Test Effort Estimation

Murali Chemuturi

I Introduction

Testing is carried out primarily for unearthing any defects present in the system and to prevent a defective product reaching the customers. Secondarily, testing is also carried out to convince the customer that the product conforms to and fulfills the specifications and the functionality agreed to.

Software Testing is recognized as a very important activity in software development. Of late, the importance of independent testing – that is testing by persons not involved in the development of software – is increasingly being recognized, so much so, software companies specialized only in testing are established and are doing good business. Also significant is the fact that as the complexity and size of the software increased multi-fold – so has the complexity of testing as well as the types of testing. As a result, there is a paradigm shift in the estimation of testing effort from being taken as a percentage of development effort, to independent size estimation and effort estimation. Practitioners have come out with a size measure called "Test Points". To the extent that I saw, the Test Points address the "Black Box" testing as used in Integration, System and Acceptance testing activities.

This paper intends to throw light on the complete testing activity and suggest a few solutions to estimation of testing effort.

II Types of Testing being carried out in the software industry

There are basically two techniques of testing

1. White Box
2. Box Testing

White Box testing involves stepping thru every line of code, and every branch in the code. In Black Box testing, a set of inputs is given and the outputs are compared with the expected outputs.

There are three ways of classifying the software testing

1. Product Stage based Testing, namely, Unit, Integration and Systems Testing
2. Purpose based testing, namely, Functional, Positive and Negative Testing
3. Reliability Testing, namely, Load, parallel, Concurrent, Stress Testing

It is also common that every testing produces some defects and it needs to be followed by Regression Testing to ensure that the defects that were unearthed were indeed rectified and the process of rectification of defects did not produce further defects.

Other, not so common types of testing are –

1. Usability Testing – to determine how easy/difficult it is to use the software
2. Installation testing - on all intended platforms
3. Troubleshooting/Recovery testing

Unit testing is carried out for every Unit of the software, a form, report etc. Unit testing is normally white box testing.

Integration testing is carried out – normally using the black box testing – for a stand-alone module of the software product being developed.

System testing is carried out for the software product on the target systems.

Functional testing is carried out to ensure that all functions intended for the software are available and are working satisfactorily with out any inaccuracies.

Positive testing is carried to prove that the software works as claimed – normally used in Acceptance testing before delivery to the clients.

Negative testing is carried out to unearth any possibility of the software failing due to un-intended usage of the software – very important in software products intended for consumer market.

Load testing is carried out, especially in web-based applications, to ensure that the software works well when the expected numbers of users use the software using different modules of the software. This unearths the issues connected with the bandwidth, database, sufficiency of RAM, hard disk etc.

Parallel testing is carried out to unearth issues if any, when a number of users same functionality in simultaneously.

Concurrent testing is carried out to unearth issues when two or more users use the same functionality and update or modify same data with different values at the same time.

Stress testing is carried out by causing stress – by making unavailable some of the system resources - to the system to unearth issues connected with unexpected events in the operation of the system.

It is rare that White box testing is resorted to in testing other than unit testing. It is not uncommon to use Black Box testing even in Unit testing too.

It is also rare that all the above types of testing are carried out for every project that is executed in the organization.

Organizations carry out some combination of the types of testing described above. Normally every organization conducts the following types of testing –

1. Functional Testing to ensure that all the functionalities allocated to the software are working and there are no inaccuracies, when used properly
2. Integration testing – to ensure that the coupling between various software modules is in order
3. Positive testing to get the software accepted by the client
4. Load testing to ensure that the system does not crash when heavy loads are placed on it

Organizations carry out the remaining types of testing, on a "if available" – time and budget – basis or "if mandated" basis.

III The "How" of testing

When we come to methodology of testing, we find –

1. Test Cases based testing – there is a set of test cases and testing is carried out only against the test cases
2. Intuitive testing – there may be a general description of the functionality and suggestions/guidelines as to how to go about unearthing defects. Testing is carried out using the experience and intuition of the tester. Some amount of creativity or common sense is expected from the tester.

