Test Effort Estimation Using Use Case Points

Abstract

This paper presents a new approach to the estimation of software testing efforts based on Use Case Points [UCP] as a fundamental project estimation measure. From preliminary applications on our web-based projects, we conjecture that this could in fact be more reliable than FP. The caveat here is that the V-model must be in use and use case generation must start becoming available right at the requirements gathering phase. The acceptance test plan is then prepared with the use cases from the requirement documents as input. Further work could provide a more exact relationship between the two.

Introduction

Probably the most crucial difference between the manufacturing industry and the software industry is that the former is able to stick to schedules most of the time. The reason why software development schedules are so unpredictable is not because workers in this industry are lazy or incompetent. To estimate the time make a product from scratch, and in many cases, without prior experience of the technology is no mean feat. However, conventional estimation techniques address only the development effort that goes into it.

It is known that a use case to test case mapping is possible. This means that the UCP figure for development can be indirectly used to provide a figure for the number of test cases. Using organizational test execution time metrics it is now possible to arrive at a figure for the total test effort. This is a viable and systematic approach towards test effort estimation and it makes a leap in providing more realistic figures. This means that the cost of testing can now be factored into projects. The other advantage is that test engineering gets treated as a process and not simply as another lifecycle phase.

Software Test Engineering

Test Engineering covers a large gamut of activities to ensure that the final product achieves some quality goal. These activities must be planned well in advance to ensure that these objectives are met. Plans are based on estimations.

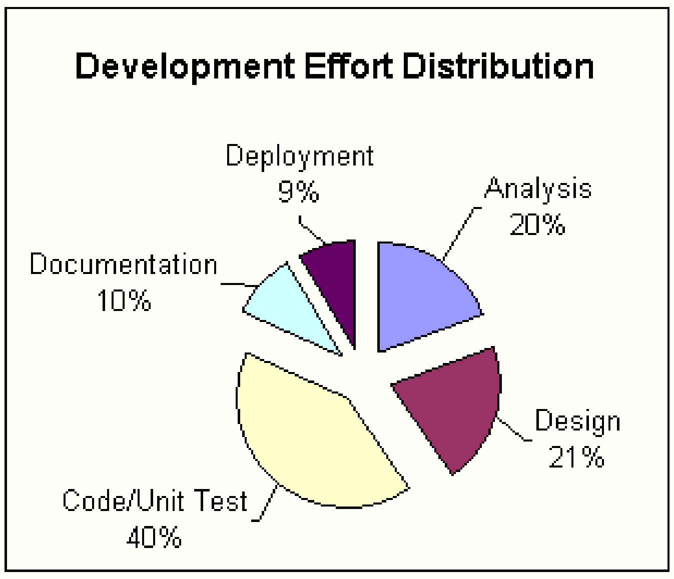
In the early years, the Waterfall model has been ap plied to software development. This model looks upon test engineering as merely a stage in the entire development lifecycle. When techniques evolved over the years for estimating development time and effort, the concept of estimating test-engineering time was overlooked co mpletely.

Test engineering is seldom planned for in most organizations and as a result, products enter the market insufficiently tested. Negative customer reactions and damage to the corporate image is the natural consequence.

To avoid this, the correct develop ment lifecycle must be chosen and planning should be done early on in the cycle.

Software Project Estimation

According to Rubin [2], the stage-wise effort distribution on software projects is as shown in the pie chart below.



Estimation is basically a four-step approach:

1. Estimate the size of the develo pment product. This is either in LOC [Lines of Code] or FP [Function Points]. The concept of usin g UCP [Use Case Points] is still in its infancy.

2. Estimate the effort in person-months or person-hours.

3. Estimate the schedule in calendar months.

4. Estimate the cost in currency.

Conventional Approach to Test Effort Estimation

Test engineering managers use many different methods to estimate and schedule their test engineering efforts. Different organizations use different methods depending on the type of projects, the inherent risks in the project, the technolo gies involved etc.

Most of the time, test effort estimations are clubbed with the development estimates and no separate figures are available.

