A TEST CASE GENERATION TECHNIQUE AND PROCESS

Nicha Kosindrdecha and Jirapun Daengdej

Autonomous System Research Laboratory
Faculty of Science and Technology
Assumption University, Thailand
p4919741@au.edu, jirapun@scitech.au.edu

ABSTRACT

It has been proven that the software testing phase is one of the most critical and important phases in the software development life cycle. In general, the software testing phase takes around 40-70% of the effort, time, and cost. This area is well researched over a long period of time. Unfortunately, while many researchers have found methods of reducing time and cost during the testing process, there are still a number of important related issues that need to be researched. This paper introduces a new high level test case generation process with a requirement prioritization method to resolve the following research problems: unable to identify suitable test cases with limited resources, lack of an ability to identify critical domain requirements in the test case generation process and ignore a number of generated test cases. Also, this paper proposes a practical test case generation technique derived from use case diagram.

Index Terms - test generation, testing and quality, test case generation, test generation technique and generate tests

1. INTRODUCTION

Software testing is known as a key critical phase in the software development life cycle, which account for a large part of the development effort. A way of reducing testing effort, while ensuring its effectiveness, is to generate test cases automatically from artifacts used in the early phases of software development. Many test case generation techniques have been proposed [2], [4], [10], [11], [12], [15], [21], [22], [42], [47], [50], mainly random, path-oriented, goal-oriented and model-based approaches. Random techniques determine a set of test cases based on assumptions concerning fault distribution. Path-oriented techniques generally use control flow graph to identify paths to be covered and generate the appropriate test cases for those paths. Goal-oriented techniques identify test cases covering a selected goal such as a statement or branch, irrespective of the path taken. There are many researchers and practitioners who have been working in generating a set of test cases based on the specifications. Modeling languages are used to get the specification and generate test cases. Since Unified Modeling Language (UML) is the most widely used language, many researchers are using UML diagrams such as state diagrams, use-case diagrams and sequence diagrams to generate test cases and this has led to model-based test case generation techniques. In this paper, an approach with additional requirement prioritization step is proposed toward test cases generation from requirements captured as use cases [23], [24], [33]. A use case is the specification of interconnected sequences of actions that a system can perform, interacting with actors of the system. Use cases have become one of the favorite approaches for requirements capture. Test cases derived from use cases can ensure compliance of an application with its functional requirements. However, one difficulty is that there are a large number of functional requirements and use cases. A second research challenge is to ensure that test cases are able to preserve and identify critical domain requirements [5]. Finally, a third problem is to minimize a number of test cases while preserving an ability to reveal faults. For example, there are a lot of functional requirements in the large software development. Software test engineers may not be able to design test cases to cover important requirements and generate a minimum set of test cases. Therefore, test cases derived from large requirements or use cases are not effective in the practical large system. This paper presents an approach with additional requirement prioritization process for automated generation of abstract presentation of test purposes called test scenarios. This paper also introduces a new test case generation process to support and resolve the above research challenges. We overcome the problem of large numbers of requirements and use cases. This allows software testing engineer to prioritize critical requirements and reasonably design test cases for them. Also, this allows us to be able to identify a high percentage of each test case's critical domain coverage.

The rest of the paper is organized as follow. Section 2 discusses the comprehensive set of test case generation techniques. Section 3 proposes the outstanding research challenges that motivated this study. Section 4 introduces a new test generation process and technique. Section 5 describes an experiment, measurement metrics and results. Section 6 provides the conclusion and research directions in

the test case generation field. The last section represents all source references used in this paper.

2. LITERATURE REVIEW

Model-based techniques are popular and most researchers have proposed several techniques. One of the reasons why those model-based techniques are popular is that wrong interpretations of complex software from non-formal specification can result in incorrect implementations leading to testing them for conformance to its specification standard [43]. A major advantage of model-based V&V is that it can be easily automated, saving time and resources. Other advantages are shifting the testing activities to an earlier part of the software development process and generating test cases that are independent of any particular implementation of the design [7]. The model-based techniques are method to generate test cases from model diagrams like UML Use Case diagram [23], [24], [33], UML Sequence diagram [7] and UML State diagram [5], [43], [22], [2], [21], [15], [32], [4]. There are many researchers who investigated in generating test cases from those diagrams. The following paragraphs show examples of model-based test generation techniques that have been proposed for a long time.

