Intelligent and Automated Software Testing Methods Classification: A Review

Seyed Reza Shahamiri
Department of Software Engineering
Faculty of Computer Science and Information Systems
University Teknologi Malaysia (UTM)
+60197558500
Admin@Rshahamiri.com

Wan Mohd Nasir
Department of Software Engineering
Faculty of Computer Science and Information Systems
University Teknologi Malaysia (UTM)
+60177341003
wnasir@utm.my

ABSTRACT
Since software applications increased rapidly in modern life, it is important to have enough reliability and minimizing the probability of faults in software products. Software testing is a process to finding faults in software products and increasing their reliability. Because testing process is very costly, automation techniques are need to reduce these costs and also, increase reliability. In automatic testing, an attempt is made to reduce human roles in testing process by converting the testing phases or part of them to performed by intelligent methods. Automatic testing has several advantages such as decrease testing time, resources and costs and on the other hand, increase quality and reliability. In this paper, after explaining software testing phases, a classification of existing automatic testing methods has shown. These methods can automate software testing phases or at least part of them with aim to reach above advantages. The aim of this classification is to show which technique can applied in which phase of software testing. The classification clarifies which problems must address in each phase before moving to next problems and how to automate each phase. Moreover, approaches for automating each testing phase have recommended.

Categories and Subject Descriptors
D.2.4 [**Software Engineering**]: Software/Program Verifications – *Model checking, Reliability, Statistical methods, Validation.*

General Terms
Reliability, Verification.

Keywords
Software Testing, Intelligent Testing, Automated Testing, Software Reliability.

1. INTRODUCTION
Software is a principle component of modern life. Nowadays, it is very difficult to live without the helping of computers. The growth effects of these machines are increasing day by day in human life and they have many critical duty. Therefore, it is very important to have enough reliability and minimizing the probability of faults in software products. Software Reliability Engineering is the probability of failure-free software operation for specific period of time in a specific environment[1]. Software Testing is a process to improve software reliability by finding errors and faults in software products. Errors or faults are any repugnance between what the software expects to do and actual outputs[2].

Error detection performs by various testing process models and techniques such as unit testing, acceptance testing, load testing, and many methods such as Black-Box and White-Box testing. Each of these technique focuses on specific aspects of software testing process to find faults. For example in Black-Box testing, we investigating on software outputs accuracy only, without considering how these outputs generated. On the other hand, in White-Box testing, we completely concentrate on outputs generation process[2].

Since software testing process is a very costly process in terms of time, budget and resources, many software developers do not keep enough attention to it. Consequently, their products became very risky to fail and loosing market[3]. Therefore, we have to find approaches to decrease testing cost and also increase reliability. One approach is using methods to automating this process. Researches show that automated and intelligent testing process or at least portion of it, can significantly decreasing the test cost. In automated testing, developers attempt to convert testing process which performs by human, to perform by software with intelligent techniques and algorithms like Artificial Intelligence and Statistical methods.

Automated testing has several advantages. First, while computers are faster than humans in repetitive tasks, the process can complete sooner. Second, we can reduce human resources during testing. Third, we can test more aspects of the software under test in shorter time. Finally, by reducing human role in the process, we can prevent intentional or unintentional human faults. Consequently, testing cost can reduce, but testing quality can improve and the software product becomes more reliable.

In this paper, after explaining software testing phases, a classification of existing automatic testing methods has shown. These methods can automate software testing phases or at least part of them with aim to reach above advantages. The aim of this classification is to show which technique can applied in which phase of software testing. The classification clarifies which problems must address in each phase before moving to next

problems and how to automate each phase. The methods for automating testing phases varied between artificial intelligence and statistical methods. Figure.1 illustrates this classification. Moreover, approaches for automating each testing phase have recommended.

