International Journal of Recent Engineering Science (IJRES),
ISSN: 2349-7157, *volume1 issue2 March to April 2014*

Version Specific Test Suite Prioritization using Dataflow Testing

M.Kalaiyarasan, Dr.H.Yasminroja
Research scholar, Assistant Professor,
Department of Computer Science and Engineering,
JJ College of Engineering and Technology, Trichy, India

Abstract

Test case prioritization optimizes the ordering of test cases to be executed to meet some criteria like maximum code coverage or rate of fault detection. While maintenance, regression testing is performed on the modified code to build confidence in the code and to ensure that modification has not introduced any new errors. One approach while regression testing is to retest all the test cases which were used while development testing. This exhaustive approach is usually non optimal as the modification may or may not affect the whole code and it is expensive approach. Regression test case prioritization techniques find a subset of prioritized test cases from the test suite used while development testing so that software testers may test the modified code effectively and efficiently and yet in an inexpensive manner. While maintenance different versions of software may be created depending on the type of modification and test cases may be prioritized according to the version of the software. Here in this paper four different categories of software modifications have been identified and regression test suite prioritizations according to the versions thus created have been suggested using dataflow information.

1. Introduction

Software maintenance is the most expensive phase of software development. Regression testing being an important activity performed during maintenance phase can account for a large proportion of software maintenance budget, and it can be very expensive [5]. Regression test Suite Prioritization attempts to reorder the execution of test suite, so that those tests with the highest priority according to some established criterion are executed earlier in the regression testing process than those with lower priority [4, 8].

While maintenance modifications are performed on the software in order to incorporate new features. These modifications may be of different categories, and may create different versions of the software. Depending upon the type of modification the test case effectiveness to detect faults may vary. So, we are interested in test cases that perform well enough to detect faults in a specified version of the software. As test suite may vary for different versions of the software the test cases have to be prioritized according to the versions.

In this paper, we have discussed the use of data flow testing to determine the priority of regression test suite for different version of software created while maintenance. Although there may be variety of modifications, we have considered four broad categories of modifications for our discussion. Code snippets along with their flow graphs have been employed to depict the category of modifications and an analysis for prioritizing test cases using data flow testing has been discussed in section 4 of this paper. Section 2 covers data flow testing and some useful definitions. Section 3 discusses background and related work. Section 5 summarizes the concepts presented in this paper. Section 6 presents conclusion and future work.

2. Literature survey

This section discusses the data flow testing concepts, anomalies and related definitions. Data flow testing is a structural testing approach and is basically a verification technique which uses the source code to guide the selection of test data . In dataflow testing we look for the use of variables and we focus on

i. Statements where variables receive values
ii. Statements where these values are used or referenced [1]

Flow graphs are used as a basis for dataflow testing as in the case of path testing. Variables used in the program may be defined and referenced throughout the program. We may have few define/referenced anomalies [1].

i. A variable is defined but not used.
ii. A variable is used but never defined.
iii. A variable is defined twice before it is used.

2.1. Definitions

We consider a program P with a flow graph F, and a set of program variables V.

i. DEF (v, n) : A node n of flow graph F where the variable v ε V is defined.
ii. USE (v, n) : A node n of flow graph F where the variable v ε V is used.
    a. denoted P if n is a predicate statement
    b. denoted C if n is a computational statement.
iii. DU Path (definition use): A path in a flow graph F where at the initial node n of the path a variable v ε V is defined and at the final node m of the path the variable v ε V is used/referenced.
iv. DC Path (definition clear): A DU path of the flow graph F in which no node between the initial node n and final node m of the path is a DEF (v, n) for the considered variable v ε V.

Dataflow testing approach provides a set of DU paths for which test cases have to be generated. There may be some DU paths which are not DC paths and for such paths test cases must be written specifically.

3. Background and related work

Various techniques for regression test suite prioritization have been proposed in research literature by several researchers. These techniques have addressed test case prioritization according to rate of fault detection or code coverage capabilities. Many prioritization techniques have been described in the research literature, and they have been evaluated through various empirical studies [3, 7, 9, 12, 13, 14, and 15].