Heumann [23] presented how using use cases to generate test cases can help launch the testing process early in the development lifecycle and also help with testing methodology. In a software development project, use cases define system software requirements. Use case development begins early on, so real use cases for key product functionality are available in early iterations. According to the Rational Unified Process (RUP), a use case is used to describe fully a sequence of actions performed by a system to provide an observable result of value to a person or another system using the product under development." Use cases tell the customer what to expect, the developer what to code, the technical writer what to document, and the tester what to test. He proposed three-step process to generate test cases from a fully detailed use case: (a) for each use case, generate a full set of use-case scenarios (b) for each scenario, identify at least one test case and the conditions that will make it execute and (c) for each test case, identify the data values with which to test. Ryser [24] raised the practical problems in software testing as follows: (1) Lack in planning/time and cost pressure, (2) Lacking test documentation, (3) Lacking tool support, (4) Formal language/specific testing languages required, (5) Lacking measures, measurements and data to quantify testing and evaluate test quality and (6) Insufficient test quality. They proposed their approach to resolve the above problems. Their approach is to derive test case from scenario / UML use case and state diagram. In their work, the generation of test cases is done in three processes: (a) preliminary test case definition and test preparation during scenario creation (b) test case generation from Statechart and from dependency charts and (c) test set refinement by application dependent strategies.

3. RESEARCH CHALLENGES

This section discusses the details of research issues related to test case generation techniques and research problems, which are motivated this study. Every test case generation technique has weak and strong points, as addressed in the literature survey. In general, referring to the literature review, the following lists major outstanding research challenges. The first research problem is that existing test case generation methods are lack of ability to identify domain specific requirements. The study [5] shows that domain specific requirements are some of the most critical requirements required to be captured for implementation and testing, such as constraints requirements and database specific requirements. Existing approaches ignore an ability to address domain specific requirements. Consequently, software testing engineers may ignore the critical functionality related to the critical domain specific requirements. Thus, this paper introduces an approach to priority those specific requirements and generates an effective test case. The second problem is that existing test case generation techniques aim to generate test cases which maximize cover for each scenario. Sometimes, they generate a huge number of test cases which are impossible to execute given limited time and resources. As a result, those unexecuted test cases are useless. The last problem is to unable to identify suitable test cases in case that there are limited resources (e.g. time, effort and cost). The study reveals that existing techniques aim to maximum and generate all possible test cases. This can lead to unable to select necessary test cases to be executed during software testing activities, in case that there are limited resources.

4. PROPOSED METHOD

This section presents a new high-level process to generate a set of test cases introduced by using the above comprehensive literature review and previous works [43].

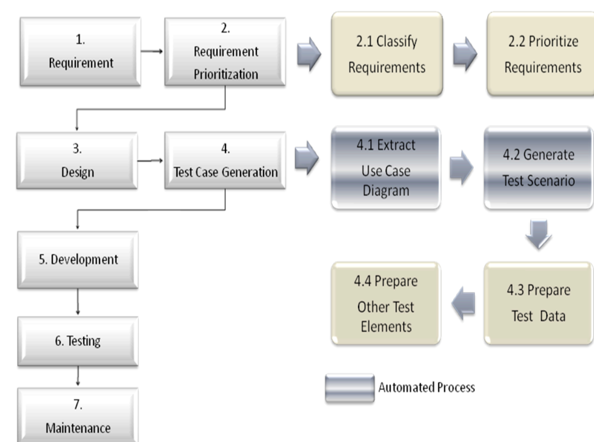

Figure 1 A Proposed Process to Generate Test Cases

From the above figure, the left-hand side process is a general waterfall process. We propose to add two additional processes: (a) requirement prioritization and (b) test case generation.

The requirement prioritization process aims to be able to effectively handle with a large number of requirements. The objective of this process is to prioritize and organize requirements in an appropriate way in order to effectively design and prepare test cases [16], [25], [37]. There are two sub-processes: (a) classify requirements and (b) prioritize requirements.

