Predicting Estimation Accuracy

Richard D. Stutzke

Abstract

Managers need estimates of project cost and schedule to plan and manage the development of software products. These estimates are approximate due to imperfect knowledge of the factors that affect these estimates. Project managers need information on the biases and uncertainty in these estimates to gauge the degree of risk and make planning decisions.

In this paper we first briefly discuss the factors that contribute to the errors (biases and uncertainties) in estimates of project cost and schedule. We then describe simple functions to estimate these errors versus time. Norman Augustine describes a curve for large complex systems that gives the ratio of the final actual cost to the estimated cost as a function of the fraction of the research and development period completed. Augustine's function is based on U.S. Department of Defense projects performed during the 1970s. We adapt this curve to represent the biases in a software cost estimate. Barry Boehm defines a trumpet-shaped curve giving the estimation accuracy versus time. We discuss the shape of this curve and propose a function for a modified curve to represent the uncertainty of the estimate versus time. We also discuss errors in the estimated schedule. Augustine states that the project always takes one-third longer than the currently estimated time-to-complete. We define a function for the ratio of the final actual schedule to the currently estimated schedule as a function of the elapsed time. We discuss various factors that could affect the shapes of these curves.

We also provide error data obtained from 19 completed projects. Specifically, we give the mean and standard deviation for the errors (actual minus initial estimate) as a function of the projects' Capability Maturity Model (CMM) level. (The CMM levels are defined by the Software Engineering Institute.) This data provides observed values of bias and uncertainty in initial estimates of cost and schedule and provides a possible way to calibrate our curves. Our curves provide a way to compute the decrease in these errors as a project proceeds, provided certain conditions are met. Although crude, this approach gives managers a possible way to gauge the accuracy of cost and schedule estimates.

1. Introduction

Project planners and managers use cost and schedule estimates to determine the feasibility of projects, to assess project risk, and to prepare budgets and plans.

Unfortunately, all estimates contain errors. There are at least three possible sources of errors. First, there are omissions. Engineers and planners often forget important functions that the software system must provide. In addition, they sometimes forget important tasks that are needed to complete the project. Examples are user training and quality assurance. Second, the engineers and managers may misunderstand some of the characteristics of the product, the process and the project. Possible areas for misunderstanding include requirements, product design, functions of the system, tasks and activities, and legal and financial constraints. These constraints include operational restrictions related to the user environment. They also include legal constraints such as penalty clauses and mandated development standards. They also may include work calendars such as the occurrence of various legal holidays. Third, the various quantities that are used to prepare estimates are sometimes measured in an

inconsistent way. Typical problem areas are the size of the software (measured in terms of source lines of code, function points, etc.) and productivity. Sometimes these measures are misunderstood. Sometimes historical data that is used for the purposes of calibration uses different definitions of the measures.

The consequences of these errors are biases and uncertainties in the estimates of project parameters such as cost and schedule. An estimated amount usually increases as a function of elapsed time on the project. This means that early estimates have a low bias. These quantities ("amounts") may be the size of the software product, the development effort, or the duration of the project (the "schedule"). Of course, on some projects certain parameters such as size, effort, or schedule may actually shrink compared to the initial estimated values. This might be caused by a decrease in the project scope, decisions to purchase commercial off-the-shelf components instead of building new software, etc. For most projects, however, growth seems to be the most common situation.[1]

Similarly, the uncertainty in the estimated quantity decreases as a function of the elapsed time on the project. This uncertainty represents the dispersion in a set of estimated values made at the same time. This uncertainty is usually expressed as the standard deviation of the estimated value.

The growth in estimated amounts can apply to size, effort, or schedule. We must be somewhat careful about measuring the growth in size because there are at least two views of size. First, the user or buyer of the system sees the size as the original functionality of the system plus the added functionality minus the deleted functionality. The developer, on the other hand, sees the size as the amount of added functionality plus the changed functionality plus the amount of deleted functionality. (These two different views of size are distinguished in function point counting.) It is not uncommon during the course of a project to observe growth and volatility in the size for either of the two definitions just mentioned (user, developer). Growth is typically viewed as the added functionality minus the deleted functionality. (The growth could also occur in the number of requirements, number of functions, etc.) Here, we tend to think of it in terms of source lines of code or function points. In its purest form, volatility refers to changes in portions of the system that have no net effect on the size. From the developer standpoint, however, any change plus any additions plus any deletions results in an impact on the project's effort and schedule. For this reason, the effort for the development of software should be computed using the developer's definition of size divided by the productivity. For the remainder of this paper we will focus only on the effort and schedule estimates for a project.

