A New Model to Reduce Testing Efforts

Moniya
M.Tech Student, SPGOI College of Engineering, MD University, Rohtak (India).

Abstract – Software Testing has played an increasingly important role in systems acquisition, engineering and development, particularly for large, complex systems. Underestimating the costs may result in management approving proposed systems that then exceed their budgets, with under developed functions and poor quality, and failure to complete on time. Over estimating may result in too many resources committed to the projects or during contract bidding, result is not winning the contract, which can lead to loss of jobs. Testing effort is often a major cost factor during software development. Many software organizations are spending up to 40% of their resources on testing [1]. Therefore, an existing open problem is how to reduce testing effort without affecting the quality level of the final software. Thus to provide solution to the problem here I have defined a new model that can be helpful in reducing testing effort.

Index Terms – Testing, Cost & Size.

1. INTRODUCTION

Testing is an activity that is carried out in software development to ensure quality. It is an attempt to find and remove errors that may have been introduced during software development and maintenance [2]. Carrying out testing may not prove the absence of errors therefore the two primary reasons for carrying out testing may be to locate and fish-out defects so the customer do not receive a flawed product secondly is to prove to the customer that the system satisfies all requirements according to the customer's expectation [3].In fact testing is so important that test teams are allocated solely for testing in companies [4]. There are even companies whose sole job is to carry out software testing [3].The process starts from the moment the test analyst starts evaluating test requirements up to the time a fully debugged test script is written and executed at least once. Test effort often means money because the time spent in testing must be paid for by the company this is why it is important to be able to calculate test effort either before testing or after testing.

Before executing test cases, the test plan and strategy to be adopted should have been finalized into the test specification. Also required in order to execute test cases is the test case specification which is "a document specifying a set of test cases (objective, inputs, test actions, expected results, and execution preconditions) for a test item" [2].

Classical estimation models are established based on linear or non-linear regression analysis, which incorporate fixed input factors and fixed outputs. The size of the project determining the scope is modeled as a main input of such models [6], [5]. One representative of such models is COCOMO [7], [8]. The most critical problem in such an approach is the wealth of data that is needed to get regression parameters, which is often impossible to get in needed quantities. It really limits their usage for project estimation. In recent years, a flexible and competitive method, Bayesian Belief Network (BBN), has been proposed for software project estimation [9], [10], and [11].

2. COST ESTIMATION TECHNIQUES

Software cost estimation methods are grouped into two methods according to my research. These are the algorithmic and the non-algorithmic methods. The algorithmic models are based on the use of mathematical rules/algorithms such as statistical means, standard deviation, regression, differential equations etc., to calculate cost. Accuracy of these models are quiet mixed and cannot be used as off-the-shelf [13]. The non-algorithmic are non-mathematical in nature and rely upon human judgments based upon professional experience, previous similar projects, resources available and other forms and means for the estimation [12][13][14].

2.1 Price-to-win

The software development effort is estimated based on the best price to win the project. This means the cheapest i.e. what the customer can afford. For instance, if the cost is estimated reasonably to be 80 persons / month but the customer can only afford 50 person/ month, the estimation will be modified to fit only 50 persons /month. This is also not a good software engineering practice. This can lead to low quality system and the developers can be forced to work overtime among others [13] [15].

2.2 Bottom-Up

In this method, estimation must start from the grassroots or bottom units such as components with all the tasks and functionality breakdown done and the summation of all the efforts and schedules will be the total estimation for the project. This method requires a detailed knowledge of software engineering principles and architecture. This method is also labor intensive but it captures the entire scope of work needed and provides a better assurance than all other methods. A requirement for this system is that an initial design must be in place showing the breakdown of the system [12] [13] [15].

