International Journal of Scientific Research Engineering & Technology (IJSRET)
Volume 1 Issue 5 pp 130-133 August 2012 www.ijsret.org ISSN 2278 - 0882

A Novel Architecture for Effective and Efficient Unit Testing

Manik Chandra Pandey[1], Dr.Jayant Shekhar[2] , Rameshwaram Tiwari [3] , Shriniwas singh[4] , Anuja Singh[5]
1 (M.TECH. student-IITT College of Engineering, Pojewal, Punjab, India)
[2] (Professor, Subharti University, U.P., India)
[3] (Assistant Professor, Radha Govind Engineering College,Meerut,India)
[4, 5] (Assistant Professor, Subharti University, U.P., India)

ABSTRACT
Testing is the most important method used to validate a software product. Unit testing is an efficient method of detecting and isolating defects in individual units of code.In this thesis we explore some issues relating to the unit testing of object-oriented systems, using the experimental approach for validation Specifically we Proposes a framework for assessing the effectiveness and the efficiency of unit testing and applies it to evaluate different coverage criteria.

Keywords: *Testing, effectiveness, efficiency*

I. INTRODUCTION
Testing is the most important method used to validate a software product. In general, many different levels of testing are applied in a software project. Unit testing is normally the first formal test activity performed in the software life cycle and it occurs during the implementation phase after a program unit is coded. Unit testing is an efficient method of detecting and isolating defects in individual units of code.. In object-oriented software, a class or a small collection of classes is generally chosen to represent a unit for unit testing purposes. Object-oriented systems introduce new issues to unit testing when compared to procedural systems.

The experimental work in software testing has so far, generally focused on evaluating the effectiveness (fault detection capability) of a coverage criterion. The important issue of testing efficiency has not been sufficiently addressed.

The automated tools, such as *J Unit* [1],has raise awareness of the importance of testing. However, there is little information about how effective testing with *J Unit* is in practice, since test-sets are constructed intentionally and are not evaluated for their quality. In other testing approaches, test coverage metrics are frequently looked such as statement, branch or multiple condition coverage [2].

II. EFFECTIVE TESTING TECHNIQUES
Testing is the most important method used to validate a software product. The word program 'testing' may be defined differently in different circumstances, but most commonly it refers to the process of exercising a program with the intent of observing errors[1,2]. A key purpose of testing is to increase the confidence in the code being developed. Software development is a complex process and its various development activities are carried out by a team of developers having different roles and responsibilities; inevitably the process is inherently error prone. In order to maximize defect detection, testing at different levels are applied in a software project. The key levels are [3]:

- **Unit Testing** - refers to the testing of individual program units in isolation.
- **Integration Testing** - refers to the testing for interface faults in the combined program units.
- **System Testing** - refers to the testing of the system as a whole by putting all parts of the system in place. The system may have electrical, mechanical, or other components as per the system requirements.
- **Acceptance Testing** - refers to the testing of the system from the perspective of the user.

The purpose of these different levels of testing is to detect defects that are introduced in the major phases of software development - requirement gathering, system specifications, design, and coding. This relationship is shown as the V-model for software testing [4].

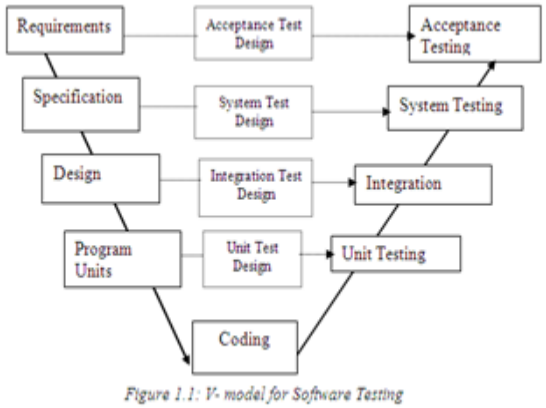

Figure 1

III. UNIT TESTING

Unit testing consists of the dynamic verification of the behavior of a program unit on a finite set of test cases, suitably. selected from the usually infinite executions domain, against the specified expected behavior [9]. Systematic unit testing assumes a model of the program unit under test, and tests are generated to cover certain aspects of that model. A model can be the program unit itself, an abstraction of the unit (e.g. a control flow graph), or a specification of the unit - be it a state model or more formal specifications of the program. During testing, the program is executed with a set of test cases and its behavior is observed to identify the presence of defects in the program. In this section we discuss the different components of unit testing. But before that we define the unit as well as some other terms related to unit testing. [14]

