

4.1 INTRODUCTION

After analyzing the SUT, this chapter will explain the process of model-based testing and the designation of test calculation process, then the conversion of the model into java class and generate test input and test oracle for salary calculation stub, also the offline test execution tool will be developed to execute all generated test cases to the salary calculation stub and report of pass/fail verdict will be generated.

4.2 MBT PROCESS

— Figure (4.1) below shows the process of model-based testing.

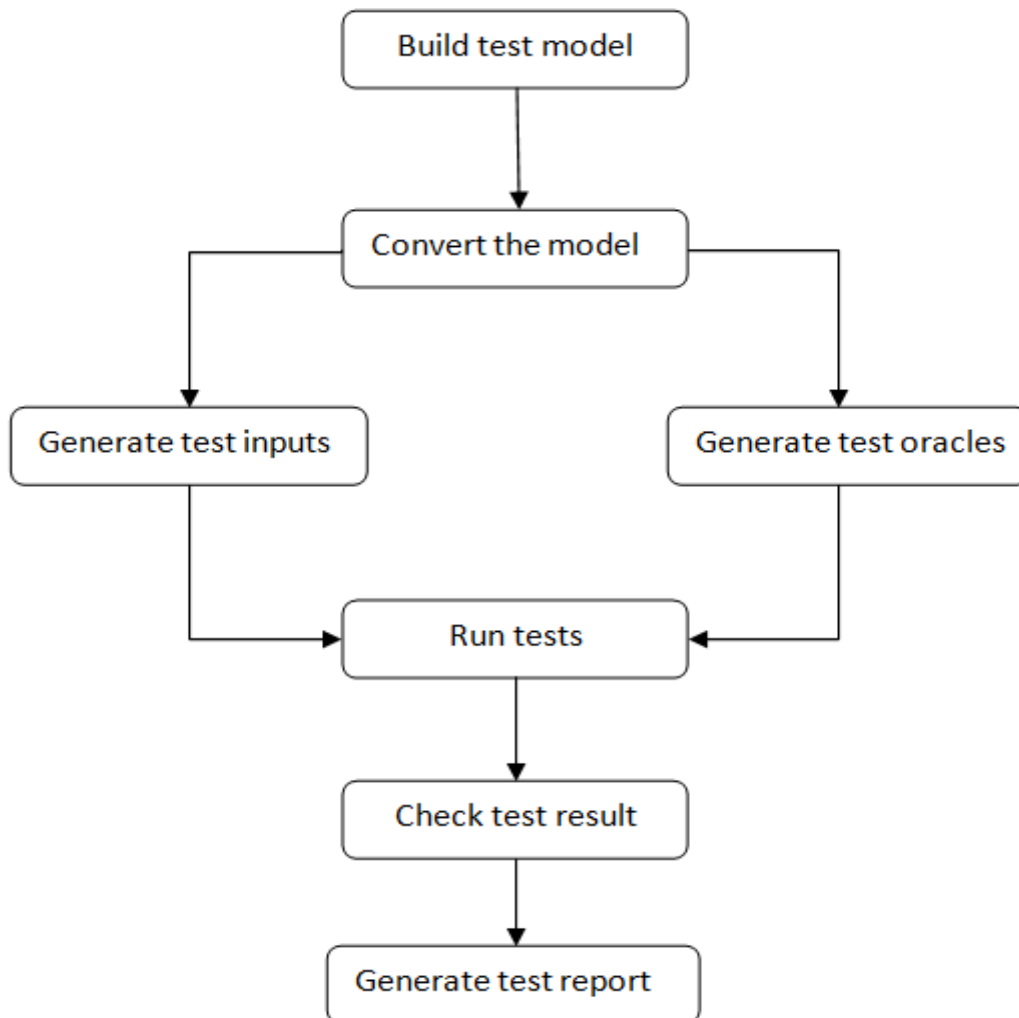


Figure (4.1): model based testing process

4.2.1 OFFLINE TESTING

Model-based testing wherein the complete test suite is generated first, then executed later. It decouples the test generation and test execution environments and tools.