2. Correction Functions

At various points during the course of a project, we prepare estimates of the effort and schedule. (Strictly speaking, these are estimates of the effort remaining and the time remaining.) If it were somehow possible to define a model for a correction factor, we could apply this factor to the value estimated at any point in time and so could better estimate what the final value of that parameter will be. The concept is shown in the following equation:

$$f(t) = E_{true}/E(t)$$

[1] The sigmoid-shaped curve in the Figure 1 resembles similar curves called logistic curves. Logistics curves arise in situations where there is population growth in the presence of constrained resources. One of the first to study these situations was Verhulst in 1844 [1]. One might speculate that the curve approaches its final asymptotic value because the project only has finite resources.

We are interested in the possible shapes for the function f(t). If we assume that the team makes continuous progress then the function f will monotonically decrease toward the value of 1. If, on the other hand, there are changes in project scope, major redirections, etc., then one could expect various discontinuities to occur in the curve. These will usually be increases since such changes are not perfectly understood and so have omissions. The value of f could possibly decrease, however, if the change eliminated poorly understood, incomplete portions of the system. (Boehm et al discuss related behaviour in the uncertainty in the estimated value in Chapter 6 of [2].) For the remainder of this paper we will assume that the project is running in a stable environment.

2.1. Correcting estimated effort

Assuming that a project operates in a stable environment with no change in scope, what is a reasonable shape for the effort correction function f(t)? Norman Augustine analysed data for a large number of major research and development projects. Being "research and development" these projects had a high degree of "newness" and so we consider them to be similar to software development projects. Augustine obtained cost data from 81 projects and time history data from 38 projects [3]. He fitted in this data to obtain what he called the cost correction factor. The functional form of his relation is:

$$f(t) = a + b/(1 + c*t^3)$$

where t denotes the fraction of the Research and Development period completed and ranges from 0 to 1. (For software this period is equivalent to the total development time.) He obtained the values: a = 1.0, b = 0.52, and c = 8.0. We note that his function has the value of 1.06 at t = 1. (It is not clear why this value is not precisely 1.)

We decided to adapt Augustine's function to obtain a correction factor for the estimated software development effort. We can simplify the equations by invoking two constraints. First, the value of the function at t = 0 is 1 + ? where ? = $E_{final}/E_{initial}$. The parameter ? represents the error expected in the first estimate compared to the final actual value. We will also assume that f(1) = 1. Since we have three parameters we need an additional constraint. We have chosen the time when the slope of the curve is the greatest. This is a way of parameterizing the width of the "shoulder" where the curve begins to drop off. Given these assumptions, the parameters for Augustine's curve are:

$$a ? 1 - 2?t_b^3$$

$$b ? ?(1 ? 2t_b^3)$$

$$c ? 1/(2t_b^3)$$

Figure 1 shows the shape of this function for ? = 1.50 and t_B = 0.22. Why do we think this particular shape is plausible? It seems reasonable that no project can have an infinite amount of error at t = 0 so the value of ? must be some finite value. The reason is that if the estimated cost is above some threshold value, the project will never be started (or will at least be re-estimated). We believe that the "shoulder shape" is reasonable because the uncertainty in the estimate decreases slowly at the beginning of the project since we must assemble the team and the necessary resources. In addition, the team must establish their context before they can begin to make more accurate estimates. (This is the learning curve.) Effects such as requirements volatility can delay the decrease in the value of the correction function.

