International Journal of Computer Applications (0975 – 8887)
Volume 27– No.1, August 2011

Analytical Review of Test Redundancy Detection Techniques

Nagendra Pratap Singh
Computer Science & Engg. Dept.
Naraina Vidya Peeth, Panki, Kanpur

Rishi Mishra
Computer Science & Engg. Dept.
United Bank of India, Candigarh, Panjab

Rajit Ram Yadav
Computer Science. Dept.
UPRTOU , Allahabad

ABSTRACT

This paper presents an analytical review of approaches used by different authors. Coverage information is very important for finding redundancy in test cases. Test redundancy detection reduces the costs of testing and maintenance of software. A redundant test case is a useless part of test suite and it increases the testing cost and test suite size. There are a lot of works that proposed different approaches for test case redundancy detection. Some effective approaches have analysed and this study is very useful for future work in this direction. Some important factors like false positive error, fault detection effectiveness etc. have discussed.

Keyword*:* Test case redundancy, test minimization, false positive error, fault detection effectiveness, coverage information.

1. INTRODUCTION

In software engineering, testing is an important task. With the evolution of software system, its test suite body also grows with the source code [1]. The process of software testing and maintenance is a time consuming and costly, so the size of test suite plays an important role. When the requirement of software systems change, test code might become unmanageable i.e., the ability of tests to check the correctness may become less effective. So there will be the need of using updated test cases. These test cases will cover the modified part of the program, but size of test-suite may increase.

The idea of reducing the size of test-suites is necessary because continuously growth of test cases may affect the cost of maintenance. For test-suite reduction can be achieved by removing the redundant test cases. A redundant test case is one, which will not affect the fault detection effectiveness when removed. There are two types of redundancy: -- first one is semantic and other is syntactic test redundancy. Redundancy that we discussed above is semantic and other is test code duplication. In this work, we are focusing on the semantic redundancy.

The motivation for test suite reduction is reducing the maintenance cost as well as fault detection capability should be constant. So truly redundant test-cases should be identified i.e., there should not be any faults that can be detected by the redundant part of a test-case, and not without. There are various techniques for test redundancy detection (also referred as test minimization). Some approaches are based on “coverage information” which is very common (e.g., [2-7]). As discussed in [8], instead of discarding test-cases out of test-suite, the test cases are transformed such that the redundancy is avoided.

For test redundancy detection, following factors are important:-

- Coverage Information
- False Positive (F^+) Errors
- Fault Detection Effectiveness
- Cost of the technique

Although coverage information can be useful in detecting redundant tests, it may suffer from large number of false-positive errors, i.e., a test case being identified as redundant while it is really not [9]. The work in [9, 10] shows that coverage information cannot be the only source of knowledge to precisely detect test redundancy. Detecting redundancy only based on this information may lead to a test suite which is weaker in detecting faults than the original one.

The problem of finding the smallest test set has been shown to be NP-complete [11]. So to find an approximation to smallest test set, many heuristics were proposed. Some of the works have used the approach of data flow coverage criteria while others have used control flow criteria. Few works have applied a collaborative approach in which semi-automated system were used with human interaction [9, 12, 13].

2. TEST CASE REDUNDANCY

“A **test case** is a set of conditions or variables under which a tester will determine whether an application or software system is working correctly or not. The mechanism for determining whether a software program or system has passed or failed such a test is known as a **test oracle”**. So a test case may be a simple input that can check the software working or it may be a proper plan.
A software test-suite that contains a set of test-cases can have redundant test-cases. These test-cases can cover the same portion of the program and hence they may increase the cost of testing. A simple example for finding the redundancy is given in the next paragraph.
Example: 1-

Suppose there is a program for finding largest number from given two integers and square that number.

```
/*Sample Program for Showing the Test Redundancy*/
1. main()
1. {
1. Integer a, b, c;
2. Input a,b;
2. if (a>b)
3. then c=a*a;
4. else c=b*b;
5. Output c;
6. End
```

Now the *Control Flow Graph* of this program is:-

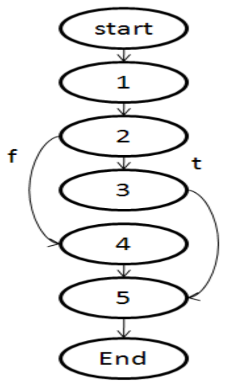

Fig- 1: CFG for the example-1

CFG of a program shows the control flow of that program.
Suppose the following is the given test suite:-

TS= {t1, t2, t3, t4}:
t 1: <a=4, b=3>
t 2: <a=1, b=2>
t 3: <a=3, b=4>
t 4: <a=3, b=3>

