MUTATION TESTING: A KEY TO PROBLEMS OF ADEQUACY CRITERIA

*Priyanka, Assistant Professor, Government College Chhachhrauli, Chhachhrauli, Haryana

Abstract

Testing plays a key role in establishing the reliability of software. Quality of testing depends upon the quality of test cases. Removal of 100% errors form any software is a tedious and human intensive job. Test adequacy criteria helps to easeuncomfort of exhaustive testing. Many adequacy criteria exist. Different criteria help to focus on different types of errors. Even after multiple rounds of testing efforts, some residual errors may persist. Mutation testing is a criterion that can help in filling the gaps in test cases provided by other testing strategies. This paper discusses about the usefulness of mutation testing in testing field and discusses on some specific issues related to mutation testing criteria.

Introduction

Software development is the most active field in this modern world. Existence of computers in all fields of life has motivated the development of different variety of software satisfying the active needs of the users. People are using these creations with great enthusiasm and belief. To guarantee the working of any software is the key responsibility of software development community. Software collapses may lead to disastrous situations.

Software development is a phased approach and takes place in many phases that enhances the probability of delivery of good quality software in affordable time and cost. Starting from requirements analysis, followed by designing, coding and then finally testing all plays vital roles. Testing phase is the place where whole emphasis is on verifying the quality of software. Key role is played by test cases. Test case generation is a tedious job. It needs a lot of human intervention to find out good quality test cases. Adequacy criteria provide a good way to manage the difficult task.

Adequacy Criteria

Test cases are designed following some test adequacy criteria. Test adequacy criteria is one that defines what constitutes an adequate test [GOOD 75,ZHU 97]. It relates to the properties of program that must be exercised to constitute a thorough test whose successful execution implies error free program. Adequacy criteria can be classified in many ways [ZHU 97].

Specification based, Program based, Combined specification and program based criteria are found in literature [ZHU 97]. Depending upon the underlying testing approach Structural testing, fault-based testing, Error-based testing are widely used criteria.Programme based structural test adequacy criteria are control flow and data flow. Control flow method constitute of statement coverage, branch coverage, condition coverage, loop coverage and path coverage. Whereas for data flow constitute of c-use coverage, p-use coverage, all-use coverage criteria [MATH 08].

Statement coverage adequacy criterion focuses on coverage of all the statements in the program. Statement coverage does not ensure that all branches in statements are covered. To ensure coverage of all branches branch criteria helps. Further condition coverage ensures the coverage of all conditions in branches. Loop coverage ensures the coverage of all loops. Above all is the path coverage which ensures execution of all executable paths from start to end at least once by the test set [ZHU 97, MATH 08, JALO 91].

Status of Testing Techniques

Each criterion has its own pros and cons. Statement coverage helps in coverage of statements but does not ensure coverage of all branches in statements. Branch coverage covers all branches but miss to check all combinations of control transfers. High coverage also does not ensure the thoroughness of test suite. Hundred percentage coverage based testing criteria is not able to find the errors of omission.

Functional testing faces the problems of redundant test cases and gaps of untested functionality. Redundancy can be judged by the recognizing test cases with same purpose but detecting gaps is difficult. Even the sophisticated methods of FT cannot guarantee to recognize the gaps.

Faults are scattered throughout the space and its difficult to remove all even with the help of adequacy criteria suggested in literature. New defects are difficult to remove with the same test case because of pesticide's paradox [SRIN 06]. Just like pests in crops test cases also develop immunity against same test cases.

Same is the case with black box testing first level bugs don't allow the test cases to access the complete external functionality and no new faults are recognized with same test cases. Only with the updated test cases some new errors might be found. Therefore test space can only be covered completely with updated test cases. There should be some way to get an idea of residual errors.

