THE INTERNATIONAL JOURNAL OF
SCIENCE & TECHNOLEDGE

Automated Software Testing for Reliable System Development

Nwandu, Ikenna C.
Research Student, Department of Computer Science,
Federal University of Technology Owerri, Nigeria
Asagba, Prince O.
Professor, Department of Computer Science, University of Port Harcourt, Nigeria

Abstract:
This paper presents an automated model that uses different metrics to estimate the reliability of software systems. Normally, the reliability of software systems is focused on engineering techniques for developing and maintaining software systems whose quality is obviously felt through the system performance and automated testing systems serve as essential tools for the sustenance of the needed reliability in software development. The proposed model was designed using software testing principles and object oriented analysis and methodology. This was aimed at maintaining one important aspect of ensuring reliability - being able to measure and predict the system's reliability accurately. The model estimates the reliability of given software systems using the context of execution speed and accuracy, exposing the beneficial effect of reliability assessment such as early fault detection and customer satisfaction.

Keywords: *Software, testing, reliability, automation, metrics, failure, fault, system development*

1. Introduction

Computer industry has been delivering impressive improvement in price performance, but the problems with software have not been decreasing. Software still come late, exceed budget and are full of residual faults. A major problem of software industry is its inability to develop bug free software. One can therefore boldly say that if software developers are asked to certify that the developed software is bug free, no software would have ever been released. However, software has become integral part of most of the fields of human life. In fact there is no named field without finding the usage of software in that field. This emphasises the need for software of high quality. Ordinarily, software does not wear out; they are written codes that can last a test of time. That is why some developers often assign perfect reliability to a software component. However, software's mode of failure is based on the assumption that design and development are not perfect processes. This means that the mistakes made during these processes manifest as faults in the code, which are revealed as inputs are processed. The failures occur when the software does not perform according to specification for an input history. The above mentioned issue triggered of the reason why software reliability engineering is centred on a key attribute, software reliability. Among other attributes of software quality such as functionality, usability, capability, and maintainability, etc., software reliability is generally accepted as the major factor in software quality since it quantifies software failures, which can make a powerful system inoperative or malfunction.

2. Literature Review

2.1. Software Reliability

Software reliability can be seen as the probability of the failure free software operation for a specified period of time in a specified environment (Eduardo, 2006). Software reliability therefore is a probability that software will not cause the failure of a system for a specified time under specified conditions. Reliability is a probabilistic measure that assumes that the occurrence of failure of software is a random phenomenon. Randomness means that the failure can't be predicted accurately. The randomness of the failure occurrence is necessary for reliability modelling. Based on these descriptions, the failure behaviour of software can be seen to be a function that depends on:

i. Number of faults in the software and
ii. Execution profile (Eduardo, 2006).

The unreliability of any product comes due to the failures or presence of faults in the system. As software does not "wear-out" or "age" (as a mechanical or an electronic system does), the unreliability of software is primarily due to bugs or design faults in the software (Quyoum et al, 2010). Software failures may be due to:

i. errors,
ii. ambiguities,
iii. oversights or misinterpretation of the specification which the software is supposed to satisfy,
iv. carelessness or incompetence in writing code,
v. inadequate testing,
vi. incorrect or unexpected usage of software or other unforeseen problems (Quyoum et al, 2010).

2.2. Software Testing

Software testing is defined as the systematic execution of the software system with the aim of revealing failures (Bo Zhou et al, 2007). Broadly speaking, it is "the process of executing a software system to determine whether it matches its specification and executes in its intended environment" (Whittaker, 2000). Software testing is an important phase to validate the correctness and/or reliability of software system. In general, since we cannot validate all of the execution paths, software testing is performed with limited test cases. Thus the quality of test cases directly affects the quality or reliability of software products, and one of the quality attributes of test cases is the ability of detecting as-yet-undiscovered failures (Bo Zhou et al, 2007).

2.3. Related Work

(Hosain, 2005) quantified software reliability by modelling the software use as a meaningful random process in which a use is selected according to some probability distribution, or use distribution. He presented reliability then as the probability that the software will perform according to specification for a randomly selected use and that when the software fails to meet specification during use, a failure occurs. (Gokhale et al, 2006) modelled a unifying framework for state-based models. They applied the state-based models to architecture-based software reliability prediction. According to their work, State-based models used the control graph to represent software architecture, and predicted reliability analytically. They outlined the information necessary for the specification of the state-based models of the software to be able to predict reliability of software. They went further to propose a systematic classification scheme for state-based approaches to reliability prediction. The classification scheme considered three aspects while categorizing the models. They are:

i. The model used to represent the architecture of the application,
ii. The model used to represent the failure behaviour of its components, and
iii. The solution method.