According to these test-cases we find the *execution-traces.* An execution-trace(t) shows the sequence of nodes that are touched by test-case 't'. Although execution-trace is used in the automation of the test-method, but we are using it here just for showing the path in an easy way. Now the execution-trace for each of the above test-case is given below:-

Test-case	**Execution-trace(t)**
t 1	Main.start, main.1, main.2, main.3, main.5, main.end
t 2	Main.start, main.1, main.2, main.4, main.5, main.end
t 3	Main.start, main.1, main.2, main.4, main.5, main.end
t 4	Main.start, main.1, main.2, main.4, main.5, main.end

The above execution-traces show that test-cases t2, t3 and t4 are covering the same part of the program. So these are redundant. We can now remove any two of them. Suppose we remove t3 and t4.
So the new test suite is TS'= {t1, t2}. This test-suite is sufficient to cover the complete program. But there is another factor that should be focused, that is fault detection effectiveness of the test-suite. Suppose the program in example-1 is updated and one condition of equality is added, which is shown in the example-2 below.

Example: 2-
Update the program given in example-1, there is an additional condition for the equality of the input values. Due to this CFG and execution-trace of test-cases will also be change.

/* Updated version of the program given in example-1*/

```
1. main()
1. {
1. Integer a,b,c;
2. Input a,b;
2. if (a>b)
3. then c=a*a;
4. else if(a<b)
5. then c=b*b;
6. else c=0;
7. Output c;
8. End
```

For the modified program, CFG will also modify as given in fig-2.
We have the test-suite TS' and now the execution trace according to this is:-

Test-case	**Execution-trace(t)**
t 1	Main.start, main.1, main.2, main.3, main.7, main.end
t 2	Main.start, main.1, main.2, main.4,main.6, main.7, main.end

These test-cases covering only above two paths but the following path is not covered by them that is- 'main.start, main.1, main.2, main.4, main.5, main.7, main.end'. This path can't be covered by TS'. The test-case t4 in TS was able to cover it but that was removed from TS.

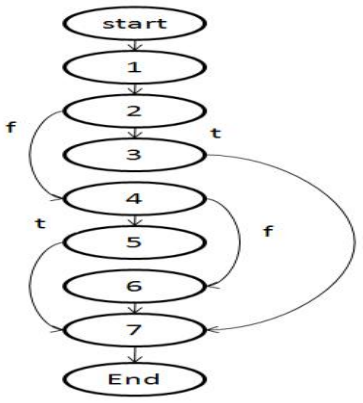

Fig-2: CFG for program in example-2

So this will decrease the fault detection effectiveness of the test-suite. There is the need to check this effectiveness before removal of any test-case. One method is mutation analysis for the measurement of fault detection effectiveness. All these issues are analyzed in the next section.

3. REVIEWED METHODS

There are various methods that used different types of test coverage criteria (e.g., [2-7]) for test redundancy detection.

In [3], the authors mentioned all-definition coverage criterion on simple program. In this work technique of selecting a representative set of test cases from a test suite proposed. The representative set provides the same coverage as the entire test suite. New test cases are designed wherever they are required to test the changed program. In practice, test cases are not removed due to future concern. The authors stated the problem of selecting a representative set. For a given program, a test selection criterion converts into a set of test case requirements. There is a heuristic approach used by authors for finding representative set and this approach is able to reduce about 40% of the size of test suite. The runtime of the proposed algorithm "Reduce Test Suite" is O(n(n+nt)Maximum_cardinality), n is the size of test suite given. A generalization of their algorithm is known as "greedy on bottlenecks" and is described in Zuev [13].

The control flow coverage criteria used in [2,5,7] are- branch coverage in [2], statement in[5] and MC/DC in[7]. In [5], authors proposed three basic orderings for executing test cases, which lead to a total of 12 heuristic reduction procedures. These procedures run the tests in different orders, with the goal of increasing the difference between the original and subsequent orderings as much as possible. The three basic orderings therefore go from the beginning to the end, the end to the beginning and the middle to both ends. The authors were able to reduce 30 % for mutation based test sets and 50% for statement coverage based test sets. They were used Mothra mutation system for creating mutants and Godzilla as test case generator in the empirical evaluation and also calculated the cost for test reduction. The Systems Under Tests (SUTs) used were small scale. In [2], authors examined costs and benefits of test suite minimization based on the study of analysis Wong, Horgan, London, and Mathur [14] and named it WHML study. This study leaves many open questions and the following questions motivated in their work.