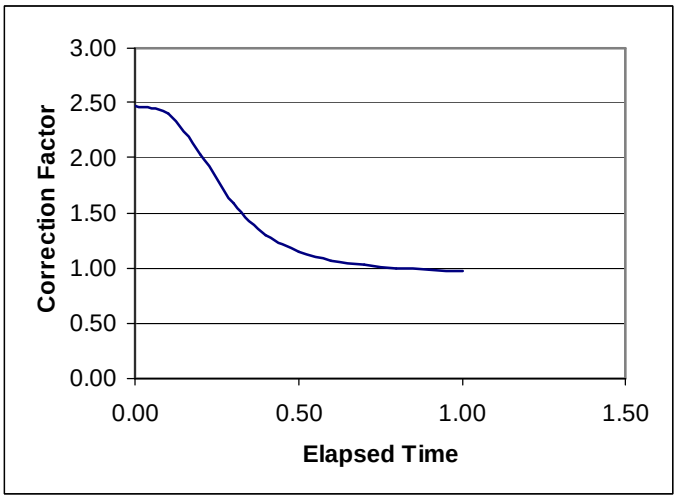

Figure 1: Typical Effort Correction Function

The smaller the value of t_b, the sooner the curve begins to drop down towards the asymptotic value. The width of the shoulder is most likely determined by factors such as the knowledge of the development team, the stability of the requirements provided by the customer, etc. High turnover of personnel (developer and customer) lead to requirements volatility and so broadens the shoulder. Turnover of leadership is particularly important, and occurs every four to five years on large Government projects. Another factor increasing the width of the shoulder is intermittent funding. Generally, the width of the shoulder is probably proportional to the total length of the project (t_B ? T) since this gives more time for changes to requirements, personnel, and funding.

What factors can affect the parameters for the function f(t)? The initial error, ?, depends on the thoroughness of the analysis, the experience of the team and the quality of the customer's specification. (Quality includes completeness, consistency, and level of detail.) If these various factors are lacking, this will lead to omissions, growth, and volatility in the product's functions and project's tasks. All of these factors will somehow contribute to the value of ?. In practice, each organisation must collect historical data and attempt to calibrate the value of ?. The experience and capabilities of the team (their learning curve) and the amount of time until the requirements stabilise or reach an acceptable level of quality influence the shoulder width, t_b.

2.2. Correcting estimated schedule

Augustine also observed that projects always take one-third longer to complete than the current estimate. This can be expressed mathematically as:

$$? - t_i = 1.33*(T_i - t_i) \text{ with } i = 0, 1, 2, ...$$

where

? = true (final) duration

t_i = time that the i-th estimate is made

T_i = the estimate of the total duration made at time t_i.

The generalised form of this is:

$$? = (1 + ?)*(T_i - t_i) + t_i = T_i + ?*(T_i - t_i)$$

We define a schedule correction function as:

$g(t) = ?/T_i = 1 + ? *(1 - t)$ where $t = t_i/T_0$.

We now look at the characteristics of this function. First we note that the value of ? would be determined by calibration for each particular organisation. Augustine quotes a value of 0.33. Microsoft seems to use a value of 0.3. [4].

Note that the value t is a normalised time. This value of time should be used in both the effort correction function and the schedule correction function, f(t) and g(t), respectively. In the above equation we have chosen to normalise t_i by T_0. The reason that we chose T_0 is that this value is available at the start of the project ($t_i = 0$) and it remains constant throughout the project. That is, its value is well known even though the value may not be accurate. After some thought, we decided instead to normalise the time by $?_0$ since this value represents an improved estimate of the actual project duration. This value is easily computed using:

$?_0 = T_0*(1 + ?)$

since at the start of the project $t_i = 0$. This is quite easy to do and it should produce better predictions. (We will see this in the next paragraph.) The normalised value of t that we will use in f(t) and g(t) is given by:

$t = t_i/[T_0*(1 + ?)]$

Note that t is normalised time which is a dimensionless number. The other parameters, t_i, T_0 and ?, have the units of time (either work days or calendar days as will be discussed shortly).

Figure 2 shows a plot of the schedule correction function applied to a hypothetical set of schedule estimates (shown by the "current estimate" line) which represent eventual convergence to the final value of 1.25. (The initial estimate was 1.0). The other two curves in Figure 2 show the schedule predicted using our function with the current values and ? = 0.3 with and without re-normalising the time variable (i.e., using T_0 and $?_0$, respectively). The project begins at t = 0 and is completed when t = 1.25. We see that re-normalisation is essential. The prediction without the normalised time fails to converge to the actual value.