The remaining parts of this paper are organized as follows. Before explain which methods can use to automate software testing process, this is necessary to know what are testing phases to understand how these methods can automate the testing process. Section 2 addressed this issue. Section 3 introduces

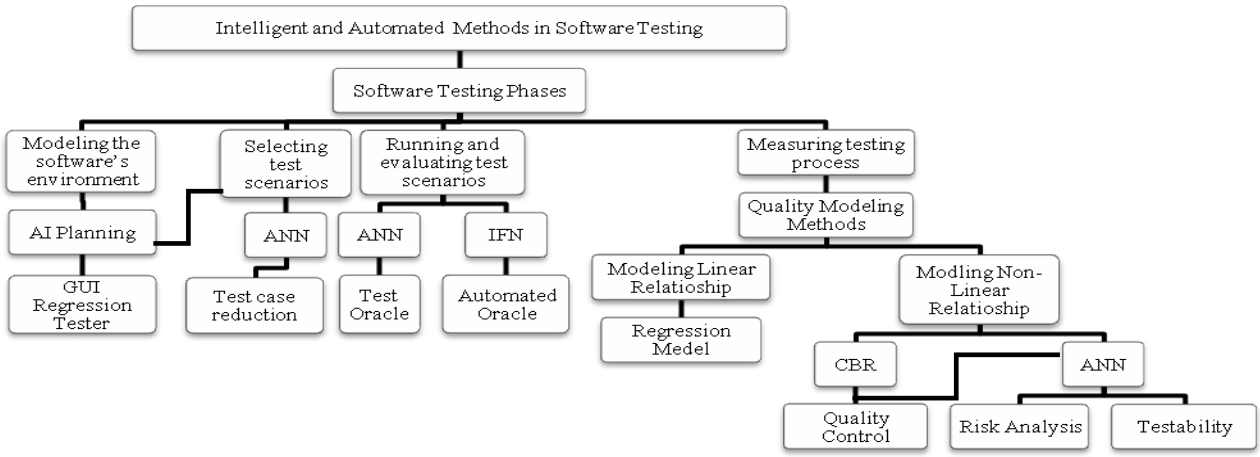

Figure 1. A classification of automated software testing methods.

applications of AI and Statistical techniques to automated software testing phases. Conclusion, constraint and future works are mentioned in section 4.

2. SOFTWARE TESTING PHASES

Based on [4] , testing process can divide into four phases which explains in following subsections. With these phases, a framework has created to depict testers must consider which problems before moving to next problem, and which phase can automate by which methods. These test automation methods will explain in section 3. Moreover, approaches for automating each phase have recommended.

2.1 Modeling the software environment

Testers must simulate relationships and interactions between software and its environment. Usually these interactions performed via interfaces such as human, software, file system and communication interfaces. Methods that can simulate the interfaces may usable for automating this phase.

2.2 Selecting test scenarios

In this phase, testers must select proper test scenarios or Test Cases that covering each line of source code, input sequences and execution paths to ensure all software modules tested adequately. Because the number of test cases can be very large to execute them all in limited testing time, this is very important to selecting test cases that have higher probability of finding errors. Approaches for automatic determination and selection of important test cases are highly beneficial.

2.3 Running and evaluating test scenarios

After preparing and selecting test cases, testers must execute them on the software under test and then, they must evaluate outputs to find if there is a fault. Testers compare the outputs generated by executed test cases (actual outputs) and the expected outputs. Expected outputs are generated by predefined software specifications and/or application logic (a testing oracle). Automation process requires a method to mapping each input to corresponding output of the entire operational environment and a tool for comparing these outputs. To put it differently, an automatic oracle is highly applicable. In section 4, a general framework of using an automated oracle is introduced.

Sometimes expected outputs are not clearly defined. This may duo to uncertainty in software behavior or lack of complete specification. Stochastic software modeling methods may use to facilitate this difficulty.

2.4 Measuring testing process

It is very important to identify what is the status of testing process and when the testing process can stop. Testers need quantitative measurement for identify the status of testing process by predicting the number of bugs in the software and the probability that any of these bugs will be discovered. Approaches for automatically predicting the number of bugs based on software specifications or previous similar projects are useful. For example, software quality estimation techniques can apply for automating this phase.