Bebugging is a software engineering technique that helps to measure the test coverage. A known number of bugs are added to the code and tester tries to detect them. The amount of errors not detected helps us to estimate the residual errors. This concept is motivated from the fishing department. In order to find out the amount of fishes in the pond some tagged fishes are added to the existing one. Suppose 100 tagged fishes are added to the existing population.

ISBN 978-93-82529-14-9

Then 100 fishes are collected out of the pond. If out of those collected 20 are the tagged then it's estimated that total amount may count to approximately 500.

Mutation Testing

Mutation as a fault-based testing technique[DEMI 78, BUDD 80]. Fault-based testing aims at demonstrating the absence of pre specified faults in a program[MORE 90].Process of mutation analysis is very simple and can be applied on code as well as specifications. Any program P whose test case adequacy has to be measured is considered and with the help of simple changes in the code it's plenty number of changed versions are created. These changed versions are called as mutants of the original. And simple changes that took place are said to be done by mutation operators. A single change is done at a time to create a new mutant. This change can be as simple as changing any arithmetic operator in single statement. Test set T whose adequacy has to be analyzed is made to run on Program P and its mutants Mi and results are compared. If different results are obtained it means that the test set has executed that changed part of the code and is having a good coverage capability. In that case the mutant is supposed to be killed. If otherwise same results are produced by P and Mutant Mi that means test set has not been able to cover that part of changed code and is not efficient and sufficient enough. This test set needs to be extended to become complete. This way if test set is able to kill all mutants it's supposed to be complete and effective. Sometimes some mutants are not killed by any way that indicates the mutants to be equivalent to the original program. Equivalent programs are those that perform exactly the same functionality as the program itself and there couldn't be any test case available to kill it. The adequacy of a test data set is measured by a mutation score, which is the ratio of the number of killed mutants to the total number of non-equivalent mutants. A set of test cases is mutation adequate if its mutation score is 100%.

Issues Related To Mutation Testing

Nothing is perfect in this world. Along with benefits some problems may come along. Same is case with usage of mutation testing.Cost of execution of so many mutants causes a hurdle in usage of Mutation testing. Another problem is of comparison of results of so many mutants with the actual results, which needs human intervention. And the most difficult problem is the detection of equivalent mutants.

Conclusions

Mutation testing is a fault based testing helps in verifying the effectiveness and completeness of test cases, which means it helps to measure test case adequacy. To some extent it helps to remove the major problem in test cases that is of being incomplete. It helps to form a comparatively complete and effective test case.

References

1. [BUDD 80] Budd, T.A. (1980): "Mutation Analysis of Program Test Data", Ph.D, Yale University, New Haven, CT
2. [DEMI 78] Demillo, R., Lipton, R. and Sayward, F. (1978): Hints on Test Data Selection: Help for the Practicing Programmer. IEEE Computer Magazine, Volume 11, Issue 4, pp. 34-41
3. [GOOD 75] Goodenough, J. B., Gerhart, S. L. (1975): "IEEE Transactions on Software Engineering", SE-3, Vol. 1, No. 3, pp. 156-173
4. [JALO 91] Jalote, P. (1991): "An Integrated Approach to Software Engineering", first reprint, Narosa Publishing House, ISBN: 81-85198-63-2, pp.6-13, 5, 242, 14, 286-288
5. [MATH 08] Mathur, A. P., (2008): "Foundation of Software Testing", Pearson Education Publication, ISBN: 9`788131707951
6. [MORE 90] Morell L (1990): "A Theory of Fault-Based Testing", IEEE Transactions On Software Engineering, Volume 16, Issue 8, pp. 844–857
7. [SRIN 06] Srinivasan, D., Gopalaswamy, R. (2006): Software Testing: "Principles and Practices", Pearson Education India Print ISBN- 10: 81-7758-121-X
8. [ZHU 97] Zhu, H. , Hall, P.A.V. And May, J.H.R. (1997): "Software Unit Test Coverage And Adequacy", ACM Computing Surveys, Vol 29, No.4, pp. 366-427