On Software Test Estimate and Requirement Tracking
Jing-Chiou Liou
Department of Computer science
Kean University
Union, NJ 07083, USA
jliou@kean.edu

Abstract
Test is a key activity for ensuring software quality. There is always pressure from project sponsor and management for software development team to commit to shorter schedule and lower cost, especially for testing. Some of the main challenges in testing today are to match the test cases with requirements correctly, and to provide accurate estimates and track the test progress accordingly. In this paper, we present a parametric model for software test estimate along with a test graph for matching test cases with requirements and test cases analysis to aid in producing a more accurate estimates and tracking. The model and the test graph can be used jointly or individually.

Keywords: Software test, estimate, requirement tracking

1. INTRODUCTION

In software development, project management holds a primary responsibility for building a successful software project. Project management not only has to commit to the quality software application being built, but also has to make sure the software project is on time and within budget. However, the commitments of schedule and budget are based on estimation. Inaccurate or incorrect estimates can cause the software project’s failure. These estimates for a software development project include size, effort, cost, schedule and/or (hardware and software) critical resources.

An estimation can be the result of one or combination of several estimating methods that include bottom-up, top-down, analogy, expert judgment, and parametric. The parametric model is probably a more accurate method than the others. It calculates the estimates by using attribute parameters in some form of formula (mathematical algorithm). However, the parametric model needs to be calibrated for each software application before its formula can be usefully applied.

The size of a test is usually in terms of the total number of test cases. With the hierarchy of test cases, different complexity of each test case, and uncertainty of error rate of codes implementation, converting the size estimate into effort and cost estimates for testing is somehow more difficult than the conversion at the other phases. Even with the aid of historical data, accurate estimates for testing effort/cost are still very difficult.

One of the issues in identifying total number of test cases needed for a software project is how to correctly match the test cases with the requirements and to track the progress accordingly. In this paper, we present a parametric estimation model for software test estimate, along with a test graph for matching test cases with requirements and test cases analysis to aid in producing a more accurate tracking. The model and the test graph had been used by multi-million-dollar software projects. It is proven that they can be used jointly or individually.

2. SIZING THE TEST EFFORT

Software application quality is the most important issue for customer satisfaction. On time and within budget may impress the customer at first, but a software application of poor quality can cause serious problems that cannot be recovered with any cost.

Software testing is an important software quality assurance activity in the software development life cycle. According to several studies, the testing effort consumes approximately 40-80% of the total software development cost [5]. But with the negative attitudes towards the testing effort, when the software development managers are asked to commit to a shorter delivery schedule, the cutting of the test effort is the first thing they consider.

During the past, many reliability models, software quality assurance processes, and software testing strategies have been proposed. Most of them were designed based on the traditional function-oriented paradigm. They have been found to be inadequate for testing OO systems [7, 13]. One significant difference between testing on traditional function-oriented software applications and on OO software applications is the object state testing [8]. In OO software application, many objects may have state dependencies and these objects may interact with each other [15].
The objective of software estimation is to generate realistic estimates that have the buy-in of both the project team and the customer (project sponsor). Anything that has unknown quantity can be estimated. While an accurate estimate may not be necessary, an inaccurate estimate may cause problems that no cost can recover. Also, producing an estimate intended to please the customer but not agreed to by the project team can lead the project into hot water.

An accurate estimate also comes with a threshold. When the project team is asked to reduce cost and schedule of development, accurate estimates can propose a bottom line of reduction without impacting on the quality. On the other hand, inaccurate estimates have no way to evaluate how much of a reduction can be taken by the project team.

To contribute to accurate estimates of a software project, software test team must carefully plan the testing activity by mapping the test plan with project plan, and matching the test cases with the requirements. This task of test plan mapping by far, unlike the mapping of code development with requirements, is not very precisely. Hence, whenever there is a cut in overall schedule, or a delay propagated from code development, test effort is always the one that got impacted severely.

The responsibility of inaccurate mapping of test plan with project plan may be not on software test team alone. Requirements change or creeping requirements is one of the critical challenges in software quality. However, the requirements traceability and test tracking are two tasks that any software test team must face and be responsible.

3. REQUIREMENTS TRACEABILITY

It is very common that requirements changes become progressively challenging after the completion of the definition phase. Not only it will ripple down to the following phases and increase the overall efforts, but also will create a possibility of losing such change until it causes big trouble to the project team.

It was reported in [6] that during the subsequent design phase, the average rate of requirements change may exceed 3% per month for many software projects. The burst of new requirements slow down to about 1% per month during coding, and eventually stabilized when testing commences. However, it is possible that requirements might still change even during the end of testing.

