Practices for Achieving Accuracy in Software Costing and Estimation

Mohammad Ayub Latif[1] Muhammad Yaseen Khan[2] Kashif Bashir[3]

Abstract

Predictability is the foremost choice of all stakeholders in software development; it could be custom software or a general solution. Many software costing models have been proposed and used in the software industry over the last 40 years. In this paper, we go down into the details of recent approaches in software estimation and besides we propose the mandatory steps which can lead towards accuracy in software cost estimation. The more mature an organization is in costing and estimation, the more accurate results it is expected to achieve. We believe that using the steps defined in this paper will lead to more accurate results in costing and estimation. Unlike the Capability Maturity Model (CMM), we don't propose any specific levels and designate key process areas to it; but specific list of procedures towards an accurate costing and estimation of software is clearly identified in this paper.

Keywords: software development, software costing, costs, estimation, software estimation techniques, project management

1 Introduction

Not everything in the world is free, people have to pay for their work to make several contributions in return, and it continues to function in a cyclic fashion. Estimating the price and cost of work involves a lot of things, especially for the intangible assets like softwares which pertains to change with the pacing technological changes and human skill-sets. Achieving accuracy in estimation is one of them. If we look at research in software costing and estimation, we will find that significant cost failures are reported in many studies: The Standish Report stands as a leading example [1], [2]. Boehm et al. in their classic paper [3], argue that a careful evaluation of estimation results generated through several techniques is most likely to produce a realistic estimate. It comes as a valid question that the mentions in the Standish Report are still justifiable or not? Many researchers have challenged the validity of the Standish Report [4], [5] and have provided evidences that much-exaggerated figures were promulgated in the report [6]. We have several software costing and estimating models. Function points [8] stay as the very classical regression approach for software costing. These estimating models based on regression techniques conventionally function the estimates based on the historical data, collected on the completed projects through equating various variables and relationships therein [9]. The other widely used parametric models for software cost estimation comprise COCOMO [10], COCOMO-II [11], SLIM [12], SEER-SEM [13], and ESTIMACS [14]. These software estimation models churn the tentative cost, duration and efforts required for completing the software development. They include the factors like the desired functional needs of the software and size of the product. Along with these regression and parametric approaches, software

[1] *Karachi Institute of Economics and Technology, Karachi | malatif@pafkiet.edu.pk*
[2] *Mohammad Ali Jinnah University, Karachi | yaseen.khan@jinnah.edu.pk*
[3] *Karachi Institute of Economics and Technology, Karachi | kashif@pafkiet.edu.pk*

engineering practitioners have also employed machine learning (ML) techniques for predicting the software cost estimates. Since we are targeting a continuous number as an output, therefore, in ML the software cost estimation will be considered as a regression problem [15], [16]. In this regard, a leading study was made by Krishnamoorthy et al. in [17] in which Neural Networks (NN) and Regression tree based ML approaches were employed. An adaptive learning model for estimating software costs is presented in [18], while other NN based ML models are employed by many researchers for software cost estimation [19]–[21].

Researchers have also dwelled on the identification of causes that lead to inaccurate estimation for a software [22]. Many have invested their energy to understand the exact cause. Todd, in his famous list has concluded that "frequent requests for changes by users" and "lack of understanding of their own requirements'' constitutes the top two causes of inaccurate estimates[23]. These surely cannot be considered as the only reasons; there are numerous other reasons as well in which errors come from the estimation process itself and also from many other areas. Right time of committing the cost of software is critically important to address. As an experience, it comes as a practical agreement that the variability factor decreases, and the cone of uncertainty shrinks as we continue with the software development [7]. Hence, keeping the cone of uncertainty in context, we can suitably find the ideal time for cost commitment.

Instead of proposing any new model for software cost estimation, we believe that a step by step procedure to attain estimation accuracy is not clearly proposed by the research community. Therefore, in this paper, we have proposed the necessary steps in a sequential order which can lead towards best estimation results. We have used the specified procedure in small projects, made professionally for the industry or for the academics.

