2017 International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE), Khartoum, Sudan

Early Cost Estimation of Software Reworks Using Fuzzy Requirement-Based Model

Tarig Ahmed Khalid
Faculty of Computing and Informatics
Multimedia University
Cyberjaya, Malaysia
t.a.khalid@ieee.org

Eng-Thiam Yeoh
Faculty of Computing and Informatics
Multimedia University
Cyberjaya, Malaysia
etyeoh@mmu.edu.my

Abstract- Defects uncovered during software testing usually consume a considerable amount of the overall project's budget. Unfortunately, project managers are not well-equipped with techniques to estimate such cost overrun. Moreover, incorporating defects removal process as part of the software testing activities has made the project managers overlook this important cost component during their planning processes. This paper suggested a fuzzy model to estimate the cost of defects removal at the early stages of project planning in terms of the project's functional size and the experience of the development team. The suggested model is supposed to enhance the accuracy of the software project cost estimation.

Keywords- software cost estimation; cost of software reworks; cost of defects removal; defects removal; software rework

I. INTRODUCTION

As software development activities complete, software products are to be transferred to the testing environment where defects would be systematically uncovered. After testing, accepted software would then be transferred to the production environment through the implementation phase. Unfortunately, implementing software products immediately after finishing the testing activities is just a day dream in the world of software development. The unavoidable fact is that many defects would be uncovered and many cycles of reworks and retests are to be performed before ending-up with accepted product. Software reworks become a fact of life in software development projects. Furthermore, reworks have negative impacts on software projects in terms of time delays and cost overruns. Charrette [1] argued that software developers spend 40 to 50 percent of their time in reworks activities. Basili et al [2] and Raja and Marietta [3] suggested that cost of software reworks is one of the main concern in software development since cost is an important parameter defining the success of software projects. Zahra et al [4] described reworks as a "prevailing scourge" consuming up to 40 to 70 percent of the overall project's budget. In general, cost of reworks becomes like a tax taken for granted and paid by the software firms as a part of the software development process [5].

This paper suggests the following:

- Splitting the software defects removals from testing so as to be considered as an independent process
- A new fuzzy model for estimating the expected cost of defect removals.

II. REWORKS ASSOCIATED PROBLEMS

Software reworks associated with defects removal are usually considered as part of the software testing process although they have two different definitions. Software testing is defined as "the process of analyzing a software item to detect the differences between existing and required conditions and to evaluate the features of the software item" [6]. Sommerville [7] stated that "testing is intended to show that a program does what it is intended to do and to discover program defects before it is put into use". Moreover, the software engineering body of knowledge (SWEBOK) stated that software testing consists of the dynamic verification that a program provides expected behaviors on a finite set of test cases, suitably selected from the usually infinite execution domain [8]. Nevertheless, none of the above definitions mentioned anything about defects removal.

This paper argued that although defects removal is considered as part of the software testing process, its associated activities are not well-defined in the context of the software testing. This situation resulted in overlooking of the details of the defects removal and its associated impact on the healthiness of the software projects in terms of schedule delays and cost overruns.

We believe that defects removal is a process that is independent from software testing due to the following reasons:

978-1-5090-1809-3/17/$31.00 ©2017 IEEE

- Testing and defects removal have iterative relationship since defects removal is usually followed by retest. However, the nature of the testing activities differs from those of the defects removal. Testing is conducted by test engineers while defects removal is performed by software developers as it is a regular software development activity.
- They have different ways to estimate costs. Cost of testing includes the cost of the test environment in addition to the person-days of the test engineers. The testing efforts depend on the size of the test scripts. On the other hand, the cost of defects removal is calculated in terms of developers' person-days. The amount of reworks depends on the number and complexity of defects found as a result of the testing process.
- Splitting defects removal as a separate process aims to bring the focus of the project managers and software engineers to its distinct nature as well as its impact on the project plan.
- Overlooking this process leads to wrong estimates of cost and schedule and results in considerable differences between the planned and actual values.

