QASymphony

Test Case Point Analysis

Vu Nguyen
QASymphony, LLC.(www.qasymphony.com)

Abstract
Software testing is a crucial activity in software application development lifecycle to ensuring quality, applicability, and usefulness of software products. It consumes a large amount of effort of the development team to achieve the software quality goal. Thus, one critical success-factor in software testing is to estimate and measure software testing accurately as a part of the software quality assurance management. This white paper proposes an approach, namely *Test Case Point Analysis*, to estimating the size and effort of software testing work. The approach measures the size of software test case based on its checkpoints, precondition and test data, and types of test. The testing effort is computed using the Test Case Point count of the testing activities.

Keywords: test case point, software test estimation, software testing, quality assurance, test management

I. INTRODUCTION

Software testing is a crucial activity in software application development lifecycle to ensuring quality, applicability, and usefulness of software products. No software can be released without a reasonable amount of testing involved. To achieve the quality goal, software project teams have been reported to devoting a substantial portion of the total effort to perform the software testing activity. According to the previous industry reports, software testing consumes about 10-25% of the total project effort, and on some projects this number may reach 50% [1].

Thus, one factor to ensure the success of software testing is the ability to provide reasonably accurate size and effort estimates for testing activities. Size and effort estimates are used as key inputs for making investment decisions, planning, scheduling, project bidding, and deriving other metrics such as productivity and defect density for project monitoring and controlling. If the estimates are not realistic, then these activities effectively use the misleading information, and the consequences could be disastrous.

Unfortunately, providing reliable size and effort estimates for software testing is challenging. One reason is that software testing is a human-intensive activity that is affected by a large number of factors such as personnel capabilities, software requirements, processes, environments, communications, and technologies. These factors cause high variance in productivities of projects of different characteristics. Another reason is that software quality attributes such as functionality, reliability, usability, efficiency, and scalability are hard to quantify objectively. Moreover, there are many different types of software testing and testing contexts even in one project. Because of the diversity of software testing activities, one approach that works well on one type of test in one environment may not work when it is applied to another type of test in another context.

Although many approaches have been introduced and applied to estimating overall software size and effort, there is a lack of methods for estimating software testing size and effort. Often, testing effort is estimated as a part of the overall software effort, and software size is used as a size measure of software testing activity. This approach, however, cannot account for specific requirements of software testing, such as intensive testing required by clients. Another limitation is that it cannot be applied to the test-only project in which the software exists but the testing team does not have access to source code or requirements in order to perform Function Point Analysis or count Source Lines of Code (SLOC). Finally, the software size in Function Points or

SLOC may not well reflect the amount of test performed by the testing team. Therefore, measuring the productivity of software testing using these metrics is misleading.

In an attempt to address this issue, this white paper presents an approach to estimating the size and effort of software testing. The sizing method is called Test Case Point. As the name indicates, the method measures the size of test case, the core artifact that testers create and use when performing software test execution. The size of test case is evaluated using four elements of test case complexity, including *checkpoint*, *precondition*, *data*, and *type* of the test case. This article also describes simple approaches to determining testing effort using Test Case Point.

Although theoretically this Test Case Point analysis can be applied to measuring the size of different kinds of software testing given test cases as input, this method is only focused on estimating the size of system, integration, and acceptance testing activities which are often performed manually by independent verification and validation (IV&V). Unit testing, automation testing, and performance testing that involve using test scripts are also beyond the scope of this method.

II. A SUMMARY SOFTWARE TESTING SIZE AND EFFORT ESTIMATION METHODS

A number of methods and metrics have been proposed and applied to estimating the size and effort of software projects. Many of these methods are also used to estimate the effort of software testing activities. SLOC is a traditional and popular metric that measures the size of software product by counting its source program. A number of SLOC definitions exist, but the most common are physical SLOC which counts the number of physical lines of code and logical SLOC which counts the number of logical statements in the source program. SLOC is used as a size input for popular software estimation models such as COCOMO, SEER-SIM, and SLIM. In these models, the testing effort is computed the overall effort estimate using a predefined percentage. This percentage can be obtained from industry average, historical data, or based on experience of the estimators.

Function Point Analysis (FPA) is another method for estimating the size of software that has been used in industry since the late 1970s [3]. This analysis measures the software size based on five components including inputs, outputs, inquiries, files, and interfaces. The size output, whose unit is Function Point, is then adjusted with technical factors to take into account technical characteristics of the system. To estimate effort, the estimated size in Function Point can be determined using a productivity index, by converting to SLOC using a method called *backfire*, or by performing a simple linear regression model. FPA has been extended and introduced in a number of sizing methods including Mark II [11], COSTMIC FFP [12], and NESMA FPA [6].