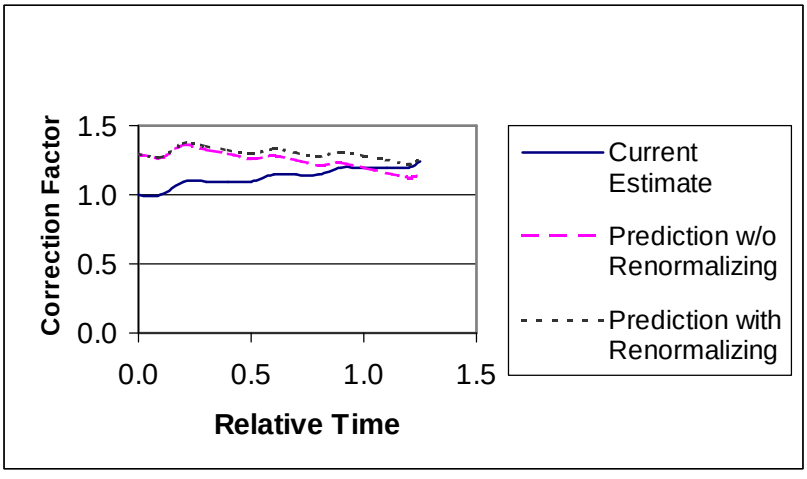

Figure 2: Current Schedule Estimates versus Predictions

This simple model for predicting the schedule assumes of course that the scope and other characteristics of the development project are not changed. This is seldom the case in practice and we will discuss this in Conclusions at the end of this paper.

2.3. Predicting the accuracy of the estimated effort

Figure 3 shows the accuracy of an effort (cost) estimate versus elapsed time to project. This curve is taken from [5] and also shows the sources of uncertainty at each point during the project's life cycle. At the start of the project, before feasibility has been determined, the uncertainty in the estimated value is shown as ranging from a factor of 4 down to a factor of 0.25. Note that this range is not symmetric. Boehm quotes the range as N and 1/N, and not ?N as one might expect. We call this the "trumpet curve" and it shows monotonic convergence toward zero error as the project approaches completion. Such monotonic behaviour arises because of the assumption that the requirements for the project are stable, the processes are well defined, and the project is well managed. Some readers might also argue that the error in the estimate never goes to 0 because it is impossible to accurately know the true cost of the project when it is completed. (This problem can arise if accurate cost accounting is not performed.)

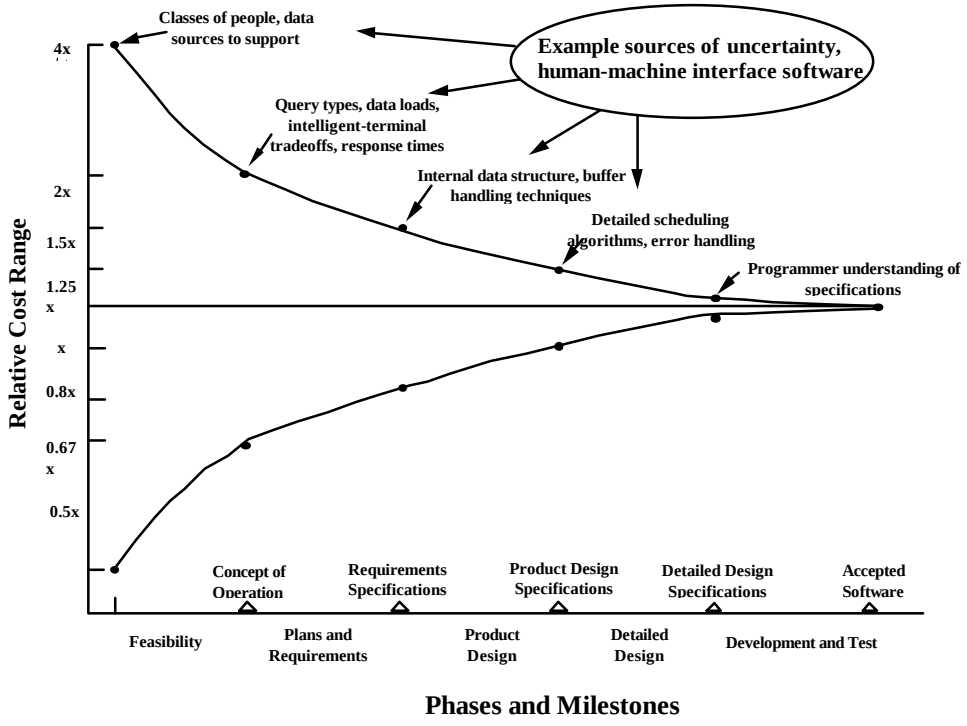

Figure 3: Decrease in Estimation Uncertainty versus Time (from [5])