Depending on the software artefacts available during a given phase of the software life cycle, and the parameters estimated from these artefacts, (Gokhale et al, 2006) provided guidance regarding which model may be appropriate to predict the reliability of an application during each phase of the software life cycle. (Cheung et al, 2007) did a work on software reliability modelling and identified several sources of uncertainties, and illustrated how to incorporate them into reliability modelling framework. The presence of uncertainties, according to them, due to the lack of information about a software system, was a major challenge to any architectural-level reliability modelling technique. They paid much attention to formulating modalities that could solve that problem. They also discussed and evaluated how well their proposed component reliability prediction framework could address these uncertainties. (Quyoum et al, 2010) carried out a work on the randomness of the failure occurrence in software systems. They viewed reliability as a probabilistic measure that assumes that the occurrence of failure of software is a random phenomenon. Randomness, according to them, means that the failure cannot be predicted accurately. Hence, it becomes necessary for reliability modelling. Their conclusion was that the exact value of software product reliability however is never precisely known at any point in its lifetime. (Isitan, 2011) in his effort to ascertain the possibility of establishing a software reliability model for free and open source software development, stated that a software product and the developed software can be called reliable if the end product can satisfy the requirements of metrics subject to criteria such as reliability, maintainability, and security as success indicators. He concentrated on the prospects of open source software reliability by following a unified model since he concluded that an open source software project can be seen to have failed when another project uses its code and advances the development of the software. (Bo Zhou et al, 2013) carried out a work on reliability evaluation and discovered that it is an important phase to validate the correctness of software system via the following testing activities:

i. making test cases and
ii. validating the behaviour of software system by executing the test cases.

(Bo Zhou et al, 2013) further presented the quality of test cases to have direct effect on the quality or reliability of software products, and one of the quality attributes of test cases is the ability of detecting as-yet-undiscovered failures. (Bindal, 2013) investigated how Markov Mode of software is drawn, its application and how it is used for estimating the reliability of software. He maintained that the main objective of the reliability testing is to evaluate the performance of the software under given conditions without any type of corrective measure with known fixed procedures. According to Bindal, other objectives of reliability testing include:

i. To find perceptual structure of repeating failures.
ii. To find the number of failures occurring in specified amount of time.
iii. To find the mean life of the software.
iv. To know the main cause of failure.
v. After taking preventive actions checking the performance of different units of software.

According to (Wang, 2014), in order to enhance the dependability of computing software system, an effective evaluation of their reliability is desired. He presented methods for evaluating software reliability, and indicated that stochastic modelling has provided an effective and unified framework for analysing various aspects of reliability. His work was devoted to combinatorial methods, Markov models, and software models as well as important techniques based on the theory of stochastic processes.

3. Methodology

The proposed automated model which assesses the reliability of different software systems was designed using object-oriented system analysis and methodology. The model tests the quality of a given number of software systems in terms of speed and accuracy. It uses a range of test cases depending on the number of functional units in the software. The system records "successful" if the software system runs within a stipulated time interval. Otherwise, it returns "failed". Different metrics then utilizes the output to evaluate the reliability of the given software. The design tools (artefacts) used in the development of the model are:

i. Use Case Diagram
ii. Data Flow Diagram.

Figure 1 gives the detailed architecture of the automated model.

