ISSN: 2277-9655
Impact Factor: 1.852

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

A Study on Automatic Test Case Generation

Sucheta Bhat[*1], Dr. Prashanth C M[2]

[*1] Masters in Technology at Sapthagiri College of Engineering, Bangalore, India

[2] Head of Department of Computer Science and Engineering with Sapthagiri College of Engineering, Bangalore, India

sucheta.india@gmail.com

Abstract

Software testing is a process of evaluating a software item to detect the difference between given input and expected output. Software testing is one of the important step in software development lifecycle. Testing is a challenging task as it requires that user requirements be completely and properly understood before testing and also be able to test and deliver the product in less time. The purpose of this paper is to identify different ways in which testing time can be reduced thereby increasing accuracy. The paper focuses on automatically generating test cases which when generated manually requires more time and effort. The paper broadcasts various methods followed to generate test cases automatically and discusses the pros and cons of the methods used. Further the currently followed approach, the pros and cons of the approach is shown.

Keywords: Testing, Test Case Generation.

Introduction

Software engineering is a discipline whose aim is the production of fault free software that satisfies the user's needs and that is delivered on time and within budget. The Software development life cycle (SDLC), sometimes referred to as the Application development life-cycle, is used in systems engineering, information systems , software engineering, and represents a process for creating or altering information systems, the models and methodologies that people use to develop these systems. The Phases of Software Development Lifecycle include:

- Requirements gathering and analysis
- Design
- Implementation or coding
- Testing
- Deployment and Maintenance

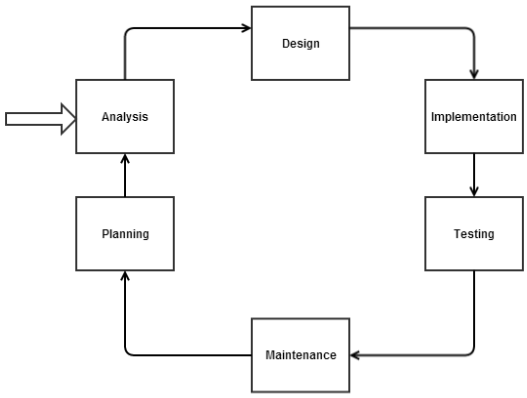

Figure 1 Software Development Lifecycle

Software testing is a process of verifying and validating a software application or program that meets the business and technical requirements that guides its design, development, works as expected and also identifies important errors or flaws categorized as per the severity level in the application that must be fixed [1]. Software testing is one of the most important aspects in the software development lifecycle. Testing software allows validation of business requirements conformance, functional correctness of individual components, quality assurance and robustness of the system.

ISSN: 2277-9655
Impact Factor: 1.852

The main objective of testing is to prove that the software product meets a set of pre-established acceptance criteria. There are two components to this objective. The first component is to prove that the requirements specification from which the software was designed is correct. The second component is to prove that the design and coding correctly respond to the requirements. Correctness means that function, performance, and timing requirements match acceptance criteria [1].

A test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly [1]. It is the responsibility of the Quality Assurance (QA) team to certify that a software system has been tested functionally. The QA team needs to setup a test strategy, create test case documents for different types of testing, execute the tests and provide a report of the defects that are subsequently identified during testing. Test case documents creation is important as it forms the basis for testing the software system.

The QA or Testing team creates test cases for each use case of the software to be tested. For effective testing, the test case documents must cover and trace all paths that are existent in a software system. Manually creating the test cases requires lot of effort by the testing team and increases the production cost for software product being developed. To reduce the time, effort and consequentially the cost associated with creating test cases, it is imperative to develop methodologies to automatically generate test cases.

Software designers and developers extensively use the Unified Modelling Language to design and develop software systems. UML is a general purpose modelling language. It was started to capture the behaviour of complex software and non-software system. UML provides elements and components to support the requirement of complex software [1] and diagrams [2][3][4] to understand a system in better and simple way. A single diagram is not sufficient to cover all aspects of the system. So UML defines various kinds of diagrams to cover most of the aspects of a system that include Structural, behavioural and interactional diagrams.

