ISSN:0975-9646

Estimating The Software Metrics Using Automatic Testing And Integrating Testing

Lithu Mathew [1], Mr. P.M.S.S. Chandu[2]

[1] *Department Of Computer Science & Engineering Sathyabama University , Chennai, India*

[2]*Asst. Professor, Department Of Computer Science & Sathyabama University, Chennai, India*

***Abstract* — A number of analytical models have been proposed to address the problem of quantifying the software dependability, one of the most important metrics of software quality. The difference of existing software reliability models can be classified according to the several different classification systems. The classification proposed is based primarily on the phase of software life cycle during which the models are applicable: debugging phase, validation phase, or operational phase. Complex systems can incur huge verification costs. Actual specification usually assigns predefined risk levels to components in the design phase, to provide some instruction for the authentication. It is a rough-grained drill that does not contemplate the costs and does not provide sufficient modelling basis to let engineers quantitatively optimize resources usage. Software accuracy allotment models partially address such affairs, but they usually make so many expectation on the input parameters that their application is difficult in practice. In this paper, we try to trim this break, proposing reliability and testing resources allocation model that is able to provide solutions at various levels of detail, depending upon the knowledge the admin has about the system. The model aims to significantly classify the most critical components of software architecture in order to best assign the testing resources to them.**

Keywords- Software Quality ,cost, process, various test cases.

I. INTRODUCTION

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test .Software testing can also provide an objective, autonomous outlook of the software to allow the business to appreciate and understand the risks of software application. Test techniques build, but are not defined to the process of executing a program or application with the intent of finding software bugs (errors or other defects).

Software testing can be stated as the action of validating and verifying that a computer program/application/product: meets the requirements that guided its design and development, works as expected, can be implemented with the same characteristics, and satisfies the needs of stakeholders.

Software testing, depending on the testing method engaged, can be implemented at any date in the software development process. Commonly most of the test effort occurs after the requirements have been defined and the coding process has been ended, but in the agile path most of the test attempt is on-going. As such, the procedure of the test is governed by the chosen software development methodology. Toward this aim, the development process of such systems is usually complemented by several analysis techniques (e.g., hazard analysis, FTA, and FMECA) in the requirement specification and in the design phase as well. Once the system has been implemented, the verification process has to provide the final assurance that the system meets the required reliability level. The verification phase is usually responsible for the major fraction of the overall costs, especially for critical systems. The efficacy of the verification phase strongly depends on the correct identification of the most critical components in the software architecture, as the convenient testing capability are usually allotted based on the components' risk levels. Several researchers have tried to quantify the required software components reliability that will assure a minimum total system reliability. This optimization problem has usually been addressed as a reliability allocation problem. Most of the papers in the software field coped with the design phase and dealt with the redundancy reliability allocation. some authors also dealt with the problem in the authentication stage , where the issue is to allocate reliabilities to be achieved during testing . Typically, these problems are addressed by proposing some kind of model that allows engineers to carry out an optimal allocation.

Software bugs will almost always exist in any software module with moderate size: not because programmers are careless or reckless, but because the intricacy of software is generally intractable and humans have only limited ability to manage complexity. Discovering the design defects in software, is equally difficult, for the same reason boundary values are not sufficient to guarantee correctness. A further complication has to do with the dynamic nature of programs. If a defeat occurs during preliminary testing and the code is alternated, the software may now work for a test case that it didn't work for previously. But its behaviour on pre-error test cases that it passed before can no longer be guaranteed.

In this paper, we propose an approach to quantitatively identify the most critical components of software architecture in order to best assign the testing resources to them. In particular, we present an optimization model for testing resources allocation that includes all of the mentioned aspects affecting the reliability of a complex software system. In order to represent the software architecture, we employ the so-called architecture-based reliability model; in particular, a Discrete Time Markov Chain (DTMC)-type state-based model is take-up. This allows us to accurately consider the effects of such architectural features as loops and conditional branching on the overall reliability. Moreover, the architectural model encompasses the operating system to consider its reliability and its influence on the application. The proposed optimization model also considers the most common fault tolerance mechanisms (such as restart a component, retry application as recovery mechanisms as also a failover to a standby) that critical systems typically employ. Furthermore, we try to impart the necessary flexibility to the model by: 1) providing different levels of solutions according to the information the user gives as input and 2) carrying out a sensitivity analysis in order to analyze the effect of the variation of some parameters on the solution. Information needed for model parameterization can be obtained by the user either considering design/code information (such as UML diagrams) and simulation before the testing of the system version under consideration or by dynamically profiling a real execution from system test cases of a previous version. Depending on the availability and the accuracy of information, the user may adopt one of the two approaches (or a combination of both). Finally, the impact of performance testing time and the second-order architectural effects are also considered for greater accuracy of the result.