To minimize the impact from late requirements change, change control processes and configuration management tools are both utilized in today’s software development organizations. A Change Control Board (CCB) is usually established in a development organization to periodically review any possible requirements change. The CCB will determine if such changes should be incorporated into current release or to be deferred to next release.

A configuration management tool, such as IBM™ Rational ClearCase, can be used for version control to trace the software work products and their changes. Some similar software tools, such as Microsoft™ SharePoint, do not provide the function of strict configuration management, but can act as a repository and provide a light fashion of version control.

Other software tools, such as IBM™ Rational ClearQuest and Mercury™ Quality Center (QC), are also available for test cases and/or defects tracking. Unfortunately, these test cases or defects tracking tools usually do not provide an automated interface for requirements traceability with respecting to test cases.

Mercury™ QC does provide a tab for requirements that is used for requirements traceability of various test cases stored in the QC repository. But it does not provide a live update for requirement creep. Hence, most of the time, this requirements traceability is left to testers to manually match their test cases with requirements. Excel spreadsheet is usually the choice in many cases.

When requirement traceability is manually tracked, it is not too difficult to uncover any missing mapping between test cases and requirements prior to test begin. However, this manual fashion of requirement traceability tracking often misses the requirements change during testing phase due to the lacking of automated interface.

Towards an automated interface for requirements traceability, we need to consider a graphical representation of test cases. Whenever a requirement is added or altered, there must be at least one corresponding test case created.

4. A PARAMETRIC MODEL

A parametric model is a useful estimation tool especially for estimating effort/cost and schedule. Parametric models typically have a particular "perspective." [14] These models generate estimates based on statistical methods. Some of these models are PRICE [12], COCOMO 2.0 [3, 10] and FPA [1].

In each model, there is a form of "central equation" along with some attribute parameters. In the central equation, nominal unit effort (coefficient) is first estimated either using the historical data or as to perform the task in an ideal world. Then the attribute parameters (Non-nominal factors) are used in the central equation to adjust the real world effort.

In short, these can be represented as five common factors [2]:

- Function point
- Difficulty factor
- Project complexity
- Size of organization
- Development environment
One general form of the central equation for estimating the effort is:

\[ E = \alpha (\text{Size})^\beta (\text{attribute parameters}) \]  

(1)

Where \( \alpha \) is the nominal unit effort (coefficient) and \( \beta \) is the expansion factor of which the software size could grow. These two parameters are primary based on historical data.

To calibrate those attribute parameters for estimating testing effort [11], we need to analyze those five common factors. These common factors can be decomposed and re-grouped into two categories of attribute factors: Resource Maturity (RM) and Test Cases Complexity (TCC). Each of the 5 common factors is contained in either or both of these categories. For instance, reliability level is part of Resource Maturity level, project size is part of Complexity level, and the development environment is part of both Resource Maturity and Complexity level.

Each category also consists of some attribute parameters. The Resource Maturity is the sum of three parameters:

- RM1: Code Maturity
- RM2: Experience of Database and/or build/system administrator(s) DBA/SA.
- RM3: Experience of tester(s)

The Test Cases Complexity is the sum of two parameters:

- TCC1: Test cases complexity
- TCC2: Test cases hierarchy

Now the central equation in Equation (1) can be re-modeled to incorporate the attribute factors for estimating testing effort:

\[ E = \alpha (\text{Size})^\beta (\text{Resource Maturity Factor})(\text{Test Cases Complexity Factor}) \]  

(2)

For commercial software applications, the expansion factor \( \beta \) is mostly 1. The reason is that software development is based on requirements. When the software size grows over a threshold (i.e., requirements change significantly), re-estimates should be performed. Therefore, the Equation (2) becomes:

\[ E = \alpha (\text{Size})^{RM1+RM2+RM3}(TCC1+TCC2) \]  

(3)

Table 1 lists the applicable parameter values of the attribute factors that can be used in the central equation of Equation (3) for estimating testing effort. In Table 1, if the code is not yet a production release (before General Availability, or GA), the value of this parameter is 0.5. If the code is in production over three months, it is treated as a mature software application and thus the parameter's value is 0.3. The classification of the test cases complexity and hierarchy relies more on expert judgment and historical data than on mathematical expression.