This paper focuses on discussing various techniques used for generating the test cases manually as well as automatically, pros and cons of the methods followed and ways to improve them.

Literature survey

Boghdady, et. al. [5] explores different approaches for generation of test cases from different models. Model based testing (MBT) refers to the type of testing process that focuses on deriving a test model using different types of formal testing methods, then converting this test model into a concrete set of test cases[6][7][8]. These formal models have many different types, but all of them are generally categorized into three main categories: requirements models, usage models, and source code dependant models. The requirements models can be behavioral, interactional, or structural models according to the perspective by which the requirements are being looked at. The test cases derived from behavioral or interactional models are functional test cases and they have the same level of abstraction as the models creating them. These kinds of test cases differ from those derived using structural models.

Quality of test cases depends on how well they cover the functionalities of the system under test [9], [10] and not only on their form [11]. The test cases should be validated against known quality standards [12], [13], [14] which determine their acceptable form as well as the degree of their functional coverage which in turn specifies their level of applicability. Many metrics have emerged and are being used to measure the quality of the test cases being generated like the time, cost, effort, complexity of generation, coverage criteria and many others [15], [16]. The Unified Modeling Language (UML) models are considered one of the most highly ranked ones being used. Categorization of UML diagrams yields to categorization of test cases generation techniques according to diagrams being used. This includes:

Behavioural Diagrams: Describes the behavioural features of a system or business process and examples are Activity, State Chart, and Use Case diagram.

Interactional Diagrams: These diagrams are subset of behavioural diagrams that accentuate object interactions and examples include Communication, Sequence and Timing Diagrams

Structural Diagrams: Emphasize elements of specification, which are irrespective of time and their examples include Class, Component, Object, Package, Deployment and Composite Structure diagrams.

Behavioral and Interactional UML Models-based Techniques

Activity diagrams can be used to derive test scenarios, a technique uses a method called gray-box method. The technique contains manual steps in the algorithm of test generation. It doesn't handle fork-join efficiently and this limits the scope of the technique. It also doesn't do by all the paths; it only defines the basic paths. The fork-join structure problem was solved by the technique which uses an abstraction model obtained from fully expanded activity diagrams produced by only subjecting the external inputs and outputs. The model is then converted into a flow graph that is finally used to

ISSN: 2277-9655
Impact Factor: 1.852

extract test cases meeting the all-paths coverage criteria.

The interaction diagram is similar to the activity diagram, in that both visualize a sequence of activities. The difference is that, for an interaction diagram, each individual activity is pictured as a frame which can contain a nested interaction diagrams. This makes the interaction diagram useful to "deconstruct a complex scenario that would otherwise require multiple if-then-else paths to be illustrated as a single sequence diagram. Other types of diagrams have been used in many approaches to generate test cases like state chart, collaboration, and sequence diagrams. An algorithm that transforms a state chart diagram into an intermediate diagram, called the Testing Flow Graph (TFG) is shown [17]; from the TFG it generates test cases that apply the full state and full transition coverage criteria

Structural UML Models-based Techniques

Class and object diagrams are used to generate test cases. The methodology accepts the application code as input and runs it to create a list called the class list which contains features of classes mentioned in the application; it then uses this class list to extract the features of each class as well as the relationships between them. Finally test cases are generated based on these features and relationships. Another approach presented uses class, object, and state diagrams to define models written in a tool language called the Intermediate Format (IF). Descriptions written in IF can be animated, verified, and used to generate tests.

Kaur, et. al. [18] presents a systematic survey of work done in the field of automatic generation of test cases particularly related to UML-based automatic test case generation. This survey aims at summarizing the current state of the art in automatic test case generation research by covering questions below. The questions are:

1) What are various UML techniques used for automatic test case generation?
    - Search-based software test case generation
    - Finite State Machine
    - Model-based testing

The authors focus on UML-based techniques. The key techniques found were: UML diagrams, State Chart Diagrams, Sequence Diagrams, Activity Diagrams, Class Diagrams, Collaboration diagrams.