The rest of the paper is structured as follows: In the next section, we discuss several studies pertaining to software costing and estimation and provide strategies for costing without the knowledge of which one can never be a good estimator. Section III provides the details of how our proposed methodology gives a step by step mechanism for accurate estimation of software. In section IV, we describe how our methodology is applied in the estimation of software for academics and industry and in section V, we discuss the result of proposed methodology followed by conclusion and direction of future research in section VI.

2 Strategies for Software Costing

A Time of cost commitment

Understanding of "the cone of uncertainty" [10], as it was coined by McConnell [24], guides us that we should not commit the cost and efforts required to complete the project before creating the prototypes of the software system. For information system, this means, the complete Graphical User Interface (GUI) of the software system should be prepared.

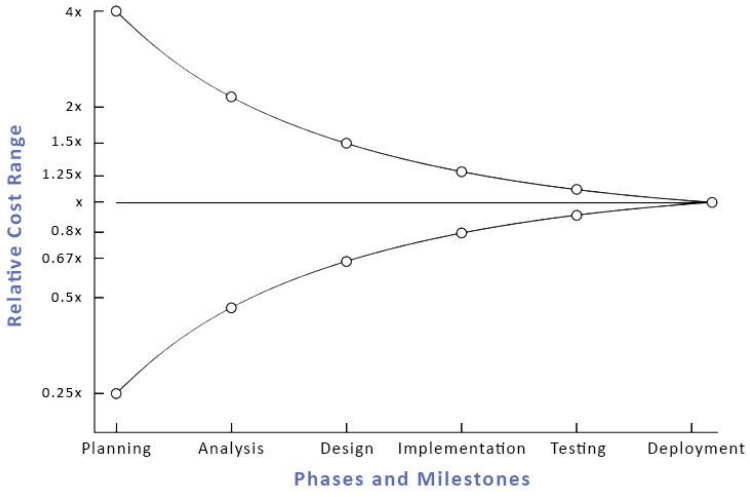

Figure 1: The cone of uncertainty with variability factor.

If we look at the figure 1, it is quite obvious that until the phase of designing is started (which is generally the time when requirements specifications are written) the cone is narrowed to 1.5 at the higher side and 0.6 at the lower side. This means the variability factor at this point has reduced from 16x to 2.5x. It could be probably a good time to commit the software cost estimate and effort.

B Understanding of Parkinson's Law

Parkinson's Law [25] stipulates that the assigned work will automatically acquire the available time, this implies that available time automatically finishes when a particular activity is finished, even if the work is much less to be done in allocated time. Chris, in [26], has explained the phenomenon of Parkinson's Law in a great detail. The irony of the fact is that even when the assigned task requires lesser time than the allocated time, the available time finishes. On the other hand, when the activity does not complete in available time, the time period overruns for that activity. Hence, stringent project control is required by the project manager and/or team lead to make sure that Parkinson's Law is not applied in the software project.

C Subjectivity

A good estimator needs to know that subjectivity engenders an element of doubt. Considering this, McConnell in his classical book [7] has identified and proved that the more controlling knobs we have in an estimation model, the more variance we can get in our estimates. This is the reason why the Intermediate and COCOMO II [11] models are criticized by researchers and experts because they have 15 and 17 effort adjustment factors respectively, and these adjustment factors can create subjectivity in estimation cases.

D Biasedness

An estimate can be said biased when it is the deliberate attempt to fudge estimation in one direction or the other, when pressure is applied to the project. It needs to be avoided at all costs.

E The Law of Large Numbers

The law guides us that the accuracies of empirical calculations can be achieved with number of trials [27]. Historically, it was named "Golden Theorem" or "Bernoulli's Theorem" [28], later in 1837, it was further explained by Simeon´ Poisson under the name "The Law of Large Numbers (LNN)" [29]. In the context of software cost estimates, whenever, we are to give an estimate which is based on multiple tasks, we should calculate the time frame and effort of the individual (near atomic) tasks, and then calculate their total. The best advantage of this repeating activity is that our estimates will cancel each other on both positive and negative sides, and therefore, we will encounter a much diminished error that could have happened if the total estimate was calculated in a consolidated manner.

