Journal of Theoretical and Applied Information Technology
10th March 2015. Vol.73 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

FRAMEWORK FOR SELECTION OF TEST CASES USING AN ETCS APPROACH

Dr. A. PRAVIN
Department of Computer Science and Engineering
Sathyabama University
Chennai
E-mail: pravin_ane@rediffmail.com

ABSTRACT

Inorder to authorize the application after reconstruction Regression Testing is conducted. A horde of analysts worked on this activity to upgrade this process. The regression test is time consuming and the cost involved is high. Our goal is to upgrade the value of fault detection and to diminish the time consumption. This can be done by selecting few test cases and are processed accordingly. The selection process automatically detects the fault during the earlier stage itself. In this paper a method based on the analysis that is carried out on the execution of each application .The analysis is done to find out how far the methods communicate with other and the result is captured. Based on the result obtained the test cases are selected from the set. The proposed Effective Test Case Selection method (ETCS) selects the test case once the application has been modified. The Effective Test Case Selection method will in turn increases the fault detection ratio by detecting faults earlier in the testing process.

Keywords: *ETCS, Regression Testing, Method analysis, Test Case, Test Behavior Extraction.*

1. INTRODUCTION

Regression test analyses the entire part where the change is performed. It will confirm whether that group of modification yields to any new trouble in the working of the given application. It's not possible for us to execute the entire Test Case during the process because it will consume time and the amount spent will be more. A.Pravin et al proposed that new set by which it can be processed in a minimum amount of duration and which covers maximum fault[1]. Many methods are used to improve the selection process. A.Pravin et al devised reduced selection method which improves the test case that is extracted from the old suite[2]. Whenever we are going to do the Ranking all these details are taken for further use. The set of input data is given to the application during the testing process and outcome is evaluated based on the user expectation. We need to design and group the data called as test cases. A.Pravin et al claimed that it is possible for us to trace and extract errors in the part of software using neural network concept. There is no need of having knowledge in detail about the code or the weight allocation[3]. We can say that the process is going in a correct path, if it can be able to detect faults at the starting point. The next process is to deliver the detail to an expert to repeat the evaluation process in a manner which is evaluated differently from the person who had tested already. A. Pravin et al described that by using the genetic approach we can be able to select a set of quality test cases when compared to the random approach[4]. The test case design is to be considered as an important process. Production of effective sets will improve the testing process, so that the time taken for performing the test operation will be minimized. The errors will be eliminated so that the quality will be enhanced because of satisfying the user requirement.

2. RELATED WORK

For conducting analysis on the change both static and dynamic features are used.

Xiaoxia R et al describes that the test is selected after static change analysis is performed [6]. Apiwattanapong T et al says that they have conducted only that set of analysis [5]. Sneed M H designed a process to find how far the communication exist between the different functionality when it is under execution [7]. Mary Jean Harrold et al have designed a framework and they have focussed on some experimental process such as the outcome after change and the process for providing support to the maintenance part [8]. Lingming Zhang et al proposed a general architecture by which the input test data can find the block where the outcome is going wrong by priotitization [9]. Arvinder Kaur performed an

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

experimental work on coverage of code by means of a genetic algorithm which gives a test output that will contain an ordered form [11]. Milos Gligoric et al have mainly concentrated their work towards the statements that performs the function and how they are being checked by providing various test data [10]. Neerja Gupta describes the techniques used for object oriented programs and for aspect oriented programming [12]. Abhishek Singhal explains about the approach which he has designed for prioritization and also tried it with his own algorithm to overcome the time for that purpose [13]. T.Prem Jacob et al have been investigated modified genetic algorithm for selection of test cases and prioritization [14]. J. Andrews et al have been investigated different strategies for improving the performance of a given application [15].

3. SELECTION PROCESS

The process is subdivided as

a) Passing information between the functionalities and their outcome
b) Evaluation after the change
c) Final selection

a) Passing information between the functionalities and their outcome

There will be a set of different functionalities in the corresponding application which is to be evaluated. The major process is to trace the behavior when the functionalities communicate with each other during the execution. The figure 1 shows the different test input called as Test Cases and the set of function that are affected during the process.

Depends upon the functional communication Figure 1 describes the path which gives the information about the various classes within each module. The interaction between methods and their behavior is traced. The captured method is invoked while executing the application.

We are going to enable the tracking of code using the steps.

Start: The initial process is to set the values for the variable in the executional environment.

Initialization of Events: The event is initialized using function delegates.

Extract events: The event related information is extracted.

b) Evaluation after the change

In this research we have considered the java or C# code for communication between functions. Some of the process used to detect the affected method is listed.

