International Journal of Hybrid Information Technology
Vol. 9, No.9 (2016), pp. 77-84
http://dx.doi.org/10.14257/ijhit.2016.9.9.08

Diagnosis of Software using Testing Time and Testing Coverage

Amol K. Kadam[1], S.D. Joshi[1], Debnath Bhattacharyya[2] and Hye-Jin Kim[3]

[1]*Bharati Vidyapeeth University College of Engineering, Pune*
akkadam@bvucoep.edu.in, sdj@live.in
[2]*Department of Computer Science and Engineering,*
Vignan's Institute of Information Technology,
Visakhapatnam-530049, India
debnathb@gmail.com
[3]*Sungshin Women's University,*
2, Bomun-ro 34da-gil,
Seongbuk-gu, Seoul, Korea
hyejinaa@daum.net
(Corresponding Author)

Abstract

Software reliability testing plays a vital role to identify many flaws in software design as well as functional aspects. Reliability testing encompasses the analysis of the software's capacity to carry out intended tasks, particular environmental conditions within the given time instance. So, reliability estimation should be carried out within the initial stage of the software development. Through this paper we have shown the importance of the testing time and testing coverage in the analysis of software reliability. Testing time is the time interim required to carry out testing mechanism. Testing coverage includes the amount of tests exercised by the test set or batch. In order to enhance the efficiency of the proposed system we have applied the non-homogeneous Poisson process.

Keywords*: Testing time, Testing coverage, Poisson process, Software reliability growth model*

1. Introduction

Software industry plays an important role in the economical growth of the country. Software development is carried out applying various development models. Software testing is the one of the umbrella activity. But only testing is unable to remove all the flaws within the software because testing mechanism is suffered from the imperfect debugging and error generation. Imperfect debugging means the error or flaws stays within the software as it is also after the application of testing mechanism. And bug or fault removal procedure induces the new fault within the system. This event is known as error generation. So have to apply the reliability growth model within the testing stage in order to make testing mechanism more effective. Application of reliability growth model increases the quality of the software. Development of reliable software has become an engineering discipline rather than an art. It is a difficult task to develop totally bug or defect free software. All program modules should be tested until the achievement of high superiority. Entire removal of each and every fault in huge software systems is impractical. Software should released at some point in specified time interim otherwise additional delay will definitely be the reason of unacceptable loss of profits plus market share.

Huge software needs the usual improvement (upgrading) in order to recover the faults in previous versions, include new function to compete with new requirements. So, its

ISSN: 1738-9968 IJHIT
Copyright © 2016 SERSC

necessary to find out the exact release time of the software based *i.e*. intent of the software testing process. Jianfeng Yang *et al*. provides the approach for identifying the exact time to release the software by evaluating the optimal release times rely on cost-efficiency[13]. Mengmeng Zhu[14] *et al*. analyze the various environmental factors which affects the software reliability and gives the algebraic analytical method for the estimation of these factors. Wasif Afzal *et al*.[15] suggests the several software testing mechanism improvement procedures for these purpose they have selected following two methodologies:

1. TPI NEXT
2. TMMi

Based on this analysis, they have estimated the various results and accordingly improve the process.

Test coverage is an important aspect while estimating the superiority of the software under the test and it also provides the guideline to determine when to terminate the testing mechanism. So some researchers have included the test coverage within the reliability growth models. Dalal Alrmuny gives the comparison of such models which encompasses the test coverage within reliability growth model[16]. Code coverage is an appraisal utilized to illustrate the extent to which the source code of a program is examined or tested by means of the particular test suite. A program which has high code coverage has been more comprehensively tested plus has a minor likelihood of including software defects or bugs than a program which has low code coverage. Coverage information that is collected while testing is utilizes to consider the effectual area of the test data. So, application of code coverage in order to regulate the failure rate before SRGM application on software enhances the reliability to great extent [5][6]. Test time is also the important factor because testing mechanism should be completed within the specified time. Mei-Hwa Chen et. Al[03] proposes a novel technique which involves the use of both time and coverage measures in order to predict the software failure. Their technique utilizes coverage information or data assembled while testing to extort only effectual data within the specified operational profile.

Shuanqi Wang et al.[04] proposes the new software reliability growth models(SRGMs) using following two aspects:

1. Failure data
2. test coverage

By using these aspects they tried to include the consequence of the test coverage. Applying SRGMs in practice is somewhat complex due to complicated manual calculations Bijoyeta Roy *et al*. put the concentration on S shaped SRGM utilizing a flexible modelling approach which substitutes the traditional manual methods with computerized techniques [11]. Shuanqi Wang *et al*. presents an accelerated testing methodology relies on test coverage for enhancing the competence and effect of software reliability testing[8]. The testing procedure is speed up by means of the strategy of not executing redundant test cases plus recompenses their execution instance.

