International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 6, August - 2012

AN IMPROVED APPROACH IN SOFTWARE DEVELOPMENT LIFE CYCLE: A NEW MODEL

Author:-Jyoti, student of M.techAtDept:-Computer Science and Engineering, College:-Gurgaon Institute of Technology and Management, Bilaspur,

Co-Author:- rajeshyadavassistant professor At Dept:-Computer Science and Engineering, College:-Gurgaon Institute of Technology and Management, Bilaspur, 2.shivani rajoria,assistant professor At Dept:-Computer Science and Engineering, College:-Gurgaon Institute of Technology and Management, Bilaspur

Abstract:-We define the new SDLC model that is developed to minimize the test effort of software project. This model is enhanced version of V-Model. The concept of stubs and drivers is used to minimize the test and efforts by test point analysis. The main feature of our model is to use the stubs and drivers for reusability of code to minimize the test effort.

Main Goals of the proposed model:

- Enhance the reusability of code.
- Minimize the test effort estimation.

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. A good model guides your thinking; a bad one warps it. V-Model is the basis for my work as it is a software development standard model. In the V-Model the development and testing are parallel activities that take up simultaneously.

V-Model is the best model for development as it is very easy to use and understood and each phase has some specific deliveries and less chances of downward flow of defects.

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.

1. INTRODUCTION

1.1 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. Effective test effort estimation is one of the most challenging and important activity in software testing. Many popular models for test effort estimation in use today. One of the popular methods is FPA.

1.1.1 FPA TECHNIQUE

The FPA technique estimates the development function points, which also include white-box testing effort. FPA is a method for measuring the size of the software on the customer's point of view and describes a unit of work suitable for measuring the size business application software. FPA can be used to measure productivity across various tools and environments. A basic knowledge of the FPA method is necessary to understand test point and maintenance analysis. The most common approach to unit testing

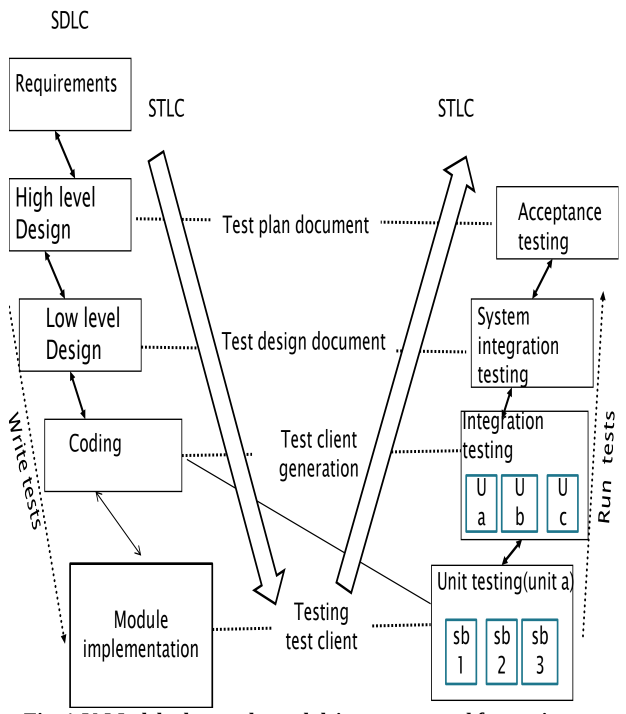


Fig. 1:V-Model when stubs and driver are reused for testing

requires drivers and stubs to be written. The driver simulates a calling unit and the stub simulates a called unit. 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.
Stub: A program that has the same interface procedures as a module that is being called by the module being tested.

1.1.2 TPA TECHNIQUE

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 As per TPA method, there are two kinds of test points-Dynamic and Static.

