Volume 2, Issue 8, August 2012 **ISSN: 2277 128X**

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Improvement in Software Development Process: A New SDLC Model

Kumar Adesh [1], **Beniwal Vikas** [2]

[1]*Research Scholar, M Tech, Department of Computer Science & Engineering*
N C College of Engineering, Israna, Panipat
[2]*Assistant Professor, Department of Computer Science & Engineering*
N C College of Engineering, Israna, Panipat

Abstract: Software testing is very important in software development process; approximately 50% of total cost is expended in testing the software being developed. A technique similar to FPA, Test Point Analysis (TPA) can be applied for the estimation of testing effort. We are using a new approach to the estimation of software testing efforts based on stubs and drivers. Stubs and drivers are needed when the unit and integration testing is done. Drivers and stubs can be reused so that constant changes that occur during the development cycle can be retested frequently without large amounts of additional test code. FP count is calculated by FPA technique earlier and then applied to calculate TPA to find the Total Testing Effort.

Keywords: TPA, FPA, V-Model, Test hours, SDLC etc.

I. Introduction

A software testing model summarizes how you should think about test development . It tells you how to plan the testing effort , what purpose tests serve when they're created, and what sources of information you use to create them.
We can use QA processes to attempt to prevent defects from entering software but the only thing we can do to reduce the number of errors already present is to test it. By following a cycle of testing and rectification we can identify issues and resolve them. Testing also helps us quantify the risk in an untried piece of software. And finally in some software development efforts, testing can actually be used to drive development. By following statistical models of software development and methods such as usability testing, software development can move from an unfocused artistic endeavor to a structured discipline. We are using the V Model as our testing model for development process [1]

The V-model was originally developed from the waterfall software process model.The four main process phases – *requirements*, *specification*, *design* and *implementation* – have a corresponding *verification* and *validation testing* phase.Implementation of modules is tested by *unit testing*, system design is tested by *integration testing*, system specifications are tested by *system testing* and finally *acceptance testing* verifies the requirements.
The V-model is easy to understand, but the timing of the phases where testing occurs after all implementation is done, is not an adequate approach in today's iterative software processes. However, the test levels from the V-model which associate a certain process phase with a testing phase is still relevant and will be used in this project. When the test levels from the V-model are used in parallel with development, and more importantly, when the different levels are tested simultaneously in multiple iterations, the V-model testing method can become an effective and manageable testing process [2].

During development the program will be tested at all levels simultaneously. The different levels are unit tests, integration tests, system test and acceptance tests. The unit tests and integration tests ensure that the system design is followed in the code. The system and acceptance tests ensure that the system does what the customer wants it to do. The traditional V-model states that testing at a higher level is begun only when the previous test level is completed. We will not follow this rule, testing in higher levels will begin when the system has been developed to the point where testing at that level can begin. Splitting the testing into V-model test levels that are performed in parallel could mean that the testing process is easily manageable and more flexible than traditional V-model testing. During the project we will try to assess this approach [2].

II. Why V-model

V- Model is one of the software development model where testing is done parallel with the application development i.e. When the development of application is in process test engineering will test each and every out come document. Waterfall model is used when requirement are clear and complete and for the small projects. Here we can't incorporate new changes in the application V-Model can be viewed as a process improvement technique or model as well. Using this model /technique we can improve the Testing Process by that way we can reduce time of testing which results in the Reduction of
the Testing Budget. Also, we can find more critical bugs in each stage which can be resolved at that time itself reducing effort and minimizing the number of faults found during Dynamic Testing. So, it can be used as Model to Improve Testing Quality and a Technique to improve V-Model is the process model to provide quality product by combining SDLC and STLC. In this

model Both SDLC & STLC works parallel. Left Hand side of the V- Model contains SDLC in downhill direction whereas right hand side of the V-Model contains STLC in Uphill direction Product Quality.[3]

