AN ENHANCED TEST CASE GENERATION TECHNIQUE USING ACTIVITY DIAGRAM FOR SYSTEM TESTING

NANSUKUSA YUDAYA

A dissertation submitted in partial fulfillment of the
requirements for the award of the degree of
Master of Science (Computer Science)

Faculty of Computer Science and Information Systems
Universiti Teknologi Malaysia

JANUARY 2013

I affably dedicate this thesis to the biggest treasures of my life, my family. Not forgetting my best friend Farooq who has always believed in me.

ACKNOWLEDGEMENT

First of all, I would like to express my utmost gratitude to Allah S.W.T for His endless blessings and guidance throughout my entire research process and stay in Malaysia, Alhamdlillah for everything.

Then, sincere appreciation goes to my supervisor **Associate Professor Dayang Norhayati Abang Jawawi** for her continued support, guidance, and patience throughout my research. Despite her tight schedule, she always tried to make herself available. I've never seen anyone as committed in nurturing their students like she is. I will always look up to her as my academic role model.

I would also like to express my gratitude to my dear sponsors, Islamic Development Bank, for always providing me with sufficient financial support, may Allah reward you abundantly.

I will forever be grateful to my family, for their undulating support, encouragement and prayers, not forgetting my best friend Farooq. To them I am truly indebted and words alone cannot describe my earnest gratitude.

Special thanks go to all my friends who have always provided aid at various occasions through their views and tips that were undeniably constructive throughout my research and stay in Malaysia. You will all forever remain at heart.

ABSTRACT

Software Testing, a process comprised of test case generation, execution and evaluation is one of the imperative phases of the development life cycle, with its cost approximated to about 50% of the overall development cost. Researchers have automated it using models with utmost focus put on Unified Modeling Language (UML) as the up to date de facto standard utilized in software modeling. Its diagrams include both behavioral and structural. This work has generated system tests (black box) early in the development lifecycle hence the use of behavioral models, activity diagrams in particular as they are one of the earliest and simplest analysis models to be created with sufficient testing information. Also, as a way of reducing test case generation effort and time, an existing technique that supposedly involved more effort and time has been focused on in this work. It has been enhanced by reducing the key steps involved through eliminating intermediate models as a way of reducing effort and time involved in the test case formation process. The enhanced technique has been applied on the same case study as in the original technique, producing four test cases in 115 milliseconds with more ease compared to the original technique that produced five test cases in 160 milliseconds with relatively more effort. It has further been compared against another already existing model based technique (based on sequence diagrams) and also one integrated with a model based tool using both general criteria and those specific to the research problem (TCG effort and time), as a way of further confirming its applicability.

ABSTRAK

Pengujian Perisian, satu proses yang terdiri daripada penjanaan kes ujian, pelaksanaan, dan penilaian adalah salah satu fasa penting untuk kitaran hayat pembangunan, dengan kos yang dianggarkan kira-kira 50% daripada keseluruhan kos pembangunan. Penyelidik telah mengautomasikan ia menggunakan model dengan tumpuan penuh diletakkan ke atas Bahasa Permodelan Bersepadu (UML) sebagai standard de facto terkini yang digunakan dalam pemodelan perisian. Rajahnya termasuk kedua-dua tingkah laku dan struktur. Kerja ini telah menjana ujian sistem (kotak hitam) di awal kitaran hayat pembangunan, maka penggunaan model tingkah laku, rajah aktiviti khususnya kerana mereka adalah salah satu model analisis yang terawal dan paling mudah untuk diwujudkan dengan maklumat ujian yang mencukupi. Sebagai satu cara untuk mengurangkan usaha dan masa untuk menjana kes ujian, satu teknik yang sedia ada yang sepatutnya melibatkan usaha dan masa yang lebih telah diberikan tumpuan dalam kerja-kerja ini. Ia telah dipertingkatkan dengan mengurangkan langkah-langkah utama yang terlibat melalui penghapusan model perantaraan sebagai satu cara untuk mengurangkan usaha dan masa yang terlibat dalam proses pembentukan ujian kes. Teknik yang dipertingkatkan ini telah digunakan pada kajian kes yang sama seperti dalam teknik asal, menghasilkan empat kes ujian dalam 115 milisaat dengan lebih mudah berbanding dengan teknik asal yang menghasilkan lima kes ujian dalam 160 milisaat dengan usaha yang agak lebih. Ia selanjutnya telah dibandingkan dengan satu lagi teknik yang sedia ada model berasaskan (berdasarkan rajah jujukan) dan juga bersepadu dengan alat berasaskan model menggunakan kedua-dua kriteria umum dan yang khusus kepada masalah penyelidikan (TCG usaha dan masa), sebagai satu cara untuk selanjutnya mengesahkan penggunaannya.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AD	Activity Diagram
ADG	Activity Dependency Graph
ADT	Activity Dependency Table
ATM	Automatic Teller Machine
BFS	Breadth First Search
CAG	Cyclomatic Activity Graph
DFS	Depth First Search
EUROCAE	European Organization for Civil Aviation Equipment
IFD	Interaction Flow Diagram
IFG	Interaction Flow Graph
LTS	Labeled Transition System
MBT	Model Based Testing
OCL	Object Constrained Language
PIN	Personal Identification Number
RTCA	Radio Technical Commission for Aeronautics
SD	Sequence Diagram
SDT	Sequence Dependency Table
SUT	System Under Test
TCG	Test Case Generation
UML	Unified Modeling Language

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Overview

The recent past has seen the software industry change at a fast pace, many software systems have become larger, additionally complex, and rather integrated. This has in turn made software very sophisticated yet it makes up the larger part of most systems, often replacing hardware (Jerraya and Wolf, 2005).