2) Which is the most widely used technique?

The most widely used techniques involve combination of various UML techniques for example use case and state diagram. This can be best analyzed from the bar graph given below. It can be easily noted that activity diagram and sequence diagram are the most widely used approaches so far. One of the oldest approaches for model-based testing is by using Use Case and State Diagrams. In this approach, the models are transformed into usage models to describe both system behavior and its usage. The method is intended for integration into an iterative software development process model.

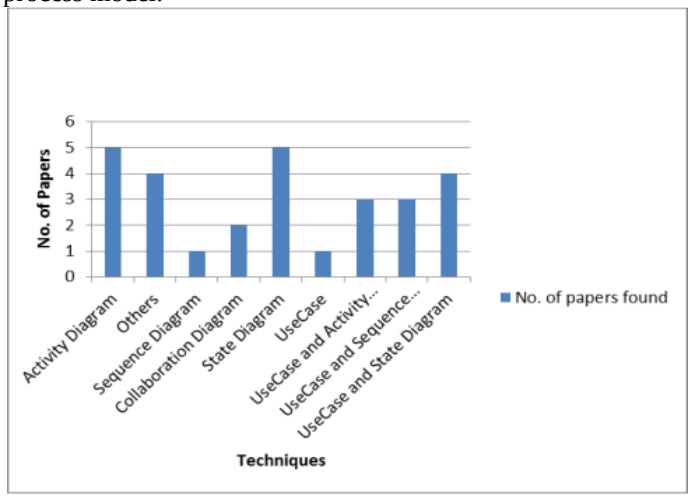

Figure 2 Most widely used techniques for test case generation[18]

3) What are the broad areas covered by these techniques?

The broad areas covered by these techniques includes Web applications, real time embedded systems, artificial intelligence planning, spreadsheets, system on chip designs and reactive systems, OO systems, SOA interacting services.

Anand, et. al. [19] discusses automatic test case generation techniques. This includes model-based testing, random-based testing and search-based testing.

1) **Model-based testing**(MBT) is a lightweight formal method, which uses models of software for derivation of test suites. In contrast to traditional formal methods, which aim at verifying programs against formal models, MBT aims at gathering insights in the correctness of a program using often-incomplete test approaches.
2) **Random-based testing** is one of the most fundamental and popular testing methods. This is simple in concept, easy to implement and can be used on its own as a component of testing methods.
3) In **Search-based testing**, an optimization algorithm is used to automate the search for test data that maximizes the achievement of test goals, while minimizing testing costs.

Frohlisch, et. al. [20] talks about automatically generating test cases from Use Cases. The steps are:

ISSN: 2277-9655
Impact Factor: 1.852

1) Formal transformation of a detailed use case description including pre- and post-conditions to a UML state model
2) Generation of test cases from the state model.

To construct a State Machine given the Use Case description following mappings are used: *Main Success Scenario* is the straightforward sequence of steps leading to the achievement of the user's goal without consideration of possible problems. With each step, possible error situation and their resolution can be described in the *Extensions*. Further, there may be different alternative ways to execute a step. These alternatives are described as *Variations*. The *Preconditions* captures constraints, which the state of the world must satisfy before the use case can be executed. These are typically properties of the user or the state of program execution. The *Post-conditions* on the other hand describes the conditions, which the use case establishes. Thus, Pre- and Post-conditions together define the *contract* of the use case. A step in the use case can refer to another use case being called. The UML state model is constructed making use of these notations.

In the next step to generate test cases from the UML model a planning problem called STRIPS is used. STRIPS is the most widely used AI planning formalism to derive a test suite for a given state chart. In the search space of a STRIPS planning problem, each state is described by a set of propositions, which hold in that state. A set of operators describes the transitions among the states. The planning task is to find a sequence of operators, which safely connects the initial state to the final state. STRIPS problem allows to systematically search for paths in the state machine, which satisfy all preconditions of the transitions. Using this planning technique ensures that the test sequences derived from the state machine are consistent in the sense that the preconditions of all transitions in the sequence are satisfied.

