International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 21 (2017) pp. 10861-10866
© Research India Publications. http://www.ripublication.com

Testing Sufficiency Test (TST) - Evolving a New Model for Estimating Software Test Cases

Bishan Dayal Chauhan[1], Dr. Ajay Rana[2] and Dr. Neeraj Sharma[3]

[1]Research Scholar, Amity School of Engineering and Technology, Amity University, Noida UP, India.
Orcid Id: 0000-0002-6462-8572
[2]Professor, Amity School of Engineering and Technology, Amity University, Noida UP, India.
[3]Professor, Harlal Institute of Management & Technology (HIMT), Greater Noida, UP, India.

Abstract

One of the major challenges faced by project managers is to control huge testing efforts being spent on testing of the software, and thus controlling cost and quality of software. For controlling cost & quality, One of the key project objectives is to plan for optimum testing efforts, so that software produced is of high quality, and this has to be achieved by applying the right amount of testing efforts, if less testing efforts are applied quality of software suffers, and if more efforts are applied then cost of the software increases, so it becomes very important to strike a right balance between cost & quality of software and thus needed to arrive at optimum amount of testing, which becomes a major challenge for managers as it requires a good prediction model to predict the right amount of testing. It is proven that the software testing phase is one of the most critical and important phases in the software development life cycle. In general, the software-testing phase takes around 40-65% of the effort, time and cost. This area has been well researched over a long period of time. Unfortunately, while many researchers have found methods of reducing time and cost of the testing phase, there are still a number of important related issues that need to be researched. This study introduces a new prediction model for estimating number of required test cases and thereby maintaining the right balance between quality & cost, using empirical approach which establishes a relationship between software size & testing efforts using the defect data, productivities for various technologies can also be used for making the testing efforts technology specific for various projects. Predicting optimum number of required test cases or steps and thus total efforts required for testing and thereby controlling overall cost of quality

Keywords: Optimum Testing Efforts, Optimum Quality, Cost Reduction, Cost of, Quality (COQ), Prediction Model, Number of Test Cases, Test Case Estimation

INTRODUCTION

Imagine driving to an important trip to a distant place where you have never been for. No-one in their right mind, set off to such a journey without knowing at least the name and the general direction of the destination. Other important considerations are the distance and the available routes that take you there. Armed with this information and a good map one can feel more comfortable about taking the trip.

Managing a Software project is much harder than planning a trip. The biggest difference is that no matter how hard you try, challenges in predicting the right amount of testing to be done for software so that cost and quality of end product in well in control. There are however striking similarities. Knowing the general direction is necessary in both the cases.

What Determines Success of a Software project

The end of a software project is generally the point at which a project is judged to be a success or not. Typical criteria for determining success are

- Did the project completed on time?
- Did the project completed within its budget?
- How is the quality of software?
- Is the software being developed is in good use.

Scope, Time & Cost management is critical. There are other criteria of course; however, these are generally additional to the critical Time, Cost and Quality criteria. In any project, software or other, these are three conflicting factors. One cannot for example lessen the duration without affecting cost and/or quality. Therefore, to manage success of the project is to keep the cost & quality in check and we are targeting this area i.e. controlling cost of quality

What is Cost of Quality?

Cost of Quality (COQ) – is a measure of the cost of the conformance (cost of control) & cost of Non-conformance (Costs of failure of control)

Software Engineering – A Practitioner's Approach – Roger S. Pressman defines as "The cost of quality includes all costs incurred in the pursuit of quality or in performing quality-related activities" [1], numerous studies are done to identify opportunities for reducing the cost of quality.

Costs of the conformance (Costs to ensure quality)

- **Prevention Costs** - Incurred from efforts to keep defects from occurring at all "Do it right first time" Example: Quality Assurance costs
- **Appraisal Costs** - Incurred from detecting defects via inspection, test; audit i.e. "Did we get it right?"

  Example: Quality Control costs

Costs of Non-conformance (Costs to fix poor quality)

- **Internal failure costs** - Incurred from fixing/repairing defects caught internally.
  - Example: Cost of Rework (Fixing of internal defects and re-testing)
- **External failure costs** - Incurred from defects that actually reach customers.

  Example: Cost of Rework (Fixing of external defects and re-testing) and any other costs due to external defects (Product service / liability / recall, loss of reputation, complaints in warranty & out of warranty etc.)

Definition by ISTQB (International Software Testing Qualifications Board): cost of quality: The total costs incurred on quality activities and issues and often split into prevention costs, appraisal costs, internal failure costs and external failure costs.

Definition by QAI (Quality Assurance International): Money spent beyond expected production costs (labor, materials, and equipment) to ensure that the product the customer receives is a quality (defect free) product. The Cost of Quality includes prevention, appraisal, and correction or repair costs.

