A Systematic Study of Test Adequacy Criteria satisfied by UML Behavioural Models

Gargi Bhattacharjee
Dept of Comp Sc & Engg
BIT Mesra
India

K. S. Patnaik
Dept of Comp Sc & Engg
BIT Mesra
India

Abstract

Software testing ensures the quality of software and in turn increases its reliability and robustness. The quality of the end product developed depends largely on "how effectively it has been tested". The quality of test is one of the vital objectives of software testing. Many test criteria have been proposed and studied for this purpose. A survey of the adequacy criteria satisfied by the UML models has been reported.

Keywords: *- Test Adequacy Criteria, Coverage Criteria, UML Behavioural Models, Software Testing.*

1. Introduction

Software engineering is a discipline concerned with all aspects of software right from its inception to its acceptance. Software testing plays a vital role in quality control of the software. Testing aims at detecting errors in the software and is carried out by executing the program on a set of tests and then comparing the actual outputs with the expected outputs. Software testing accounts for nearly 50% of the total development cost of the software. Therefore, there is a need for effective testing strategies. Exhaustive testing is not possible because there are no limits on how much we can test. Thus, to limit the process of testing, the concept of testing criteria was introduced. Satisfying the testing criteria marks an end to testing process.

Numerous techniques have been proposed for software testing in the literature [1]. Due to the increased use of the object oriented (OO) paradigm, several new testing strategies have been specifically proposed for OO software [2]. Furthermore, with the increasing use of the Unified Modelling Language (UML) to model OO systems, researchers have begun investigating how UML can be used in the testing phase. Consequently, several UML-based approaches to software testing have been proposed [3, 4, and 5]. In these approaches, test requirements and coverage criteria are derived from UML models. A software test adequacy criterion is a predicate whose successful execution assures no error in the tested program. For example: generate all inputs or seed with faults or cause certain parts of the system to be exercised. A testing technique guides the tester through the testing process by including a testing criterion and a process for creating test case values. Testers measure the extent to which a criterion is satisfied in terms of coverage. Test criteria help in defining test objectives or goals that are to be achieved while performing software testing. Cost considerations and available resources often determine the selection of one criterion over another. Testing can be stopped when tests that satisfy all the criteria have been carried out successfully.

In this paper, we present a survey of various test adequacy criteria satisfied by UML behavioural models. Prior to that, we present an insight into the various aspects of test adequacy criteria. The rest of the paper is organized in the following way: section 2 gives an insight to adequacy criteria, section 3 discusses about UML models, section 4 shows the various test adequacy criteria followed by various UML behavioural models. Finally section 5 discusses the conclusion.

2. Test Adequacy Criteria

One important aspect of software testing is to decide when enough testing has been done. Then, the question arises as how to decide if a test set is adequate? This question was first addressed by Goodenough and Gerhart [6]. According to them, a test adequacy criterion is a predicate that defines "what properties of a program must be exercised to constitute a thorough test. It can be viewed as a rule or a set of rules that impose requirements on a test set. It is a set of test obligations that have to be fulfilled to ensure complete testing.

A test suite satisfies an adequacy criterion if:

- All the tests succeed (pass).
- Every test obligation in the criterion is satisfied by at least one of the test cases in the test suite.

2.1. Classification of Adequacy Criteria

There are various ways to classify test adequacy criteria [7].One of the most common is by the source of information used to specify testing requirements and in the measurement of test adequacy. Hence, an adequacy criterion can be classified as:

- Specification-based: The test cases should be developed in such a manner that they should cover all the features that have been identified from the requirements specification.
- Program-based: The test cases are designed in terms of the program under test. A test set is said to be adequate if the program under test has been thoroughly exercised.
- Design based: The test criterion determines the configurations that must be covered in an adequate design-level test.
- Combined specification and program based criteria: This criterion uses the ideas of both program-based and specification based criteria.

