

Monte Carlo Simulation for Software Maintenance Cost Estimation

P. MacDonald
General Dynamics
macdonal@gdls.com

F. Mili
Oakland University
mili@oakland.edu

Abstract

All software estimation techniques rely in some way on past experience to predict the time and effort required to accomplish a software task. The accuracy of these techniques depends on 1. the extent to which past experience is relevant to the a new project at hand, 2. the extent to which any variation has been accounted for and parameterized, and 3. the skill of the user estimator in assigning values to parameters. In this paper, we explore the use of in-house organization's metrics for the purpose of obtaining an estimation tool method that is both more accurate and easier to implement. This approach is evaluated using existing data from software maintenance programs of embedded systems at General Dynamics. We develop a simple, flexible and accurate model using historical data and simulation modeling. The model is simple because it is mostly based on extrapolation from historical data, additional parameters are added in a graphical way. It is flexible because the simulation modeling tool allows users to account for a variety of parameters. It is accurate because the simulation modeling tool allows users to account for as many parameters as needed.

1. Introduction: Problem, Approach

There are an ever-increasing number of legacy software systems and systems with a lengthy life span. The software embedded in combat vehicles and in aircrafts such as the F-16 as well as major operating systems such as Microsoft Windows and Sun Solaris are only a few examples of long-lived systems. The maintenance of these systems is a major endeavor and continues through the life of the program (the F-16 has been operational for over 20 years.) Even though software maintenance is a predominant activity—generally estimated to consume 70 % of the overall cost [1,2]—it is inadequately served by tools and techniques for cost estimation. Most software estimation techniques have been designed to estimate the cost of initial development of new programs, to the exclusion of maintenance activities [5]. Some of these methods have been adapted to estimate the cost of making changes to existing software[16]. Generally, we have found that using these adapted techniques for maintenance is not adequate for the following reasons:

- The software estimation techniques, whether based on lines of code or function points, rely on parametric models. Using these parametric models [3,4,7] involves calibration by setting of parameters and tailoring of scale factors to capture the specifics of a program and the environment in which it is developed. The overhead involved in using these parametric methods and the amount of information that they require are too high. Maintenance contracts are often negotiated without the benefit of a detailed analysis for each of the changes included in the contract.
- The parametric models are geared towards the cost estimation of new programs rather than on a set of individual changes. Maintenance is in general negotiated and performed in contracts. A maintenance contract consists of a number of changes (anywhere from 20 to a few hundreds). The general accuracy of parametric cost

models has been described to be “within 20% of actuals, 70% of the time.” [9] What is needed is a method that can provide accurate estimates for complete contracts.

- In most software maintenance environments, there is a wealth of historical data about previous maintenance programs with very similar characteristics. This data is rarely used to its full potential.

In this paper, we examine historical data from maintenance programs at General Dynamics, and explore the possibility of their use for the estimation of cost of maintenance contracts. We are particularly interested in finding mechanisms for the estimation of software cost that are

- *Simple.* We would like to account for the fact that contracts are negotiated with little prior information. The method must require minimal information and must not depend on the skill level of the person using it. The model should be no more complex than the existing parametric models (i.e., COCOMO II, REVIC, SPQR20)
- *Contract-Accurate.* The goal is to be able to negotiate contracts with a good estimate. We will tolerate low accuracy at the change level as long as the estimate for a whole contract is relatively accurate. Since the estimate is made prior to detailed analysis, historical data indicates that a variance greater than 25% can be expected [18]

The work presented in this paper is experimental in nature. We start from the fundamental premise underlying all work in software estimation, with is:

Underlying Premise:

“When software engineers with similar skills develop software with similar specifications under similar environments, they incur similar costs.”

We aim at using the similarity between programs maintained within the same company (e.g. embedded software in combat vehicles). Instead of formulating hypotheses and then testing them, we take a pragmatic approach. We analyze historical data and look for patterns of similarity that can be used for formulating an estimation technique that would meet the criteria of simplicity and contract-accuracy identified above.

