International Journal of Computer Science and Informatics

Volume 1 | Issue 3 Article 7

January 2012

An Efficient Regression Testing By Computing Coverage Data For Software Evolution

Machani SivaPrasad
CSE dept, JNTUA College of Engineering, Anantapur, Andhra Pradesh,India, msivaprasad88@gmail.com

Recommended Citation
SivaPrasad, Machani (2012) "An Efficient Regression Testing By Computing Coverage Data For Software Evolution," *International Journal of Computer Science and Informatics*: Vol. 1 : Iss. 3 , Article 7.
DOI: 10.47893/IJCSI.2012.1033
Available at: https://www.interscience.in/ijcsi/vol1/iss3/7

Available online at www.interscience.in

An Efficient Regression Testing By Computing Coverage Data For Software Evolution

Machani SivaPrasad
M.Tech in CSE dept, JNTUA College of Engineering, Anantapur, Andhra Pradesh,India
E-mail : msivaprasad88@gmail.com

Abstract - Software systems is evolve continuously during development and maintenance. After software is modified regression testing is applied to software to ensure that It behaves intended and modifications not negatively impacts its original functionality .It is time consuming to rerun test suite T of program Pi on modified program Pi+1.So there are many regression testing techniques are there for doing regression testing. These are based on coverage data. So computing coverage data for Pi+1 without rerunning all test cases is the problem for doing regression testing of program Pi+1.This paper proposed a new approach that computes coverage data with selecting test cases T' for the subsequent versions of the software .By computing coverage data for subsequent version of software on without rerunning entire test suit T we can improve overall time taken to retest the evolving software using Regression testing.

This paper focus on improving the performance of regression testing for software evolve continuously during maintenance, by implementing a new approach for regression testing by computing coverage data for evolving software using dataflow analysis and execution tracing .

Keywords*-Software Engineering, Software testing , Regression testing ,Coverage data.*

I. INTRODUCTION

As software systems mature maintenance activities is dominant. Reports estimate that regression testing consumes as much as 80 percent of the testing budget [1] and 50 percent of development effort in development life cycle spent on maintenance because evolving software by inducing changes[2][3]. Software System is continuously evolve because of adaptive, corrective and perfective. Thus the more effort is required to verify that changes induced affects it original functionality of the software .So regression testing is applied to the modified version of the software to ensure its original behavior. One approach to regression testing saves the test suite T used to test one version of the program Pi and uses it to test the next (modified) version of the program Pi+1. As it is sometimes too expensive or time-consuming to rerun all of T on Pi+1, researcher s have developed techniques to improve the efficiency of the retesting . For example, regression test selection (RTS) techniques select a subset of T as T' and use it to test Pi+1(Nos.4,5,6) . If the RTS technique is safe, then the test cases that it omits(i.e., T – T') will give the same results on Pi and Pi+1, and thus, do not need to be rerun on Pi+1Studies have shown that RTS can be effective in reducing the time and cost of regression testing.

Many of these regression testing techniques use coverage data collected when testing Pi using T to assist the testing that should be performed on Pi+1. For example, several RTS techniques collect coverage data, such as which statements[7] , branches, or methods [8][9] are covered when Pi is executed with T, for testing Pi+1. As subsequent versions of Pi are created, coverage data of predecessor version are needed for regression testing tasks. In presentations of these regression testing techniques, especially to practitioners, there are usually questions about how the coverage data will be obtained for these subsequent versions, when only a subset of T is used to test Pi+1. The coverage data on Pi for those test cases in T that are not run on Pi+1 (i.e., T –T') cannot simply be copied for Pi+1 unless the development environment maintains a mapping between entities (such as statements, branches, and methods) in Pi and entities in Pi+1. Because this mapping is not typically maintained, another approach for obtaining the coverage data for test cases in T – T' is needed[10].

In this paper we developed new approach for regression testing by computing coverage data for selecting test case to achieve savings in testing time of continuously evolving software. Our approach involves several steps First step is indentifying the program entities in the program for which to compute the coverage data in the prrogram. Second stepby applying execution tracing and dataflow analysis at dynamically computing coverage data for identified program entities in the step1. Third step apply first two steps on different versions of the software and compute coverage data to identifies changed entities in the two versions of the application and selected test cases to test changed program entities in the software application .

```
Public class Grade{
     Public char ComputeGrade(int
totalMarks, int midMarks )  {

S1       char Grade;
S2   if(totalMarks>70){
S3         if(midMarks>80){
S4           Grade='A';
           }else{
S5               Grade='B';
             }
           }else
S6
if(totalMarks>60&&totalMarks<70){
S7         Grade='C';
      } else {
S8         Grade='D';
        }
S9 return Grade;
    }
}
```

```
Public class Grade{
     Public  char  ComputeGrade(int
totalMarks, int midMarks )  {

S1       char Grade;
S2   if(totalMarks>70){
S3         if(midMarks>80){
S4           Grade='A';
           }else{
S5               Grade='B';
             }
           }
S6elseif(totalMarks>60&&totalMarks<70)
{
S7     if(midMarks>80) { /*change*/
S8         Grade='C';
         } else {
S9         Grade='D';
        }
      } else  {
 S10       Grade='E';
     }
 S11 return Grade;
 }
}
```

```
Public class Grade{
     Public  char  ComputeGrade(int
totalMarks, int midMarks )  {

S1       char Grade;
S2   if(totalMarks>70){
S3         if(midMarks>80){
S4           Grade='A';
           }else{
S5               Grade='B';
             }
           }
S6elseif(totalMarks>60&&totalMarks<70)
{
S7     if(midMarks>80) {
S8         Grade='C';
       }          else               /*change*/
S9if(midMarks<80&&midMarks>70){
S10          Grade='D';
        }else {
 S11         Grade='E';
      } else  {
 S12       Grade=\'F';
      }
 S13 return Grade;
 }
}
```