4.2.2 TEST STUB

Stubs are computer programs that act as replacement for a called module and give the same output as the actual product or software. In this research the salary calculation method and the employee registration method will be replaced with stubs that act as the actual calculation method and employee registration method of SUT because of the actual method queries a database to obtain the information of each employee, in this case, the query may be slow and consumes a large number of system resources. This reduces the number of test runs per day. Secondly, tests may need to include values outside those currently in the database. The method (or call) used to perform this is *get_salary()*, For testing purposes, the source code in *get_salary()* and can be replaced with a simple statement that returns a specific value. That is what will be explained in the section of applying of generated test cases to the salary calculation stub which shown below.

4.2.2.1 SALARY CALCULATION STUB

```
public class CalculationStub {  
  
    public static int salaryStub(int main,int premium,int  
deduction){  
        int salary=main+premium-deduction;  
        return salary;  
    }  
}
```

4.2.3 DESIGNING OF FSM MODEL FOR SALARY CALCULATION PROCESS (OFFLINE TESTING)

This section describes the salary calculation model of SUT, which is based on finite state machines (FSMs). Each node of an FSM corresponds to a particular state of the SUT and each arc corresponds to an action of SUT, so to generate test sequences we can just traverse the FSM. From figure (3.17) in

chapter 3 the model will be rearranged with actual instances to design the FSM model. Figure (4.2) below shows the finite state machine of salary calculation process with random test instances for main salary, premiums and deductions values.

The offline testing technique is used when every input has a specific output, which means that there is no more than one output for a specific input.

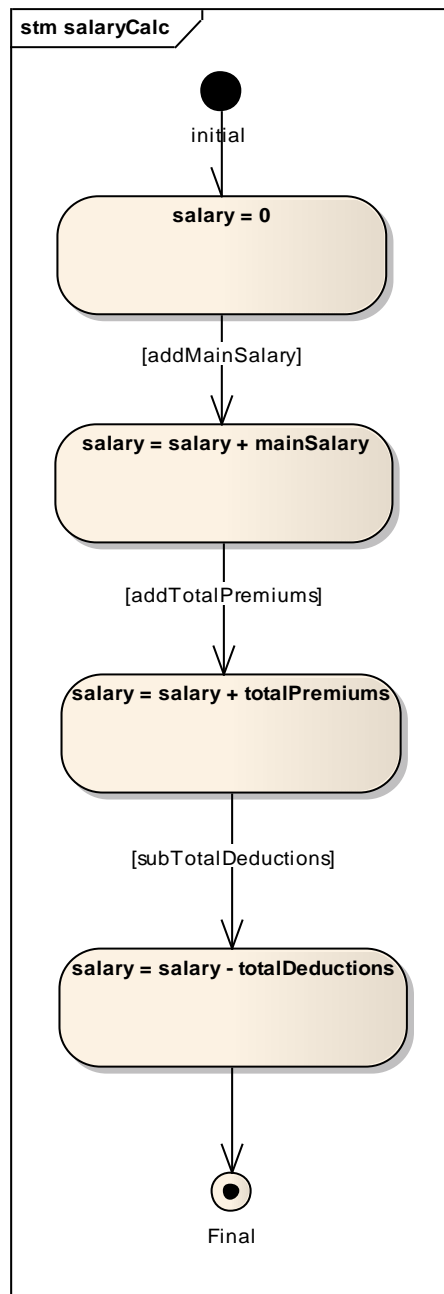


Figure (4.2) FSM

4.2.4 CONVERSION OF MODEL INTO JAVA CODE, GENERATION OF TEST CASES AND DETERMINATION OF ORACLE PROBLEM.

The operation of writing an EFSM in the ModelJUnit style, the model will be converted into an EFSM and then generate a small test suite from the model.