Rummel, Kapf hammer, and Andrew Thall. (Rummel et al. 2005) suggest that test suite can be prioritized according to all DU's with minimal time and space overhead. Huchins et al (Huchins et al 1994) showed that both control flow and data flow testing can be very useful at instigating the generation of high yield test cases that may be otherwise omitted. Frankl et al. (Frankl et al 1997) suggest that mutation based criteria is better than all DU criteria when desirable code coverage level is high. Jones et al (Jones and Harold, 2001) describe a technique for prioritization of test cases for use with the modified condition /decision coverage (MCDC) criteria. Srivastava and Thiagarajan (Srivastava and Thiagarajan, 2002) present a technique based on basic block coverage using both feedback and change information. Rothermel et al. (Rothermel et al, 1999, 2001) and Elbaum et al. (Elbaum et al, 2001b, 2002) were the first to provide formal definition of prioritization problem and present metric for measuring the rate of fault detection of test suite.
Among the papers mentioned above only few (Rothermel et al, 1999, 2001; Elbaum et al, 2001b, 2002) report results of studies or experiments explicitly accessing the ability of prioritization techniques to improve rate of fault detection relative to each other or unpriortized test cases.

4. Version specific prioritization approach

While maintenance various activities are performed on the code like simple modifications, adding new functionalities, removing some old functionalities etc. Thus different versions of the software may be created depending upon the type of modification. Dataflow testing can play a vital role in determining the definition and use of new and old variables after modifications in the code. New DU paths may emerge after modification and these paths need to be tested specifically to ensure that no new faults have crept in the code.

Here in this paper, we are interested in prioritizing the regression test suite according to the version of the code created while maintenance using dataflow information. In this work we identify four broad categories of modifications for our discussion and we describe test case prioritization approach using dataflow information.

4.1. Category 1

Table 1: Code Snippet

```
1. Start
2. Read x, y
3. If y >0 then goto 6          //modification
4. z := y
5. goto 7
6. z :=y + x
7. Stop
```

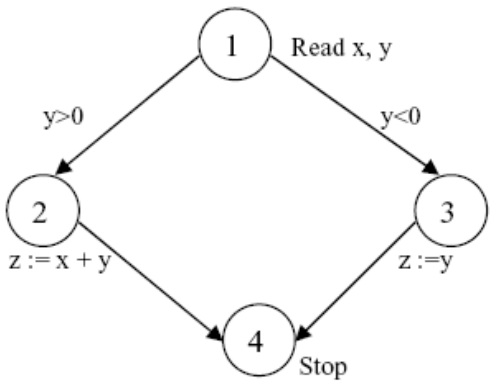

Figure 1: Flow Graph for Category 1 modification

In this category 1, we consider the type of modification in which small changes are done in any line of the code, let's say line 3 of the Table 1. Then it will affect the use of variable y which in turn will affect the computation of variable z.

We recommend that all DU paths which relate to variable y must be tested to ensure that the modification has not introduced any new error. Accordingly, we can say that test cases related to variable y must be assigned highest priority.

4.2. Category 2

Table 2: Code Snippet

```
1. Start
2. Read x, y
3. If y >=0 then goto 8
4. z := y
5. if z >0 then goto 7    // new code fragment added
6. x := x + y
7. x := x – y
8. z := y + x
9. Stop
```

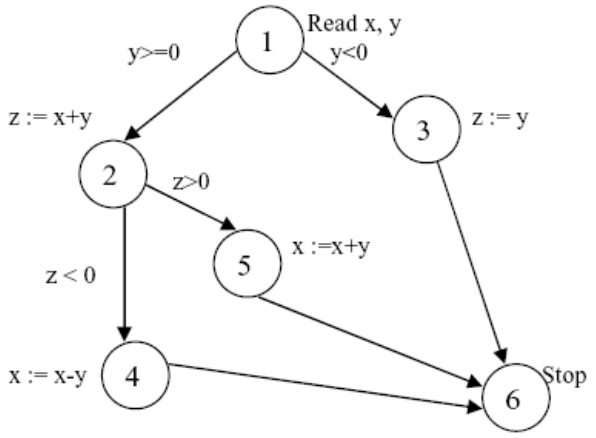

Figure 2: Flow Graph for Category 2 modification

In this category it may happen that a new code fragment is added to the original code while maintenance. This can introduce new DU associations which may not be DC paths. In the above solve code snippet new code fragment has been added in line number 5 through 7. This modification could have redefined the variable y which can be reason to a potential fault. We can say that for category 2 we have to identify newly introduced DU paths due to the modifications. We must ensure that these new introduced DU paths are DC paths. These new paths must be tested well and we recommend that test cases which execute these new DU paths must be given highest priority while regression testing.