F The Need of Data

Data for the purpose of estimation can be classified into three main categories: i) industrial data, ii) historical data and iii) project data. In many research papers, the different type of data is used. We know that the most accurate among the three is project data as it is the data of the ongoing project itself. The only disadvantage it poses is that for the use of project data our software development model has to be an iterative model.

G Avoiding arguments and let data do its job

We unknowingly enter into subjectivity when we evaluate people through assigning values of adjustment factors used in different costing models. COCOMO-II [11] provides us with people classification in its 15 adjustment factors where we need to select analyst capability, programmer capability etc. as low, marginal, high etc. Evaluating a person through these value can give rise to politically charged statements like, "my programmers are below average" etc. [7] A better way to say the same thing in front of management is, "We averages 300 LOC productivity in the previous project we have used the same average for the new project" [7].

H Accuracy, not Precision

In software costing and estimation, our goal is to be accurate, not precise [7]. For example, rather than saying a particular project will be completed in 27.5 staff months it is better to say that the project will take between 25 - 29 staff months. This will give us more chances of being accurate, so the trick of the trade is to commit your estimates in ranges, this way we will achieve accuracy.

I Flat or Dynamic Staffing Model

Broadly the costing models can be classified as the flat staffing model [3] or dynamic model [7]. A flat model is the one which assumes that same number of people will be working for the software from its early requirements phase to the implementation phase. A dynamic model assumes that team size can be changed with respect to phases of the software during development. For example, the team size can be of 2 people in the requirement phase and 10 people in the development phase.

3 Proposed Methodology for Software Estimation

In this section, we propose 7 steps which should be followed in a proper sequence which will surely lead towards the most accurate estimate.

1) **Not too early, even not too late to commit.** As an estimator is it very important that we should commit our estimate to the customer when the Requirements Specifications phase has finished. If we look at this with respect to the cone of uncertainty then by this time the 16x factor has reduced to 2.25 [23]. How can an estimator make sure that he/she has reached the desired point? The signed Requirements Specifications Document (SRS) will be a good proof of this that we have reached this level. For small projects, this point can even be until the completion of GUI where the factor can be between 1.6x to 1.8x.

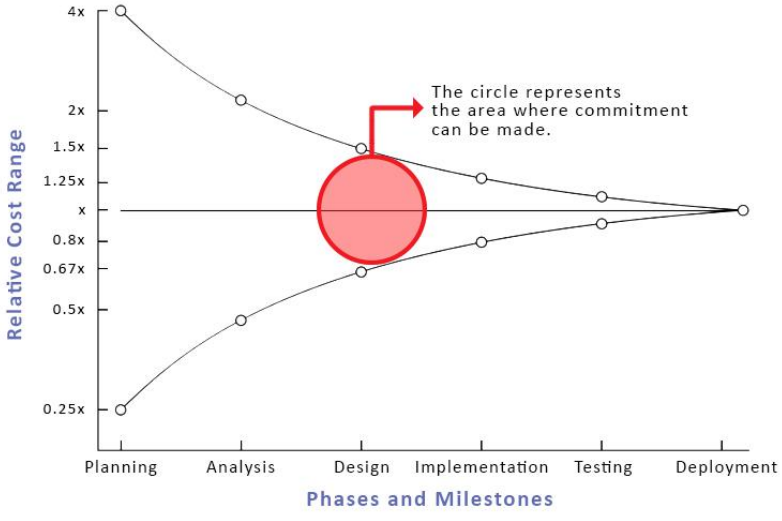

Figure 2: The circle in the cone suggesting the best time to make commitment.

In the figure 2, the circle shows the area where commitment can be made as the cone of uncertainty has narrowed to a point where variability is in a manageable state and going on either side will not create a huge problem. On the left edge of the circle the variability factor is 0.55 and 1.5 on the higher and the lower sides that makes it 3.1x and on the righter edge of the circle the variability factor is 1.2 and 0:8 that makes it 1.5x.