Areas for example the embedded systems fields are growing enormously important that they potentially represent a revolution in information technology (IT). Their growth is determined by the rising potential and ever-declining costs of computing and communications devices, resulting in networked systems of embedded computers whose functional components are nearly invisible to end users (Mercuri et al., 2008).

Software is applied extensively in a variety of system applications from small sized ones like mobile phones to enormous systems such as machine condition monitoring, airbag control systems, marine and military security (Hua-ming and Chun, 2009) and many others. Because of rapid changes in this field, future applications will most likely contain even more software. The forecast for the next 10 years is an exponential growth of the market for these products (Liggesmeyer and Trapp, 2009). When system intricacy rises, the development lifecycle must also

change. For this reason, efficient system development methodologies have to be used so as to handle the team size, the product requirement (scope) and to meet the project's restraints like time-to-market and costs.

Software in various system applications is steadily fetching even higher importance than the hardware because of the formation of novel inventive meanings based on software and also because of the execution of previously automatically included functions in software (Zander-Nowicka et al., 2007). Accuracy of software applications' functionality, usability and performance gives a crucial task in software quality. Software testing is a vital means of software quality assurance, satisfied by various attributes such as effort and time (Hua-ming and Chun, 2009).

The intricacy of software applications is growing because of the increased customer's demand together with the time restrictions needed to get the artifact on marketplace. Even when its intricacy is mounting, the time to market is diminishing. Because of this, effort and time slotted to generating software is kept as a significant aspect in product selling so as to ensure its quality (Seon-Jae et al., 2008). This therefore means that the manual (traditional) way of testing cannot meet software industry demands, hence automated testing is introduced, commonly referred to as model based testing (Schieferdecker, 2012b).

To define software testing, some researchers would call it a method, or a sequence of processes, of dynamically executing a program with given inputs, so as to ensure that the computer code does exactly that for which it is designed. Various applications are turning out to be ever more omnipresent, taking care of an extensive diversity of well accepted and safety-critical devices. Testing is the frequently utilized technique used in authenticating software applications, and efficient testing methods may well be supportive for enhancing the reliability of such systems.

Figure 1.1 illustrates some of the general examples of software applications such as Automatic Teller Machines (ATMs), cellular telephones, satellites, airplanes, security alarm systems robotics and many others that are used in different fields.

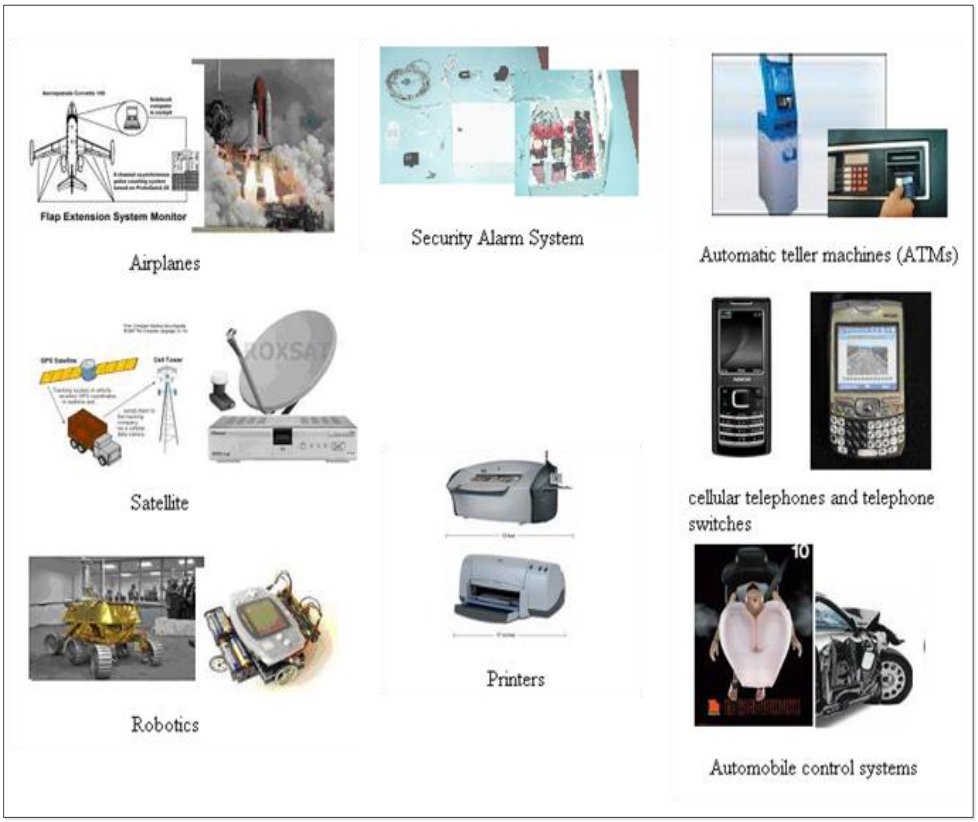

Figure 1.1 Examples of Software System Applications (Sabil, 2010).