3. CLASSIFICATION OF TEST AUTOMATION METHODS IN TESTING PHASES

These methods have applied for automating a phase or at least some part of a phase in software testing process. As mentioned before, the classification has made based on applications of automated and intelligent methods in the software testing phases. In following, an attempt is made to explain such methods.

3.1 Modeling the Software Environment (Phase 1)

Nowadays, most of the software has Graphic User Interface (GUI). Modeling a GUI is a challenging task in testing process. In following, a method has explained to model the software GUI in

regression testing and used this model to derived proper test cases.

Since regression testing is a process to retest functionalities of software that remain in new versions, Regression GUI Testing is a process to reevaluate pre-tested parts of the software GUI in modified version of the software. The GUI test designer must regenerate test cases to target these common functionalities, and keeping track of such parts is an expensive and challenging process. So, usually in practice, no regression testing of GUI is performed. Many of GUI test cases from previous software testing process are unusable.

Commonly, a GUI test case contains a reachable initial state, a legal event sequence and expected states. The initial state applied to initialize the GUI to a desired state for specific test case. An expected state is the state after execution of specific event. Therefore, a modification to the GUI can affect any of these parts and lead to useless of pre-designed test cases.

The GUI regression test cases can divide into two groups: *affected* test cases and *unaffected* test cases. Affected are test cases who should rerun but duo to modifications in GUI, they must redesign. Unaffected are test cases that can execute exactly like original software GUI testing process but because they already evaluated in previous testing process, there is no need to test them again. These unaffected test cases are verified functionalities of the software GUI that do not change in the new version. As mentioned above, redesigning of affected test cases are expensive and challenging.

Memon [10] presents a method to perform GUI regression testing using AI Planner. He presents GUI test cases using *tasks* as pair of initial and goal states. These tasks remain valid in modified GUI, even changes to GUI cause test cases unusable. Each task represents a GUI's functionality. As a result, it is possible to generate affected test cases from these tasks automatically. Also, this technique uses a GUI model to automatically detect changes to the GUI and identify test cases that must rerun.

In this study a *Regression Tester* designed to determine and regenerate affected test cases. The overview of this regression tester is shown in Figure 2. One of the inputs is *Original test suits* that generated to test the original GUI. Other inputs are *representations* of original and modified GUIs. Regression Tester determined which test cases are affected, unaffected or must be discarded. Because discarded test cases verified functionalities that not further exist to modified software GUI, they must eliminate from testing process. *Test case selector* partitions the original test suits into (1) unaffected test cases, (2) obsolete tasks test cases, (3) illegal event sequence affected test cases and (4) incorrect expected states affected test cases. Illegal event sequence affected test cases are regenerated by *Planning-based test case regenerator*. But if planner failed to find a plan, the test case marks as discarded because it belongs to absolute tasks. *Expected-state regenerator* is used to regenerate expected state for incorrect expected state test cases and if it fails, test case will discard.

Consequently, this method performed regression testing based on re-planning affected test cases and associating a task with each test case and also create an interface between original and modified GUI to generate test cases. Furthermore, this method automate test case selection phase (the second phase of software testing phases) in regression GUI testing.

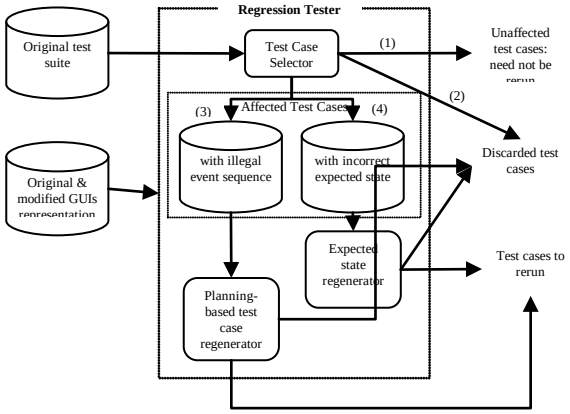

Figure 2. An overview of the Regression GUI Tester.