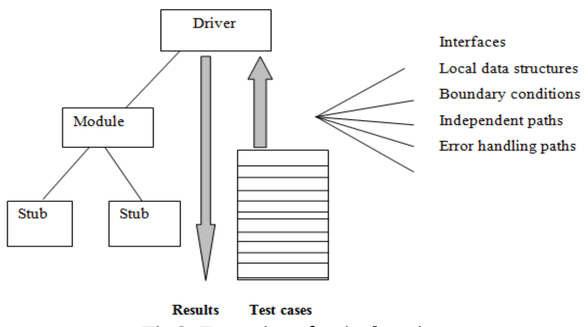


Fig 2: Execution of a single unit

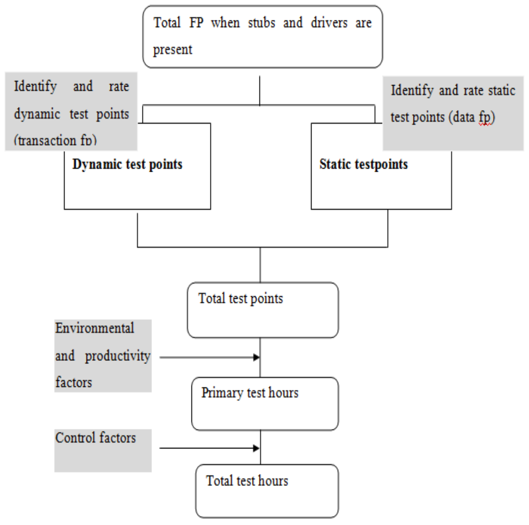
2. PROPOSED MODEL

As FPA is doing white box testing only, we need the TPA model to find the black box testing. The FP count we use to calculate the TPA is estimated earlier in the FPA technique. As per the FPA technique, there are two sets of elementary processes-transaction function points (data in motion), data function points (data in rest).TPA is one such method which can be applied for estimating test effort in black box testing. It is a 6-step approach to test estimation and planning. This approach has a good potential for providing test estimation for various projects. Ineffective test effort estimation leads to schedule and cost overruns. This is due to lack of understanding of development process and constraints faced in this process. 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.

2.1 TPA APPROACH FOR ESTIMATION

2.1.1. 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.


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 testpoints
Total test points
Environmental and productivity factors
Primary test hours
Control factors
Total test hours
Fig 3: Derived TPA model

- 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.

2.1.2. 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.

2.1.3. 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)

2.1.4. 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 vary from one organization to another organization. So, this factor can be called organization dependent factor.

2.1.5. 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).

2.1.6. 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

2.1.7. 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

2.1.8. 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.

3.RESULTS

This chapter consist an example of the TPA method. This chapter describes how we can use the TPA method to find the test effort and produce a more cost effective and reliable software. Stubs and drivers are reused in this model so that time and cost can be reduced as less test efforts are applied when we reuse the stubs and drivers.The drivers and stubs may have bugs themselves that result in a lot of additional debugging effort. Automationof code generation for drivers and stubs can result in a useful saving of effort for thetester. It also will ensure that there are no defects in the stubs or drivers that results inavoidable loss of time.
DCM Data Systems Ltd. had a number of software products. One of the newly developed products was installed locally and abroad. It is found that some of the program functionality claimed did not adequately function. The management of the company then handed over the project to a LEVEL 5 company---KR V&V. KR V&V decided to use TPA method to estimate the testing effort. System study by KR V

and V requests a 2 day systems and requirements

Weight without stubs and drivers	Weight with stubs and drivers	Category	Rating
10%	10%	Low intensity	3
70%	70%	Medium intensity	6
[illegible]	[illegible]	[illegible]	[illegible]

Weight without stubs and drivers	Weight with stubs and drivers	Category	Rating
0	0	Low complex	3
100%	100%	Medium complex	6
0	0	High complex	12

and
PA
ing
urs
by
the
and
compare the result, when the coding is done without writing stubs and drivers and when stubs and drivers are written and reused for minimized the cost of rewriting code again and again. The data count is 650 and transaction count is 600 for this project.
All this data of the company is used in this paper to calculate the test point analysis.

User importance (U_p): It implies how important the function to the users related to other system function is

.Table 4.1: User importance

Usage intensity (U_i): It depicts how many users process a function and how often.
Weights:

Weight without stubs and drivers	**Weight with stubs and drivers**	**Category**	**Rating**
20%	20%	Low importance	3
60%	60%	Medium importance	6
20%	20%	High importance	12

Table 4.2: Usage intensity

Interfacing (I): It implies how much one function affects other parts of the system.
Weights:

Weight without stubs and drivers	Weight with stubs and drivers	Category	Rating
50%	25%	Low interfacing	2
	25%	Medium interfacing	4
50%	50%	High interfacing	8

Table 4.3: Interfacing

Complexity (C): The complexity of a function is determined on the basis of its algorithm. The complexity rating of the function depends on the number of conditions in the functions algorithm.
Weights:

Table 4.4: Complexity

Uniformity factor (U): It checks the reusability of the code.

Weights:

Weight without stubs and drivers	Weight with stubs and drivers	Category	Rating
40%	60%	Repetitive test cases	0.6
60%	40%	Unique testcases	1

Table 4.5: Uniformity factor

Dynamic quality characteristics (Q_d): Four dynamically explicit measurable quality characteristics are defined in TPA.