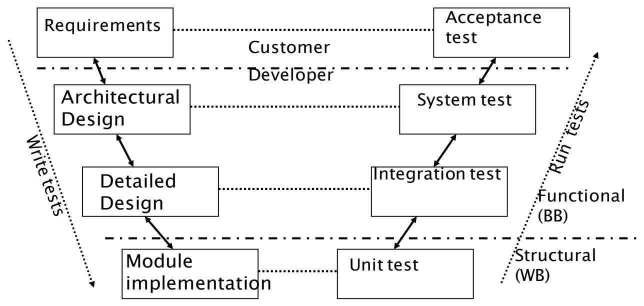

Figure 1: Testing in V-Model

III. Improving software testing process

We discuss key activities in the testing process and some ideas to bring in efficiencies in these elements to improve the testing process [4].

Test case generation:- Test case generation takes up 40-45% of the testing effort. Efficient and complete test cases ensure efficiency of the test process. Automating the test case generation process could reduce the test case generation time by up to 60-70%.

Test case execution: - Test case execution consumes 40-50% of test effort. Application changes during maintenance result in need for increased number of regressions. In manual test execution, the test effort increases with increase in number of regressions. By automating the test execution, the testing effort per regression test round reduces as the number of regressions increase [4].

Defect detection: - On average, a programmer makes 10-12 defects per 100 LOC. This can be reduced to 2-3 defects per LOC for well-documented and constructed code. Also 70% of the defects relate to the analysis and design phase and only 30% to the coding phase. Detecting defects early reduces the cost of defect detection and removal [4].

Unit testing: - The test specification process involves specifying tests to be performed against certain requirements, and listing the results expected in the test result. The test specification process itself may discover many bugs and identify omitted scenario and functionality. Test logs should be maintained to report the bugs found, and bugs waiting re-testing.[5]

Integration testing: - Integration begins after 2-3 units have been successfully unit-tested. Integration testing formally integrates a set of programs [5].

System testing:-During the system test phase the concern is not how the functionality is achieved but that the required functionality is correctly achieved. Knowledge of the business and its requirements should be combined with knowledge of how to use the system [5].

Acceptance testing: - Acceptance testing builds confidence of clients and users. Often, a sub-set of system tests can be used to demonstrate key functionality that provides 80% value to business.

The most common approach to unit testing requires drivers and stubs to be written. The driver simulates a calling unit and the stub simulates a called unit. The investment of developer time in this activity sometimes results in demoting unit testing to a lower level of priority and that is almost always a mistakes. It allows for automation of the testing process, reduces difficulties of discovering errors contained in more complex pieces of the application, and test coverage is often enhanced because attention is given to each unit. Finding the error (or errors) in the integrated module is much more complicated than first isolating the units, testing each, then integrating them and testing the whole.

Driver: A program that calls the interface procedures of the module being tested and reports the results. A driver simulates a module that calls the module currently being tested.[5]

***Stub*:** A program that has the same interface procedures as a module that is being called by the module being tested but is simpler. A stub simulates a module called by the module currently being tested.[5]

Drivers and stubs can be reused so the constant changes that occur during the development cycle can be retested frequently without writing large amounts of additional test code. In effect, this reduces the cost of writing the drivers and stubs on a per-use basis and the cost of retesting is better controlled. We are using this approach as the stubs and drivers are reused then the less coding is to be done, and less will be the test effort for test the code. [5]

Estimation

Adequate estimation of software development, maintenance and testing effort is essential, as absence of it leads to programmers compromising on quality. Ineffective estimating leads to schedule and cost overruns.

The size estimate is based on customer requirements, proposal, system specifications, approach used, user and system requirement description and any design documentations provided by the customer.[6]

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. Sometimes 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
5. **Testing Schedule** – The duration in calendar days or months that is necessary for conducting the tests.