Test adequacy criteria can be also classified by the underlying testing approach. There are three basic approaches to software testing:

- Structural testing: It specifies testing requirements in terms of the coverage of a particular set of elements in the structure of the program or the specification.
- Fault-based testing: It focuses on detecting faults (i.e., defects) in the software. An adequacy criterion of this approach is some measurement of the fault detecting ability of test sets.
- Error-based testing: It requires test cases to check the program on certain error-prone points according to our knowledge about how programs typically depart from their specifications.

2.2. Axiomatic properties of Test Data Adequacy

Axiomatic theories [8] have traditionally been used in two complimentary ways. On one hand, they serve to make underlying assumptions explicit. On the other hand, they tend to derive properties common to a collection of different structures. The axiomatic property proposed is with respect to the first use.

Axiom 1: Applicability – For every program, there exists a finite adequate test set.

Axiom 2: Nonexhaustive Applicability – There is a program P and test set T, such that P is adequately tested by T and T is not an exhaustive test set. A program is exhaustively tested if it has been tested on all representable points of the specification's domain. Such a test set which performs exhaustive testing is called an exhaustive test set.

Axiom 3: Monotonicity – If a test set T, is adequate for a program P, and $T \subseteq T'$, then T' is adequate for P.

Axiom 4: Inadequate Empty Set – The empty set is not adequate for any program.

Axiom 5: Antiextensionality- There are programs P and Q, such that P Q, T is adequate for P, but T is not adequate for Q.

Axiom 6: General Multiple Change – There are programs P and Q which are of the same shape, and a test set T such that T is adequate for P, but T is not adequate for Q.

Axiom 7: Antidecomposition – There exists a program P and a component Q, such that a test set T is adequate for P, T' is the set of vectors of values that variables can assume on entrance to Q for some t of T, and T's is not adequate for Q.

Axiom 8: Anticomposition – There exists programs P and Q such that a test set T is adequate for P and P(T) is adequate for Q but T is not adequate for P; Q.

2.3. Properties of Test Adequacy Criteria

Zhu et al [7] presented the following properties in their work.

- Stopping rule- An adequacy criterion is considered to be a stopping rule that determines whether sufficient testing has been done that it can be stopped. Testing stops if 100% of the statements have been tested.
- Specifies requirements- An adequacy criterion specifies a particular software testing requirement, and hence determines the test cases to satisfy the requirement.
- Provides measurement- Test data adequacy criteria provide measurements of test quality when a degree of adequacy is associated with each test set so that it is not simply classified as good or bad. In practice, the percentage of code coverage is often used as an adequacy measurement. Thus, an adequacy criterion C can be formally defined to be a function C from a program p, a specification s, and a test set t to a real number r = C(p, s, t), the degree of adequacy.
- Test Case Generator- Test case selection criteria are generators, i.e., functions that produce a class of test sets from the program under test and the specification. Any test set in this class is adequate, so that we can use any of them equally. Here we look for a test which exercises some statements which have not covered by the tests so far.

2.4. Use of Test Adequacy Criteria

- Test adequacy criteria guide us in selecting the proper testing technique and in turn, we are able to choose the correct test suite.
- Test adequacy criteria reveal the test cases that might have been missed in the test suite.
- Adequacy criteria provide a way to define a notion of "thoroughness" in a test suite.
- Adequacy criteria can be used for either selecting test cases or for providing a measurement for test quality.

2.5. Test Coverage

Goodness of a test suite is determined by the coverage of the product by the test set so far i.e. percentage of statements or requirements tested. There have been 3 types of coverage proposed in the literature so far.

- Statement coverage criterion requires that every statement in the program is executed at least once. A test set that satisfies this requirement is considered to be adequate according to the statement coverage criterion. The percentage of the statements exercised by testing is a measurement of the adequacy.
- Branch coverage criterion requires that all control transfers in the program under test are exercised during testing. The percentage of the control transfers executed during testing is a measurement of test adequacy.

- Path coverage criterion requires that all the execution paths from the program's entry to its exit are executed at least once during testing.

2.6. Comparing Test Criteria

To distinguish stronger criteria from weaker criteria, we have the subsumes relation. It states;

"Test adequacy criterion A subsumes test adequacy criterion B if, for every program P, every test suite satisfying A with respect to P also satisfies B with respect to P".

