipkart.com	shopping																
Railway reservation	Irctc.co.in	Ticket booking site																
Train searching	Erail.in	Train searching																
3	<p>Calculate the mutation score of programs given in 1(a) to 1 (f) using jumble Tool.</p>																	
4	<p>Calculate the coverage analysis of programs given in 1 (a) to 1 (f) using Eclemma Free open source Tool.</p>																	

Office of Dean Academic Affairs
Rajasthan Technical University, Kota

Scheme & Syllabus of 4th Year B. Tech. (CS) for students admitted in Session 2017-18 onwards. Page 10

**RAJASTHAN TECHNICAL UNIVERSITY, KOTA****Scheme & Syllabus****IV Year- VII Semester: B. Tech. (Computer Science & Engineering)****5**

Generate Test sequences and validate using Selenium tool for given websites below:

Site	Website	Type
Amazon	Amazon.com	shopping
Flip kart	Flipkart.com	shopping
Railway reservation	Irctc.co.in	Ticket booking site
Train searching	Erail.in	Train searching

Office of Dean Academic Affairs
Rajasthan Technical University, Kota

Scheme & Syllabus of 4th Year B. Tech. (CS) for students admitted in Session 2017-18 onwards. Page 11

MARKS SCHEME

RTU Marks Scheme

Maximum Marks Allocation		
Sessional	End-Term	Total
60	40	100

Marks Division

Mid Term – I & II		
Performance	Viva	Total
30	10	40
Attendance & Performance		
Performance	Attendance	Total
30	10	40
End-Term Practical		
Performance	Viva	Total
30	10	40

Internal Assessment System

Total Marks – 10

Attendance	Discipline	Performance	Record	Viva	Total
2	2	2	2	2	10

LAB PLAN

Total number of experiment 12

Total number of turns required 12

Number of turns required for

Experiment Number	Turns	Scheduled Day
Exp. 1	1	Day 1
Exp. 2	1	Day 2
Exp. 3	1	Day 3
Exp. 4	1	Day 4
Exp. 5	1	Day 5
Exp. 6	1	Day 6
Exp. 7	1	Day 7
Exp. 8	1	Day 8
Exp. 9	1	Day 9
Exp. 10	1	Day 10
Exp. 11	1	Day 11
Exp. 12	1	Day 12

Distribution of lab hours

Attendance 05 minutes

Explanation of the concept 25 minutes

Explanation of experiment 25 minutes

Performance of experiment 95 minutes

Viva / Quiz / Queries 30 minutes

Total 180 minutes

Software required

JABUTI – Java Bytecode Understanding and Testing Tool

J Meter

Jumble

Eclema

Selenium

Introduction to the subject

This Subject gives an in-depth knowledge about Testing and Validation. This subject provides us the way to specify various testing technologies and tools, some of design strategies and many of fundamental ideas used in test case analysis.

Software project management, software testing, and software engineering, verification and validation is the process of checking that a software system meets specifications and requirements so that it fulfills its intended purpose. It may also be referred to as software quality control.

This subject is not important for a single lab but for all labs as all labs is concerned with testing the solution of a problem by programming and it is best way for verified programming and solving a problem.

Lab Objective

Objectives: Upon successful completion of this course, students should be able to:

- Coverage Testing with the help of JaBUTi tool
- Performance Testing with Apache JMeter Tool
- Mutation Testing with Jumble Tool
- Utilization of Eclemma Tool
- Coverage testing with Selenium

A. It is expected that teachers will assign experiments to the students for testing. Problems and Test cases may be chosen by the teachers.

B. Problem on performing testing to meet required constraints may be assigned. For example, a problem on testing a web site for performance or coverage testing of program byte code.

List of Lab Exercises

1. Write a program that calculates the area and perimeter of the circle. And find the Coverage & Test Cases of that program using JaButi Tool.
2. Write a program which read the first name and last name from console and matching with expected result by using JaButi.
3. Write a program that takes three double numbers from the java console representing , respectively, the three coefficients a,b, and c of a quadratic equation.
4. Write a program that reads commercial website URL from a url from file. you should expect that the URL starts with www and ends with .com. retrieve the name of the site and output it. For instance, if the user inputs www.yahoo.com, you should output yahoo. After that find the test cases and coverage using JaButi.
5. Write a program for a calculator and find the test case and coverage and Def-use-graph.
6. Write a program that reads two words representing passwords from the java console and outputs the number of character in the smaller of the two. For example, if the words are open and sesame, then the output should be 4, the length of the shorter word, open. And test this program using JaButi
7. Analyze the performance of following website using JMeter.

Site	Website	Type
Amazon	Amazon.com	shopping
Flip kart	Flipkart.com	shopping
Railway reservation	Irctc.co.in	Ticket booking site
Train searching	Erail.in	Train searching

8. Calculate the mutation score of programs given in 1(a) to 1 (f) using Jumble Tool.
9. Calculate the coverage analysis of programs given in 1 (a) to 1 (f) using Eclemma Free open source Tool.
10. Generate Test sequences and validate using Selenium tool for given websites below:

Site	Website	Type
Amazon	Amazon.com	shopping
Flip kart	Flipkart.com	shopping
Railway reservation	Irctc.co.in	Ticket booking site
Train searching	Erail.in	Train searching

Experiments

Experiment No.: 1

Name of Experiment: Write a program that calculates the area and perimeter of the circle. And find the Coverage & Test Cases of that program using JaButi Tool.

Aim :

Steps to perform Experiment:

Program Code:AreaOfCircle.java

```
import java.util.Scanner;
import java.lang.Math;
public class AreaOfCircle
{
    public static void main(String[] args)
    {
        int option;
        double radius, circumference, diameter, area;
        //object of the Scanner class
        Scanner sc=new Scanner (System.in);
        //options available
        System.out.println("1. If the radius is known");
        System.out.println("2. If the diameter is known");
        System.out.println("3. If the circumference is known");
        System.out.print("Enter your choice: ");
        //taking an option as input from the user
        option=sc.nextInt();
        switch(option)
        {
            //Case statements
            case 1:
                System.out.print("Enter the radius of the circle: ");
                radius=sc.nextDouble();
                area=(Math.PI*(radius*radius));
```

```
    System.out.print("The area of the circle is: "+area);  
    break;  
case 2:  
    System.out.print("Enter the diameter of the circle: ");  
    diameter=sc.nextDouble();  
    area=Math.PI*(diameter*diameter)/4;  
    System.out.print("The area of the circle is: "+area);  
    break;  
case 3:  
    System.out.print("Enter the circumference of the circle: ");  
    circumference=sc.nextDouble();  
    area=(circumference*circumference)/(4*Math.PI);  
    System.out.print("The area of the circle is: "+area);  
    break;  
//default case statement executes when an invalid choice is entered  
default: System.out.println("invalid choice!");  
}  
}  
}
```

Compile Java Program:-> _javac AreaOfCircle.java

It will create a new file AreaOfCircle.class // **BYTE CODE**

Now assign this ByteCode to JaBUTi Tool for Coverage Testing by following steps

1. JaBUTi Functionality and Graphical Interface

JaBUTi implements a subset of the functionalities provided by xSuds, a complete tool suite for C and C++ programs . This section describes the operations that can be performed by JaBUTi. The graphical interface allows the beginner to explore and learn the concepts of control-flow and data-flow testing. Moreover, it provides a better way to visualize which part of the classes under testing are already covered and which are not. A general view of the JaBUTi graphical interface, including its menus, is presented in Figure.

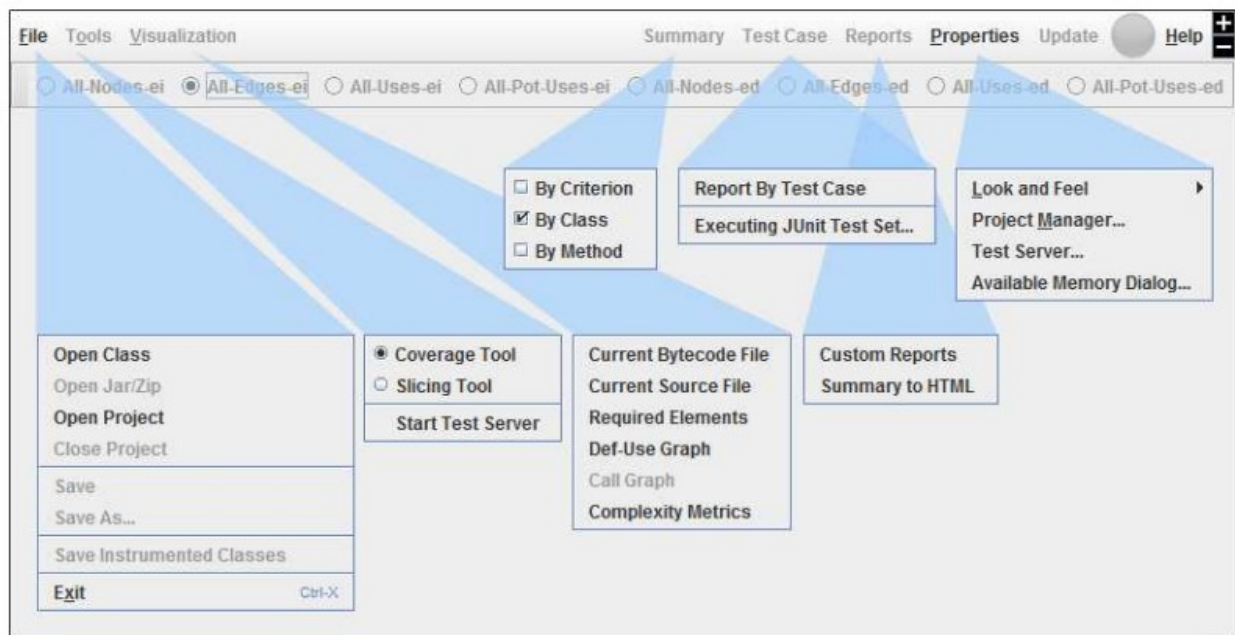
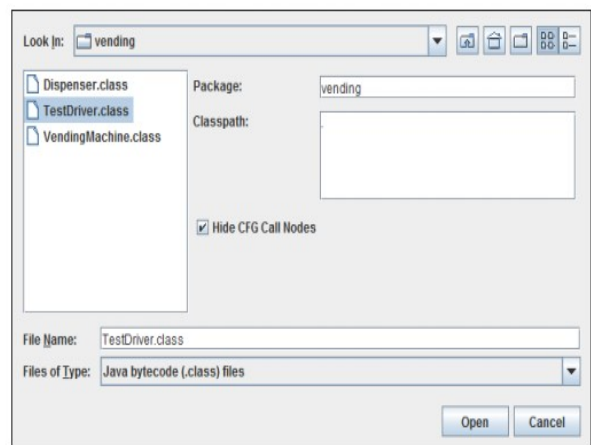
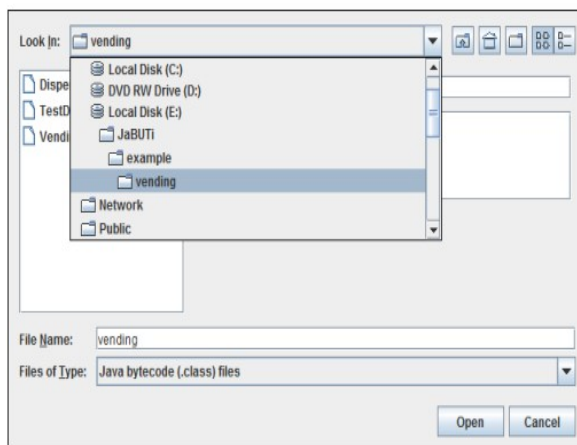


Figure 1. A brief description of each option on each menu

2 How to Create a Testing Project In JaBUTi

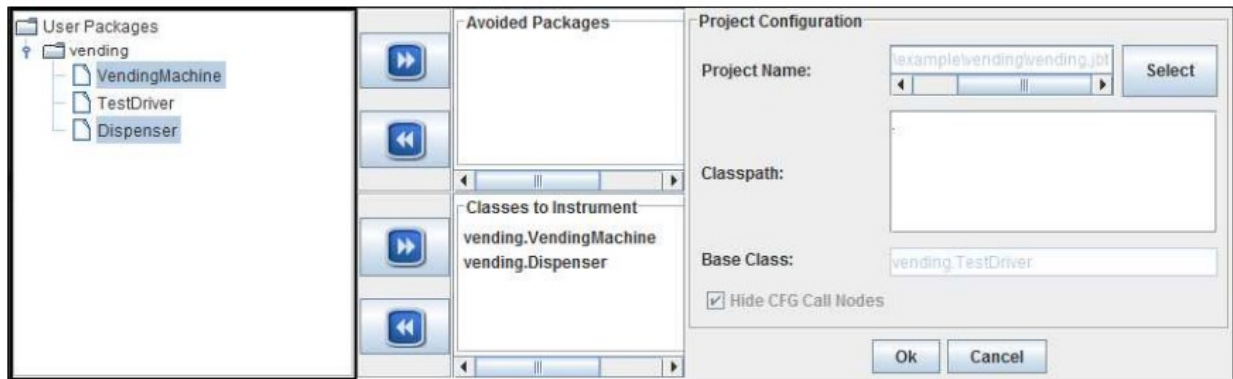
The testing activity requires the creation of a testing project. A testing project is characterized by a file storing the necessary information about (i) the base class file, (ii) the complete set of classes required by the base class, (iii) the set of classes to be instrumented (tested), and (iv) the set of classes that are not under testing. Additional information, such as the CLASSPATH environment variable necessary to run the base class is also stored in the project file, whose extension is .jbt. During the execution of any .class file that belongs to the set of classes under testing of a given project, dynamic control-flow information (execution trace) is collected and saved in a separate file that has the same name of the project file but with a different extension (.trc).