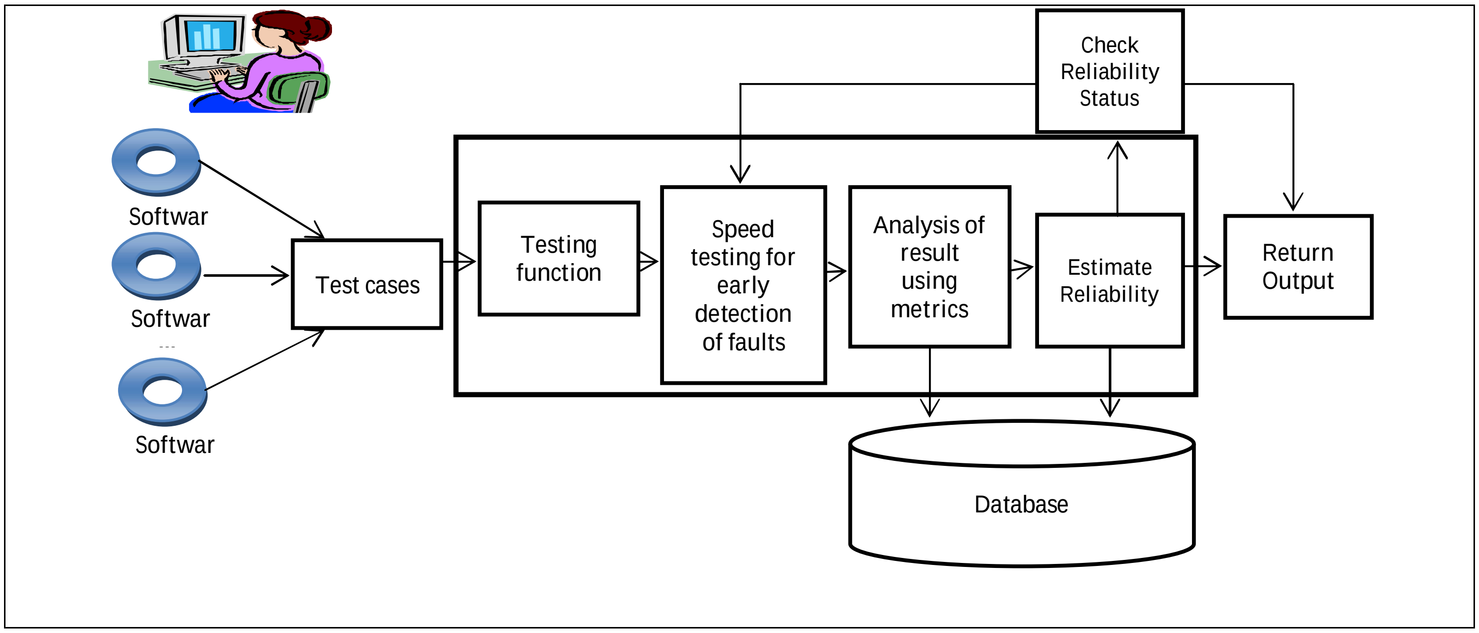

Figure 1: Detailed Architecture of the Model

3.1. System Model Assumptions

The model assumes that the time instants at which the software executes are randomly generated and that fault counts for each of the testing intervals are available.
It also assumes that the middle value between the time intervals represents the speed (in seconds) of software execution.
The smaller the values indicates higher tendency of failure occurrence.

3.2. System Model Performance Measures

The model generates a number of test units depending on the duration of testing indicated by the tester/user. At a higher speed, the model returns "successful" status but returns "failed" status at very low unacceptable execution rate. The time of each execution is recorded in the database. The time of each failure is also recorded in the database and the intervals between the failure times are used to determine the Mean Time to Failure (MTTF) as follows:

$$\text{MTTF} = \sum_{i=1}^{n_e} \frac{t_{i+1} - t_i}{n_e - 1}$$

where n_e is the number of failures and
t_i is the time instants at which failure occurs.

Other metrics used by the model include

i. Mean Time to Repair (MTTR) which is given by a value that is randomly selected from a range of generated numbers.
ii. Mean Time Between Failures (MTBF) = MTTF + MTTR
iii. Availability = $\frac{\mathrm{MTTF}}{MTBF} * 100$

The model further utilizes all the metrics calculations to estimate the reliability of the software that is being tested. Reliability (R) is given by

$$R = 1 - \frac{n_e}{n}$$

where n is the total number of software executions.

3.3. Use Case Diagram of the Automated Model

Figure 2 shows the use case diagram of the automated reliability estimation model.