However, this approach had additional manual steps, such as:

1) The expected system response has to be added to the test sequence manually to yield complete test cases.
2) Although constraints are derived on the test inputs automatically, the concrete test data still has to be defined manually

So, in the future, this approach can be extended by providing stronger coverage criteria. The authors concentrate on use cases, so in future it can be done for model-based and structural-based diagrams.

Shanthi, et. al. [21] presents a survey on automatic test case generation using model-based testing through use of a UML model of systems. Three types are discussed here: Model-based, Scenario-based, and Genetic-based. Scenario-based mainly focuses on concurrent processes only in an activity diagram. Model-based systems focus on State Charts, Sequence, Object and Use-Case diagrams but do not produce optimal solution. Genetic-based produces optimal but generates faulty test cases. This survey tells that in spite of not producing an optimal solution model-based testing is preferred by many researchers as they utilize less human effort and low cost.

Gupta, et. al. [22] discusses Model-Based Testing (MBT) and automatic test case generation using MBT. Model-based test generation basically means functional testing for which test specification is given as a test model. In MBT test cases are derived automatically. The paper introduces MBT and software models as examples in addition to advantages of MBT and limitations of the approaches. A comparison between traditional manual testing and MBT shows that MBT is cheaper and faster. Test cases generated using MBT are effective in terms of code coverage and saves significant amount of man-hours required for test case generation per application.

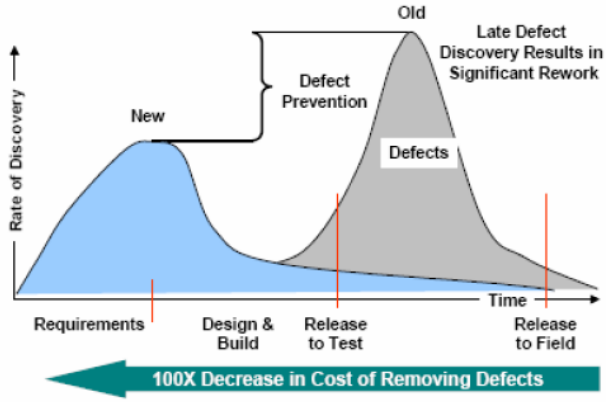

Figure 3 Savings due to early defect discovery[22]

The MBT approach is associated with several boons and benefits, which offer certain advantages over manual test case generation. They are:

1) Ability to Detect Functional faults earlier
2) Allows finding Faults even before Implementation Phase
3) Comprehensive test cases
4) Design is spontaneous.

Cavarra, et. al. [23] presents architecture for model-based testing using profile of UML. Class, Object and State diagrams can be used to define essential models. Models written in this profile can be compiled into a tool language: The Intermediate Form (IF). Description written in IF can be animated, verified and used to generate tests.

ISSN: 2277-9655
Impact Factor: 1.852

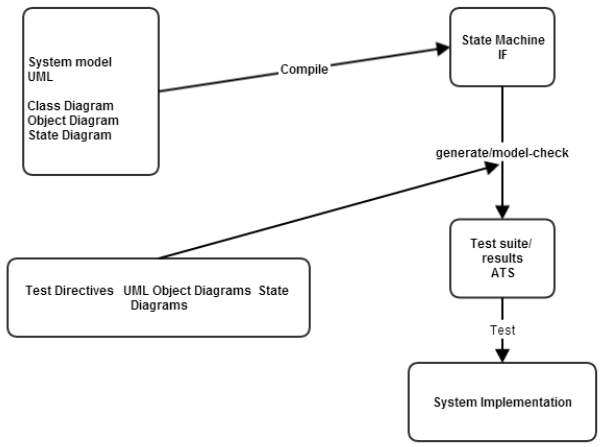

Figure 4 An Architecture for Automatic Test Case Generation[23]