For any software project there would always be a high-level test plan. For every intended testing, there ought to be set of test cases against which testing is carried out. But consider the implications –

1. For every numeric (including date type data) data input, we need to have five test cases – using the Partitioning and Boundary Value Analysis techniques –
    a. One value in the acceptable range
    b. One value above acceptable range
    c. One value below acceptable range
    d. One value at the upper boundary of the acceptable range
    e. One value at the lower boundary of the acceptable range
2. Size checks for all non-numeric data, one per every data item
3. Logical testing for presence of invalid data – like numerics and special characters in name data fields etc.

Thus, the test case set for even a moderately complex unit will be voluminous. Modern projects are large in size and the effort required to prepare exhaustive test case set would be significantly high. Therefore, it is common (not always, perhaps) to prepare test cases where it is expected that the tester cannot intuitively figure out the test cases all by him/her self.

It is not uncommon that unit testing is carried out without any test cases. Integration testing, system testing and acceptance testing are normally carried out against test cases.

IV Issues in Sizing the Testing

When we attempt to specify a Unit of Measure, there must be a clear definition of what is included in it. Secondly, there must be a means to measure the size. Then there must be some uniformity – need not be identical – in the practices of testing, namely, preparation of test plans, the comprehensiveness of test cases, the types of testing carried out and availability of empirical data to normalize various situations to a common measure. The following aspects need consideration –

1. Type of application – Standalone, Client-Server, Web-Based
2. Type of testing – White Box or Black Box
3. Stage of testing – Unit, Integration, System
4. Purpose of testing – Reliability, client-acceptance, ensure-functionality
5. Test case coverage – how much is covered by test cases and how much is left to the intuition of the tester
6. Definition of the granularity test case – one input field is tested with five input values – is it one test case or five test cases?
7. The size of the test cases at the levels of unit, integration and system vary – we need a normalization factor to bring them all to a common size
8. The impact of the usage of tools and the effort needed to program them
9. Environmental factors – the tester experience and knowledge, complexity of the application, resources (time and budget) allowed for testing, existence of a clean testing environment etc. and their impact on testing.
10. Existence of the practice of code walkthrough before testing software, in the organization

The literature I have seen so far does not suggest that all these aspects are well considered and covered in defining the size measure for testing effort.

One question – Is size measure necessary to estimate testing effort? No – testing effort can be estimated using Task Based Estimation.

But size measure is important so that comparison can be made between tow projects; it is important to assess the reasonableness of the effort estimates.

V Who Needs Test Estimation?

There are four categories of people who would like to carry out test estimation –

1. The project Team who obviously have to carry out testing to ensure that their work is satisfactory and they can certify it so and pass it on to professional

independent testing team. This set would carry out all the tests described in section II

2. The in-house Software Quality Assurance (SQA) Team that would test and certify the product for customers. This team also would carry out all the tests described in section II. Their objective is to unearth all defects and make the product as defect proof as possible while ensuring that the product meets all the specified functionality.
3. Testing Organizations, whose mainline of business is to test other's software and certify the products. Their objective is to ensure that the product meets the end user expectations. This set would carry out –
    a. Mainly Black Box testing
    b. Functional Testing
    c. System Testing
    d. Negative Testing
    e. Regression Testing
4. Customers who entrusted their software development to a vendor. Their objective is to ensure that they are getting what they are paying for. This set would carry out
    a. Acceptance Testing / Positive Testing
    b. Regression Testing

The type of test estimation carried out by these sets would be different.

1. Project Team – since they carry out testing as part of development work – it may be taken as a percentage of overall development work. Their purpose would be mainly to estimate resource requirement and meeting the delivery schedule
2. In-house SQA Team – they need to estimate mainly for the purpose of resource estimation and meet the delivery schedule
3. Testing Organizations – they need to estimate so that they can offer a fixed price to the customers and manage a healthy profit for their organizations besides resource estimation and meeting delivery schedule. For them "Size" of testing project would have significant appeal.
4. Customers – They really are supported in this activity of acceptance testing by the vendor organization.

Therefore, the tool for testing estimation, needs to address two different needs –

1. For the in-house SQA team
2. For Testing Organizations

VI Sizing and Effort Estimation

The term "Test Points" is catchy and perhaps the Unit of Measure for the estimation of Testing size and effort. This can be used for sizing testing projects. Test Points can be extracted from the software size estimates.