\[
\begin{array}{|c|c|c|c|}
\hline
\text{RM} & \text{Code} & <\text{GA} & <\text{GA 3 months} & \text{Mature} \\
\hline
\text{RM}1 & <6 months & 0.5 & 0.4 & 0.3 \\
\text{RM}2 & <12 months & 0.5 & 0.4 & 0.3 \\
\text{RM}3 & <6 months & 0.5 & 0.4 & 0.3 \\
\hline
\text{TCC} & \text{DBA/SA} & \text{Max} & \text{Medium} & \text{Less} \\
\text{TCC}1 & <6 months & 0.5 & 0.4 & 0.3 \\
\text{TCC}2 & <12 months & 0.5 & 0.4 & 0.3 \\
\hline
\end{array}
\]

However, it is not difficult to distinguish simple from complex (or maximum complexity from less complexity). Nevertheless, in an ideal world of software development, the Test Cases Complexity is a combination of either a maximum test cases complexity plus a mix hierarchy, or a medium complexity plus a complicated hierarchy. Therefore, by substituting some of the ideal world values in Table 1 into Equation (3), we have an optimal estimation model:

\[
E_{\text{optimal}} = \alpha (\text{Size})(0.5+0.3+0.3)(0.5+0.4) = 0.99\alpha (\text{Size})
\]

(4)

By substituting the values in Table 1 into Equation (3), one can derive upper- and lower-bound of the effort estimates:

\[
0.54 E_{\text{optimal}} \leq E_{\text{estimate}} \leq 1.5 E_{\text{optimal}}
\]

(5)

In other words, in the worst case, no more than 50% of deviation can be derived from the estimates, regardless who or how the estimates are performed. Moreover, with the aid of test graphs discussed in the Section 5, one can derive from the Equation (3) an estimate of less deviation. After the estimate of testing effort is derived, a cost estimate can be performed according to the unit cost from the historical data or from the software development organization. The schedule can be also estimated based on the effort estimation.
5. TEST GRAPH AND TEST CASES
ANALYSIS

5.1 Test Graph

A test graph is a weighted Directed Acyclic Graph (DAG) \( G=(V,E,\mu,\lambda) \), that consists of nodes (test cases) \( v \in V \) whose cost function (weight) is \( \mu(u) \) and arcs (dependency) \( (u,v) \in E \) whose cost function is \( \lambda(u,v) \) [4, 9]. This cost function is an estimated effort needed to execute the test case. The cost function can be a real measurement unit (e.g., days, hours … etc) or a logical measurement unit (e.g., 5 for a complicated test case, 3 for a medium complexity, and 1 for a low complexity).

Based on the test cases that will be executed during the test, one can derive a test graph to depict the dependency (arcs) between test cases (nodes). With the test graph, a testing coordinator can determine on how many testers (if available) are required to complete the test within the shortest timeframe, by identifying the degree of parallelism and makespan on the test graph. The degree of parallelism is the maximum number of task threads that can be executed in parallel. The makespan of a test graph is the length of the critical path (\( CP \)) in the test graph, i.e., the path with the maximum sum of node and arc weights.

A test graph can be very simple and consist of only nodes and arcs that have no cost function associating with them. Figure 1 shows an example of such simple test graph.

![Figure 1. An example of simple test graph](image)

There are nine test cases of which each test case needs the same amount of time to execute. Test cases T1 and T2 have no dependency. Therefore, they can be executed simultaneously with two testers. However, test cases T3 and T4 can be not started until test case T1 is finished. Furthermore, test case T5 has to wait for completion of test cases T3 and T4.

The degree of parallelism of Figure 1 is 3, and the makespan is 4. In other words, we may divide the test cases into 3 subset \{T1, T4, T6\}, \{T2, T7, T9\} and \{T3, T5, T8\}, and assign each subset of test cases to a tester. The time needed to complete the test is 4 time units which is equivalent to the time needed to complete test cases T1, T4, T6, and T8 in the critical path of the test graph.

Assuming that each test case needs one business day on average to execute, with 3 testers and the assignment mentioned above, we then use the Equation (3) to compute the total effort and schedule. By applying the Equation (3) to each tester, the best case for the test takes 2.16 business days, while the worst case takes 6 business days to complete.

5.2 Test Graph with Cost Functions

A more accurate model of the test graph is that there are not only nodes and arcs in the test graph, but a cost function may be also associated with each node and arc. A cost function that is associated with a node (or an arc) is an estimates derived from the parametric model described in the Section 4. With this graphic representation, we may say that, in the previous example in Figure 1, the cost function is 1 for each node and there is no delay in passing information to next test case(s).

![Figure 2. A test graph with cost function associated with each node](image)

If each test case requires different amount of time to test, and the dependencies do not incur any delays, we only need to associate cost function (amount of time) to each of the nodes as shown in Figure 2.

From time to time, delays do occur between test cases executions. In these cases, estimates on such delays can be considered as cost functions associated with arcs that link test cases (nodes).