III RELATED RESEARCH

Zimmermann et al [9] conducted a study based on the defects database of the Eclipse open source projects. They established a mathematical model for defects predictions in terms of some complexity metrics associated with the source code. From project planning perspective, this prediction is, unfortunately, too late as coding is part of the execution phase. Project managers usually need to predict costs including the cost of defects removal as early as possible during the planning phase.

Efe and Demirors [10] conducted a case study investigating the impacts of software reworks on project estimations. The case study showed that during the testing phase 152 issues were found resulting in additional 133 unplanned persons-day for defects removal. They concluded that using project management tools and techniques in their basic forms may not meet the needs of the software projects without considering the special challenges related to these software projects. Khalid and Yeoh [11] suggested that the current models should be modified to incorporate the cost of the unplanned reworks.

Bhardwaj and Rana [12] described the relationship among software size, number of software defects, productivity, and efforts for software development projects. They applied a multiple linear regression technique on a benchmarking historical data-set. This ended up with two linear regression equations describing the web-based and non-web-based projects. They suggested that web based projects are delivered faster. Moreover, they also argued that lower productivity will lead to fewer defects while higher productivity resulted in a higher number of defects. Furthermore, they suggested that executing non-web based project with experienced team require less time for development and most of the defect can be identified during unit testing resulting in reduced rework efforts. They concluded that software size has much significant impact on total number of defect in comparison to efforts.

However, depending on historical data obtained from previous projects to formulate relationships among some variables is not consistent with the very basic attribute of the projects i.e. uniqueness. The unique nature of every software project may make the use of data provided by other projects unreliable.

IV. PROPOSED MODEL

The proposed model aims to estimate the cost of rework, at an early stage of the project, as a percentage of the cost of the software development phase. This paper focused on the software size and team experience as the main independent variables affecting the amount of reworks.

Bharwaj and Rana [12] argued that software sizing is very important in order to estimate other software project aspects like cost, schedule, and reworks.

Since, this research focuses on the early estimation of the rework cost as part of planning activities, a requirement-based sizing will be adopted through the use of function point analysis. The functional size of software is calculated in terms of the inputs, outputs, files, inquiries, and interfaces as shown in Table 1 [13], [14].

TABLE 1 IFPUG FUNCTIONAL SIZING WEIGHTS

Parameter	Simple	Average	Complex
Inputs	3	4	6
Outputs	4	5	7
Internal Files	7	10	15
Interfaces	5	7	10
Inquiries	3	4	6

Software projects would be classified according to their size. This paper adopts the classification scheme of the International Software Benchmarking Standards Group (ISBSG) as shown in Table 2 [15].

TABLE 2 ISBSG CLASSIFICATION FOR SOFTWARE SIZING

Relative software size	Software size in in function points
Small	Less than 100
Average	From 100 to 999
Large	From 1000 to 3999
Very Large	Greater than 4000

On the other hand, it is expected that, as the team experience increased the quality of the deliverables increased resulting in a reduced amount of reworks. Following the ISBSG classification, the levels of the average experience is as shown in Table 3 [15].

TABLE 3 ISBSG CLASSIFICATION FOR TEAM EXPERIENCE

Experience Level	Average experience in years
Low	Less than 1
Average	From 1 to 3
High	From 3 to 9
Very High	Greater than 9

Hence, this research suggests establishing a fuzzy model relating both the software size calculated in function points and the experience level of the development team with the expected amount of reworks following the testing process. Fig. 1 show the suggested fuzzy model.

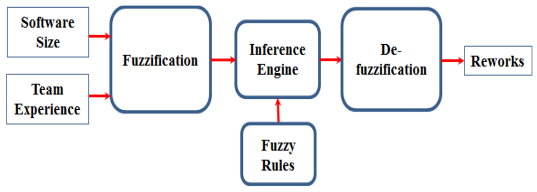

Fig. 1 Reworks Estimation Fuzzy Model

The actual cost at complete (ACAC) of the development activities including the reworks is expressed as:

ACAC= CoD + CoR

Where CoD and CoR are the cost of development and cost of reworks respectively.

