A Generic Model-Based Methodology of Testing Techniques to Obtain High Quality Software

Khaled Almakadmeh
Assistant Professor
Department of Software Engineering
Hashemite University
P.O. Box 330136 Zarqa (13115), Jordan
(00962) 797676367
khaled.almakadmeh@hu.edu.jo

Fatima Abu-Zitoon
Master Student
Department of Software Engineering
Hashemite University
P.O. Box 330136 Zarqa (13115), Jordan
(00962) 53903333
fatima.dream2005@itc.hu.edu.jo

ABSTRACT
Testing techniques have been widely used as a method to help software engineers in detecting defects in a software system in order to develop high-quality software system and achieve customer satisfaction. Different techniques reveal different quality aspects of a software system. This paper proposes a model-based methodology of the major accepted categories of testing techniques to evaluate many aspects like functional, structural, reliability, usability requirements and check their consistency. Evaluating all these aspects in a software project will help ensure the success of such software development project and will also assist software testers in error handling in order to achieve the desired quality for software customers.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management – *Software Quality Assurance.*

General Terms
Measurement, Design, Standardization.

Keywords
Testing Techniques, Software Testing, Software Quality Control (SQC)

1. INTRODUCTION
Software testing is one of the main components of software quality control (SQC). This refers to have a control over the quality of software engineering products which is examined by using tests of the software system for evaluating software quality and for improving it [1].
Software testing is very important due to many reasons. First, it helps pointing out defects and errors that occur during the software development phases. Second, software testing ensures the customers' reliability and their satisfaction with the application. In addition, it ensures the quality of the product. Finally, software testing ensures an effective performance of a software product.
Software testing is a technique of multi-step strategy along with a way for producing test cases that are used to ensure effective defect detection. Moreover, it can be defined as a technique used to test the software for quality factors; such as usability, efficiency, security, reliability, maintainability, usability...etc [2]. Software testing has an important role as the final roadblock to release the software before handling it to the customer and it often consumes approximately 50% of software development costs and efforts. Therefore, software testing helps in improving the testing process and reducing costs. For this reason, is desirable to adopt a generic model of test techniques.

Testing techniques specify the strategy used in testing to select input test cases and analyze test results. In the past few decades, several test techniques have been developed and used to detect the defects and faults that exist in the software. However, not all of these techniques are accepted because of many reasons, such as inefficiency, inaccuracy or slowness. Therefore, such techniques are not acceptable because they cannot improve the software testing process and thus should not be adopted.

Given the importance of software testing in the life cycle of the program and the testing techniques' impact on the success of this process in particular as well as in the whole program in general, this paper propose a generic model of the major accepted categories of testing techniques in order to obtain high quality software system, to assist software testers in error handling, and to achieve the desired quality for the customer to ensure software success.
This paper is organized as follows: section 2 presents related work; section 3 discusses testing techniques; section 4 describes how to design a generic–based model of testing techniques; section 5 presents a generic based model to represent the methodology of testing techniques to obtain high quality. Finally, section 6 provides a conclusion of this work.

2. RELATED WORK
Many researchers have presented several testing techniques. However, a limited number of them tried to develop testing techniques to increase quality. For instance, Dianxiang Xu et al [3] presented an automated test generation technique, which called Model-based Integration and System Test Automation (MISTA). The proposed technique generates test code that can be executed immediately with the implementation under test. Furthermore, Iqbal and Qureshi [4] provided a solution for the main problems of software testing that are related to quality assurance and testing practices. In addition, they recommended some strategies to provide solution for these problems. Moreover, Reza and Lande [5] proposed a hybrid testing method to generate test cases. This method combines the benefits of model-based testing with the benefits of software architecture in a unique way.

IPAC '15, November 23-25, 2015, Batna, Algeria
 ISBN 978-1-4503-3458-7/15/11…$15.00
DOI: http://dx.doi.org/10.1145/2816839.2816903

Luo [6] presented technology maturation of testing techniques, including these functional and structural techniques that have been influential in the academic world and that have been widely used in practice. Vegas and Basili [7] presented the results of developing and evaluating a software artifact (i.e. characterization schema that helps in testing technique selection). This schema provided developers with a catalogue that contains enough information for them to select the best suited techniques for a given project. Moreover, Belli and Beyazit [8] proposed a grammar-based mutation testing framework, with effective mutation operators.