3.2 Selecting the Test Scenarios (Phase 2)

Test case selection is second phase in software testing process. Testers consider in effective test cases. Effective test cases can reveal the majority of software faults. According to[11], an effective test case should:

- Have a high probability of finding an error
- Not reevaluate tested sections
- Be the best of its breed
- Be neither too complex nor too simple

Each test case is defined by a set of inputs and expected output values. Basically, since the numbers of all test cases are very large in modern software, it is impossible to execute all of them in limited time and resources. Also, because many of test cases evaluate same section or part of the software, there is no need to execute all of them. Therefore, testers must wisely select effective test cases with higher probability to finding faults. Likewise, if executing a test case does not report any faults, testers must not imagine the software is fault free and reliable. In fact, in these situations, testers only waste their time.

So, this is very important to determine and select effective test cases. Automating this process can significantly decrease testing cost and increase testing quality. A good effective test case selection approach introduced in [12]. This research has revealed that program's input-output analysis can identify which input attributes mostly affect the value of specific outputs. It has shown I/O analysis can significantly reduced the number of test cases. An Artificial Neural Networks (ANN) used to automating I/O analysis by identifying important attributes and ranking them. An ANN is a mathematical modeling of human neural networks that can learn from past experience using <input, output> pairs in a training phase and generate outputs for unknown inputs based on previous data. An ANN consists of layers -each layer represented by one or more processing unit called neurons- and connections between them. ANN's can learn by adjusting connections values in the network[6].

This study modeled the software behavior using ANNs and identified which input has less effect on producing outputs by an ANN pruning algorithm. Pruning an ANN removes unnecessary connections between neurons but retaining significance ones. The removing process deletes unimportant inputs and also decreases the number of test cases. Finally, they generated test cases by remaining most significant inputs. Figure 3 depicts this process.

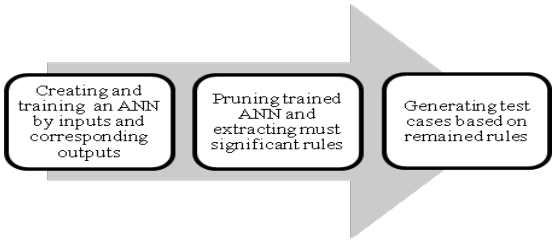

Figure 3. Automated effective test case selection and generation

3.3 Running and Evaluating Test Scenarios (Phase 3)

As mentioned in section 2, evaluating test results in third phase of software testing phases required software fault free output or accurate oracle. Testers need an approach to generate expected outputs for each input. Then, they can compare this output with the actual outputs and if these outputs are not the same, a fault is detected. This is a place which testers need automatic testing Oracle. The Oracle is a fault free source of expected outputs. Non-automatic testing oracle can be a program specification or the developer knowledge of software behavior [13]. An Oracle must accept every input specified in software specification and should always generate a correct result. The process of using automated oracle has shown in Figure 4.

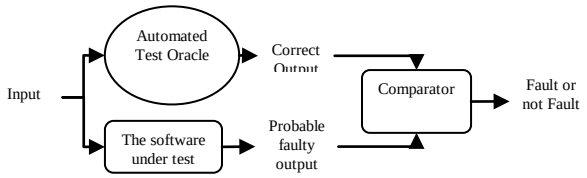

Figure 4. Using Automated Test Oracle in software testing.

Let $y = F(x)$ which $x = (x_1, x_2, ..., x_n)$ is software input vector, $y = (y_1, y_2, ..., y_m)$ is corresponding output vector and F is software behavior as a continuous function. In [14], Ye et al. modeled software behavior with modeling the relationship between the inputs and outputs (F) and developed an automatic Oracle. In this study, an ANN has used to approximate this behavior. Then, this model can use as automated Oracle for generating correct outputs. Because ANNs have a suitable capability to modeling continues deterministic functions, this method of approximation has a good accuracy if F is deterministic and without ambiguity. For situations with uncertain behavior, testers must use another approaches.