The classify requirement process primarily divides and classifies requirements into four groups [30]: (a) "Must-Have" (b) "Should-Have" (c) "Could-Have" and (d) "Wish". The "Must-Have" requirements are mandatory requirements that need to be implemented in the system. The "Should-Have" requirements are requirements that should be implemented if there are available resources. The "Could-Have" requirements are additional requirements that are able to be implemented if there are adequate resources. The "Wish" requirements are "would like to have in the future" requirements that may be ignored if there are inadequate resources. This paper introduces five factors to classify the above requirements, as follows:

Table 1 Requirement Classification

Group	Time	Cost	People	Scope	Success
Must have	Y	Y	Y	N	Y
Should have	Y	Y	Y	N	N
Could have	N	N	Y	Y	N
Wish	N	N	N	Y	N

From the above table, the following shortly describes a meaning of the above factors:

- *Time* – The requirement must be implemented in the current version or release of software.
- *Cost* – There is an available of budget or fund to implement the requirement.
- *People* – There is an available of human resources to develop and test the requirement.
- *Scope* – The requirement can be removed out of the current version or release of software.
- *Success* – The success of system development rely on the requirement.

In addition, this paper secondary divides those requirements into two groups: (a) functional and (b) non-functional. The functional requirements can be categorized into two groups: (a) domain specific requirements and (b) non- domain specific requirements. The domain specific requirements are able to identify as database specific and constraints requirements. For example, database connection specific requirements and requirements for an interface with other systems. The non-functional requirements can be vary, such as performance, security, operability and maintainability requirements. The following displays the classify requirement tree:

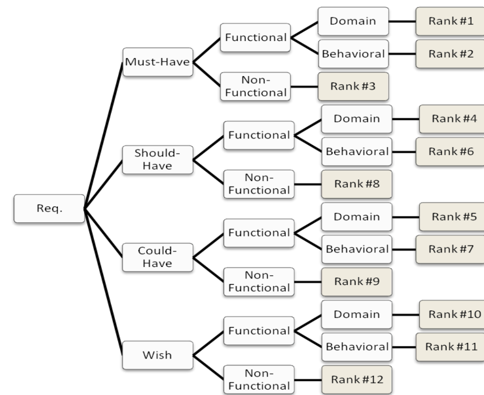

Figure 2 A Classify Requirement Tree

From the above figure, we propose a ranking number for each requirement. This paper prioritizes "Must-Have" requirements as top three ranking and "Wish" requirements as last three ranking. The study [5] reveals that domain specific requirements should have higher priority than both of behavioral and non-functional requirements.

However, when the requirement is already classified, the next process is to prioritize those requirements. In the requirement prioritization process, this paper proposes to use a cost-value approach to weight and prioritize requirements. This paper also proposes to use the following formula:

$$P(Req) = (Cost * CP) \quad (1)$$

Where:

- *P* is a prioritization value.
- *Req* is a requirement required to be prioritized.
- *Cost* is a total estimated cost of coding and testing for each requirement.
- *CP* is an user-defined customer priority value. This value is in the range between 1 and 10. 10 is the highest priority and 1 is the lowest priority. This value aims to allow customers to identify how important of each requirement is from their perspective.

To compute the above cost for coding and testing, this paper proposes to apply the following formula:

$$Cost = (ECode*CostCode)+(ETest*CostTest) \quad (2)$$

Where:

- *Cost is a total estimated cost.*
- *ECode is an estimated effort of coding for each requirement. The unit is man-hours.*
- *CostCode is a cost of coding that is charged to customers. This paper applies the cost-value approach to identify the cost of coding for each requirement group (e.g. "Must-Have", "Should-Have", "Could-Have" and "Wish"). The unit is US dollar.*
- *ETest is an estimated effort of testing for each requirement. The unit is man-hours.*
- *CostTest is a cost of testing that is charged to customers. The approach to identify this value is*

similar to CostCode's approach. The unit is US dollar.

In this paper, we assumed the following in order to calculate *CostCode* and *CostTest*. Also, this paper assumes that a standard cost for both activities is $100 per man-hours.

