

Types of Test Effort Estimation

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

As testing is so varied and diverse, it is not possible to directly jump into discussion about Test Effort Estimation without a brief discussion about the topic of Testing itself.

Testing Basics

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. To use this technique, the tester should be knowledgeable about the programming language and should know the structure of the program.

In Black Box testing, a set of inputs is given to the software and the outputs delivered by the software are compared with the expected outputs. To use this techniques, the tester should have knowledgeable about the functionality of the system and should be able to use the computer.

Testing Scenarios

Software testing as stated above is carried in two independent scenarios –

1. Project Testing or Embedded Testing – that is testing, which is carried out as part of a software development project – this is carried out to ensure that the development work is defect-free.
2. Product Testing – testing that is carried out for a COTS (Commercial Off-The-Shelf) software product. This is to ensure that the products work without any defects in a variety of customer scenarios.

These scenarios are described below.

Project Testing / Embedded Testing

When software is developed as product that is delivered to a single client or intended to be used at a single location, the following testing takes place, in addition to software inspections (peer reviews) –

1. **Unit Testing** – this is certainly carried out by the person who wrote the code and by an independent peer using white box testing technique.
2. **Integration Testing** – carried out either as one-off (that is, when all integration is completed) or incrementally (that is, whenever one unit of software is integrated and continued till all units are integrated). Black Box testing is used in one-off Integration Testing and white box testing can be, perhaps, used in incremental integration testing
3. **System Testing** to ensure that the software works in all intended target systems.
4. **User Acceptance Testing** to obtain customer sign-off so that software can be delivered and payments received

Optionally, many other tests can be conducted at the behest of the customer.

Product Testing

Product would be developed as a project first and would undergo all the tests that a project normally undergoes, namely, unit, integration, and system testing. System testing is carried out more rigorously and on multiple systems. In addition, it needs some more rigorous tests. These are –

1. **Load Testing** – in web applications and multi-user applications, large numbers of users are logged in and try to use the software in a random manner. The objective is to see if the software is managing multiple requests and serving up accurate results or mixing them up. This unearths the issues connected with the bandwidth, database, sufficiency of RAM, hard disk etc
2. **Volume Testing** – subject the software to a high volume of data and see the performance, whether it degrades.
3. **Functional Testing** – test that all functions expected of the software are functioning correctly.
4. **End-to-End Testing** – in this type of testing, one entity is tracked from birth to death in the application. For example, in a payroll application, an employee joins the system; then is promoted; then is demoted; salary increases are effected, salary decreases are effected; kept in abeyance; transferred, then retired, dismissed, terminated and so on to ensure that the state transitions designed in the applications happen as desired
5. **Parallel Testing** – a number of users using the same function and are either inputting or requesting same data. This brings out the system's ability to handle requests at the same time and preserving the data integrity.
6. **Concurrent Testing** – 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 – normally using a testing tool. For example, take ticket reservation scenario, there is only one seat and it is shown as available to two people. When both confirm purchase, the system should give to only one and reject the other request. It should not happen that money is

collected from both credit cards and reserve for only one – the credit card transaction must be reversed for the rejected party. Scenarios like this will be tested.

7. **Stress Testing** – cause stress to the software by making expected resources unavailable or causing deadlock like scenarios or not releasing resources and so on to ensure that the software has routines built in to handle such stress. This will bring out software responses for events like machine-rest, Internet disconnection, server timeouts etc.
8. **Positive Testing** – test the software as specified and not trying any negative acts – to ensure that all defined functions are performing. Used mostly for customer / end user acceptance testing.
9. **Negative Testing** – using the software in a manner that is not expected to be used – this will bring out all hidden defects in the software. This is to ensure even malicious usage would not affect the software or data integrity.
10. **User Manual Testing** – use the software conforming to the user manual to ensure that they both are in synch with each other
11. **Deployment Testing** – Simulate the target environment and deploy the software and ensure that deployment specified is appropriate.
12. **Sanity Testing** – this cursory testing to ensure that the components, of software package, are complete and are of appropriate versions, carried out before delivery or before making a software-build.
13. **Regression Testing** – testing carried out after unearthed defects are fixed
14. **Security Testing** – testing to ensure vulnerability against the threat of viruses and spy-ware
15. **Performance Testing** – testing to ensure that the response times are in acceptable range
16. **Usability Testing** – testing the software for different types of usage to ensure that it satisfactorily fulfills the requirements of specified functional areas
17. **Install / uninstall Testing** – test the software on all target platforms to ensure that install and uninstall operations are satisfactorily performed
18. **Comparison Testing** – testing the product with competing products to contrast the differences for determining the relative position of the product
19. **Intuitive Testing** – testing without reference to user manuals to see if the product can be used without much reference to user guides

It is rare that all the above types of testing are carried out for every project that is executed in the organization. But it is common for product testing to include many of the above tests.