4.2.4.1 CONVERSION OF MODEL TO EFSM IN THE MODELJUNIT STYLE

```
import java.sql.*;
import java.util.Random;
//import junit.framework.Assert;
import nz.ac.waikato.modeljunit.*;
import nz.ac.waikato.modeljunit.coverage.TransitionCoverage;
/** A model of a set with four states : zero salary,add premium
state,add deduction state and add main salary state.
 *
 *zero salary describes the initial state of salary
 *add premium state for adding total premiums to the salary.
 *add deduction state for adding total deductions to the salary.
 *add main salary state for adding the main salary to the salary.
 *after generation of testcases all of them will be inserted in
database table
 *and used to test the salaryStub method(stub).
 */
public class SalaryTestCases implements FsmModel
{
    static int salary;
    static int main;
    static int premiums;
    static int deductions;
    public enum state
    {
        zero,mainSalary,Premiums,Deductions;
    };
    private state current;
    public SalaryTestCases()
    {
        salary=0;
```

```

        current=state.zero;
    }

    public Object getState()
    {
        return salary;
        //return current;
    }

    public void reset(boolean testing)
    {
salary=0;
current=state.zero;
    }

        public boolean addMainSalaryGuard()
        {return current==state.zero;}
    @Action public void addMainSalary()
    {
        Random r=new Random();
        main=r.nextInt(500);
        salary+=main;
        current=state.mainSalary;
    }

        public boolean addPremiumsGuard()
        {return current==state.mainSalary;}
    @Action public void addPremiums()
    {
        Random r1=new Random();
        premiums=r1.nextInt(100);
        salary+=premiums;
        current=state.Premiums;
    }

    public boolean addDeductionsGuard()
    {return current==state.Premiums;}
    @Action public void addDeductions()
    {
        Random r2=new Random();
        deductions=r2.nextInt(50);
        salary-=deductions;
        // System.out.println(main+" "+premiums+"-
"+deductions+"="+salary);
        current=state.Deductions;
        try
        {
            Connection c;
            Statement s;
            //ResultSet r;
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            c=DriverManager.getConnection("jdbc:odbc:Testdb");
            s=c.createStatement();
            String s1="INSERT INTO testcases
VALUES("+main+", "+premiums+", "+deductions+", "+salary+)";
            s.executeUpdate(s1);

```

```

        c.close();
    }catch(Exception e){}
}

/** Check that the SUT is in the expected salary. */
/** An example of generating tests from this model. */
public static void main(String[] args)
{
    Tester tester = new GreedyTester(new SalaryTestCases());
    //tester.buildGraph(); // to get better statistics
    tester.addListener(new VerboseListener());

    // uncomment this line if you want to stop when the first error is
    found.
    // tester.addListener(new StopOnFailureListener());

    tester.addCoverageMetric(new TransitionCoverage());
    tester.generate(30);
    tester.printCoverage();
}
}

```

4.2.4.2 GENERATION OF TEST CASES FROM EFSM

Figure (4.3) below illustrates the generation of test cases process from the model.

```

static int premiums;
static int deductions;
public enum state
{
    zero, mainSalary, Premiums, Deductions;
};
private state current;
public SalaryTestCases()
{

```

```

<terminated> SalaryTestCases [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Apr 29, 2014 3:32:36 AM)
done (0, addMainSalary, 24)
done (24, addPremiums, 93)
done (93, addDeductions, 85)
done Forced reset(true)
done (0, addMainSalary, 184)
done (184, addPremiums, 238)
done (238, addDeductions, 206)
done Forced reset(true)
done (0, addMainSalary, 80)
done (80, addPremiums, 124)

```

generated test cases from the model

Figure (4.3), the process of generation of test cases.

4.2.4.3 THE TEST ORACLE

Test oracle is the instance that decides whether a test case passed or failed. From the previous model, the test cases will be stored into database with its test oracle to decide whether the test cases passed or failed.