Fig1.Three versions of the program Pi, Pi+1,Pi+2

2. RELATED WORK

In this section we consider a related work that illustrate the problem we are solving. In the fig 1 there three versions of the programs Pi, Pi+1, Pi+2 that evolves with changes to one another. Now test suit T in fig2 is used to test the program Pi and found that coverage data CDi in the form of matrix as shown in fig2.after doing some changes to the program Pi. it evolves to Pi+1 now we want to test program Pi+1 to ensures that previous functionality does not affect apply regression testing without rerunning all test cases by select some test case as T' we need coverage data of program Pi that is available as CDi. so for doing regression testing for program Pi we have no problem. Now program pi+1 is evolves after adopting some changes as program Pi+2.Now we want to check the previous functionality of the program Pi+2 does not affect by doing regression testing So coverage data CDi+1 for the program Pi+1 is needed to perform regression test on program Pi+1.There two ways to compute the coveragedataCDi+1forprogram Pi+1.One is rerun test suit T on program Pi+1 which is time consuming process for regression test. So Second our proposed method with running application with our frame work by some techniques in order to achieve greater savings in time of regression testing to continuously evolving software.

Test cases	Input
t1	Total Score=71,midScore=81
t2	Total Score=71,midScore=71
t3	Total Score=64,midScore=82
t4	Total Score=51,midScore=72

Fig 2 Test Suite T

	Coverage Data								
	S1	S2	S3	S4	S5	S6	S7	S8	S9
T1	1	1	1	1	0	0	0	0	1
T2	1	1	1	0	1	0	0	0	1
T3	1	1	0	0	0	1	1	0	1
T4	1	1	0	0	0	1	0	1	1

Fig 3 Coverage data CDi of program Pi on T

3. COMPUTING COVERAGE DATA

Our proposed system to compute coverage data for the software applications with out rerunning testsuit using execution tracing and dataflow analysis . Studies shown that by computing coverage data without rerunning entire test suite by selected test cases can provide significant savings in regression testing time. Thus, our proposed system can be an important part of an efficient regression testing process.

Proposed system involves 2 phases to compute the coverage data without rerunning entire test suite. First step is indentifying the program entities in the program for which to compute the coverage data in the program called as coverage criteria. Second step by applying execution tracing and dataflow analysis at dynamically computing coverage data for identified program entities in the step1.

A. Coverage Criteria

There are different coverage criterias basing upon program entites that considered while testing the program.some program entites considered in our approach are stament methods, classes, exceptions. Different coverage criteria is described below.

i Statement Coverage

This criteria reports whether each executable statement is encountered. Declarative statements that generate executable code are considered executable statements. Control-flow statements, such as if, for, and switch are covered if the expression controlling the flow is covered as well as all the contained statements. Implicit statements, such as an omitted return, are not subject to statement coverage

ii Method Coverage

This criteria reports whether you invoked each function or procedure. It is useful during preliminary testing to assure at least some coverage in all areas of the software.

iii Call Coverage

This Criteria reports whether you executed each function call. The hypothesis is that bugs commonly occur in interfaces between modules.

iv Condition Coverage

Condition coverage reports the true or false outcome of each condition. A condition is an operand of a logical operator that does not contain logical operators. Condition coverage measures the conditions independently of each other.

B Execution tracing and Dataflow analysis

i Execution tracing

An execution trace of program P for some test suite T is the sequence of program entites is executed against T. for above example the execution traceis shown below

Test cases	Execution trace
t1	S1,S2,S3,S4,S9
t2	S1,S2,S3,S5,S9
t3	S1,S2,S6,S7,S9
t4	S1,S2,S6,S8,S9

Fig 3 Execution trace of program Pi on T

ii Dataflow Analysis

It is the process of collecting information about the way the variables are used , defined in the program. Analysis is done at basic block granularity.Dataflow analysis can be performed at both static and dynamic levels.But in our approach we use dynamic dataflow analysis to compute coverage data

i Static dataflow analysis

In static level Identify potential defects, to Analyze source code with out execution of code

ii Dynamic dataflow analysis

In dynamic level involves actual program execution..Identify paths to execute them.Paths are identified based on data flow diagrams.dynamic dataflow analysis is carried by fallowing steps

1. Execute the program
2. Draw a data flow graph from a program.
3. Select one or more coverage criteria.
4. Identify paths in the data flow graph satisfying the coverage criteria.

