DOI: http://dx.doi.org/10.26483/ijarcs.v8i7.4352

ISSN No. 0976-5697

Volume 8, No. 7, July – August 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

AN EFFICIENT APPROACH OF REGRESSION TESTING USING HIERARCHICAL DECOMPOSITION SLICING

Surbhi Bansal
Research Scholar
HCTM Technical Campus, Kaithal

Dr. Mukesh Kumar Rana
Assistant Professor
HCTM Technical Campus, Kaithal

Abstract: The purpose of regression testing is to ensure that bug fixes and new functionality introduced in a new version of a software do not adversely affect the correct functionality inherited from the previous version. I have proposed a new slicing method to decompose a Java program into affected packages, classes, methods and statements identified with respect to the modification made in the program.Because java program has hierarchical characteristics(Package/Class/Method), this decomposition is based on this characteristic. I also have proposed an intermediate representation for Java programs that shows Package Level, Class Level, and Method Level Dependency. I have named this intermediate representation as Program Dependency Tree(PDT). This PDT is used to identify the program parts that are possibly affected by the change made to the program. With the help of PDT, we can obtain program slices. The packages, classes, methods, and statements thus affected are identified by traversing the Program Dependency Tree. The proposed approach maps the decomposed slice (comprising of the affected program parts) with the coverage information of the existing test suite to select the appropriate test cases for regression testing.

1. REGRESSION TESTING

Regression testing is the process of validating modified software to provide confidence that the changed parts of the software behave as intended and that the unchanged parts of the software have not been adversely affected by the modifications. Modifications in the user requirements and growing expectations of the clients have forced the software to evolve at regular intervals of time. As the complexity of software increases, the cost and effort to maintain such complex software also increase. After making the required changes to the software, regression testing should be carried out in order to assure the validity of the modified part and to ensure that the changes do not affect other parts of the program.
A system is said to regress if 1) a new component is added, or 2) a modification done to the existing component affects other parts of the program. Therefore, it is absolutely necessary to retest not only the changed code but also to retest the possible affected code due to the change. Regression testing is an expensive activity and normally accounts for half of the total cost of software maintenance [1]. It is essential to cut-down the cost of retesting the software by pursuing a selective approach to identify and retest only those parts of the program that are affected by the change. Gupta et al. [2] have identified two important problems in selective regression testing: (1) identifying those existing tests that must be rerun since they may exhibit different behavior in the changed program and (2) identifying those program components that must be retested to satisfy some coverage criterion.

2. PROGRAM SLICING

Program slicing is the computation of the set of programs statements, the program slice that may affect the values at some point of interest, referred to as a slicing criterion. Program slicing can be used in debugging to locate source of errors more easily. Program slicing is a method of separating out the relevant parts of a program with respect to a particular computation. Thus, slice of a program is a set of statements of the program that affects the value of a variable at a particular point of interest. A program slice at a statement s consists of a set of relevant statements of a program those directly or indirectly affects s. Program slicing was originally introduced by Mark Weiser [3] as a method for automatically decomposing programs by analyzing their data flow and control flow dependences starting from a subset of a program's behavior. Finding all statements in a program that directly or indirectly affect the value of a variable occurrence is referred to as Program Slicing.

3. APPLICATIONS OF PROGRAM SLICING

In the following, we briefly discuss some of the applications of program slicing.

- **Testing:** Software maintainers often carry out regression testing. Regression testing essentially implies retesting software after modification [4,5,6]. Even after the smallest change to a piece of code, extensive tests may be necessary which might involve running a large number of test cases to eliminate any unwanted behavior arising due to the change.
- **Debugging:** Programmers mentally slice a code while debugging it. Program slicing is useful for debugging, because it potentially allows one to ignore many statements in the process of localizing the bug.
- **Software Maintenance:** One of the problems in software maintenance is that of the ripple effect, i.e., whether a code change in a program will affect the behavior of other codes of the program. To avoid this problem, it is necessary to know which variables in which statements will be affected by a modified variable, and which variables in which statements will affect a modified variable during software maintenance. The needs can be satisfied by slicing the program being maintained [7].

- **Change Impact Analysis:** In regression testing only those parts are tested that are affected by the changes made to the program. Software change impact analysis is the mechanism of finding out the unpredicted and potential effect of the changes and the propagation of the impact to other parts of the program.
- **Software Quality Assurance:** Software quality assurance auditors have to locate safety critical code and to ascertain its effect throughout the system. Program slicing can be used to locate all code that influences the values of variables that might be part of a safety critical component. But beforehand these critical components have to be determined by domain experts.
- **Other Applications of Program Slicing:** Program slicing methods have been used in several other applications such as compiler optimization, detecting dead code, software portability analysis, program understanding, program verification, measuring class cohesion, etc.

4. PROPOSED WORK

The change in the user requirements and growing expectations of the customers has forced the software to evolve at regular intervals of time. As the complexity of the software increases, the cost and effort to maintain such complex software also increases. Therefore, regression testing has become an integral part of the software maintenance process. It is indispensable to make changes and modifications to an already tested program.

Program slicing is an effective and efficient technique to debug, test, analyze, understand and maintain software. However, while applying the same techniques to OO programs, we fail because of the presence of many other dependences originating from the OO features. Although OO features have improved program understandability and readability, but have complicated the maintenance activities. The dependences that arise due to the class and object concepts are inheritance dependence, message dependence, data dependence, type dependence, reference dependence, concurrency dependence, etc. The presence of the features like *packages*, *super*, *dynamic method dispatch*, *interface*, *exception handling*, *multi-threading*, etc, in Java add to the list of dependences and thus make the maintenance even more difficult. Their effects on the maintenance of the programs need to be considered separately.

Keeping in view the above motivations and to overcome the challenges, we fix our goal on following points.

1. To construct a Proposed Dependency Tree for representing different dependences in Java programs arising due to the different object-oriented features.
2. To develop and implement a slicing algorithm that identifies different program parts affected by the changes made to the program.