Last and his colleges [7, 15] introduced a full automated black-box regression testing method using Info Fuzzy Network (IFN). IFN is an approach developed for knowledge discovery and data mining. The interactions between the input and the target attributes of any type (discrete and continuous) are represented by an information theoretic connectionist network. An IFN represents the functional requirement by an "oblivious" tree-like structure, where each input attribute is associated with a single layer and the leaf nodes corresponds to combinations of input value[7].

The authors developed automated Oracle which can generate test cases, execute and evaluate them automatically based on previous version of the software under test. The structure of their method is shown in Figure 5. As can be seen in Figure 5, *Random Test Generator* provides test case inputs by means of *Specification of System Inputs*. These specifications contain information about system inputs such as data type and values domain. *Test Bed* executes these inputs on *Legacy Version* (Previous version of the software under test) and receives system outputs. Next, these test cases are used to train and model IFN as automated Oracle. Therefore, this Oracle can be used to detect faults in new software version. This method completely automated the third phase of software testing in regression testing.

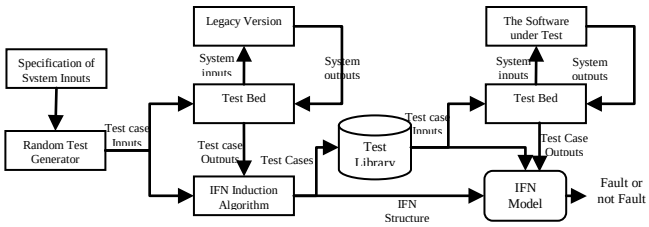

Figure 5. Using IFN for running and evaluating test cases.

3.4 Measuring Testing Process (Phase 4)

As mentioned earlier, measuring testing process is possible by a model of software quality. Software quality model has many applications in modeling the software reliability engineering. It predicts a statistical measure of software reliability and enables the testers to perform quality control and risk analysis. Quality Control can use to answer the question "When Stop Testing?" Answering this question can help testers to "Measuring Testing Process". One approach is to use software metrics.

Prior studies have shown that software metrics are correlated to number of faults. Therefore, software metrics can apply to predict the number of faults in program modules. Consequently, testers can evaluate quality level of the software under test and make a decision when to stop testing process based on previous experiences. These metrics are quantitative descriptors of modules attributes. Also, software metrics are usable to perform risk analysis. Risk analysis helps developers to identify risky module and causes to have special attention to these modules.

They are two types of methods to perform quality modeling automatically: methods that can model linear relationship between input and output patterns such as regression analysis, and methods who can model non-linear relationship such as ANN and Case-Based Reasoning (CBR). A CBR system is a computational intelligent expert system which can find solutions to a new problem based on the solution of similar past problems. This solution represented by cases in a library, based on prior experience. A CBR system consists of a case library, a solution process algorithm, a similarity function and the associated retrieval and decision rules. CBR is useful in situations where the environmental knowledge is not enough and when an optimal solution is not known. To put it differently, CBR is an automated reasoning process aimed to solve new problems[5].

Since the relationships between software metrics and quality factors are usually complex and non-linear, approaches which use former methods have better accuracy.

Khoshgoftar et al. [16] proposed a method for using ANN and Regression Modeling to predict the number of faults in program, using of software metrics, and compared results of both methods. The process is shown in Figure 6. Software metrics are used as input for trained ANN and independent variable to regression model. Also, outputs of ANN and regression model (dependent

variable) are predications of number of faults in the module under test. By comparing the prediction fault in both methods, this study has shown that ANN prediction was superior to regression model. In addition, in regression modeling, testers must manually choose which metrics related to program quality and have effects on fault prediction. On the other hand, because during the learning process of ANN effective metrics are automatically chosen by adjusting the network parameters, metrics choosing is not necessary in ANN approach.

In modern complex software systems, number of these metrics can be very large and some of them have a little influence in prediction of faults. Therefore, modeling the quality control may need a lot of processing resources and time. As a result of a study conducted in [17], using of Principle Component Analysis (PCA) is suggested to reduce the number of software metrics and deriving most important and effective metrics for modeling the quality of the software. PCA is a statistical technique for finding patterns in data of high dimension, and expressing the data in such a way as to highlight their similarities and differences. Once these patterns have found in data, PCA compress the data by reducing the number of dimensions, without much loss of information[8]. If we have an n × m matrix, we can reduced it to an n × p matrix (p<m) using PCA, by extracting linear combination of the original data. The findings of this research has revealed that using PCA has a proper prediction in both ANN and regression model.