As we can see in Figure 3, delay may not occur to every arc. If there is no delay between the executions of two test cases, the cost function associated with the arc shall be zero. Otherwise, estimates obtained by using the parametric model are entered for the arcs.

With cost functions associated with both nodes and arcs, the degree of parallelism and the makespan of the test graph in Figure 3 are 3 and 38 time units (in bold links, \{T1, T4, T6, T8\}), respectively. That is, 3 testers is the maximum number of testers needed for the shortest period of test time. Assigning less than 3 testers will increase the test period. On the other hands, assigning
more than 3 tests into the test will not shorten further the test period as the critical path is 38 time units.

Figure 3. A test graph with cost functions associated with both nodes and arcs

Comparing Figure 2 and Figure 3, we also notice that, when taking into account of both cost functions of nodes and arcs, the critical path might be changed. It in turns may change the makespan of the test graph. However, the degree of parallelism should not be changed.

5.3 Test cases and Requirements Mapping

Test cases are usually created based on requirements. To have at least one-to-one mapping of test case and requirement is essential to any experienced testers. However, it is very common to have missing link between requirements and test cases, especially with requirements creep.

By using the concept of test graph, we can produce a primary node for every requirement. Any requirement changes will alter the status of that primary node, and hence remind the testers to review and update the test case(s). Newly added requirements at any time will produce unattended primary nodes (highlighted in gray) in the test graph and alert test to add new test cases.

Let us exam the concept with an example. Assuming a telecommunication network management software application has 4 requirements of major enhancements for current release. They are:

- R1: Adding a new network equipment DAX into inventory that can be used to create an OC3 network service.
- R2: Adding a new ADM equipment for OC12 service and can also provide OC12 time slots.
- R3: Adding a GEN equipment that can provide an OC48 service and produce OC12 and OC3 time slots.
- R4: Adding a DWDM equipment that has OC48 and OC3 timeslots.

Also assume that while in the test phase, the Requirements R3 got updated with new scope and a new feature that was never considered before is now required for the release.

- R3 (update): Adding a GEN equipment can provide an OC48 service and produce OC12 and OC3 time slots. A new DS3 card can also be plugged into the equipment to create a DS3 time slots.
- R5 (new): Adding a new OC12 service that can be used in Optical Ring with 1+1 protection.

Figure 2 shows the test graph that has embedded with requirements traceability for those 4 requirements, along with requirements change in the Requirement R3 and a newly added Requirement R5.

Each primary node (T1.0, T2.0 ...etc.) representing a test case that links to the original requirement. In this case, test case T1.0 is to create the new network equipment DAX in the inventory, T2.0 is to add the ADM in the inventory … and so on. In Figure 2, after T1.0 that adds the DAX in the inventory, T1.1 is to create an OC48 service with other network equipments and produce OC12 time slots. And the time slots of that OC12 can then be used to create an OC12 service in T1.2. Finally an OC3 service along with this DAX is created in T1.3.

Figure 4. A test graph with requirements traceability

Assuming after T2.0, we can use one of the time slots created in T1.1 for T2.1, or one time slot from T3.1, inT2.1. Now, depending on the requirements R2, we may need to have two separate test cases for each of the two cases mentioned above after T2.0.

Since R3 is updated during the test phase, the test case T3.0 is now highlighted in gray to indicate that the test cases following T3.0 need to be reviewed to reflect
the requirements change. Until they are review and updated accordingly, the T3.0 remains grayed. T5.0 is alone and also highlighted in gray because it is a new requirement that has yet linked to any test cases.

With the graphical automated interface, System Analysts can enter their requirements into the tool and the interface will generate a primary node automatically. If testers would like to create some test cases that are beyond the requirements, they can create their own test requirements to generate primary nodes.

If color code is used in the graphical automated interface, the following color code convention in project management can be used:

- Red: In jeopardy or need immediate action.
- Yellow: Require attention.
- Green: In progress.
- Blue: Completed.

Hence, those updated or new requirements will be highlighted in red. And the color code of every test case is reflecting its current status.

6. CONCLUSION

A parametric estimation model for estimating testing effort and a test graph to aid in test cases analysis and requirements traceability are presented in this paper. The results of applying the techniques to actual software development projects are also shown. The deviations of the estimates that result from using the techniques are within about 10%. In most of the cases, the deviations are within about 4%.

7. ACKNOWLEDGMENTS

The author would like to thank Su-Chiou Tsay, and Usha Bala of AT&T and Michelle Li of IBM for their assistance in performing some of the estimates on testing effort using the parametric model and test graph.

8. REFERENCES