II. RELATED WORK

For a certain amount of total testing time, only a fraction of the injected faults are removed. At the end of the testing, the reliability predicted by the model is compared with the In existing systems faulty version of the program is created by reinserting faults belonging to real fault sets discovered during integration testing .This faulty version emulates the previous version of the application. Testing execution for the faulty version is done only actual achieved Reliability. A lot of work in the past considered the optimal allocation of the reliabilities to minimize a cost function, related to the design or the verification phase costs. Much initial research dealt with hardware systems (e.g., the series-parallel redundancy- allocation problem has been widely studied); software systems received attention more recently. Most of the work in the software area is concerned with the design phase in which the goal is to select the right set of components with a known reliability and the amount of redundancy for each one of them, minimizing the total cost under a reliability constraint or maximizing the total reliability under a cost constraint (more specifically, this is a redundancy reliability allocation problem). In some cases, they also considered the redundancy strategies and the hardware. For instance, the work in when redundancy is not considered, the reliability allocation problem can still refer either to the design or to the verification phase. For instance, authors in proposed an economic model to allocate reliabilities during the design phase, minimizing a cost function counting on fixed development costs and a previously experienced failure decrease cost. The task in also refers to the design phase and authors define a general-behaviour cost function to relate the costs to the reliability of a component.

Not many papers considered the problem in the software verification phase, where the concern is to earmark reliabilities that components need to achieve during their testing. Among these papers, authors in proposed an optimization model with the cost function based on well known reliability growth models. They also include the use of a coverage factor for each component, to take into account the possibility that a failure in a component could be tolerated. Some of the cited papers also consider the solution for multiple applications, i.e., they aim to satisfy reliability requirements for a set of applications. However, none of the cited papers explicitly considers the architecture of the application. Work in considers the software architecture implicitly, by taking into account the utilization of each component with a factor assumed to be known. Among these, only Everett refers to the verification phase. Almost all of the cited papers about reliability allocation belong to the class of the so called additive models. However, there are other ways to describe a software application which can explicitly consider the architecture and lend themselves to an easy integration with the other aspects described in Section 1, such as the Operating System, the fault tolerance mechanisms, the sensitivity analysis and the performance testing. They are the state-based models and the path-based models. Both the latter ones and additive models belong to the class of the so-called Architecture-based models. This kind of model has gained importance since the advent of object-oriented and component-based systems, when the need to consider the internal structure of the software to properly characterize its reliability has become important (in the past, reliability analysis was conducted mainly considering the software as a black box). This led to an increasing interest in the architecture-based reliability and performance analysis. State-based models use the control flow graph to represent software architecture; they assume that the transfer of control among components has a Markov property, modelling the architecture as a DTMC, a Continuous Time Markov Chain (CTMC), or semi- Markov Process (SMP). Path-based models compute the system reliability considering the possible execution paths of the program.

Additive models, mentioned above, where the component reliabilities are modelled by no homogeneous Poisson process (NHPP) and the system failure intensity is computed as the sum of the individual components failure intensities So far, state-based and path-based models have been mainly used to analyze system reliability, starting from its Component reliabilities, while the reliability allocation problem has been mainly based on additive models, as described above. In the former, the software Architecture and the failure behaviour of the software are combined in the same model, while a hierarchical approach separately solves the architectural model and then superimposes The failure behaviour of the components on the solution. Although hierarchical models provide an approximation to the composite model solution, they are more Flexible and computationally tractable. In the composite model, evaluating different architectural alternatives or the effect of changing individual components behaviour is computationally expensive.

III. Proposed System

Optimization model is allocated to achieve the target in less period of time. The testing resources are implemented manually and automatically for different modules. Ordering for the system is done to achieve a required reliability level. Minimum verification costs are produced. Manual testing is proved to best one rather than automated testing.

A. **Performance evaluation.**

3.1. Quality Attribute for Manual Testing

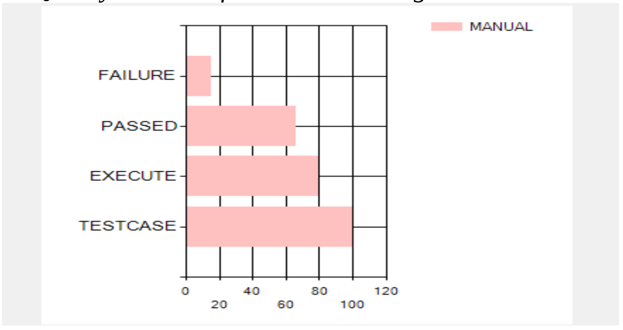

Fig: 3.1: Quality attribute for manual testing graph

X-axis-Test cases

Y-axis- Quality attribute

This graph shows the x axis value of the manual efficiency of quality in numeric values. And y axis shows value of the metrics of the quality attribute.