To create a new project, the first step is to select a base .class file from **File** → **Open Class** menu. A dialog window, as illustrated in following Figure.



From this dialog window, the tester selects the directory where the base class file is located and then select the base class file itself . Once the base class file is selected the tool automatically identifies the package that it belongs (if any) and fills out the Package field with the package's name. The Classpath should contains only the path necessary to run the selected base class.

By clicking the Open button the **Project Manager window**, will be displayed.



From the selected base class file the tool identifies the complete set of system and non-system class files necessary to execute class file

Currently, JaBUTi does not allow the instrumentation of system class files. Therefore, the complete set of non-system class files (user's classes) related to the selected class can be instrumented. From the Project Manager window (left side) the user can select the class files that will be tested. At least one class file must be selected.

Moreover, the tester must give a name to the project being created by clicking on the Select button.

By clicking on the Ok button, JaBUTi creates a new project, constructs the DUG for each method of each class under testing, derives the complete set of testing requirements for each criterion, calculates the weight of each testing requirement, and presents the bytecode of a given class under testing. By creating a project, the tester does not need to go through this entire process again if he/she intends to test the same set of class files. Next time, he/she can just go to File → Open Project menu option and reload a previously saved project.

Once the project is created, the tester just need to open it and the tool automatically detects the classes that compose it.

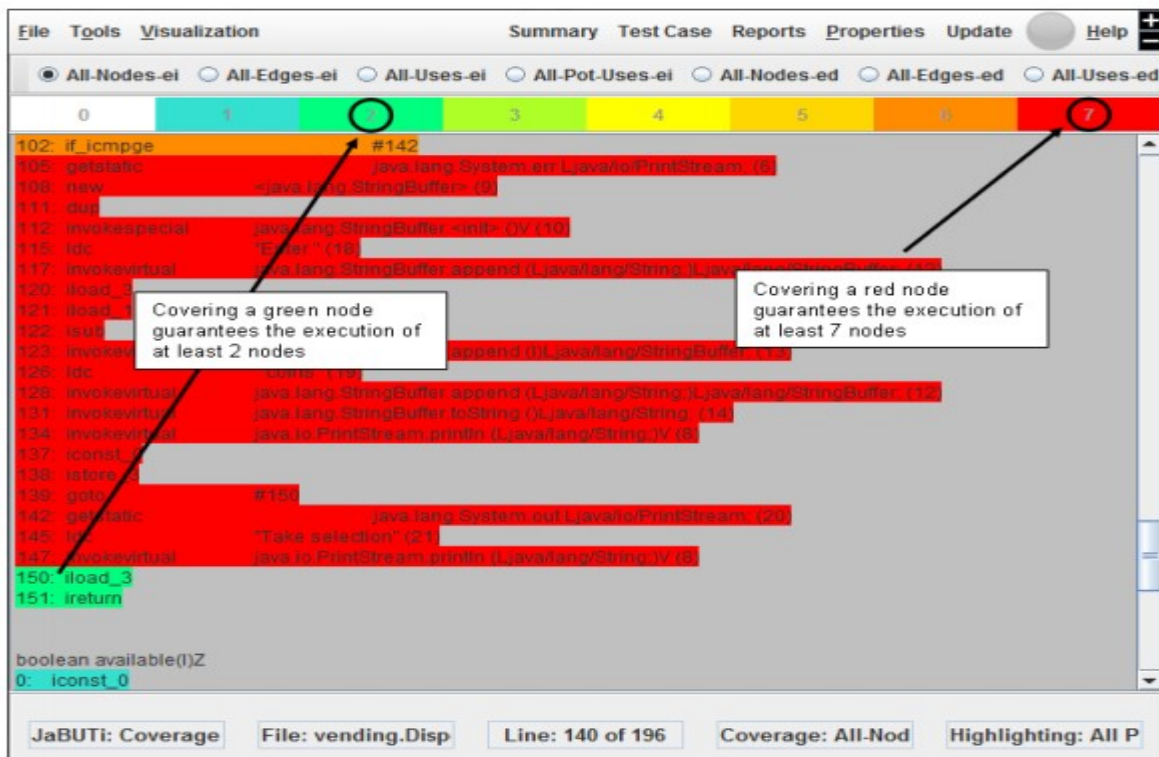
JaBUTi can be used for analyzing the coverage of a given class file, for debugging the class file using the slice tool and for collecting static metrics information.

JaBUTi also allows the visualization of the Def-Use Graph (DUG) of each method in a given class file as well as the visualization of the source code, when available.

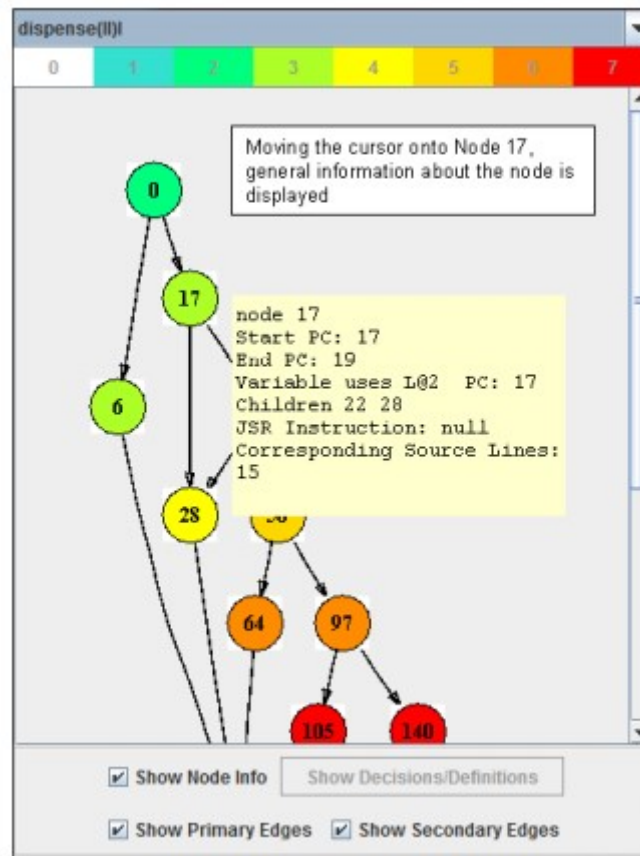
Different kinds of testing reports can also be generated considering different levels of abstraction.

3 How to use JaBUTi as a Coverage Analysis Tool

By default, the tool displays the bytecode instead of the Java source code, since the source code may not be available. The user can use the Visualization menu to change the display to show the bytecode or source code, as well as the DUG (Depth Use Graph) of each method in the current class.



Example - Dispenser.dispenser method: bytecode prioritized w.r.t. All-Pri-Nodes criterion



Example - Dispenser.dispenser method: DUG prioritized w.r.t. All-Pri-Nodes criterion.

Observe that methods in Java can have more than one exit node due to the exception-handling mechanism. All the other nodes are represented as single line circles. We also have two different types of edges to represent the “normal” controlflow (continuous line – Primary Edges) and exception control-flow (dashed lines – Secondary Edges). Figure 6 does not contain exception edges. Primary and secondary edges can be hidden by deselecting the Show Primary and Show Secondary Edges check box, respectively. The node information shown when the cursor is moved onto the node, as illustrated in Figure 6, can also be disabled by deselecting the Show Node Info check box.

4. How the testing requirements are highlighted

As can be observed above the main menu in, JaBUTi supports the application of six structural testing criteria:

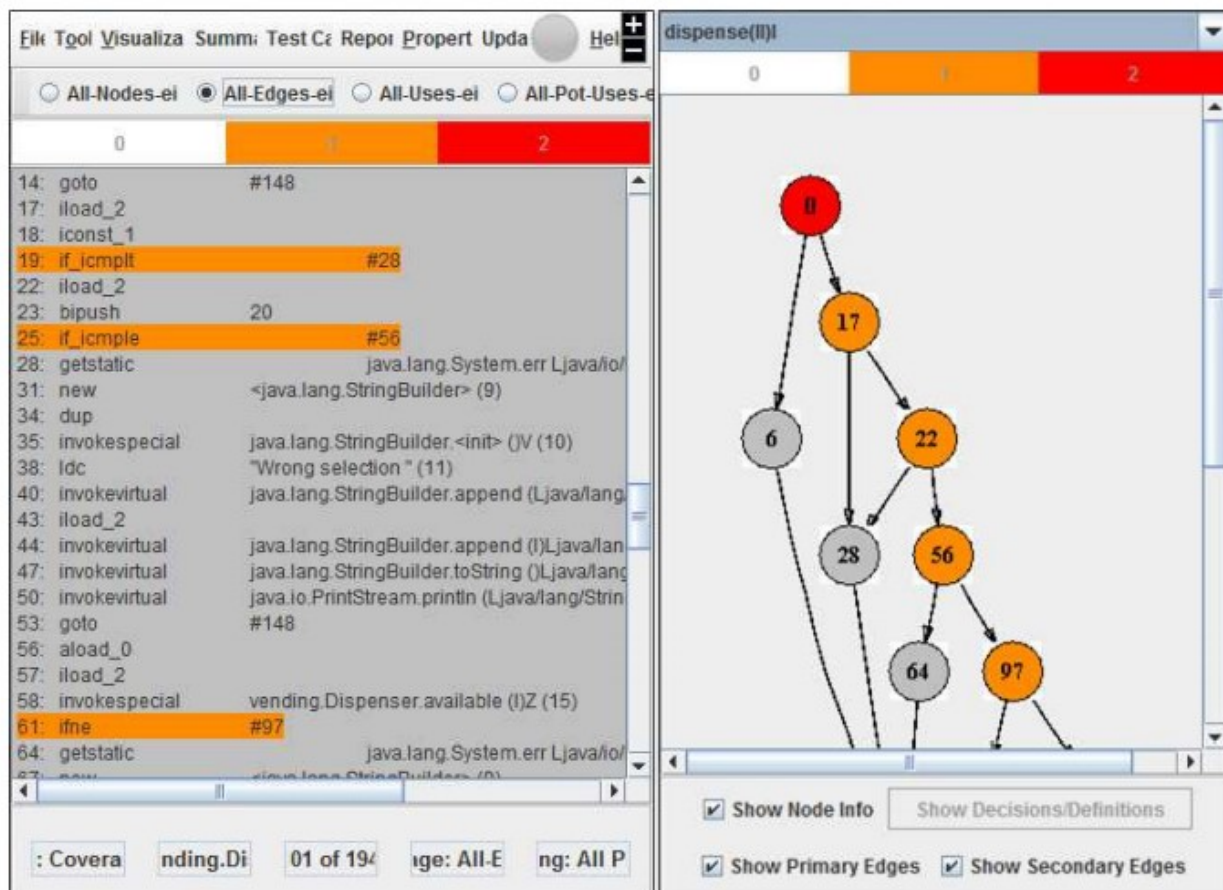
All-Nodes-ei, All-Nodes-ed, All-Edges-ei, All-Edges-ed, All-Uses-ei, and All-Uses-ed.

Depending on which criterion is active, the bytecode, source code or DUG is colored in a different way.

The color schema considering the All-Nodes-ei criterion, which is the same as the All-Nodes-ed, i.e., for these criteria, each requirement is a node, therefore, since each node has its own weight, the complete bytecode, source code or DUG appears highlighted according to the weight of each node.

Considering the **All-Edges-ei and All-Edges-ed criteria**, their requirements (DUG edges) are colored using a 2-layer approach.

For All-Edges-ei criterion, only the nodes with more than one out-going edge (decision nodes) are painted in the first layer.



By clicking either in a colored bytecode instruction or in a DUG node, all destination nodes of the branches associated with the selected decision node are highlighted or the decision node itself changes to a different color.

4 How to include a test case

JaBUTi is designed to support test case set adequacy evaluation and to provide guidance in test case selection. If there is a previously developed test set, e.g. a functional test set, such a test set can be evaluated against the structural testing criteria implemented by JaBUTi. This allows the tester to check, for example, whether all instructions/statements are executed by a particular test set. If they are not yet executed, additional test cases can be developed to improve the quality of the previous test set. On the other hand, if there is no previous test set available, the tester can use the hints provided by the tool to create test cases aiming at to cover the different testing requirements generated by JaBUTi based on its structural criteria. Once the test criteria can be applied incrementally, the tester can start by developing an adequate test set w.r.t. the All-Nodes-ei criterion, then, if necessary, to develop additional test cases to evolve the All-Nodes-ei-adequate test set to a All-Edges-ei-adequate test set, and later, if necessary, to develop additional test cases to evolve it to an All-Uses-ei-adequate test set. All these criteria, prefixed by All-Pri-, are related with the normal program execution. If desired, the tester can check the coverage of the exception-handling mechanism, by using the All-Nodes-ed, All-Edges-ed and All-Uses-ed criteria. The idea is to use these criteria incrementally to reduce their complexity and also to allow the tester to deal with the cost and time constraints.

JaBUTi allows to include test cases in two different ways:

- 1) using the JaBUTi's class loader; or
- 2) importing from a JUnit test set.

The first is done by a command line application (probe.ProberLoader, the JaBUTi's class loader). This command line application extends the default Java class loader such that, from a given testing project, it identifies which classes should be instrumented before to be loaded and executed. By detecting that a given class belongs to the set of classes under testing, JaBUTi's

class loader inserts the probes to collect the execution trace and then loads the instrumented class file to be executed. The trace information w.r.t. the current execution is appended in a trace file with the same name of the testing project but with the extension .trc instead of .jbt.

