Contents lists available at Science-Gate

International Journal of Advanced and Applied Sciences

Journal homepage: http://www.science-gate.com/IJAAS.html

Accurate cost estimation for software with volatile requirements

Ibrahim Hassan [1,*], Ayub Latif [1], Khalid Khan [1], Fadzil Hassan [2], Muhammad Saeed [3]

[1]Department of Computer and Information Sciences, PAF KIET University, Karachi, Pakistan
[2]Department of Computer and Information Sciences, Universiti Teknologi PETRONAS, Seri Iskandar, Malaysia
[3]Department of Computer and Information Sciences, University of Karachi, Karachi, Pakistan

A R T I C L E I N F O

Article history:
Received 11 July 2019
Received in revised form
20 December 2019
Accepted 22 December 2019

Keywords:
Volatile requirements
Software costing
Software estimation
Accuracy

A B S T R A C T

Requirements volatility in the software development lifecycle is considered to be one of the biggest risks during development. Countering this is mandatory in order to achieve success in a software project. Such requirements if they exist and not handled at an appropriate time can also result in a huge amount of error in estimation, whether it relates to cost or time. This paper first provides a clear understanding of requirements volatility along with its major contributors. The paper then mentions some widely-used techniques to achieve maximum accuracy in software costing. In section 4, the paper highlights how costing accuracy can be achieved if the software has volatile requirements by measuring an existing survey's result costing impact on the project. Finally, the paper concludes that volatile requirements cannot be eliminated but can be minimized using the approaches mentioned in the paper.

1. Introduction

The achievement of a software product depends on how accurately the requirements of the client have been understood and implemented into the software. If the requirements are not well understood, this would result in ambivalence in the SRS (Shah and Jinwala, 2015). Requirements volatility, which has been elaborated as the change in requirements (in terms of the number of additions, modifications, and deletions) during software project development. It creates supplementary tasks in architecture and code which leads to the increase in software development timeline and cost (Abd Elwahab et al., 2016) causing the system size to expand and extensive rework and effort (Peña and Valerdi, 2015).

The rest of the paper is structured like this; Section 2 gives a detailed understanding of volatile requirements. Section 3 talks about the major focusing areas that are mandatory for achieving estimation accuracy. In Section 4 we discuss the idea of getting an accurate estimate but for a software development project which comprises unstable (volatile) requirements and finally, we conclude our paper in Section 5 by also giving future directions for research. Most of the authors have not discussed how to achieve costing accuracy when the requirements are volatile in the project. So, our paper tries to highlight this major problem.

2. Volatile requirements

Requirements that arise after software application has been implemented and deployed remain valid for some period of time after which they are removed to be reactivated later or to disappear forever. We call such requirements "Volatile Requirements" and the associated functionalities as "Volatile Functionalities" (Urbieta et al., 2013).

2.1. Costing and volatile requirements

It is obvious that the main cause for huge cost and time delay is a reliance on the human factor which has many drawbacks. In practice, when an application size of 50KLOC requires a high level of rework, research shows that a simple change can require 1.5 KLOC updates and which can constitute as much as 25-person-days of average work. Another discovery is that when components are added to a system after the project has finished and deployed, the costs are higher as compared to the costs that would have been incurred if the functionalities were suggested before the software was developed (Kalbani and Nguyen, 2010). It is notable that

* Corresponding Author.
Email Address: ibrahim_hassan@live.co.uk (I. Hassan)
https://doi.org/10.21833/ijaas.2020.03.001
Corresponding author's ORCID profile:
https://orcid.org/0000-0003-4757-6800
2313-626X/© 2020 The Authors. Published by IASE.

delaying time to implement a change will decrease profits, increase costs and hold back potential business opportunities in a competitive world.

2.2. Causes of volatile requirements

The identified causes of volatile requirements include:

- Conflicts of unambiguity among requirements (Nurmuliani et al., 2004).
- User/customer knowledge evolutions (Al-Saiyd and Zriqat, 2015).
- Change of user/customer priorities (Al-Saiyd and Zriqat, 2015).
- Schedule, technical or cost-related problems (Al-Saiyd and Zriqat, 2015).
- Work environment change (Al-Saiyd and Zriqat, 2015).
- Selection of process model (Madachy and Khoshnevis, 1994).