Usability –Characteristics relating to the effort needed for use and on the individual assessment of such use by a set of users.
Weights:

Weight without stubs and drivers	Rating	Weight with stubs and drivers	Rating
Highly important	5	Highly important	5

Table 4.7: Usability

Suitability – This characteristics relating to the achievement of the basic purpose for which the software is being prepared.
Weights:

Weight without stubs and drivers	Rating	Weight with stubs and drivers	Rating
Medium important	4	Extremely important	6

Table 4.8: Suitability

Security –Ability to prevent unauthorized access.
Weights:

table 4.9 security

Efficiency- characteristics related to the relationship between the level of performance of software and the amount of resources used.
Weights:

Table 4.10: Efficiency

3.1 CALCULATION OF TPA WITHOUT STUBS AND DRIVERS:

1. **Dynamic test point**: $D_t = FP_f * D_f * Q_d$

Where, $FP_f =$ Transaction FP = 600 (given)

D_f = Dependency Factor = Weighted rating on Importance to user, usage intensity, interfacing of functions, complexity of functions.

- Rating on user importance(U_p):
  U_p= 3*20%+6*60%+12*20%
  =0.6+3.6+2.4 =6.6
- Rating on usage intensity(U_i):
  U_i=2*10%+4*70%+12*20%
  = 0.2+2.8+2.4 =5.4
- Rating on interfacing (I):
  I= 2*50%+8*50% =5
- Rating on Complexity (C):
  C= 6(nominal complexity)

$D_f = (U_p + U_i + I + C)/ 20 * U$

U =Uniformity Factor= 60%*1+40%*0.

= 0.6+ 0.24 =0.84

$D_f = (U_p + U_i + I + C)/ 20 * U$

D_f = (6.6+5.4+5+6)/20 *0.84 =0.97

Q_d = Dynamic quality characteristics = weighted score on following 4 quality characteristics:

- Suitability(weight=0.75, medium importance—rate =4)
- Security (weight=0.05, extremely importance—rate =6)
- Usability(weight=0.10, highly importance—rate =5)
- Efficiency(weight=0.10, medium importance—rate=4)

So,

weighted score = (0.75*4+0.05*6+0.10*5+0.10*4)

Q_d = 3+0.3+0.5+0.4= 4.2

Hence,

$D_t = FP_t * D_f * Q_d$

D_t =600 *0.97 *4.2=2444.4

2. Static test point

S_t=total FP * Q_i/500

Total FP = Data FP+ Transaction FP= 650+600= 1250

S_t=total FP * Q_i/500

=1250*80/500 =200

3. Total test point

TP= D_t+ S_t = 2444.4+200= 2644.4

4. Productivity Factor (PF) = 1.4 tests hours per test point

Rating on test tools=1

Rating on development testing =4

Rating on test basis = 6

Rating on development environment =2

Rating on test environment =2

Rating on test ware =2

5. Environmental Factor

EF =1+4+6+2+2+2/21 =0.81

Weight without stubs and drivers	Rating	Weight with stubs and drivers	Rating
Extremely important	6	Extremely important	6

6. Primary test hours

P=TP* PF *EF=2644*1.4*0.81 = 2999

Planning control allowance =6%+2% = 8%

7. Total test hours = P+ 8% of P

=2999+8% of 2999 = $\boxed{3239}$

3.2 CALCULATION OF TEST HOURS WITH STUBS AND DRIVERS:

1. Dynamic test point: $D_t = FP_f * D_f * Q_d$

Where, FP_f=Transaction FP = 600 (given)

D_f = Dependency Factor = Weighted rating on Importance to user, usage intensity, interfacing of functions, complexity of functions.

- Rating on user importance(U_p):
  U_p= 3*20%+6*60%+12*20%
  =0.6+3.6+2.4 =6.6
- Rating on usage intensity(U_i):
  U_i =2*10%+4*70%+12*20%
  = 0.2+2.8+2.4 =5.4
- Rating on interfacing (I):
  I= 2*25%+4*25%+8*50% =5.5
- Rating on Complexity (C):
  C= 6(nominal complexity)

$D_f = (U_p + U_i + I + C)/ 20 * U$

U =Uniformity Factor= 60%*0.6+40%*1

Weight without stubs and drivers	Rating	Weight with stubs and drivers	Rating
Medium important	4	Medium important	4

= 0.36+ 0.4 =0.76

$D_f = (U_p + U_i + I + C)/ 20 * U$

D_f = (6.6+5.4+5.5+6)/20 *0.76 =0.89

Q_d = Dynamic quality characteristics = weighted score on following 4 quality characteristics:

- Suitability(weight=0.75, medium importance—rate =4)
- Security (weight=0.05, extremely importance—rate =6)
- Usability(weight=0.10, highly importance—rate =5)
- Efficiency(weight=0.10, extremely importance—rate=6)

so, weighted score = (0.75*4+0.05*6+0.10*5+0.10*6)