Why to measure cost of quality? "You cannot manage what you cannot measure"

- Benchmark against peers in industry
- Articulate costs and benefits in Money terms
- CoQ provides a holistic and quantitative view of process maturity
- Identify focus areas for software process improvement

Cost of Quality (COQ) = **(1)**

Cost of Conformance

(Prevention Cost + Appraisal Cost)

+

Cost of Non-Conformance

(Internal Failure Cost + External Failure Cost)

Where

Cost of Control = Prevention Cost + Appraisal Cost

And

Cost of Failure of Control = Internal Failure Cost + External Failure Cost

Software Testing & Size

Software sizing is an important activity in software engineering that is used to estimate the size of a software application or component in order to estimate other software project characteristics like efforts, schedule, resources, defects etc. Size is an inherent characteristic of a piece of software just like weight is an inherent characteristic of a tangible material. Size is not effort, it is essential to differentiate between software sizing and software effort estimation. Measuring the size of software is different from measuring the effort needed to build it. Size is an independent measure and it does not depend on technology while effort will depend on many factors.

In general, we present size estimates as lines of code (KSLOC or KLOC or SLOC), function points, uses case count, object count. Selection of sizing unit depends on the nature of the project and also the form in which requirements are presented. There are various scientific methods for calculating the software size that can be used based on the project type.

Example: If a software application of size 5000 LOCs (Lines of Code) or 30 Function Points is to be developed, and productivity (somewhat like speed) of engineering team is 1 Function Point / Day or 166 LOCs /day, then total effort required to develop the software can be computed as

Efforts Required = Software Size / Productivity **(2)**

= 5000/166 = 30 Person Days

Or

= 30/1 = 30 Person days

RESEARCH METHOD

The objective of this study is to perform extensive analysis towards developing prediction model for testing efforts and eventually facilitating strategic planning. Often Project Managers or people responsible for quality of the software have a big question in their mind that,

- Are we doing right amount of testing?
- Are we doing enough testing for uncovering all the defects in software?
- Determination of that if sufficient testing is done?

Data Source

Data used in this study is taken from Benchmarking Release 10 by the International Software Benchmarking Standard Group (ISBSG)[2]. The ISBSG established in 1994 a not for profit organization that has been established to improve the global understanding of software practices, so that they can be better managed. ISBSG has gathered on 4,106 software projects from around the world, and made available on Release 10 of Estimating, Benchmarking & Research Suite CD.

Methodology

We have addressed these problems & researching the effective solution to these problems

- Extensively worked on Testing Strategy – mainly for determination of sufficiency of the testing.
- Solution devised not only answers the questions mentioned but also reduces the overall cost of testing phase.

Following steps were followed

i. For sizing initially lines of code (LOC) is used as a measure

ii. Determining base value of no of test steps

iii. Applying base value of test steps per KLOC vs. actual test steps per KLOC

Let us first understand, how does size of software is related to no. of defects Injection?

In smaller software i.e. small in size, it is highly likely that developers inject fewer defects compared to larger size software.

In larger size software defects injection by developers is likely to be more as it is bigger and has more user functionality.

Now, from this it is evident that software size is directly proportional to defects count. Now, Assuming number of defects logically depends on Size, i.e. more the size number of defect.

Defects ∞ Size **(3)**

And also, if there more defects injected into software then more amount of testing is required to uncover all the defects injected

Defects Count ∞ Required Testing **(4)**

From these two dependencies, we can logically infer that size and required amount of testing can be seen related

Software Size ∞ Required Testing **(5)**

However, we cannot increase the testing efforts too much, for big size requirement. So, need arises to quantify this relationship of dependency

Now, let us consider simplification of this dependency is a major problem, which we have to provide solution for.

Testing process as a whole starts with writing of testing steps for testing the software, It would be amazing to the know that how many testing steps would be sufficient for testing the whole software and which are enough to uncover all the defects. Here we introduce an empirical approach to address this problem

Results

"Access to accurate historical data helps projects counter unrealistic expectations and negotiate plans that support quality instead of undermining it" Quantitative Software Management (QSM) [3]

Let's take a project we finished and delivered for user testing, and take the data for this project, for the all the programs of this project i.e. if this project has twenty five programs, take data for all twenty five programs covering the fields "Lines of Code" "Defects" – Defects leaked to the user, "Test Steps Prepared", now calculate no. Test Steps per Kilo Lines of code i.e. "No of Test Steps per KLOC".

Arranging all these data lines in ascending order of "No of Test Steps / KLOC", now on observing defect count trend, we can see as "No of Test Steps / KLOC" is increasing defect count is decreasing.