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 / Acceptance 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, sometimes, on a “if time and budget are available” — basis” or “if mandated” basis.

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 for intuitive testing, 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 – to be accepted by the system
 - b. One value above acceptable range – to be rejected by the system
 - c. One value below acceptable range – to be rejected by the system
 - d. One value at the upper boundary of the acceptable range – to be accepted by the system
 - e. One value at the lower boundary of the acceptable range– to be accepted by the system
2. Size checks for all non-numeric data, one per every data item
3. Logical testing for presence of invalid data – like two decimal points in numeric data, numeric 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 common to have guidelines for the following tests –

1. GUI Testing
2. Navigation Testing
3. Negative Testing
4. Load Testing
5. Stress Testing
6. Parallel and Concurrent Testing
7. Unit Testing

Organizations make use of these guidelines and avoid, not in all organizations perhaps, preparing test cases, exhaustively.

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.

Test Strategy

Before we can start our discussion on Test Effort Estimation, we need to understand test strategy. **Test Strategy is concerned with unearthing as many defects as possible within the allocated budget of time and cost and maximizing the impact of such testing .**

First step in finalizing the test strategy is to set testing objectives. These could be –

1. **Quality Objectives** – these are concerned with the level of unearthing of defects ranging from
 - a. All defects irrespective of time or cost
 - b. Almost all possible defects within the time available – cost and time are a criterion
 - c. All possible defects within the time available – time is the main criteria
2. **Customer Acceptance Objectives** – the main objective of testing to obtain customer sign-off so that customer pays our money
3. **Product Certification Objectives** – carry out the tests specified by the customer and certify as requested by the client. These certifications could be –
 - a. Virus and Spy-ware free
 - b. Functionality
 - c. Usability
 - d. Comparison and relative position
 - e. Product Rating
 - f. Etc.

In addition to objectives, the following are also part of Test Strategy

1. **Types of tests to be included in testing** – what are the tests that have to be conducted to achieve the project objectives
2. **How of testing** – the methodology of testing
 - a. Plan and Test Case based testing or intuitive testing
 - b. White Box or Black Box
 - c. Manual Testing or Tool based testing
3. **Regression Testing** – number of iterations for Regression Testing – only once or iterated till all defects are closed
4. **Criteria for successful completion of testing** – is it time and cost based or closure of defects or until all defects are unearthed
5. **Mechanisms** for defect closure and escalation when necessary
6. **Progress Reporting** during the project execution
7. **Defect Analysis** – such as ABC analysis, Category analysis etc – whether required or not

Now, we are ready for a discussion on Test Effort Estimation!

First - Definition of Test Estimation

Test Estimation is the estimation of the testing size, testing effort, testing cost and testing schedule for a specified software testing project in a specified environment using defined methods, tools and techniques.

1. **Estimation** – defined in the earlier chapters
 2. **Testing Size** – the amount (quantity) of testing that needs to be carried out. Some times this may not be estimated especially in Embedded Testing (that is, testing is embedded in the software development activity itself) and in cases where it is not necessary
 3. **Testing Effort** – the amount of effort in either person days or person hours necessary for conducting the tests
 4. **Testing Cost** – the expenses necessary for testing, including the expense towards human effort
- Testing Schedule** – the duration in calendar days or months that is necessary for conducting the tests

Now, having understood the definition of Test Estimation, we are ready for looking at the approaches to Test Estimation.

Approaches to Test Effort estimation

Now the following approaches are available for carrying out Test Effort Estimation

1. Delphi Technique
2. Analogy Based estimation
3. Software Size Based Estimation
4. Test Case Enumeration Based Estimation
5. Task (Activity) based Estimation
6. Testing Size Based Estimation

Delphi Technique and **Analogy Based Estimation** are explained earlier for estimation of software development projects. The techniques are same for testing projects also. Hence these are not discussed again here.

Let us look at each of the remaining techniques more closely.

Software Size Based Estimation

By the time a testing project is in its initiation phase, software size would have been available. Now we adopt this software size as the testing project size. Then we assign a Productivity figure (rate of achievement) for the software size to arrive at the required effort to execute the testing project.

Let us say that it takes 2 person hours to test software of size one Function Point – using this norm we can arrive at the amount of effort required for the testing project based on the size of software to be tested.

