International Journal of Advanced Technology in Engineering and Science www.ijates.com
Volume No.03, Special Issue No. 02, February 2015 ISSN (online): 2348 – 7550

IMPACT OF SIZE AND PRODUCTIVITY ON TESTING AND REWORK EFFORTS FOR NON-WEB-BASED DEVELOPMENT PROJECTS

[1]Mridul Bhardwaj, [2]Prof (Dr.) Ajay Rana

[1,2]Amity School of Engineering and Technology, Amity University, Noida UP, (India)

ABSTRACT

In project planning of any software development project, major challenge faces by project managers is to predicate the amount of re-work required to deliver the software that not only meets time and cost requirements but also the quality requirements given by client. To ensure the quality of the software, many iterations of testing cycle conducted before it final delivered to client for acceptance. Each testing cycle is a very costly affair as it involves running all possible test scenarios in all possible user environments and followed by defect fixing and re-verification of defects. On an average, there are 2-3 iterations of testing cycle conducted but this depends on number of defects identified during testing. Number of defects will depend on the expertise and experience of the team to work on similar type of projects and technology. Hence it become very critical to predict and control the numbers of defects identified during testing but this is very challenges task as it requires a good predicating model to predict the re-work effort.In this paper, we describe the relationships among software size, number of software defects, productivity and efforts for non-web-based development projects. These relationships are established by using the multiple linear regression technique on the benchmarking data published by International Software Benchmarking Standard Group.Results suggest that lower productivitywill led to more number of defects; therefore, non-web-based project should be planned with experience resources who have earlier worked on similar projects and technologies. It will help in reducing the number of defects and rework efforts. Executing non-web based project with experience team require less time for development and most of the defect can be identified during unit testing and this will directly contribute in reducing the rework efforts. Relationship among defect, size and efforts also suggest that size has a significant impact on the total number of defects in comparison to efforts. Change in unit software size will have bigger impactin comparison to unit change in effort. So study also infer that while planning the software project we should use appropriate tools to reduce the margin of error in size estimation and we should re-estimate the size, after every phase of development life cycle, to re-calibrate overall efforts and to minimize the impact on the project plan.

Keywords: Re-Work Efforts, Software Size, Productivity, Defects, Software Quality, Software Cost, Project Delivery Rate, Multi Liner Regression, Non-Web Bases Project.

1 INTRODUCTION

In the planning of any software development project, one of the major challenge faced by project managers is to predict the amount of rework required to deliver the software that not only meets time and cost requirements but also the quality requirements given by the client. To ensure the quality of the software, many iterations of the testing cycle are conducted before it is finally delivered to the client for acceptance. Each testing cycle is a costly affair as it involves running all possible test scenarios in all possible user environments, followed by defect fixing and re-verification of defects. Typically the test cycles ends when there is no defect to be identified during the test cycle. On an average, two to three iterations of the testing cycle are conducted but this depends on the number of defects identified during testing. The number of defects will depend on the expertise and experience of the team (productivity) to work on similar type of projects and technology. Therefore, it becomes critical to predict the numbers of defects identified during testing and this is a very challenging task as it requires a good model to predict the rework effort. To predict the rework effort we should first estimate the overall size of the software being developed, hence, size estimation becomes the most important first step to plan software project. Importance of software size is explained in my paper "Estimate Software Functional Size before Requirement phase of Development Life Cycle" [12]. Below paragraph and section 1.1 to section 1.3 are the extracted from that paper.

Imagine driving on an important trip to a distant place you have not been before. No-one will feel comfortable to start such a journey without knowing at least general direction of the destination e.g. distance, the available routes, road condition etc. Armed with this Information and a good map one can feel more comfortable about taking the trip. These are essential; however your comfort may be seriously compromised during the journey. Road works can hit you badly, but at least you have a good chance of having early warning, if you think to check. A puncture or a car fault can be less predictable, but they do happen. Managing a Software project is much harder than planning a trip. The biggest difference is that no matter how hard you try, the specification are not static and also challenges in predicting team productivity and quality of end product i.e. no of defects. The biggest challenge with the Software project is to define the size of the overall application as the requirements are not always provided in structured form. There are two type of Software sizing one is based on Functional Size (user perspective) and other is technical Size (developer perspective e.g. KLOC) but technical sizing cannot be used to compare functionality provided by software application. With user point of view, software providing the same functionality will be treated similar irrespective of the technology used so functional size is the only independent variable that not only can be used to compare the software but also to plan software development. There are various Functional Size measurement are available to measure the software functional size but all these methods require detail understanding of functionality but most of time detail understand of requirement available only after completion of requirement phases so the challenge remain to measure the software functional size before the requirement phase. Let's first understand how to define the successful project.