Eg: Exercising all program branches (branch coverage) subsumes exercising all program statements.

3. The Unified Modelling Language

The Unified Modelling Language [9,10] (UML) is an Object Management Group (OMG) Object-Oriented (OO) modelling language standard that is gaining widespread usage in the software development industry. It is used for specifying, visualizing, constructing, and documenting the artifacts of software systems. It is also used for business modelling and other non-software systems. The language is primarily intended to be used with object-oriented software. The UML represents a collection of engineering practices that have been used to model large and complex systems. Modelling a large, complex system can result in a system model that consists of a variety of diagrams representing different views of the model.

The UML defines nine graphical diagrams to specify and design software. These diagrams are grouped under two heads: Structure Diagrams & Behavioural Diagrams.

Structure Diagrams

Structure diagrams lay emphasize on the things that must be present in the system to be modelled. Since structure diagrams represent a static view of the system, they are used extensively in documenting the software architecture of these systems.

- Class diagram describes the structure of a system by showing the system's classes, their attributes, and the relationships among the classes.
- Component diagram describes how a software system is split up into components and shows the dependencies among these components.
- Composite structure diagram describes the internal structure of a class and the collaborations that this structure makes possible.
- Deployment diagram describes the hardware used in system implementations and the execution environments and artifacts deployed on the hardware.
- Object diagram shows a complete or partial view of the structure of an example modelled system at a specific time.
- Package diagram describes how a system is split up into logical groupings by showing the dependencies among these groupings.

Behavioural diagrams

- Behavioural diagrams emphasize what must happen in the system to be modelled. Since behaviour diagrams illustrate the behaviour of a system, they are used extensively to describe the functionality of software systems.
- Activity diagram describes the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.
- State machine diagram describes the states and state transitions of the system.
- Use Case Diagram describes the functionality provided by a system in terms of actors, their goals represented as use cases, and any dependencies among those use cases.

Interaction diagrams, a subset of behaviour diagrams, emphasize the flow of control and data among the things in the system being modelled.

- Communication diagram shows the interactions between objects or parts in terms of sequenced messages. They represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behaviour of a system.
- Interaction overview diagram provides an overview in which the nodes represent communication diagrams.
- Sequence diagram shows how objects communicate with each other in terms of a sequence of messages. Also indicates the lifespans of objects relative to those messages.
- Timing diagrams a specific type of interaction diagram where the focus is on timing constraints.

Since Behavioural models represent the dynamic nature of the system, more importance is laid on them.

This paper lays focus on all the possible test adequacy criteria that can be satisfied by these diagrams.

4. Test Coverage Criteria satisfied by UML Behavioural Models

4.1. Test criteria based on Activity Diagram

Let ‘p’ be a program,‘t’ be a test case and ‘ts’ be a test suite.

M.Chen et al [11] in their work considered activity coverage and transition coverage as their test criteria.

- Activity Coverage requires that all the activity states in the activity diagram be covered. For any t ∈ ts, the corresponding program execution trace ‘pet’ is found out. If there exists any function in pet whose activity is not marked in the activity diagram, the corresponding unmarked activities of pet are marked and then test case ‘t’ is recorded. The value of activity coverage is the ratio of the marked activities to all activities in the activity diagram.
- Transition Coverage requires that all transitions in the activity diagram be covered. For any t ∈ ts, the corresponding program execution trace ‘pet ’is found out. If there exists any function in pet whose transition is not marked in the activity diagram, the corresponding unmarked transitions of pet are marked and then test case t is recorded. The value of transition coverage is the ratio between the checked transitions to all transitions in the activity diagram.

Debasish Kundu and Debasis Samanta [12] in their paper considered the following two approaches.