2.3. Effects of volatile requirements

The identified effects of volatile requirement include:

- Effort and schedule overruns (Nurmuliani et al., 2004).
- A decrease in productivity (Zowghi et al., 2000) (Zowghi and Nurmuliani, 2002).
- Increase in a number of defects (Javed and Durrani, 2004).

2.4. Types of volatile requirements

Harker et al. (1993) have classified the volatile requirements into four major classes and they are depicted in Table 1.

Table 1: Classes of volatile requirements

Requirement Type	Description
Mutable requirements	This category of requirements change is a result of an operational environment changes in an organization.
Emergent requirements	It has been witnessed that as the software development process progresses, the customer gains more insight into the software and this improves his/her understanding of the software and that's why they request a change. This can also be referred to as the transformation or evolvement of the user and customer.
Consequential requirements	The requirements which come up after the system deployment because of a change in the working procedures of the organization where the software was deployed.
Compatibility requirements	Requirements popping up because of changes in the business processes of an organization. The new and altered processes now want the deployed system to change as well.

2.5. Process models and volatile requirements

Process models are the reason behind requirement volatility in projects, a little prior analysis in their selection can prompt the developers about the type of requirement change that a project will undergo. Also, process models are used to manage volatility and by carefully selecting the most suitable process model, control over the project can be maximized.

Sudhakar (2005) discussed a few process models and summarizes their pros and cons while handling project volatility in Table 2. Based on Chari and Agrawal (2018), the waterfall model is seen as a static and inflexible model when faced with changing requirements.

Table 2: Classification of process models

Process Models	Definition
Waterfall Model	In this model development is sequential, moves from one phase to another and only has one iteration/phase. The entire iteration revolves around the traditional phases which are requirements analysis, design, coding, testing, integration, and implementation.
Prototyping Model	In this, the development process starts with requirement collection, moves to prototype and ends with user evaluation
Incremental-Iterative Model	A project gets divided into mini-project; each mini-project being iteration represents a mini-waterfall model and results in an increment.
Agile Methodologies	These approaches are revolutionary and are often undertaken in places where teams collaborate rigorously to accomplish smaller tasks that are divided in a way so that they can be achieved in a small duration of time. These small iterations or cycles are also cost-effective. Many agile methodologies are proposed but Extreme Programming (XP) and Scrum are very popular in agile methodologies.

3. Achieving costing accuracy

We have several types of software costing and estimation models. Function points stay as the very classical regression approach for software costing. It is dependent on the analysis of system requirements (Sheta et al., 2015). These estimation models based on regression techniques conventionally function the estimates based on the historical data, collected on the completed projects by equating various variables and relationships therein (Fairley, 1992). The other widely used parametric models for software cost estimation comprises of COCOMO-II (Boehm et al., 2000), SLIM (Putnam and Myers, 2013), SEER-SEM (Galorath and Evans, 2006) and ESTIMACS (Rubin, 1983). These software estimation models churn the tentative cost, duration and efforts required for completing the software development. They include factors like the desired functional needs of the software and the size of the product. Along with these regression and parametric approaches, software engineering practitioners have also employed machine learning (ML) techniques for predicting software cost estimates (Latif et al., 2018).

In software development one of the most important yet difficult activities is effective software project estimation. In the absence of reliable estimates, project planning and control is impossible and generally, the whole software industry fails to either estimate projects accurately or make use of the estimates in a fruitful manner. Because of this, the industry suffers needlessly and it is very vital that these problems are addressed. We have divided this section into two more subsections that can help us in achieving accuracy in software costing.

3.1. Estimating software size

While the magnitude or size of a project is not everything, it undoubtedly holds an overwhelming influence over most aspects of development particularly that relates to the cost and resources that are required for developing software.

Without an accurate estimation of project size, planning is exceptionally difficult. We shall discuss four size estimating approaches:

- The consensus agreed by experts, a very popular technique for this approach is Wideband-Delphi.
- The proxy-based technique that utilizes the use of components for costing and estimation known as the standard components technique.
- The classical Function Points technique
- An algorithmic technique using constants and factors, very popular among which is COCOMO

3.1.1. Wideband Delphi

This technique invites experts generally from a similar domain to predict the cost and effort of a software system that is under development. The idea is that they all ultimately form a consensus after arguing with each other to reach a conclusion. The generic sequence of actions for wideband Delphi method is as under:

1. Different experts from a similar domain are identified and called.
2. They all meet to discuss the project at a specific location or meet through video conferencing.
3. Each member of the meeting gives a cost that he/she feels is appropriate.
4. All of them can see the costs that are given by each individual.
5. If all the estimates are close then this process is stopped else, they continue again from step number 2.