Software testing offers help not only in locating bugs but also in reducing effort and time spent in mitigating these bugs. Current studies from Electric Cloud carried out together with Osterman Research show that a greater part of software bugs are ascribed to poor testing procedures or infrastructure limitations rather than design problems (Kandl, 2010).

Despite the fact that manual testing is an easy process to follow through its clearly laid out steps, it still involves more effort and time as compared to automated testing which has fewer time takings. As a result, automated testing is taking the lead for software applications nowadays because it is faster given that time is considered a very significant constraint as a determinant of quality (Schieferdecker, 2012).

According to Seidewitz (2003), Unified Modeling Language (UML) 2.0 offers various models applied in the development lifecycle such as use case diagrams

that are primarily made up of use cases (represented using behavioral diagrams (Dhir, 2012)), sequence, activity, class, state machine diagrams, etcetera. It is also recognized as a visual language (Dai, 2004).

Apart from extensively using it for modeling object oriented systems, it is also employed in designing tests on the different levels such as unit, integration and system, (Tahiliani and Pandit, 2012). Many researchers are now proposing ways of re-using its design models for test case generation despite the fact that many other techniques using different models exist. This is to help reduce software testing effort and time as the models are already created by system developers, also encouraging leverage by testers.

Because software is applied in many fields with a stringent quality requirement, any failure will in one way or the other have an effect on individual life and assets, not forgetting the security of the ecological environment. Accuracy of software functionality and performance takes up a critical responsibility in the software quality with software testing as a significant means of quality assurance. This therefore calls for efficient testing of software application so as to avoid compromising any company's name once customer satisfaction is not met.

1.2 Problem Background

The notion of quality started way back in time when people attempted to discover an apt quality in every artificial object. The last decade has also seen a sudden boost of the Internet, meaning that the quality uprising is now circulating all through the world swiftly. Various organizations have then recognized that triumph in the novel worldwide market necessitates developing quality results (products).

Originally; to enhance feature quality, many attempts were intense around testing products mainly towards development (lifecycle) end. Of late, the idea of

quality is giving attention to the works involving all production phases from the analysis of the requirements until the delivery of the product to customer, paying attention to uncovering and mitigation of defects at a lesser cost (Beck et al., 2001).

In our daily lives, change is part of human nature, so it is for software applications. Software requirements change almost on a daily basis, calling for change in software development and testing techniques. This also introduces system design intricacy together with an extremely tense time-to market period because of the inconsistence, inaccuracy and ambiguity of the requirements.

Testing software systems such as those for mission-critical applications is particularly tricky because of their unique features like timeliness, owing to design complexities and frequent requirements changes (Wei-Tek et al., 2005). Other small sized applications like mobile phones have limited storage space which is also challenging for designers and testers especially with choosing the adequate test attributes for it.

The software development process distinguishes testing as a time and resource (effort) overwhelming subject matter. According to Micskei and Majzik (2006), over 30% of the process' emphasis ought to be set aside for testing. Coming up with a simple and efficient test suite typically involves lots of manual work and specialist (expertise) familiarity. A testing engineer's usual responsibilities include coming up with test cases (input-output pairs) for imperative tasks, clustering them into test sequences and test suites, after which they are executed and later analyzed.

Traditionally, test suites are generated from the start (scratch) using different, frequently insufficient manual techniques. Finding the test data physically by hand enormously takes much effort and time, particularly if the software has complicated features to test. As a result, various efforts have been made to automate the process, making it faster and more reliable hence genesis of model based testing (MBT). Numerous researchers have deeply surveyed this field with an intention of finding out how exactly the test data (input and output) is generated, executed and later

evaluated against any given system. Test data generation has proven the most challenging step that determines the correctness of the next phases.

Copious attributes depicting quality of software exist for example effort, time and cost (Yacoub et al., 1999). According to Bruntink and van Deursen (2006), effort of the classes of object-oriented software involved in the test case generation process can be estimated by Lines of Code for Class (LOCC) and Number of Test Cases (NOTC). Other researchers including Shrivastava and Jain (2011) estimate test design effort by Depth of Inheritance Tree (DIT), Fan out (FOUT), Lack of Cohesion on Methods (LCOM) and many others. Time can be measured either using a wall clock or CPU cycles (Ali et al., 2010). Sometimes, test efforts are considered dependent on the test time (Nageswaran, 2001).