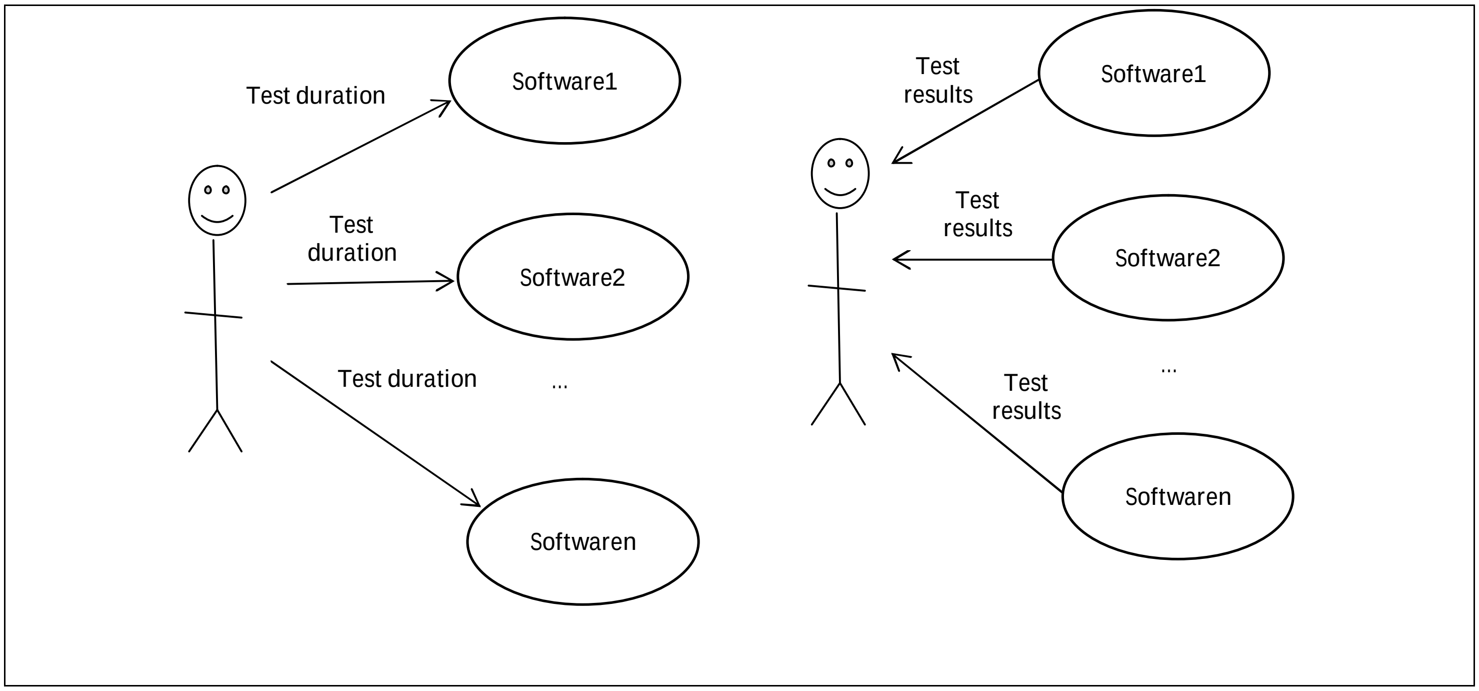

Figure 2: Use Case Diagram of the Model

3.4. Data Flow Diagram of the Automated Model

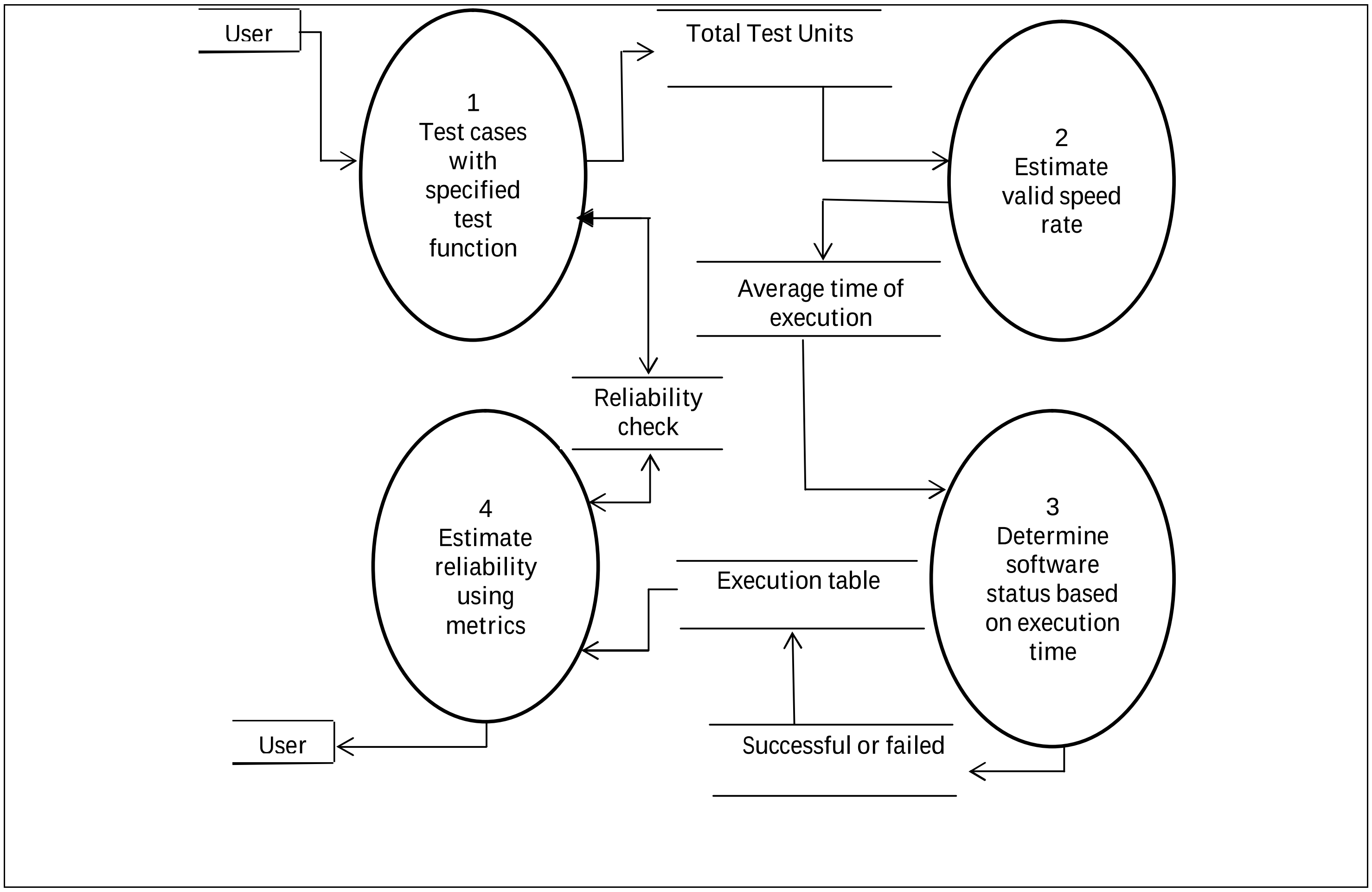
User
1
Test cases with specified test function
Total Test Units
2
Estimate valid speed rate
Average time of execution
Reliability check
4
Estimate reliability using metrics
3
Determine software status based on execution time
Execution table
User
Successful or failed

Figure 3: Data Flow Diagram of the Proposed Model

The diagram shown in Figure 3 represents the flow of data in the automated software reliability estimation model.

4. Benefits of the Automated Model

i. The model is simple enough and provides inexpensive approach to collection of required data.
ii. The model estimates, with satisfactory accuracy, quantities needed by software developers and testers in evaluating the reliability of given software.
iii. The model is useful across different software products and different development environments.

4.1. Other Benefits of the Model Include

- Simplicity of use.
- Clarity of results.
- Developers can quickly generate new reliability estimates for updated code to see which metric ratios need improvement.

5. Simulation and Discussion

5.1. Simulation

This paper carried out a simulation test for ArcPetro, EctroPet and MegaPC software. Their results were plotted in figure 4. Their mean time to failure (MTTF), mean time to repair (MTTR) and mean time between failure (MTBF) were compared in figure 5.