Moreover, CoR = R x CoD, where R is the percentage of reworks.

Hence, ACAC = CoD + R x CoD = CoD x (1 + R)

The value of R is fuzzy. Its value depends on the results of the testing process. If the software is fully rejected then R will approach 1. Conversely, if the deliverable is fully accepted, then R is zero.

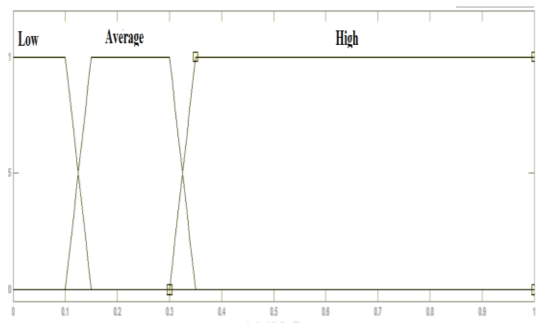

Fig. 4 A fuzzy trapezoidal membership function (TExp)

However, if the deliverable is partially accepted/rejected, then 1> R> 0. The fuzzy membership of R is shown in terms of trapezoidal shapes as in Fig. 2. A set of fuzzy numbers is assigned to the linguistic terms as shown in the Table 4. The fuzzy numbers are assigned according to the percentages of reworks suggested by the Micro Focus's [16] study.

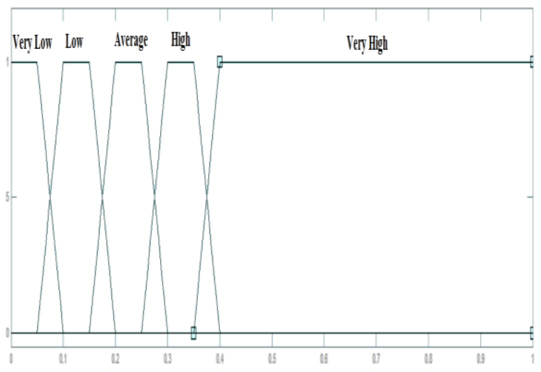

Fig. 2 A fuzzy trapezoidal membership function (Rework)

TABLE 4 FUZZY NUMBERS (R) AND THE CORRESPONDING LINGUISTIC TERMS

Fuzzy Number (R)	Linguistic Term
[0,0, 0.05, 0.10]	Very Low
[0.05,0.10, 0.15, 0.20]	Low
[0.15,0.20, 0.25, 0.30]	Average
[0.25,0.30, 0.35, 0.40]	High
[0.35,0.40, 1, 1]	Very High

On the other hand, the fuzzy inputs; software size (SSize) and team experience (TExp) are shown in terms of trapezoidal shapes as in Figures 3 and 4 respectively. A set of fuzzy numbers is assigned to the linguistic terms for each one as shown in Tables 5 and 6 respectively.

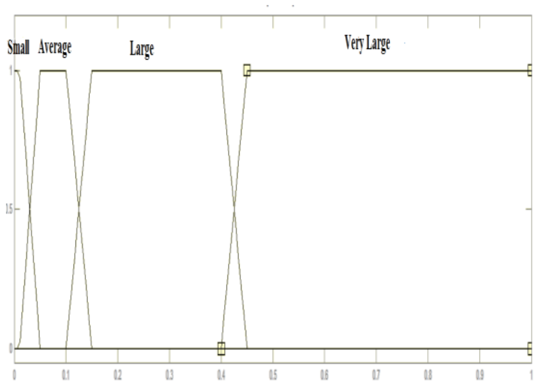

Fig. 3 A fuzzy trapezoidal membership function (SSize)

TABLE 5 FUZZY NUMBERS (SSize) AND THE CORRESPONDING LINGUISTIC TERMS

Fuzzy Number (SSize)	Linguistic Term
[0, 0, 0.01, 0.05]	Small
[0.01, 0.05, 0.1, 0.15]	Average
[0.1, 0.15, 0.4, 0.45]	Large
[0.4, 0.45, 1, 1]	Very Large

TABLE 6 FUZZY NUMBERS (TEXPE) AND THE CORRESPONDING LINGUISTIC TERMS

Fuzzy Number (TExp)	Linguistic Term
[0, 0, 0.1, 0.15]	Low
[0.10, 0.15, 0.3, 0.35]	Average
[0.3, 0.35, 0.9, 1.1]	High
[0.9, 1.1, 2, 2]	Very High

Finally, the relationship among SSize, TExp, and R is expressed as in the following fuzzy rule:
If SSize IS [Value] and TExp IS [Value] THEN R is [VALUE]
The set of the fuzzy rules are shown in Table 7.