2) **Dynamic cost estimation models.** These are generally suggested as it helps in many cases. Initially when the customers are quickly looking for the estimate to give approval whether they would approve the software to be made or not, adding more people to the project investigating will help in an effective manner. Once the cone of uncertainty has narrowed to a point when the commitment can be made, the commitment can be made and immediately after the customer approval, the team size can be reduced during the post requirements and analysis phases. Again the

team size can be increased on "as needed" basis. This liberty is possible only when we use dynamic models for estimation.

3) **Avoid cloud of uncertainty.** We need to make sure that due to weak controls, cloud of uncertainty is not formed in our project. Because, with its formation we will head towards a disaster and our project can reach the referent point. The figure 3 shows the effect of cloud in the cone of uncertainty. The variability factor remains until the end of the project.

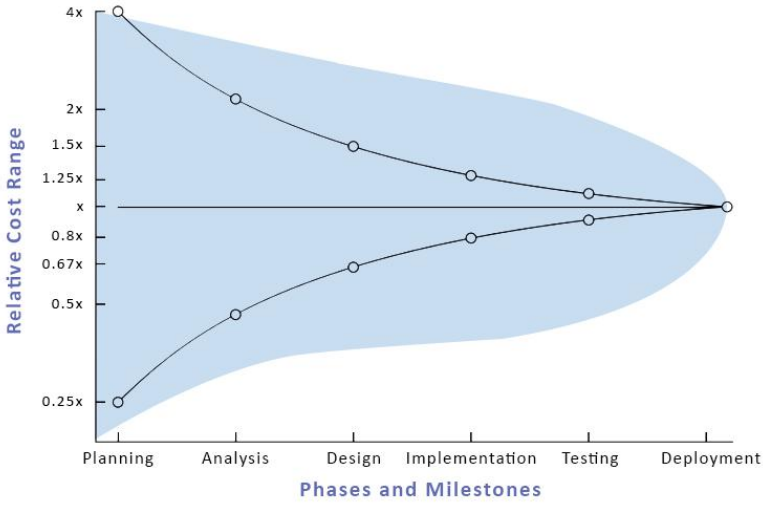

Figure 3: The cloud of uncertainty which appears because of weak project controls.

4) **Tasks up-to 2 weeks.** The overall task of the entire project must be broken down into small task list where each task must not take more than 2 weeks. We have already discussed in our strategy section II-E that how breaking the estimate can help us in taking advantage of the law of large numbers.

5) **Iterative development model.** Select an iterative model for the development of the software so immediately after the first iteration; we can use project data for later estimation instead of historical data. We have already seen in section II-F that project data is always the most accurate to use.

6) **Convergence in estimating techniques.** Always use multiple techniques for your estimation case and look for conver-gence among them. In next section, we have used multiple techniques in our estimation case and have shown how we have used convergence among two techniques to rule out the third technique that showed divergence from the other two.

7) **Standard deviation.** When there is convergence in multiple techniques we have a consensus to believe that we are near to a good estimate. At this time instead of committing the estimate as it is; a better way is to use standard deviation and, propose the estimate with specific confidence.

4 Case Studies

We demonstrate the usage of principles of our proposed methodology in student registration software for a small sized institute. At first, we didn't apply the principles discussed in this paper and just did a general estimate using the proxy-based technique of standard components. Standard Components is discussed in depth in [7] and the authors also discuss components as a modularity technique for Agile developments [30]. Later, we decided to apply the principles discussed in this paper to the estimation case of students' registration system and came up with the following results. In this section, we discuss the complete estimation case, a brief discussion of the problem statement is also given in this paper, but a detailed discussion is avoided as it is not necessary for the domain of this paper.

A Problem Statement of the Student Registration System

The 'student and course registration problem' is used to calculate the effort and cost required for the software. The process of assigning teachers to courses and the registration of students is a frustrating and time-consuming job for any institute.