Boehm's curve assumes that the requirements are firm. It assumes that the team can make steady, uniform progress. We believe that a better model might have the shape similar to that of Augustine's function that was described earlier. The reason is that projects with large uncertainties (dispersions) will either be re-estimated or cancelled. Also, during the start, the team will probably not make much progress in reducing the uncertainty in the estimate. There is a certain period of time that is needed to assemble the necessary resources and to get the team organized.

For these reasons we propose using the form of Augustine's equation and simply fitting the parameters to match certain points on Boehm’s curve. We will only consider the upper

excursion of Boehm's curve since we are interested only in potential cost overruns, not in underruns. Our function for the accuracy of the estimated effort is:

$$? E/E = a + b/(1 + c*t^3)$$

We fitted the following points to this equation:

$$f(0) = 4.0$$

$$f(1) = 1.0$$

$$f(0.56) = 1.25$$

where the last point corresponds to the Product Design Review (PDR). The value of 4 comes from the start of concept definition ("feasibility"). The value of 1.0 is an assumed characteristic of our function. That is, there is no correction necessary when the project is finished because we know the actual value. The resulting parameters are: a = 0.941, b = 3.06, and c = 50.6.

2.4. Predicting the accuracy of the estimated schedule

We did not locate any published models predicting schedule accuracy (dispersion) so we propose the following approach.

$$? = \text{time now} + \text{time remaining} = t_i + k*E_{remaining}/R \quad (1)$$

where $E_{remaining}$ = estimated effort needed to finish (person-hours), R = average effort delivery rate for the remainder of the project (person-hours/work-day), and k = factor to convert work-days to calendar-days.

In this equation we must carefully distinguish the units of time. We measure duration in work-days. We measure schedule in calendar-days. Duration is important for estimating time because the project staff only delivers effort to the project tasks on work-days. On the other hand, managers are interested in calendar time and so we must convert from work-days to calendar-days. We show the factor k to accomplish this. For more details on this topic see [6].

There are, of course, sources of error in each factor in this equation. The current time, t_i, has no error since it is just the number of days elapsed since the start of the project. The factor k is also well-defined if the resource calendar is specified. (The resource calendar indicates weekends and holidays.) The estimated effort remaining, $E_{remaining}$, has the usual sources of errors. One factor is omissions, misinterpretations, and errors in measurements. The effort delivery rate, R, is affected by several factors. One factor is the continuity of the staff (turnover). Another is the availability of the staff to work on the particular project. (Sharing staff between projects often results in wasted effort because of the need to switch contexts.) Another factor affecting the effort delivery rate is the ability of the management to deliver additional staff to the project as planned. A final possible factor is the exhaustion of the staff resulting in lower productivity. (This could occur due to fatigue or burnout.)

We can estimate the error in the predicted schedule of the project by taking partial derivatives of the equation defining ? (Equation 1 above) which gives:

$$??? \frac{???}{??_r} ??_r ? \frac{???}{?R} ?R$$

This is called a logarithmic derivative. The usual approach is to calculate the average fractional error by taking the square root of the sum of the squares of each contribution to the total error. This gives the equation:

$$\frac{??}{?} ? \left[\left(\frac{?? _r}{? _r} \right)^2 ? \left(\frac{??}{R} \right)^2 \right]^{\frac{1}{2}}$$