2.3 Top-Down

This is the opposite of the bottom-up approach. System level activities are broken down into various components or units. It

starts from the overall system's global properties and using algorithmic or non-algorithmic methods to derive cost or schedule estimates, this estimate is then shared among the different sub-levels such as components and other units. This method does not require knowledge of software architectural design and it is more suitable for cost and schedule estimate at the initial level of estimation of the project. However, the detailed-level activities can be under-estimated under this method [13] [15].

3. SOFTWARE SIZE ESTIMATION

The software size may be the one factor that affects the software cost the most [13] and invariably test effort. This is why we must estimate software size correctly and not too low in order to avoid cost and schedule overruns.To size software, we need to employ a variety of software sizing techniques and not to rely only on one method in order to avoid budget schedule risks from too low size estimates. According to experts, software size can be determined from the following:

1. The number of functional requirements in both the requirement specification and the interface specification.
2. The number of software units as contained in the software design description.
3. The source lines of codes SLOC or the function point estimates.

Software size should be measured, tracked and controlled throughout the development of the software so that estimates can be compared to the actual size and to determine trends and progress [12].

Furthermore, it is very important to estimate software size early in the software development lifecycle so that early significant deviations can be detected. These deviations can be as a result of the following problems:

1. Error in the model or logic used for in the development of the estimates
2. Error in the requirement design, coding, process etc.
3. Unrealistic or wrongly interpreted requirements and resource estimates used for the development of the software.
4. Error in the development rate estimation Perhaps most importantly, size based estimation should be compared to existing similar project's size, scope and complexity.

4. NEW APPROACH

A new frame work is introduced for measuring software testing process. This framework has combined different metrics of testing projects and derived a new metric which covers different testing aspects. Different metric used in this new approach for reducing testing efforts are: Efficiency, Mean time to Failure, Failure Rate, Error Coverage and Defect Density.

Working of the Model – Data for several project or system is collected from various sources over the internet to perform testing on it. At first the test cases are generated and then proper planning is done for each test case. Then the further steps will be taken to have different measures for same project.

5. PROPOSED MODEL

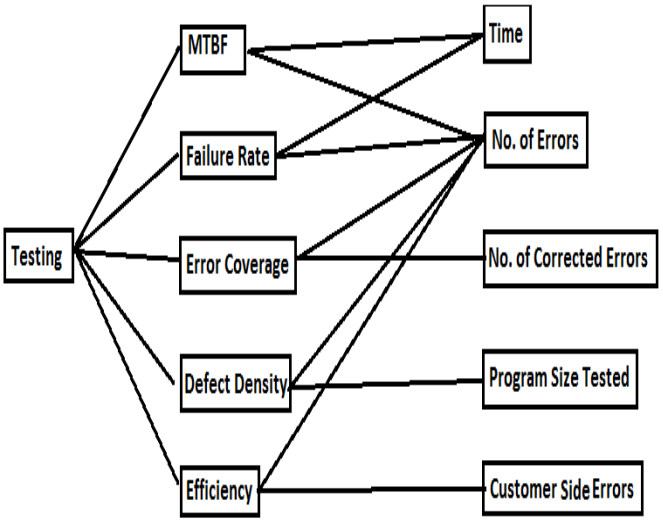


Figure 1: Proposed Model

6. TEST CASES

Test Case ID	Time	No. of Errors	Program Size	Corrected Error	Customer Side Errors
01	12	3	85	1	4
02	10	2	49	2	6
03	16	5	73	3	5
04	20	1	96	1	3