TABLE 7 REWORK ESTIMATION FUZZY RULES

SSize	TExp	R
Small	Low	Average
Small	Average	Low
Small	High	Very Low
Small	Very High	Very Low
Medium	Low	High
Medium	Average	Average
Medium	High	Low
Medium	Very High	Very Low
Large	Low	Very High
Large	Average	High
Large	High	Average
Large	Very High	Low
Very Large	Low	Very High
Very Large	Average	High
Very Large	High	Average
Very Large	Very High	Low

V. ILLUSTRATIVE EXAMPLE

We design a simple example illustrating the basic calculations of the proposed model. It shows a software development project that consists of five tasks to develop five components and one task to integrate them. The planned durations (AD), actual durations (AD), planned cost (PC), and actual costs (AC) are all shown in Table 8. Durations and costs are in working-days and person-days respectively.

TABLE 8 PROJECT BASIC INFORMATION

Tasks	PD	AD	Developers	PC	AC
Comp1	4	5	2	8	10
Comp2	5	6	2	10	12
Comp3	11	12	4	44	48
Comp4	8	10	2	16	20
Comp5	8	10	3	24	30
System Integration	10	8	2	20	16
Defects Removal	N/A	13	2	0	26
Total				122	162

The size of software is calculated in function points as shown in Table 9.

TABLE 9 FUNCTION POINTS CALCULATIONS

Item	#	Simple	#	Average	#	Complex	Total
Inputs	12	3	7	4	5	6	94
Outputs	6	4	7	5	5	7	94
Internal Files	6	7	9	10	7	15	237
Interfaces	3	5	6	7	2	10	77
Inquiries	4	3	5	4	3	6	50
Total							552

From Table 9, the size of the software is equal to 552 function points.
It is give that the average experience of the development team is equal to 3.28 years.
Table 8 shows the following values:
The actual cost of defects removal = 26 person-days.
The estimated development cost = 122 person-days.
The actual development cost (without rework cost) = 136 person-days
The actual development cost (including the rework cost) = 162 person-days

Percentage error in estimation (without rework) = ((136 – 122)/122) x 100 = 11.5 %
Percentage error in estimation (with rework) = ((162 – 122)/122) x 100 = 32.8 %

The crisp values of SSize TExp are the inputs to the fuzzy model. Using MatLab and applying the set of functions and rules described in Tables 4, 5, 6, and 7, the rework is de-fuzzified to a crisp value of 22.5%
Then, the overall development is calculated.
Estimated cost (with reworks) = 1.225 x 122 = 149.5 person-days
Comparing this value with value obtained from the traditional method yields:
Error in the traditional technique = 32.8%
Error in the proposed model = ((162 – 149.5) / 149.5) x 100 = 8.4%

This example shows that using the proposed fuzzy model is expected to enhance the accuracy of the development cost estimation by incorporating the expected cost of rework.

VI. DISCUSSION AND CONCLUSION

Looking at software defect removals as a distinct phase is expected to bring the focus of the project managers and software engineer to its impact on the overall project cost.

The suggested model shows the impact of the software size and team experience on the cost of the expected rework, which is missing in the current project management techniques. The illustrative example shows a reduction in the estimation error from 32.8% to 8.4%.

Incorporating the estimation of the rework cost in project plans is supposed to increase the accuracy of the software cost estimation and therefore enables the project managers' focus on this overlooked process.