A. A Unit

In procedural systems, a unit is represented by an autonomous function (also called procedures or sub-routines). A function is a program unit with a defined interface. If invoked by another function, the function takes the input in the form of a parameter (and/or global variable), performs some computation, and returns some values. The integration of a set of related functions is typically called a module, and a system is the integration of all such modules. An object-oriented system consists of a set of objects, and computations are performed by sending messages from one object to another. A message invokes one of the objects methods, which may then modify the object state and may send messages to other objects.[13] A class is the fundamental building block of an object-oriented system as it represents the template for creating objects, which encapsulates an object's data and operations that can be applied on that data. Due to the high potential and the frequency of reuse, the class is the focal point of the unit testing in object-oriented systems. It has been pointed out that due to encapsulation; the methods are meaningless apart from their class [10, 11]. Moreover, inheritance, polymorphisms, and the reuse of the classes introduce new problems in testing and have to be tackled accordingly. Therefore, unit testing in object-oriented systems is different from unit testing in procedural systems.

IV. CONSTRAINTS ABOUT TESTING

It is assume that testing is very easy process and goes along with the process in a specific manner but there are some constraints that lead to misconception that the testing is waste of time and not should be given that much preference than other processes[5][15]

*1) **It's Time Consuming Process:*** After writhing the code it is assumed that the process is ready to work and is integrated with the other unit for proper functioning .Unit testing is the real process from where the actual testing begins but is treated as sort of doing the things in proper manner.

*2) **It's tells about the functionality of code:*** The programmers think that jumping directly to code is the better option and easiest way to develop the program. After written the code it is the task to check weather it is written in correct manner or not. Programmer just read the code to find out the compiler error only not emphasis on the scope.

*3) **Programmer misconception about their ability*:** The programmer sometimes think that they are mature and intelligent enough that whatever they write will never contain any kind of error. so doing unit testing is kind of waste of time and human effort.

*4) **Another options are available to catch the error:*** It is often assumed by the programmer that other types of testing levels are available which are efficient enough to catch the error in the code, so doing it in initial stage is totally a waste of time.

V. PROPOSED WORK FOR EVALUATING EFFECTIVENESS & EFFICIENCY OF COVERAGE CRITERIA FOR UNIT TESTING

We propose a general approach for comparison of coverage criteria experimentally using tool supported mutation analysis. This approach enables us to compare coverage criteria based on their effectiveness and efficiency. This is achieved by evaluating the effectiveness and efficiency of a set of coverage adequate test suites for each of the coverage criteria under study over a set of test programs, which are seeded with artificial faults[12]. For a test program, this approach requires availability of a test-pool, which is 'sufficiently large' so as to facilitate construction of multiple coverage-adequate test suites for each coverage criterion under study, randomly. As an application of the proposed approach, we describe an experiment comparing three control- flow-based coverage criteria, namely, block, branch, and predicate coverage [19].

B. An Approach for Evaluating Coverage Criteria

We develop a general framework to facilitate comparison among different coverage criteria. The proposed approach is a two-phase testing process for obtaining coverage information and the fault data. First we define some important terms that are used in this work.

1) Coverage-adequate Test Suite : A test suite satisfying some completeness criterion, e.g. 90% branch coverage.

2) Minimal Test Suite : A test suite which is constructed by selecting tests cases monotonically, in such a fashion that it takes minimum number of test cases required to satisfy a given criterion.

3) Test-Pool : A collection of a large number of test cases which can be used for the generation of multiple coverage-adequate test suites, for the different coverage criteria under study, by selecting a subset of the test cases from the test pool.

4) Mutation Operator : A handle to seed faults in a test-program in some specific context.

5) Mutant : A faulty version of a program containing exactly one known fault. It is obtained by applying a mutation operator at an appropriate place in the program.

Mutation testing is a fault-based testing technique that is based on the assumption that a program is well tested if all simple faults are predicted and removed; complex faults are coupled with simple faults and thus are also detected by the tests that can detect simple faults [11]. A mutant program can be obtained by applying a specific mutation operator to create a known fault in the code.