3. To select the test cases that are relevant for regression testing of the program under consideration.

5. PROGRAM DEPENDENCY TREE AND HIERARCHICAL SLICING

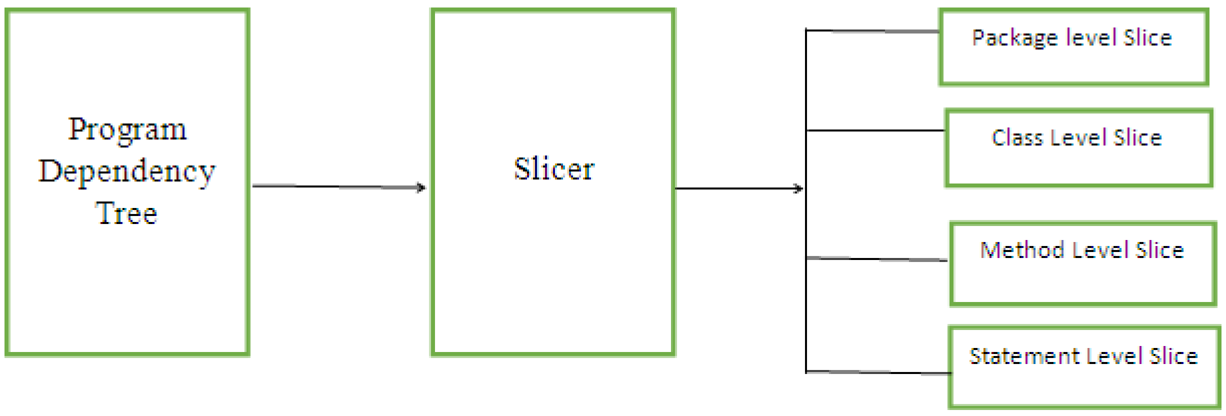


Figure1: Model for Dependency Tree &Hierarchical Slicing.

Instead of analyzing the data flow and control flow for an OO program as a whole, it is useful to employ the hierarchical structure of the OO programs (e.g. Java programs), to detect the impact of the changes made to the program. A Java program P, is composed of a set of packages, classes, methods and statements, organized in a hierarchical manner. Therefore, in *hierarchical slicing*, we first try to slice out the packages that might have been affected by the change. From the set of affected packages, we then slice out the affected classes. Then the affected methods and the statements inside those methods are sliced out for retesting. The above concept of *hierarchical slicing* can be explained by considering a slicing criterion (i.e. the point of modification) $< s, v >$, where s is the statement containing variable v. Let $S(P)$ be the set of packages, classes, methods and statements of a program P. The steps of hierarchical slicing are as follows:

Step-1 First, we detect the package p containing s and v and all other packages, based on their direct or indirect dependences on p caused due to import statements. All those packages which are not related to the package p are removed. By following this process, the package level slice obtained is marked as $S1(P)$.

Step-2 Then, we analyze $S(P)$, to find out all those classes that are related to the class containing s and v. All other irrelevant classes are removed to get the class level slice. The class level slice is marked as $S2(P)$.

Step-3 Next, we analyze $S(P)$ and delete all the member methods and variables that are not related to the method containing s and v. This results in the method level slice, which is marked as $S3(P)$.

Step-4 Finally, to find out the statement level slice, we analyze $S(P)$ and delete all the statements and predicates that are not related to statement *s* containing variable *v*. The slice thus obtained is marked as $S4(P)$.

This step wise extraction of the slices is known as *hierarchical slicing*. We use this concept of hierarchical slicing for selecting our regression test cases.

Java being the most popular OO language, we are encouraged to consider Java programs. We propose a dependency tree of java program suitable for slicing and test case generations.

6. PROPOSED PROGRAM DEPENDENCY TREE OF JAVA PROGRAM

The proposed Program Dependency Tree is a hierarchical structure which displays dependency information. We classify these dependences based on their role in representing some Object-Oriented features at different hierarchical levels.

- **Package level dependences:** Package level dependence specifies the dependence of a package on another package.
- **Class level dependences:** When a class uses the features of another class, this kind of dependency is shown under this.
- **Method level dependences:** Message passing is an important feature in object-oriented programs realized through method invocation by the objects. When one method invokes another method, it passes messages in the form of parameters. *In method level dependency, we can see all methods and their dependent methods.*

7. PROPOSED HIERARCHICAL DECOMPOSITION (HD) SLICING ALGORITHM

Following is the proposed algorithm named *Hierarchical Decomposition (HD) Slicing*, for finding those program parts that are affected by the change. The node that corresponds to the statement of modification is taken as the slicing criterion to compute the slices.

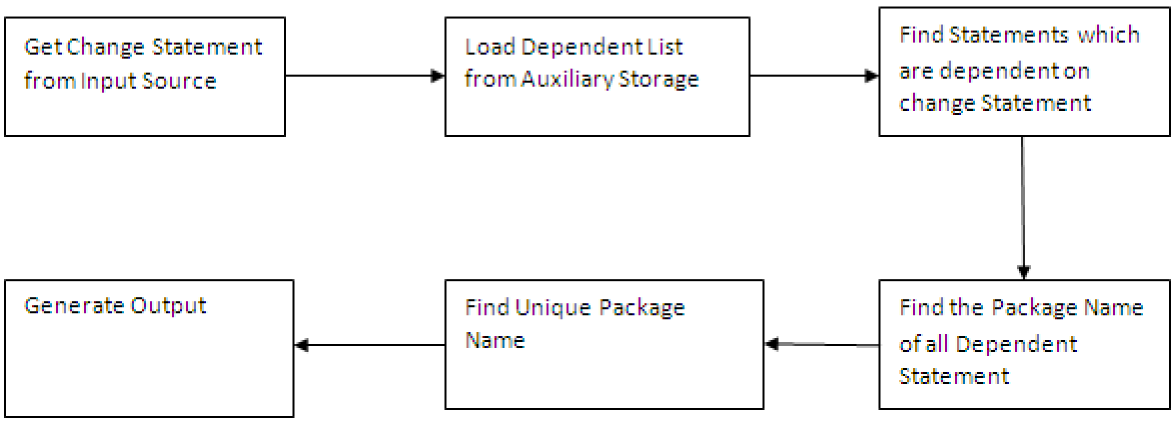

Get Change Statement from Input Source
Load Dependent List from Auxiliary Storage
Find Statements which are dependent on change Statement
Find the Package Name of all Dependent Statement
Find Unique Package Name
Generate Output