Although FPA is widely used in industry, it has been criticized for its complexity that required much time and skilled estimators to perform the analysis. A similar but simplified method named Use Case Point was proposed [7]. This method counts the number Use Case Points of the software project by measuring the complexity of its use cases. The use case complexity is based on actors and transactions of the use case and then adjusted with the technical complexity and the environment factors to obtain the final Use Case Point count. The idea of FPA and UCP methods has inspired the introduction of Test Case Point Analysis [4] and other methods from the practice and research community [2, 5, 8, 9, 10, 13]. In [8], Aranha and Borba proposed a model for estimating test execution effort by measuring the size and execution complexity of test cases. They introduced the size unit of *execution point* which is based on the characteristics of every step in the test case. The effort is then estimated by converting the execution point count to effort using a conversion factor.

III. TEST CASE POINT ANALYSIS

1. Method Overview

The Test Case Point measures the software testing size, reflecting the complexity of the testing activities performed to ensure the quality goal of the software. As a complexity measure, it needs to reflect the effort required to perform the testing activities including planning, designing, executing tests, and reporting and tracking defects.

The Test Case Point Analysis uses test cases as input and generates Test Case Point count for the test cases being measured. The complexity of the test case is based on four elements including checkpoint, precondition, test data, and types of test case, which effectively assumes that the complexity is centered at these four elements. These elements are classified into two types, one reflecting the largeness of the test case, which includes checkpoint, precondition, test data, and the other normalizing the complexity of different types of test by adjusting the Test Case Point by weights of test types.

2. Method Detail

2.1. Measure Test Case Complexity

Each test case is assigned a number of Test Case Points based on the number of checkpoints, the complexity of precondition, and test data used in the test case.

Checkpoint is the condition in which the tester verifies whether the result produced by the target function matches the expected criterion. One test case consists of one or many checkpoints.

Rule 1: One checkpoint is counted as one Test Case Point.

Precondition. Test case's precondition specifies the condition to execute the test case. The precondition mainly affects the cost to execute the test case, same as that of the test data. Some precondition may be related to data prepared for the test case.

Table 1. Precondition Complexity Level Description

Complexity level	Description
None	The precondition is not applicable or important to execute the test case. Or, the precondition is just reused from the previous test case to continue the current test case.
Low	The condition for executing the test case is available with some simple modifications required. Or, some simple setting-up steps are needed.
Medium	Some explicit preparations are needed to execute the test case. The condition for executing is available with some additional modifications required. Or, some additional setting-up steps are needed.
High	Heavy hardware and/or software configurations are needed to execute the test case.

The complexity of test case's precondition is classified into four levels, None, Low, Medium, and High (see Table 1).

Test data is used to execute the test case. It can be generated at the test case execution time, or is already prepared by previous tests, or is generated by testing scripts. Test data is test case specific or general to a group of test cases or the whole system. In the latter cases, the data can be reused in multiple test cases.

The complexity of test data is classified into four levels, None, Low, Medium, and High, which is shown in Table 2.

Table 2. Test Data Complexity Level Description

Complexity level	Description
None	No test data preparation is needed.
Low	Test data is needed, but it is simple so that it can be created during the test case execution time. Or, the test case uses a slightly modified version of the existing test data, i.e., little effort required to modify the test data.
Medium	Test data is deliberately prepared in advance with extra effort to ensure its completeness, comprehensiveness, and consistency.
High	Test data is prepared in advance with considerable effort to ensure its completeness, comprehensiveness, and consistency or by using support tools to generate and database to store and manage. Scripts may be required to generate test data.

In many cases, the test data has to be supplied by a third-party. In such cases, the effort to generate test data for the test case is expectedly small, thus, the test data complexity should be low.

Rule 2: each complexity level of precondition and test data is assigned a number of Test Case Points. This measure is called unadjusted Test Case Point.

Table 3. Test Case Point Allocation for Precondition

Complexity level	Number of Test Case Point	Standard deviation (±σ)
None	0	0
Low	1	0
Medium	3	0.5
High	5	1

Table 3 and 4 show the number of Test Case Points allocated to each complexity level for the precondition and test data components of the test case. These constants were obtained through a survey of 18 experienced quality assurance engineers in our organization. The standard deviation

values reflect the variance among the survey's results. It is worthy to note that the estimator should adjust these constants to better reflect the characteristics of their projects and environments.