- Identification of methods which are not modified after performing more set of comparison .
- At last it is going to analyse semantically.

Modification can be done either at syntatic level or semantic level. Even if there is any change once the modification can be done it will not directly affect the other.

Semantic analysis

Comparing objects from both original and new versions of a program written in C#, if an interface undergoes any set of modifications and what all are the internal changes that are done inside the interface is noted.

The Figure 1 shows the initial part of the code and the changed one. The two methods that are declared in the initial part is plusop() and minusop(). These two methods are inherited from class Arit. The initial part is used by the user. Since the client will be using the initial part he will be affected due to any small modification. The changed part of an interface consists of the following changes. plusop() and minusop() methods are modified by passing parameters (ie) int plusop(opx,opy) int minusop(opx,opy).

As shown in Figure 4, in the old version there are methods called sx1() and sx2() that are present in interface Ias and methods sy1() and sy2() in the interface Ias and Ibs .All the methods will just print some string ,but they are not going to return any value. Here the change is implemented in the initial part due to that the operation will differ.

C) Suitable Selection

Finally a small group of test cases that will cover the instruction newly generated will be listed out.

4. RESULTS

The C# code which defines an functionality is used for doing the work. As described in the Figure 3 a code which is having an interface called Isah which has the methods plusop() and minusop() is considered. The outcome is evaluated and the output is stored in the repository. My making change that is given by the client in opplus() and opminus() a new part of the functionality is generated.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The process is repeated for the code shown in Figure 4 here the change is done due to multiple Interface. The code has two Interface Ias and Ibs which is having methods. Some set of strings will be displayed by the methods present in the interface. Improvement in the FDT(Fault Detection Ratio) due the better performance that is given by the entire selection framework.

The Figure 5 describes the Test case execution for both the original and modified program for both Interface and Multiple Interface. Normal selection and the ETCS is compared in Figure 6 with the execution of test. Compared to the other process ETCS select tiny list of test so fault produced is less and diminish in time will increase the performance.

5. CONCLUSION

This paper deals with ETCS approach for considering only c#.net application. The paper does not cover the other aspects of object oriented concepts of different programming languages especially such as polymorphism, Inheritance etc.

To perform an analysis of initial and the code where change is updated architecture is proposed. The framework ETCS will extract the dynamic behavior of the program and analysis is done to find out the changes. The sample program from C# is taken and modified. The code is checked both for change in Interface and change in Multiple Inteface. The corresponding test cases that affect the changes are listed out and finally the result will be of the selected set of test case for testing .

In future the work can be extended by considering different aspects of different programming languages. In future the work can also be extended by extending the ETCS framework for considering different applications.

REFERENCES

[1] Pravin A and Srinivasan S (2013), "Effective Test Case Selection And Prioritization in Regression Testing", Journal of Computer Science, Vol. 9, Issue 5 pp.654-659.

[2] Pravin A and Srinivasan S (2013), "An Empirical Study on Fault Localization and Effective Test Case Selection By Neural Network", Indian Journal of Computer Science and Engineering, Vol. 3, Issue 6, pp. 812-817.

[3] Pravin A and Srinivasan S (2012), "An Efficient Programming Rule Extraction and Detection of Violations in Software Source Code Using Neural Networks", IEE-Fourth International Conference on Advanced Computing , ICoAC 2012 MIT, Anna University,Chennai, pp.1-4.

[4] Pravin A and Srinivasan S (2012), Efficient Algorithm Selection for Detecting Suitable Test Case Prioritization", International Conference on Recent Advances and Future Trends in Information Technology, Punjabi university, Punjab, proc. International Journal of Computer Applications, No.7 , pp. 28-31.

[5] Apiwattanapong T, Orso A, and Harrold M J, "JDiff: A Differencing Technique and Tool for Object-Oriented programs", Journal of Automated Software Engineering, Vol. 14, No. 1,March 2007, pp.3-36.

[6] Xiaoxia R, Barbara G R, Maximilian S and Frank T, "Chianti: A Prototype change impact analysis tool for Java", Proceedings of 27th International Conference on Software Engineering (ICSE), St. Louis, USA, May 15-21, 2005, pp.664-665.

[7] Sneed M H, "Selective Regression Testing of a Host to DotNet Migration", Proceedings of the 22nd IEEE International Conference on Software Maintenance (ICSM), Philadelphia, Pennsylvania .

[8] Alessandro Orso, Taweesup Apiwattanapong, and Mary Jean Harrold, "Leveraging Field Data for Impact Analysis and Regression Testing", Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering, pp 128-137.