Considering Figure 21(b), observe that the CLASSPATH variable is set containing all the paths necessary to run the JaBUTi's class loader and also the vending machine example. The next parameter, probe.ProberLoader, is the name of the JaBUTi's class loader. It demands two parameters to execute: the name of the project (-P vending.jbt), and the name of the base class to be executed (vending.TestDriver). In our example, since vending.TestDriver requires one parameter to be executed,

```
auri@AURIMRV ~/example
$ cat input1
insertCoin
vendItem 3
```

Figure 20: Example of test case file.

<pre>auri@AURIMRV ~/example \$ java -cp "." vending.TestDriver input1 VendingMachine ON Current value = 25 Enter 25 coins Current value = -25 VendingMachine OFF</pre>	<pre>auri@AURIMRV ~/example \$ java -cp ";\..\Tools\jabuti;\ >..\Tools\jabuti\lib\BCEL.jar;\ >..\Tools\jabuti\lib\crimson.jar;\ > probe.ProberLoader -P vending.jbt \ > vending.TestDriver input1 Project Name: vending.jbt Trace File Name: vending.trc Processing File vending.jbt VendingMachine ON Current value = 25 Enter 25 coins Current value = -25 VendingMachine OFF</pre>
(a)	(b)

Figure 21: vending.TestDriver output: (a) default class loader, and (b) JaBUTi class loader.

this parameter is also provided (input1). During the execution of vending.TestDriver, VendingMachine and Dispenser classes are required to be loaded. From the project file (vending.jbt) the JaBUTi's class loader detects that these class files have to be instrumented before to be executed. The instrumentation is performed on-the-fly every time the probe.ProberLoader is invoked. The resultant execution trace information, corresponding to the execution of the test case input1, is appended in the end of the trace file vending.trc . Every time

the size of the trace file increase, the Update button in the JaBUTi's graphical interface becomes red, indicating that the coverage information can be updated considering the new test case(s).

Figure 22 illustrates this event



Figure 22: Update button with a different background color: new test case(s) available.

By clicking on the Update button, JaBUTi imports the new test case(s), empties the trace file vending.trc and updates the coverage information and the weights for each testing criterion. For example, after the importation of test case input1, the weight of the Java source code w.r.t. the All-Nodes-ei criterion changed as illustrated in Figure 23. Comparing the updated source of Figure 23 (front) with the one presented in Figure 23 (back) it can be observed that the requirement with the highest changed from source lines 22 and 24 (now covered – painted in white) to source lines 16, 18, 20 and 27. Considering the entire project, Figure 24(b) shows the updated coverage for each method in the Dispenser and VendingMachine classes w.r.t. the All-Nodes-ei criterion. Observe that input1 covered 7 primary nodes (63%) of Dispenser.dispense method and 19 of 29 primary nodes (65%) considering the entire project. By accessing the Test Case → Report by Test Case menu option, the tester can visualize the coverage of the entire project by test case, as illustrated in Figure 24(a). If desired, the tester can develop additional test cases to improve the coverage w.r.t. All-Nodes-ei criterion. For example, besides input1, Table 1 shows four additional test cases developed to improve the coverage of such a criterion.

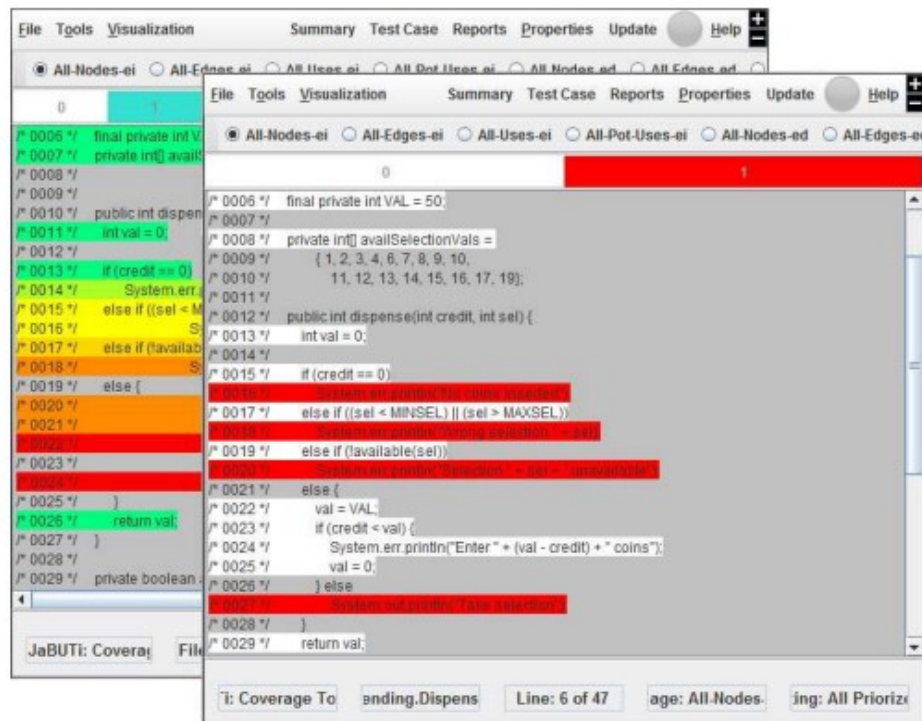


Figure 23: Dispenser.dispenser() method before and after the weight's updating w.r.t. All-Pri-Nodes criterion.

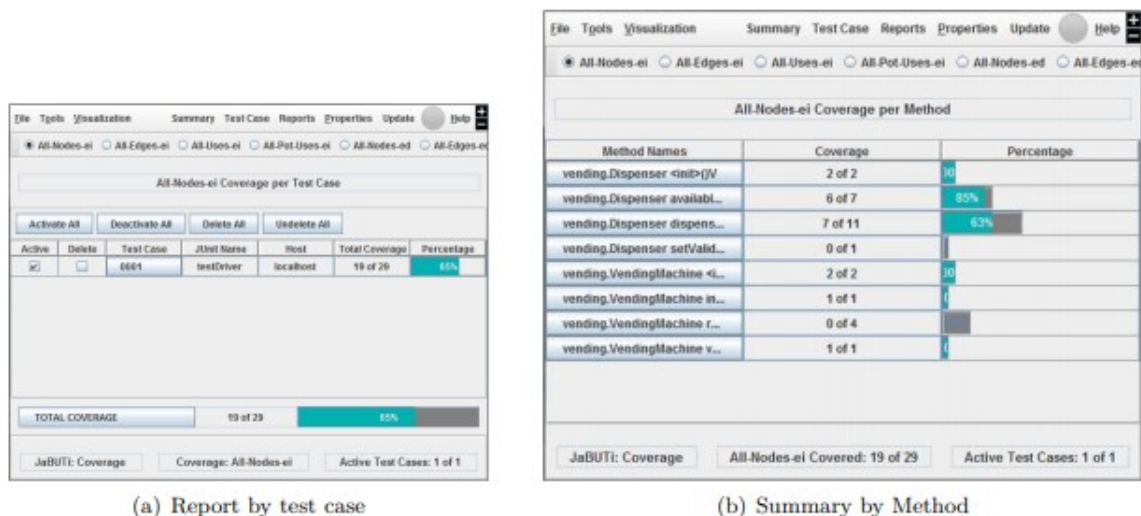


Figure 24: Updated coverage after test case input1.

Table 1: Adequate test set to the Dispenser component.

Name	Content	Correct output
input1	insertCoin vendItem 3	no
input2	insertCoin vendItem 18	yes
input3	insertCoin insertCoin vendItem 25	yes
input4	vendItem 3	yes
input5	insertCoin insertCoin vendItem 3	yes

Experiment No.: 2

Name of Experiment: 2. Write a program which read the first name and last name from console and matching with expected result by using JaBuTi.

Steps to perform Experiment:

Program Code:AreaOfCircle.java

```
import java.util.Scanner;

/**
 * An example program to read a Name from console input in Java
 */
public class Example {

    public static void main(String[] args) {
        System.out.print("Enter a Name : ");
        Scanner scanner = new Scanner(System.in);
        String inputString = scanner.nextLine();
        System.out.println("String read from console is : 
\n"+inputString);
    }
}
```

Note : Repeat all step according to Experiment 1 in JABUTI

Experiment No.: 3

Name of Experiment: Write a program which read the first name and last name from console and matching with expected result by using JaBuTi.

Steps to perform Experiment:

Program Code: qeqvation.java

```
public class qeqvation
{

    public static void main(String[] args) {

        // value a, b, and c
        double a = 2.3, b = 4, c = 5.6;
        double root1, root2;

        // calculate the determinant (b2 - 4ac)
        double determinant = b * b - 4 * a * c;

        // check if determinant is greater than 0
        if (determinant > 0) {

            // two real and distinct roots
            root1 = (-b + Math.sqrt(determinant)) / (2 * a);
            root2 = (-b - Math.sqrt(determinant)) / (2 * a);

            System.out.format("root1 = %.2f and root2 = %.2f", root1, root2);
        }

        // check if determinant is equal to 0
        else if (determinant == 0) {
```

```
// two real and equal roots
// determinant is equal to 0
// so -b + 0 == -b
root1 = root2 = -b / (2 * a);
System.out.format("root1 = root2 = %.2f;", root1);
}

// if determinant is less than zero
else {

    // roots are complex number and distinct
    double real = -b / (2 * a);
    double imaginary = Math.sqrt(-determinant) / (2 * a);
    System.out.format("root1 = %.2f+%.2fi", real, imaginary);
    System.out.format("\nroot2 = %.2f-%.2fi", real, imaginary);
}
}
}
```

Note : Repeat all step according to Experiment 1 in JABUTI

Experiment No.: 4

Name of Experiment: Write a program that reads commercial website URL from a url from file. you should expect that the URL starts with www and ends with .com. retrieve the name of the site and output it. For instance, if the user inputs www.yahoo.com, you should output yahoo. After that find the test cases and coverage using JaButi.

Steps to perform Experiment:

Program Code:

```
package mypkg;

import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

public class HelloServlet extends HttpServlet {

    @Override

    public void doGet(HttpServletRequest request, HttpServletResponse response)

        throws IOException, ServletException {

        // Set the response message's MIME type

        response.setContentType("text/html;charset=UTF-8");

        // Allocate a output writer to write the response message into the network socket

        PrintWriter out = response.getWriter();

        // Write the response message, in an HTML page

        try {

            out.println("<!DOCTYPE html>");

            out.println("<html><head>");
```

```
out.println("<meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>");
out.println("<title>Hello, World</title></head>");
out.println("<body>");
out.println("<h1>Hello, world!</h1>"); // says Hello
// Echo client's request information
out.println("<p>Request URI: " + request.getRequestURI() + "</p>");
out.println("<p>Protocol: " + request.getProtocol() + "</p>");
out.println("<p>PathInfo: " + request.getPathInfo() + "</p>");
out.println("<p>Remote Address: " + request.getRemoteAddr() + "</p>");
// Generate a random number upon each request
out.println("<p>A Random Number: <strong>" + Math.random() + "</strong></p>");
out.println("</body>");
out.println("</html>");
} finally {
    out.close(); // Always close the output writer
}
}
```

Note : Repeat all step according to Experiment 1 in JABUTI

Experiment No.:5

Name of Experiment: Write a program for a calculator and find the test case and coverage and Def-use-graph.

Steps to perform Experiment:

Program Code:

```
import java.util.Scanner;

class Main {

    public static void main(String[] args) {

        char operator;

        Double number1, number2, result;

        // create an object of Scanner class

        Scanner input = new Scanner(System.in);

        // ask users to enter operator

        System.out.println("Choose an operator: +, -, *, or /");

        operator = input.next().charAt(0);

        // ask users to enter numbers

        System.out.println("Enter first number");

        number1 = input.nextDouble();

        System.out.println("Enter second number");

        number2 = input.nextDouble();
```

```
switch (operator) {

    // performs addition between numbers
    case '+':
        result = number1 + number2;
        System.out.println(number1 + " + " + number2 + " = " + result);
        break;

    // performs subtraction between numbers
    case '-':
        result = number1 - number2;
        System.out.println(number1 + " - " + number2 + " = " + result);
        break;

    // performs multiplication between numbers
    case '*':
        result = number1 * number2;
        System.out.println(number1 + " * " + number2 + " = " + result);
        break;

    // performs division between numbers
    case '/':
        result = number1 / number2;
        System.out.println(number1 + " / " + number2 + " = " + result);
        break;
```

default:

```
System.out.println("Invalid operator!");
```

```
break;
```

```
}
```

```
input.close();
```

```
}
```

```
}
```

Note : Repeat all step according to Experiment 1 in JABUTI

Experiment No.: 6

Name of Experiment: Write a program that reads two words representing passwords from the java console and outputs the number of character in the smaller of the two. For example, if the words are open and sesame, then the output should be 4, the length of the shorter word, open. And test this program using JaButi.

Steps to perform Experiment:

Program Code:

```
import java.util.Scanner;

public class Exercise11 {

    public static final int PASSWORD_LENGTH = 8;

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        System.out.print(
            "1. A password must have at least eight characters.\n" +
            "2. A password consists of only letters and digits.\n" +
            "3. A password must contain at least two digits\n" +
            "Input a password (You are agreeing to the above Terms and Conditions.): ");
        String s = input.nextLine();

        if (is_Valid_Password(s)) {
            System.out.println("Password is valid: " + s);
        } else {
            System.out.println("Not a valid password: " + s);
        }
    }

    public static boolean is_Valid_Password(String password) {

        if (password.length() < PASSWORD_LENGTH) return false;

        int charCount = 0;
        int numCount = 0;
        for (int i = 0; i < password.length(); i++) {

            char ch = password.charAt(i);

            if (is_Numeric(ch)) numCount++;
            else if (is_Letter(ch)) charCount++;
        }
    }
}
```

```
        else return false;
    }

    return (charCount >= 2 && numCount >= 2);
}

public static boolean is_Letter(char ch) {
    ch = Character.toUpperCase(ch);
    return (ch >= 'A' && ch <= 'Z');
}

public static boolean is_Numeric(char ch) {

    return (ch >= '0' && ch <= '9');
}
}
```

Note : Repeat all step according to Experiment 1 in JABUTI

Experiment No.: 7

Name of Experiment: Analyze the performance of following website using JMeter.