Table 4. Test Case Point Allocation for Test Data

Complexity level	Number of Test Case Point	Standard deviation (±σ)
None	0	0
Low	1	0
Medium	3	0.6
High	6	1.3

2.2. Adjust Test Case Point by Type of Test

To account for the complexity differences among test types, the Test Case Point counted for each test case is adjusted using a weight corresponding to its test type. The weight of the most common test type, *user interface and functional testing*, is established as the baseline and fixed at 1.0. The weights of other test types are relative to this baseline.

Table 5 shows the types of test, the corresponding weights, and standard deviations of the weights. The weights were obtained via the same survey of 18 experienced quality assurance engineers in our organization. Again, the estimators can review and adjust these weights when applying the method in their organizations.

Table 5. Weights of Test Types

Type of Test	Weight (W)	Standard deviation (±σ)	Comments
User interface and functional testing	1.0	0	User interface and functional testing is considered baseline.
API	1.22	0.32	API testing verifies the accuracy of the interfaces in providing services.
Database	1.36	0.26	Testing the accuracy of database scripts, data integrity and/or data migration.
Security	1.39	0.28	Testing how well the system sustains from hacking attacks, unauthorized and unauthenticated access.
Installation	1.09	0.06	Testing of full, partial, or upgrade install/uninstall processes of the software.

Type of Test	Weight (W)	Standard deviation (±σ)	Comments
Networking	1.27	0.29	Testing the communications among entities via networks.
Algorithm and computation	1.38	0.27	Verifying algorithms and computations designed and implemented in the system.
Usability testing	1.12	0.13	Testing the friendliness, ease of use, and other usability attributes of the system.
Performance (manual)	1.33	0.27	Verifying whether the system meets performance requirements, assuming that the test is done manually.
Recovery testing	1.07	0.14	Recovery testing verifies the accuracy of the recovery process to recover from system crashes and other errors.
Compatibility testing	1.01	0.03	Testing whether the software is compatible with other elements of a system with which it should operate, e.g. browsers, Operating Systems, or hardware.

The total Adjusted Test Case Point (*ATCP*) is counted as

$$ATCP = \sum(UTCP_i * W_i) \qquad (Eq\ 3.1)$$

- Where $UTCP_i$ is the number of Unadjusted Test Case Points counted using checkpoints, precondition, and test data for the test case i*th*.
- W_i is the weight of the test case i*th*, taking into account its test type.

It is important to note that the list of test types is not exhaustive. Estimators can supplement it with additional test types and the corresponding weights based on the weight of the user interface and functional testing.

IV. EFFORT ESTIMATION

Testing activities can be classified into four categories, test planning, test design, test execution, and defect reporting. Of these activities, test execution and defect reporting may be performed multiple times for a single test case during project. However, the size measured in Test Case Point takes into account all of these activities, assuming that each activity is performed once. The distribution of effort of these activities allows estimating the effort in case the test execution and defect reporting activities are performed more than once. The distribution of testing effort can be generated using historical data.

The distribution of effort of testing phases shown in Table 6 was obtained from the same survey performed to collect the constants described above. Again, these values mainly reflect the

experience in our organization, and thus, the estimators are encouraged to adjust these numbers using their own experience or historical data.

Table 6. Testing Effort Distribution

Testing Phase	Description	Percent of Effort	Standard deviation (±σ)
Test Planning	Identifying test scope, test approaches, and associated risks. Test planning also identifies resource needs (personnel, software, hardware) and preliminary schedule for the testing activities.	10%	3%
Test Analysis and Design	Dealing with detailing test scope/objectives, evaluating/clarifying test requirements, designing the test cases and preparing necessary environments for the testing activities. This phase needs to ensure combinations of test are covered.	25%	5%
Test Execution	Is the repeating phase dealing with executing test activities and objectives outlined in the test planning phase and reporting test results, defects, and risks. This phase also includes test results analysis activities to check the actual output versus the expected results.	47%	4%
Defect Tracking & Reporting	Is the repeating phase dealing with tracking the progress and status of tests and defects and evaluating current status versus exit criteria and goals	18%	3%

Dependent on the availability of information and resources, the testing effort can be estimated using one of the following simple methods:

- Productivity Index
- Simple Linear Regression Analysis

Estimate Effort Using Productivity Index

The Productivity Index is measured as the number of Test Case Points completed per person-month. The Productivity Index is then used to compute the testing effort by multiplying it with the total Adjusted Test Case Point (see Eq. 4.1).