It is useful to show all the rounds of estimates; around means the first accepted change by everyone after their initial estimates. This helps the estimators in observing how their estimates are converging or diverging. Fig. 1 shows a case where data has been collected from 25 different groups; it is evident that the error rate is much higher from simple averaging of their initial estimates compared to the error rate from Wideband Delphi estimating.

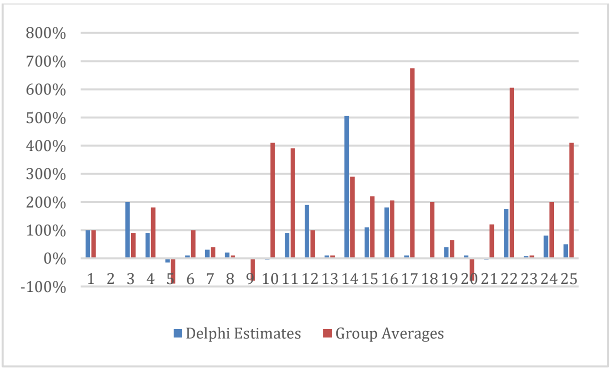

Fig. 1: Estimation accuracy of simple averaging compared to Wideband Delphi estimation. Wideband Delphi reduces estimation error in about two-thirds of cases

Fig. 1 shows that averaging the estimates of all experts can never give an accurate result which can be yielded by coming to a consensus for an estimate between all experts. This is the essence of success and accuracy in the Wideband Delphi technique.

3.1.2. Components estimating

Components estimating falls in the category of proxy-based technique. Proxy-based techniques are used when we use some artifacts related to the software that can help us in getting the expected cost and time duration of the software. In this technique, the components of the software system are used as a proxy. The number of components and their size is predicted by using the components information of some previous software system. Therefore, it is clear that this technique utilizes historical data or industrial data for calculating the size and the cost of the software. This technique works like this:

1. Collect data that is historically related to components of previous software developed by the same organization. If the data is of the developing organization, then its historical data whereas if it is of some other organization then its industrial data. It is a known fact that historical data will give a more accurate result than compared to industrial data.
2. Predict the most likely value for the total number of components for the software that is under development. We will call it (M).
3. Predict the optimistic and pessimistic values for the number of components. Obviously, the pessimistic value will have a large number of components and optimistic value will be of a lesser number of components. We denote the optimistic value by (S) and pessimistic value by (L).
4. The expected value formula can now be used to calculate the number of components required for the software to be developed.

$$E = \frac{(S+(4*M))+L}{6} \tag{1}$$

5. Once the total numbers of components are known again historical data that has the LOC/component can be used to calculate the total number of Lines of Code (LOC) for the entire software system.
6. A variant can be done to this technique and components can be classified differently with respect to their functionality. For example, the GUI component, database components, business logic components, etc. In this case, different categories of components will have a different number of LOC. This variant can be used in organizations that have a higher level of maturity.

This is a simple direct approach related to the end product. Generally, this technique only helps in identifying the total number of LOC for the software system.

3.1.3. Function points

A number of costing models have adopted the FP approach (SPQR/20, ESTIMACS). One correlation is between the size and functionality of the software which is often directly proportional. Software with larger functionality is just as big in size and function point estimation uses this relation. The weighted count of a common function of the software is used to estimate a software size. These functions are usually:

- **The number of inputs:** Any data supplied by the user is input.
- **Number of outputs:** The data which is produced by the software be it for the user of some other software is classified as an
- **The number of inquiries:** The output that is generated on the basis of some input by the user is regarded as an inquiry.
- **The number of data files:** The overall number of files which are expected to be generated by the system.
- **The number of interfaces:** The interface with which the software communicates with other software systems or even hardware devices.

The total estimate for software by function point analysis is based on the number of times a single function occurs; weight is based on its complexity in the given project. For example Table 3 shows an estimation case using the function points technique.