Table 1: Test Cases

7. CALCULATIONS

MTBF = (Time / No. of Errors) * 100

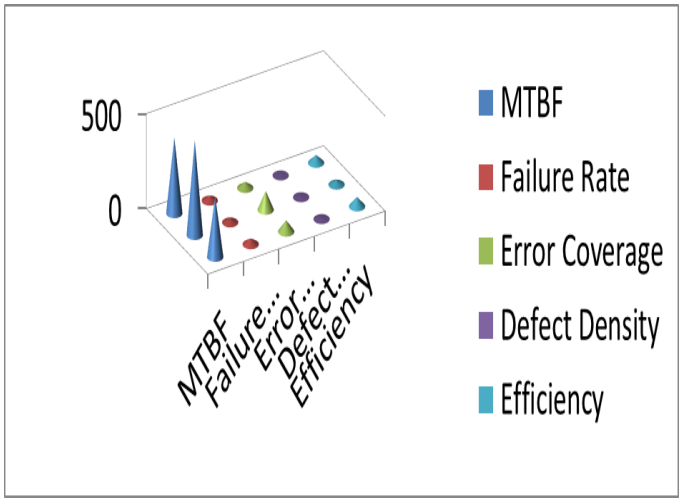
Failure Rate = (No. of Errors / Time) * 100

Error Coverage = (No. of Corrected Errors / No. of Errors) * 100

Defect Density = (No. of Errors / Program Size) * 100

Efficiency = (No. of Errors / (No. of Errors + Customer Side Errors)) * 100

8. RESULTS



9. CONCLUSION

The model that I have described here is not rigid in nature. I have used few metrics that will be helpful in reducing testing effort. Number of more metrics can also be used that will helpful in reducing testing effort. So it's all up to the user he/she can add or remove metrics from model as per requirements. Thus number of more cases can be produces with different implementation of my proposed model which can be consider under future work under my research work

REFERENCES

[1] F. Elberzhager, A. Rosbach, J. Münch and R. Eschbach, "Reducing test effort: A systematic mapping study on existing approaches," *Information and Software Technology 54*, p. 1092–1106, 2012.

[2] Erik Erik Van Veenendaal, (2002), *the Testing Practitioner*, UTN Publishers, the Netherlands.

[3] http://www.effortestimator.com/Test%20Effort%20Estimation.pdf (last visited 2011-08-11)

[4] http://promisedata.org/pdf/phil2006AranhaBorba.pdf (last visited-2011-07-31)

[5] Chen Qingzhang, Fang Shuojin, Wang Wenfu, "Development of the Decision Support System for Software Project Cost Estimation", World Congress on Software Engineering,IEEE, 2009.

[6] Yinhuan Zheng, Yilong Zheng, Beizhan Wang, Liang Shi, "Esti- mation of software projects effort based on function point", 4th International Conference on Computer Science and Education, 2009.

[7] Jin Yongqin, Li Jun, Lin Jianming, Chen Qingzhang, "Software Project Cost Estimation Based On Groupware", World Congress on Software Engineering, IEEE, 2009.

[8] Pichai Jodpimai, Peraphon Sophatsathit, and Chidchanok Lursin- sap, "Analysis of Effort Estimation based on Software Project Models", IEEE, 2009.

[9] Hao Wang, Fei Peng, Chao Zhang, Andrej Pietschker, "Software Project Level Estimation Model Framework based on Bayesian Belief Networks", Sixth International Conference on Quality Software (QSIC'06), IEEE, 2006.

[10] angyang Yu, Charlottesville, "A BBN Approach to Certifying the Reliability of COTS Software Systems", annual reliability and maintainability symposium, IEEE, 2003.

[11] Ying Wang, Michael Smith, "Release Date Prediction for Telecommunication Software Using Bayesian Belief Networks", E Canadian Conference on Electrical and Computer Engineering, IEEE, 2002.

[12] http://www.stsc.hill.af.mil/resources/tech_docs/gsam3/chap13.pdf (last visited 2011-07-30)

[13] ftp://cs.pitt.edu/chang/handbook/42b.pdf (last visited 2011-08-11)

[14] http://www.bth.se/fou/cuppsats.nsf/all/1ce4270c92a2ca97c1257547004 5b588/$file /DV2403_Master_Thesis_Sohaib_Shahid_Bajwa.pdf (Last visited 2011-08-11)

[15] ranger.uta.edu/~khalili/Somerville%20Chapter%2026.ppt (Last visited 2010-04-10)