- *A value is 1.5 of ("Must-Have", "Should-Have") – this means that "Must-Have" requirements have one and half times cost value than "Should-Have" requirements.*
- *A value is 3 of ("Must-Have", "Could-Have") – this means that "Must-Have" requirements have three times cost value than "Could-Have" requirements.*
- *A value is 2 of ("Should-Have", "Could-Have") – this means that "Should-Have" requirements have two times cost value than "Could-Have" requirements.*
- *A value is approximately 3 of ("Could-Have", "Wish") – this means that "Could-Have" requirements have three times cost value than "Wish" requirements.*

Therefore, the procedure of requirement prioritization process can be shortly described below:

1. Provide estimated efforts of coding and testing for each requirement.
2. Assign cost value for each requirement group based on the previous requirement classification (e.g. "Must-Have", "Should-Have", "Could-Have" and "Wish").
3. Calculate a total estimated cost for coding and testing, by using the formula (2).
4. Define a customer priority for each requirement.
5. Compute a priority value for each requirement by using the formula (1).
6. Prioritize requirements based on the higher priority value.

Once the requirements are prioritized, the next proposed step is to generate test scenario and prepare test case.

This section presents an automated test scenario generation derived from UML Use Case diagram. Our approach is built based on Heumann's algorithm [23]. The limitation of our approach is to ensure that all use cases are fully dressed. The fully dressed use case is a use case with the comprehensive of information, as follows: use case name, use case number, purpose, summary, pre-condition, post-condition, actors, stakeholders, basic events, alternative events, business rules, notes, version, author and date.

The proposed method contains four steps, as follows: (a) extract use case diagram (b) generate test scenario (c) prepare test data and prepare other test elements. These steps can be shortly described as follows:

1. The first step is to extract the following information from fully dressed use cases: (a) use case number (b) purpose (c) summary (d) pre-condition (e) post-condition (f) basic event and (g) alternative events. This information is called use case scenario in this paper. The example fully dressed use cases of ATM withdraw functionality can be found as follows:

Table 2 Example Fully Dressed Use Case

Use Case Id	Use Case Name	Summary	Basic Event	Alternative Events	Business Rules
UC-001	Withdraw	To allow bank's customers to withdraw money from ATM machines anywhere in Thailand.	1. Insert Card 2. Input PIN 3. Select Withdraw 4. Select A/C Type 5. Input Balance 6. Get Money 7. Get Card	1. Select Inquiry 2. Select A/C Type 3. Check Balance	(a) Input amount <= Outstanding Balance (b) Fee charge if using different ATM machines
UC-002	Transfer	To allow users to transfer money to other banks in Thailand from all ATM machines	1. Insert Card 2. Input PIN 3. Select Transfer 4. Select bank 5. Select "To" account 6. Select A/C Type 7. Input Amount 8. Get Receipt 9. Get Card	1. Select Inquiry 2. Select A/C Type 3. Check Balance	Amount <= 50,000 baht

The above use cases can be extracted into the following use case scenarios:

Table 3 Extracted Use Case Scenarios

Scenario Id	Summary	Basic Scenario
Scenario-001	To allow bank's customers to withdraw money from ATM machines anywhere in Thailand.	1. Insert Card 2. Input PIN 3. Select Withdraw 4. Select A/C Type 5. Input Balance 6. Get Money 7. Get Card
Scenario-002	To allow bank's customers to withdraw money from ATM machines anywhere in Thailand.	1. Insert Card 2. Input PIN 3. Select Inquiry 4. Select A/C Type 5. Check Balance 6. Select Withdraw 7. Select A/C Type 8. Input Balance 9. Get Money 10. Get Card

Scenario-003	To allow users to transfer money to other banks in Thailand from all ATM machines	1. Insert Card 2. Input PIN 3. Select Transfer 4. Select bank 5. Select "To" account 6. Select A/C Type 7. Input Amount 8. Get Receipt 9. Get Card
Scenario-004	To allow users to transfer money to other banks in Thailand from all ATM machines	1. Insert Card 2. Input PIN 3. Select Inquiry 4. Select A/C Type 5. Check Balance 6. Select Transfer 7. Select bank 8. Select "To" account 9. Select A/C Type 10. Input Amount 11. Get Receipt 12. Get Card

2. The second step is to automatically generate test scenarios from the previous use case scenarios [23]. From the above table, we automatically generate the following test scenarios:

Table 4 Generated Test Scenarios

Test Scenario Id	Summary	Basic Scenario
TS-001	To allow bank's customers to withdraw money from ATM machines anywhere in Thailand.	1. Insert Card 2. Input PIN 3. Select Withdraw 4. Select A/C Type 5. Input Balance 6. Get Money 7. Get Card
TS-002	To allow bank's customers to withdraw money from ATM machines anywhere in Thailand.	1. Insert Card 2. Input PIN 3. Select Inquiry 4. Select A/C Type 5. Check Balance 6. Select Withdraw 7. Select A/C Type 8. Input Balance 9. Get Money 10. Get Card
TS-003	To allow users to transfer money to other banks in Thailand from all ATM machines	1. Insert Card 2. Input PIN 3. Select Transfer 4. Select bank 5. Select "To" account 6. Select A/C Type 7. Input Amount 8. Get Receipt 9. Get Card
TS-004	To allow users to transfer money to other banks in Thailand from all ATM machines	1. Insert Card 2. Input PIN 3. Select Inquiry 4. Select A/C Type 5. Check Balance 6. Select Transfer 7. Select bank 8. Select "To" account 9. Select A/C Type 10. Input Amount 11. Get Receipt 12. Get Card

3. The next step is to prepare test data. This step allows to manually prepare an input data for each scenarios.

The last step is to prepare other test elements, such as expected output, actual output and pass / fail status.

5. EVALUATION

The section describes the experiments design, measurement metrics and results.

5.1. Experiments Design

A comparative evaluation method has proposed in this experiment design. The high-level overview of this experiment design can be found as follows:

1. **Prepare Experiment Data**. Before evaluating the proposed methods and other methods, preparing experiment data is required. In this step, 50 requirements and 50 use case scenarios are randomly generated.
2. **Generate Test Scenario and Test Case**. A comparative evaluation method has been made among the proposed test generation algorithm, Heumann's technique Jim [23], Ryser's method [24], Nilawar's algorithm [33] and the proposed method presented in the previous section.
3. **Evaluate Results**. In this step, the comparative generation methods are executed by using 50 requirements and 50 use case scenarios. These methods are also executed for 10 times in order to find out the average percentage of critical domain requirement coverage, a size of test cases and total generation time. In total, there are 500 requirements and 500 use case scenarios executed in this experiment.

The following tables present how to randomly generate data for requirements and use case scenarios respectively.

Table 5 Generate Random Requirements

Attribute	Approach
Requirement ID	Randomly generated from the following combination: Req + *Sequence Number*. For example, Req1, Req2, Req3, …, ReqN.
Type of Requirement	Randomly selected from the following values: Functional AND Non-

	Functional.
MoSCoW Criteria	Randomly selected from the following values: Must Have (M), Should Have (S), Could Have (C) and Won't Have (W)
Is it a critical requirement (Y/N)?	Randomly selected from the following values: True (Y) and False (N)

Table 6 Generate Random Use Case Scenario

Attribute	Approach
Use case ID	Randomly generated from the following combination: uCase + *Sequence Number*. For example, $uCase_1, uCase_2, \ldots, uCase_n$.
Purpose	Randomly generated from the following combination: Pur + *Sequence Number same as Use case ID*. For example, $Pur_1, Pur_2, \ldots, Pur_n$.
Basic Scenario	Randomly generated from the following combination: uCase + *Sequence Number*. For example, $basic_1, basic_2, \ldots, basic_n$.

5.2. Measurement Metrics

The section lists the measurement metrics used in the experiment. This paper proposes to use three metrics, which are: (a) size of test cases (b) total time and (c) percentage of critical domain requirement coverage. The following describe the measurement in details.

1. **A Number of Test Cases**: This is the total number of generated test cases, expressed as a percentage, as follows:

$$\% Size = (\# Size / \# \text{ of } Total Size)*100 \quad (3)$$

Where:

- *% Size is a percentage of the number of test cases.*
- *# of Size is a number of test cases.*
- *# of Total Size is the maximum number of test cases in the experiment, which is assigned 1,000.*

2. **A Domain Specific Requirement Coverage:** This is an indicator to identify the number of requirements covered in the system, particularly critical requirements, and critical domain requirements [5]. Due to the fact that one of the goals of software testing is to verify and validate requirements covered by the system, this metric is a must. Therefore, a high percentage of critical requirement coverage is desirable.

It can be calculated using the following formula:

$$\% CRC = (\# \text{ of } Critical / \# \text{ of } Total)*100 \quad (4)$$

Where:

- *% CRC is the percentage of critical requirement coverage.*
- *# of Critical is the number of critical requirements covered.*
- *# of Total is the total number of requirements.*

3. **Total Time**: This is the total number of times the generation methods are run in the experiment. This metric is related to the time used during the testing development phase (e.g. design test scenario and produce test case). Therefore, less time is desirable.