Table 3: An estimation case using function points technique

Function	Count	Weight	Total
Inputs	8	4	32
Outputs	12	5	60
Inquiries	4	4	16
Data Files	2	10	20
Interfaces	1	7	7
Total			135

Function point analysis has a straightforward and elaborate method such as adjusting the function point total using influence factors. This makes the estimation and correlation more useful and there are better chances of achieving more accurate estimation results.

3.1.4. COCOMO model

When a mean between variables is derived, using statistical interpretation on historical data, it is called a regression model. COCOMO exists in three forms that are hierarchal in nature. It is simple to estimate the cost of this model (Shekhar and Kumar, 2016). The first or basic COCOMO foregoes the Cost Drivers so it is valuable for quick, cursory and fast estimates of software costs but not more. The intermediate COCOMO takes these project attributes aka cost drivers into account and is far more detailed. Lastly, detailed COCOMO also factors in the effects of each project phase to estimate software costs.

3.2. Estimating software effort, schedule, and cost

We discuss estimating the effort, schedule and cost separately. The metric for the effort is persons-month, which means that an effort of 4 persons-months means that if four people work for a month on a particular project the project will be completed in a month. The schedule is always in calendar months and finally, the cost is always in terms of some currency, the actual expected amount that will spend on the project.

3.2.1. Estimating software effort

When you have a measure of the size of your product, you can calculate the effort estimate. The calculated effort will need some form of data along with the software costing model which has already been discussed in section 3.1. The evolvement of the process to acquire the cost of the software system from the size and effort requires the existence of a defined process that has major activities well defined and followed. The product is already known does not only constitute the code, but it has also been argued by researchers that coding is actually 15% to 25% of the total software development process. The other activities can be but not restricted to composing and reviewing documentation, making prototypes, structuring the deliverables, testing the code etc., takes up the bigger part of your overall effort. The ultimate goal is to develop the software which is equivalent to the estimate that was initially made. There are a couple of fundamental approaches to get an accurate effort and size:

1) the most ideal route is to utilize your organization’s own recorded information to decide how much effort has past projects of almost the same estimated size have taken. This, obviously, is based on the premise that your organization has been reporting genuine outcomes from past

ventures; it is better if you have a few tasks of comparative size as this strengthens you reliability and highlights that you require specific resources to build the products of a given size; lastly it is also important that you will pursue a similar project lifecycle, utilize a comparable development strategy, utilize similar tools, and utilize a group with similar abilities and experience for the new project.

2) The other approach can come into play when an organization has not been collecting historical data as the processes were not so strong and management never focused on the collection of data to improve the effort estimation process. Another reason for this could be that you are developing a new kind of software for the first time, the likes of which have not been made by your organization before. For such a case algorithmic approach such as Boehm et al. (2000) COCOMO model or the Putnam Methodology can be used to achieve an accurate effort estimate. These models have been derived by studying a significant number of completed projects and hence can give good results if they are used with proper analysis. But it should be clear that such a model can never achieve accuracy equal to models that based on historical data and they will always be less accurate. Still, they can give a good starting estimate when much of the things about the software which are under development are not known.

3.2.2. Estimating software schedule

From the effort estimate, if we combine the staffing profile and the work breakdown structure, we can get the overall project schedule. This contains the complete details of all the practitioners who will be working for a particular project distributed with respect to time duration. When you fill in these details you can acquire the project plan of the software which is under development. The project plan of the software will show the actual time for which the team will stay together working on a specific problem. The estimation of the schedule can also make use of historical data. Historical data with respect to productivity for individuals can further help the management is generating an accurate software schedule. In case if your organization does not follow a well-defined process for the estimate of a software schedule a rule of thumb is if you have the specified effort in staff month, you can divide that value of the effort with the team size which you wish to designate to a particular project. This will give you the completion times in months if the effort was in persons-month. For a simple example assume that if particular software has been estimated to have the effort of 60 staff-months and a team size of 10 will be used for developing the software a rough idea is that the project will require about 6 calendar months to complete.

3.2.3. Estimating software cost

We all know that the easiest way to calculate the labor cost is by multiplying the effort hours by the hourly rate of the labor. If we can add some more details and have salaries with respect to the designations and the effort in staff-hours with respect to the designations, then it can lead towards a much more accurate cost estimate of the software. The different wage structures can be with respect to the following designations like Technical, QA, Project Management, Documentation, Support, Junior Programmer, Senior Programmer etc. It also clearly shows the use of historical data.