Various techniques have been used by different researchers and practitioners to automate test case generation both in academia and industry because the activities involved are consuming an increasing amount of resources allocated to software development projects (Chimisliu and Wotawa, 2012). Automating test case generation involves use of manually chosen algorithms to automatically and systematically form test cases from a set of models of the system under test or its environment (Schieferdecker, 2012b).

According to Hyungchoul et al (2007), among the several models that exist, those created with Unified Modeling Language (UML) are thought of as the most highly ranked type. Boghdady et al (2012) then comes in to state that even within these UML models, activity diagrams are one of the most famed models that represent business and operational workflows of a system. Using gray box method to generate test cases directly from a UML activity diagram is one of the methods proposed, where the design model is reused to avoid the cost of test model creation. The test scenarios are extracted directly from the activity diagram modeling an operation while the test case generation data such as input or output sequence and parameters, the constraint conditions and expected object method sequence, is derived from each test scenario. (Linzhang et al., 2004).

An approach to capture, store and output usage scenarios derived automatically from UML activity diagrams was also reviewed. In this work, they presented an approach "dubbed AD2US", which automatically extracts Usage Scenarios (USs) from Activity Diagrams (ADs); thereby extending the time available for other activities such as test-case generation or the verification of consistency between ADs, use cases and usage scenarios (Mingsong et al., 2006).

Generation of test scenarios using activity diagrams with specialty in handling fork-join pairs present in activity diagrams is a technique sought by P. Nanda (2008). A test case generation technique based on activity diagrams proposes a model that introduces an algorithm which automatically creates a table called Activity Dependency Table (ADT), and then uses it to create a directed graph called Activity Dependency Graph (ADG) which is later traversed to create test paths translated into test cases (Boghdady et al., 2012; Boghdady et al., 2011a; Boghdady et al., 2011b).

Sequence diagrams are also important UML models in the creation of test cases under the MBT umbrella as reported by several research attempts. Sarma et al, (2007) as well as Sawant and Shah (2011b) both describe an approach to transform a UML sequence diagram into a sequence diagram graph (SDG) and augmenting its nodes with information needed to compose test vectors so as to later traverse it for test case formation. To create mobile phone application functional test cases, a UML sequence diagram is translated into Labeled Transition Systems (LTSs) where Depth First Search (DFS) technique is applied on the LTS so as to obtain test paths transformed into test cases.

In other approaches, a Seditec tool is proposed to take sequence diagrams as input to automatically generate test stubs for the classes and methods whose behavior is specified in the sequence diagrams (Fraikin and Leonhardt, 2002). Test case generation by means of UML Sequence diagram using Genetic Algorithm in which the best test cases are optimized and the test cases are validated by prioritization is also reviewed (Shanthi and MohanKumar, 2012).

In summary, traditional generation of test cases is carried out towards the end of the development lifecycle, ultimately compromising quality of the software. UML models have been considered the most popular, especially activity and sequence diagrams in the effort to boost early test case creation. Considering researchers who have attempted to reverse the ideology of observing testing at the end, a few of them have considered using these models directly hence decreasing the steps involved in reaching the tests, eventually decreasing the effort and time involved there within.

In addition, all the techniques conferred focus on generating tests through intermediate models such as those proposed by Sawant and Shah (2011a), Boghdady et al (2011a) and many others. Aside increasing effort and time involved in their construction, errors likely to be encountered also increase. Therefore, besides observing test case generation as a last phase in the development lifecycle, most researchers also continued to use intermediate models, increasing effort and time that would rather be reduced if they were being used directly.

While Activity diagrams give a clear control flow of activities in a given system, sequence diagrams explain its dynamic behavior through exchange of messages between the objects involved. Because these are among the first behavioral models created in the analysis phase, early tests are expected to be generated with less effort and time by eliminating the intermediate models that were rather delaying the entire test case generation process.

1.3 Problem Statement

Testing is a crucial part of quality assurance but as the complexity of software applications grows, more effort and time is entailed. Because testing consists of three phases, such as test case generation, execution and evaluation in that order, the first one is a more challenging task compared to the rest because its correctness determines that of those which follow. Carrying it out manually would only increase

the effort and time hence increasing the costs involved in identifying and mitigating any errors (Zander-Nowicka et al., 2007). This has then prompted many researchers into automation, also referred to as Model Based Testing (Schieferdecker, 2012b).

Because Test case creation (generation) is of indispensable significance, this study has focused on the use of Unified Modeling Language (UML) behavioral models to accomplish it successfully. Various techniques have used activity diagrams because of their simplicity and sufficient testing information, also because they are one of the earliest models created in the development lifecycle thus encouraging early testing. Most of them have emphasized creation of either intermediate tables, graphs or both. In due course, this has increased test case generation effort and time needed to create tests. This research therefore proposed elimination of these intermediate models by directly traversing activity diagrams, reducing on likely errors created during their construction besides the effort and time wasted there within.

Therefore, the research question posed is *"Is it possible to reduce the effort and time spent on test case generation by eliminating intermediate models without compromising the quality of the test cases?"*

1.4 Research Aim

The research is aimed at enhancing an existing UML activity diagram test case generation technique in terms of effort and time by reducing steps involved through intermediate models riddance.