Here is a description of some conventional methods in use:

1. Ad-hoc method

   The test efforts are not based on any definitive timeframe. The efforts continue until some p re-decided timeline set by managerial or marketing personnel is reached. Alternatively, it is done until the budgeted finances run out.

   This is a practice prevalent in extremely immature organizations and has error margins of over 100% at times.

2. Percentage of development time

   The fundamental premise here is that test engineering efforts are dependent on the development time / effort. First, development effort is estimated using some techniques such as LOC or Function Points. The next step is using so me heuristic to p eg a value nex t to it. This varies widely and is usually based on previous experiences.

   This method is not defendable since it is not based on any scientific principles or techniques. Schedule overruns could range from 50 – 75% of estimated time. This method is also by far the most used.

3. From the Function Po int estimates

   Capers Jones [1] estimates that the number of test cases can be determined by the function points estimate for the corresponding effort. The formula is

   Number of Test Cases = $(\text{Function Points})^{1.2}$

   The actual effort in person-hours is then calculated with a conversion factor obtained from previous project data.

   The disadvantage of using FP is that they require detailed requirements in advance. Another issue is that modern object-oriented systems are designed with Use Cases in mind and this technique is incompatible with them.

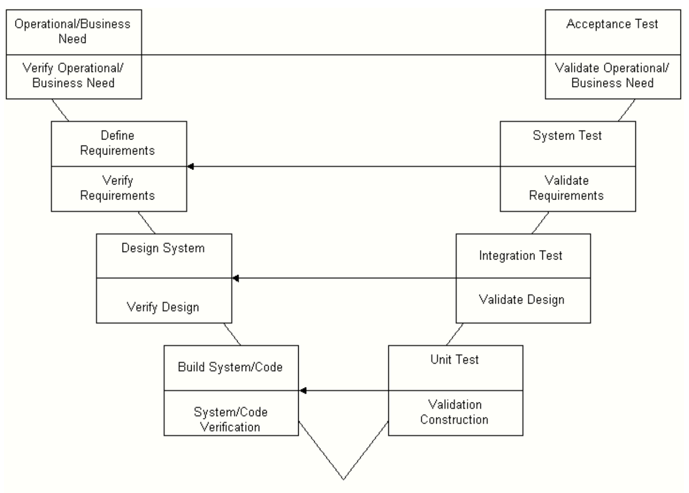
Use Cases

Alistair [5] has this description o f a use case:

A use case captures a contract between the stakeholders of a system about its behavior. The use case describes the system's behavior under vario us conditions as it responds to a request from one of the stakeholders, called the *primary acto* r. The primary actor initiates an interaction with the system to accomplish so me goal. The system responds, protecting the interests of all the stakeholders. Different sequences of behavior, or scenarios, can unfold, depending on the particular requests made and conditions surrounding the requests. The use case collects together those different scenarios.

Mapping Use Cases to Test Cases

Use cases in their mo st primitive forms are basically representative of what the user wants from a system. The advantages of Use Cases are that they start becoming available early on in the project lifecycle. The appropriate project lifecycle model is the V-Model. The figure below illustrates the same.



The Expanded V Chart

The model clearly has one test engineering activity associated with a corresponding development activity. The topmost rung of the model associates the business requirement identification with the acceptance plan preparation. Each successive step makes sure that the test documentation beco mes complete and comprehensive. If the estimation process is fitted in the seco nd rung after the business requirements are available, it is obvious that use cases will serve as the inputs.
The identification of the nu mber of test cases here can be made quite directly. Each scenario and its exception flows for each use case are input for a test case. Subsequently, the estimation calculations can commence.
As the requirements become clearer further downstream, the estimates will also undergo revision.

UCP Approach to Estimation

Estimation using UCP [Use Case Points] is rapidly gaining a faithful following. The approach for estimation using UCP only needs slight mod ification in order to be useful to estimate test efforts.

1. Determine the number of actors in the system. This will give us the UAW – the unadjusted actor weights.

   Actors are external to the system and interface with it. Examples are end-users, other programs, data stores etc.
   Actors come in three types: simple, average and complex. Actor classification for test effort estimation differs from that of development estimation.