- Basic path coverage: A basic path is a sequence of activities where an activity in that path occurs exactly once. A basic path considers a loop to be executed at most once. Basic path coverage requires that for a set of basic paths PB obtained from an activity graph and a set of test cases ts, for each basic path pi ∈ PB, there must be at least one test case t ∈ ts such that when the system is executed with the test case t, pi is exercised.
- Simple Path Coverage: A simple path for activity diagrams is one that contains concurrent activities. Given a set of simple paths PS for an activity graph which contains concurrent activities and a set of test cases ts, for each simple path pi ∈ PS, there must be a test case t ∈ ts such that when the system is executed with a test case t, pi is exercised in such a way that all simple paths in the activity diagram are covered. The value of simple path coverage is the ratio of the traversed simple paths to all simple paths in the activity diagram.

4.2. Test criteria based on Sequence Diagram

Vikas Panti et al [13] executed object coverage and boundary value testing criterion as their adequacy criteria while Santosh Swain et al [14] thought of using all message path coverage, full predicate coverage, concurrent coverage and branch coverage criteria to test their work. Ashalatha Nayak tested her work using loop adequacy criterion [15].

- Object Coverage covers every object in the sequence diagram for basic test case generation. Object coverage is a test adequacy criterion that requires tests to check program's output variables. All variables still defined when executing in test scope (even those which are not visible, such as private fields of objects) are considered by object coverage.

- Boundary-testing criterion: The boundary-testing criterion is satisfied for inequality borders. If each selected inequality border B is tested by two points (ON-OFF) of test input domain such that, if for one of the point the outcome of a selected predicate 'r' is true, then for the other point the outcome of 'r' is false Also the points should satisfy the initial path associated with B and the considered points should be as close as possible to each other. It should be tested carefully because domain boundaries are particularly fault prone. Boundary-testing criterion is a criterion for ensuring that a boundary is tested adequately. Instead of generating several test data values that achieve transition path coverage, the border is simply tested as determined by a simple predicate. It helps to reduce the number of test cases significantly; at the same time, the generated test cases achieve very high test coverage.

- All Message Path coverage: A set of concurrent message paths P satisfies the all-message-paths coverage criterion if and only if P contains all start-to-end message paths in a sequence diagram. A start-to-end message path in a sequence diagram is a sequence of messages that begins with an externally generated event and ends with the production of a response that satisfies this event.

- Full-Predicate-Coverage: A test set T satisfies the full predicate coverage criterion if and only if for each clause 'c' in each condition in a sequence diagram there exist t_1 in T such that t_1 causes c to evaluate to TRUE and there exists t_2 in T such that t_2 causes c to evaluate to FALSE while all other clauses in the condition have values such that the value of the condition will always be the same as the clause under test. This criterion ensures that all the predicates are checked i.e. all possible combinations of the different predicates in the condition are checked.

- Concurrent coverage criterion: For each concurrent node in sequence diagram, T must include one scenario corresponding to every valid interleaving of message sequences.

- Branch Coverage Criterion: Given a test set T and sequence diagram, if a message is sent under some condition c, the set of test cases should ensure that at least one path which covers the condition with c is FALSE. This criterion was initially proposed by Binder [16]. The primary difference between this and concurrent coverage is that in branch coverage if a condition appears more than once and if it is covered at least once, the criterion is satisfied.

- Loop adequacy criterion: For each loop fragment,

  - T must include at least one scenario in which the control reaches the loop and then the body of the loop is not executed ("zero iteration" path).

  - T must include at least one scenario in which control reaches the loop and then the body of the

loop is executed at least once before control leaves the loop ("more than zero iteration" path).

4.3 Test criteria based on State Chart Diagram

Ranjita Swain et al [17] tested her work with the following coverage criteria.

- State Coverage requires that all the state nodes in a state chart diagram be covered. The value of state coverage is the ratio between the covered states and all the states in the state chart diagram.

- Transition Path Coverage: A test suite 'ts' is said to achieve transition path coverage if for a given state chart graph 'G', 'ts' causes each possible transition path in 'G' to be taken at least once. It should cover all arbitrarily long distinct paths through transitions for exhaustive test generation. As there is a defined set of transitions in the state model, a coverage measure associated with this strategy is to measure the proportion of transitions exercised by a set of test cases. The value of transition coverage is the ratio between the transitions exercised and the total number transitions in the state model.