The main objective of this project is to formalize a methodology for model-based mutation testing which enables complementary and alternative use of regular grammars, and this depends on the preferences of the test. Furthermore, Zhao [9] suggested a data-flow-based unit testing approach for aspect oriented programs. This approach tests two types of units for an aspect-oriented program. In addition, it performs three levels of testing, i.e., intra-module, inter module, and intra-aspect or intra-class testing. Schaefer et al [10] proposed a software testing method called MBET (Model-Based Exploratory Testing). They conducted an empirical study to investigate the possibility of finding more effective technique for detecting defects compared to MBT (Model Based Testing). The results showed that MBET was more effective than MBT in detecting defects. Finally, Trivedi [11] identified a set of testing techniques for generating test, that assure maximum effectiveness with the least possible number of test cases. Shiyi Xu [12] proposed a new approach using the concept of fault dominance and equivalence, for reducing the number of what so called 'conditional faults' in a software system, reduce the number of faults hidden in the system so that the efficiency of mutation testing can be increased all of this in order to improve the testing techniques. Yin et al [13] presented a set of demands and challenges on how to improve the quality of software testing, starting from the basic concept of software testing, states and analyze the status quo at that time for software testing and then suggested a set of perspectives on how to improve the quality of software testing.

Catelani et al [14] proposed an approach which combines accelerated automated tests for the study of software regression and memory overflow; concerning the automated software testing as an aid to maximize the test plan coverage within the time available and also to increase software reliability and quality in use. Jang et al [15] proposed a manual testing mechanism aimed to improve the quality of embedded software; the proposed model and technique targeted testing on software and related hardware areas and the way to share the testing experience and using quality evaluation. Suma and Nair [16] provided information on various methods and practices supporting defect detection and prevention leading to thriving software generation and achieving better quality levels. The defect prevention technique unearths 99% of defects.

3. TESTING TECHNIQUES

One of the aims of testing is to detect as many defects as possible; many techniques have been developed in the past few decades. Testing technique basically helps the testers and developers to choose the best set of tests from the total number of all possible tests for a given system. There are many different types of software testing technique, each of which has its own weaknesses and strengths. Each individual technique is good at finding specific types of defects and is relatively unable to highlight others.

The Software Testing Techniques can be divided into two main types: the first type is Manual testing which is described as a slow and hard process where testing is done statically; it is also known as static testing. It is done in an early phase of life cycle of software system, and it is done by a testing team of analysts and developers. Moreover, it contains some techniques like: informal review, walk through, inspection, and technical /internal review. The second type is automated testing. In this case, the tester runs the script on the testing tool and thus testing is done. It is also called dynamic testing. It can be classified into four main types; i.e. Security testing, Performance testing, Correctness testing, and Reliability testing which measures the time used during the testing process, and this depends on these techniques to detect the defect [2].

This paper is mainly concerned with the second type of software techniques, which is called dynamic testing to enhance the basic techniques in an attempt to design a general model of test techniques. The study hopes that this model will be used as a guide for testers and developers during their implementation of this type of testing, to ensure complete this process good results. Through, detect the most or all the defects with less time, cost and effort.

Development of efficient testing techniques will help in the creation of high-quality software. Because of this strong relationship between testing and quality techniques, the next section will offer the most acceptable dynamic testing techniques, which will contribute to design a generic model of software testing techniques to achieve the desired quality.

4. DESIGNING OF MODEL BASED METHODOLOGY OF TEST TECHNIQUES

This section describes the design of accepted test techniques. To obtain high accuracy and quality for any software we need to test this software for many aspects such as functional, structural, reliability, usability requirements, and check their consistency. The classification of the model is presented based upon the way by which tests are generated: starting from the software engineer's intuition and experience, and ending up with a combined use of two or more techniques to test the software, as follows:

1. The first step is performed by a software engineer when s/he is faced with a particular software needs to test it. In this case, the software engineer relies on her/his intuition and experience to try to understand the nature of the software to initiate the testing process, using either an:
   - **Ad Hoc testing**: derived depending on the software engineer's skill, intuition, and experience.
   - **Exploratory testing**: depends on the software engineer's knowledge, which could be derived from several sources. (See figure 1).