Some software generate correct result for some inputs and generate incorrect result for some inputs so it is depend upon the various software requirement contexts. Software reliability growth models perform the quantitative analysis of the failures[02][03].

Ganesh Pai[7] categorized the reliability models as black box models and white box models. White box models deals with the internal structure *i.e*. source code of the software. And black box models deals with functional behavior of the software. Various SRGMs have been proposed by many researchers. They have assumes various postulations depends on the environmental circumstances. Jelinski-Moranda de-eutrophication model [8] is one of the simplest plus earlier developed model. This model assumes various postulations. This model supposes that removal process of the fault eliminate the fault completely and without the introduction of new faults. But this assumption is totally impractical. Goel and Okumoto[11] have proposed the new

 Copyright © 2016 SERSC

approach towards the software reliability growth model relying on non-homogeneous Poisson process. Kapur and Garg [09] have introduced the concept of imperfect debugging in Goel and Okumoto model. Non-homogeneous Poisson Process dependant SRGMs are effective. Shinya Ikemoto *et al*. evaluates the algebraic estimation of software testing mechanism plus reliability of product by incorporating the metric dependant SRMs[17]. A. H. S. Garmabaki el al. [18] proposes the reliability model which can designed for the multi-release open source software (OSS), they have estimates the various factors in order to evaluate such software systems[18].

We have implemented Non-homogeneous Poisson Process dependant SRGM with the inclusion of testing time and testing coverage.

2. Background and Motivation

Maintaining software reliability and quality is the challenge in today's software industry. So developers and testers must have to develop efficient testing technique in order to design the reliable software that can performs its tasks without departure of the software requirements. This is the main and motivating factor behind the development of this system. Kiyoshi Honda *et al*. presents their view about SRGM[20]. They have analyzed the results of testing phase after the application of SRGM and an expectation values. By observing the deviation between these factors they have concluded problems with in the development of software. So it is the challenge to reduce this deviation to get better results. This is also the motivating factor.

3. Software Development Process

Software is developed by applying the systematic software development activities. Different types of models are utilized to develop the software. Waterfall is the basic and simplest development model. But waterfall model doesn't support the iterative process so; iterative software development model has been developed. After the first iterative development product is delivered to the customers and if the customer is not satisfied with the developed product then again it will be redesigned according to the customer feedback. Systematic development process helps to release the software within the appropriate time. Basic activities within the software development are as follows [7]:

a. Communication: Customer should state the needs or the requirements at the very initial stage. Through the communication between the customers and developers, requirements exactly discovered.

b. Planning: Resource management activity includes within the planning.

c. Modeling: For the sake to develop the software there should be the correct design according to which software is developed.

d. Construction: Designed system is implemented with the help of appropriate programming languages.

e. Deployment: After the complete development of software it is delivered to the customer and customer provides the feedback

4. Testing Coverage

Testing Coverage supposes to accomplish a enormously substantial task within envisaging the software reliability. TC assists software programmers to carry out the appraisal of the superiority of inspected software project plus to decide the amount of supplementary efforts required to progress the reliability of the software. Consequent are the classes for testing coverage [5]:

1. Statement coverage: - This class is articulated by means of the estimation of the fraction of statements enclosed via the group of designed test cases.

Copyright © 2016 SERSC

2. Decision/condition coverage: - This coverage is evaluated by means of evaluation of the fraction of decision branches enclosed via the group of designed test cases.

3. Path coverage: - This coverage is intended to estimate the fraction of execution paths or conducts throughout a program covered by the bunch of planned test cases.

4. Function coverage:- Total function count exercised by test cases is nothing but the functional coverage.

5. Testing Time

Testing time includes the instance necessary to carry out test process. Testing instance rely on the variety of factors such as testing technique, size of code, and sometimes the efficiency of testing tool *etc*. Total testing time comprises summation of instance calculation essential to examine each and every module within the software.