Site	Website	Type
Amazon	Amazon.com	shopping
Flip kart	Flipkart.com	shopping
Railway reservation	Irctc.co.in	Ticket booking site
Train searching	Erail.in	Train searching

Steps to perform Experiment:

JMeter Introduction

JMeter is a software that can perform load test, performance-oriented business (functional) test, regression test, etc., on different protocols or technologies.

Stefano Mazzocchi of the Apache Software Foundation was the original developer of JMeter. He wrote it primarily to test the performance of Apache JServ (now called as Apache Tomcat project). Apache later redesigned JMeter to enhance the GUI and to add functional testing capabilities.

JMeter is a Java desktop application with a graphical interface that uses the Swing graphical API. It can therefore run on any environment / workstation that accepts a Java virtual machine, for example – Windows, Linux, Mac, etc.

The protocols supported by JMeter are –

- Web – HTTP, HTTPS sites 'web 1.0' web 2.0 (ajax, flex and flex-ws-amf)
- Web Services – SOAP / XML-RPC
- Database via JDBC drivers
- Directory – LDAP
- Messaging Oriented service via JMS
- Service – POP3, IMAP, SMTP
- FTP Service

JMeter Features

Following are some of the features of JMeter –

- Being an open source software, it is freely available.
- It has a simple and intuitive GUI.

- JMeter can conduct load and performance test for many different server types – Web - HTTP, HTTPS, SOAP, Database via JDBC, LDAP, JMS, Mail - POP3, etc.
- It is a platform-independent tool. On Linux/Unix, JMeter can be invoked by clicking on JMeter shell script. On Windows, it can be invoked by starting the jmeter.bat file.
- It has full Swing and lightweight component support (precompiled JAR uses packages javax.swing.*).
- JMeter store its test plans in XML format. This means you can generate a test plan using a text editor.
- Its full multi-threading framework allows concurrent sampling by many threads and simultaneous sampling of different functions by separate thread groups.
- It is highly extensible.
- It can also be used to perform automated and functional testing of the applications.

JMeter Working Techniques

JMeter simulates a group of users sending requests to a target server, and returns statistics that show the performance/functionality of the target server/application via tables, graphs, etc.

Take a look at the following figure that depicts how JMeter works –

JMeter is a framework for Java, so the very first requirement is to have JDK installed in your machine.

System Requirement

JDK	1.6 or above.
Memory	No minimum requirement.
Disk Space	No minimum requirement.
Operating System	No minimum requirement.

Step 1: Verify Java Installation

First of all, verify whether you have Java installed in your system. Open your console and execute one of the following **java** commands based on the operating system you are working on.

OS	Task	Command
Windows	Open Command Console	c:\> java -version
Linux	Open Command Terminal	\$ java -version
Mac	Open Terminal	machine: ~ joseph\$ java -version

If you have Java installed in your system, you would get an appropriate output based on the OS you are working on.

OS	Output
Windows	java version "1.7.0_25" Java(TM) SE Runtime Environment (build 1.7.0_25-b15) Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode)
Linux	java version "1.7.0_25" Java(TM) SE Runtime Environment (build 1.7.0_25-b15) Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode)
Mac	java version "1.7.0_25" Java(TM) SE Runtime Environment (build 1.7.0_25-b15) Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode)

If you do not have Java installed, install the Java Software Development Kit (SDK) from www.oracle.com/technetwork/java/javase/downloads/index.html. We are assuming Java 1.7.0_25 as the installed version for this tutorial.

Step 2: Set Java Environment

Set the **JAVA_HOME** environment variable to point to the base directory location, where Java is installed on your machine. For example –

OS	Output
Windows	Set the environment variable JAVA_HOME to C:\Program Files\Java\jdk1.7.0_25
Linux	export JAVA_HOME=/usr/local/java-current
Mac	export JAVA_HOME=/Library/Java/Home

Append Java compiler location to System Path.

OS	Output
----	--------

Windows	Append the string; C:\Program Files\Java\jdk1.7.0_25\bin to the end of the system variable, Path.
Linux	export PATH=\$PATH:\$JAVA_HOME/bin/
Mac	not required

Verify Java Installation using **java -version** command as explained above.

Step 3: Download JMeter

Download the latest version of JMeter from https://jmeter.apache.org/download_jmeter.cgi. For this tutorial, we downloaded *apache-jmeter-2.9* and copied it into C:\>JMeter folder.

The directory structure should look like as shown below –

- apache-jmeter-2.9
- apache-jmeter-2.9\bin
- apache-jmeter-2.9\docs
- apache-jmeter-2.9\extras
- apache-jmeter-2.9\lib\
- apache-jmeter-2.9\lib\ext
- apache-jmeter-2.9\lib\junit
- apache-jmeter-2.9\printable_docs

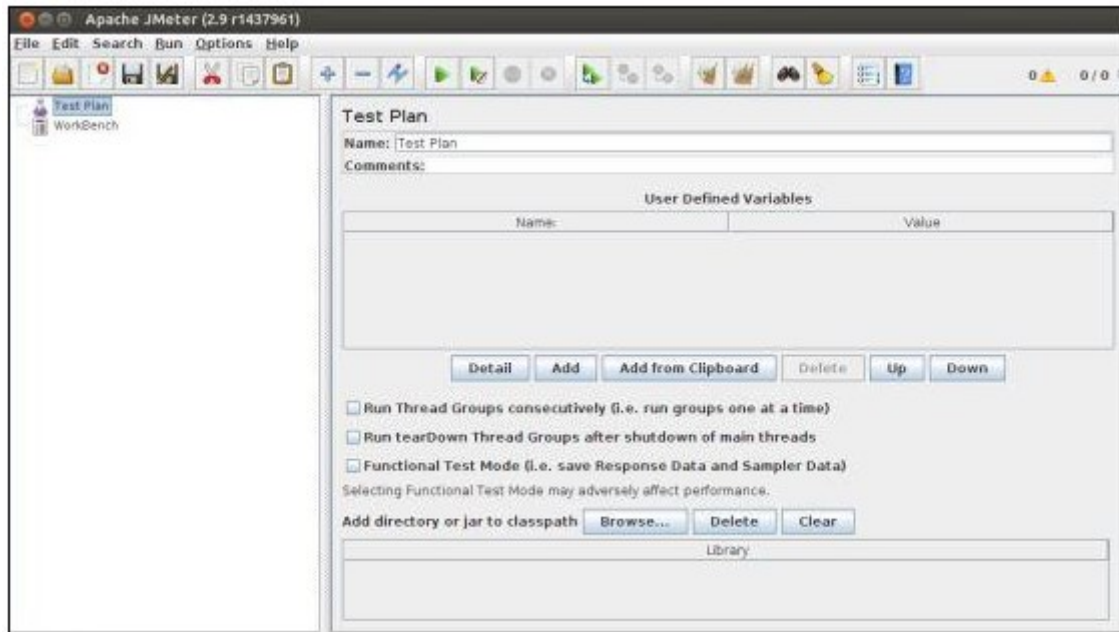
You can rename the parent directory (i.e. apache-jmeter-2.9) if you want, but do not change any of the sub-directory names.

Step 4: Run JMeter

After downloading JMeter, go to the *bin* directory. In this case, it is */home/manisha/apache-jmeter-2.9/bin*. Now click on the following –

OS	Output
Windows	jmeter.bat
Linux	jmeter.sh
Mac	jmeter.sh

After a short pause, the JMeter GUI should appear, which is a Swing application, as seen in the following screenshot –



This is the main page and the default page of the tool.

Apache JMeter - User's Manual - <https://jmeter.apache.org/usermanual/index.html>

Experiment No.: 8

Name of Experiment: Calculate the mutation score of programs given in 1(a) to 1 (f) using jumble Tool.

Steps to perform Experiment:

Introduction to Jumble :

The primary entry point to Jumble is a single Java class that takes as parameters a class to be mutation tested and one or more JUnit test classes. The output is a simple textual summary of running the tests. This is in the style of JUnit test output and includes an overall score of how many mutations were successfully caught as well as details about those mutations that are not. This includes the source line and the mutation performed. Variants of the output include a version compatible with Java development in emacs where it is possible to click on the line and go to the source line containing the mutation point. A plugin for the Eclipse IDE [10] has been provided. Currently it permits only jumbling of a single class. When running the mutation tests a separate JVM [11] is used to prevent runaway or system-exiting code disturbing the testing process. It is given a number which specifies the mutation to start at and it continues until all mutations have been tested or some catastrophic failure occurs. It has been very important to separate the test execution in this way for reasons discussed below. If a failure occurs in one mutation then the child JVM can be restarted and testing continues with the next mutation. Before running the mutation tests, a full run is done of the unit tests both to ensure that they all pass, to ensure that there are no environmental problems when running the tests in this way and to collect timing information to later detect mutations that lead to infinite loops. At Reel Two, Jumble is integrated with an internal continual integration system, on several source code repositories. Every fifteen minutes this checks out all the source code, clean compiles it and then runs all the unit tests for packages that have been modified. It also places all modified classes on a queue to be mutation tested. After the unit testing has been done, Jumble is used to test classes until the fifteen minute time limit is exceeded. Overnight more time is dedicated to mutation testing so that any tests that have been accumulated during the day can be cleared. The results of the Jumble tests for a project are presented in a web interface. These give results for individual classes and accumulate results over the package hierarchy. The web interface permits a manual override of the test queuing so that classes which have not been modified can be retested. This is sometimes necessary when the Jumble code itself has been modified or when a class needs to be

retested because of changes in other classes that it interacts with. In summary we typically use three different ways of running Jumble:

- directly from the command line where the class `com.reeltwo.jumble.Jumble` is provided with the name of the class to be tested and the class(es) which test it (among many other options). Executing directly from the distributed `jumble.jar` will achieve the same effect.
- from the Eclipse IDE where there is a menu item for a class which runs Jumble on that class (it is possible to configure other options for such runs)
- as part of a web based system where all classes are tested incrementally as they are committed and results accumulated in a set of webpages.

4. Jumble System

In this section we discuss individual parts of the Jumble system and the issues that arose in implementing them.

4.1 Bytecode Mutation

Jumble performs its mutations by directly modifying Java bytecodes using the BCEL package [12]. This allows the mutation testing to be done with no compilations. The bytecode translation approach has been used before. Mothra used an intermediate code form to store and execute mutants. The intermediate form was not a standard code form, however, and was directly designed to represent mutations, which were still done by analyzing source code. MuJava also used bytecode translation for structural mutations. Behavioural mutations however are still done with source code analysis using the MSG method [6]. The reason for this is that the behavioral mutations in MuJava were intended to be the same mutations as those implemented in Mothra. Thus, while MuJava manages to avoid multiple compilations, it still requires source code analysis to perform the mutations. As mentioned above, Jester does not use any sophisticated techniques and performs a compilation for every mutation. Basing the mutations entirely on bytecode also simplifies the implementation of Jumble and allows code to be tested even when the source code is not available. Jumble mutates all the Java bytecode instructions that can be mutated safely in a context-free way. That is, each instruction eligible for mutation is replaced by another instruction independently of the instructions around it. An instruction A can be replaced by an instruction B if A and B operate on the operand stack in the same way – they expect the same number and type of arguments on the stack before the operation and leave the same number

and type of arguments on the stack after operation. For example, the `iadd` (integer addition) instruction can be replaced by the `isub` (integer subtraction), since both pop the top two operands from the top of the stack and push the result of the computation (respectively the sum and difference of the two operands) onto the stack. Particularly helpful is BCEL's ability to generate bytecode for the negation of arbitrary conditionals. A wide range of bytecode instructions are mutated including conditionals, arithmetic operations including increments and decrements, and switch statements. As well inline constants, class pool constants, and return values are mutated. More details can be found in the "Mutations" link in [4]. The mutations are all implemented using the facilities of BCEL and within the limitations of BCEL's facilities the addition of new mutations is straightforward. Care needs to be taken to avoid mutating instructions in some parts of the code. For example, it makes no sense to mutate assertion statements. Detection of assertion statements is done by their reference to the class level flag which indicates when assertions are enabled. The pattern of code generated by the compiler is then used to detect the end of the assertion. Unfortunately such patterns are compiler dependent and care is needed when moving to new compilers and new JDK releases. Other code that needs to be excluded are conditionals generated when class constants are accessed, lengths for certain array allocations, and switch code generated for enumerations. A facility is provided to globally exclude certain named methods from mutation. In practice this is used to exclude main methods (this is coupled with a coding standard that main methods should be as short as possible and that they should call another method with such things as input and output files as parameters). Also excluded are "integrity" methods which can be called on a class to check that its internal state is currently consistent. These effectively function as postconditions and like assertions should be excluded. Jumble also mutates constants, both those that occur inline and those that occur in the constant pool. An integer constant x is transformed to $(-x + 1)$ which works correctly even when the integer represents a boolean. In Java boolean, short, and char datatypes, are implemented in bytecode as 32-bit ints. Consequently, Jumble cannot readily discriminate among these types. Constants in the constant pool (such as String literals) are also mutated. The constant pool for a class often contains entries not referenced by any line of code or only referenced by unmutateable code such as assertions. Care is taken not to modify such literals.