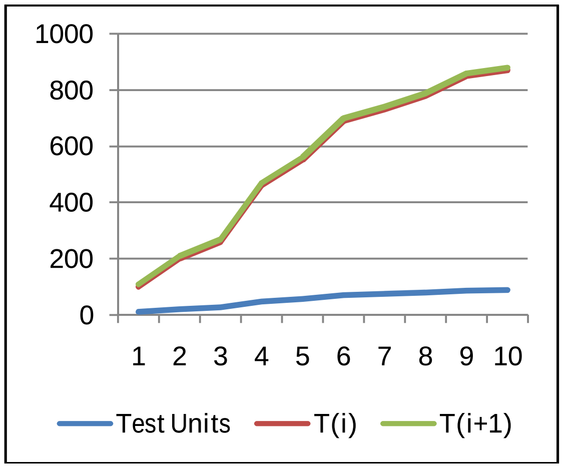

Figure 4: Plot of TestUnits versus failed time intervals

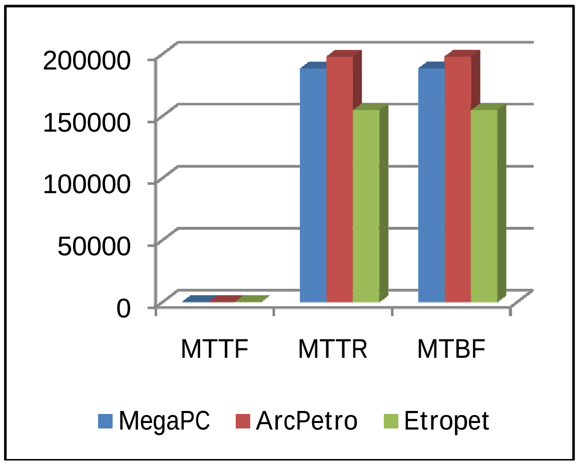

Figure 5: Comparison of MTTF, MTTR and MTBR

5.2. Discussion

The graph of TestUnits versus execution times of the cases that resulted to failure(T(i) and T(i+1))indicates that as more units are being tested, the likelihood of meeting more failures increases progressively in a manner that yields an ascending trajectory. The comparison of the metrics – MTTF, MTTR and MTBR is a clear indication that failure in the systems are not closely envisaged but should be meticulously avoided because the repairs may take a long period of time.

6. Conclusion

In this paper, we presented a model that implement testing processes to realise a system that evaluates the reliability level of software system. This is important because the increase in number of software failures had enormously affected many life enhancement activities and the problem of how to conduct a proper and effective evaluation of the reliability of software systems before their release continues to need more attention. Hence, software reliability is of utmost importance in order to provide a useful measure for giving confidence to the user about software correctness. The implementation of the model will go a long way in establishing the true quality of the software that is being tested. It will also influence user dependency and trust on the software performance.

7. Future Work

One direction of future work recommended in this area of research is to design a model that tests the reliability of software systems in three or more domains since we only focused on speed and accuracy.
Future research work can also take the direction of integration mapping of reliability estimation with other software quality attributes.

8. References

i. Bindal, D. (2013). A Review of Markov Model for Estimating Software Reliability, International Journal of Advanced Research in Computer Science and Software Engineering, Kurukshetra University, Haryana, India, 3(6):426-433.
ii. Bo-Zhou, B., Okamura, H. and Dohi, T. (2013). Enhancing Performance of Random Testing Through Markov Chain Monte Carlo Methods, IEEE Transactions on Computers, University of California Riverside, 62(1):1-4.
iii. Cheung, L., Golubchik, L., Medvidovic, N. and Sukhatme, G. (2007). Identifying and Addressing Uncertainty in Architecture-Level Software Reliability Modeling, IEEE, 423-432.
iv. Eduardo, V. C. (2006). Software reliability methods, Technical University of Madrid, Spain, 3, 8-10.
v. Gokhale, S. S. and Trivedi, K. S. (2006). Analytical Models for Architecture-Based Software Reliability Prediction: A Unification Framework, IEEE Transactions on Reliability, 55(4).
vi. Hossain, M. S. (2005). Software Reliability Using Markov Chain Usage Model, Bangladesh University, Dhaka, 9,34-42.
vii. Isitan, K. K. (2011). An Approach to Establish a Software Reliability Model for Different Free and Open Source Software Development Paradigms, Tampere University, Tampere, 6-8.
viii. Lyu, M. R. (2014). Software Reliability Engineering: A Roadmap, Chinese University of Hong Kong, Hong Kong, 2-6.
ix. Quyoum, A., Dar, M. and Quadri, S. M. K. (2010). Improving Software Reliability Using Software Engineering Approach – A Review, International Journal of Computer Applications, Volume 10 No. 5, University of Kashmir. India, 41-43.
x. Wang, R. T. (2014). Reliability Evaluation Techniques, Department of Statistics, Tunghai University, Taichung, Taiwan, 31-69.
xi. Whittaker, A. J. (2000). What is Software Testing and Why It is So Hard? Practice Tutorial, Florida Institute of Technology, IEEE, 36-42.