[9] Lingming Zhang, Dan Hao, Lu Zhang, Gregg Rothermel, Hong Mei ," Bridging the Gap between the Total and Additional Test-Case Prioritization Strategies ",ICSE 2013,San Francisco,CA,USA.

[10] Milos Gligoric, Alex Groce, Chaoqiang Zhang, Rohan Sharma, Mohammad Amin Alipour, and Darko Marinov,"Comparing Non-adequate Test Suites using Coverage Criteria", ISSTA '13, July 15-20, 2013, Lugano, Switzerland.

[11] Arvinder Kaur," A Genetic Algorithm For Regression Test Case Prioritization Using Code Coverage", International Journal on Computer Science and Engineering (IJCSE), Vol. 3 No. 5 May 2011.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[12] Neerja Gupta, " A Survey on Regression Test Selection Techniques on Aspect-Oriented Programming", International Journal of Computer Applications Volume 59– No.11, December 2012

[13] Abhishek Singhal," A Novel Approach For Priortization Of Optimized Test Cases" International Journal on Computer Science and Engineering (IJCSE), Vol. 4 No. 05 May 2012.

[14] T. Prem Jacob, T. Ravi. Optimal Regression Test Case Prioritization using genetic algorithm. Life Sci J 2013;10(3):1021-1033] (ISSN:1097-8135). http://www.lifesciencesite.com. 149

[15] J. Andrews, T.Sasikala , "Efficient framework architecture for improved tuning time and normalized tuning time", WSEAS Transactions on Information Science and Applications, Vol.10, No.7, July 2013, pp.no 230-240.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Test cases	Paths	Paths	Paths
TC01	A1.C1:M0	A1.C4:M3	A2.C7:M20
TC02	A1.C2:M2	A1.C3.M4	
TC03	A1.C8:M5	A2.C10:M24	
TC04	A1.C7:M9	A1.C9:M17	A2.C8:M24