Figure2: Package Level Slice

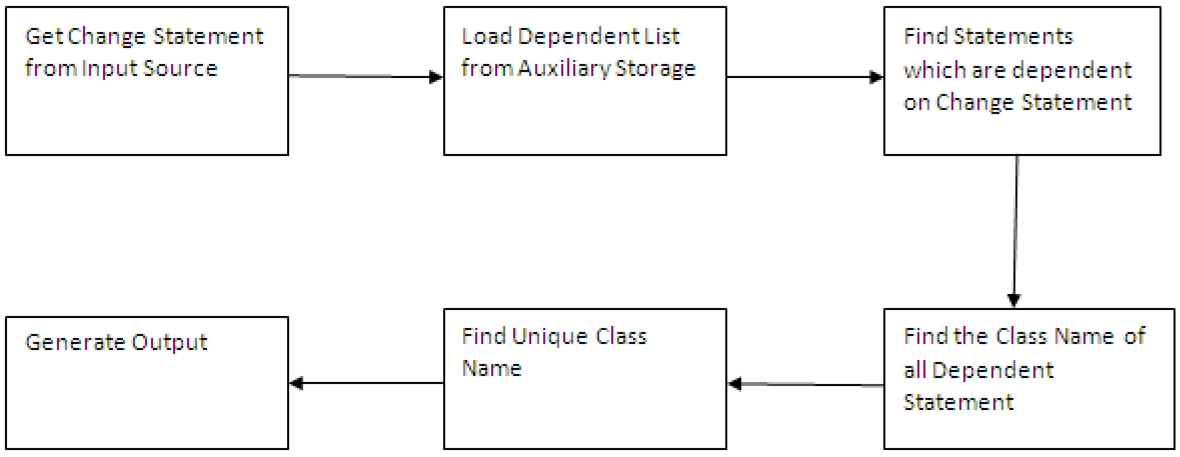

Get Change Statement from Input Source
Load Dependent List from Auxiliary Storage
Find Statements which are dependent on Change Statement
Find the Class Name of all Dependent Statement
Find Unique Class Name
Generate Output

Figure3: Class Level Slice

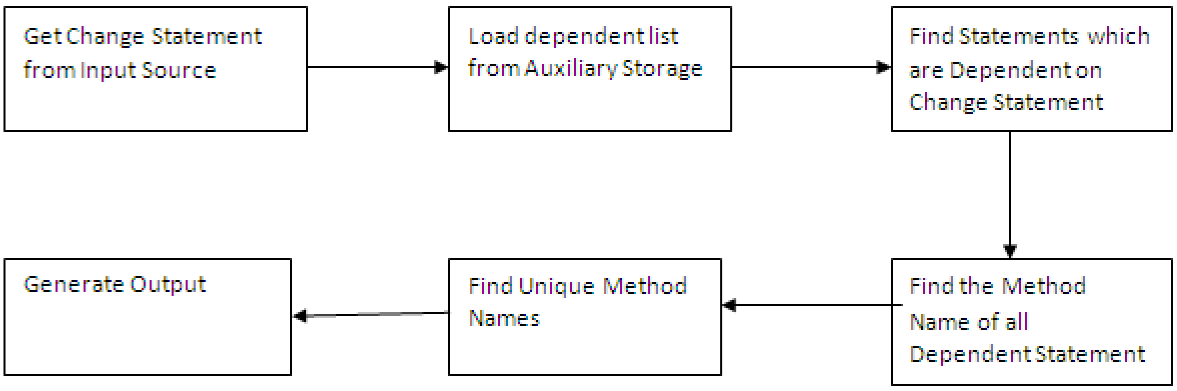

Get Change Statement from Input Source
Load dependent list from Auxiliary Storage
Find Statements which are Dependent on Change Statement
Find the Method Name of all Dependent Statement
Find Unique Method Names
Generate Output

Figure4: Method level Slice

Algorithm: Hierarchical Decomposition Slicing Algorithm
Input: PROGINFO, Reference of statements to be changed(STM), Program Dependency Tree (PDTROOT)
Output: Package Level Slice, Class Level Slice, Method Level Slice

```
1. Set DepNodeList:=[];
2. Repeat and traverse each node N of PDTROOT using Preorder method
     a. If there is parent-child relation between STM and N [STM is parent, N is child or STM is child, N is parent]
          i. Add Node N to DepNodeList
        [End of If]
        [End of Repeat Traverse Loop]
3. Set PackSlice=[]
4. Repeat for each Statement S inDepNodeList
     a. Get PackageName PN of statement S from PROGINFO
     b. Add PN to PackSlice
   [End of Repeat Loop]
5. Set ClassSlice=[]
6. Repeat for each Statement S inDepNodeList
     a. Get ClassName CN of statement S from PROGINFO
     b. Add CN to ClassSlice
   [End of Repeat Loop]
7. Set MethodSlice=[]
8. Repeat for each Statement S inDepNodeList
     a. Get MethodName MN of statement S from PROGINFO
     b. Add MN to MethodSlice
   [End of Repeat Loop]
9. End
```

8. PROPOSED REGRESSION TEST CASE SELECTION ALGORITHM

Following is the proposed algorithm named *Regression Test Selection*, for generating selective regression test cases. Our proposed algorithm takes the validation rules defined by the client and after analyzing the slices, it generates all test cases. The outcome of the algorithm is a set of change-based selected test cases suitable for regression testing. Algorithm gives the steps of our proposed regression test case selection approach in pseudo code form.