4.2.4.3.1 THE TEST ORACLE FOR GENERATED TEST CASES

Table (4.3) shows sample of test cases that generated from the model and its test oracle.

Testcases			
mainSalary	premiums	deductions	Result (test oracle)
107	11	46	<u>372</u>
390	19	27	382
311	75	29	357
131	51	2	180
497	58	9	546
81	36	44	73
252	46	45	253

Table (4.1)

Now each test case will be passed if and only if its output equals its test oracle, otherwise it will be failed. The first test oracle in the table is 72 actually, but it has been modified with invalid test oracle to show fail verdict in the execution later.

4.2.5 APPLYING OF TEST CASES TO SALARY CALCULATION STUB

In this section the test cases that stored in a database will be applied to the salary calculation stub then the results will be shown in a GUI window.

4.2.5.1 OFFLINE TEST CASES EXECUTION TOOL

In this phase, the offline test cases execution tool is developed to execute all test cases stored in database automatically. This tool receives the class name, method name and the name of database table that hold test cases to be executed to the method and then returns a report showing the pass/fail verdict of each test cases.

4.2.5.1.1 OFFLINE TEST CASES EXECUTION TOOL – START WINDOW

Figure (4.4) below shows the start window of the tool.

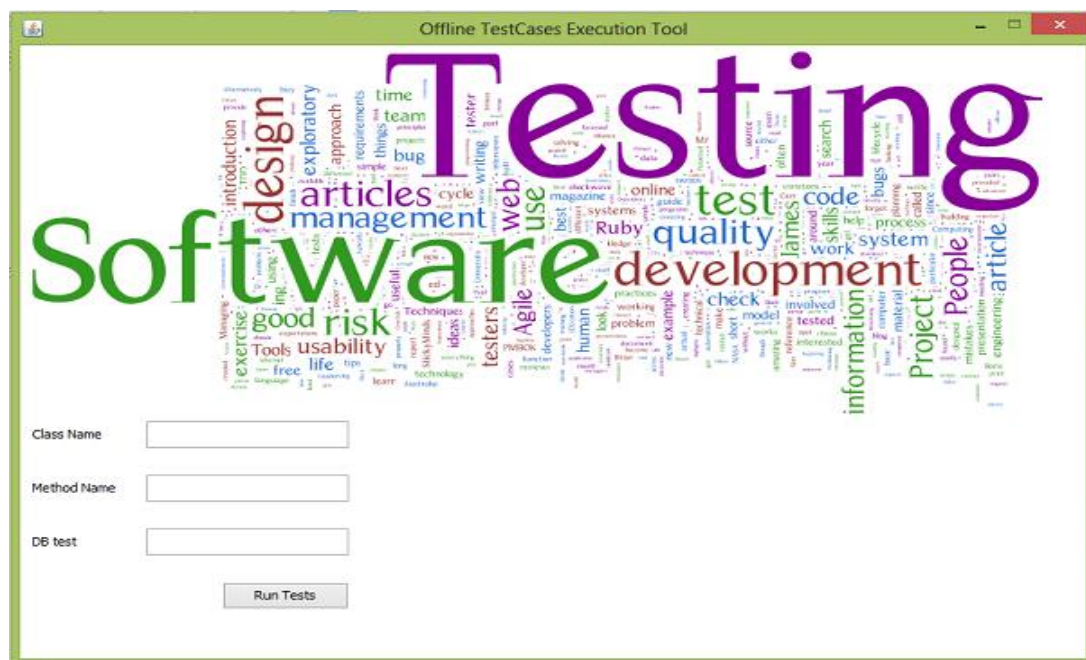
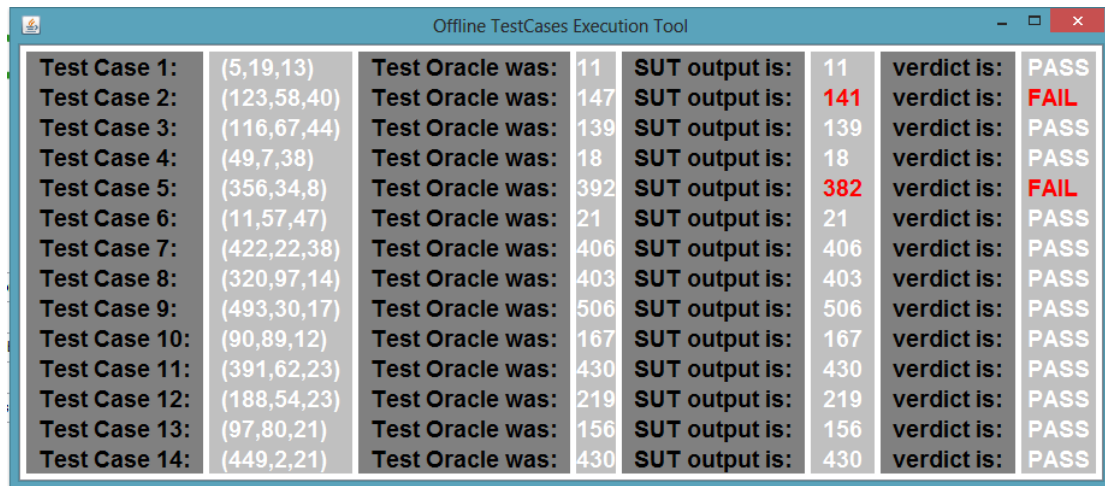