Figure 1 Functional Communication Path And Test Set

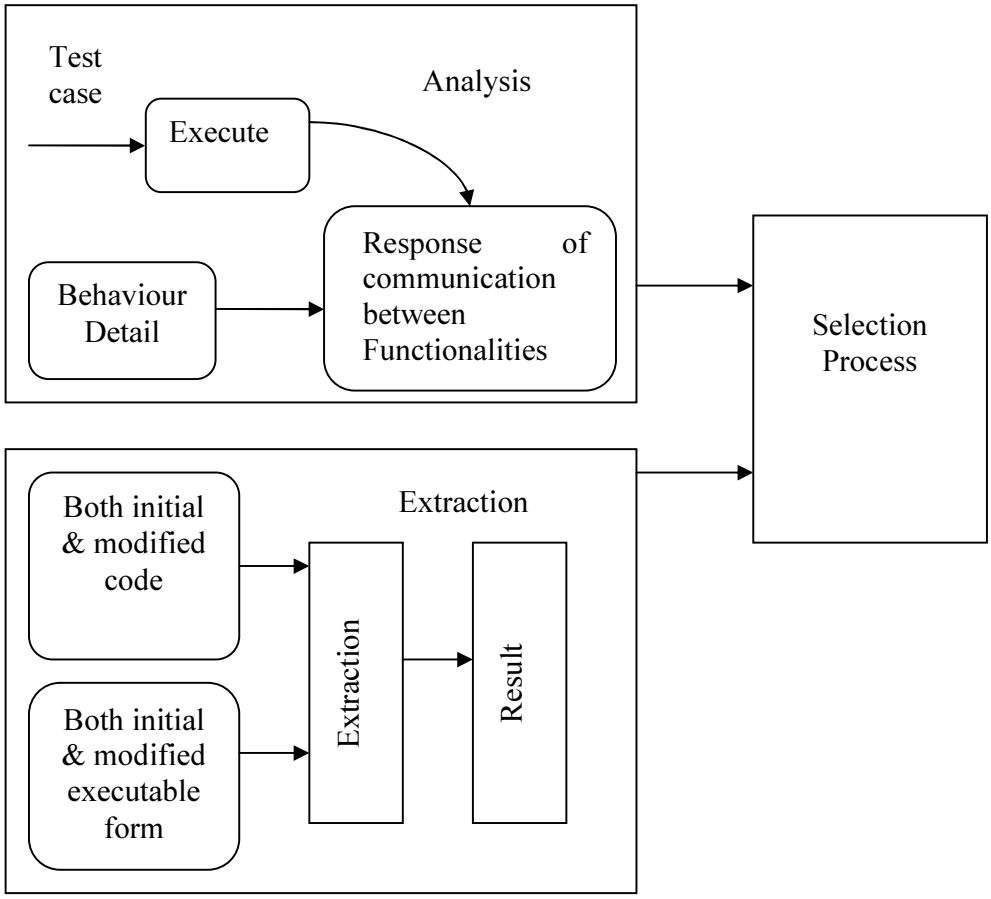

Figure 2 Block Diagram Of Etcs

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
interface  Isah
{
void plusop();
void minusop();
}
class Arit:Isah
{
public int opa,opb,opc;
Public void plusop()
{
opa =Iint16.Parse(Console.ReadLine());
opb =Iint16.Parse(Console.ReadLine());
opc = opa + opb;
console.WriteLine(opc);
}
public void minusop ()
{
opa =Iint16.Parse(Console.ReadLine());
opb =Iint16.Parse(Console.ReadLine());
opc = opa - opb;
console.WriteLine(opc);
}
public static void Main()
{
Arit op=new Arit();
Isah   reop=(Isah)op;
reop. plusop();
reop. minusop();
}
}
```

(a)Original source code

```
interface  Isah
{
int plusop(opx,opy);
int minusop(opx,opy);
}
class Arit:Isah
{
public int opa, opb, opc;
public int plusop (int opa,int opb)
{
opc = opa + opb;
console.WriteLine(opc);
return opc;
}
public int minusop(int opa,int opb)
{
opc = opa - opb;
console.WriteLine(opc);
return opc;
}
public static void Main()
{
Arit op =new Arit();
Isah   reop=(Isah) op;
reop. plusop (2,3);
reop. minusop(2,3);
}
}
```

(b)modified program

Figure 3:Changes In Interface

```
interface  Ias
{
void sx1();
void sx2();
}
interface Ibs
{
void sy1();
void sy2();
}
class Texa:Ias
{
public void sx1()
{
console.WriteLine("This is SX1");
}
public void sx2()
{
console.WriteLine("This is  SX2");
}
class Texb:Ibs
{
public void sy1()
{
console.WriteLine("This is SY1");
}
public void sy2()
{
console.WriteLine("This is  SY2");
}

public static void Main()
{
Texa pr=new Texa();
Texb pr1=new Texb();
Ias   acc=(Ias)pr;
acc.sx1();
acc.sx2();
Ibs   acc1=(Ibs)pr1;
acc1.sy1();
acc1.sy2();
}
}
```

(a) Original source code

```
interface  Ias
{
string sx1();
string sx2();
}
interface Ibs
{
string sy1();
string sy2();
}
class Texa:Ias
{
public string sx1()
{
```

Journal of Theoretical and Applied Information Technology
10th March 2015. Vol.73 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
string sdep;
sdep=”cse”;
return sdep;
}
public string sx2()
{
String sdep1;
Sdep1=”ece”;
return sdep1;
}
class Texb:Ibs
{
public string sy1()
{
string sdep2;
sdep2=”mech”;
return sdep2;
}
public string sy2()
{
string sdep4;
```

(a)original program

```
sdep4=”prod”;
return sdep4;
}

public static void Main()
{
Texa pr=new Texa ();
Texb pr1=new Texb ();
Ias   acc =(Ias)pr;
acc.sx1();
acc.sx2();
Ibs   acc1=(Ibs)pr1;
acc1.sy1();
acc1.sy2();
}
}
```

(b)modified program

Figure 4:Change Due To Multiple Interface

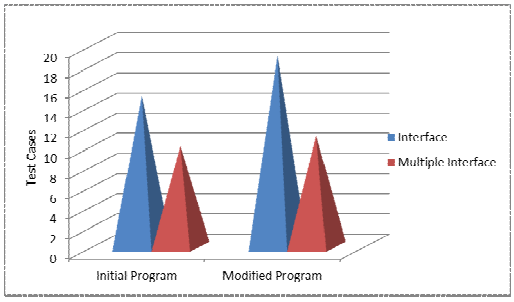

Figure 5. Test Executed for Original and Modified Program

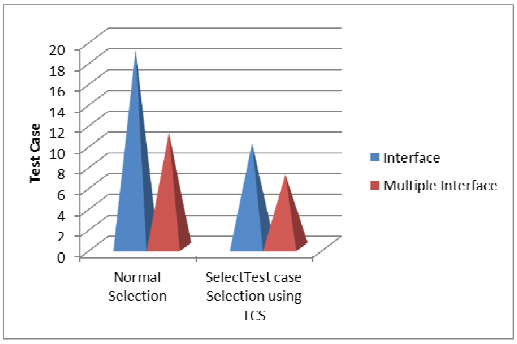

Figure 6. Normal selection vs Selection using ETCS