It can be calculated using the following formula:

$$Total = PTime + CTime + RTime \quad (5)$$

Where:

- *Total* is the total amount of times consumed by running generation methods.
- *PTime* is the total amount of time consumed by preparation before generating test cases.
- *CTime* is the time to compile source code / binary code in order to execute the program.
- *RTime* is the total time to run the program under this experiment.

5.3. Results and Discussion

This section discusses an evaluation result of the above experiment. This section presents a graph that compares the above proposed method to other three existing test case generation techniques, based on the following measurements: (a) size of test cases (b) critical domain coverage and (c) total time. Those three techniques are: (a) Heumman's method (b) Ryser's work and (c) Nilawar's approach. There are two dimensions in the following graph: (a) horizontal and (b) vertical axis. The horizontal represents three measurements whereas the vertical axis represents the percentage value.

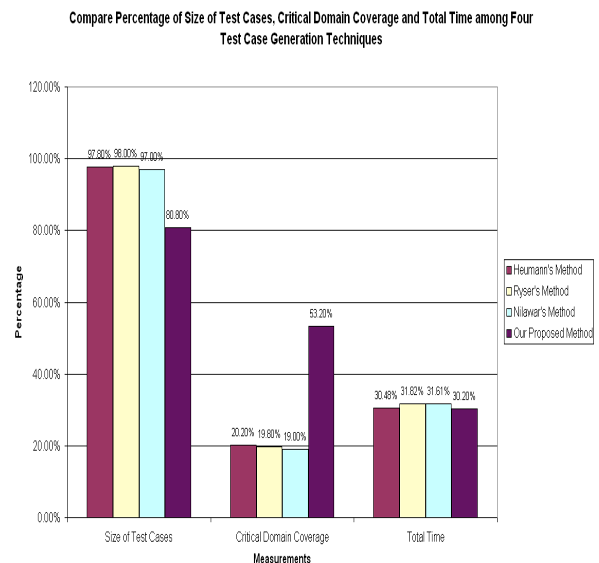

Figure 3 An Evaluation Result

The above graph shows that the above proposed method generates the smallest set of test cases. It is calculated as 80.80% where as the other techniques is computed over 97%. Those techniques generated a bigger set of test cases, than a set generated by the proposed method. The literature review reveals that the smaller set of test cases is desirable. Also, the graph shows that the proposed method consumes the least total time during a generation process, comparing to other techniques. It used only 30.20%, which is slightly less than others. Finally, the graph

presents that the proposed method is the best techniques to coverage critical domains. Its percentage is much greater than other techniques' percentage, over 30%.

6. CONCLUSION

This paper concentrates on resolving the following research problems: (a) an inefficient test case generation method with limited resources (b) a lack of ability to identify and coverage the critical domain requirements and (c) an ignorance of a size of test cases. Furthermore, this paper proposes an effective test case generation process by adding additional prioritization process. The new process aims to improve the ability to: (a) generate test cases with limited resources (b) include more critical domain specific requirements and (c) minimize a number of test cases. Also, this paper introduces an automated test scenario generation technique to address critical domain specific requirements. This paper proposes to compare to other three test case generation techniques, which are: Heummann's work, Ryser's method and Nilawar's technique. As a result, this study found that the proposed method is the most recommended method to generate the smallest size of test cases with the maximum of critical domain specific requirement coverage and the least time consumed in the test case generation process.

7. REFERENCES

[1] Ahl, V., "An Experimental Comparison of Five Prioritization Methods", Master's Thesis, School of Engineering, Blekinge Institute of Technology, Ronneby, Sweden, 2005.

[2] Alessandra Cavarra, Charles Crichton, Jim Davies, Alan Hartman, Thierry Jeron and Laurent Mounier, "Using UML for Automatic Test Generation", Oxford University Computing Laboratory, Tools and Algorithms for the Construction and Analysis of Systems, TACAS'2000, 2000.

[3] Amaral, "A.S.M.S. Test case generation of systems specified in Statecharts", M.S. thesis – Laboratory of Computing and Applied Mathematics, INPE, Brazil, 2006.

[4] Annelises A. Andrews, Jeff Offutt and Roger T. Alexander, "Testing Web Applications", *Software and Systems Modeling*, 2004.