It only remains to determine what the error functions are for the remaining effort and for the effort delivery rate. It is plausible that the same type of function used for the total estimated effort represents the uncertainty in the estimated error. (This would be either Boehm's trumpet curve or our analogy to it based on Augustine's function.) Many factors contribute to the uncertainty in the effort delivery rate. We discussed these factors previously. We can say, however, that we expect this function to scale as (1 - t) raised to some power. The reason is that such a functional form guarantees that the error will be 0 when the project completes. (As discussed previously, however, this may not be realistic. If the cost accounting system is not accurate, it is possible that errors will remain at project completion in both the estimated effort and in the correct effort delivery rate.) As the first step, we suggest the following functional form:

$$?R/R = ?*(1 - t)$$

where $0 < ? < 0.25$. Each organisation will have to determine a suitable functional form and a value of ?. Here we chose the value of 0.25 to agree with the comments of many authors that programmers spend from one-fourth to one-third of their time on non-project tasks. For example, see page 341 of [5]. DeMarco and Lister also point out the importance of uninterrupted time on pages 62-66 of [7].

3. Actual Data on Cost and Schedule Estimation Accuracy

We were able to obtain data on 19 completed projects from an organisation that desires to remain anonymous. However, we can say that the projects in this organisation dealt with the same application domain and the same general types of technology. That is, the organisation was involved in a business area where products of a similar type were being produced. In addition, this organisation was involved in military contracts and consequently all projects followed a formal, requirements-driven development process.

We were able to separate the data for the 19 projects into three groups by SEI SW-CMM level as shown in Table 1. Table 1 shows the estimation accuracy for both cost and schedule as a function of the CMM level. These accuracies are stated in percent and are computed in terms of the initial estimated value compared to the final actual value. For Level 1, the average cost overrun was approximately 17% and the average schedule overrun was 27%. For Level 2, the average cost overrun was 6.3% and the schedule overrun was 2.5%. For Level 3,the average cost overrun was 7.2% and the average schedule overrun was 5.3%. This data seems to indicate that there is an improvement in the organisation’s ability to estimate cost and schedule as the process maturity advances from Level 1 to Level 2. Specifically, for effort the mean bias and standard deviation both decrease over 60%. For schedule, they decrease by 80-90%. In contrast, however, there seems to be a slight *decrease* in estimation accuracy as the organisation went from Level 2 to Level 3. This decrease in accuracy is not statistically significant, however, since there are only four data points for the Level 3 projects. Overall, we note that the standard deviations, which are also shown in Table 1, are nearly as large as the mean values. Based on Student's t-test, the probabilities for these mean values being

consistent with zero are very high. Thus, the above comparisons are only suggestive and not validated by the data.

Table 1: Estimating Accuracy (%) versus CMM Level

CMM Level	# Data Points	Cost (? ±?)	Schedule (? ±?)
1	9	17.2 ± 12.9	27.1 ± 22.4
2	6	6.3 ± 5.1	2.5 ± 3.8
3	4	7.2 ± 7.3	5.3 ± 7.5

4. Conclusions

We have developed some crude models that hopefully provide insight into how estimation errors should behave. We looked at estimates of both cost and schedule, and investigated both biases and uncertainties (dispersions). Our models assumed a stable environment. These include the requirements for the system, the design of the product, and the quality of the personnel and the project management. Readers may question why these models have utility since the environment seems to always change on any software project. We believe, however that even if certain factors do change in the project environment, it is possible to reinitialize the equations and continue calculating from that point onward. It is merely necessary to set the value of ? to the "time remaining".

5. References

[1] Çambel, A.B., "Applied Chaos Theory: A Paradigm for Complexity", Academic Press, 1993. Page 94 describes Verhulst's work.

[2] Boehm, B.W., Steece, B., Reifer, D., Madachy, R., Horowitz, E., Clark, B.K., Chulani, S., Brown, A.W., Abts, C., "Software Cost Estimation with COCOMO II", 2000.

[3] Augustine, N.R., "Augustine's Laws", Viking Penguin Inc. First published by the American Institute of Aeronautics and Astronautics in 1983.

[4] Cusumano, M.A., Selby, R.W., "Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People", The Free Press, 1995.

[5] Boehm, B.W., "Software Engineering Economics", Prentice-Hall, 1981. The curve is taken from page 311.

[6] Stutzke, R.D., "A Mathematical Expression of Brooks' Law", Proceedings of the Ninth International Forum on COCOMO Conference and Cost Modeling, Los Angeles, CA, October, 1994.

[7] DeMarco, T., Lister, T., "Peopleware: Productive Projects and Teams", Dorset House, 1987, ISBN 0-936633-05-6.