Test Effort Estimation With & Without Stub And Driver Using Test Point Analysis (TPA)

Adesh kumar[1], Vikas Beniwal[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
Development of software has initiated the new role of software testing. At the beginning of software products development the majority of the testing was performed by the developer himself due to the simplicity of the product. As the complexity of software products has increased, the role of all parts in the software development process has been modified including with the role and importance of the testing process. The testing process has important position in the process of software product development, 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.

Keywords
TPA, Test point, Driver, Stub etc.

I. Introduction
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.

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.

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.

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.

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.

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.

II. TPA Approach for Estimation

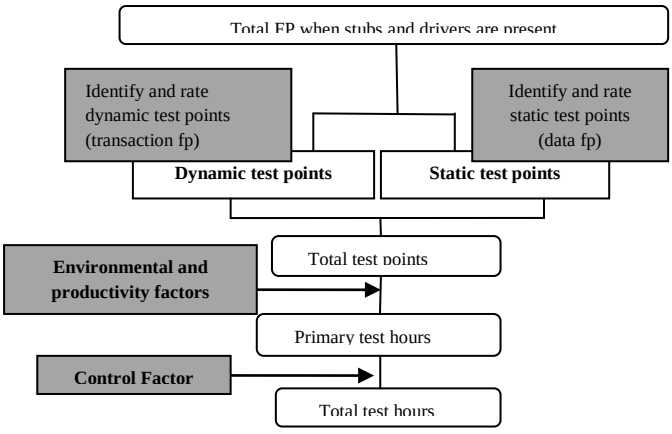

Figure 1: Derived TPA model

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.

III. Results

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 study to understand the scope of testing work and assess the testing requirement to arrive at TPA estimate. Earlier experience of KR V and V using TPA technique suggests it requires 1.4 tests per hours per unit test point. FP count is estimated earlier by using FPA estimate technique and then applies the TPA method to calculate the testing effort 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.

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

Weights:

Weight with-out stubs and drivers	Weight with stubs and drivers	Category	Rating
30%	30%	Low importance	3
40%	40%	Medium importance	6
30%	30%	High importance	12

Table 3.1: User importance

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

Weight with-out stubs and drivers	Weight with stubs and drivers	Category	Rating
20%	20%	Low intensity	3
60%	60%	Medium intensity	6
20%	20%	High intensity	12

Table 3.2: Usage intensity

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

Weight with-out stubs and drivers	Weight with stubs and drivers	Category	Rating
25%	25%	Low interfacing	2
25	25%	Medium interfacing	4
50%	50%	High interfacing	8

Table 3.3: 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:

Weight without stubs and drivers	Weight with stubs and drivers	Category	Rating
10%	10%	Low complex	3
80%	80%	Medium complex	6
10%	10%	High complex	12

Table 3.4: Complexity

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

Weight with-out stubs and drivers	Weight with stubs and drivers	Category	Rating
40%	60%	Repetitive test cases	0.6
60%	40%	Unique test cases	1

Table 3.5: Uniformity factor

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

Quality characteristics	Weight
Functionality	0.75
Security	0.05
Usability	0.10
Efficiency	0.10

Table 3.6: Dynamic quality characteristics

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

Weight with-out stubs and drivers	Rating	Weight with stubs and drivers	Rating
Highly important	5	Highly important	5

Table 3.7: 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	Medium important	4

Table 3.8: Suitability

Security –Ability to prevent unauthorized access.
Weights:

Weight with-out stubs and drivers	Rating	Weight with stubs and drivers	Rating
Extremely important	6	Extremely important	6

Table 3.9: Security

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

Weight with-out stubs and drivers	Rating	Weight with stubs and drivers	Rating
Medium important	4	Extremely important	6

Table 3.10: 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*30%+6*40%+12*30%

=0.9+2.4+3.6 =6.9

- Rating on usage intensity(U_i):

U_i=2*20%+4*60%+12*20%

= 0.4+2.4+2.4 =5.2

- Rating on interfacing (I):

I= 2*25%+4*25%+8*50%

=0.5+1.0+4.0=5.5

- Rating on Complexity (C):

C= 3*10%+6*80%+12*10%

=0.3+4.8+1.2=6.3

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

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

= 0.6+ 0.24 =0.84

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

$D_{f=}$ (6.9+5.2+5.5+6.3)/20 *0.84 =1.0

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 *1.0 *4.2=2520

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 = 2520+200= 2720

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=2720*1.4*0.81 = 3085

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

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

=3085+8% of 3085 = **3332**

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*30%+6*40%+12*30%

=0.9+2.4+3.6 =6.9

- Rating on usage intensity(U_i):

U_i=2*20%+4*60%+12*20%

= 0.4+2.4+2.4 =5.2

- Rating on interfacing (I):

I= 2*25%+4*25%+8*50%

=0.5+1.0+4.0=5.5

- Rating on Complexity (C):

C= 3*10%+6*80%+12*10%

=0.3+4.8+1.2=6.3

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

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

= 0.36+ 0.4 =0.76

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

$D_{f=}$ (6.9+5.2+5.5+6.3)/20 *0.76 =0.9

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 *0.9 *4.4=2376

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 = 2376+200= 2576

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=2576*1.4*0.81 = 2922

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

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

=2922+8% of 2922 = **3156**

IV. 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. 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.

V. 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.

VI. References

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

[2]. Dr. Ing Michael Kaiser,"The V Model of project execution specification phases & QA", *iXIT Engineering Technology GmbH QA-IX04-ProjExecution-* &BDB01-0003/ 2006-02-28, http://www.ixit.de

[3]. Raymond Lewallen,"Software Development Life Cycle",2005

[4]. Hee-Gyun Yeom, and Sun-Myung Hwang, "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,"V-Model Testing –Process model configuration using SVG", PMoC 14/04/2003 Version 1.5

[6]. Brian Marick,"New models for test development" *Reliable Software Technologies,*1999. Version 1.0 of 03/28/00

[7]. John Callahan, George Sabolish,"A Process Improvement Model for Software Verification and Validation",*Proceedings of 19th Annual Software Engineering Workshop,* NASA Goddard Space Flight Center, Greenbelt, MD, November 30-December 1, 1994.

[8]. Dr. Dwayne L. Knirk,"software Testing Process Improvements" *Thirteenth International Conference on Testing Computer* ,Software Sandia National Laboratories,NM 87185-0638

[9]. Bor-Yuan Tsai, Simon Stobart, Norman Parrington and Barrie Thompson "Iterative Design and Testing within the Software Development Life Cycle" *Software Quality Journal*, 6(4), December 1997, 295-309

[10]. Mary Jean Harrold,"Testing: a roadmap",*In Proceedings of the conference on the future of software engineering*, pages 61–72. ACM Press, 2000.

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

[12]. Andreas Leitner, Ilinca Ciupa, Manuel Oriol, Bertrand Meyer Arno Fiva, "Contract Driven Development =Test Driven Development – Writing Test Cases" ESEC/FSE'07, September 2007 ACM 978-1-59593-811-4/07/0009

[13]. Kuhn, D.R. and D.R. Wallace, "Software Fault Interactions and Implications for Software Testing."*IEEE Trans. Softw. Eng.*, 2004. 30(6): p. 418-421.

[14]. R.Venkat Rajendran, Director, Deccanet Designs Ltd., "White paper on Unit Testing"

[15]. R. C. Bryce, A. Rajan, and M. P. E. Heimdahl, "Interaction testing in model-based development: Effect on model-coverage". *Proc. of the 13th Asia-Pacific Software Engineering Conf.*, pages 258--269, Dec.2006.