4.2 Class Loader

One feature of the BCEL which has been extremely useful in the development of Jumble is its customizable class loader. It allows classes to be modified as they are loaded into the JVM. The process is much simpler than creating a class loader from scratch, as the processing of the actual class file is done by the BCEL and the user only has to implement a `modifyClass` method to perform bytecode modifications on the fly. The Jumble class loader is derived from the BCEL class loader. Given the name of the class to modify (the target class) and the mutation number, it loads the class with the mutation inserted. All classes other than the target class are loaded normally with no modifications. The Jumble test suite is an extension of the JUnit test suite. It runs a JUnit test in a way suitable for Jumble testing, by running the tests until one fails. A “PASS” message is returned at that stage and no further tests are run. If no tests fail, a “FAIL” message is returned. The Jumble test suite is given the name of the test to run as a string and loads the test class dynamically (using `Class.forName()`). The Jumble test suite is intended to be loaded in the Jumble class loader so that the tests are run with a mutated class. When running several mutations inside a single JVM, a separate class loader must be used for each mutated version of the class being tested. To ensure independence between mutation tests, the entire set of (non-system) classes are reloaded in each class loader, and the classes being tested are not available directly from the system class loader. Heavy usage of class loaders in this way can use a lot of memory in the permgen space in the JVM, particularly for classes that make use of many thirdparty libraries. In practice it has been difficult to prevent increasing usage of the permgen space as the tests are run. This is dealt with in two ways. Firstly the child JVM that is used to run the tests is given additional permgen space when it is started. Secondly after each test is run a check is done to see if the available permgen space is nearly exhausted, if so the child process is terminated and restarted (it will then start at the next mutation). Some code remains difficult to run within the Jumble environment. Typically this involves code that attempts to directly access the system class loader (such code is usually incorrectly written, and unlikely to function when run in similar environments, such as within the Tomcat servlet container).

4.3 Detecting and Terminating Infinite Loops

Some mutations become stuck in an infinite loop. It is reasonable to consider an infinite loop as a failed test and hence mutation points which cause infinite loops can be considered tested. Thus Jumble needs to be able to detect infinite loops caused by mutations and terminate them. Infinite loops are detected by timing the original test running on a class without any mutations. Timing measurements are made using the `System.currentTimeMillis` method. This introduces two difficulties. Firstly, the granularity of the value returned from the method is dependent on the underlying operating system. Secondly, the value returned is a measure of elapsed time, not CPU time allocated to the current process. Thus, the time measures obtained are somewhat imprecise and nondeterministic. This is not a problem as long as the nondeterminism is accounted for. Each mutated test run can then be timed and the runtime can be periodically compared against a runtime limit and an infinite loop is considered detected if the limit is exceeded. The formula for the runtime limit is: $\text{RUNTIME LIMIT} = 10 \times \text{ORIGINAL RUNTIME} + 2 \text{ s}$ This formula is somewhat arbitrary but works reasonably well in practice. The most difficult situation is when a test is the first to be run, then effects such as classes being loaded and static code being executed can increase the apparent execution time. Once an infinite loop is detected, it needs to be terminated. One option is to run the mutation testing in a separate Java thread, and terminate the thread when an infinite loop is detected. The problem with this approach is that there is no inherently safe way of terminating a Java thread while being guaranteed to leave the rest of the system in a consistent state. Use of the `Thread.stop()` method is strongly discouraged. The method used in Jumble is to terminate the child JVM that is running the tests and for it to note that the test has been successful. Then the child JVM is restarted at the next mutation.

4.4 Applicability and Limitations

The Jumble system will run with code generated for Java 1.3 to 1.6. It has been run on code that uses multi-threading and concurrent processes. The biggest difficulties have been with systems that implement their own class loaders (see discussion above) and with variations of different compilers. The greatest difficulties with compilers is in detecting code such as assertions where the patterns may differ from compiler to compiler. Jumble has been successfully used with `javac`, `jikes` and the Java compiler in the Eclipse IDE.

Experiment No.: 9

Name of Experiment: Calculate the coverage analysis of programs given in 1 (a) to 1 (f) using Eclemma Free open source Tool.

Steps to perform Experiment:

Eclemma is a free Java code coverage tool for Eclipse, available under the Eclipse Public License. It brings code coverage analysis directly into the Eclipse workbench:

- **Fast develop/test cycle:** Launches from within the workbench like JUnit test runs can directly be analyzed for code coverage.
- **Rich coverage analysis:** Coverage results are immediately summarized and highlighted in the Java source code editors.
- **Non-invasive:** Eclemma does not require modifying your projects or performing any other setup.

Since version 2.0 Eclemma is based on the JaCoCo code coverage library. The Eclipse integration has its focus on supporting the individual developer in an highly interactive way. For automated builds please refer to JaCoCo documentation for integrations with other tools.

Originally Eclemma was inspired by and technically based on the great EMMA library developed by Vlad Roubtsov.

The update site for Eclemma is <http://update.eclemma.org/>. Eclemma is also available via the Eclipse Marketplace Client, simply search for "Eclemma".

Features

Launching

Eclemma adds a so called *launch mode* to the Eclipse workbench. It is called *Coverage* mode and works exactly like the existing *Run* and *Debug* modes. The *Coverage* launch mode can be activated from the *Run* menu or the workbench's toolbar:



Simply launch your applications or unit tests in the *Coverage* mode to collect coverage information. Currently the following launch types are supported:

- Local Java application
- Eclipse/RCP application
- Equinox OSGi framework
- JUnit test
- TestNG test
- JUnit plug-in test
- JUnit RAP test
- SWTBot test
- Scala application

Analysis

On request or after your target application has terminated code coverage information is automatically available in the Eclipse workbench:

- **Coverage overview:** The *Coverage* view lists coverage summaries for your Java projects, allowing drill-down to method level.

- **Source highlighting:** The result of a coverage session is also directly visible in the Java source editors. A customizable color code highlights fully, partly and not covered lines. This works for your own source code as well as for source attached to instrumented external libraries.

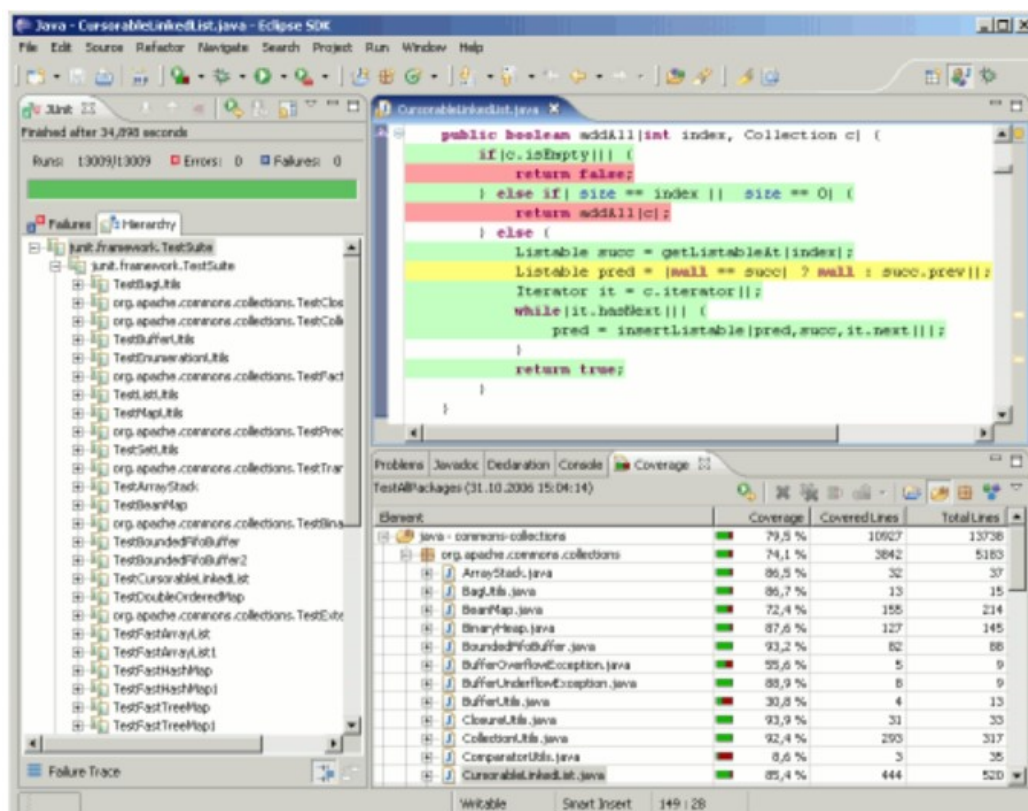
Additional features support analysis for your test coverage:

- **Different counters:** Select whether instructions, branches, lines, methods, types or cyclomatic complexity should be summarized.
- **Multiple coverage sessions:** Switching between coverage data from multiple sessions is possible.
- **Merge Sessions:** If multiple different test runs should be considered for analysis coverage sessions can easily be merged.

Import/Export

While Eclemma is primarily designed for test runs and analysis within the Eclipse workbench, it provides some import/export features.

- **Execution data import:** A wizard allows to import JaCoCo *.exec execution data files from external launches.
- **Coverage report export:** Coverage data can be exported in HTML, XML or CSV format or as JaCoCo execution data files (*.exec).



Eclemma - User Guide - <https://www.eclemma.org/userdoc/index.html>

Experiment No.: 10

Name of Experiment: Generate Test sequences and validate using Selenium tool for given websites below:

Site Amazon	Website Amazon.com	Type shopping
Flip kart	Flipkart.com	shopping
Railway reservation	Irctc.co.in	Ticket booking site
Train searching	Erail.in	Train searching

Steps to perform Experiment:

Selenium is an open-source and a portable automated software testing tool for testing web applications. It has capabilities to operate across different browsers and operating systems. Selenium is not just a single tool but a set of tools that helps testers to automate web-based applications more efficiently.

Creating Selenium IDE Tests

The following steps are involved in creating Selenium tests using IDE:

- ☐ Recording and adding commands in a test
- ☐ Saving the recorded test
- ☐ Saving the test suite
- ☐ Executing the recorded test

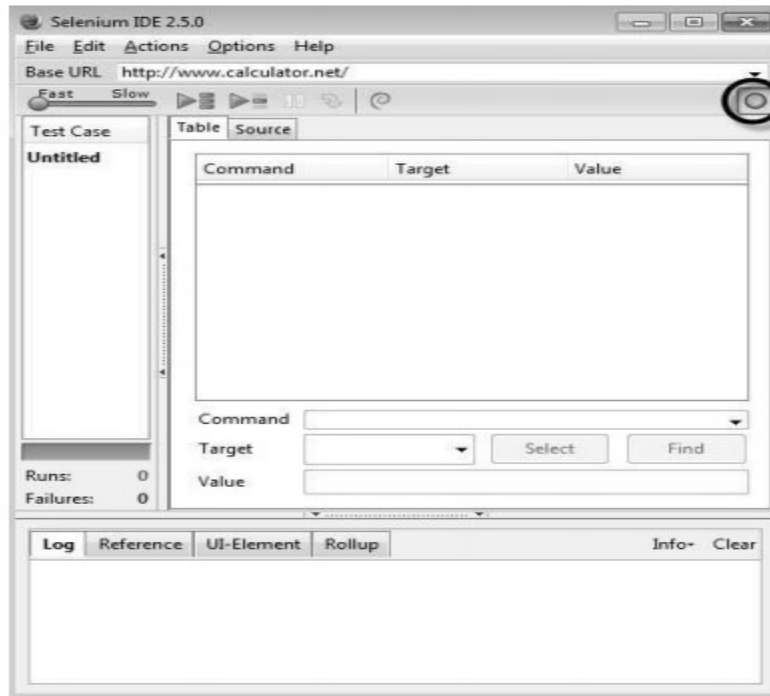
Recording and Adding Commands in a Test

We will use www.ncalculators.com to demonstrate the features of Selenium.

Step 1 : Launch the Firefox browser and navigate to the website –

<http://www.ncalculators.com/>

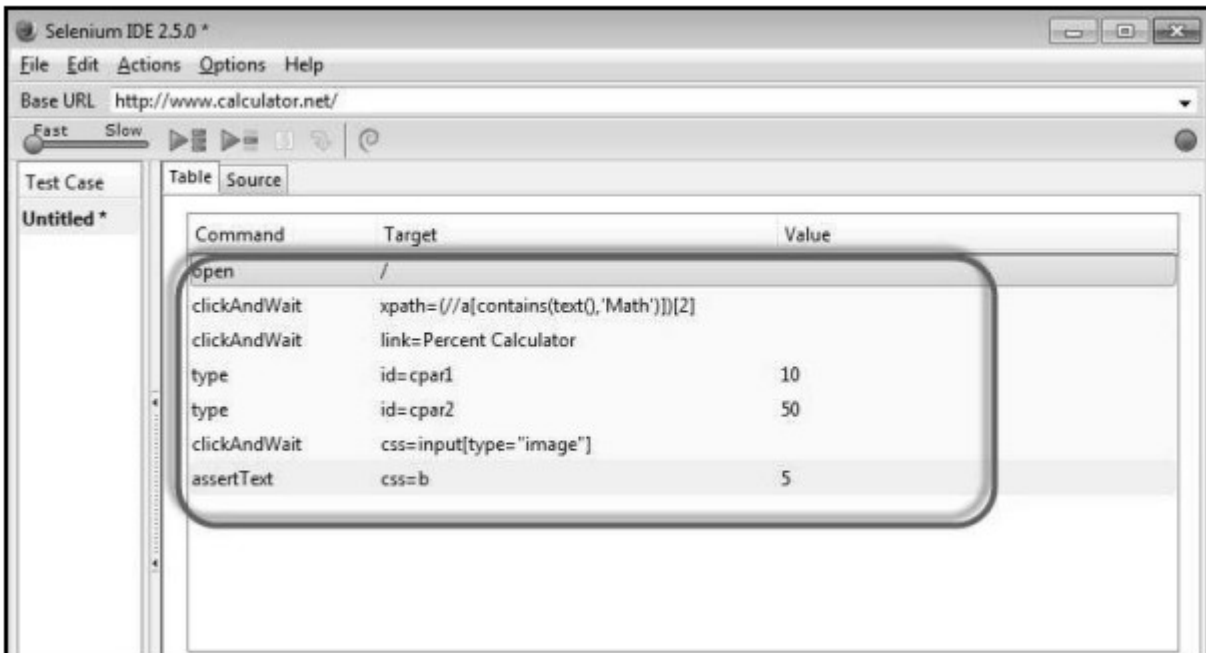
Step 2 : Open Selenium IDE from the Tools menu and press the record button that is on the top-right corner



Step 3 : Navigate to "Math Calculator" >> "Percent Calculator" >> enter "10" as number1 and 50 as number2 and click "calculate".