IV. New SDLC Model

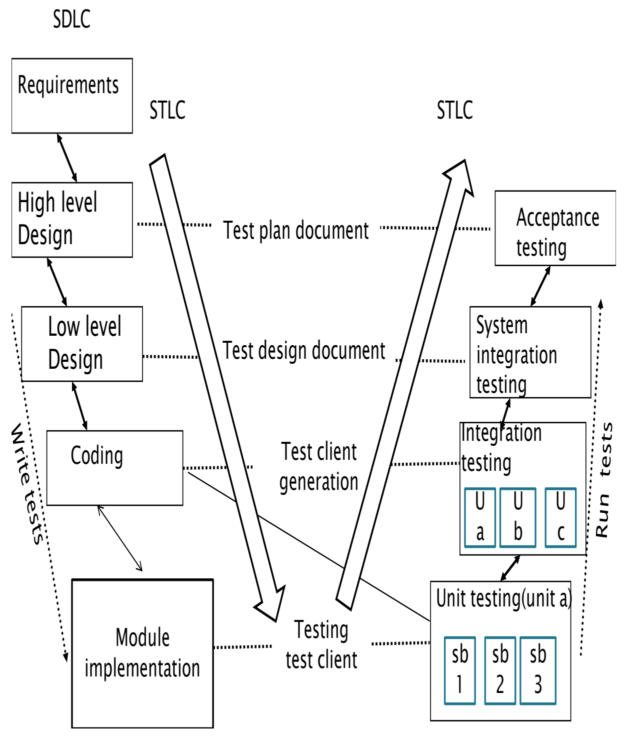

Figure 2: V-Model when stubs and driver are reused for testing

A. *FPA Technique*

The FPA technique estimates the development function points, which also include white-box testing effort. Additional techniques, test points and maintenance points provide size for purpose of estimating black-box testing effort and maintenance effort. [4]

Steps in FPA technique are:

1. DET, FTR, and RETs

DET in simple words is a field in a table, or a GUI element

FTR is the number of files referenced in accepting an external input or generating external output or external inquiry. The files referenced are either internal logical files (ILF) or external interface files (EIF)

Record element type (RET) is a sub-group of DETs in an ILF/EIF. If the ILF/EIF is in fully normalized form, there would be only one RET in the ILF/EIF.
2. Unadjusted Function Point Counts (UFPC)
Standard tables are available to compute (UFPC) for data (ILF/EIF) and for transactions (EI/EO/EQ) based on number of DET, RET and FTRs.
3. Value Adjustment Factor and Adjusted Function Point
A value adjustment factor based on general system characteristics is applied to compute adjusted function point count.

VAF= (total DI*0.01)+0.65

DI-degree of influence, maximum value of DI is 70 and minimum value is 0, so the VAF varies between 0.65 and 1.35. VAF is applied on UFPC to convert it to adjusted FP count [4].

B. Test Point Analysis (TPA)

While white-box test activities are included in the size calculation produced by FPA, the black box testing activities are not included in size computation of FPA. TPA is one such method which can be applied for estimating test effort in black-box testing. The goal of this technique is to outline all major factors that affect testing projects and to ultimately do accurate test effort estimation. If one has a predetermined estimate of test hours allocated, TPA can help on testing of areas that pose higher risk. This is accomplished by determining relative importance of functions and using the available test time on testing of functions of relatively higher importance. As per TPA method, there are two kinds of test points-dynamic and static.[7]
In the many approaches to test effort estimation, the use of stubs and drivers may be one. This could become a robust method of estimation over a period of time. The estimation technique is not claimed to be rigorous, but the approach offers practical advantages over techniques currently in use [7].