6. Mathematical Model

Basic concepts:
Block coverage: Total count of blocks that have been executed by test cases
Branch coverage: total count of branches that have been executed by test cases

$$Block\ coverage = \frac{Number\ of\ the\ blocks\ covered\ by\ the\ case}{total\ no\ of\ the\ blocks} \quad (1)$$

Notations:
m(t) = Expected no of Faults detected at time t
λ (t) = Failure Intensity of the software at time t
c(t) = Coverage function over a time interval t
α = Expected no. of faults that may be detected given infinite testing time

Basic NHPP :

$$\frac{dm(t)}{d(t)} = \lambda(t) \quad (2)$$

Test Case1 :

$$\frac{dm_1(t)}{dt} = \propto \int_0^{t_1} c_1\, dt \quad (3)$$

Where
t1 is first phase end of testing,
α Expected no. of faults detected given infinite no. of testing time
C1 is coverage function over interval time $0 \leq t < t1$

Test Case 2 :

$$\frac{dm_2(t)}{dt} = (\propto -(m_1(t)) \int_{t_1}^{t_2} c_2\, dt \quad (4)$$

Where
t_2 is second phase end of testing
C_2 is coverage function over interval time $t_1 \leq t < t_2$

Test Case3:

 Copyright © 2016 SERSC

$$\frac{dm_3(t)}{dt} = (\propto -(m_1(t) + m_2(t)) \int_{t_2}^{t_3} c_3\, dt \tag{5}$$

Where
t_3 is third phase end of testing
C_3 is coverage function over interval time $t_2 \leq t < t_3$

Proposed EHPP Model:

$$\frac{\boldsymbol{dm_i(t)}}{\boldsymbol{dt}} = (\propto - \sum_{\boldsymbol{k=1}}^{\boldsymbol{i}} (\boldsymbol{m_k(t)}) \int_{\boldsymbol{t_{i-1}}}^{\boldsymbol{t_i}} \boldsymbol{c_i\, dt} \tag{6}$$

Where
ti is end of ith phase of testing
Ci is coverage function over interval time ti-1 ≤ t < ti

7. Proposed System Architecture

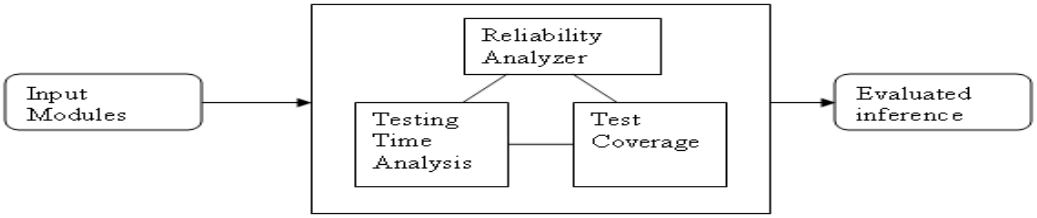

Figure 1. System Architecture

Figure 1 show the internal system architecture composed of various functional components. System encompasses the subsequent three components:

1. Input modules
2. Software reliability analyzer depends on testing time and testing coverage
3. Evaluated inference

First of all, user has to give the software with its entire coding module. Our project will check the reliability based on white box testing strategy so internal modules should have be given as an input to the system. Then Estimation of each module within the software is analyzed separately through the use of testing time and testing coverage. Object oriented metrics are estimated in this component

After this estimation evaluated inference is shown by expressing the result in the form of graph. Figure 2 and 3 are shows the graph for the testing time detection rate and Error rate.