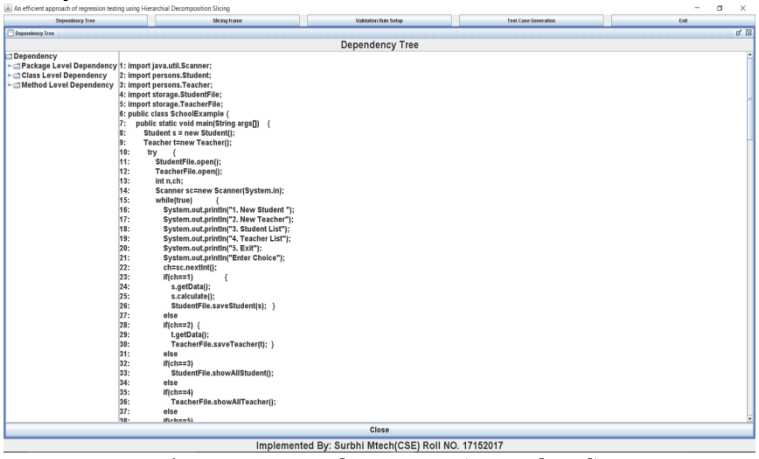
Algorithm: Hierarchical Regression Test Case Selection Algorithm
Input: PackageName, ClassName, Decomposed Slices, Validation Rules
Output: A set of selected change based test cases

```
1. Set Testset=[]
2. Repeat and Apply Process for each rule in RuleSet
     a. Get Rule from RuleSet
     b. Analyse the rule, generate positive testcase and insert testcase into Testset
     c. Analyse the rule, generate negative testcase and insert into Testset.
   [ End of Repeat Loop Step 3]
3. View Testset
4. End
```

9. IMPLEMENTATION

To implement the proposed methods of Program Slicing, Program Dependency Tree and Generating Regression Test cases, I have used Java Language with Net Beans IDE.

Dependency Tree (Unexplored): In this tree, there are two regions. First region is the Tree on left side, in which 3 types of dependencies are shown. Second region is the programs of all dependency which is placed in the center. Dependencies shown in the left side tells that which packages are dependent on any other package, which classes are dependent on any of the class & which methods are dependent on any methods.

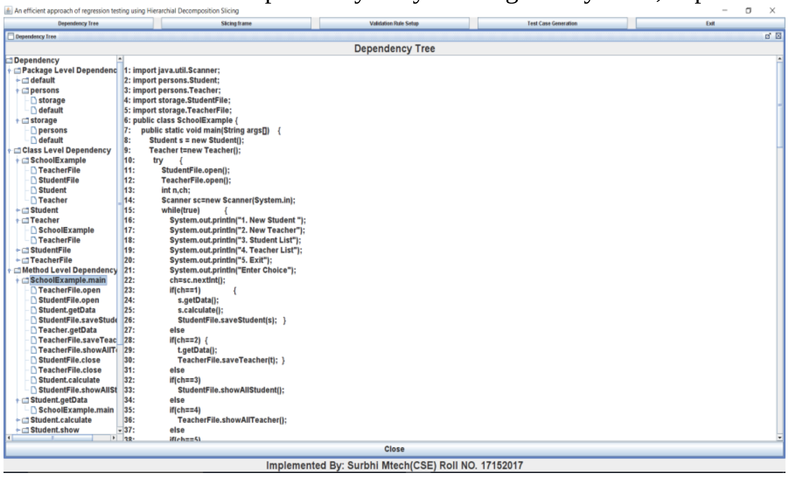


```
import java.util.Scanner;
import persons.Student;
import persons.Teacher;
import storage.StudentFile;
import storage.TeacherFile;
public class SchoolExample {
    public static void main(String args[])    {
        Student s = new Student();
        Teacher t=new Teacher();
          try       {
             StudentFile.open();
             TeacherFile.open();
             int n,ch;
             Scanner sc=new Scanner(System.in);
             while(true)         {
                System.out.println("1. New Student ");
                System.out.println("2. New Teacher");
                System.out.println("3. Student List");
                System.out.println("4. Teacher List");
                System.out.println("5. Exit");
                System.out.println("Enter Choice");
                ch=sc.nextInt();
                if(ch==1)             {
                   s.getData();
                   s.calculate();
                   StudentFile.saveStudent(s);   }
                else
                if(ch==2)  {
                   t.getData();
                   TeacherFile.saveTeacher(t);  }
                else
                if(ch==3)
                   StudentFile.showAllStudent();
                else
                if(ch==4)
                   TeacherFile.showAllTeacher();
                else
```

Implemented By: Surbhi Mtech(CSE) Roll NO. 17152017

Figure 5: Dependency Tree (Unexplored)

Dependency Tree(Explored): Three types of dependency are shown in the Dependency Tree. When we click on Package Level Dependency all the packages appears & as we click on any package, all the package dependent on that package also appears. Classes get appears when we click on Class Level Dependency &By clicking on any class, dependent classes get appears.