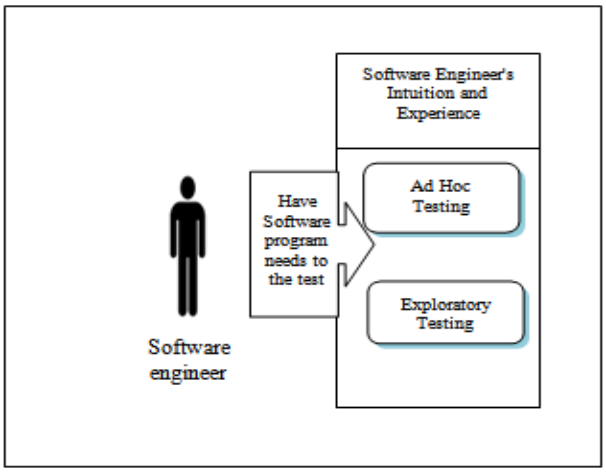

Figure 1. Exploratory testing using software engineer experience & intuition

2. The second step, tries to enter the input value to the software system after the input domain is known to check. This step used to test the functionality of the software by input value and observed the result. Its validity depends on many techniques for input domain:
   - **Equivalence Partitioning:** involves partitioning the input domain to equivalence class.
   - **Pairwise testing:** involves a pair of group of input variables instead of considering all possible combinations with a reduced set of test cases [17].
   - **Boundary-Value Analysis**: involves test cases that are selected on or near the borders.
   - **Random testing:** involves test cases that are generated randomly (See Figure 2).

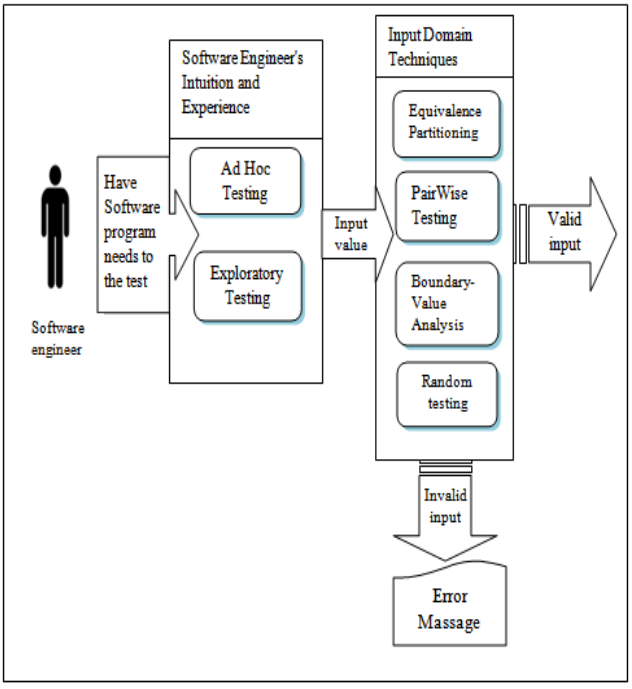

Figure 2. Input domain techniques

3. This step is Code-Based Technique which is used to test the structure of the all entry-to-exit control flow paths in a program's control flow graph. This technique divided into:
   - **Control flow-based coverage criteria:** aim at specifying combinations of statements in a program.
   - **Data flow-Based Criteria:** involve annotated of the control flow graph with more information about how the program variables are defined and used.
   - **Reference Models for Code-Based Testing:** although it is not considered as a technique by itself, the control structure of a program can be graphically represented using a flow graph to visualize code-based testing techniques [18] (See Figure 3).

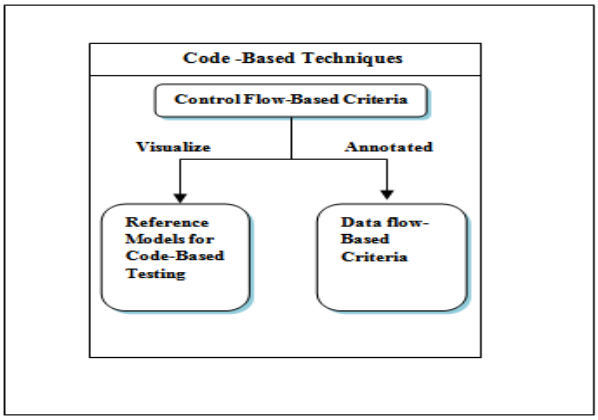

Figure 3. Code-based techniques

4. Fault-Based Techniques: aim at revealing categories of likely or predefined emits in order to increase the concentration of generation test case selection. Fault model can classify different types of defect:
   - **Error Guessing:** includes test cases which are specifically designed by software engineers.
   - **Mutation Testing:** includes a simple modified version of the program under test .(See Figure 4).