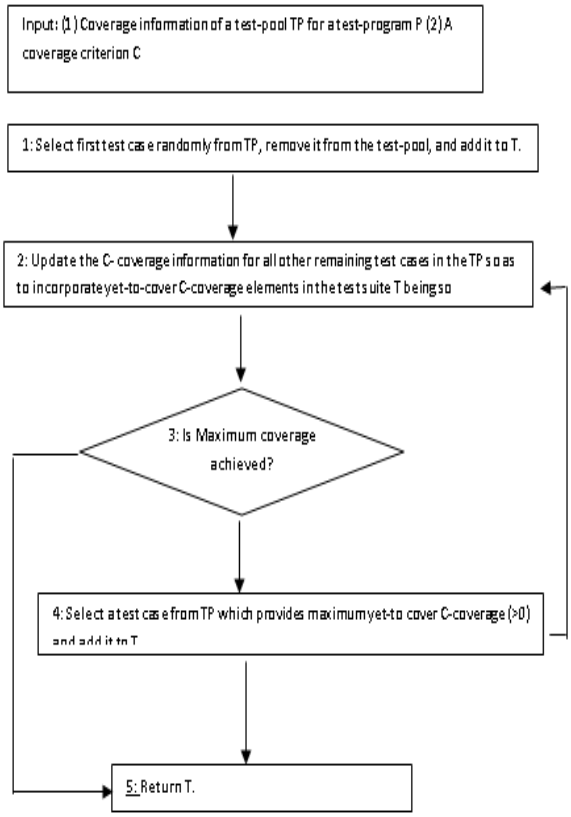

Fig1. Constructing Coverage Adequate Minimal Test Suites

The systematic generation of mutant programs means that different types of fault can be compared. In particular, one can identify whether one coverage criterion is better than the other at revealing faults of a specific type.

VI. CONCLUSION

Improving the effectiveness and the efficiency of unit testing in industry requires the evidence to drive the change [16] This work investigates critical issues relating to unit testing and collates evidence to accept or refute various propositions by performing sophisticated measurements of effectiveness and The first contribution this work makes is to propose a tool- supported controlled experimentation framework that facilitates comparison between different coverage criteria. The framework allows the reliable measurement of testing effect effectiveness and (the relatively under-researched variable) efficiency. We demonstrate that the proposed approach has practical application by describing the results of an experiment comparing three code-based testing criteria, namely, block

coverage, branch coverage, and predicate coverage efficiency. The proposed approach will facilitate the comparison of any coverage criteria for which test coverage information can be obtained. We not only can effectively compare within a family (a particular section of the classification), but across the classification as well. With the help of proper tool support, for instance, we can use the proposed approach for comparing data-flow based coverage criteria or mutation based testing.

REFERENCES

[1]P. Jalote. An integrated approach to software engineering. Springer-Verlag, NewYork, NY,USA,

[2]G. J. Myers. The Art of Software Testing. Wiley Inc., 1979.

[3] L. Peeger and J. M. Atlee. Software Engineering: Theory and Practice. Pearson international , New Jersey, 3rd edition, 2006.

[4]Vegas and V. Basili. A characterisation schema for software testing techniques. Empirical Softw. Engg., 10(4):437–466, 2005.

[5] J. A. Whittaker. What is software testing? and why is it so hard? IEEE Softw.,17(1):70–79,2000.

[6] B. Beizer. Software Testing Techniques. Thomson Computer Press, 2nd edition,1990

[7] K. Beck. Simple smalltalk testing: With patterns. Technical report, First ClassSoftware,Inc.,1994

[8] S. P. Fiedler. Object-oriented unit testing. Hewlett-Packard Journal, 40(2):69–75,1989

[9] P. Bourque, R. Dupuis, A. Abran, J. W. Moore, and L. Tripp. The guide to the software engineering body of knowledge. IEEE Softw., 16(6):35–44, 1999.

[10] M. D. Smith and D. J. Robson. Object-oriented programming-the problems of validation. In ICSM'90: Proceedings of the Seventh International conference on Software Maintenance, pages 272–281, 1990.

[11] C. D. Turner and D. J. Robson. The state-based testing of object-oriented programs. In ICSM '93: Proceedings of the International Conference on Software Maintenance, pages 302–310, Washington, DC, USA, 1993. IEEE Computer Society.

[12] IEEE. Standard glossary of software engineering terminology, IEEE Std 610.12-1990 edition, 1990.

[13] JUnit Home Page. http://www. junit.org, [Accessed July 2011].

[14] E. J. Weyuker. The evaluation of program-based software test data adequacy criteria. Commun. ACM, 31(6):668–675, 1988.

[15] T. Korson and J. D. McGregor. Understanding object-oriented: a unifying paradigm.Commun.ACM,33(9):40–60,1990.

[16] L. C. Briand, Y. Labiche, and Y. Wang. An investigation of graph-based class integration test order strategies. IEEE Trans. Softw. Eng., 29(7):594–607,2003.

[17] M. J. Harrold, J. D. McGregor, and K. J. Fitzpatrick. Incremental testing of object-oriented class structures. In ICSE '92: Proceedings of the 14th International Conference on Software Engineering, pages 68–80, NewYork, NY, USA, 1992. ACM Press.

[18] D. Kung, J. Gao, P. Hsia, J. Lin, and Y. Toyoshima. Class firewall, test order, and regression testing of object-oriented programs. Jour of OOP, 8(2):51–65,1995.

[19] W. E. Howden. Reliability of the path analysis testing strategy. IEEE Trans. Softw. Eng., SE-2(3):208–215, 1976