[5] Avik Sinha, Ph.D and Dr. Carol S. Smidts, "Domain Specific Test Case Generation Using Higher Ordered Typed Languages fro Specification" Ph. D. Dissertation, 2005.

[6] A. Bertolino, "Software Testing Research and Practice", *10th International Workshop on Abstract State Machines (ASM'2003)*, Taormina, Italy, 2003.

[7] A.Z. Javed, P.A. Strooper and G.N. Watson. "Automated Generation of Test Cases Using Model-Driven Architecture", Second International Workshop on Automation of Software Test (AST'07), 2007.

[8] Beck, K. & Andres, C., "Extreme Programming Explained: Embrace Change", 2nd ed. Boston, MA: Addison-Wesley, 2004.

[9] Boehm, B. & Ross, R.. "Theory-W Software Project Management: Principles and Examples", IEEE Transactions on Software Engineering 15, 4: 902-916, 1989.

[10] B.M. Subraya, S.V. Subrahmanya, "Object driven performance testing in Web applications", in: *Proceedings of the First Asia-Pacific Conference on Quality Software (APAQS'00)*, pp. 17-26, Hong Kong, China, 2000.

[11] Chien-Hung Liu, David C. Kung, Pei Hsia and Chih-Tung Hsu, "Object-Based Data Flow Testing of Web Applications", *Proceedings of the First Asia-Pacific Conference on Quality Software* (APAQS'00), pp. 7-16, Hong Kong, China, 2000.

[12] C.H. Liu, D.C. Kung, P. Hsia, C.T. Hsu, "Structural testing of Web applications", in: *Proceedings of 11th International Symposium on Software Reliability Engineering* (ISSRE 2000), pp. 84-96, 2000.

[13] Davis, A., "The Art of Requirements Triage", IEEE Computer 36, 3 p: 42-49, 2003.

[14] Davis, A., "Just Enough Requirements Management: Where Software Development Meets Marketing", New York: Dorset House (ISBN 0-932633-64-1), 2005.

[15] David C. Kung, Chien-Hung Liu and Pei Hsia, "An Object-Oriented Web Test Model for Testing Web Applications", *In Proceedings of the First Asia-Pacific Conference on Quality Software (APAQS'00)*, page 111, Los Alamitos, CA, 2000.

[16] Donald Firesmith, "Prioritizing Requirements", *Journal of Object Technology*, Vol.3, No8, 2004.

[17] D. Harel, "On visual formalisms", Communications of the ACM, vol. 31, no. 5, pp. 514-530, 1988.

[18] D. Harel, "Statecharts: A Visual Formulation for Complex System", Sci.Comput. Program. 8(3):232-274, 1987.

[19] Flippo Ricca and Paolo Tonella, "Analysis and Testing of Web Applications", *Proc. of the 23rd International Conference on Software Engineering*, Toronto, Ontario, Canada. pp.25-34, 2001.

[20] Harel, D., "Statecharts: a visual formalism for complex system", *Science of Computer Programming*, v. 8, p. 231-274, 1987.

[21] Hassan Reza, Kirk Ogaard and Amarnath Malge, "A Model Based Testing Technique to Test Web Applications Using Statecharts", *Fifth International Conference on Information Technology*, 2008.

[22] Ibrahim K. El-Far and James A. Whittaker, "Model-based Software Testing", 2001.

[23] Jim Heumann., "Generating Test Cases From Use Cases", Rational Software, 2001.

[24] Johannes Ryser and Martin Glinz, "SCENT: A Method Employing Scenarios to Systematically Derive Test Cases for System Test", 2000.

[25] Karl E. Wiegers, "First Things First: Prioritizing Requirements", Published in Software Development, 1999.

[26] Karlsson, J., "Software Requirements Prioritizing", *Proceedings of the Second International Conference on Requirements Engineering (ICRE'96)*. Colorado Springs, CO, April 15-18, 1996. Los Alamitos, CA: IEEE Computer Society, p 110-116, 1996.

[27] Karlsson, J., "Towards a Strategy for Software Requirements Selection. Licentiate", Thesis 513, Linköping University, 1995.

[28] Karlsson, J. & Ryan, K., "A Cost-Value Approach for Prioritizing Requirements", *IEEE Software* September/October, p67-75, 1997.