The paper is organized as follows. In section 2, we present data collected and analyze it In section 3, we formulate simple linear models based on the data. We assess these models with respect to the two criteria of simplicity and accuracy. In light of the results with the linear models, we turn to the use of simulation. The model based on simulation is discussed in section 4. We summarize and conclude in section 5.

2. Historical Data Used

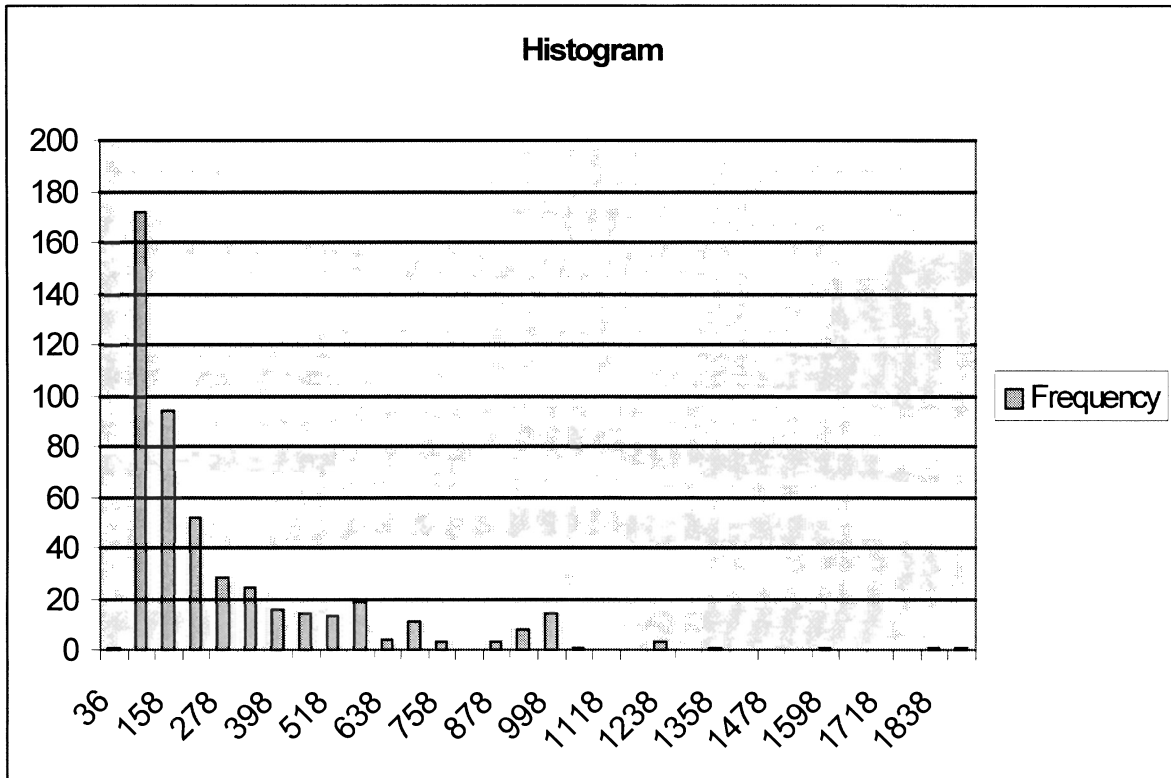
The Capability Maturity Model (CMM) requires the use of historical data to produce new software estimates [8]. As a result, it is not uncommon for organizations involved in the development of software to have available data related to the development and maintenance of software. The metrics data collected from maintenance programs would include such things as

- Some identification or brief description of the change performed,
- Effort required to make the change.
- The type of change, i.e. user interface, communications, diagnostics, etc.
- The size of the change (LOC, number of units, etc.)
- Test procedures and test results after change.
- Personnel involved in performing the change.
- Root cause that triggered the change (change in requirements, coding error, etc.)

- Files affected by the change.
- Other changes triggered by this one (errors introduced in the process of executing this change).