1.1 What Determines a Successful Software Project?

Time, Cost and Quality are key Success parameters to determine if project is successful or not [7]. There are other criteria of course; however these are generally additional to the Time, Cost and Quality criteria. In any project, these are three conflicting factors: These are often viewed as a triangle (see Figure 1). One cannot for example lessen the duration without affecting cost and/or quality. In software engineering terms, functionality is the counterpart for quality, while duration and cost are still applicable. The correctness of the code (lack of bugs) is bundled into quality for simplicity. The attributes we consider for software projects are Effort and Schedule. If these are available and controlled, then the cost and time can also be defined and controlled.

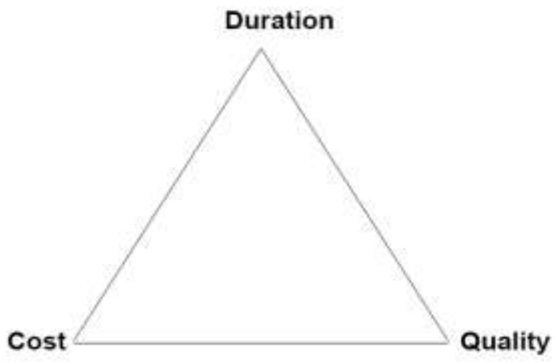


Figure 1 Conflicting Triangle

Duration is the total number of time units (e.g. weeks or months) needed to complete the project. This may break down to effort from more than one person, so as to take advantage of certain skills and parallelize the work to gain overall time. We can reduce the duration by putting more resources but the caveat here is that the more people one adds to a project then more one needs to work so as to coordinate them and the more they communicate so as to interact successfully, thus yielding overheads. This coordination may also lead to occasional idle periods of some of these people. Schedule is sometimes associated with the total time for the project; however the term usually includes the breakdown of effort per person at any given time, so as to track the coordination issues. It is a fact that the schedule is derived from effort

1.2 Software Size, Efforts and Productivity

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 efforts, it is essential to differentiate between software sizing and software effort estimation. Measuring the size of a piece of software is different from measuring the effort needed to build it. There are two type of Software sizing one is based on Functional Size (user perspective) and other is technical Size (developer perspective)[1] but technical sizing cannot be used to compare functionality provided by software application. With user point of view, software providing the same functionality will be treated similar irrespective of the technology used so functional size is the only independent variable that not only can be used to compare the software but also to plan software development.Functional Size is an independent measure and it does not depend on technology while effort will depend on many factors. (See Figure 2).

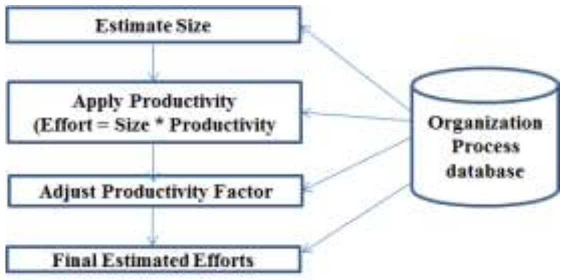

Estimate Size
Apply Productivity
(Effort = Size * Productivity
Adjust Productivity Factor
Final Estimated Efforts
Organization Process database

Figure 2 Driving Efforts Using Size and Productivity

Functional Size Measurement is the most accepted approach to measuring the size of software functional size. There are various sizing methods available like IFPUG function points, COSMIC, NASMA function points etc. but these standard functional size methods are not very suitable to be used in early stage of software life cycle as they require structured analysis of user requirements.[2] Functional size measurement are simple in concept but they are not easy to apply specially in early stage of software development e.g. in proposal stage where not much scope is defined and clear to carry out detail sizing analysis. An accurate estimate of software size is an essential element in the calculation of estimated project costs and schedule. The fact that these estimates are required very early on in the project (often while a contract bid is being prepared) makes size estimation a formidable task. Initial size estimates are typically based on the known system requirements. We must hunt for every known detail of the proposed system, and use these details to develop and validate the software size estimates.