3.2. Cost Attribute for Manual Testing

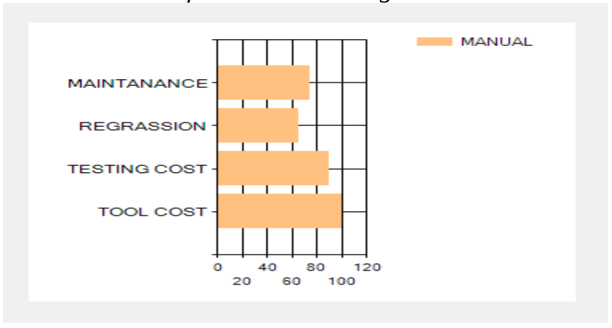

Fig: 3.2: Cost attribute for manual testing graph.

X-axis-Test cases

Y-axis- Cost attribute

This graph shows the x axis value of the manual efficiency of quality in numeric values. And y axis shows value of the metrics of the quality attribute.

3.3. Process Attribute in Manual Testing

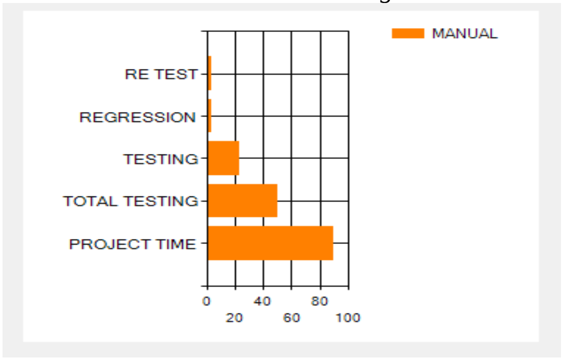

Fig: 3.3: Process attribute for manual testing graph.

X-axis-Test cases

Y-axis- Process attribute

This graph shows the x axis value of the manual efficiency of process in numeric values. And y axis shows value of the metrics of the quality attribute.

3.4.Graph name: Quality Attribute for Automated Testing

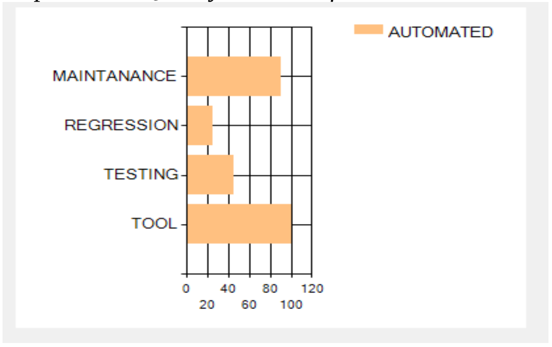

Fig: 3.4:Quality attribute for automated testing graph.

X-axis-Test cases

Y-axis- Quality attribute

This graph shows the x axis value of the automated efficiency of quality in numeric values. And y axis value shows of the metrics of the quality attribute.

3.5.Graph name: Cost attribute for automated testing

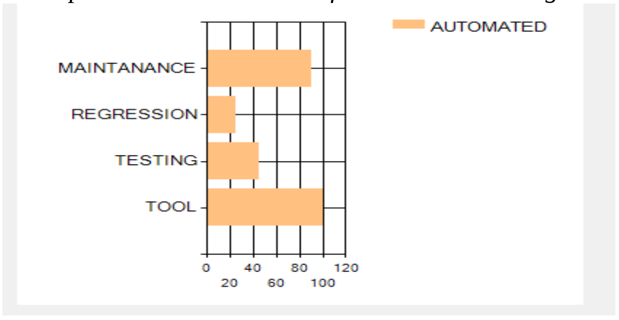

Fig: 3.5: Cost attribute for automated testing graph.

X-axis-Test cases

Y-axis- Cost attribute

This graph shows the x axis value of the automated efficiency of cost in numeric values. And y axis value shows of the metrics of the cost attribute.