V. TPA Approach for Estimation

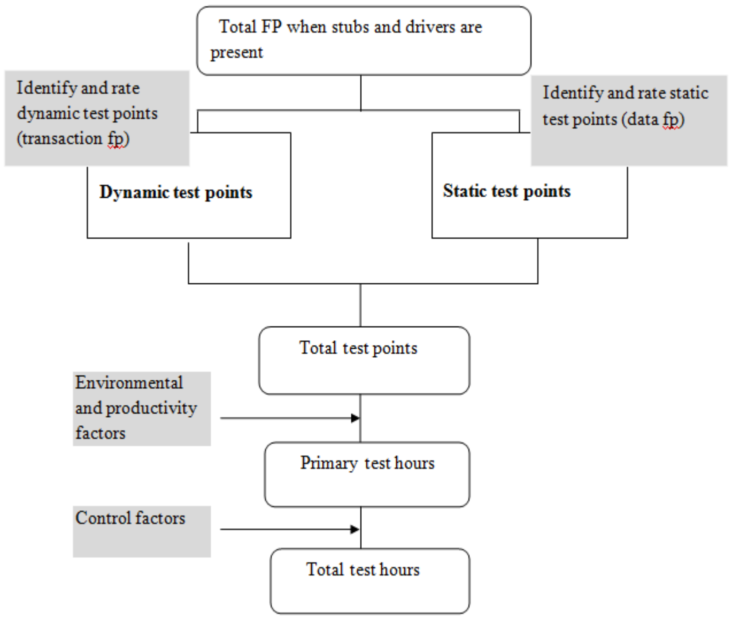
Total FP when stubs and drivers are present
Identify and rate dynamic test points (transaction fp)
Identify and rate static test points (data fp)
Dynamic test points
Static test points
Total test points
Environmental and productivity factors
Primary test hours
Control factors
Total test hours

Figure 3: Derived TPA model

1. Computing Dynamic Test Points (TPs)

Dynamic test points are related to individual function and are based on FPA transaction function points. Dynamic test points are computed by summing the product of Transaction Function points (FP_t), Dependency Factor (D_f), and Dynamic Quality Characteristics (Q_d) for individual function points.
Dependency factor (D_f): A rating is assigned for the individual functions points. A useful heuristics is to have 25% functions in low, 50% in medium and 25% in high category.

- User Importance of the functions: Rating—3-low, 6-medium, 12-high.
- Usage Intensity of the functions: Rating—2-low, 4-medium, 12-high.
- Interfacing with other functions: Rating—2-low, 4-medium, 8-high.

- Complexity of function: Rating—3-low, 6-medium, 12-high.

These ratings are added and divided by 20 (sum of medium rating) to arrive at weighted rating, and uniformity factor could be 0.6 or 1. The uniformity is taken at 0.6 in case of second occurrence of unique function, where test specs can be reused else, uniformity factor is taken at 1.
Dependency factor is calculated by multiplying weighted rating with uniformity factor.
Dynamic quality characteristics (Q_d): This calculation is based on rating and weighing factor for the variables-suitability, security, usability, efficiency. Weighing factors for these four variables are 0.75, 0.05, 0.10, and 0.10 respectively. For each of these variables the rating is (0-not important, 3-relatively unimportant, 4-medium importance, 5- very important, 6- extremely important).
Total dynamic test points equal sum of FP_t* D_f*Q_d for individual functions.

2. *Computing Static Test Points*
Static test points are related to overall FP of the system and static quality characteristics of the system. Overall
FP of the system is assumed at minimum 500(in case it is below 500)recommends functionality, usability, reliability, efficiency, portability and maintainability as quality characteristics and several sub- characteristics within these as desirable. For each quality characteristics statistically tested, a value of 16 is added to Qi.

3. *Total test points*
Total test points are equal to sum of Dynamic and Static test points.
TP = (Sum of FP_t* D_f *Q_d for individual functions) + (Total FP* Qi/500)

4. *Productivity factor (P)*
Indicates tests hours required per test point. It ranges from 0.7(if test team is highly skilled) to 2(if test team has insufficient skills) hours per test point. Productivity factor requires historical data of the projects and it can very from one organization to another organization. So, this factor can be called organization dependent factor.