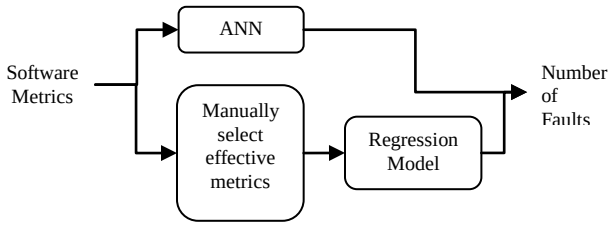

Figure 6. Copmaring two methods for predicting faults.

Another application of quality modeling is risk analysis. With risk analysis in earlier phases of SDLC[1], developers and project managers can determine error-prone modules and assigning testing resources more accurately. Moreover, because this determination has performed at earliest SDLC phases, testing and maintenance cost can reduce remarkably. An ANN based approach has recommended in [18] to classifying error-prone modules based on module attributes and quality factors, to fault-prone and not fault-prone modules. Then, developers can concentrate on designing and testing the fault-prone modules more carefully.

Similar application of ANN is in testability. Testability is the probability of test case inability to finding faults in a faulty module. To put it differently, it is the probability that a test case cannot find faults, if there are faults. Testers can use testability to find parts of the software that may hide errors. Because testability is a dynamic aspect of software attribute, this is a bit challenging to measure directly. The study of predicting testability with ANN conducted in [19]. The findings of this study have indicated that ANN modeling of static measurement in source code can predict the module testability.

Another intelligent technique in software quality modeling is CBR. Khoshgoftaar and Seliya [5] presented a three-group quality classification technique using CBR. In this research, they applied a two-group classification method three times on a given data set. By combining these three iterations, it is possible to classify modules into any of the three groups. A two-group risk analysis classifier divides modules into low-risk and high-risk, but a three-group classifier divides modules into low, medium and high risk modules. Figure 7 explains the process of a two-group risk classifier.

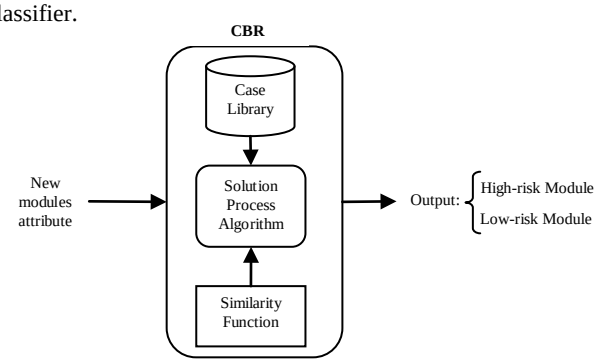

Figure 7. CBR two-group risk classifier.

The case library includes previous project data. This data has been collecting form prior similar systems. In this case, the case library contains software quality factors and corresponding risk class. The risk class determined with the number of faults detected in related software modules. Similarity function measures how much new case relates to those on the library. A solution process algorithm decides which case in the library similar to this new case, based on similarity function measurement. At the end, most relevant cases fetch from the library and used to identify risk class of new case.

This is possible to perform a two-group classification by any method mentioned above. But, using a CBR as classifier has several advantageous. As an illustration, users can understand each solution has derived with a reasonable way. Therefore, unlike ANNs, CBR does not treat as Black-boxes.

In addition to these methods, there are other techniques which researchers are using for software quality modeling [5] such as decision trees, logistic regression, optimized set reduction, fuzzy logic and genetic programming. The methods that discussed above are must effective and prevalent.