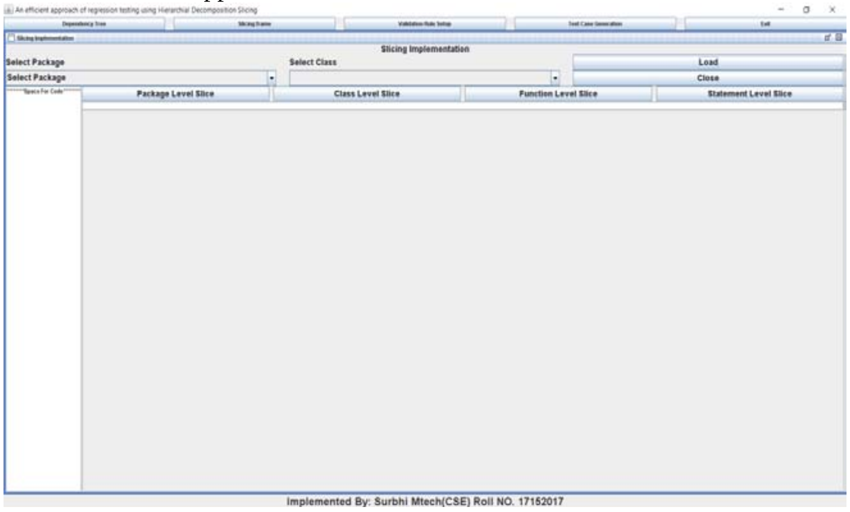
```
Dependency
  Package Level Dependenc
    default
    persons
      storage
      default
    storage
      persons
      default
  Class Level Dependency
    SchoolExample
      TeacherFile
      StudentFile
      Student
      Teacher
    Student
    Teacher
      SchoolExample
      TeacherFile
    StudentFile
    TeacherFile
  Method Level Dependency
    SchoolExample.main
      TeacherFile.open
      StudentFile.open
      Student.getData
      StudentFile.saveStud
      Teacher.getData
      TeacherFile.saveTeac
      TeacherFile.showAllT
      StudentFile.close
      TeacherFile.close
      Student.calculate
      StudentFile.showAllSt
    Student.getData
      SchoolExample.main
    Student.calculate
    Student.show
```

```
import java.util.Scanner;
import persons.Student;
import persons.Teacher;
import storage.StudentFile;
import storage.TeacherFile;
public class SchoolExample {
    public static void main(String args[])    {
        Student s = new Student();
        Teacher t=new Teacher();
          try       {
             StudentFile.open();
             TeacherFile.open();
             int n,ch;
             Scanner sc=new Scanner(System.in);
             while(true)         {
                System.out.println("1. New Student ");
                System.out.println("2. New Teacher");
                System.out.println("3. Student List");
                System.out.println("4. Teacher List");
                System.out.println("5. Exit");
                System.out.println("Enter Choice");
                ch=sc.nextInt();
                if(ch==1)             {
                   s.getData();
                   s.calculate();
                   StudentFile.saveStudent(s);   }
                else
                if(ch==2)  {
                   t.getData();
                   TeacherFile.saveTeacher(t);  }
                else
                if(ch==3)
                   StudentFile.showAllStudent();
                else
                if(ch==4)
                   TeacherFile.showAllTeacher();
                else
```

Implemented By: Surbhi Mtech(CSE) Roll NO. 17152017

Figure 6: Dependency Tree(Explored)

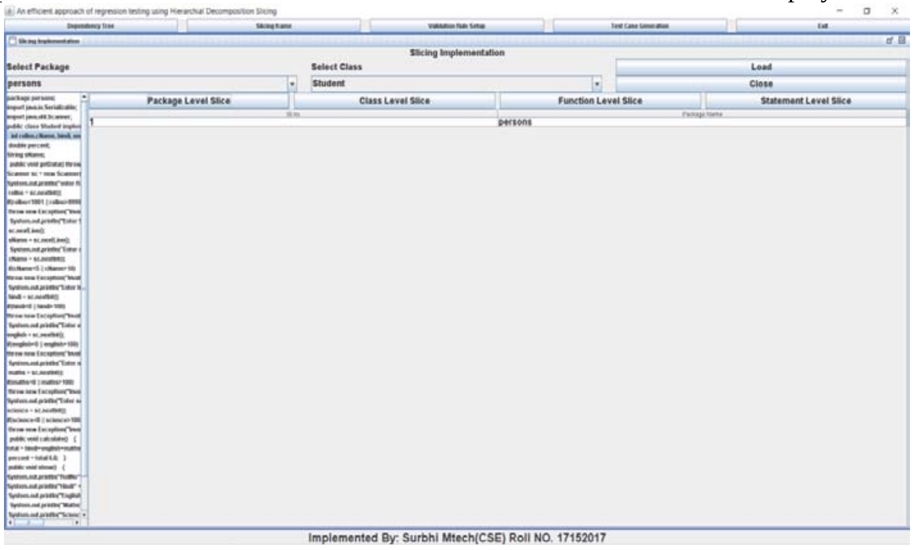
Slicing Implementation: Figure7 represents the implementation work of Program Slicing. In this frame, first we have to select any package in the first combo box, and then all the classes of selected package will appear in second combo box. Press load button & all the statements of selected class will appear in left corner of the frame. Then select any statement& it gives you 4 options to select any of four button named Package Level slice, Class level slice, Function level slice, Statement level slice. So the slices corresponding to button selected will appear in the frame.



Implemented By: Surbhi Mtech(CSE) Roll NO. 17152017

Figure 7: Slicing Implementation

Slicing Implementation (Package level Slice): This frame displays package level slice of selected statement. It means all those packages which are dependent on the statement which we have selected from the left side are displayed.



An efficient approach of regression testing using Hierarchial Decomposition Slicing

Slicing Implementation

Select Package: persons

Select Class: Student

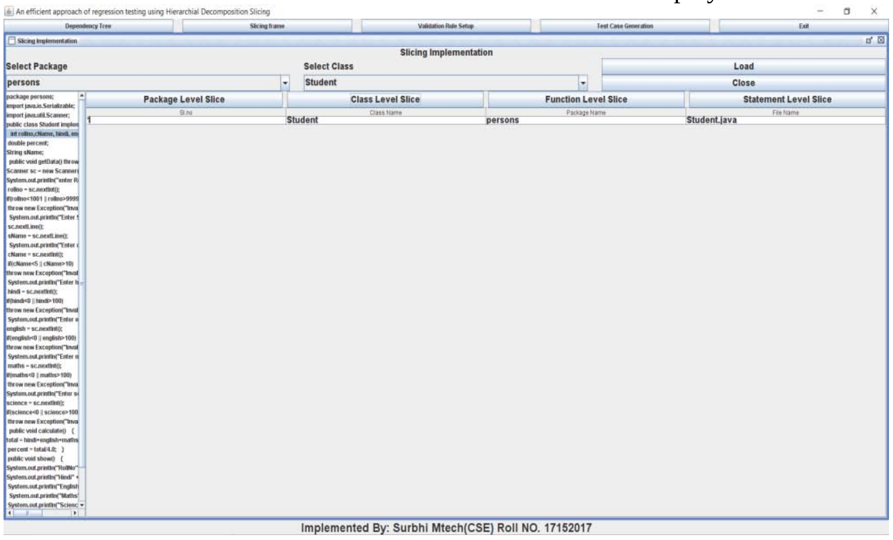
```
package persons;
import java.io.Serializable;
import java.util.Scanner;
public class Student implem
  int rollno,cName, hindi, en
 double percent;
String sName;
  public void getData() throw
Scanner sc = new Scanner(
System.out.println("enter R
 rollno = sc.nextInt();
if(rollno<1001 || rollno>9999
 throw new Exception("Inva
  System.out.println("Enter S
 sc.nextLine();
 sName = sc.nextLine();
  System.out.println("Enter c
 cName = sc.nextInt();
 if(cName<5 || cName>10)
throw new Exception("Inval
 System.out.println("Enter h
 hindi = sc.nextInt();
if(hindi<0 || hindi>100)
throw new Exception("Inval
 System.out.println("Enter e
english = sc.nextInt();
if(english<0 || english>100)
throw new Exception("Inval
 System.out.println("Enter m
 maths = sc.nextInt();
if(maths<0 || maths>100)
 throw new Exception("Inva
System.out.println("Enter sc
science = sc.nextInt();
if(science<0 || science>100
 throw new Exception("Inva
  public void calculate()   {
total = hindi+english+maths
 percent = total/4.0;   }
 public void show()   {
System.out.println("RollNo"
System.out.println("Hindi" +
 System.out.println("English
  System.out.println("Maths"
 System.out.println("Scienc
```