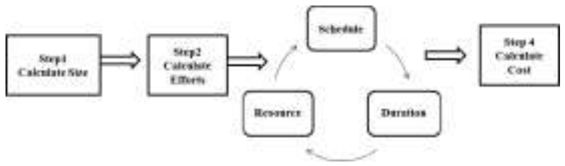


Figure 3

In general, we present size estimates as lines of code (KSLOC 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. Regardless of the unit chosen, we should store the estimates in the metrics database and use these estimates to determine project progress and to also to estimate future projects. As the project progresses, revise them so that cost and schedule estimates remain accurate and corrective action can be taken if required. There are various scientific methods for calculating the software size that can be used based on the project type.
Productivity can be defined as average efforts required to deliver unit size. [3]

1.3 Early Approximation of Functional Size using Logical Data Model

IFPUG functions point count following functional component types [4].
Internal Logical Files (ILF) – Logical Business entities used to store data within the boundary of software.
External Interface Files (EIF) - Logical Business entities referred by Software, Software is not the owner of these entities but these are owned by other software outside the boundary of Software.
External Inputs (EI) –Process insert data to be stored in ILF.
External Outputs (EO) - Process that extract data from ILF or EIF and derived to generate report to be sent outside the boundary of software.
External Query (EQ) - Process that extract data from ILF or EIF and generate report (with no derived data) to be sent outside the boundary of software

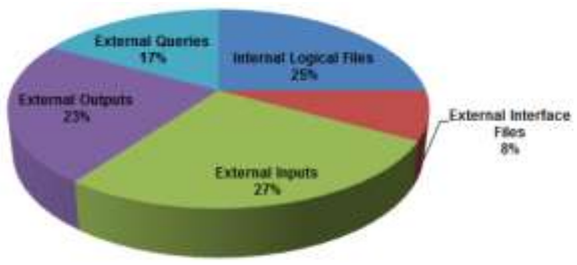


Figure 4 Distribution of Function Points Component in overall Size

Based on the ISBSG data [5], distribution of these five entities are shown in the above pie chart (figure 4). If we can drive the number of internal logical files using the data model, we can approximate the size of the Software. For example if there are 10 internal logical file identified software size can be estimated using the following calculation.
Software size = Number of ILF * 8.6 (mean score of internal logical file) * 4 (as ILF contribute only 25% in total software size as per the pie chart above)
Software size = 10*8.6*4 =344 Function points.
Similarly if we know the number of reports or output generated by the software then we can we can also approximate the size as reports contribute 40% of the total software size.

II OBJECTIVE

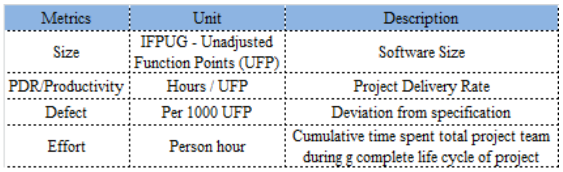
The objective of this study is to perform extensive data analysis towards developing a more in-depth understanding of impact of functional size and productivity on the testing and rework effort non-web basedprojects, and eventually facilitating strategic planning by to control these factors. Hence, the key objectives of the study for this paper are:

- Is there any relation between productivity and defects injected?
- How functional size impact the overall rework effort?
- How functional size impact the overall testing efforts?
- How productivity impact the testing efforts ?

2.1 Data Source

Data used in this study is taken from Benchmarking Release 10 by the International Software Benchmarking Standard Group (ISBSG). 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.

2.2 Unit of Key Metrics

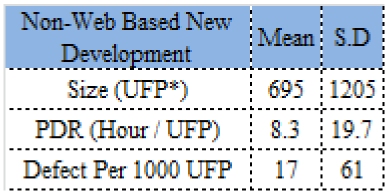


Metrics	Unit	Description
Size	IFPUG - Unadjusted Function Points (UFP)	Software Size
PDR/Productivity	Hours / UFP	Project Delivery Rate
Defect	Per 1000 UFP	Deviation from specification
Effort	Person hour	Cumulative time spent total project team during g complete life cycle of project

Table 1

2.3 Approach

ISBSG has published Mean and Standard deviation of Size, Productivity (Product delivery rate) and Defect data for Non-web based. These data is shown in Table 2.