From decreasing trend we can select optimum band of programs for lower limit & higher limit of "No of Test Steps / KLOC", mean of this can be taken as base value of No of Test Steps per KLOC.

Base Value = Mean (Test Steps/KLOC) of lower line & higher line, see yellow band in below table 1 i.e. from program no 10 to 16, 10th program marks a lower band & 16th program make a higher band.

In below data (as specified in Table 1), (36+51)/2 = 43, so 43 is base value now computing the required no of Test Steps as per this computed base value.

Now computing "Test Steps Prepared vs. Base Value" i.e.

$$\frac{\text{Test Steps Prepared}}{\text{Test Steps required as per Base Value}} = \% \qquad \textbf{(6)}$$

If % is less than 100% then testing is insufficient i.e. more test steps should be been written. If it is more than 100% then excess testing has been done & thus incurred extra cost which should have saved.

Table 1: Software Programs data along with test steps required

Program No.	LOC	Defect#	Test Steps Prepared	Test Steps / KLOC	Test Steps Req. as per base value	Test Cases Prepared Vs Base Value	Result
1	2174	0	22	10	93	24%	No
2	5184	2	57	11	223	26%	No
3	6254	3	74	12	269	28%	No
4	894	1	18	20	38	47%	No
5	833	0	18	22	36	50%	No
6	4634	2	105	23	199	53%	No
7	1534	5	45	29	66	68%	No
8	884	0	27	31	38	71%	No
9	1516	2	50	33	65	77%	No
10	862	2	31	36	37	84%	No
11	664	0	25	38	29	88%	No
12	700	1	28	40	30	93%	No
13	1783	1	72	40	77	94%	No
14	235	0	10	43	10	99%	No
15	556	1	26	47	24	109%	Yes
16	572	0	29	51	25	118%	Yes
17	455	0	24	53	20	123%	Yes
18	706	0	43	61	30	142%	Yes
19	257	0	19	74	11	172%	Yes
20	281	0	26	93	12	215%	Yes
21	587	0	56	95	25	222%	Yes
22	338	0	46	136	15	316%	Yes
23	249	0	38	153	11	355%	Yes
24	425	0	66	155	18	361%	Yes
25	340	0	62	182	15	424%	Yes

Now using this computed base value can be applied a project which is new, in planning stage of testing we can use the base value of the previous recent project.

So, if the project-2 is consisting of 14 programs then, then 43 as a base value can used for sufficiency of testing in advance, this can save the efforts and therefore cost can be saved. In below table 2 program wise illustration is provided to check for sufficiency of testing.

Gap between required test steps & prepared test steps is highlighted using following figure 1: gaps are highlighted for which less test cases are prepared than required and where more test cases are more than required, so these gaps need to be addressed.

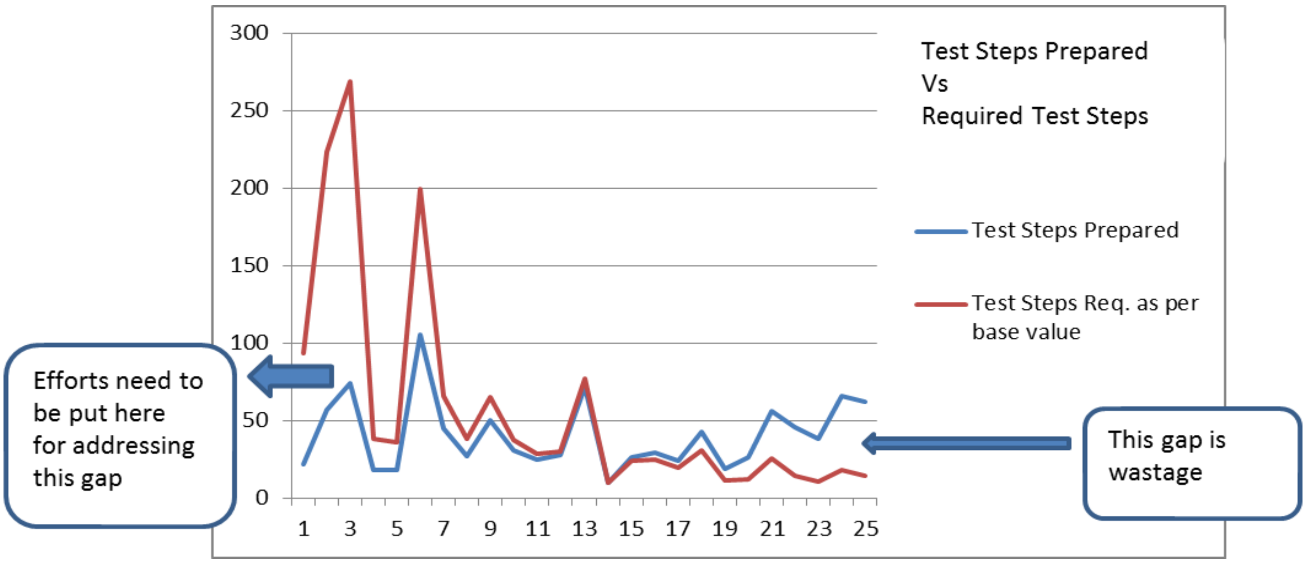

Figure 1: Comparison of Test Steps Prepared Vs. Test Steps Required

Illustration :