The Productivity Index can be determined using historical data. It is recommended to use the Productivity Index of similar projects with the project being estimated. The testing effort is computed as

$$Effort = TCP * Productivity\ Index \qquad (Eq.\ 4.1)$$

Estimate Effort Using Simple Linear Regression Analysis

Regression is a more comprehensive and expectedly a more accurate method to estimate the testing effort given that many historical data points are available. With this method, the estimators can compute estimation ranges and confidence level of the estimate. This method requires having at least three similar projects for providing meaningful estimates.

Given N completed projects in which the ith project has the number of Test Case Point (TCP_i) and effort (E_i) in person-month, the simple linear regression model is presented as

$$E_i = \alpha + \beta * TCP_i + \varepsilon_i \qquad \textit{(Eq. 4.2)}$$

Where, A and B are coefficients and ε_i is the error term of the model. Using the least squares regression one can easily compute the estimates A and B of the coefficients α and β, respectively. The A and B values are then used in the model for estimating the effort of new testing projects as follows

$$E = A + B*TCP \qquad \textit{(Eq. 4.3)}$$

Where, TCP is the size of the test cases in Test Case Point, and E is the estimated effort in person-month of the project being estimated.

V. CONCLUSION

Software testing plays an important role in the success of software development and maintenance projects. Estimating testing effort accurately is a key step towards to that goal. In an attempt to fill the gap in estimating software testing, this paper has proposed a method called Test Case Point Analysis to measure and computing the size and effort of software testing activities. The input of this analysis is test cases, and the output is the number of Test Case Points for the test cases being counted.

An advantageous feature of this approach is that it measures the complexity of test case, the main work product that the tester produces and uses for test execution. Thus, it better reflects the effort that the tester spends on their activities. Another advantage is that the analysis can be performed easily by counting the number of checkpoints, measuring the complexity of precondition and test data, and determining the type of each test case. However, there are several limitations of this approach, hence suggesting directions for future improvements of the approach. One limitation is that the Test Case Point measure has not been empirically validated. Data of the past testing projects need to be used to validate the effect and usefulness of the size measure in estimating the effort of the software testing activities. Another limitation is the concern of whether the complexity ranges of the test case's precondition and test data can properly reflect the actual complexity of these attributes. Future improvements on the method need to address these limitations.

VI. REFERENCES

[1] Y. Yang, Q. Li, M. Li, Q. Wang, An empirical analysis on distribution patterns of software maintenance effort, International Conference on Software Maintenance, 2008, pp. 456-459

[2] S. Kumar, "Understanding TESTING ESTIMATION using Use Case Metrics", Web: http://www.stickyminds.com/sitewide.asp?ObjectId=15698&Function=edetail&ObjectType=ART

[3] A.J. Albrecht, "Measuring Application Development Productivity," Proc. IBM Applications Development Symp., SHARE-Guide, pp. 83-92, 1979

[4] TestingQA.com, “Test Case Point Analysis”, Web: http://testingqa.com/test-case-point-analysis/

[5] TestingQA.com, “Test Effort Estimation”, Web: http://testingqa.com/test-effort-estimation/

[6] NESMA (2008), "FPA according to NESMA and IFPUG; the present situation", http://www.nesma.nl/download/artikelen/FPA%20according%20to%20NESMA%20and%20IFPUG%20-%20the%20present%20situation%20(vs%202008-07-01).pdf

[7] G. Karner, “Metrics for Objectory”. Diploma thesis, University of Linköping, Sweden. No. LiTHIDA-Ex-9344:21. December 1993.

[8] E. Aranha and P. Borba. “An estimation model for test execution effort”. In Proceedings of First International Symposium on Empirical Software Engineering and Measurement, 2007.

[9] E. R. C. de Almeida, B. T. de Abreu, and R. Moraes. “An Alternative Approach to Test Effort Estimation Based on Use Cases”. In Proceedings of International Conference on Software Testing, Verification, and Validation, 2009.

[10] S. Nageswaran. “Test effort estimation using use case points”. In Proceedings of 14th International Internet & Software Quality Week 2001, 2001

[11] C.R. Symons, "Software Sizing and Estimating: Mk II FPA". Chichester, England: John Wiley, 1991.

[12] A. Abran, D. St-Pierre, M. Maya, J.M. Desharnais, "Full function points for embedded and real-time software", Proceedings of the UKSMA Fall Conference, London, UK, 14, 1998

[13] D.G. Silva, B.T. Abreu, M. Jino, A Simple Approach for Estimation of Execution Effort of Functional Test Cases, In Proceedings of 2009 International Conference on Software Testing Verification and Validation, 2009