- Condition coverage: A decision consists of conditions separated by logical operators (e.g. and, or). A single condition is covered, if it evaluates to both true and false at some point during test execution. Decision coverage is also been called branch coverage or predicate coverage.

- Transition-pair coverage: This requires covering each pair of adjacent transitions at least once in some test case. Therefore, the transition-pair coverage subsumes the transitions path coverage. The transition-pair coverage criterion generates more test cases than the transition path coverage criterion.

5. Conclusion

In this article, various types of software test adequacy criteria that have been proposed till date are reviewed. Test criteria have become a vital part of research for software testing. While covering these criteria, test cases bring out the faults associated with the software under test. It has been noted that relatively little work has been done on how effective the criteria are at detecting faults. Therefore, they can be considered an important topic for future research.

REFERENCES

[1]. B. Beizer, "Software Testing Techniques", *John Wiley & Sons, Inc.*, New York, NY, USA, 1990.
[2]. R.V.Binder, "Testing object-oriented software: a survey", *Software Testing Verification and Reliability*, 6(3/4): 125-252, 1996.
[3]. L. Briand and Y. Labiche, "A UML-based approach to system testing", *Journal of Software and Systems Modelling*, 1:10–42, 2002.
[4]. J. Hartmann, C. Imoberdorf, and M. Meisinger, "UML based integration testing", *SIGSOFT Software Engineering Notes*, 25(5):60–70, September 2000.
[5]. Bertolino, F. Basanieri, "A practical approach to UML based derivation of integration tests", in: *Proceedings of the Fourth International Software Quality Week Europe and International Internet Quality Week Europe (QWE)*, Brussels, Belgium, 2000.
[6]. J.B. Goodenough and S.L. Gerhart, "Toward a theory of test data selection", *In Proceedings of the International Conference on Reliable Software*, pages 493–510, Los Angeles, California, 1975.
[7]. H. Zhu, P. Hall, and J. May, "Software Unit Test Coverage and Adequacy", *ACM Computing Surveys*, 29(4):366–427, December 1997.
[8]. E.J.Weyuker, "Axiomatizing software test data adequacy", IEEE *Trans. Software Eng*, vol. SE-12, pp.1128 -1138 1986.
[9]. J. Rumbaugh, I. Jacobson, and G. Booch, "The Unified Modeling Language Reference Manual", *Addison-Wesley*, 2001.
[10] J. Rumbaugh, I. Jacobson, and G. Booch, "The Unified Modeling Language User Guide", *Addison-Wesley*, 2001.
[11]. M.Chen, X.Qiu, and X.Li, "Automatic test case generation for UML activity diagrams", Proc. *Int. Workshop on Automated Software Testing (AST06)*, Shanghai, China, 23 May, pp. 2–8. ACM Press, NJ.
[12]. Debasish Kundu, Debasis Samanta, "A Novel Approach to Generate Test Cases from UML Activity Diagrams", *Journal of Object Technology*, Vol. 8, No. 3, pp.65 -83, May-June 2009.
[13]. Vikas Panthi, D.P.Mohapatra, "Automatic Test Case Generation using Sequence Diagrams",

International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868 Foundation of Computer Science FCS, New York, USA Volume 2– No.4, May 2012.

[14]. Santosh Kumar Swain, Durga Prasad Mohapatra, and Rajib Mall, "Test Case Generation Based on Use case and Sequence Diagram", *Int.J. of Software Engineering, IJSE* Vol.3 No.2 July 2010.

[15]. Ashalatha Nayak and Debasis Samanta, "Automatic Test Data Synthesis using UML Sequence Diagrams", *Journal of Object Technology*, Vol. 09, No. 2, March-April 2010.

[16]. R.V Binder, "Testing Object-Oriented Systems Models, Patterns, and Tools", Object Technology Series. *Addision Wesley*, Reading, Massachusetts, October 1999.

[17]. Ranjita Swain, V Panthi, Prafulla Kumar Behera and Durga Prasad Mohapatra, "Automatic Test case Generation From UML State Chart Diagram", *International Journal of Computer Applications* (0975 – 8887) Volume 42– No.7, March 2012.