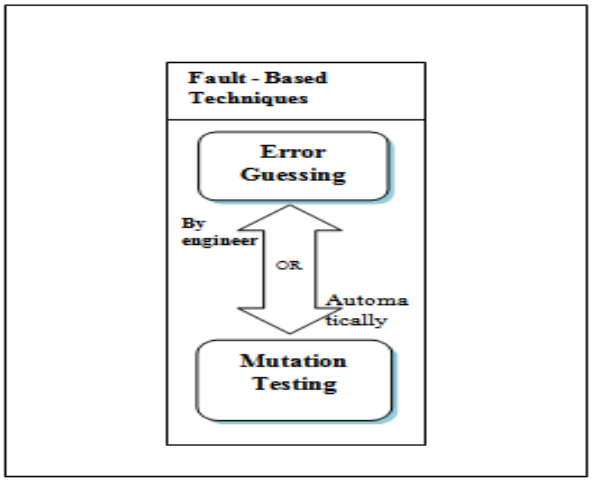

Figure 4. Fault-based techniques

5. After all of the above steps, the software engineer needs to test the environment and the system during the testing process; this technique is called Usage-Based Technique. The goal of this technique to the observed test results the future reliability of the software when in actual use. This step includes two testing types:-
   - **Operational Profile:** tests environment and reproduces the operational environment of the software.
   - **User Observation Heuristics:** involves observation of system usage under controlled conditions to determine how well people can use the system and its interfaces. (See figure 5) in the general to evaluate the usability of the software.

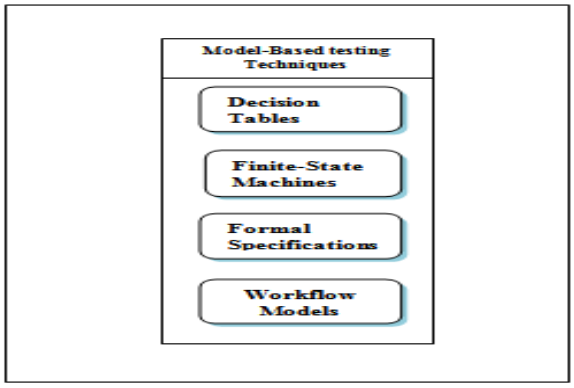

Figure 5. Usage-based technique

6. Model-Based Testing Techniques: model-based testing methods are new testing techniques designed to increase the reliability of software, and to reduce costs by automatically generating a set of test cases from a formal behavioral model of a system , but In the first place is used to validate requirements, check their consistency, and generate test cases focused on the behavioral aspects of the software which includes the following:
   - **Decision Tables:** include logical relationships between conditions (roughly, inputs) and actions.
   - **Finite-State Machines:** include the tests that can be chosen in order to cover the states and transitions.
   - **Formal Specifications:** a language designed for writing test cases, and allowing the automatic derivation of functional test cases.
   - **Workflow specification:** focuses on the roles in a targeted business process testing. (See Figure 6).

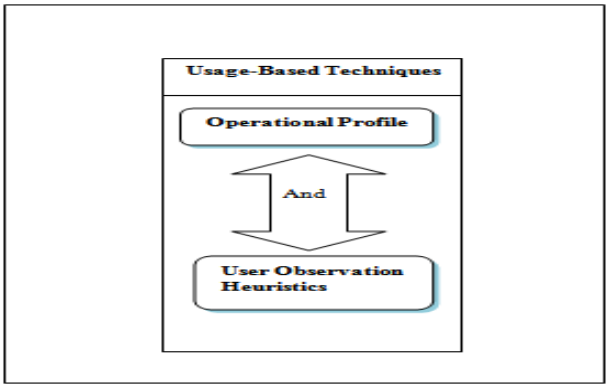

Figure 6. Model-Based testing techniques

7. Techniques Based on the nature of the application: all kinds of software could apply with the above techniques, but some types of software that need additional techniques to test derivation and execution are based on the nature of the software being tested; such as component-based software, protocol-based software, object-oriented software…etc.

8. Finally, there are many test techniques that are listed here; however, their suitability for given software must be taken into consideration. Therefore, it should be considered as the final step, which is called "selection and combining testing techniques" and which includes two kinds of Selecting and Combining Techniques:
   - **Combining Functional and Structural:** model-based and code-based techniques should not be seen as alternatives as structural vs. functional testing, but rather a complements for each other.
   - **Deterministic or Random:** test cases can be chosen in a deterministic way, according to one of many techniques or randomly (see Figure 7).