4.3. Category 3

Table 3: Code Snippet

```
1. Start
2. Read x, y
3. If y >=0 then goto 8
4. z := y
5. goto 7
6. z  := y + x
7. Read m, n          //new feature added
8. If m > 0 then goto 10
9. n := -m
10. n := m
11. if n < 0 then goto 7
12. Stop
```

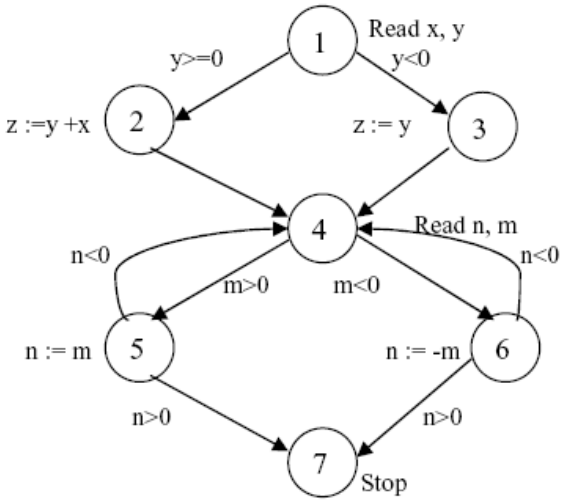

Figure 3: Flow Graph for Category 3 modification

In this category we have considered the case when modification introduces an entirely new feature to the existing code. It may be the case that this new feature does not uses variables or computed variables from the old code, i.e. the newly added feature functions entirely on its own. For this category we have to identify all DU paths and generate new test cases related to these DU paths. We can assign equal priority to both of the test cases, i.e. that of the original code and that of the added module. We can say that we have to generate new test cases for the added feature and both the test cases (new and old) should be given equal importance while testing.

4.3. Category 4

Table 4: Code Snippet

```
1.  Start
2.  Read x, y
3.  If y >=0 then goto 8
4.  z := y
5.  goto 7
6.  z  := y + x
7.  Read m, n          //new feature added
8.  If m > 0 then goto 10
9.  n := m + z
10. n := m - z
11. if n < 0 then goto 7
12. Stop
```

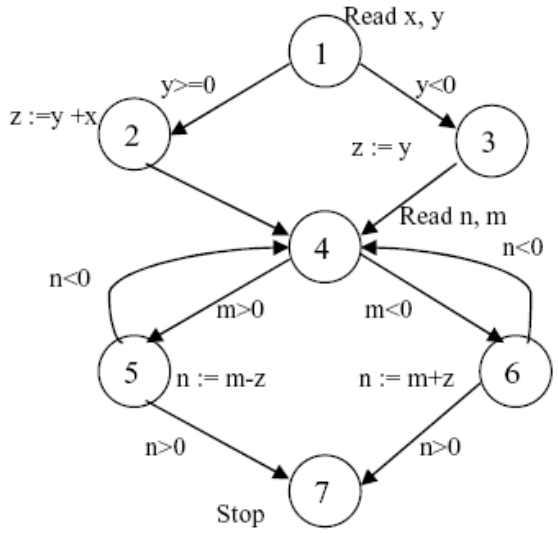

Figure 4: Flow Graph for Category 4 modification

If the new feature added uses some variables (computed or defined) from one of the old features of the software then it must be ensured that the variable referenced in the new feature has a consistent value. As shown in the above given code snippet and its related flow graph the computed variable z from the old feature is referenced (line 9 and 10) in the newly added feature. Before z is used its value must be consistent, to ensure this the old features Du paths regarding variable y must be tested well enough. As variable z could have been redefined in the new function, a new DU association could have been introduced. We can say that test cases related to such new DU paths, and test cases related to the DU path of old features variables (used in computation of the referenced variables in new feature) must be assigned highest priority so as to improve fault detection efficiency of the regression test suite.