1.5 Research Objectives

This study consists of a set of objectives that lead to the research process. They include:

- To identify contemporary issues of test case generation techniques in system (black box) testing.
- To propose and apply a test case generation technique that improves effectiveness (effort and time) for system testing with a software application case study.
- To compare the proposed technique with the original one, an existing model based (sequence diagram) technique and automated tool so as to deem it successful in reducing effort and time as anticipated.

1.6 Scope of the Study

This study is intended for several small to medium-sized software applications that require accuracy and consistence in their system functionality before they are released to the market, i.e., those where testing the customer requirements for quality is vital. The following were paramount:

- The study has focused on only UML behavioral models, activity diagrams and sequence diagrams in particular for system level (black box) test case generation, leaving out the rest of the other models.
- Techniques focusing on only one model at a time have been considered, leaving out those dealing with model integrations.
- It has also chosen an existing technique using activity diagrams studied and enhanced it in terms of effort and time elapsed during test case generation, and then applied it on a software application example.

- The enhanced test case generation technique is then compared with the original one, an already existing technique using sequence diagrams and another technique integrated with automated model based tools.
- The work has been concluded by proposing primary ideas to address some of the practical challenges identified in applying them on both academia and industrial test case design.

1.7 Significance of Study

This study is primarily intended to benefit the software applications industry by focusing on less costly and earlier alternative test case generation techniques that will have testers leverage on analysts and developers by re-using UML analysis behavioral models. Most of the existing techniques do not particularly compare test cases generated by either activity and sequence or any other UML models. This will in turn do well to the different areas that employ various small sized software applications in their daily operations for example, the telecommunications, banking, transportation and other industries.

It will also be helpful in homes where performing daily chores for example washing and vacuuming requires use of the small to medium sized software applications since test cases to validate them before being released for use will be created. It will also be beneficial to the education sector as it will largely contribute to the body of knowledge for the students interested in carrying out their research in the area of software testing.

1.8 Dissertation Organization

This thesis is made up of six chapters, where by the first chapter clearly states the motivation and objectives of this work. The second chapter described the relevant literature review, the fundamentals model based testing. Chapter three introduced the methodology that was used to build up test cases as input for the subsequent chapters. Test cases were experimentally generated using an activity diagram in chapter four and later compared with already existing ones of a sequence diagram in chapter five. They were further compared with those generated from techniques integrated with automated tools. The final chapter of this work concluded it all by proposing initial ideas to identified challenges during test cases generation and also regarding the tests and techniques employed throughout this work.

REFERENCES

Abdurazik, A., and Offutt, J. (2000). *Using UML collaboration diagrams for static checking and test generation*, Lecture Notes in Computer Science Volume 1939, 2000, pp 383-395.

Ali, S., Briand, L. C., Hemmati, H., and Panesar-Walawege, R. K. (2010). A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation. *Software Engineering, IEEE Transactions on*, 36(6), 742-762.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., et al. (2001). Manifesto for agile software development. *The Agile Alliance*, 2002-2004.

Boberg, J. (2008). *Early fault detection with model-based testing*, Proceedings of the 7th ACM SIGPLAN workshop on ERLANG, Pages 9-20.

Boghdady, P., Badr, N. L., Hashem, M. A., and Tolba, M. F. (2012, 14-16 May 2012). *An enhanced technique for generating hybrid coverage test cases using activity diagrams.* Paper presented at the Informatics and Systems (INFOS), 2012 8th International Conference on, SE-20-SE-28.

Boghdady, P. N., Badr, N. L., Hashem, M., and Tolba, M. F. (2011a). A Proposed Test Case Generation Technique Based on Activity Diagrams. *International Journal of Engineering & Technology IJET-IJENS, 11*(03).

Boghdady, P. N., Badr, N. L., Hashim, M. A., and Tolba, M. F. (2011b). *An enhanced test case generation technique based on activity diagrams*, 289-294.

Bruntink, M., and van Deursen, A. (2006). An empirical study into class testability. *Journal of Systems and Software*, 79(9), 1219-1232.

Burnstein, I. *Practical software testing: a process-oriented approach*: Springer, 2003.

Cartaxo, E. G., Neto, F. G. O., and Machado, P. D. L. (2007). *Test case generation by means of UML sequence diagrams and labeled transition systems*, This paper

appears in: Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on 1292-1297.

Chen, M., Mishra, P., and Kalita, D. (2010). Efficient test case generation for validation of UML activity diagrams. *Design Automation for Embedded Systems, 14*(2), 105-130.

Chimisliu, V., and Wotawa, F. (2012, 19-21 March 2012). *Model based test case generation for distributed embedded systems.* Paper presented at the Industrial Technology (ICIT), 2012 IEEE International Conference on, 656-661.

Clarke, E. M., Grumberg, O., and Peled, D. A. (2000). *Model checking*: MIT press.