In the new system, after the management and teachers of the institute have decided which course in which timings they are going to teach for the session, the administration office enters the information into the computer system. The registered students when they log in the system will see a report of course timings and teachers' information. They will register themselves for specific timings with a specific teacher. None of the course will have more than 12 students. Once 12 students have been completed for a particular course, the system will not show that timings and teacher option to the students for registration.

When the student has registered himself/herself, a message is prompted on their screen of successful registration and they can even see the monthly schedule of their classes. The system gives a tentative date for the start of the course and if a course is closed for registration, it gives out a specific date also. One day before the class, the system also generates a message to the student on his/her cell phone (short texts / SMS) so that they know that their class is starting from tomorrow.

Teachers can also log into the system and generate the list of students which they will be teaching in a particular time slot. Obviously, the teachers will generate the list when the registration of a particular slot is closed. The entire system is web-based and is developed using ASP.Net.

B Initial requirements and analysis of the problem domain

The following actors and use cases were identified for the problem statement:

1) Actors
   a) Teacher
   b) Academic/Admin Staff
   c) Student
2) Main use cases of the system with actors are shown in the table 1.

Table 1: The main use cases and actors of the problem statement

Use Case	Actor
Entering course and timings	Admin/Academic
Selecting and registering course	Student
Generating course schedule	Student
Generating Student list for a course	Teacher

C Costing Metric

Let C be the set of all type of components used in Student Registration System C = {c1, c2, … cn}, and L be the set of LOC required for completing individual components respective, such that, L = { ∫c1, ∫c2, … ∫cn} where ∫c1 indicates the LOC required for component c1. Hence by aggregating the productions of C and L accordingly, we can have the total LOC required for completing the whole Student Registration System. The function f(C; L) models the metric below under the condition that provides ||C|| = ||L||.

$$f(C + L) = \sum_{i=1}^{\|c\|} (c_i \,.\, l_{ci}) \quad (1)$$

Case 1: Without applying the principles to achieve estimation accuracy

The broader category of standard components is proxy based techniques and it is one of the types of the proxy-based technique. The core idea is that if you develop many programs that are architecturally similar to each other, you can use the standard components approach to estimate size.

For estimation purpose, we have chart of historical data as depicted in table II to apply estimation using the standard components technique.

Table 2: Components for Case Study I

Standard Component	LOC/Component
Dynamic Web Pages	417
Static Web Pages	78
Database Tables	1227
Reports	388

Table 3: Estimation for Case Study-I w.r.t Number of Components and LOC

Type of Component	Number of components in Case Study I	LOC/component	Total LOC for specific component
Dynamic Web Pages	14	417	417 x 14 = 5, 838
Static Web Pages	7	78	78 x 7 = 546
Databases	12	1,227	1, 227 x 12 = 14, 724
Reports	11	388	388 x 11 = 4, 268
Business Rules	3	4,237	4, 237 x 3 = 12, 711
Total LOC for the Students Registration System			38, 087

Table 4: Estimation for Case Study-II w.r.t Number of Components and LOC

Type of Component	Number of components in Case Study I	LOC/component	Total LOC for specific component
Dynamic Web Pages	18	367	367 x 18 = 6, 606
Static Web Pages	9	110	110 x 9 = 990
Databases	10	1, 037	1, 037 x 10 = 10, 370
Reports	14	270	270 x 14 = 3, 780
Business Rules	2	3, 100	3, 100 x 2 = 6, 200
Total LOC for the Students Registration System			27,946

As such systems are generally web-based information systems so this much of historical data with the specified LOC is sufficient enough for an early estimate of effort. For the three stated actors, we identified the number of components for the five stated types for each of the actor. Then we sum up the total number of components of different types to reach to an early effort estimate based on the Lines of Code (LOC). With the simple analysis, we made out that this number of components of each type will be used in our Students Registration System.