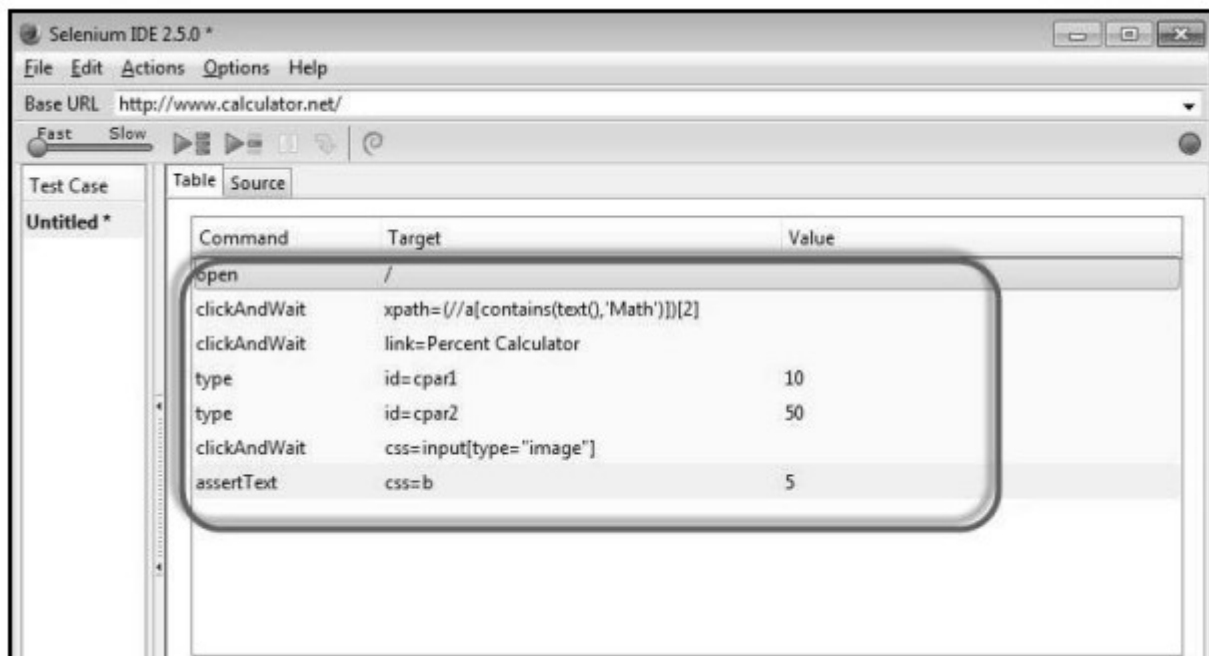


Step 5 : The recorded script is generated and the script is displayed as shown below.



Saving the Recorded Test

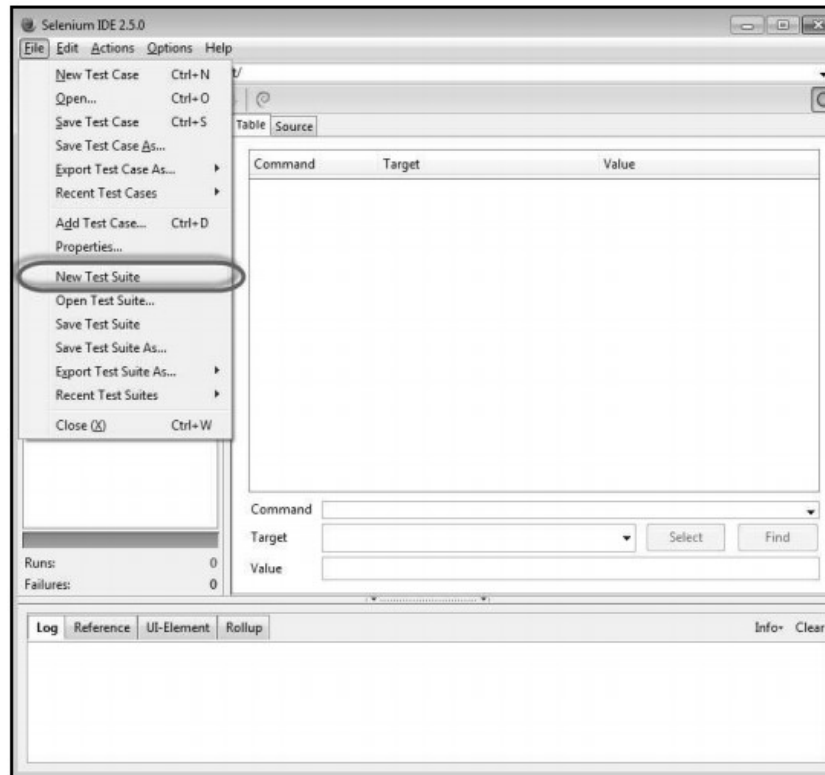
Step 1 : Save the Test Case by navigating to "File" >> "Save Test" and save the file in the location of your choice. The file is saved as .HTML as default. The test can also be saved with an extension htm, shtml, and xhtml



Saving the Test Suite

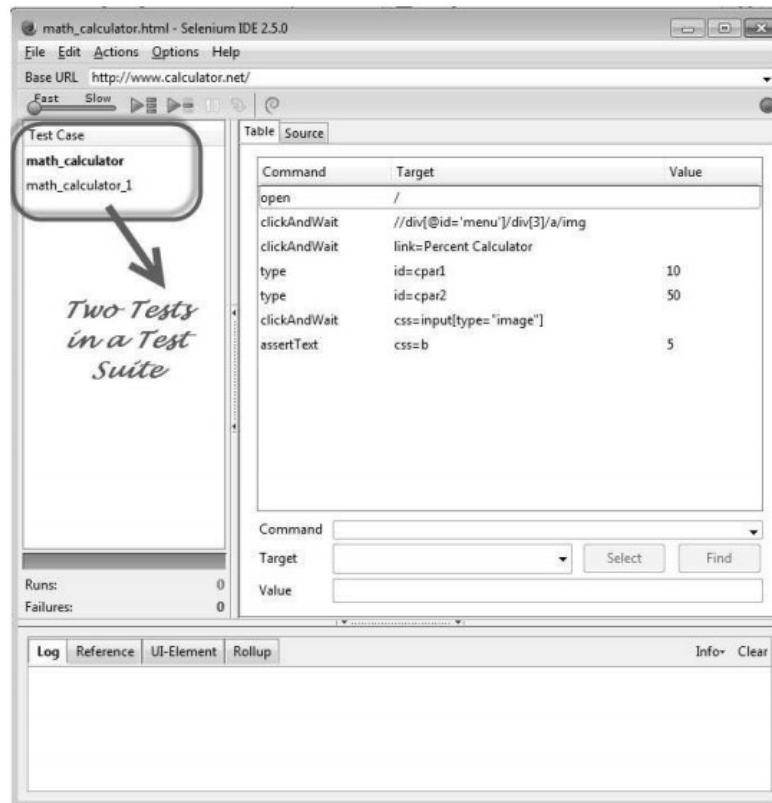
A test suite is a collection of tests that can be executed as a single entity.

Step 1 : Create a test suite by navigating to "File" >> "New Test Suite" as shown below.



Step 2 : The tests can be recorded one by one by choosing the option "New Test Case" from the "File" Menu.

Step 3 : The individual tests are saved with a name along with saving a "Test Suite"



Executing the Recorded Test

The recorded scripts can then be executed either by clicking "Play entire suite" or "Play current test" button in the toolbar.

Step 1 : The Run status can be seen in the status pane that displays the number of tests passed and failed.

Step 2 : Once a step is executed, the user can see the result in the "Log" Pane.

Step 3 : After executing each step, the background of the test step turns "Green" if passed and "Red" if failed as shown below

The screenshot displays the Selenium IDE 2.5.0 interface for a test case named 'math_calculator_1'. The 'Table' section lists the following commands and targets:

Command	Target	Value
open	/	
clickAndWait	xpath=//a[contains(text(), 'Math')][2]	
clickAndWait	link=Log Calculator	
type	name=yv	100
click	xpath=//input[@name='base'][2]	
clickAndWait	css=input[type="image"]	
assertText	css=b	2

A callout bubble points to the table with the text: "Highlighted in 'Green' if Passed or in 'RED' if Failed".

The 'Run Status' section shows:

- Runs: 2
- Failures: 0

The 'Log' section at the bottom shows the execution steps:

- [info] Executing: |open | / |
- [info] Executing: |clickAndWait | //div[@id='menu']/div[3]/a/img |
- [info] Executing: |clickAndWait | link=Percent Calculator |
- [info] Executing: |type | id=cpar1 | 10 |
- [info] Executing: |type | id=cpar2 | 50 |
- [info] Executing: |clickAndWait | css=input[type="image"] |
- [info] Executing: |assertText | css=b | 5 |
- [info] Changed test case
- [info] Executing: |open | / |

A callout bubble points to the log with the text: "Log of the the Execution".

Viva Questions

1. What is Exploratory Testing?

Exploratory testing is a hands-on approach in which testers are involved in minimum planning and maximum test execution. The planning involves the creation of a test charter, a short declaration of the scope of a short (1 to 2 hour) time-boxed test effort, the objectives and possible approaches to be used. The test design and test execution activities are performed in parallel typically without formally documenting the test conditions, test cases or test scripts. This does not mean that other, more formal testing techniques will not be used. For example, the tester may decide to use boundary value analysis but will think through and test the most important boundary values without necessarily writing them down. Some notes will be written during the exploratory-testing session so that a report can be produced afterward.

2. What is "use case testing"?

In order to identify and execute the functional requirement of an application from start to finish "use case" is used and the techniques used to do this is known as "Use Case Testing."

3. What is the difference between the STLC (Software Testing Life Cycle) and SDLC (Software Development Life Cycle)?

SDLC deals with development/coding of the software while STLC deals with validation and verification of the software

4. What is traceability matrix?

The relationship between test cases and requirements is shown with the help of a document. This document is known as a traceability matrix.

5. What is Equivalence partitioning testing?

Equivalence partitioning testing is a software testing technique which divides the application input test data into each partition at least once of equivalent data from which test cases can be derived. By this testing method, it reduces the time required for software testing.

6. What is white box testing and list the types of white box testing?

White box testing technique involves selection of test cases based on an analysis of the internal structure (Code coverage, branches coverage, paths coverage, condition coverage, etc.) of a component or system. It is also known as Code-Based testing or Structural testing. Different types of white box testing are

Statement Coverage

Decision Coverage

7. In white box testing, what do you verify?

In white box testing following steps are verified.

Verify the security holes in the code

Verify the incomplete or broken paths in the code

Verify the flow of structure according to the document specification

Verify the expected outputs

Verify all conditional loops in the code to check the complete functionality of the application

Verify the line by line coding and cover 100% testing

8. What is black box testing? What are the different black box testing techniques?

Black box testing is the software testing method which is used to test the software without knowing the internal structure of code or program. This testing is usually done to check the functionality of an application. The different black box testing techniques are

Equivalence Partitioning

Boundary value analysis

Cause-effect graphing

9. What is the difference between static and dynamic testing?

Static testing: During Static testing method, the code is not executed, and it is performed using the software documentation.

Dynamic testing: To perform this testing the code is required to be in an executable form.

10. What are verification and validation?

Verification is a process of evaluating software at the development phase. It helps you to decide whether the product of a given application satisfies the specified requirements. Validation is the process of evaluating software at the after the development process and to check whether it meets the customer requirements.

11. What are the different test levels?

There are four test levels

Unit/component/program/module testing

Integration testing

System testing

Acceptance testing

12. What is Integration testing?

Integration testing is a level of software testing process, where individual units of an application are combined and tested. It is usually performed after unit and functional testing.

13. What Test Plans consists of?

Test design, scope, test strategies, approach are various details that Test plan document consists of.

Test case identifier

Scope

Features to be tested

Features not to be tested

Test strategy & Test approach

Test deliverables

Responsibilities

Staffing and training

Risk and Contingencies

14. What is the difference between UAT (User Acceptance Testing) and System testing?

System Testing: System testing is finding defects when the system undergoes testing as a whole; it is also known as end-to-end testing. In such type of testing, the application suffers from beginning till the end.

UAT: User Acceptance Testing (UAT) involves running a product through a series of specific tests which determines whether the product will meet the needs of its users.