Program Name	Test Steps Req. as per base value	Test Steps Prepared	Test Cases Prepared Vs Base Value	Result
Prg-1	51	50	98%	Insufficient Testing
Prg-2	46	48	104%	Sufficient Testing
Prg-3	40	23	58%	Insufficient Testing
Prg-4	30	11	37%	Insufficient Testing
Prg-5	50	48	96%	Insufficient Testing
Prg-6	40	55	138%	Sufficient Testing
Prg-7	62	48	77%	Insufficient Testing
Prg-8	43	79	184%	Sufficient Testing
Prg-9	32	110	344%	Sufficient Testing
Prg-10	27	141	522%	Sufficient Testing
Prg-11	17	23	135%	Sufficient Testing
Prg-12	40	111	278%	Sufficient Testing
Prg-13	30	33	110%	Sufficient Testing
Prg-14	20	22	110%	Sufficient Testing

DISCUSSION

As specified by “Research and realization of software testing model based on CSCW”, Wenjian Liang, Xiufen Fu[4] - Using the reasonable model for software testing can reduce the expenses of testing in the developing activity in the course of testing, thus cut down the cost of software development, we have tried to cut down the cost of quality of software development, We have established the a method for computing count of test cases required using the benchmarking data published by International Software Benchmarking Service Group[2]. Our study results shows that based on the size of a program, number of test cases or steps required can be predicted for a program using base matrix created empirically by considering projects completed in past. This gives number of required test cases and eliminate the need of preparing too many test steps. Our findings are in line with the finding published by Jorgensen, M. and Boehm, B. “Software Development Effort Estimation: Formal Models or Expert Judgment?” IEEE Software [5] in his research edition 2009. Our study shows that to compute the efforts in person hours we need to consider technology wise productivity. Our finding also in line with numerous work listed by Alessandro Orso, Gregg Rothermel – “Software Testing: A Research Travelogue (2000–2014")”[6] – our study gives empirical evidence for saving the overall cost of quality and at the same time maintain high quality of software thus maintain balance between cost & quality of software and ensuring the success of the project.

CONCLUSION

We can conclude that cost of quality (COQ) can be saved using this prediction model specified by this paper and at the same time maintaining high software quality of the software. Also, we could establish the relationship between size & testing of the software. Technology wise productivity figures can also be used to determine technology specific testing efforts, as software size is independent of technology.

A new prediction model is evolved for estimating / predicting required number of test steps per Kilo lines of code – a measure of size, as an empirical approach which is practical and also have good evidence in past projects data as evident by equations (1) to (6)

First, a base value matrix of Test Steps is prepared based on the completed projects data, for various different projects types, Base value of testing steps is determined based on the lower band & higher band, which typically describes a range of Test Steps per Kilo Lines of code.

Then this matrix can be used to compute the sufficiency of testing for various programs in another project of similar nature as illustrated in detailed approach, required number of test steps are determined using the prediction model of base value matrix.

But we should remember, we cannot be absolutely sure that the software will not fail, but definitely with a sound and experimentally validated statistical model, we can say that we have done sufficient testing, say with 98 percent confidence

REFERENCES

[1] Software Engineering – A Practitioner’s Approach – Roger S. Pressman, Fifth Edition Published by McGraw-Hill

[2] ISBSG “The Benchmark data for Software estimation” Release 10 (2011)

[3] QSM Software Almanac, Application Development Series. 2014 Research Edition.

[4] Wenjian Liang, Xiufen Fu ; Zhenkun Li ; Rong Xiao ; Junfeng Hu – Research and realization of software testing model based on CSCW - Computer Supported Cooperative Work in Design, 2005, IEEE, Proceedings of the Ninth International Conference on (Volume:2), Page 704 - 709 Vol. 2, May 2015

[5] Jorgensen, M. and Boehm, B. “Software Development Effort Estimation: Formal Models or Expert Judgment?” IEEE Software, March-April 2009, pp. 14-19

[6] Alessandro Orso, Gregg Rothermel – “Software Testing: A Research Travelogue (2000–2014")”, 2014