5. Discussion on the criteria

Version specific test case prioritization ensures that test cases are executed according to the version of the software. It may be the case that for any new version almost all prioritized test cases may be new. As we have proposed in our approach different categories of modifications may demand different prioritization approach's for the test cases. In the first category where small modification is done in any line of code, the definition and may be the use, of variables on that line may be affected ,which in turn may affect the computed variables . Accordingly all DU paths related to the modified variables must be tested well, and test cases related to those DU paths must be assigned highest priority while testing. For category two where new code fragment is added, new DU paths may be introduced, and test cases related to these paths must be assigned the highest priority while testing. For category three and four where an entire new feature is added , it may be the case that the new feature has no interactions with old features (category three) ,then we may assert new test cases for the newly added features must be assigned equal priority to the test cases of old features while testing . If the added feature define or uses, variables or computed variables from the old feature new DU associations may be introduced. So test cases for the newly introduced DU paths must be assigned highest priority.

6. Conclusion and future work

In this paper, we have described a version specific test case prioritizing scheme using data flow information. Though our approach does not take into consideration the code coverage capabilities of the possible test cases, it surely considers the fault detection capabilities of the possible test cases as we have emphasized on the testing of newly introduced DU paths which may be reason for faults. This paper introduces a theoretical foundation for which proper experimentation, analysis and further study is required.

First, we intend to perform experiments regarding this approach to determine its effectiveness, in terms of fault detection or in terms of code coverage. Second, we will compare the experimental results with some existing work regarding test case prioritization, to check for its efficiency and cost effectiveness.

References

1. K.K. Aggarwal & Yogesh Singh, "Software Engineering", New Age International Publishers, Third Edition-2008.
2. Matthew J. Rummel Gregory M. Kapfhammer, Andrew Thall. Towards the Prioritization of Regression Test Suites with Data Flow Information 2005 ACM Symposium on Applied Computing.
3. H. Do, G. Rothermel, and A. Kinneer. Empirical studies of test case prioritization in a JUnit testing environment. In *Proc. Int'l. Symp. Softw. Rel. Engr.*, pages 113–124, Nov. 2004.
4. Sebastian Elbaum, Alexey G.Malishevsky, and G. Rothermel. Prioritizing test cases for regression testing. In Proceedings of the International Symposium on Software Testing and Analysis, pages 102-112. ACM Press, August 2000.
5. A.J.Offutt, J.Pan, K.Tewary, and T.Zhang. An experimental evaluation of data flow and mutation testing. *Softw. Pract. and Exp.*, 26(2):165–176, Feb. 1996.
6. Monica Hutchins, Herb Foster, Tarak Goradia, and Thomas Ostrand. Experiments of the effectiveness of dataflow- and controflow-based test adequacy criteria. In Proceedings of the 16th International Conference on Software Engineering, pages 191{200. IEEE Computer Society Press, 1994.
7. S. Elbaum, D. Gable, and G. Rothermel. Understanding and measuring the sources of variation in the prioritization of regression test suites. In *Proc. Int'l. Softw. Metrics Symp.*, pages 169–179, Apr. 2001a.
8. G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Test case prioritization: An empirical study. In Proceedings of the International Conference on Software Maintenance, pages 179-188, August 1999.
9. S.Elbaum, A.Malishevsky, and G. Rothermel. Incorporating varying test costs and fault severities into test case prioritization. In *Proc. Int'l. Conf. Softw. Eng.*, pages 329–338, May 2001b.
10. Phyllis G. Frankl, Stewart N. Weiss, and Cang Hu. All-uses vs mutation testing: an experimental comparison of effectiveness. J. Syst. Softw., 38(3):235-253, 1997.
11. J. Jones and M. Harrold. Test-suite reduction and prioritization for modi¯ed condition/decision coverage. In *Proceedings of the International Conference on Software Maintenance*, Nov. 2001.
12. A. Srivastava and J. Thiagarajan, "Effectively Prioritizing Tests in Development Environment," Proc. Int'l Symp. Software Testing and Analysis, pp. 97-106, July 2002.
13. S. Elbaum, A. G. Malishevsky, and G. Rothermel. Test case prioritization: A family of empirical studies. *IEEE Trans. Softw. Eng.*, 28(2):159–182, Feb. 2002.
14. G. Rothermel, R. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. *IEEE Trans. Softw. Eng.*, 27(10):929–948, Oct. 2001.
15. W.Wong, J. Horgan, S. London, and H. Agrawal. A study of effective regression testing in practice. In *Proc. Int'l. Symp. Softw. Rel. Engr.*, pages 230–238, Nov. 1997.