Sl.no	Package Name
1	persons

Implemented By: Surbhi Mtech(CSE) Roll NO. 17152017

Figure 8: Slicing Implementation (Package level Slice)

Slicing Implementation (Class level Slice): This frame displays class level slice of selected statement. It means all those classes which are dependent on the statement which we have selected from the left side are displayed.



An efficient approach of regression testing using Hierarchial Decomposition Slicing

Slicing Implementation

Select Package: persons

Select Class: Student

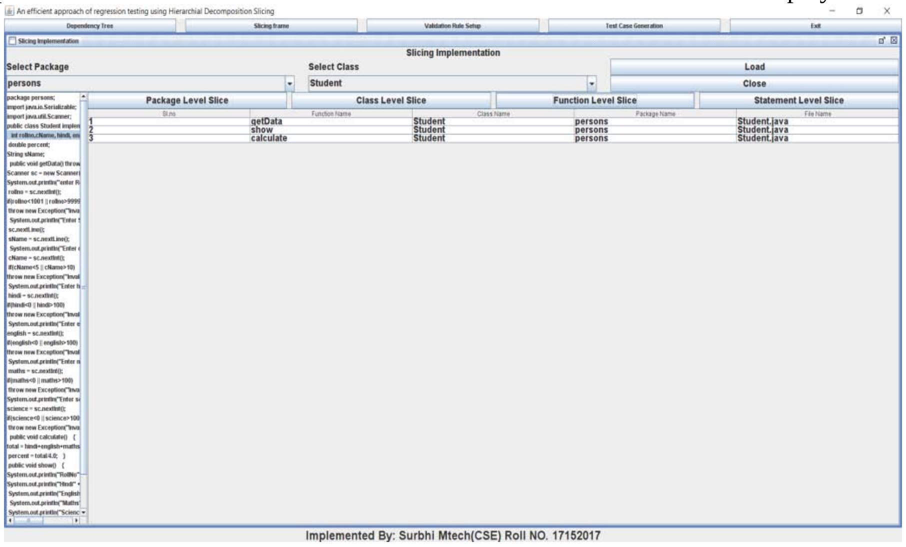
```
package persons;
import java.io.Serializable;
import java.util.Scanner;
public class Student implem
  int rollno,cName, hindi, en
 double percent;
String sName;
  public void getData() throw
Scanner sc = new Scanner(
System.out.println("enter R
 rollno = sc.nextInt();
if(rollno<1001 || rollno>9999
 throw new Exception("Inva
  System.out.println("Enter S
 sc.nextLine();
 sName = sc.nextLine();
  System.out.println("Enter c
 cName = sc.nextInt();
 if(cName<5 || cName>10)
throw new Exception("Inval
 System.out.println("Enter h
 hindi = sc.nextInt();
if(hindi<0 || hindi>100)
throw new Exception("Inval
 System.out.println("Enter e
english = sc.nextInt();
if(english<0 || english>100)
throw new Exception("Inval
 System.out.println("Enter m
 maths = sc.nextInt();
if(maths<0 || maths>100)
 throw new Exception("Inva
System.out.println("Enter sc
science = sc.nextInt();
if(science<0 || science>100
 throw new Exception("Inva
  public void calculate()   {
total = hindi+english+maths
 percent = total/4.0;   }
 public void show()   {
System.out.println("RollNo"
System.out.println("Hindi" +
 System.out.println("English
  System.out.println("Maths"
 System.out.println("Scienc
```

Sl.no	Class Name	Package Name	File Name
1	Student	persons	Student.java

Implemented By: Surbhi Mtech(CSE) Roll NO. 17152017

Figure 9: Slicing Implementation (Class level Slice)

Slicing Implementation (Function level Slice): This frame displays function level slice of selected statement. It means all those functions which are dependent on the statement which we have selected from the left side are displayed.



An efficient approach of regression testing using Hierarchial Decomposition Slicing