The Intermediate Format (IF) language was developed to sit between high-level specification languages, such as Sdl, Promela or Lotos, and tool-specific internal representations. IF representations can be passed between tools, and translated into other languages: for example, Sdl specifications can be analyzed using the Spin model-checker. Moreover, translating high-level languages into IF may also allow extending some parts of their semantics: for example IF is used to give a precise timed semantics to Sdl.
The choice of using IF as an intermediate format between the AML modeling language and the test generation tool is motivated by several arguments:

- First of all, it allows us to *re-use* some of the tools already developed within the IF environment.
- Moreover, using IF offers a relative *flexibility* when defining the AGEDIS Modeling Language semantics: for a given AML construct, translation schemes can be foreseen, independently of the simulation engine.
- Finally, the potential loss of efficiency caused by the use of an intermediate representation is largely compensated by the optimization tools available at the IF level.

Class diagrams are description of set of objects that share same attributes, operation, relationships and semantics. Each class is drawn as a rectangle with 3 compartments: top holds class name, middle holds list of attributes, bottom holds list of operations. The Class diagrams are created as part of a software product's development process, this depicts all the components in the software. This helps in creating test cases as all the components and the relation between them can be laid out in the diagram. In order to generate a successful test case it is important that all components of a system and all the paths traversed by these components are covered in the test cases.
Object diagrams represent the state of a system at certain point in time, as collection of objects, each in particular state. Also describes initial configuration of system model.

Kaur, et. al. [24] describes an approach used for generating test cases automatically using a Sequence Diagram. The steps followed here are:

1. Using Rational Rose software, construct a Sequence diagram and save it with an .mdl extension.
2. Capture the object names by parsing the .mdl file.
3. Build a tree using object names and apply genetic algorithm's crossover technique.
4. Then convert new generated trees into binary trees.
5. Traversing is done by Depth First Search method of binary trees.
6. All the valid, invalid and termination of the application can be obtained using step 5.

This approach generates test sequences automatically but does not concentrate on number of faults revealed in the unit level or in the integration level.

Prasanna, et. al. [25] speaks about generating test cases automatically using Object Diagrams. Here are the steps that are followed to automatically generate the test cases:

1. Construct object diagram using Rational Rose software. The diagram is stored with an .mdl extension.
2. Parse the .mdl file and capture the object names.
3. Build a tree using object names and apply genetic algorithm's cross over technique.
4. New generation of trees are formed which are then converted it to binary trees.
5. Traverse new generation of binary trees using Depth First Search technique.
6. All the valid and termination sequences of the application can be obtained using Step 5.

Review Outcome

After the test sequences are generated by using mutation testing on the generated test cases method used to generate test cases revealed 80% fault in unit level and 88% in the integration level.

The approach applied for object diagram is applied on the class diagram. The intention of choosing class diagram is that class diagrams provide more information than object diagrams and hence it is expected that accuracy of revealing the faults in the

unit level and integration level would be higher than that of object diagrams.

Conclusion

A discussion on various approaches used for generating the test cases manually as well as automatically has been covered in this paper. Also, comparison is made between generating test cases manually and automatically[22] showing that automatic test case generation provides more accurate results than manually generated test cases. Finally, an approach called mutation testing is discussed that when applied on the method that generates test cases automatically reveals number of faults[25] in the unit level as well as integration level.