We are motivated to continue this research by applying the proposed model using real-live data of software projects with different scales. The direction of the future research is also to study factors, other than the software size and team experience, such as the team size, software complexity measures and software development methodology which may have impacts on the cost estimation of software reworks.

REFERENCES

[1] R. N. Charette, "Why software fails" in IEEE Spectrum, vol. 42, no. 9, pp. 42-49, Sept. 2005. doi: 10.1109/MSPEC.2005.1502528

[2] V. R. Basili, S. E. Condon, K. El Emam, R. B. Hendrick and W. Melo, "Characterizing and Modeling the Cost of Rework in a Library of Reusable Software Components," *Software Engineering, 1997., Proceedings of the 1997 (19th) International Conference on*, Boston, MA, USA, 1997, pp. 282-291.
doi: 10.1145/253228.253289

[3] U. Raja and M. J. Tretter, "Defining and Evaluating a Measure of Open Source Project Survivability," in *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 163-174, Jan.-Feb. 2012. doi: 10.1109/TSE.2011.39

[4] S. Zahra, A. Nazir, A. Khalid, A. Raana, and M.N. Majeed, "Performing Inquisitive Study of PM Traits Desirable for Project Progress," in *International Journal of Modern Education and Computer Science*, vol. 6, no. 2, p. 41. 2014

[5] Blue Print, "The rework tax: reducing software development rework by improving requirements", White Paper, Blueprint Software Systems Inc., Toronto, Canada, July 2015. Available at: http://www.blueprintsys.com/content/the-rework-tax-reducing-software-development-rework-by-improving-requirements/ (Accessed: 2/7/2016)

[6] ISO/IEC/IEE Software and Systems Engineering- Software Testing Part 1: Concepts and definitions,", ISO/IEC/IEEE 29119-1:2013(E) , vol., no., pp.1-64, Sept. 1 2013.

[7] I. Sommerville, "Software testing" in Software Engineering, 9th ed. Boston: Edison Wesley, 2010.

[8] P. Bourque and R. E. Fairley (eds.). Guide To The Software Engineering Body Of Knowledge, Version 3.0: IEEE Computer Society; www.swebok.org, 2014.

[9] T. Zimmermann, R. Premraj and A. Zeller, "Predicting Defects for Eclipse," Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on, Minneapolis, MN, 2007, pp. 9-9. doi: 10.1109/PROMISE.2007.10

[10] P. Efe and O. Demirörs, "Applying EVM in a Software Company: Benefits and Difficulties," *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, Santander, 2013, pp. 333-340. doi: 10.1109/SEAA.2013.55

[11] T. A. Khalid and E. T. Yeoh, "Controlling software cost using fuzzy Quality based EVM," *Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE), 2015 International Conference on*, Khartoum, 2015, pp. 275-280.

[12] M. Bhardwaj and A. Rana, "Estimation of Testing and Rework Efforts for Software Development Projects," in Asian Journal of Computer Science and Information Technology, vol. 5, no. 5, pp. 33-37, May 2015. doi:10.15520/ajcsit.v5i5.15.T

[13] Function Point Counting Practices Manual, Release 4.3.1, International Function Points Users Groups (IFPUG), Netherland, Jan. 2010.

[14] A. B. Nassif, L. F. Capretz and D. Ho, "Software Effort Estimation in the Early Stages of the Software Life Cycle Using a Cascade Correlation Neural Network Model," Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS International Conference on, Kyoto, 2012, pp. 589-594. doi: 10.1109/SNPD.2012.40

[15] ISBSG Data Demographics D&E Repository, Release 2016 R1, International Software Benchmarking Standards Group (ISBSG), Australia, March 2016. Available at: http://isbsg.org/wp-content/uploads/2016/04/ISBSG-DE-Demographics-2016-R1.pdf (Accessed: 01/06/2016).

[16] Micro Focus, "Successful projects start with high quality requirements", Micro Focus International plc , White Paper, Rockville, Maryland, Feb. 2016. Available at: https://www.microfocus.com/media/white-paper/WP-Successful-projects-start-with.pdf (Accessed: 10/07/2016).