Suppose that the size of software to be tested is 1000 Function points, then, using the norm of 2 person hours per function point, we have 2000 person hours for testing the software of size 1000 Function Points.

However, such norms for converting software size to effort are not available from any standards body and therefore are to be developed and maintained within the organization using historical data, adhering rigorously to a process. We have to derive this norm for all the software size measures used in the organization as well as maintain them.

Merits of Software Size Based Estimation

Here are the merits of this technique

1. Very Simple to learn and use
2. Very fast – takes very little time to arrive at the effort estimate
3. If the organization derives and maintain these norms using right process, the results of effort estimation using this technique could be surprisingly accurate

Demerits of Software Size Based Estimation

1. Too simplistic – not auditable
2. As testing size is not available, productivity cannot be derived. However, testing productivity can be derived against software size.
3. Amount of testing differs depending on the application type even though the size may be same. For example a stand-alone software application and a web based software application need different amount of testing even though their size may be same. Hence the norm per software size may not be applicable in all cases.
4. Organizations have to keep meticulous records and employ full-time specialists to derive norms and maintain them. The timesheets need to be tailored to capture suitable data to derive these norms accurately. Data Collection has to be rigorous

<----- Person Hours per unit size ----->

Type of Application	Function Points	Use Case Points	Object Points	FPA	Mark II	Feature Points	SSU
Stand-alone	0.25	0.5	0.25	0.5	0.25	0.5	
Client Server	0.3	0.65	0.3	0.65	0.3	0.65	
3-tier Application	0.5	0.9	0.5	0.9	0.5	0.9	
4-tier application	0.75	1.1	0.75	1.1	0.75	1.1	

We need to maintain a table like the above in the organization, to use this technique. Please note that the values indicated in the table are by no means validated – they are just indicative only.

Test Case Enumeration based Estimation

The following steps describe this technique –

1. Enumerate the test cases – list down all the test cases
2. Estimate testing effort required for each test case – use person hours or person days - consistently
3. Use Best Case, Normal Case and Worst Case scenarios for estimating effort needed for each test case
4. Compute Expected Effort for each case using Beta Distribution

$$\text{Best Case} + \text{Worst Case} + (4 * \text{Normal Case}) / 6$$

Sum up the –

1. **Expected times** to get Expected effort estimate for the project
2. **Best-Case** times to obtain best-case effort estimate
3. **Worst-Case** times to obtain worst-case effort estimate
4. **Normal-Case** times to obtain normal-case effort estimate

The below table is an example of this method.

Here is an example of Test Effort Estimation using Test Case Enumeration. PH stands for Person Hours.

Test Case Id Test Case Description	<----- Effort in PH ----->		
	Best Case	Worst Case	Normal Case Expected
US1 Setup Test Environment			
US1.1 Check Test Environment	1.2	1.5	1.500
US2 install Screen recorder	0.75	1.5	1.042
US1.2 Ensure Defect Reporting mechanism	1.25	3.2	2.042
UI1 Login Screen on IE			0.000
UI1.1 Correct Login	0.05	0.2	0.108
UI2 Wrong id and Correct Passsword	0.07	0.2	0.112
UI1.2 Correct Id and wrong Password	0.07	0.2	0.112
UI3 Forgot Password Functionality	0.15	0.3	0.208
UF2 Login Screen on Firefox			0.000
UF2.1 Correct Login	0.05	0.2	0.108
UF3 Wrong id and Correct Passsword	0.07	0.2	0.112
UF2.2 Correct Id and wrong Password	0.07	0.2	0.112
UF4 Forgot Password Functionality	0.15	0.3	0.208
Total Effort Estimate	3.680	8.300	5.500 5.663

Merits of Test Case Enumeration Based Estimation

The following are the merits of this technique –

1. Auditable estimate – the estimate has adequate detail so that another peer can review the estimate and ensure that the estimate is comprehensive and as accurate as possible
2. Fairly accurate – accuracy is ensured as all test cases are enumerated and three times are used to arrive at the expected effort
3. Progress monitoring is facilitated – by marking the test cases completed and percentage completion can be computed easily
4. Facilitates giving a range of values for the estimates – such as –
 - a. The project can be executed with a minimum effort of so many person hours and a maximum of so many person hours with an expected effort of so many person hours. This allows the decision makers to set negotiation margins in their quotes

Demerits of Test Case Enumeration Based Estimation

1. No Testing size – hence productivity can not be derived
2. All test cases and attendant overheads need to be enumerated – takes time to complete the estimation

Task (Activity) Based Estimation

This method looks at the project from the standpoint of tasks to be performed in executing the project. Any project is executed in phases. Phases in a testing project could be –