In this section, we have just talked about the cost that is incurred on personnel which is the practitioners who are involved in developing the software. There are many other factors that need to be incorporated for calculating the correct estimate of a software project. So, all the factors that can be considered for the calculation of overall project cost include the human capital cost, hardware equipment and programming resources, travel for the meeting or testing purposes, communications costs (e.g., long-distance telephone calls, video-conferences, etc.), instructional classes, office space, etc. Precisely how you add up total project cost will rely upon how your company designates costs. Maybe the organization does not dispense a few expenses to a single project and might deal with it by increasing the value of work rates ($ every hour). The most basic cost can be estimated by multiplying the project's effort estimate (in hours) by a general work rate ($ every hour).

4. Accurate estimates for software with volatile requirements

This section is divided into three subsections; the first section discusses the strategies which are used to handle volatile requirements. The second subsection highlights the reality about costing impact for using the strategies for handling volatile requirements and the third subsection goes in detail of the software costing models that accommodates volatile requirements.

4.1. Existing strategies to handle volatile requirements

Thakurta and Ahlemann (2010), interviewed 11 software project managers in Germany and the results of their survey findings are given in Table 3. The conducted interviews helped in identifying nine approaches to handle requirement volatility. The queries were based on Morehouse's (1994) guidelines and included questions regarding the managers' demographics, SDLC methods, success rates, requirement volatility awareness, organizational setting, patterns of requirement change, background information on the project and the impact of requirement volatility on the project.

Table 4 shows the nine approaches which were used by the authors and they added four more approaches and suggested a total of 13 approaches. They have also identified the frequency of a particular approach. The frequency shows what percentage of a particular approach is used by the management to handle volatile requirements. As our paper gives a strategy to accurately estimate software with volatile requirements, we have added another column to show the costing impact of each approach. The costing impact shows that when we follow a particular approach for handling volatile requirements whether it will increase the overall cost of the software or will decrease it.

One important factor here to observe is that the costing impact listed in Table 4 is the illusion in the management's mind for incorporating a particular approach during software development. The actual impact can be different than as to what the management initially feels. In fact, we believe that the costing factor given in terms of implementing a particular approach is a myth that is there in management's mind and the reality can be different from the stated cost impact.

Table 4: Strategies of Thakurta and Ahlemann (2010) to handle volatile requirements for software

Number	List of Approaches	Freq (%)	Costing Impact
1	Involving the business side in the project	11.3	Increase
2	Negotiating the project scope	9.8	Decrease
3	Rescheduling the project deadlines	9.0	Increase
4	Active involvement in requirements management activities	8.3	Increase
5	Proper documentation of procedures, activities, etc.	6.8	Increase
6	Adjustment in the human resource of a project	6.4	Decrease
7	Taking and Using Expert's knowledge	5.6	Increase
8	Making the project communications effective	5.3	Decrease
9	Reducing the complexity factors of project	4.1	Decrease
10	Readjustment of the overall project effort	8.6	Decrease
11	Including an additional variable in cost for additional requirements	4.1	Increase
12	Making a robust software architecture to withstand change	3.4	Increase
13	Running training programs for the workforce	2.3	Increase

As per the data in Table 4 an interesting phenomenon here is to notice that even a costing impact that management feels will increase the overall cost of the software will end up is decreasing the overall cost. Before going into the elaboration of a specific strategy from Table 4 as an example a brief reality about software reviews needs to be investigated. It is a known fact that reviews help in uncovering 60% of the overall software error and static testing is considered as a best practice in software development. It is again proved that investing a single dollar on review will help the organization save many dollars later. The same goes for the strategies to handle volatile requirements. If we look at the strategy number eleven from Table 4, the management feels that a robust architecture will increase the overall cost of the software. The reality is if a robust architecture is in place at the right time this can help in saving the cost later that is a huge amount of cost can be saved in programming and the testing phase.

On the other hand, management feels that negotiating the project schedule or adjustment in the human resource of a project will lessen down the overall development cost. Well, this varies from case to case, if while negotiating the project's scope management is able to get the scope lesser than to actually what it was initially then it will surely decrease the cost, otherwise if the customer is stringent about the project scope, it will end up in increasing the overall cost of the software. The other strategy here is adjustment in human resource of a project; management feels that they will be able to reduce the human resources allocated for a project and this will decrease the overall cost or will change the star performing people from this project that is under development and use the star performers in some other generally newer projects.