References

[1] *V.Mary Sumalatha, Dr G.S.V.P.Raju, "UML based Automated Test Case Generation technique using Activity-Sequence diagram", The International Journal of Computer Science & Applications (TIJCSA), Volume 1, No. 9, November 2012 ISSN – 2278-1080*

[2] *S. K. Swain. UML-based Testing of Software System, Technical Report, KIIT, 2005.*

[3] *T. Dinh Trong: "A Systematic Procedure for Testing UML Designs". ISSRE(2003).*

[4] *Korel, B. 1990. Automated software test data generation. IEEE Trans. Software Engineering,, 16(8), 1990, pp. 870 – 879*

[5] *Pakinam N. Boghdady, Nagwa L. Badr, Mohamed Hashem and Mohamed F.Tolba "Test Case Generation and Test Data Extraction Techniques" International Journal of Electrical & Computer Sciences IJECS-IJENS Vol: 11 No: 03, 2011.*

[6] *Utting M, Legeard B. Practical Model-based Testing: A tools approach. A Morgan Kaufmann publisher is an imprint of Elsevier: San Francisco, CA, 2007.*

[7] *Berenbach B, Paulish D, Kazmeier J, Rudorfer A. Software and Systems Requirements Engineering in practice. The McGraw-Hill Companies Inc.: USA, 2009.*

[8] *Everett GD, McLeod R, Jr. Software Testing: Testing across the Entire Software Development Life Cycle. IEEE press, John Wiley & Sons, Inc., Hoboken: New Jersey; 2007.*

[9] *Lazić L and Medan M. Software Quality Engineering versus Software Testing Process. The Telecommunication Forum (TELFOR) journal: Beograd, 2003.*

[10] *Smolander K. Quality Standards in Business Software Development. Master of Science Thesis, Lappeenranta University of Technology, Department of Information Technology, 2009.*

[11] *Graham D, Veenendaal E, Evans I, Black R. Foundations of Software Testing ISTQB Certification. International Software testing Qualifications Board, 2010.*

[12] *IEEE standard for software test documentation, IEEE Std 829-1998.*

[13] *CMMI product team. CMMI for development v 1.3 (CMU/SEI-2010-TR-033). Carnegie Mellon University, Software Engineering Institute, 2010.*

[14] *Andriole SJ (Editior). Software Validation, Verification, Testing and documentation. Petrocelli Books: Princeton, New Jersey, 1986.*

[15] *Nirpal PB and Kale KV. A Brief Overview of Software Testing Metrics. International Journal on Computer Science and Engineering (IJCSE), 2011.*

[16] *Kan SH, Parrish J, and Manlove D. In-process metrics for software testing. IBM Systems Journal, 2001.*

[17] *Huang CY, Lo JH, Kuo SY, and Lyu MR. Software Reliability Modeling and Cost Estimation Incorporating Testing-Effort and Efficiency. Proceedings of the 10th International Symposium on Software Reliability Engineering, IEEE Computer Society: Washington, DC, USA, 1999.*

[18] *Dr Arvinder Kaur, Vidhi Vig, "Systematic Review of Automatic Test Case Generation by UML Diagrams".*

[19] *Saswat Anand, Edmund Burke, Tsong Yueh Chen, et al., "An Orchestrated Survey on Automated Software Test Case Generation", Journal of Systems and Software, February 2013.*

[20] *Peter Frohlisch, Johannes Link, "Automated Test Case Generation from Dynamic Models", 2000 .*

[21] *A V K Shanthi, D Parthibhan, Dr G Mohan Kumar, "A Survey of UML-based Automatic Test Case Generation for software testing".*

[22] *Gaurav Gupta, Parampreet Kaur, "Inclination Towards Automated Model-based Test Generation", IJCSMC, Vol. 2, Issue. 7, July 2013, pg.302 – 311.*

[23] *Alessandra Cavarra, Thierry Jeron, Alan Hartman, Laurent Mounier, Sergey Olvovsky , "Using UML for Test Generation", ISSTA, 2002.*

[24] *Paramjit Kaur, Rupinder Kaur, "Approaches for Generating Test Cases Automatically to Test the Software", IJEAT, Volume-2, Issue-3, February 2013*

[25] *M. Prasanna, K.R. Chandran, "Automatic Test Case Generation for UML Object diagrams using Genetic Algorithm", Int. J. Advance. Soft Comput. Appl., Vol. 1, No. 1, July 2009 ISSN 2074-8523;*

ISSN: 2277-9655
Impact Factor: 1.852