Dai, Z. R. (2004). Model-driven testing with UML 2.0. Proceedings of the 2nd European Workshop on Model Driven Architecture, *Computer Science at Kent*, 179.

Debbarma, M. K., Kar, N., and Saha, A. (2012). *Static and dynamic software metrics complexity analysis in regression testing*. Paper presented at the Computer Communication and Informatics (ICCCI), 2012 International Conference on, 1-6.

Devaraj, G., Heimdahl, M. P. E., and Liang, D. (2005). *Coverage-directed test generation with model checkers: Challenges and opportunities*, This paper appears in: Computer Software and Applications Conference, 455-462 Vol. 2.

Dhir, S. (2012). *Impact Of UML Techniques In Test Case Generation*, International journal of engineering science & advanced technology, Volume-2, Issue-2, 214 – 217.

Do, R. (1992). 178B, Software considerations in airborne systems and equipment certification. *Radio and Technical Commission for Aeronautics*.

Elish, K. O., and Alshayeb, M. (2009). *Investigating the Effect of Refactoring on Software Testing Effort*. Paper presented at the Software Engineering Conference, 2009. APSEC'09. Asia-Pacific, 29-34.

Fenton, N.E., and Pfleeger, S.L. (1996) *Software Metrics: A Rigorous and Practical Approach,* PWS Publishing Company, Boston, Massachusetts, USA.

Fenton, N. E., and Neil, M. (1999). Software metrics: successes, failures and new directions. *Journal of Systems and Software*, 47(2), 149-157.

Fenton, N. E., and Neil, M. (2000). *Software metrics: roadmap*. Paper presented at the Proceedings of the Conference on the Future of Software Engineering, 357-370.

Fraikin, F., and Leonhardt, T. (2002). *SeDiTeC-testing based on sequence diagrams*, Proceedings. ASE 2002. 17th IEEE International Conference on 261-266.

Gutiérrez, J. J., Escalona, M. J., Mejías, M., and Torres, J. (2006). *Generation of test cases from functional requirements. A survey*.In: 4th Workshop on System Testing and Validation , Potsdam, Germany.

Harrold, M. J., McGregor, J. D., and Fitzpatrick, K. J. (1992). *Incremental testing of object-oriented class structures.* Paper presented at the Proceedings of the 14th international conference on Software engineering, 68-80.

Hasling, B., Goetz, H., and Beetz, K. (2008). *Model based testing of system requirements using UML use case models*, in Proceedings of 2008 International Conference on Software Testing, Verification, and Validation, pp.367-376.

Heinecke, A., Bruckmann, T., Griebe, T., and Gruhn, V. (2010). *Generating Test Plans for Acceptance Tests from UML Activity Diagrams*, Engineering of Computer Based Systems (ECBS), 2010 17th IEEE International Conference and Workshops, pp.57-66.

Heineman, G. P., S. Selkow. (2008')." *Algorithms in a Nutshell"* book by O'Reilly Media.

Hua-ming, Q., and Chun, Z. (2009, 11-13 Dec. 2009). *A Embedded Software Testing Process Model.* Paper presented at the Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on, 1-5.

Hunter, J. (2009). improve automation, documentation and transparency of the entire software development process. Retrieved 15/10/2012, 2012.

Hutcheson, M. L. (2003). *Software testing fundamentals: methods and metrics*: book by Wiley.

Hyungchoul, K., Sungwon, K., Jongmoon, B., and Inyoung, K. (2007, July 30 2007-Aug. 1 2007). *Test Cases Generation from UML Activity Diagrams.* Paper presented at the Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on, 556-561.

Jerraya, A. A., and Wolf, W. (2005). *Hardware/software interface codesign for embedded systems.* Computer journals, volume(2), pg.63-69.

Kandl, S. (2010). *A Requirement-Based Systematic Test-Case Generation Method for Safety-Critical Embedded Systems.* University Technology Vienna.

Kansomkeat, S., and Rivepiboon, W. (2003). *Automated-generating test case using UML statechart diagrams*, Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology, Pg. 296-300.

Khan, M. B. A., and Shang, S. (2009). *Evaluation of Model Based Testing and Conformiq Qtronic.* Final Thesis Evaluation, Sweden, Linköping.

Khandai, M., Acharya, A. A., and Mohapatra, D. P. (2011). *A novel approach of test case generation for concurrent systems using UML Sequence Diagram*, Electronics Computer Technology (ICECT), 2011 3rd International Conference, Vol 6, pg. 157-161.

Kuhn, R., Kacker, R., Yu, L., and Hunter, J. (2009). Combinatorial Software Testing. *Computer journals and magazines, 42*(8), 94-96.

Kumar, A. V. K. S. a. G. M. (2011). Automated Test Cases Generation from UML Sequence Diagram.

Li, K., Kou, J., and Gong, L. (2011). Predicting software quality by optimized BP network based on PSO. *Journal of Computers, 6*(1), 122-129.

Liggesmeyer, P., and Trapp, M. (2009). Trends in embedded software engineering. *Software, IEEE, 26*(3), 19-25.