Both cases that are either the lessening of human resources for a particular project or changing the practitioners of an ongoing project can lead to an increase in the overall cost.

4.2. The reality about costing impact on using strategies to control volatile requirements

As discussed in the previous subsection the reality of the costing impact of using a particular strategy to control volatile requirements can be different as to what the management perceives about a specific strategy. The management feels that some of the strategies can lead to an increase in the overall cost of the software which is undergoing development. We strongly believe and it has been witnessed in the case of software reviews also that if the strategies to handle the volatile requirements are performed at the right time, this will ultimately help in reducing the overall cost and effort of the software. However, the right time to use a particular strategy can vary from project to project and also on other factors whether they are internal or external to the software development activity.

We form the opinion that if estimating effort, schedule and cost procedures are followed in a way as discussed in section 3 and handling volatile requirements strategy of section 4 are used intelligently then organizations can control the impact of volatile requirements which will lead them to achieve accuracy in their estimates.

4.3. Existing software cost models that accommodate volatile requirements

Organizations which are at lesser maturity level and where proper systems are not implemented generally will not use the discussed approaches of section 4.1. They can still use the Post Architecture Model of COCOMO II. This model estimates in advance the amount of code you might need to discard and adjusts your product size due to requirement change by using a REVL factor which is (Requirements Evolution and Volatility).

As we already know that neural net algorithms constantly change, so do fault diagnostic requirements and we can reduce this risk by adding 20% to the project size based on REVL while the development code is being written. This helps to cope up with program growth and volatility. The impact of the increased size can be estimated in the COCOMO II model to compute project duration and delta effort. When the effort is adjusted at the appropriate time, the cost increase would have been included in the initial calculated cost. During the project development if volatile requirements creep in, so it will not affect the management as they must have calculated the overall cost of the software including the volatile factor.

5. Conclusion

Change of requirements can occur at any phase of development. Requirements Volatility is the measure of the rate of change of requirements not only in the development phase but in the operational phase too. It has a great influence on the software project schedule, cost and effort (Al-Saiyd, 2016). This cannot be completely eliminated but minimized. In this paper, we studied the impact of volatile requirements on a software project. We investigated their causes and suggested measures for handling them.

As the focus of the paper was to accurately estimate software with volatile requirements, we presented approaches to correctly estimate software which has a volatile requirement. The paper also reviewed the important process models and also the software estimation techniques and models. Existing work exists to handle volatile requirements and we extending that by including the cost impact to the suggested strategies. The cost impact of using a strategy was given from the viewpoint of the management of the software developing organization.

We intend to extend this study and would investigate case studies where the strategies for handling volatile requirements are used. It will help us in noticing the impact of those strategies on the overall cost of the software. We would also like to investigate the factors that help in identifying the right time in an ongoing software process to use different strategies for handling volatile requirements. Volatile requirements are a reality and they exist in almost all software projects. Lastly, the classification of volatile requirements with respect to different types of software can be another future direction for research and it can help the software community.

Compliance with ethical standards

Conflict of interest

The authors declare that they have no conflict of interest.

References

Abd Elwahab K, Abd El Latif M, and Kholeif S (2016). Identify and manage the software requirements volatility proposed framework and casestudy. International Journal of Advanced Computer Science and Applications, 7(5): 64-71. https://doi.org/10.14569/IJACSA.2016.070510

Al-Saiyd NA (2016). A multistep approach for managing the risks of software requirements volatility. Computer and Information Science, 9(1): 101-112. https://doi.org/10.5539/cis.v9n1p101

Al-Saiyd NA and Zriqat IA (2015). Analyzing the impact of requirement changing on software design. European Journal of Scientific Research, 136(1): 62-73.

Boehm BW, Madachy R, and Steece B (2000). Software cost estimation with Cocomo II. Prentice-Hall, New York, USA.