Slicing Implementation

Select Package: persons

Select Class: Student

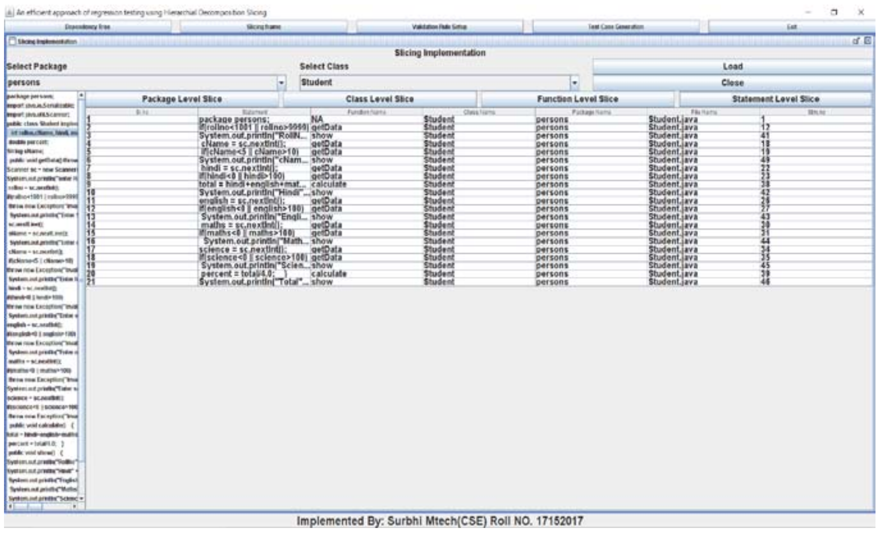
```
package persons;
import java.io.Serializable;
import java.util.Scanner;
public class Student implem
  int rollno,cName, hindi, en
 double percent;
String sName;
  public void getData() throw
Scanner sc = new Scanner(
System.out.println("enter R
 rollno = sc.nextInt();
if(rollno<1001 || rollno>9999
 throw new Exception("Inva
  System.out.println("Enter S
 sc.nextLine();
 sName = sc.nextLine();
  System.out.println("Enter c
 cName = sc.nextInt();
 if(cName<5 || cName>10)
throw new Exception("Inval
 System.out.println("Enter h
 hindi = sc.nextInt();
if(hindi<0 || hindi>100)
throw new Exception("Inval
 System.out.println("Enter e
english = sc.nextInt();
if(english<0 || english>100)
throw new Exception("Inval
 System.out.println("Enter m
 maths = sc.nextInt();
if(maths<0 || maths>100)
 throw new Exception("Inva
System.out.println("Enter sc
science = sc.nextInt();
if(science<0 || science>100
 throw new Exception("Inva
  public void calculate()   {
total = hindi+english+maths
 percent = total/4.0;   }
 public void show()   {
System.out.println("RollNo"
System.out.println("Hindi" +
 System.out.println("English
  System.out.println("Maths"
 System.out.println("Scienc
```

Sl.no	Function Name	Class Name	Package Name	File Name
1	getData	Student	persons	Student.java
2	show	Student	persons	Student.java
3	calculate	Student	persons	Student.java

Implemented By: Surbhi Mtech(CSE) Roll NO. 17152017

Figure 10: Slicing Implementation (Function level Slice)

Slicing Implementation (Statement level Slice): This frame displays statement level slice of selected statement. It means all those statements which are dependent on the statement which we have selected from the left side are displayed.

g 

Slicing Implementation

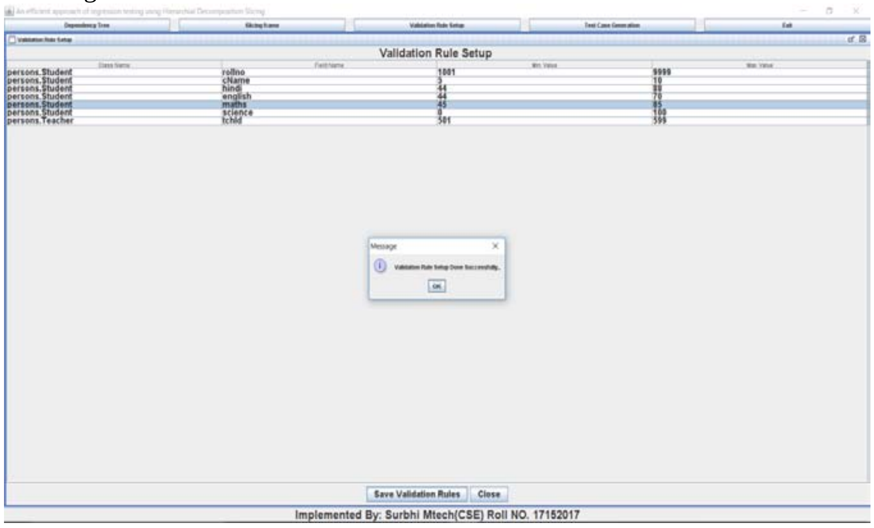
Select Package: persons
Select Class: Student

Sl.No	Statement	Function Name	Class Name	Package Name	File Name	Stmt.No
1	package persons;	NA	Student	persons	Student.java	1
2	if(rollno<1001 || rollno>9999)	getData	Student	persons	Student.java	17
3	System.out.println("RollN...	show	Student	persons	Student.java	41
4	cName = sc.nextInt();	getData	Student	persons	Student.java	18
5	if(cName<5 || cName>10)	getData	Student	persons	Student.java	19
6	System.out.println("cNam...	show	Student	persons	Student.java	49
7	hindi = sc.nextInt();	getData	Student	persons	Student.java	22
8	if(hindi<0 || hindi>100)	getData	Student	persons	Student.java	23
9	total = hindi+english+mat...	calculate	Student	persons	Student.java	38
10	System.out.println("Hindi"...	show	Student	persons	Student.java	42
11	english = sc.nextInt();	getData	Student	persons	Student.java	26
12	if(english<0 || english>100)	getData	Student	persons	Student.java	27
13	System.out.println("Engli...	show	Student	persons	Student.java	43
14	maths = sc.nextInt();	getData	Student	persons	Student.java	30
15	if(maths<0 || maths>100)	getData	Student	persons	Student.java	31
16	System.out.println("Math...	show	Student	persons	Student.java	44
17	science = sc.nextInt();	getData	Student	persons	Student.java	34
18	if(science<0 || science>100)	getData	Student	persons	Student.java	35
19	System.out.println("Scien...	show	Student	persons	Student.java	45
20	percent = total/4.0; }	calculate	Student	persons	Student.java	39
21	System.out.println("Total"...	show	Student	persons	Student.java	46

Implemented By: Surbhi Mtech(CSE) Roll NO. 17152017

Figure11: Slicing Implementation (Statement level Slice)

Validation Rule Setup: This form displays validation rules of the program. Validation rules are stored in the database. Based on the validation rules test case are generated.