Linzhang, W., Jiesong, Y., Xiaofeng, Y., Jun, H., Xuandong, L., and Guoliang, Z. (2004). *Generating test cases from UML activity diagram based on gray-box method*, Software Engineering Conference, 11th Asia-Pacific, pg. 284-291.

Mahesh S., Rajeev K. (2010). *A Hybrid Genetic Algorithm Based Test Case Generation Using Sequence Diagrams*, Contemporary Computing - Third International Conference, IC3 2010, Noida, India, pg 53-63.

Mark Utting, B. L. (2007). *practical model-based Testing a tools approach.* In 7th International Conference on Software Engineering (ICSE).

McCabe, T. J. (1976). A complexity measure. *Software Engineering, IEEE Transactions on*(4), pg. 308-320.

Mercuri, M., Joseph, J., Hogg, J., Ossipov, D., Mascaro, M., Garber, D., et al. (2008). Considerations for Designing Distributed Systems. *The Architecture Journal*.

Michael, J. B., Bossuyt, B. J., and Snyder, B. B. (2002). *Metrics for measuring the effectiveness of software-testing tools.* Paper presented at the Software

Reliability Engineering, 2002. ISSRE 2003. Proceedings. 13th International Symposium on, 117-128.

Micskei, Z., and Majzik, I. (2006, 25-27 May 2006). *Model-based Automatic Test Generation for Event-Driven Embedded Systems using Model Checkers.* Paper presented at the Dependability of Computer Systems, 2006. DepCos-RELCOMEX '06. International Conference on, 191-198.

Mingsong, C., Xiaokang, Q., and Xuandong, L. (2006). *Automatic test case generation for UML activity diagrams*, Proceedings of the 2006 international workshop on Automation of software test, pg. 2-8.

Nageswaran, S. (2001). *Test effort estimation using use case points.* Paper presented at the Quality Week, 1-6.

NichaKosindrdecha, J. (2005 – 2010.). A Test Generation Method Based On State Diagram. *Journal of Theoretical and Applied Information Technology*.

Nirpal, P. B., and Kale, K. (2011). A Brief Overview Of Software Testing Metrics. *International Journal on Computer Science and Engineering* (IJCSE), 3(1).

P. Nanda, D. P. M. a. S. K. S. (2008). *Generation of Test Scenarios Using Activity Diagram*. Paper presented at the Proceedings of SPIT-IEEE Colloquium and International Conference.

Pan, J. (1999). Software testing. Book *Retrieved January*, *5*, 2006.

Patel, P., and Patil, N. N. (2012). Test case formation using UML activity diagram. *World Journal of Science and Technology*, *2*(3).

Paulish, M. K.-H. a. D. J. (1993). *Software metrics: A practitioner's guide to improved product development.* Retrieved 8 January, 2013.

Periyasamy, K., and Liu, X. (1999, Aug 1999). *A new metrics set for evaluating testing efforts for object-oriented programs.* Paper presented at the Technology of Object-Oriented Languages and Systems, 1999. TOOLS 30 Proceedings, 84-93.

Riebisch, M., Philippow, I., and Götze, M. (2003). UML-based statistical test case generation. *Objects, Components, Architectures, Services, and Applications for a Networked World*, pg. 394-411.

Rodrigues Barbosa, J., Eduardo Delamaro, M., Carlos Maldonado, J., and Marcelo Rizzo Vincenzi, A. (2011). *Software Testing in Critical Embedded Systems: a Systematic Review of Adherence to the DO-178B Standard*, 126-130.

Rosenberg, L., Hammer, T. F., and Huffman, L. L. (1998). *Requirements, testing, and metrics, In: 15th Annual Pacific Northwest Software Quality Conference*.

Sabharwal, S., Singh, S. K., Sabharwal, D., and Gabrani, A. (2010, 17-19 Sept. 2010). *An event-based approach to generate test scenarios.* Paper presented at the Computer and Communication Technology (ICCCT), 2010 International Conference on, 551-556.

Sabil, S. B. (2010). *A Syatematic Component-Based Development Process Model Using Integrated MARMOT And PECOS Methods.* Unpublished Full Research, Universiti Teknologi Malaysia, Johor Bahru.

Saglietti, F. (2010). *Testing for Dependable Embedded Software*, Software Engineering and Advanced Applications (SEAA), 36th EUROMICRO Conference, pg. 409-416.

Samuel, P., Mall, R., and Kanth, P. (2007). Automatic test case generation from UML communication diagrams. *Information and Software Technology, 49*(2), 158-171.

Santiago, V., Vijaykumar, N. L., Guimaraes, D., Amaral, A. S., and Ferreira, E. (2008, 9-11 April 2008). *An Environment for Automated Test Case Generation from Statechart-based and Finite State Machine-based Behavioral Models.* Paper presented at the Software Testing Verification and Validation Workshop, 2008. ICSTW '08. IEEE International Conference on, 63-72.

Sarma, M., Kundu, D., and Mall, R. (2007). *Automatic test case generation from UML sequence diagram*, Advanced Computing and Communications, ADCOM. International Conference, pg. 60-67.