Chari K and Agrawal M (2018). Impact of incorrect and new requirements on waterfall software project outcomes. Empirical Software Engineering, 23(1): 165-185. https://doi.org/10.1007/s10664-017-9506-4

Fairley RE (1992). Recent advances in software estimation techniques. In the 14th International Conference on Software Engineering, Association for Computing Machinery, New York, USA: 382-391. https://doi.org/10.1145/143062.143155

Galorath DD and Evans MW (2006). Software sizing, estimation, and risk management: When performance is measured performance improves. Auerbach Publications, Boca Raton, USA. https://doi.org/10.1201/9781420013122

Harker SD, Eason KD, and Dobson JE (1993). The change and evolution of requirements as a challenge to the practice of software engineering. In the IEEE International Symposium on Requirements Engineering, IEEE, San Diego, USA: 266-272. https://doi.org/10.1109/ISRE.1993.324847

Javed T and Durrani QS (2004). A study to investigate the impact of requirements instability on software defects. ACM SIGSOFT Software Engineering Notes, 29(3): 1-7. https://doi.org/10.1145/986710.986727

kalbani AA and Nguyen K (2010). Designing flexible business information system for modern-day business requirement changes. In the 2nd International Conference on Software Technology and Engineering, IEEE, San Juan, USA, 2: V2-112. https://doi.org/10.1109/ICSTE.2010.5608774

Latif MA, Khan MY, and Bashir K (2018). Practices for achieving accuracy in software costing and estimation. In the 2nd International Conference on Computing and Information Sciences, At Karachi, Pakistan.

Madachy RJ and Khoshnevis B (1994). A software project dynamics model for process cost, schedule and risk assessment. Ph.D. Dissertation, University of Southern California, Los Angeles, USA.

Morehouse RE (1994). Beginning qualitative research: A philosophic and practical guide. Psychology Press, UK.

Nurmuliani N, Zowghi D, and Powell S (2004). Analysis of requirements volatility during software development life cycle. In the Australian Software Engineering Conference, IEEE, Melbourne, Australia: 28-37. https://doi.org/10.1109/ASWEC.2004.1290455

Peña M and Valerdi R (2015). Characterizing the impact of requirements volatility on systems engineering effort. Systems Engineering, 18(1): 59-70. https://doi.org/10.1111/sys.21288

Putnam LH and Myers W (2013). Five core metrics: The intelligence behind successful software management. Pearson Education, London, UK.

Rubin HA (1983). Macro estimation of software development parameters: The ESTIMACS system. In the SOFTFAIR Conference on Software Development Tools, Techniques and Alternatives, IEEE, Arlington, USA: 109-118.

Shah US and Jinwala DC (2015). Resolving ambiguities in natural language software requirements: A comprehensive survey. ACM SIGSOFT Software Engineering Notes, 40(5): 1-7. https://doi.org/10.1145/2815021.2815032

Shekhar S and Kumar U (2016). Review of various software cost estimation techniques. International Journal of Computer Applications, 141(11): 31-34. https://doi.org/10.5120/ijca2016909867

Sheta A, Rine D, and Kassaymeh S (2015). Software effort and function points estimation models based radial basis function and feedforward artificial neural networks. International Journal of Next-Generation Computing, 6(3): 192-205.

Sudhakar M (2005). Managing the impact of requirements volatility. M.Sc. Thesis, Umeå University, Umeå, Sweden.

Thakurta R and Ahlemann F (2010). Understanding requirements volatility in software projects-an empirical investigation of volatility awareness, management approaches and their applicability. In the 43rd Hawaii International Conference on System Sciences, IEEE, Honolulu, USA: 1-10. https://doi.org/10.1109/HICSS.2010.420

Urbieta M, Rossi G, Distante D, and Schwinger W (2013). Managing volatile requirements in web applications. In the 15th IEEE International Symposium on Web Systems Evolution, IEEE, Eindhoven, Netherlands: 77-82. https://doi.org/10.1109/WSE.2013.6642420

Zowghi D and Nurmuliani N (2002). A study of the impact of requirements volatility on software project performance. In the 9th Asia-Pacific Software Engineering Conference, IEEE, Gold Coast, Australia: 3-11. https://doi.org/10.1109/APSEC.2002.1182970

Zowghi D, Offen R, and Nurmuliani N (2000). Impact of requirements volatility on the software development lifecycle. In the 16th IFIP World Computer Conference: Software Theory and Practice, Publishing House of Electronics Industry, Beijing, China: 19-27.