3.6.Graph name: Process Attribute for Automated Testing

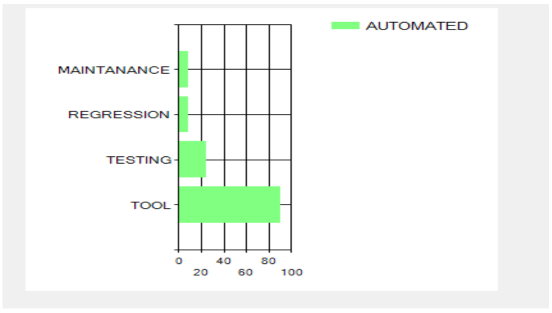

Fig: 3.6: Process attribute for automated testing graph

X-axis: Test cases

Y-axis: Process attribute

This graph shows the x axis value of the automated efficiency of process in numeric values. And y axis value of the metrics of the process attribute.

3.7.Comparison between Manual and Automated Testing

This graph shows the x axis value of the both manual and automated efficiency of quality, cost, and process in numeric values. And y axis value of the metrics of those attributes.

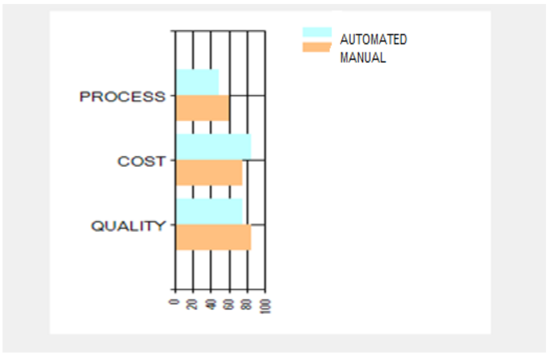

Fig: 3.7: Comparison between manual and automated testing graph.

IV. CONCLUSION

We proposed a Metrics model to allocate the testing resources to different system components like quality, cost, and process in order for the system to achieve a required reliability level at minimum verification costs. The purpose of the model, through the tool implementing it, is, therefore, to drive engineers in the verification phase. The optimization model was used to provide flexible solutions, at different levels in manual testing as well as automation tools, to the information provided by the user.

REFERENCES

[1] J. Onishi, S. Kimura, R.J.W. James, and Y. Nakagawa, “Solving the Redundancy Allocation Problem with a Mix of Components Using the Improved Surrogate Constraint Method,” IEEE Trans. Reliability, vol. 56, no. 1, pp. 94-101, Mar. 2007

[2] C. Huang, S. Kuo, and M.R. Lyu, “An Assessment of Testing- Effort Dependent Software Reliability Growth Models,” IEEE Trans. Reliability, vol. 56, no. 2, pp. 198-211, June 2007.

[3] V. Almering, M. Van Genuchten, G. Cloudt, and P.J.M. Sonnemans, “Using Software Reliability Growth Models in Practice,” IEEE Software, vol. 24, no. 6, pp. 82-88, Nov./Dec. 2007.

[4] S. Gokhale, M.R. Lyu, and K.S. Trivedi, “Incorporating Fault Debugging Activities into Software Reliability Models: A Simulation Approach,” IEEE Trans. Reliability, vol. 55, no. 2, pp. 281-292, June 2006.

[5] C. Huang and M.R. Lyu, “Optimal Release Time for Software Systems Considering Cost, Testing-Effort, and Test Efficiency,” IEEE Trans. Reliability, vol. 54, no. 4, pp. 583-591, Dec. 2005

[6] I. Rani and R.B. Misra, “Economic Allocation of Target Reliability in Modular Software Systems,” Proc. Ann. Reliability and Maintainability Symp. pp. 428-432, 2005.

[7] SAF.ET1.ST03.1000-MAN-01, “Air Navigation System Safety Assessment Methodology (v2-0),” EUROCONTROL EATMP Safety Management, Apr. 2004.

[8] N. Wattanapongsakorn and S.P. Levitan, “Reliability Optimization Models for Embedded Systems with Multiple Applications,” IEEE Trans. Reliability, vol. 53, no. 3, pp. 406-416, Sept.2004.

[9] M.R. Lyu, S. Rangarajan, and A.P.A. van Moorsel, “Optimal Allocation of Test Resources for Software Reliability Growth Modeling in Software Development,” IEEE Trans. Reliability, vol. 51, no. 2, pp. 183-192, June 2002.

[10] A.O.C. Elegbede, C. Chu, K.H. Adjallah, and F. Yalaoui, “Reliability Allocation through Cost Minimization,” IEEE Trans. Reliability, vol. 52, no. 1, pp. 106-111, Mar. 2003.

[11] K.S. Trivedi, Probability and Statistics with Reliability, Queuing and Computer Science Applications, John Wiley and Sons, 2001.