Task Based Effort Estimation technique can be used for estimation of effort for testing projects.

The size of software has a direct bearing on the testing effort. The exception to this, perhaps, is processing intensive scientific applications. For example, testing a Login screen would take less effort than registration screen.

We can also conclude that the testing effort is also dependent on the number of testing types included in the testing project.

What constitutes a "**Test point**"?

Test Point as a size measure for measuring the size of a software-testing project and that a **Test Point** is equivalent to a normalized test case.

It is common knowledge that test cases differ widely in terms of complexity and the activities necessary to execute it. Therefore, the test cases need to be normalized - just the way Function Points are normalized to one common measure using weighting factors. Now there are no uniformly agreed measures of normalizing the test cases to a common size. Also, what is the relation between other software size measures like Function Points, or Use Case Points etc? Would it be fair to say one Adjusted Function Point would result in one normalized Test Point? - again no agreement. Perhaps, we may say that one Adjusted Function Point results in one (or 1.2 or 1.3 etc.) Test Points depending on the software application at hand. Similarly, we may derive a similar relation in case of other software size measures too.

There are many types of testing carried on software. Is there a standard saying that these are the tests that should be included in a Testing Projects? I am afraid - that there is no agreement here either. Generally - not necessarily - a Testing Project would include Integration Testing, System Testing and Acceptance Testing - all using the black box testing technique.

But the reality could be different.

The variety in applications - on which testing depends - is significantly large. The method for normalization between various application types is not commonly agreed to.

The types of testing carried out varies from project to project. There is no uniformly agreed to the types of testing to be carried out on any given project.

There is barely enough research and empirical data that strict guidelines can be drawn as the profession of testing itself is very nascent.

However, we may estimate Test Points converting the size estimate using a set of conversion factors into test points and adjust the Test Point size using various weights.

Weights

The following weights may be considered -

1. Application weight
2. Programming language Weight
3. Weights for each of the types of testing, namely,
    a. Unit Testing
    b. Integration Testing
    c. System Testing
    d. Acceptance Testing (Positive Testing)
    e. Load Testing
    f. Parallel Testing
    g. Stress Testing
    h. End to End Testing
    i. Functional Testing
    j. Negative Testing

All weights are project specific, and are applied only when selected.

Test Point has a weight 1, when the combined weights of the three tests, namely, Integration, System, and Acceptance tests - is equal to one. When other tests are added to the project, their weights may be assigned.

We have to compile weights-data for all these tests, and validate them in-house by comparing the estimates with actual data and conducting a causal analysis of the variance.

Tools usage is expected to reduce the total time of testing. Perhaps true - but it really depends on the tool itself. Hence, the weight for the tools usage may be set appropriately. A weight of 1 indicates that tools usage is indifferent to the testing size - more than 1, the tool increases the size and hence the effort - less than 1, the tool reduces the size and hence the effort.

We need to add another weight for the development language in projects that envisage independent Unit Testing, that is, a person, other than the person who developed the program, carries out Unit Testing. The weight for the language comes into effect only when Unit Testing is selected - the reasons being –

1. Unit Testing is white box, normally
2. White box testing is conducted from within the code

If Unit Testing were not selected, this weight need not be used.

The Method of computing Test Points

1. Use an existing development size & effort estimation
2. Convert development project size into Unadjusted Test Points (UTP) using a conversion factor which is based on the application type
3. Compute a Composite Weightage Factor (CWF)
    a. Sum up all individual weights of selected tests
    b. Multiply it by the weight of the application weight
    c. Multiply it by the language weight if Unit Testing is selected
    d. Multiply it by Tools Weight if Tools Usage is selected
4. UTP are multiplied by CWF to obtain the testing size in Test Points size
5. The Productivity Factor indicates the number of Test Points that can be covered by one test engineer.
6. Testing Effort in Person Days is computed by dividing Test Point Size by the Productivity Factor.

This methodology is implemented in EstimatorPal. Interested persons can download a fully-functional but time-limited to 30 days demo from effortestimator.com web site.