Q_d = 0.6+0.3+3+0.5= 4.4
Hence,
$D_t = FP_t * D_f * Q_d$
D_t =600 *0.89 *4.4=2349.6

2. Static test point
S_t=total FP * Q_i/500
Total FP = data FP+ transaction FP= 650+600= 1250
S_t=total FP * Q_i/500
=1250*80/500 =200

3. Total test point
TP= D_t+ S_t = 2349.6+200= 2549.6

4. Productivity Factor (PF) = 1.4 tests hours per test point
Rating on test tools=1
Rating on development testing =4
Rating on test basis = 6
Rating on development environment =2
Rating on test environment =2
Rating on test ware =2

5. Environmental Factor
EF=1+4+6+2+2+2/21 =0.81

6. Primary test hours
P=TP* PF *EF=2549.6*1.4*0.81 = 2891
Planning control allowance =6%+2% = 8%

7. Total test hours = P+ 8% of P
=2891+8% of 2891 = $\boxed{3122}$

4.CONCLUSION AND FUTURE WORK

Testing effort is the number of hours that is required for the testingprocess 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.
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 beneficialthan 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.

5. REFERENCES

[1] Nick Jenkins, et. el, “A Software Testing primer, An Introduction to Software testing”, e-book, 2008.
[2] Dr.-Ing Michael Kaiser, et. el., “The V Model of project execution specification phases & QA, iXIT Engineering Technology”, GmbHQA-IX04-ProjExecution-&BDB01-0003, Feb 2006.
[3] Raymond Lewallen, et. el., “ Software Development Life Cycle”, 2005.
[4] Hee-GyunYeom and Sun-Myung Hwang, et. el., “ A Study on Tool for supporting the Software Process Improvement”, International Journal of Software Engineering and Its Applications,Vol.3, No.2, April 2009.
[5] Jakobsson, et. el., “ V-Model Testing –Process model configuration using SVG”, PMoC 14/04/2003, Version 1.5.
[6] Goldsmith, Robin F, et. el., “Software Development Magazine”, 4-part series, July-October, 2002.
[7] Brian Marick, et. el., “ New models for test development, Reliable Software Technologies, 1999. Version 1.0 of 03/28/00.
[8] Christian Bucanac, et. el., “ The V Model”, University of KarlskronaRonneby, 1999.
[9] IABG Technology: V-Model, Lifecycle Process Model, 1999.
[10] Sira Vegas, Natalia Juristo, and Victor R. Basili, et. el., “ Maturing Software Engineering Knowledge through Classifications: A Case Study on Unit Testing Techniques”, IEEE Transactions on Software Engineering, Vol. 35, No. 4, July/August 2009.
[11] Jaya Gupta Asanani, et. el., “ V-Model (Software Development)”.
[12] Dr. Dwayne L. Knirk, et el, “ software Testing Process Improvements”, Thirteenth International Conference on Testing Computer, Software Sandia National Laboratories,NM 87185-0638.
[13] Patrick Oladimeji, Dr. Markus Roggenbach, and Prof. Dr. HolgerSchlingloff, et. el., “ Levels of Testing, Advance topics in Computer Science”, University of Wales.
[14] Bor-Yuan Tsai, Simon Stobart, Norman Parrington and Barrie Thompson, et. el., “Iterative Design and Testing

within the Software Development Life Cycle", Software Quality Journal, 6(4),pp295-309. December 1997.

[15] Mary Jean Harrold, et. el., " Testing: a roadmap,In Proceedings of the conference on the future of software engineering",pp 61–72. ACM Press, 2000.

[16] Vijay.N,et.el., "Little Joe Model of Software Testing", Software Solutions Lab, Honeywell, Bangalore, PACT-Product Assurance and Capability Team.

[17] Andreas Leitner, IlincaCiupa, Manuel Oriol, Bertrand MeyerArnoFiva, et. el., " Contract Driven Development =Test Driven Development – Writing Test Cases", ESEC/FSE'07, September 2007, ACM 978-1-59593-811-4/07/0009.

[18] Kuhn, D.R. and D.R. Wallace, et. el., " Software Fault Interactions and Implications for Software Testing", IEEE, Trans. Softw. Engg, pp 418-421, 2004.

[19] MaaretPyhäjärvi, KristianRautiainen and JuhaItkonen, et. el., "Increasing Understanding of the Modern Testing Perspective in Software ProductDevelopmentProjects,Proceedings of the 36th Hawaii International Conference on System Sciences – 2003.