1. How does minimization fare in terms of costs and benefits when test suites have a wider range of sizes than the test suites utilized in the WHLM study?

2. How does minimization fare in terms of costs and benefits when test suites are coverage-adequate?

3. How does minimization fare in terms of costs and benefits when test suites contain additional coverage redundant test cases, including multiple test cases that execute equivalent execution traces?

In this work, authors were used following formula to measure the percentage of reduction—

$\{(|T|- |T_{min}|) / |T|\}*100$; it is assumed that all test cases have same cost.

In this work, "the researchers at Siemens Corporate Research sought to study the fault-detecting effectiveness of coverage criteria. Therefore, they created faulty versions of the seven base programs by manually seeding those programs with faults, usually by modifying a single line of code in the program. In a few cases they modified between two and five lines of code".

In [7], a case study was done to analyse "using coverage based criteria for reducing the size of test case is reasonable or not". The following steps summarize the experimental methodology used in this work.

1. Program selection
2. Test set generation
3. Test set minimization
4. Determination of the fault set and preparation of faulty version
5. Size and effectiveness reduction
6. Null hypothesis testing

In this work, faults, injected manually, were obtained from the error-log maintained during its testing and integration phase. Out of 18 injected faults, eight were in the "logic omitted or incorrect", seven belong to the type of "computational problems" and the remaining three faults had "data handling problem".

The reference [4], "presents an algorithm for test-suite reduction and prioritization that can be tailored effectively for use with MC/DC". There are three main contribution of this paper-1. Analysing the problems that occur at the time when existing test-suite reduction and prioritization algorithms are used with MC/DC, so the new and effective techniques are necessary; 2. A description of general approach for new algorithm design; 3. A set of empirical studies performed on two real C projects. Modified condition and decision coverage is stricter than decision coverage and requires execution coverage at condition level. A condition is answered in true or false, that is, Boolean valued expression. This paper presents two new algorithms- one for test suite reduction and other is for test suite prioritization. The size of reduced suite for break down technique was lower than the size of build up technique. Break down approach was used in test suite reduction algorithm and build up approach was used in prioritization algorithm. Authors were implemented these algorithm on a prototype tool and found good reduction in test suite.

A different approach in [8] was proposed. According to this approach, to avoid redundancy test-cases can be transformed instead of discarding. Authors were used a model checker based test case generation approaches. Model checkers use Kripke structures as model formalism i.e., a Kripke structure K is a tuple K =(S, s0, T, L), where S is the set of states, $s0 \in S$ is the initial state, $T \subseteq S \times S$ is the transition relation, and $L: S \rightarrow 2^{AP}$ is the labeling function that maps each state to a set of atomic propositions that hold in this state. AP is the countable set of atomic propositions. The approach used here is significant (slightly) for test reduction but not as large as with approaches that heuristically discard test-cases. Although there is one draw back with run-time complexity but test minimization is improved.

In [9], some experiments were performed to evaluate coverage-based test redundancy detection. This paper presents a different approach i.e., human computer interaction for inspecting the process of test suite reduction, and named as "tester assisted methodology". The approach of this paper shows that "coverage information can't be the only source of knowledge to precisely detect test redundancy". There are different granularity levels (3 levels) of JUnit* tool, as shown in the following figure:-

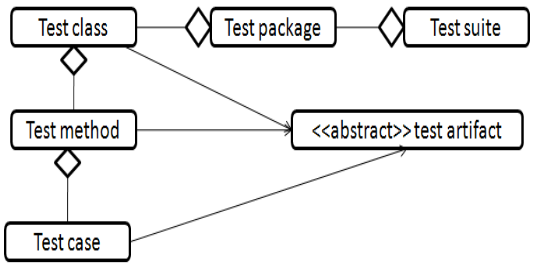

Fig-3: Test Granularity in JUnit (from [9])

Three levels of package, class and methods were considered in this granularity. Two redundancy metrics were proposed- Pair Redundancy (PR) and Suite Redundancy (SR). PR is the ratio of covered items in SUT by the first test artifact with respect to second one while SR is the ratio for first test artifact with respect to all other tests. Following equations (1) and (2) were given for PR and SR. Equation (1) shows PR (**t**$_j$) with (**t**$_k$):- PR(**t**$_j$,**t**$_k$)

$$= \left(\sum_{i \in CoverageCriteria} \left| CoveredItems_i(t_j) \cap CoveredItems_i(t_k) \right| \right) / \left(\sum_{i \in CoverageCriteria} \left| CoveredItems_i(t_j) \right| \right),$$

--------(1)

Equation (2) shows SR:-

SR(**t**$_j$)

$$= \left(\sum_{i \in CoverageCriteria} \left| CoveredItems_i(t_j) \cap CoveredItems_i(TS - t_j) \right| \right) / \left(\sum_{i \in CoverageCriteria} \left| CoveredItems_i(t_j) \right| \right).$$

-----------(2)

The result of the case study in [9], shows that using a collaborative process that involves human testers is more effective in test redundancy detection as well as fault detection effectiveness is unchanged and same as original test suite. But cost of human effort was not discussed.