15. Mention the difference between Data Driven Testing and Retesting?

Retesting: It is a process of checking bugs that are actioned by the development team to verify that they are fixed.

Data Driven Testing (DDT): In data driven testing process, the application is tested with multiple test data. The application is tested with a different set of values.

16. What are the valuable steps to resolve issues while testing?

Record: Log and handle any problems which have happened

Report: Report the issues to higher level manager

Control: Define the issue management process

17. What is the difference between test scenarios, test cases, and test script?

Difference between test scenarios and test cases is that

Test Scenarios: A Test Scenario is any functionality that can be tested. It is also called Test Condition or Test Possibility.

Test Cases: It is a document that contains the steps that have to be executed; it has been planned earlier.

Test Script: It is written in a programming language and it's a short program used to test part of the functionality of the software system. In other words a written set of steps that should be performed manually.

18. What is Latent defect?

Latent defect: This defect is an existing defect in the system which does not cause any failure as the exact set of conditions has never been met

19. What are the two parameters which can be useful to know the quality of test execution?

To know the quality of test execution, we can use two parameters

Defect reject ratio

Defect leakage ratio

Manual Testing Interview Questions

20. What is the function of the software testing tool "phantom"?

Phantom is a freeware and is used for windows GUI automation scripting language. It allows us to take control of windows and functions automatically. It can simulate any combination of keystrokes and mouse clicks as well as menus, lists and more.

21. Explain what Test Deliverables is?

Test Deliverables are a set of documents, tools and other components that have to be developed and maintained in support of testing.

There are different test deliverables at every phase of the software development lifecycle

Before Testing

During Testing

After the Testing

22. What is mutation testing?

Mutation testing is a technique to identify if a set of test data or test case is useful by intentionally introducing various code changes (bugs) and retesting with original test data/ cases to determine if the bugs are detected.

23. What all things you should consider before selecting automation tools for the AUT?

Technical Feasibility

Complexity level

Application stability

Test data

Application size

Re-usability of automated scripts

Execution across environment

24. How will you conduct Risk Analysis?

For the risk analysis following steps need to be implemented

Finding the score of the risk

Making a profile for the risk

Changing the risk properties

Deploy the resources of that test risk

Making a database of risk

25. What are the categories of debugging?

Categories for debugging

Brute force debugging

Backtracking

Cause elimination

Program Slicing

Fault tree analysis

26. What is fault masking explain with example?

When the presence of one defect hides the presence of another defect in the system, it is known as fault masking.

Example: If the "Negative Value" cause a firing of unhandled system exception, the developer will prevent the negative values input. This will resolve the issue and hide the defect of unhandled exception firing.

27. Explain what Test Plan is? What is the information that should be covered in Test Plan?

A test plan can be defined as a document describing the scope, approach, resources, and schedule of testing activities and a test plan should cover the following details.

Test Strategy

Test Objective

Exit/Suspension Criteria

Resource Planning

Test Deliverables

28. How can you eliminate the product risk in your project?

It helps you to eliminate product risk in your project, and there is a simple yet crucial step that can reduce the product risk in your project.

Investigate the specification documents

Have discussions about the project with all stakeholders including the developer

As a real user walk around the website

29. What is the common risk that leads to project failure?

The common risk that leads to a project failure are

Not having enough human resource

Testing Environment may not be set up properly

Limited Budget

Time Limitations

30. On what basis you can arrive at an estimation for your project?

To estimate your project, you have to consider the following points

Divide the whole project into the smallest tasks

Allocate each task to team members

Estimate the effort required to complete each task

Validate the estimation

31. Explain how you would allocate a task to team members?

Task	Member
------	--------

Analyze software requirement specification	
--	--

All the members	
-----------------	--

Create the test specification	
-------------------------------	--

Tester/Test Analyst	
---------------------	--

Build up the test environment	
-------------------------------	--

Test administrator	
--------------------	--

Execute the test cases	
------------------------	--

Tester, a Test administrator	
------------------------------	--

Report defects	
----------------	--

Tester	
--------	--

32. Explain what is testing type and what are the commonly used testing type?

To get an expected test outcome, a standard procedure is followed which is referred to as Testing Type.

Commonly used testing types are

Unit Testing: Test the smallest code of an application

API Testing: Testing API created for the application

Integration Testing: Individual software modules are combined and tested

System Testing: Complete testing of the system

Install/UnInstall Testing: Testing done from the point of client/customer view

Agile Testing: Testing through Agile technique

33. While monitoring your project what all things you have to consider?

The things that have to be taken in considerations are

Is your project on schedule

Are you over budget

Are you working towards the same career goal

Have you got enough resources

Are there any warning signs of impending problems

Is there any pressure from management to complete the project sooner

34. What are the common mistakes which create issues?

Matching resources to wrong projects

Test manager lack of skills

Not listening to others

Poor Scheduling

Underestimating

Ignoring the small problems

Not following the process

35. What does a typical test report contain? What are the benefits of test reports?

A test report contains the following things:

Project Information

Test Objective

Test Summary

Defect

The benefits of test reports are:

Current status of project and quality of product are informed

If required, stakeholder and customer can take corrective action

A final document helps to decide whether the product is ready for release

36. What is test management review and why it is important?

Management review is also referred to as Software Quality Assurance or SQA. SQA focusses more on the software process rather than the software work products. It is a set of activities designed to make sure that the project manager follows the standard process. SQA helps test manager to benchmark the project against the set standards.

37. What are the best practices for software quality assurance?

The best practices for an effective SQA implementation is

Continuous Improvement

Documentation

Tool Usage

Metrics

Responsibility by team members

Experienced SQA auditors

38. When is RTM (Requirement Traceability Matrix) prepared?

RTM is prepared before test case designing. Requirements should be traceable from review activities.

39. What is the difference between Test matrix and Traceability matrix?

Test Matrix: Test matrix is used to capture actual quality, effort, the plan, resources and time required to capture all phases of software testing

Traceability Matrix: Mapping between test cases and customer requirements is known as Traceability Matrix

40. In manual testing what are stubs and drivers?

Both stubs and drivers are part of incremental testing. In incremental testing, there are two approaches namely bottom-up and top-down approach. Drivers are used in bottom-up testing and stub is used for a top-down approach. In order to test the main module, the stub is used, which is a dummy code or program.

41. What is the step you would follow once you find the defect?

Once a defect is found you would follow the step

- a) Recreate the defect
- b) Attach the screenshot
- c) Log the defect

42. Explain what is "Test Plan Driven" or "Key Word Driven" method of testing?

This technique uses the actual test case document developed by testers using a spreadsheet containing special "key Words". The key words control the processing.

43. What is the DFD (Data Flow Diagram)?

When a "flow of data" through an information system is graphically represented, then it is known as Data Flow Diagram. It is also used for the visualization of data processing.

44. Explain what LCSAJ is?

LCSAJ stands for 'linear code sequence and jump.' It consists of the following three items

- a) Start of the linear sequence of executable statements
- b) End of the linear sequence
- c) The target line to which control flow is transferred at the end of the linear sequence

45. Explain what N+1 testing is?

The variation of regression testing is represented as N+1. In this technique, the testing is performed in multiple cycles in which errors found in test cycle 'N' are resolved and re-tested in test cycle N+1. The cycle is repeated unless there are no errors found.

46. What is Fuzz testing and when it is used?

Fuzz testing is used to detect security loopholes and coding errors in software. In this technique, random data is added to the system in an attempt to crash the system. If vulnerability persists, a tool called fuzz tester is used to determine potential causes. This technique is more useful for bigger projects but only detects a major fault.

47. Mention what the main advantages of statement coverage metric of software testing are?

The benefit of statement coverage metric is that

- a) It does not require processing source code and can be applied directly to object code
- b) Bugs are distributed evenly through the code, due to which percentage of executable statements covered reflects the percentage of faults discovered

48. How to generate test cases for "replace a string" method?

- a) If characters in new string > characters in the previous string. None of the characters should get truncated
- b) If characters in new string < characters in the previous string. Junk characters should not be added
- c) Spaces after and before the string should not be deleted
- d) String should be replaced only for the first occurrence of the string

49. How will you handle a conflict amongst your team members?

I will talk individually to each person and note their concerns

I will find a solution to the common problems raised by team members

I will hold a team meeting, reveal the solution and ask people to co-operate

50. Mention what are the categories of defects?

Mainly there are three defect categories

Wrong: When a requirement is implemented incorrectly

Missing: It is a variance from the specification, an indication that a specification was not implemented or a requirement of the customer is not met

Extra: A requirement incorporated into the product that was not given by the end customer. It is considered as a defect because it is a variance from the existing requirements

51. Explain how does a test coverage tool work?

The code coverage testing tool runs parallel while performing testing on the actual product. The code coverage tool monitors the executed statements of the source code. When the final testing is done, we get a complete report of the pending statements and also get the coverage percentage.

52. Mention what the difference between a "defect" and a "failure" in software testing is?

In simple terms when a defect reaches the end customer, it is called a failure while the defect is identified internally and resolved; then it is referred to as a defect.

53. Explain how to test documents in a project that span across the software development lifecycle?

The project span across the software development lifecycle in the following manner

Central/Project test plan: It is the main test plan that outlines the complete test strategy of the project. This plan is used till the end of the software development lifecycle

Acceptance test plan: This document begins during the requirement phase and is completed at the final delivery

System test plan: This plan starts during the design plan and proceeds until the end of the project

Integration and Unit test plan: Both these test plans start during the execution phase and last until the final delivery

54. Explain which test cases are written first black boxes or white boxes?

Black box test cases are written first as to write black box test cases; it requires project plan and requirement document all these documents are easily available at the beginning of the project.

While writing white box test cases requires more architectural understanding and is not available at the start of the project.

55. Explain what the difference between latent and masked defects is?

Latent defect: A latent defect is an existing defect that has not caused a failure because the sets of conditions were never met

Masked defect: It is an existing defect that has not caused a failure because another defect has prevented that part of the code from being executed

56. Mention what bottom-up testing is?

Bottom-up testing is an approach to integration testing, where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested.

57. Mention what the different types of test coverage techniques are?

Different types of test coverage techniques include

Statement Coverage: It verifies that each line of source code has been executed and tested

Decision Coverage: It ensures that every decision in the source code is executed and tested

Path Coverage: It ensures that every possible route through a given part of the code is executed and tested

58. Mention what the meaning of breath testing is?

Breath testing is a test suite that exercises the full functionality of a product but does not test features in detail

59. Explain what the meaning of Code Walk Through is?

Code Walk Through is the informal analysis of the program source code to find defects and verify coding techniques

60. Mention what the basic components of defect report format are?

The essential components of defect report format include

Project Name

Module Name

Defect detected on

Defect detected by

Defect ID and Name

Snapshot of the defect

Priority and Severity status

Defect resolved by

Defect resolved on

61. Mention what the purpose behind doing end-to-end testing is?

End-to-end testing is done after functional testing. The purpose behind doing end-to-end testing is that

To validate the software requirements and integration with external interfaces

Testing application in real-world environment scenario

Testing of interaction between application and database

62. Explain what it means by test harness?

A test harness is configuring a set of tools and test data to test an application in various conditions, and it involves monitoring the output with expected output for correctness.

63. Explain in a testing project what testing activities would you automate?

In testing project testing activities, you would automate are

Tests that need to be run for every build of the application

Tests that use multiple data for the same set of actions

Identical tests that need to be executed using different browsers

Mission critical pages

A transaction with pages that do not change in a short time

64. What is the MAIN benefit of designing tests early in the life cycle?

It helps prevent defects from being introduced into the code.

65. What is risk-based testing?

Risk-based Testing is the term used for an approach to creating a Test Strategy that is based on prioritizing tests by risk. The basis of the approach is a detailed risk analysis and prioritizing of risks by risk level. Tests to address each risk are then specified, starting with the highest risk first.

66. What is the KEY difference between preventative and reactive approaches to testing?

Preventative tests are designed early; reactive tests are designed after the software has been produced.

67. What is the purpose of exit criteria?

The purpose of exit criteria is to define when a test level is completed.

68. What determines the level of risk?

The likelihood of an adverse event and the impact of the event determine the level of risk.

69. When is used Decision table testing?

Decision table testing is used for testing systems for which the specification takes the form of rules or cause-effect combinations. In a decision table, the inputs are listed in a column, with the outputs in the same column but below the inputs. The remainder of the table explores combinations of inputs to define the outputs produced.

Learn More About Decision Table Testing Technique in the Video Tutorial [here](#)

70. Why we use decision tables?

The techniques of equivalence partitioning and boundary value analysis are often applied to specific situations or inputs. However, if different combinations of inputs result in different actions being taken, this can be more difficult to show using equivalence partitioning and boundary value analysis, which tend to be more focused on the user interface. The other two specification-based techniques, decision tables, and state transition testing are more focused on business logic or business rules. A decision table is a good way to deal with combinations of things (e.g., inputs). This technique is sometimes also referred to as a 'cause-effect' table. The reason for this is that there is an associated logic diagramming technique called 'cause-effect graphing' which was sometimes used to help derive the decision table

71. What is the MAIN objective when reviewing a software deliverable?

To identify defects in any software work product.

72. Which of the following defines the expected results of a test? Test case specification or test design specification.

Test case specification defines the expected results of a test.

73. What is the benefit of test independence?

It avoids author bias in defining effective tests.

74. As part of which test process do you determine the exit criteria?

The exit criteria are determined on the bases of 'Test Planning'.

75. What is Alpha testing?

Pre-release testing by end user representatives at the developer's site.

76. What is beta testing?

Testing performed by potential customers at their own locations.