Validation Rule Setup

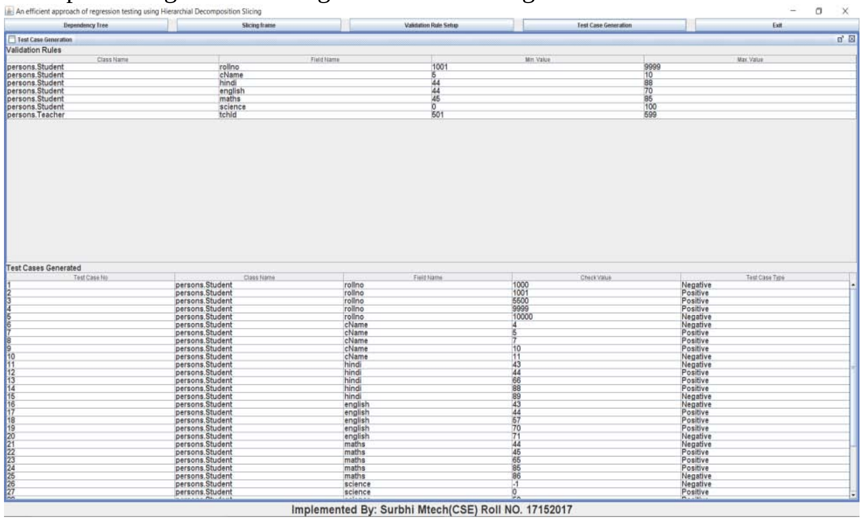
Class Name	Field Name	Min. Value	Max. Value
persons.Student	rollno	1001	9999
persons.Student	cName	5	10
persons.Student	hindi	44	88
persons.Student	english	44	70
persons.Student	maths	45	85
persons.Student	science	0	100
persons.Teacher	tchld	501	599

Message
Validation Rule Setup Done Successfully..

Implemented By: Surbhi Mtech(CSE) Roll NO. 17152017

Figure12: Validation Rule Setup

Test Case Generation: This form displays validation rules and list of test cases. Test case type is given which is either positive or negative. This is very much helpful in regression testing due to automatic generation of test cases.



Validation Rules

Class Name	Field Name	Min. Value	Max. Value
persons.Student	rollno	1001	9999
persons.Student	cName	5	10
persons.Student	hindi	44	88
persons.Student	english	44	70
persons.Student	maths	45	85
persons.Student	science	0	100
persons.Teacher	tchld	501	599

Test Cases Generated

Test Case No	Class Name	Field Name	Check Value	Test Case Type
1	persons.Student	rollno	1000	Negative
2	persons.Student	rollno	1001	Positive
3	persons.Student	rollno	5500	Positive
4	persons.Student	rollno	9999	Positive
5	persons.Student	rollno	10000	Negative
6	persons.Student	cName	4	Negative
7	persons.Student	cName	5	Positive
8	persons.Student	cName	7	Positive
9	persons.Student	cName	10	Positive
10	persons.Student	cName	11	Negative
11	persons.Student	hindi	43	Negative
12	persons.Student	hindi	44	Positive
13	persons.Student	hindi	66	Positive
14	persons.Student	hindi	88	Positive
15	persons.Student	hindi	89	Negative
16	persons.Student	english	43	Negative
17	persons.Student	english	44	Positive
18	persons.Student	english	57	Positive
19	persons.Student	english	70	Positive
20	persons.Student	english	71	Negative
21	persons.Student	maths	44	Negative
22	persons.Student	maths	45	Positive
23	persons.Student	maths	65	Positive
24	persons.Student	maths	85	Positive
25	persons.Student	maths	86	Negative
26	persons.Student	science	-1	Negative
27	persons.Student	science	0	Positive

Implemented By: Surbhi Mtech(CSE) Roll NO. 17152017

Figure13: Test Case Generation

10. CONCLUSION

We proposed an algorithm of java program slicing which is very useful in regression testing. This algorithm takes Program Dependency Tree as input and generates Package level slice, class level slice, method level slice and statement level slices. Using this slicing approach, we can quickly identify the affected program parts with respect to some modifications made to the program. The selected test cases are also found to be very efficient in detecting the regression errors.

The HD (Hierarchical Decomposition) Slicing algorithm is useful for software engineering applications that require computing slices at different program points. Because of its efficiency, very large systems can now be sliced in a very short time, opening new avenues for research. Applications for testing may also benefit from the fast computation time.

11.FUTURE SCOPE

The slicers can be used to develop efficient debuggers and test drivers for large scale object-oriented programs. We can plan to explore this possibility.An investigation into the suitability of the proposed Program Dependency Tree to represent dependences in other object-oriented programs (such as C#) will be studied in future. We also aim to calculate efficiency achieved in regression testing. Meaning, we must know that how much time our slicer saves in regression testing.

12.REFERENCES:

[1] Leung, H., and White, L. Insights into Regression Testing Selection. In Proceedings of the Conference on Software Maintenance (1989), pp. 60–69.

[2] Gupta, R., Harrold, M. J., and Soffa, M. L. Program Slicing-Based Regression Testing Techniques. Software Testability, Verifiability and Reliability 6, 2 (1996), 83– 111.

[3] Weiser, M. Program Slicing. In Proceedings of the 5th International Conference on Software (1981), San Diego, California, USA, pp. 439–449.

[4] Binkley, D. The Application of Program Sli9cing to Regression Testing. Information and Software Technology 40, 11 (1998), 583–594.

[5] Mall, R. Fundamentals of Software Engineering, 3rd ed. PHI Learning Pvt. Ltd., 2010, pp. 159–160.

[6] Tao, C., Li, B., Sun, X., and Zhang, C. An Approach to Regression Test Selection Based on Hierarchical Slicing Technique. In 34th Annual IEEE Computer Software and Applications Conference Workshops (2010), pp. 347–352.

[7] Gallagher, K. B., and Lyle, J. R. Using Program Slicing in Software Maintenance. IEEE Transactions on Software Engineering 17, 8 (1991), 751–761.