Table III shows that the basic standard components technique and applying the historical data, the first estimate using equation 1 was around 38, 000 LOC or ≈38 KLOC.

Case 2: Applying the estimation principles to discussed in this paper

For the simple estimation case, we tried to include maximum steps from our proposed methodology and came up with a visible accuracy of x%. The implementation of our step by step estimation accuracy, in this case of student registration system, is stated as below:

We made use of our step number 1, by delaying the estimate till we have not generated the GUI of the software. Step 2 was also followed as we deployed maximum people at the start of the project to get to the User Interface of the software and after getting the estimate when we were sure of an accurate estimate we lessened the number of members of the team for later phases. This was possible as we used dynamic costing models. Step 3 was also used as we had

made projects before and we knew that which wrong practices can lead us towards a cloud of uncertainty. Talking about our suggested step 4, we believe that in the first case also the project was nicely broken down in smaller tasks so we did not bother breaking it further as we believed that law of large numbers advantage was already applied in the first estimation case. Secondly, with so much smaller tasks there were lesser chances that the project estimate can get away with hidden work.

We already follow the rational unified approach for creating software in our software development unit that is why we followed step 5 also as Rational Unified Process (RUP) [31] is an iterative model and as we suggested earlier and we immediately started using project data instead of historical data as soon as we started with the next iteration.

When we applied the principles we were able to commit deadline for the completion of the project after 15 days of the inception phase, in case 1 the commitment was made after 5 days of the inception phase.

Please note that the historical data figures as used in case number 1 were changed as we used project data for this estimation and we derived to project data after performing the first iteration. The project data value chart used for the second estimation case is given in table V.

Table 5: Components for Case Study II

Standard Component	LOC/Component
Dynamic Web Pages	367
Static Web Pages	110
Database Tables	1, 037
Reports	270
Business Rules	3, 100

The summary of our estimation with equation 1 is shown in table IV. Through the basic standard components technique and using the steps for accurate estimates as identified in this paper and using project data, the second estimate was around 2, 8000 LOC or ≈28 KLOC.

5 Results

The project was completed in 2 months using a team of 4 dedicated people in the team, two working as analyst and designers and other two working as programmers; considering a productivity of 3,000 LOC per person per month, it shows clearly that project was completed in 24, 000 LOC or ≈24 KLOC. Considering the result it is clearly evident that following the estimation steps as specified in this paper we can reach towards more accurate results in software costing and estimation.

6 Conclusion

In this paper, we studied about the strategies used in software costing and estimation and then we proposed few steps which if implemented during the estimation process can lead toward more accurate estimate of the software under development. Our results show that using five out of the seven specified steps we were able to commit an estimate which was only deviating 16.66% from the actual outcome than compared to the other estimate which was 58.33% away from the actual outcome. It was completely unintentional that both the estimation cases were on the overestimate side.

There are few more observations which pave the way for future research; the first is that the more accurate estimate was given after 15 days of the inception phase, although the first estimate which was low in accuracy was given after 5 days of the inception phase. It needs to be investigated what out of the two is a bigger benefit, early commitment or accuracy. For this estimation case, we confined to the simple information system, the likes of which were previously made by the development organization. The future investigation can be done on different types of software and of unfamiliar areas where expertise lack.

References

[1] G. Standish, "The chaos report," The Standish Group, 1994.

[2] M. Jørgensen and K. Moløkken-Østvold, "How large are software cost overruns? a review of the 1994 chaos report," Information and Software Technology, vol. 48, no. 4, pp. 297–301, 2006.

[3] B. Boehm, C. Abts, and S. Chulani, "Software development cost estimation approachesa survey," Annals of software engineering, vol. 10, no. 1-4, pp. 177–205, 2000.

[4] R. L. Glass, "It failure rates-70% or 10-15%?" IEEE Software, vol. 22, no. 3, 2005.

[5] R. L. Glass, "The standish report: does it really describe a software crisis?" Communications of the ACM, vol. 49, no. 8, pp. 15–16, 2006.