Sarma, M., and Mall, R. (2007). *System testing using UML Models*, Asian Test Symposium, 155-158.

Sawant, V., and Shah, K. (2011a). Construction of Test Cases from UML Models. *Technology Systems and Management*, 61-68.

Sawant, V., and Shah, K. (2011b). Construction of Test Cases from UML Models

Shrivastava, D. P., and Jain, R. C. (2011, 3-5 March 2011). *Unit test case design metrics in test driven development.* Paper presented at the Communications, Computing and Control Applications (CCCA), 2011 International Conference on, 1-6.

Technology Systems and Management. In K. Shah, V. R. Lakshmi Gorty and A. Phirke (Eds.), (Vol. 145, pp. 61-68): Springer Berlin Heidelberg.

Schieferdecker,I.(2012a).http://www.fokus.fraunhofer.de/en/motion/ueber_motion/technologien/fokusmbt/index.html. Retrieved 15/10/2012, 2012

Schieferdecker, I. (2012b). Model-Based Testing. *Software, IEEE, 29*(1), 14-18.

Seidewitz, E. (2003). What models mean. *Software, IEEE, 20*(5), 26-32.

Seon-Jae, J., Hae-Geun, K., and Youn-Ky, C. (2008, 14-16 May 2008). *Manual Specific Testing and Quality Evaluation for Embedded Software.* Paper presented at the Computer and Information Science, 2008. ICIS 08. Seventh IEEE/ACIS International Conference on, 502-507.

Shanthi, A., and MohanKumar, G. (2012). A Heuristic Approach for Automated Test Cases Generation from Sequence Diagram using Tabu Search Algorithm. *European Journal of Scientific Research, 85*(4), 534-540.

Sommerville. (2000). *Software Engineering, 6th Edn*. Book by Addison-Wesley, England.

Sundmark, D., Pettersson, A., Eldh, S., Ekman, M., Thane, H., and Ericsson, A. (2008). *Efficient system-level testing of embedded real-time software*, 5.

Swain, S. K., Mohapatra, D. P., and Mall, R. (2010a). Test case generation based on state and activity models. *Journal of Object Technology, 9*(5), 2010.

Swain, S. K., Mohapatra, D. P., and Mall, R. (2010b). Test case generation based on use case and sequence diagram. *Int. J. of Software Engineering, IJSE, 3*(2).

Swain, S. K., Mohapatra, D. P., and Mall, R. (2010c). Test case generation based on use case and sequence diagram. *International Journal of Software Engineering, IJSE, 3*(2), 21-52.

Tahiliani, S., and Pandit, P. (2012). A Survey of UML-Based approaches to Testing, *International Journal Of Computational Engineering Research (ijceronline.com) Vol. 2 Issue. 5*.

Torsel, A. M. (2011, 18-22 July 2011). *Automated Test Case Generation for Web Applications from a Domain Specific Model.* Paper presented at the Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual, 137-142.

Utting, M., Pretschner, A., and Legeard, B. (2006). A taxonomy of model-based testing, (Working paper series. University of Waikato, Department of Computer Science. No. 04/2006). Hamilton, New Zealand: University of Waikato.

Utting, M., Pretschner, A., and Legeard, B. (2011). A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*.

Wang, R., and Huang, N. (2008). *Requirement Model-Based Mutation Testing for Web Service*, Next Generation Web Services Practices (NWESP). 4th International Conference71-76.

Wang, Y. (2012). *The Study of Test Case Generation from UML Models.* University of Wisconsin.

Wei-Tek, T., Lian, Y., Feng, Z., and Paul, R. (2005). Rapid embedded system testing using verification patterns. *Software, IEEE, 22*(4), 68-75.

Wiederseiner, C., Jolly, S., Garousi, V., and Eskandar, M. (2010). An open-source tool for automated generation of black-box xunit test code and its industrial evaluation. *Testing–Practice and Research Techniques*, 118-128.

Wiessalla, J. (2010). Towards Fault-based Generation of Test Cases for Dependable Embedded Software, Austrian Institue of Technology, Austria.

Yacoub, S. M., Ammar, H. H., and Robinson, T. (1999). *Dynamic metrics for object oriented designs.* Paper presented at the Software Metrics Symposium, 1999. Proceedings. Sixth International, 50-61.

Yongfeng, Y., and Bin, L. (2009, 19-20 Dec. 2009). *A Method of Test Case Automatic Generation for Embedded Software.* Paper presented at the Information Engineering and Computer Science, 2009. ICIECS 2009. International Conference on, 1-5.

Yongfeng, Y., Zhen, L., and Bin, L. (2010, 14-15 Aug. 2010). *Real-time Embedded Software Test Case Generation Based on Time-extended EFSM: A Case Study.* Paper presented at the Information Engineering (ICIE), 2010 WASE International Conference on, 272-275.

Zander-Nowicka, J., Pérez, A. M., Schieferdecker, I., and Dai, Z. R. (2007). *Test Design Patterns for Embedded Systems,* Reseacrh and Practices.