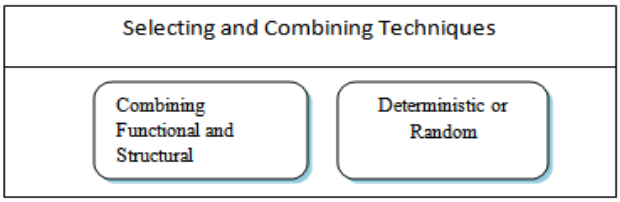

Figure 7. Selection & combining testing techniques step

Sometimes, all of the above techniques are categorized as either Black box testing or White box testing according to the information that is known and owned for the tester or developer and that is related to the software. If s/he has the ability to access the source code, it is White box testing, but if the tester or developer only knows the behavior of software, which is considered as a black box testing.

5. GENERIC MODEL-BASED METHODOLOGY OF TESTING TECHNIQUES

After examining all the previous techniques in different ways, and exploring the cases in which they are used, this section will combine all the above mentioned techniques together in one model. This model represents all the major accepted and used testing techniques. After applying this model will improve the software testing process, ensure the high-quality software since it provides roadmap for the tester and developer, especially beginners, and finally ensure the implementation of the testing process that occurs with less cost, time, and effort (See Figure 8).

6. CONCLUSION

This paper spotted the light on software testing techniques in general and on the dynamic testing techniques in particular. Software testing is an important process; it is part of the software development process to ensure that the objectives of the software are completed and reduce problems and mistakes in order to ensure customer satisfaction with

the software provided to her/him. However, the maturation of testing techniques process is significant but it is not adequate.