[6] K. Moløkken and M. Jørgensen, "A review of software surveys on software effort estimation," in Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on. IEEE, 2003, pp. 223–230.

[7] S. McConnell, Software estimation: demystifying the black art. Microsoft press, 2006.

[8] J. E. Matson, B. E. Barrett, and J. M. Mellichamp, "Software development cost estimation using function points," IEEE Transactions on Software Engineering, vol. 20, no. 4, pp. 275–287, 1994.

[9] R. E. Fairley, "Recent advances in software estimation techniques," in Proceedings of the 14th international conference on Software engineering. ACM, 1992, pp. 382–391.

[10] B. W. Boehm et al., Software engineering economics. Prentice-hall Englewood Cliffs (NJ), 1981, vol. 197.

[11] B. W. Boehm, R. Madachy, B. Steece et al., Software cost estimation with Cocomo II with Cdrom. Prentice Hall PTR, 2000.

[12] L. H. Putnam and W. Myers, Five core metrics: the intelligence behind successful software management. Pearson Education, 2013.

[13] D. D. Galorath and M. W. Evans, Software sizing, estimation, and risk management: when performance is measured performance improves. CRC Press, 2006.

[14] H. A. Rubin, "Macroestimation of software development parameters: The estimacs system," in SOFTFAIR Conference on Software Development Tools, Techniques and Alternatives, 1983, pp. 109–118.

[15] N. M. Nasrabadi, "Pattern recognition and machine learning," Journal of electronic imaging, vol. 16, no. 4, p. 049901, 2007.

[16] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Machine learning: An artificial intelligence approach. Springer Science & Business Media, 2013.

[17] K. Srinivasan and D. Fisher, "Machine learning approaches to estimating software development effort," IEEE Transactions on Software Engineering, vol. 21, no. 2, pp. 126–137, Feb 1995.

[18] A. Kaushik, A. Soni, and R. Soni, "An adaptive learning approach to software cost estimation," in Computing and Communication Systems (NCCCS), 2012 National Conference on. IEEE, 2012, pp. 1–6.

[19] N. Tadayon, "Neural network approach for software cost estimation," in Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on, vol. 2. IEEE, 2005, pp. 815–818.

[20] D. E. Neumann, "An enhanced neural network technique for software risk analysis," IEEE Transactions on Software Engineering, vol. 28, no. 9, pp. 904–912, 2002.

[21] G. E. Wittig and G. Finnic, "Using artificial neural networks and function points to estimate 4gl software development effort," Australasian Journal of Information Systems, vol. 1, no. 2, 1994.

[22] A. L. Lederer and J. Prasad, "Causes of inaccurate software development cost estimates," Journal of systems and software, vol. 31, no. 2, pp. 125–134, 1995.

[23] T. Little, "Schedule estimation and uncertainty surrounding the cone of uncertainty," IEEE software, vol. 23, no. 3, pp. 48–54, 2006.

[24] S. McConnell, Software project survival guide. Pearson Education, 1998.

[25] C. N. Parkinson and R. C. Osborn, Parkinson's law, and other studies in administration. Houghton Mifflin Boston, 1957, vol. 24.

[26] C. F. Kemerer, "An empirical validation of software cost estimation models," Communications of the ACM, vol. 30, no. 5, pp. 416–429, 1987.

[27] L. Mlodinow, The drunkard's walk: How randomness rules our lives. Vintage, 2009.

[28] J. Bernoulli, "Ars conjectandi: Usum & applicationem praecedentis doctrinae in civilibus," Moralibus & Oeconomicis, vol. 4, 1713.

[29] S. D. Poisson and C. H. Schnuse, Recherches sur la probabilite´ des jugements en matiere` criminelle et en matiere` civile. Meyer, 1841.

[30] S. K. T. Ziauddin and S. Zia, "An effort estimation model for agile software development," Advances in computer science and its applications (ACSA), vol. 2, no. 1, pp. 314–324, 2012.

[31] P. Kruchten, The rational unified process: an introduction. Addison-Wesley Professional, 2004.