Figure (4.4): start window

4.2.5.1.2 OFFLINE TEST CASES EXECUTION TOOL – REPORT WINDOW

When the “run tests” button is clicked, the offline execution tool will execute all tests on the method specified and generate the execution report with pass/fail verdict as shown below in figure (4.5).



Test Case	Test Oracle	SUT output	Verdict
Test Case 1: (5,19,13)	Test Oracle was: 11	SUT output is: 11	verdict is: PASS
Test Case 2: (123,58,40)	Test Oracle was: 147	SUT output is: 141	verdict is: FAIL
Test Case 3: (116,67,44)	Test Oracle was: 139	SUT output is: 139	verdict is: PASS
Test Case 4: (49,7,38)	Test Oracle was: 18	SUT output is: 18	verdict is: PASS
Test Case 5: (356,34,8)	Test Oracle was: 392	SUT output is: 382	verdict is: FAIL
Test Case 6: (11,57,47)	Test Oracle was: 21	SUT output is: 21	verdict is: PASS
Test Case 7: (422,22,38)	Test Oracle was: 406	SUT output is: 406	verdict is: PASS
Test Case 8: (320,97,14)	Test Oracle was: 403	SUT output is: 403	verdict is: PASS
Test Case 9: (493,30,17)	Test Oracle was: 506	SUT output is: 506	verdict is: PASS
Test Case 10: (90,89,12)	Test Oracle was: 167	SUT output is: 167	verdict is: PASS
Test Case 11: (391,62,23)	Test Oracle was: 430	SUT output is: 430	verdict is: PASS
Test Case 12: (188,54,23)	Test Oracle was: 219	SUT output is: 219	verdict is: PASS
Test Case 13: (97,80,21)	Test Oracle was: 156	SUT output is: 156	verdict is: PASS
Test Case 14: (449,2,21)	Test Oracle was: 430	SUT output is: 430	verdict is: PASS

Figure (4.5): test execution report window

4.3 CONCLUSION

This chapter illustrated the process of model based testing, the design of salary calculation process model, the conversion of the model into java code and then generation of test cases (test inputs and test oracles), also showed the developed offline test execution tool that uses stored test cases to execute it automatically to the method specified, finally, we thank god for help us in proving that the model based testing reduces costs and efforts ,this approach should be the future approach to use in software testing according to its benefits that is proved in this project.