Non-Web Based New Development	Mean	S.D
Size (UFP*)	695	1205
PDR (Hour / UFP)	8.3	19.7
Defect Per 1000 UFP	17	61

Table 2

Assuming data is normally distributed; following steps were followed to generate data
First generate value for Software Size that is independent variables, by generating positive random variables in the range of 0-1000 (Figure 5)

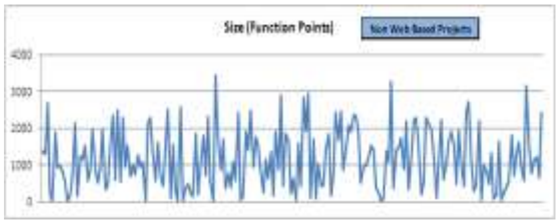


Figure5

Total efforts calculated after multiply size with PDR (Figure 6)

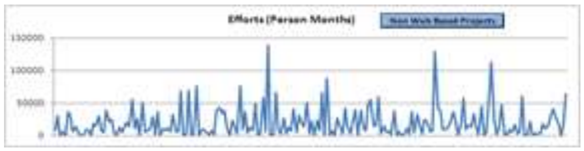


Figure 6

Defect data is also generated based on defect statistics as per table 2 and assuming it is also normally distributed. (Figure 7)

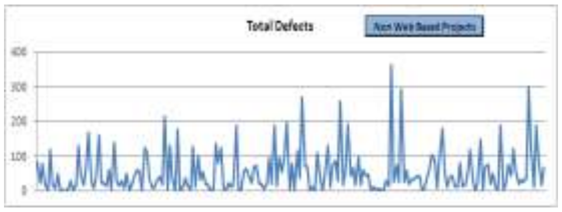


Figure 7

For each data points of size generate value for Productivity / PDR (Product Delivery Rate) based on value mean and standard deviation value as per table 1 after assuming it is normally distributed. Statistical Software MINITAB Release 14 has been used to generate the Value for Normal distribution

III EXPERIMENT RESULTS

Assuming number of defects logically depends on Size, Productivity and Total efforts e.g. Bigger the size higher the number of defect , similarly by more efforts we increase chance of injecting more defects. We tried to establish relationship between number of defect, Size, PDR, Efforts, using Multi linear Regression equation. Following multi liner equations are established based.

$$\boldsymbol{Total\ defects = -11.3 - 0.000272\ Efforts + 0.0529\ Size + 0.538\ PDR\ (1)}$$

It is clear from the above equation that Software size play a pivot role to determine the estimated number of defects, hence, it is very important to estimate software size even if there is very high level software requirements are available.

IV CONCLUSIONS

From the equation 1 , we can infer that higher productivity (means lower value for PDR) will led to the less number of defects so non-web based project should be planned with much experience team(higher productivity) so that we should have less defect and rework. (Please remember that cost of rework is much higher than cost of development). Relationship among defect, size and efforts also suggest that size has much significant impact on total number of defect in comparison to efforts. In multi linear regression equation size co-efficient is much higher than the efforts co-efficient that means size has much significant impact on total number of defect in comparison to efforts.Change in unit size will have bigger impact in comparison to unit change in effort. So study conclude that while planning the software project we should use appropriate tools to reduce the margin of error in size estimation and we should re-estimate the size, after every phase of development life cycle, to re-calibrate overall efforts and to minimize the impact on the project plan.

REFERENCES

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

[2] Jorgensen, M. and Shepperd, M. “A Systematic Review of Software Development Cost Estimation Studies,” IEEE Trans. Software Eng., vol. 33, no. 1, 2007, pp. 33-53

[3] Peter R. Hill “Practical Software Project Estimation, A Tool Kit for Estimating Software Development Effort and Duration” by McGraw-Hill (2011)

[4] International Function Points Users Groups (IFPUG): Supports and maintains the IFPUG methods (release 4.3): www.igpug.com

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

[6] Object-Oriented Software Development Effort Prediction Using Design Patterns from Object Interaction Analysis Systems (ICONS), 2010 Fifth International Conference on Adekile, O.; Simmons, D.B.; Lively, W.M.;

[7] Essential Metrics for Outsourcing Testing Services, Analyst(s) by Gilbert van der Heiden - 2011

[8] Peter R. Hill “Practical Software Project Estimation, A Tool Kit for Estimating Software Development Effort and Duration” by McGraw-Hill (2011)

[9] Phase Distribution of Software Development Effort by Ye Yang, Q ing Wang &Mingshu Li (Institute of Software, Chinese Academy of Sciences, China), Mei He (Graduate University of Chinese Academy of Sciences, China) & Barry Boehm (University of Southern California, USA)- 2008

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

[11] UKHEC Report on Software Estimation - K. Kavoussanakis, Terry Sloanm The University of Edinburgh – 2010

[12] Mridul Bhardwaj and Ajay Rana “Estimate Software Functional Size before Requirement phase of Development Life Cycle” International Journal of Innovations & Advancement in Computer Science, vol. 3 Issue 4 June-2014, pp 79-83