1. Project Initiation
2. Project Planning
3. Test Planning
4. Test Case Design
5. Set up Test Environment
6. Conduct Testing
 - a. Integration Testing
 - b. System Testing
 - c. Load Testing
 - d. Etc.
7. Log and report test results
8. Regression Testing
9. Prepare Test Report
10. Project closure

Of course, the phases may differ from project to project and organization to organization. Now each of these phases could be further broken down into tasks. Here is an example –

Phase – Project Initiation

1. Study the scope of testing and obtain clarifications if necessary
2. Identify the Project (Test) Manager
3. Retrieve data of past similar projects and make it part of the project dossier
4. Prepare PIN (Project Initiation Note) and obtain approval
5. Conduct Project Kick off meeting and hand over project dossier to project (Test) Manager

Phase – project Planning

1. Prepare Effort Estimates
2. Determine Resource Requirements
3. Raise Resource Request Forms
4. Prepare Project Management Plan
5. Prepare Configuration Management Plan
6. Prepare Quality Assurance Plan
7. Arrange peer review of project plans
8. Obtain approval for project plans
9. Baseline Project Plans

We can breakdown each of the phases into their constituent tasks. Now using these tasks, we can carryout test effort estimation.

The following are the steps in Task based effort Estimation -

1. Assign durations for each of the Tasks – either in person hours or person days - consistently
2. Use three time estimates – Best Case, Worst Case and Normal Case for each of the tasks
3. Compute the expected time using formula
 - a. $[\text{Best Case} + \text{Worst Case} + (4 * \text{Normal Case})] / 6$
4. Make adjustments for project complexity, familiarity with the platform, skill of the developers, tools usage
5. Sum up the total effort estimate of the project
6. Use Delphi technique to validate the estimate, if in doubt or felt necessary

The below table gives an example of Task Based Effort Estimation for Testing Projects

Task Id	Phase	Task	----- Effort in PH ----->		
			Best Case	Worst Case	Normal Case Expected
1	Test Planning	Study Specifications		2.5	3.167
2	Test Planning	Determine types of tests to be executed	0.5	1.0	0.683
3	Test Planning	Determine Test Environment	0.5	1.0	0.792
4	Test Planning	Estimate Testing Project Size, Effort, Cost & Schedule	2.4	3.0	3.500
5	Test Planning	Determine team size	0.5	1.25	0.75
6	Test Planning	Review of Estimation	1.2	1.5	1.750
7	Test Planning	Approval of Estimation	0.5	2.0	0.75
8	Design Test Cases	Design Test Cases for Module 1	5.8	6.0	7.167
9	Design Test Cases	Design Test Cases for Module 2	6.9	7.0	8.333
10	Design Test Cases	Design Test Cases for Module 3	4.6	5.0	5.833
11	Conduct Tests	Conduct Tests for Module 1	15.18	16.0	18.833
12	Conduct Tests	Conduct Tests for Module 2	16.19	17.0	20.000
13	Conduct Tests	Conduct Tests for Module 3	14.16	15.0	17.500
14	Defect Report	Defect Report for Module 1	1.2	1.5	1.750
Total Effort Estimate			68.000	94.250	77.800
			91.267		

1. **Expected times** to get Expected effort estimate for the project
2. **Best-Case** times to obtain best-case effort estimate
3. **Worst-Case** times to obtain worst-case effort estimate
4. **Normal-Case** times to obtain normal-case effort estimate

Merits of Task Based Effort Estimation for Testing Projects

1. This most closely reflect the way projects are executed
2. This technique takes into consideration all the activities that are performed and gives effort estimates as accurately as possible

3. It has adequate details that makes it amenable for reviewing and auditing and postmortem analysis by comparing with the actual values
4. Simple and easy to prepare an estimate
5. Makes project progress monitoring easy by marking completed tasks and percentage completion can be easily computed
6. Suitable for use in Analogy Based Test Effort Estimation also

Demerits of Task Based Effort Estimation for Testing Projects

1. Testing size is not computed – therefore, testing productivity can not be arrived at

Issues in Sizing the Testing Projects

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 and practices 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 other techniques mentioned above.

But size measure is important so that comparison can be made between two projects; it is important to assess the reasonableness of the effort estimates. It also facilitates computation of Testing Productivity – rate of achievement.

Who needs Test Size Estimation?

1. **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 customer specifications and expectations. This set would carry out –
 - a. Mainly Black Box testing
 - b. Functional Testing
 - c. System Testing
 - d. Negative Testing
 - e. Regression Testing
2. **Customers** who entrusted their software development to a vendor. Their objective is to ensure that they are getting what they are paying for.