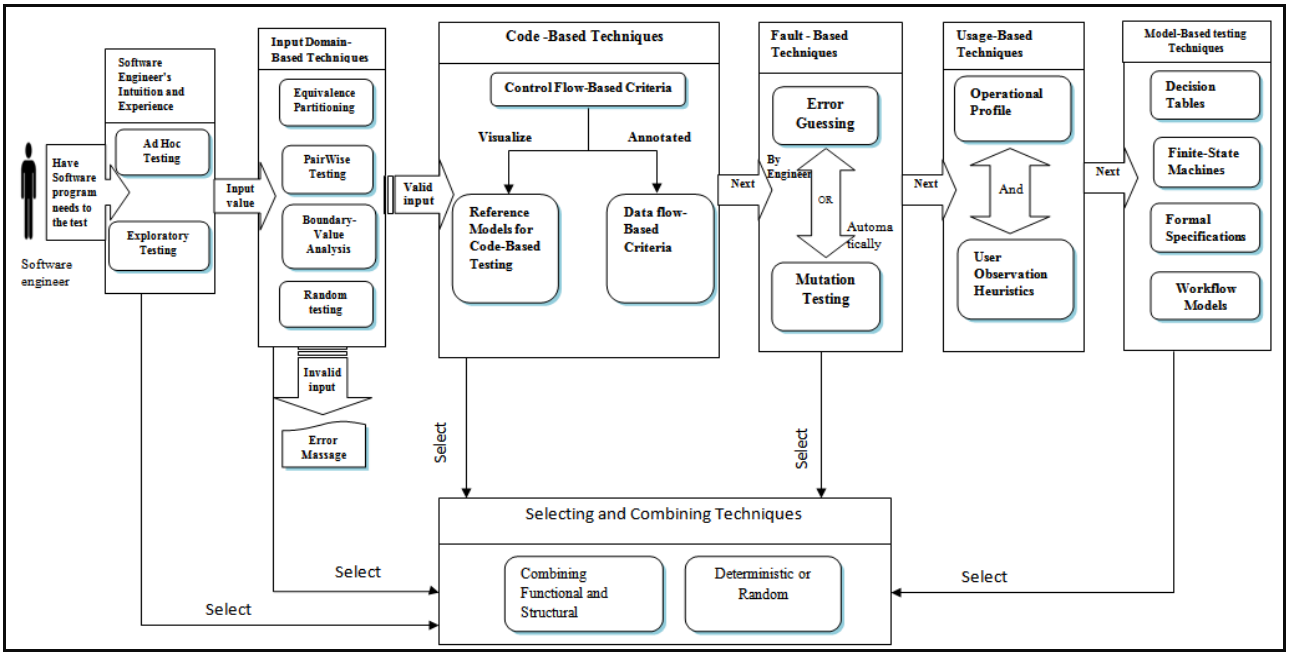

Figure 8. A Generic Model-Based Methodology of testing techniques

What is important is to produce provide a design for all acceptable techniques as a generic model. The main contribution of this paper is a generic model-based methodology of test techniques to be considered as a roadmap that helps the software engineer; tester or developers; especially the beginners and to ensure the implementation of the testing process with less cost, time, and effort. All of this would increase the effectiveness of testing process. The model-based methodology represents and classifies the test techniques based which tests are generated: starting from the software engineer's intuition and experience and ending up with a combined use of two or more techniques to test the software. Finally, the proposed model-based methodology is aimed to assist software testers in error handling and to achieve the desired quality of the customer, and thus the success of the product. Future work will be devoted to verify the applicability of the proposed model on a variety of case studies that represents different software systems.

7. REFERENCES

[1] Jovanovic, I. 2008. Software Testing Methods and Techniques. *The IPSI BgD Transactions on Internet Research*, 30-41.

[2] Sawant, A. A., Bari, P. H., & Chawan, P. M. 2012. Software Testing Techniques and Strategies. *International Journal of Engineering Research and Applications (IJERA)*, *2*(3), 980-986.

[3] Xu, D., Xu, W., Kent, M., Thomas, L., & Wang, L. 2014. An Automated Test Generation Technique for Software Quality Assurance Reliability, *IEEE Transactions on Reliability*. IEEE Reliability Society, 61 (1), 247- 268.

[4] Iqbal, N., & Qureshi, M. 2012. Improvement of Key Problems of Software Testing in Quality Assurance. *Science International*. 21(1), 25-28.

[5] Reza, H., & Lande, S. 2010. Model Based Testing Using Software Architecture. In *Proceedings of the 2010 Seventh International Conference on Information Technology: New Generations*. IEEE Computer Society, 188-193

[6] Luo L. 2001. Software testing techniques. *Institute for software research international Carnegie Mellon University, Pittsburgh, PA*, *15232*, 1-19.

[7] Vegas, S., Basili, V. 2005. A characterization schema for software testing techniques. *Empirical Software Engineering*, *10*(4), 437-466.

[8] Belli, F., Beyazit, M. 2010. A formal framework for mutation testing. *Fourth International Conference on Secure Software Integration and Reliability Improvement,* 121-130.

[9] Zhao, J. 2003. Data-flow-based unit testing of aspect-oriented programs. *Proceedings. 27th Annual International Computer Software and Applications Conference, 2003. COMPSAC 2003.* 188-197.

[10] Schaefer, C. J., Do, H. 2014. Model-Based Exploratory Testing: A Controlled Experiment. *IEEE 7th International Conference on Software Testing, Verification and Validation Workshops*, 284-293.

[11] Trivedi, S.H. 2012. Software Testing Techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*, *2*(10), 433-438.

[12] Xu, S. 2006. A new approach to improving the test effectiveness in software testing using fault collapsing. *12th Pacific Rim International Symposium on Dependable Computing*, 73-80.

[13] Yin, R., Ding, X. M. 2012. How to Improve the Quality of Software Testing. *2012 International Conference on Systems and Informatics*, 2533-2536.

[14] Catelani, M., Ciani, L., Scarano, V. L., & Bacioccola, A. 2011. Software Automated Testing: A solution to maximize the test plan coverage and to increase software reliability and quality in use. *Computer Standards & Interfaces*, 33(2), 152-158.

[15] Jang, S. J., Kim, H. G., & Chung, Y. K. 2008. Manual Specific Testing and Quality Evaluation for Embedded Software. *7th IEEE/ACIS International Conference on Computer and Information Science ICIS 08*, 502-507.

[16] Suma, V., & Nair, T. R. 2010. Effective Defect Prevention Approach in Software Process for Achieving Better Quality Levels.

[17] Monteiro, C. B., Dias, L. A. V., & Cunha, A. M. D. 2014. A Case Study on Pairwise Testing Application. *Proceedings of the 11th International Conference on Information Technology: New Generations*, 639-640, IEEE Computer Society.

[18] Bourque, P., & Fairley, R. E. 2014. *Guide to the Software Engineering Body of Knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press.