Copyright © 2016 SERSC

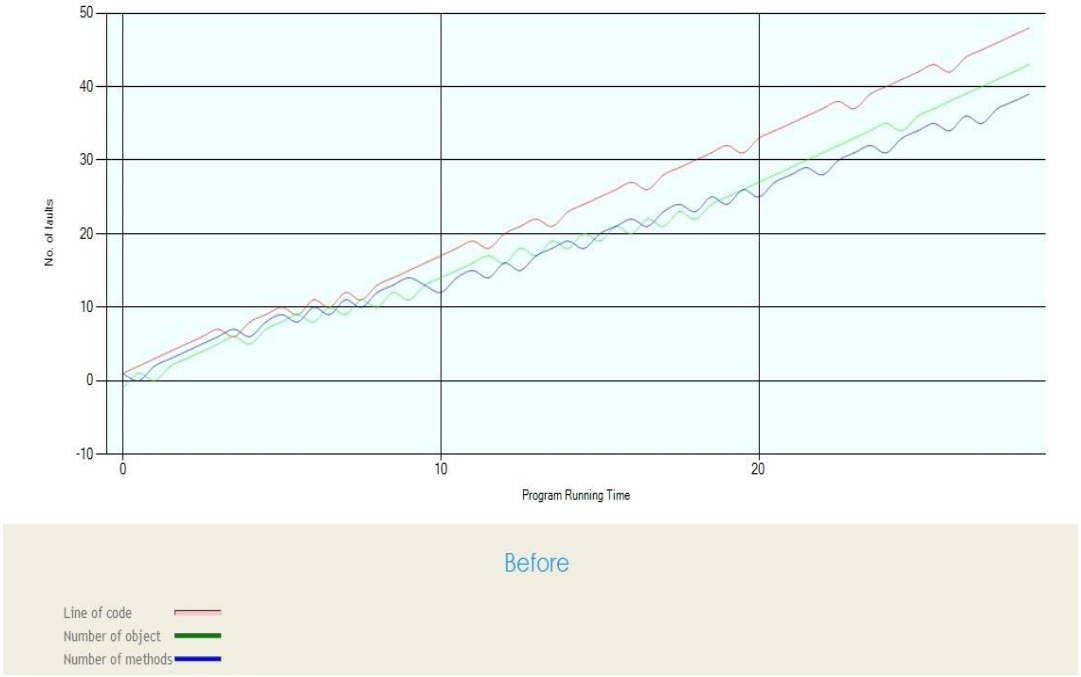

Figure 2. Before Using Threshold Value

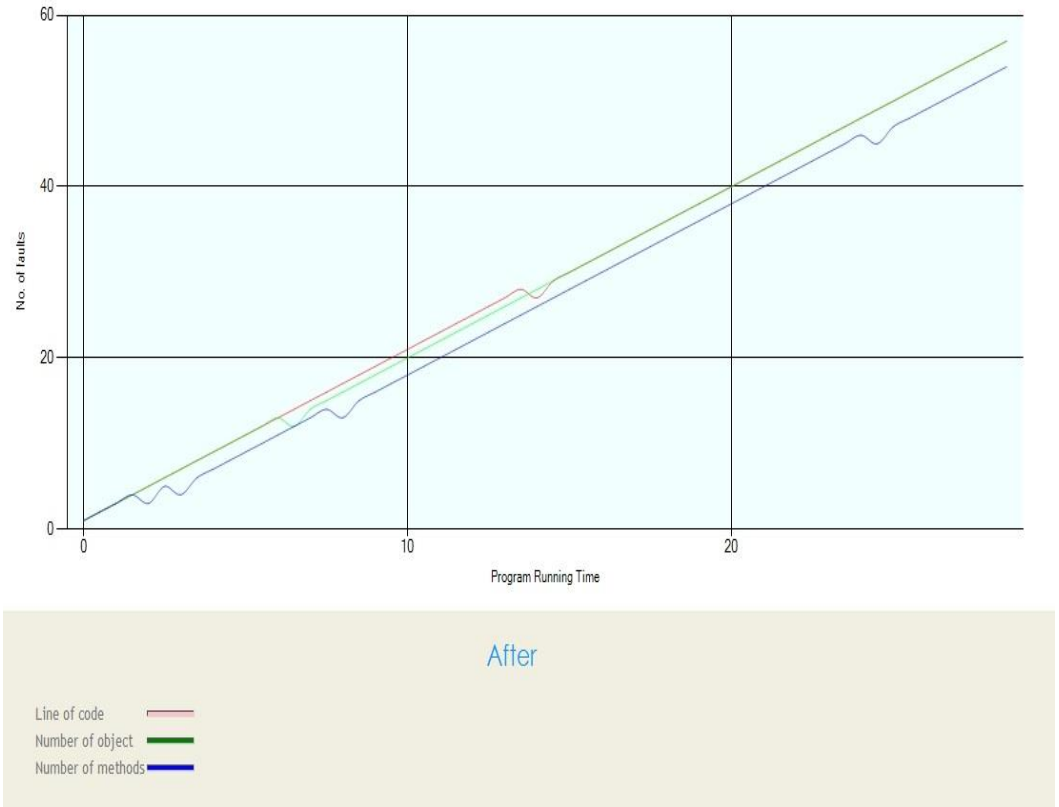

Figure 3. After Using Threshold Value

Figure 2 shows before using threshold value the complexity of software is high because maximum lines in graph are zigzag in nature. But in Figure 3 shows result analysis after using threshold value. After the given the suggestion by our model and make changes in

 Copyright © 2016 SERSC

model then definitely increases the reliability of software shown in Figure 3 maximum lines are straight means reliability of software is better than previous one.

8. Conclusion

Maintaining software superiority is the important task while developing the software. Reliability is estimated through the use of software reliability growth model. This paper presents the novel approach towards the reliability by considering the two factors *i.e.* testing time and testing coverage which definitely be very useful in industry to analyze the reliability of the software.