5. *Environmental factor (E)*
The number of test hours required for each test point is not only influenced by productivity factor but also by the environmental factor.The following environmental factor might affect the testing effort: test tools, development testing, test basis, test ware, development environment, and test environment. Environmental factor is calculated by adding the rating on all the above environmental factors and divided by value 21(the sum of nominal ratings).

6. *Primary test hours*
The number of primary test hours is obtained by multiplying the number of test points by productivity factor (P) and environment factor (E).
Primary test hours = Test points (TP)*P*E

7. *Planning and control allowance*
The standard value of this is 10%.this value may be increased or decreased depending on two factors
Team size: The bigger the team, the more effort it will take to manage the project. The ratings for this value are:
3- if team consists of up to 4 persons, 6- if team consists of up to 5 and 10 persons, 12- if team consists of more than 10 persons.
Management tools: More the number of tools used to automate management and planning less are the amount of effort required. The ratings for this value are:
2-both an automated time registration system and automated defect tracking system are available, 4- either an automated time registration system or automated defect tracking system is available, 8- no automated systems are available.
Planning and control allowance =Team size factor +Management tools factor

8. Total test hours
The total number of test hours is obtained by adding primary test hours and the planning and control allowance.
Total test hours= Primary test hours+ Planning and control allowance
In the many approaches to test effort estimation, the use of stubs and drivers may be one. This could become a robust method of estimation over a period of time. The estimation technique is not claimed to be rigorous, but the approach offers practical advantages over techniques currently in use.

VI. Conclusion

Testing effort is the number of hours that is required for the testing process of software that is being developed. Effective test effort estimation is one of the most challenging and important activity in software testing. There are many popular models for test effort estimation in vogue today. Ineffective test effort estimation leads to schedule and cost overruns. This is due to lack of understanding of development process and constraints faced in the process. But we believe that our approach overcomes all these limitations. My dissertation work is aimed to find out that how effectively we can minimize the test effort for a project. We used the TPA method for our proposed work. Test Case Point Analysis is a tool to estimate the effort required to test a software project, based on the number of use cases and the other features of object-orientation used in software development. Testing is an important activity that ensures the quality of the software. TCP is such a method which is almost equal to the actual effort.

VII. Future work

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. In the many approaches to test effort estimation, the use of stubs and drivers may be one. Drivers and stubs can be reused so the constant changes that occur during the development cycle can be retested frequently without writing large amounts of additional test code. In effect, this reduces the cost of writing the drivers and stubs on a per-use basis and the cost of retesting is better controlled. We are using this approach as the stubs and drivers are reused then the less coding is to be done, and less will be the test effort for test the code. Either it takes more code writing for stubs or drivers but the reusability of these minimizes the overall coding and the test effort also. So using the stubs and drivers approach is more beneficial than without them. This could become a robust method of estimation over a period of time. It leads to accurate estimation of test effort by this estimation we can easily calculate the test effort for the each phases of a testing life cycle. We can apply this estimation to find the estimated test plan and it is also a very powerful method to generate realistic test cases.

References

[1]. Nick Jenkins,“A Software Testing Primer”,An Introduction to Software Testing,2008. http://www.nickjenkins.net

[2]. Jakobsson,“V-Model Testing –Process model configuration using SVG”, PMoC 14/04/2003 Version 1.5

[3]. Jaya Gupta Asanani,“ V-Model (Software Development)”

[4]. Renu Rajani, Pradeep Oak, (2004) “Software Testing-Software Test Effort Estimation Techniques”,Tata McGraw Hill, New Delhi

[5]. Patrick Oladimeji,Dr. Markus Roggenbach,Prof. Dr. HolgerSchlingloff, “Levels of Testing”,*Advance topics in Computer Science,University of Wales Swansea*

[6]. William E,(1999), “Effective Method For Software Testing”, Perry, (pp-177-205), Wiley

[7]. Rajiv Chopra, (2009), Second Edition ,“Software Testing- Test Point Analysis” (TPA), (pp 309-321), S.K Kataria & Sons, New Delhi.