4. CONCLUSION

In this paper, we have introduced a review work for test redundancy detection techniques. A lot of works have been completed on this issue that is redundancy. After doing the detailed review of all discussed work we conclude that coverage information plays an important role to find the redundancy in test cases. Different types of coverage criteria have been used; combination of more than one criterion gives better result. In all these works, there is a problem of false positive error that is a test case is declared redundant but actually it is not. There is another problem of fault detection effectiveness of test suite after minimization. The approach of human-interaction is effective for these problems, but the cost of human testers and their efficiency is still an issue. Some works show that redundant test cases should be transformed instead of discarding, that will effective for fault detection capability. So all these works motivate us to do some new work in this area.

5. REFERENCES

[1] R. Reichart and T. Girba, "Rule-base Assessment of Test Quality", vol. 6, no. 9, 2007.

[2] G.Rothermel, M.J.Harrold, J.Ostrin, andC. Hong," An empirical study of the effects of minimization on the fault detection capabilities of test suites," in Proceedings of the Conference on Software Maintenance (ICSM '98), pp. 34–43, Bethesda, Md, USA, November 1998.

[3] M. J. Harrold, R. Gupta, and M. L. Soffa, "A Methodology for controlling the size of a test suite," ACM Transactions on Software Engineering and Methodology, vol. 2, no. 3, pp. 270–285, 1993.

[4] J.A. Jones and M.J. Harrold, "Test-suite reduction and prioritization for modified condition/decision coverage," IEEE Transactions on Software Engineering, vol. 29, no. 3, pp. 195–209, 2003.

[5] A.J.Offutt, J. Pan, and J. M. Voas, "Procedures for reducing the size of coverage-based test sets," in Proceedings of the 11th International Conference on Testing Computer Software (ICTCS '95), pp. 111–123,Washington, DC, USA, June 1995.

[6] W. E.Wong, J. R.Morgan, S. London, and A. P.Mathur, "Effect of test set minimization on fault detection effectiveness", Software- Practice & Experience, vol. 28, no. 4, pp. 347–369, 1998.

[7] W. E. Wong, J. R. Horgan, A. P. Mathur, and Pasquini, "Test set size minimization and fault detection effectiveness: a case study in a space application," in Proceedings of the IEEE Computer Society's International Computer Software and Applications Conference (COMPSAC '97), pp. 522–528, Washington, DC, USA, August 1997.

[8] Gordon Fraser and Franz Wotawa, "Redundancy Based Test-Suite Reduction", M.B. Dwyer and A. Lopes (Eds.): FASE 2007, LNCS 4422, pp. 291-305, 2007.

[9] Negar Koochakzadeh and Vahid Garousi, "A Tester Assisted Methodology for Test Redundancy Detection", Software Quality Engineering Research Group, University of Calgary, Canada, 2009.

[10] N. Koochakzadeh, V. Garousi, and F. Maurer, "Test redundancy measurement based on coverage information: evaluations and lessons learned," in Proceedings of the 2nd International Conference on Software Testing, Verification, and Validation (ICST '09), pp. 220–229, Denver, Colo, USA, April 2009.

[11] M. R. Garey and D. S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness,W.H.Freeman,San Francisco, Calif, USA, 1990.

[12] A. Ngo-The and G. Ruhe, "A systematic approach for solving the wicked problem of software release planning," *Soft Computing*, vol. 12, no. 1, pp. 95–108, 2008.

[13] R. Milner, "Turing, computing, and communication," in *Interactive Computation: The New Paradigm*, pp. 1–8, Springer, Berlin, Germany, 2006.

[14] W. E.Wong, J. R. Horgan, S. London, and A. P.Mathur. Effect of test set minimization on fault detection effectiveness. In *17th Int'l. Conf. on Softw. Eng.*, pages 41–50, Apr. 1995.

[15] Y. A. Zuev, "A set-covering problem: The combinatorial-local approach and the branch and bound method," U. S. S. R Computationa1 Math ematics and Mathematical Physics, 19(6):217-226, June 1979.