[29] Leffingwell, D. & Widrig, D., "Managing Software Requirements: A Use Case Approach", 2nd ed. Boston, MA: Addison-Wesley, 2003.

[30] Leslie M. Tierstein, "Managing a Designer / 2000 Project", *NYOUG Fall'97 Conference*, 1997.

[31] L. Brim, I. Cerna, P. Varekova, and B. Zimmerova, “Component-interaction automata as a verification oriented component-based system specification”, *In: Proceedings (SAVCBS'05)*, pp. 31-38, Lisbon, Portugal, 2005.
[32] Mahnaz Shams, Diwakar Krishnamurthy and Behrouz Far, “A Model-Based Approach for Testing the Performance of Web Applications”, *Proceedings of the Third International Workshop on Software Quality Assurance (SOQUA’06)*, 2006.
[33] Manish Nilawar and Dr. Sergiu Dascalu, “A UML-Based Approach for Testing Web Applications”, Master of Science with major in Computer Science, University of Nevada, Reno, 2003.
[34] Moisiadis, F., “Prioritising Scenario Evolution”, *International Conference on Requirements Engineering (ICRE 2000)*, 2000.
[35] Moisiadis, F., “A Requirements Prioritisation Tool”, *6th Australian Workshop on Requirements Engineering (AWRE 2001)*. Sydney, Australia, 2001.
[36] M. Prasanna S.N. Sivanandam R.Venkatesan R.Sundarrajan, “A Survey on Automatic Test Case Generation”, Academic Open Internet Journal, 2005.
[37] Nancy R. Mead, “Requirements Prioritization Introduction”, Software Engineering Institute, Carnegie Mellon University, 2008.
[38] Park, J.; Port, D.; & Boehm B., “Supporting Distributed Collaborative Prioritization for Win-Win Requirements Capture and Negotiation 578-584”, *Proceedings of the International Third World Multi-conference on Systemics, Cybernetics and Informatics (SCI'99)* Vol. 2. Orlando, FL, July 31-August 4, 1999. Orlando, FL: International Institute of Informatics and Systemic (IIIS), 1999.
[39] Rajib, “Software Test Metric”, *QCON*, 2006.
[40] Robert Nilsson, Jeff Offutt and Jonas Mellin, “Test Case Generation for Mutation-based Testing of Timeliness”, 2006.
[41] Saaty, T. L., “The Analytic Hierarchy Process”, New York, NY: McGraw-Hill, 1980.
[42] Shengbo Chen, Huaikou Miao, Zhongsheng Qian, “Automatic Generating Test Cases for Testing Web Applications”, International Conference on Computational Intelligence and Security Workshops, 2007.
[43] Valdivino Santiago, Ana Silvia Martins do Amaral, N.L. Vijaykumar, Maria de Fatima, Mattiello-Francisco, Eliane Martins and Odnei Cuesta Lopes, “A Practical Approach for Automated Test Case Generation using Statecharts”, 2006.
[44] Vijaykumar, N. L.; Carvalho, S. V.; Abdurahiman, V., “On proposing Statecharts to specify performance models”, *International Transactions in Operational Research*, 9, 321-336, 2002.
[45] Wiegers, K., “E. Software Requirements”, 2nd ed. Redmond, WA: Microsoft Press, 2003.
[46] Xiaoping Jia, Hongming Liu and Lizhang Qin, “Formal Structured Specification for Web Application Testing”. *Proc. of the 2003 Midwest Software Engineering Conference (MSEC'03)*. Chicago, IL, USA. pp.88-97, 2003.
[47] Yang, J.T., Huang, J.L., Wang, F.J. and Chu, W.C., “Constructing an object-oriented architecture for Web application testing”, *Journal of Information Science and Engineering* 18, 59-84, 2002.
[48] Ye Wu and Jeff Offutt, “Modeling and Testing Web-based Applications”, 2002.
[49] Ye Wu, Jeff Offutt and Xiaochen, “Modeling and Testing of Dynamic Aspects of Web Applications, *Submitted for publication*. Technical Report ISE-TR-04-01, www.ise.gmu.edu/techreps/, 2004.
[50] Zhu, H., Hall, P., May, J., “Software Unit Test Coverage and Adequacy”, ACM Comp. Survey 29(4), pp 366~427, 1997.