   End users are simple actors. In the context of testing, end-user actions can be captured easily using automated tool scripts. Average actors interact with the system through some protocols etc. or they could be Data stores. They qualify as average since the results of test case runs would need to be verified manually by running SQL statements on the store etc. Complex users are separate systems that in teract with the SUT through an API.

   The test cases for these users can only be written at the unit level and involves a significant amount of internal system behavioral knowledge.

   Actor Weights

Actor Type	*Description*	*Factor*
Simple	GUI	1
Average	Interactive or protocol-driver interface	2

Complex	API / low-level interactions	3

The sum of these products gives the total unadjusted actor weights. [UAW]

2. Determine the number of use cases in the system. Get UUCW.

   The use cases are assigned weights depending on the number of transactions / scenarios.

<u>**Use-case Weights**</u>

Use Case Type	*Description*	*Factor*
Simple	<=3	1
Average	4-7	2
Complex	>7	3

The sum of these products gives the total unadjusted actor weights. [UAW]

3. UUCP = UAW + UUCW

   The calculation of the unadjusted UCP is done by adding the unadjusted actor weight and the unadjusted use case weights determined in the previous step s.

4. Compute technical and environmental factors

   The technical and environmental factors for a test project are listed in the table b elow.

   To calculate one needs to assign weights and multiply them with the assigned values to give the final values. The products are all ad ded up to give the TEF multiplier. The TEF multiplier is then used in the next step.

Technical Complexity Factor

Factor	*Description*	*Assigned Value*
T1	Test Tools	5
T2	Documented inputs	5
T3	Development Environment	2
T4	Test Environment	3
T5	Test-ware reuse	3
T6	Distributed system	4
T7	Performance objectives	2
T8	Security Features	4
T9	Complex interfacing	5

5. Compute adjusted UCP.

   We use the same formula as in the UCP method for development.

   AUCP =UUCP *[0.65+(0.01*TEF)]

6. Arrive at final effort.
   We no w have to simply multiply the adjusted UCP with a conversion factor. This co nversion factor denotes the man-ho urs in test effort required for a language/technology combination. The organization will have to determine the co nversion factors for various such combinations.

   E.g. Effort = AUCP * 20
   Where 20 man-hours are required to plan, write and execute tests on one UCP when using EJB.

Example

The project under study is a product support web site for a large North American software company. The estimation was done from the business level use cases made available at the time of signing the requirements. The actors at this time were the different types of users identified in those use cases.

1. **UAW Calculation**

Actor	*No of Use Cases*	*Factor*	*UAW*
B2C User	15	2	30
Subscribers	13	2	26
Admin User	4	2	8
Total UAW			**64**

2. **UUCW Calculation**

Legend: Simple – S, Average – A, Complex – C, Very Complex - VC

Use Case	*Type*	*Factor*	*Reason*
Login	C	15	Server Integration
Support Request	VC	20	External Sys Query
User Creation	A	10	
Support Resource Mgt.	S	5	Code Reuse
Fix Notifications	S	5	Trivial
Total		**55**	

3. Calculation of the UUCP - Unadjusted Use Case Points
UUCP = UAW + UUCW = 64 + 55 = 119

4. Technical factor computation

Factor	*Description*	*Assigned Value*	*Weight*	*Extended Value*
T1	Test Tools	5	3	15
T2	Documented inputs	5	5	25
T3	Development Environment	2	1	2
T4	Test Environment	3	1	3
T5	Test-ware reuse	3	2	6
T6	Distributed system	4	4	16
T7	Performance objectives	2	1	2
T8	Security Features	4	2	8
T9	Complex interfacing	5	2	10
			Total	**87**

5. **Adjusted UCP calculation**

AUCP =UUCP *[0.65+(0.01*TEF)] = 119 * [0.65+0.01*87] = 180.88

6. **Final Effort**

Effort = AUCP * Conversion Factor for COM / DCOM testing
Effort = 180.88 * 13 = 2351.44

Project Complexity needs 15% of the estimated effort to be added. 10% is spent in co -ordination and management activity.

Total Effort = 2351.44 + 352.72 + 235.144 = 2939.304 man-hours = 367 man-days

Actual Effort = 390 man-days [Project End]