4. EXPERIMENT DESIGN

To evaluate our technique, we develop an java frame work called Dynamic Code Analyzer (DCA). Dynamic Code Analyzer that implements our techniques used it to conduct empirical studies to compute coverage and estimates time for computing coverage data for regression testing. For our experiment we used three versions GDownloader.

GDownloader is an downloading software developed in java that has six versions and 3,000-4,000 lines of code, depending on the version. Some of these versions have additional versions that can be obtained by enabling different numbers of faults: v1 has seven versions, v2 has seven versions, v3 has 10 versions, and v5 has nine versions. Using these versions, we performed our studies on 3 versions of GDownloader. By testing GDownloader version 1,version2,version3. achieve, on average coverage of 67.76 , 77.15,88.15 percent the results shown in figure .

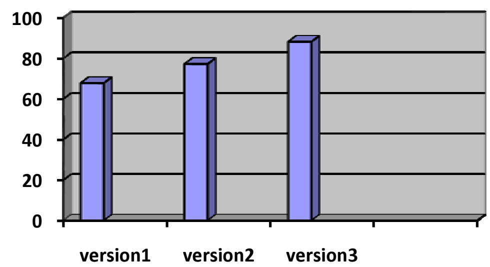

Fig 5 : Results of experiment for computing coverage data

After computing coverage data can compare the coverage data of 3 versions of software to identify the changed entites for doing regression testing .so that run testcases of that changed entites to regression test of the software. The results of our experiment shows that there will be significant savings in time of regression testing by computing coverage data. The results of time taken for doing regression are shown in fig6

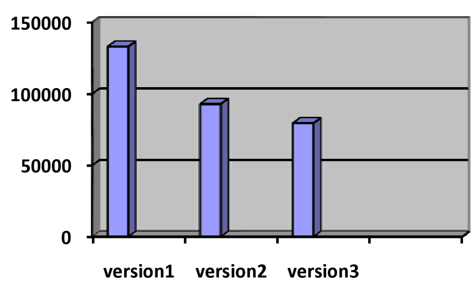

Fig5 : Results of experiment for doing regression test

5. CONCLUSION AND FUTURE WORK

In this paper we presented a new approach for doing regression testing by computing coverage data without incurring the expense of rerunning entire test cases. so we can achieve greater savings in time testing continuously evolving software by efficient regression test. The frame work developed for testing application consists all our technique to computing coverage data for different versions software using execution trace and dynamic dataflow analysis achive greater savings in time of regression testing for continuously evolving software.

In future work we extend obtain the dynamic slice by after tracing and dynamic analysis to achieve more performance in regression testing . Another scope for the extending this work is by considering the test case prioritizations along with selection for improve the quality of the testing.

6. REFERENCES

[1] C. Kaner, “Improving the Maintainability of Automated Test Suites,” Proc. Quality Week Conf., May 1997.

[2] B. Beizer, Software Testing Techniques. Van Nostrand Reinhold, 1990.

[3] H.K.N. Leung and L.J. White, “Insights into Regression Testing,”Proc. IEEE Conf. Software Maintenance, pp. 60-69, Oct. 1989

[4] T. Ball, “On the Limit of Control Flow Analysis for Regression Test Selection,” Proc. ACM SIGSOFT Int’l Symp. Software Testing andAnalysis, pp. 134-142, Mar. 1998.

[5] Y.F. Chen, D.S. Rosenblum, and K.P. Vo, “Testtube: A System forSelective Regression Testing,” Proc. 16th ACM/IEEE Int’l Conf.Software Eng., pp. 211-222, May 1994.

[6] A. Orso, N. Shi, and M.J. Harrold, “Scaling Regression Testing toLarge Software Systems,” Proc. 12th ACM SIGSOFT Symp.Foundations of Software Eng., pp. 241-252, Nov. 2004.

[7] F. Vokolos and P. Frankl, “Pythia: A Regression Test Selection Tool Based on Text Differencing,” Proc. IEEE Int’l Conf. Reliability, Quality and Safety of Software Intensive Systems, pp. 3-21, June 1997.

[8] A. Orso, N. Shi, and M.J. Harrold, “Scaling Regression Testing to Large Software Systems,” Proc. 12th ACM SIGSOFT Symp.Foundations of Software Eng., pp. 241-252, Nov. 2004.

[9] G. Rothermel and M.J. Harrold, “A Safe, Efficient Regression Test Selection Technique,” ACM Trans. Software Eng. and Methodology,vol. 6, no. 2, pp. 173-210, Apr. 1997.

[10] Pavan Kumar Chittimalli and Marry Jean Harrold,” Recomputing Coverage Information to Assist Regression Testing". IEEE Transaction July/ August 2009.

❑❑❑