References

[1] B. Roy, S. Kr. Misra, A. Basak, A. Roy and D. Hazra, "A Quantitative Analysis of NHPP Based Software Reliability Growth Models", International Journal of Innovative Research in Computer and Communication Engineering, vol. 2, Issue 1, ISSN (Print): 2320-9798, **(2014)**, pp. 2432-2438.
[2] J. Yang, Y. Liu ,M. Xie, M. Zhao, "Modeling and analysis of reliability of multi-release open source software incorporating both fault detection and correction processes", Journal of Systems and Software, vol. 115, **(2016)**, pp. 102–110.
[3] M. Zhu, X. Zhang and H. Pham, "A comparison analysis of environmental factors affecting software reliability", Journal of Systems and Software, vol. 109, pp. 150–160, **(2015)**.
[4] W. Afzal, S. Alone, K. Glocksien and R. Torkar," Software test process improvement approaches: A systematic literature review and an industrial case study", Journal of Systems and Software, vol. 111, **(2016)**, pp. 1–33.
[5] M.-H. Chen, M. R. Lyu, IEEE and W. E. Wong, "Effect of Code Coverage on Software Reliability Measurement", IEEE Transactions on Reliability, vol. 50, no. 2, **(2001)**.
[6] G. Pai, "A Survey of Software Reliability Models", Department of ECE University of Virginia, VA Dec. 6, **(2002)**.
[7] A. L. Goel and K. Okumoto, "Time dependent error detection rate model for software reliability and other performance measures," IEEE Trans. on Reliability, vol. R-28, no. 3, **(1979)**, pp. 206–211.
[8] P. K. Kapur and R. B. Garg, "Optimal software release policies for software reliability growth model under imperfect debugging", RAIRO, vol. 24, **(1990)**, pp. 295–305.
[9] S. Wang, Y. Wu, M. Lu and H. Li, "Software Reliability Accelerated Testing Method Based on Test Coverage", IEEE, **(2011)**.
[10] C. D. Scott and R. E. Smalley, "Diagnostic Ultrasound: Principles and Instruments", Journal of Nanosci. Nanotechnology., vol. 3, no. 2, **(2003)**, pp. 75-80.
[11] H. Pham and Editor, "Handbook of Reliability Engineering", Springer, **(2003)**.
[12] R. S. Pressman, "Software Engineering: A Practitioner's Approach", McGRAW Hill international publication, seventh edition, **(2004)**.
[13] M.-H. Chen, M. R. Lyu and W. E. Wong, "An empirical study of the correlation between code coverage and reliability estimation", Proceedings of the 3rd International conference on Software Metrics Symposium, Berlin, **(1996)**.
[14] J. Kimura and H. Shibasaki, "Recent Advances in Clinical Neurophysiology", Proceedings of the 10th International Congress of EMG and Clinical Neurophysiology, Kyoto, Japan, **(1995)**.
[15] W. E. Wong, J. R. Horgan, S. London and A. P. Mathur, "Effect of test set size and block coverage on fault detection effectiveness", Proc. 5th IEEE Int'l. Symp. Software Reliability Engineering, **(1994)**, pp. 230–238.
[16] S. Wang, Y. Wu, M. Lu and H. Li, "Software reliability modeling based on test coverage ", 9th International Conference on Reliability, Maintainability and Safety (ICRMS), **(2011)**, pp. 665-671, ISBN: 978-1-61284-667-5.
[17] S. Ikemoto, T. Dohi and H. Okamura, "Quantifying software test process and product reliability simultaneously", IEEE 24th International Symposium on Software Reliability Engineering (ISSRE) , ISSN :1071-9458, **(2013)**, pp. 108-117.
[18] A. H. S. Garmabaki, A. Barabadi, F. Yuan and J. Lu, "Reliability modeling of successive release of software using NHPP", IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), **(2015)**, pp. 761-766.
[19] K. Honda, S.-K. Tokyo, Japan, H. Washizaki, Y. Fukazawa and K. Munakata, "Detection of unexpected situations by applying software reliability growth models to test phases", 2015 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW),**(2015)**, pp.2-5.
[20] D. Alrmuny, "A Comparative Study of Test Coverage-Based Software Reliability Growth Models", 11th International Conference on Information Technology: New Generations (ITNG), pp. 255-259, ISBN:978-1-4799-3187-3, **(2014)**.

Copyright © 2016 SERSC

 Copyright © 2016 SERSC