4. CONCLUSION

In this chapter, we have explained software testing process and divided it into four activities. Each activity shows which software testing problem must address before others. Then, we have introduced how to automate each of them and showed approaches to automate them. At least one automated approach has mentioned for automating whole or part of the each activity. Moreover, a classification of existing automated and intelligent methods in software testing has performed. This classification has made based on approaches that mentioned for automating the software testing phases. Using these methods may decrease testing cost while increase the testing quality. These methods are varied between AI methods like ANNs, CBR, IFN and AI planning, or statistical methods such as Regression Modeling and PCA. Some

[1] Software Development Life Cycle: The process to developing a software-based system.

of these methods can applied in any type of test and some of them in special tests like regression testing.

Each of these methods has limitations based on the tools they used. For example, ANN models of software cannot be accurate enough if software behavior is non deterministic; or IFN model can applied if application is data oriented. Furthermore, testers must consider overhead costs, extra knowledge and specialist needed for using such techniques. On the other hand, by comparing costs of use and not use of these methods, it is possible to find out automatic approaches have positive affects in reducing testing cost and increasing software quality.

Finally, because each method has influenced in special type of test, elimination of human role in testing process needs more study. Fully automation of testing process required more comprehensive methods which applicable in any type of test. Consequently, more researches are necessary in order to automate the whole testing process.

5. REFERENCES

[1] Su, Y.-S. and Huang, C.-Y. Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models. Journal of Systems and Software, 80, 4 2007), 606-615.Ding, W. and Marchionini, G. 1997 A Study on Video Browsing Strategies. Technical Report. University of Maryland at College Park.

[2] Myers, G. J. The Art of Software Testing, Second Edition. Wiley, 2004.

[3] Pressman, R. J. Software Engineering :A Practitioners Approach . Sixth Edition. McGraw-Hill 2005.

[4] Whittaker, J. A. What is software testing? And why is it so hard? Software, IEEE, 17, 1 2000), 70-79.

[5] Khoshgoftaar, T. M. and Seliya, N. Three-Group software quality classification modeling using an Automated Reasoning approach. World Scientific, City, 2004.

[6] Fausett , L. Fundamentals of Neural Networks: Architecture , Algorithms and Applications. Prentice Hall 1994.

[7] Last, M. and Freidman, M. Black-Box Testing with Info-Fuzzy Networks. World Scientific, City, 2004.

[8] Smith, L. I. A tutorial on Principal Components Analysis. City, 2002.

[9] Stockburger, D. W. Regression Models. Atomic Dog Publishing, City, 2001.

[10] Memon , A. M. Automated GUI Regression Testing using AI Planning. World Scientific, City, 2004.

[11] Saraph, P., Last, M. and Kandell, A. Test case generation and reduction by automated input-output analysis. Institute of Electrical and Electronics Engineers Inc., City, 2003.

[12] Saraph, P., Kandel, A. and Last, M. Test Case Generation and Reduction with Artificial Neural Networks. World Scientific, City, 2004.

[13] Aggarwal , K. K., Singh, Y., Kaur , A. and Sangwan , O. P. A Neural Net based Approach to Test Oracle. ACM Software Engineering Notes2004).

[14] Ye, M., Feng, B., Zhu, L. and Lin, Y. Neural networks based automated test oracle for software testing. Springer Verlag, Heidelberg, D-69121, Germany, City, 2006.

[15] Last, M., Friendman, M. and Kandel, A. Using data mining for automated software testing. International Journal of Software Engineering and Knowledge Engineering, 14, 4 2004), 369-393.

[16] Khoshgoftaar, T. M., Pandya, A. S. and More, H. B. A neural network approach for predicting software development faults. City, 1992.

[17] Khoshgoftaar, T. M., Szabo, R. M. and Guasti, P. J. Exploring the behavior of neural network software quality models. Software Engineering Journal, 10, 3 1995, 89-96.

[18] Khoshgoftaar, T. M., Allen, E. B., Hudepohl, J. P. and Aud, S. J. A. A. S. J. Application of neural networks to software quality modeling of a very large telecommunications system. Neural Networks, IEEE Transactions on, 8, 4 1997, 902-909.

[19] Khoshgoftaar, T. M., Allen, E. B. and Xu, Z. Predicting testability of program modules using a neural network. City, 2000.