77. Mention what the difference between Pilot and Beta testing is?

The difference between a pilot and beta testing is that pilot testing is actually done using the product by the group of users before the final deployment, and in beta testing, we do not input real data, but it is installed at the end customer to validate if the product can be used in production.

78. Given the following fragment of code, how many tests are required for 100% decision coverage?

```
if width > length
  thenbiggest_dimension = width
  if height > width
    thenbiggest_dimension = height
  end_if
elsebiggest_dimension = length
  if height > length
    thenbiggest_dimension = height
  end_if
end_if
4
```

79. You have designed test cases to provide 100% statement and 100% decision coverage for the following fragment of code. if width > length then biggest_dimension = width else biggest_dimension = length end_if The following has been added to the bottom of the code

fragment above. print "Biggest dimension is " & biggest_dimension print "Width: " & width print "Length: " & length How many more test cases are required?

None, existing test cases can be used.

80. What is the difference between Testing Techniques and Testing Tools?

Testing technique: – Is a process for ensuring that some aspects of the application system or unit functions properly there may be few techniques but many tools.

Testing Tools: – Is a vehicle for performing a test process. The tool is a resource to the tester, but itself is insufficient to conduct testing

Learn More About Testing Tools here

81. We use the output of the requirement analysis, the requirement specification as the input for writing ...

User Acceptance Test Cases

82. Repeated Testing of an already tested program, after modification, to discover any defects introduced or uncovered as a result of the changes in the software being tested or in another related or unrelated software component:

Regression Testing

83. A wholesaler sells printer cartridges. The minimum order quantity is 5. There is a 20% discount for orders of 100 or more printer cartridges. You have been asked to prepare test cases using various values for the number of printer cartridges ordered. Which of the following groups contain three test inputs that would be generated using Boundary Value Analysis?

4, 5, 99

84. What is component testing?

Component testing, also known as unit, module, and program testing, searches for defects in and verifies the functioning of software (e.g., modules, programs, objects, classes, etc.) that are separately testable. Component testing may be done in isolation from the rest of the system depending on the context of the development life cycle and the system. Most often stubs and drivers are used to replace the missing software and simulate the interface between the software components simply. A stub is called from the software component to be tested; a driver calls a component to be tested.

Here is an awesome video on Unit Testing

85. What is functional system testing?

Testing the end to end functionality of the system as a whole is defined as a functional system testing.

86. What are the benefits of Independent Testing?

Independent testers are unbiased and identify different defects at the same time.

87. In a REACTIVE approach to testing when would you expect the bulk of the test design work to be begun?

The bulk of the test design work begun after the software or system has been produced.

88. What are the different Methodologies in Agile Development Model?

There are currently seven different agile methodologies that I am aware of:

Extreme Programming (XP)

Scrum

Lean Software Development

Feature-Driven Development

Agile Unified Process

Crystal

Dynamic Systems Development Model (DSDM)

89. Which activity in the fundamental test process includes evaluation of the testability of the requirements and system?

A 'Test Analysis' and 'Design' includes evaluation of the testability of the requirements and system.

90. What is typically the MOST important reason to use risk to drive testing efforts?

Because testing everything is not feasible.

91. What is random/monkey testing? When is it used?

Random testing is often known as monkey testing. In such type of testing data is generated randomly often using a tool or automated mechanism. With this randomly generated input, the system is tested, and results are analyzed accordingly. These testing are less reliable; hence it is normally used by the beginners and to see whether the system will hold up under adverse effects.

92. Which of the following are valid objectives for incident reports?

Provide developers and other parties with feedback about the problem to enable identification, isolation, and correction as necessary.

Provide ideas for test process improvement.

Provide a vehicle for assessing tester competence.

Provide testers with a means of tracking the quality of the system under test.

93. Consider the following techniques. Which are static and which are dynamic techniques?

Equivalence Partitioning.

Use Case Testing.

Data Flow Analysis.

Exploratory Testing.

Decision Testing.

Inspections.

Data Flow Analysis and Inspections are static; Equivalence Partitioning, Use Case Testing, Exploratory Testing and Decision Testing are dynamic.

94. Why are static testing and dynamic testing described as complementary?

Because they share the aim of identifying defects but differ in the types of defect they find.

95. What are the phases of a formal review?

In contrast to informal reviews, formal reviews follow a formal process. A typical formal review process consists of six main steps:

Planning

Kick-off

Preparation

Review meeting

Rework

Follow-up.

96. What is the role of moderator in the review process?

The moderator (or review leader) leads the review process. He or she determines, in co-operation with the author, the type of review, approach and the composition of the review team. The

moderator performs the entry check and the follow-up on the rework, in order to control the quality of the input and output of the review process. The moderator also schedules the meeting, disseminates documents before the meeting, coaches other team members, paces the meeting, leads possible discussions and stores the data that is collected.

Learn More about Review process in Video Tutorial [here](#)

97. What is an equivalence partition (also known as an equivalence class)?

An input or output ranges of values such that only one value in the range becomes a test case.

98. When should configuration management procedures be implemented?

During test planning.

99. A Type of Functional Testing, which investigates the functions relating to the detection of threats, such as virus from malicious outsiders?

Security Testing

100. Testing wherein we subject the target of the test, to varying workloads to measure and evaluate the performance behaviors and the ability of the target and the test to continue to function properly under these different workloads?

Load Testing

101. Testing activity which is performed to expose defects in the interfaces and in the interaction between integrated components is?

Integration Level Testing

102. What are the Structure-based (white-box) testing techniques?

Structure-based testing techniques (which are also dynamic rather than static) use the internal structure of the software to derive test cases. They are commonly called 'white-box' or 'glass-box' techniques (implying you can see into the system) since they require knowledge of how the software is implemented, that is, how it works. For example, a structural technique may be concerned with exercising loops in the software. Different test cases may be derived to exercise the loop once, twice, and many times. This may be done regardless of the functionality of the software.

103. When should "Regression Testing" be performed?

After the software has changed or when the environment has changed Regression testing should be performed.

104. What is negative and positive testing?

A negative test is when you put in an invalid input and receives errors. While positive testing is when you put in a valid input and expect some action to be completed in accordance with the specification.

105. What is the purpose of a test completion criterion?

The purpose of test completion criterion is to determine when to stop testing

106. What can static analysis NOT find?

For example memory leaks.

107. What is the difference between re-testing and regression testing?

Re-testing ensures the original fault has been removed; regression testing looks for unexpected side effects.

108. What are the Experience-based testing techniques?

In experience-based techniques, people's knowledge, skills, and background are a prime contributor to the test conditions and test cases. The experience of both technical and business people is important, as they bring different perspectives to the test analysis and design process. Due to previous experience with similar systems, they may have insights into what could go wrong, which is very useful for testing.

109. What type of review requires formal entry and exit criteria, including metrics?

Inspection

110. Could reviews or inspections be considered part of testing?

Yes, because both help detect faults and improve quality.

111. An input field takes the year of birth between 1900 and 2004 what the boundary values for testing this field are?

1899,1900,2004,2005

112. Which of the following tools would be involved in the automation of regression test? a. Data tester b. Boundary tester c. Capture/Playback d. Output comparator.

d. Output comparator

113. To test a function, what has to write a programmer, which calls the function to be tested and pass test data.

Driver

114. What is the one Key reason why developers have difficulty testing their own work?

Lack of Objectivity

115. "How much testing is enough?"

The answer depends on the risk for your industry, contract and special requirements.

116. When should testing be stopped?

It depends on the risks for the system being tested. There are some criteria based on which you can stop testing.

Deadlines (Testing, Release)

Test budget has been depleted

Bug rate fall below a certain level

Test cases completed with certain percentage passed

Alpha or beta periods for testing ends

Coverage of code, functionality or requirements are met to a specified point

117. Which of the following is the primary purpose of the integration strategy for integration testing in the small?

The primary purpose of the integration strategy is to specify which modules to combine when and how many at once.

118. What are semi-random test cases?

Semi-random test cases are nothing, but when we perform random test cases and do equivalence partitioning to those test cases, it removes redundant test cases, thus giving us semi-random test cases.

119. Given the following code, which statement is true about the minimum number of test cases required for full statement and branch coverage?

```
Read p
```

```
Read q
```

```
IF p+q> 100
```

```
THEN Print "Large"
```

```
ENDIF
```

```
IF p > 50
```

```
THEN Print "p Large"
```

```
ENDIF
```

1 test for statement coverage, 2 for branch coverage

120. Which review is normally used to evaluate a product to determine its suitability for the intended use and to identify discrepancies?

Technical Review.

121. Faults found should be originally documented by whom?

By testers.

122. Which is the current formal world-wide recognized documentation standard?

There isn't one.

123. Which of the following is the review participant who has created the item to be reviewed?

Author

124. A number of critical bugs are fixed in software. All the bugs are in one module, related to reports. The test manager decides to do regression testing only on the reports module.

Regression testing should be done on other modules as well because fixing one module may affect other modules.

125. Why does the boundary value analysis provide good test cases?

Because errors are frequently made during programming of the different cases near the 'edges' of the range of values.

126. What makes an inspection different from other review types?

It is led by a trained leader, uses formal entry and exit criteria and checklists.

127. Why can be tester dependent on configuration management?

Because configuration management assures that we know the exact version of the testware and the test object.

128. What is V-Model?

A software development model that illustrates how testing activities integrate with software development phases

129. What is maintenance testing?

Triggered by modifications, migration or retirement of existing software

130. What is test coverage?

Test coverage measures in some specific way the amount of testing performed by a set of tests (derived in some other way, e.g., using specification-based techniques). Wherever we can count things and can tell whether or not each of those things has been tested by some test, then we can measure coverage.

131. Why is incremental integration preferred over "big bang" integration?

Because incremental integration has better early defects screening and isolation ability

132. What is called the process starting with the terminal modules?

Bottom-up integration

133. During which test activity could fault be found most cost-effectively?

During test planning

134. The purpose of the requirement phase is

To freeze requirements, to understand user needs, to define the scope of testing

135. Why we split testing into distinct stages?

We split testing into distinct stages because of the following reasons,

Each test stage has a different purpose

It is easier to manage to test in stages

We can run different test into different environments

Performance and quality of the testing is improved using phased testing

136. What is DRE?

In order to measure test effectiveness, a powerful metric is used to measure test effectiveness known as DRE (Defect Removal Efficiency) From this metric we would know how many bugs we have found from the set of test cases. The formula for calculating DRE is

$$\text{DRE} = \frac{\text{Number of bugs while a testing}}{\text{number of bugs while testing} + \text{number of bugs found by a user}}$$

137. Which of the following is likely to benefit most from the use of test tools providing test capture and replay facilities? a) Regression testing b) Integration testing c) System testing d) User acceptance testing

Regression testing

138. How would you estimate the amount of re-testing likely to be required?

Metrics from previous similar projects and discussions with the development team

139. What studies data flow analysis?

The use of data on paths through the code.

140. What is failure?

Failure is a departure from specified behavior.

141. What are Test comparators?

Is it really a test if you put some inputs into some software, but never look to see whether the software produces the correct result? The essence of testing is to check whether the software produces the correct result and to do that, and we must compare what the software produces to what it should produce. A test comparator helps to automate aspects of that comparison.

142. Who is responsible for document all the issues, problems and open point that were identified during the review meeting

Scribe

143. What is the main purpose of Informal review

An inexpensive way to get some benefit

144. What is the purpose of test design technique?

Identifying test conditions and Identifying test cases

145. When testing a grade calculation system, a tester determines that all scores from 90 to 100 will yield a grade of A, but scores below 90 will not. This analysis is known as:

Equivalence partitioning

146. A test manager wants to use the resources available for the automated testing of a web application. The best choice is Tester, test automation, web specialist, DBA

147. During the testing of a module tester, 'X' found a bug and assigned it to a developer. But developer rejects the same, saying that it's not a bug. What 'X' should do?

Send the detailed information of the bug encountered and check the reproducibility

148. A type of integration testing in which software elements, hardware elements, or both are combined all at once into a component or an overall system, rather than in stages.

Big-Bang Testing

149. In practice, which Life Cycle model may have more, fewer or different levels of development and testing, depending on the project and the software product. For example, there may be component integration testing after component testing, and system integration testing after system testing.

V-Model

150. Which technique can be used to achieve input and output coverage? It can be applied to human input, input via interfaces to a system, or interface parameters in integration testing.

Equivalence partitioning

151. "This life cycle model is driven by schedule and budget risks" This statement is best suited for.

V-Model

152. In which order should tests be run?

The most important one must be tested first

153. The later in the development life cycle a fault is discovered, the more expensive it is to fix. Why?

The fault has been built into more documentation, code, tests, etc

154. What is Coverage measurement?

It is a partial measure of test thoroughness.

155. What is Boundary value testing?

Test boundary conditions on, below and above the edges of input and output equivalence classes. For instance, let say a bank application where you can withdraw maximum Rs.20,000 and a minimum of Rs.100, so in boundary value testing we test only the exact boundaries, rather than hitting in the middle. That means we test above the maximum limit and below the minimum limit.

156. What does COTS represent?

Commercial Off The Shelf.

157. The purpose of which is to allow specific tests to be carried out on a system or network that resembles as closely as possible the environment where the item under test will be used upon release?

Test Environment

158. What can be thought of as being based on the project plan, but with greater amounts of detail?

Phase Test Plan

159. What is Rapid Application Development?

Rapid Application Development (RAD) is formally a parallel development of functions and subsequent integration. Components/functions are developed in parallel as if they were mini projects, the developments are time-boxed, delivered, and then assembled into a working prototype. This can very quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirements. Rapid change and development of the product are possible using this methodology. However the product specification will need to be developed for the product at some point, and the project will need to be placed under more formal controls before going into production.