Sizing of Testing Project

The term “ **Test Points** ” is catchy and perhaps the Unit of Measure for the estimation of Testing size and effort. This term is being used by many persons and is popular to size software testing projects. Test Points can be extracted from the software size estimates.

Test Point is a size measure for measuring the size of a software-testing project and that a Test Point is equivalent to a normalized test case. Here a test case is the one having one input and one corresponding output.

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 universal agreement. Perhaps, we may say that one Adjusted Function Point results in one (or 1.2 or 1.3 etc.)

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 always - 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 are no uniformly agreed types of testing to be carried out on any given project.

There is barely enough research and empirical data that accurate 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 could be considered

1. Application weight
2. Programming language weight
3. Weights for each type 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 Negative Testing
 - j. And so on

All weights are project-specific.

Test Point has a weight equal to 1 when the combined weights of three tests, namely, Integration Testing, System Testing and Acceptance Testing is equal to 1. That is to say that the sum of weights of these three tests cannot be more than 1 nor less than 1.

When other tests are added to the project, their weights may be assigned and added to Test Point weight.

We need to compile weights data for all these tests and maintain them in-house by comparing the estimated values with actual values at the end of every testing project after conducting a rigorous causal analysis in each case.

Testing Tools usage is expected to reduce the effort, even though there are views that tools would not reduce the effort for the first iteration of testing. Perhaps, but it really depends on the tool itself. Therefore, the weight for tools usage also may be assigned suitably based on the tool itself and the project at hand. A weight of 1 for tools usage indicates that the tool would not have any impact on the effort required for testing. A weight of more than 1 indicates that the tool increases the testing effort and a weight of less than 1 indicates that the tool would reduce the testing effort.

If we include Unit Testing in the proposed tests for the project, we need to assign another weight for the programming language used for developing code for the project. Here we mean independent Unit Testing carried out by a person who had not written the code in the first place. The reasons for this additional weight are –

1. Unit testing is white box testing – that is from within the code
2. The development environment for different language is different from one another and differs in the amount of effort required for the testing project

The following are the steps in computing the testing project size in Test Points. We are using the software size as the basic input to this model.

1. Use an existing software development size
2. Convert the software size into Unadjusted Test Points (UTP) using a conversion factor which is based on the application type
3. Compute a Composite Weight 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. Unadjusted Test Points are multiplied by CWF to obtain the testing size in Test Points size
5. The Productivity Factor indicates the amount of time for a test engineer to complete the testing of one Test Point
6. Testing Effort in Person Hours is computed by multiplying Test Point Size by the Productivity Factor.

The below table illustrates the Test Points estimation

Sl. No Aspect Test Points

- 1 Product Size in FP 2500
- 2 Conversion Factor (TP per FP) 4.5
- 3 Unadjusted Test Points 11250
- 4 Application weight for Client-Server Application 1.1
- 5 Composite Weight Factor (CWF) 1.375

- 6 Adjusted Test Points 15468.75

- 7 Productivity Factor in Person Hours /TP 0.2

- 8 Test Effort in Person Hours 3093.75

The below table gives the various test weights used in computing the CWF in the above table

Test Weights

- 1 Functional Test 0.35
- 2 System Test 0.3
- 3 Acceptance Test 0.2
- 4 Virus-Free Test 0.2
- 5 Spy-ware-free test 0.2
- Total Weight 1.25

Merits of Test Point Estimation

1. Size is estimated – makes it amenable to productivity computation, comparison and benchmarking
2. Size makes it useful in Analogy Based estimation

Demerits of Test Point Estimation

1. There is universally accepted definition of what is a Test Point
2. Perhaps, not as simple as other Test Effort Estimation methods
3. No universally accepted or benchmarked data available on various weights used in the method. These have to be developed and maintained adhering to rigorous methodology and record keeping. This puts overheads on the organization
4. Timesheet has to be oriented for deriving required data

Well, nothing that is good ever comes free or easily. So is the case with Test Point estimation for sizing Testing projects – one needs to spend effort and time to set the norms – as in perhaps, any other case.

Final words about Test Effort Estimation

Here is an area where further work is necessary, obviously. However, there are methods that make it possible to estimate effort required for executing Testing projects. Test Points are slowly emerging for sizing Software Testing projects.

It is suggested that the project is scheduled, just the way, software development projects are scheduled, resources allocated and the schedule is re-worked with resource constraints and only then the schedule and effort are committed.

It is also suggested that the presentation of Test Effort Estimate also be subjected to the format suggested for Software Development Project Estimates to ensure that all aspects of estimation are communicated to the decision makers.