This data can be more or less extensive, and most of it is potentially relevant for formulating similarity patterns.

In this project, we have used data collected from several maintenance projects over the last four years at General Dynamics. This covered 500 different changes recorded. Because we are trying to identify the simplest possible model, and because not all information is recorded for all data points, we have elected to start with only one attribute: the effort required to make the change. The data used is shown in the Histogram[10] in figure below.



In the histogram, each bar represents the number of changes (y value) that required a specific number of hours (x value). For example, there were two changes that took 36 hours each. On the other hand, there were 170 changes requiring 100 hours. Overall, changes took anywhere between 40 and 1800 hours with the majority (more than 75%) taking between 100 and 220 hours.

In addition to this historical data, we have used three on-going contracts to test and refine hypotheses and models. The three contracts contained respectively 36, 30, and 16 changes.

2. Linear Contract estimation models

We start by using the simplest possible model. In light of the high concentration of points around the 131.4 mean (75% of the changes within the range [100,220]), we start with a model that is based solely on that median.

3.1 Model 0: Mean Cost

Model 0

For any maintenance contract MC, the cost estimate of the contract is

$$\text{Cost_estimate(MC)} = \text{NumberOfChanges(MC)} * \text{MeanCost}$$

Where the MeanCost is the mean of the cost of changes in the historical data.

Intuitively, the larger the number of changes considered, the more accurate is the cost estimate. This model meets the criterion of simplicity. We need to assess its level of accuracy.

Accuracy of Model 0:

Traditionally, accuracy of models is measured by the probability that the estimate is within a reasonable ratio of the actual value, i.e., we are interested in estimating the likelihood that the estimate is within 20% (or 10%, or 5%). Calculation of this likelihood will depend on the data distribution. In the case of the data we have, the mean is 131.4 hours. Given k changes, we are interested in knowing the probability that the actual cost falls within 20% of the estimated cost of $k * 131.4$. If we assume that the different changes are independent,

$$\begin{aligned} P(\text{Cost of a contract with } k \text{ changes is within } 20\% \text{ of estimate}) = \\ P(.8 * k * \text{Mean} \leq C_1 + C_2 + \dots + C_k \leq 1.2 * k * \text{Mean}) \end{aligned}$$

where C_i is the cost of the i th change. Because of the compensating effect, the probability increases with k and tends to 1 when k tends to infinity.

This shows that even this simplistic model can provide a very good estimate of the cost of contracts. The accuracy – and thus usefulness- of this model depends on the data distribution of past changes and on the size of contracts.

Because there will be cases where the distribution is not sufficiently concentrated around the mean (large variance) or contracts contain a small number of changes, we need to identify other models. We consider exchange some of its simplicity for a higher level of accuracy. This can be done by customizing the estimation of cost of changes. In other words, rather than considering all of the changes in the historical record as part of a common category, we aim at identifying different categories of changes with narrower ranges of cost.

3.2 Model 1: Mean Cost by category

We turn back to the historical data available. We note that attempts to capture the data with a single random distribution function were unsuccessful. We subsequently attempted to capture it with a small number of random distribution functions. We continue to use only one attribute about the changes, the cost. Eventually, through trial and error, the data was segmented into three (slightly overlapping) data sets each with its own random distribution function.

We validate the distributions identified statistically. The statistical validation provides results within the acceptable limits, confirming that there is a good likelihood that the random

distribution function is acceptable [11]. Complete identification of the random distribution functions and the test results are shown in the table 1 below.

Because the three categories span the cost ranges [39-79], [75-499], and [505-1860] respectively, we label them as Easy, Typical, Complex. These labels represent an *à posteriori* interpretation. Overall, the categorization of the changes into one of three classes is useful only to the extent that it can be done *à priori*. Stating after the fact that a change that took 1100 hours to make was complex is of little value. The three categories are of interest only to the extent that a contract negotiator can easily (with no detailed analysis) and accurately classify changes as easy, typical, and complex.

	Easy	Typical	Complex
Distribution:	Triangular	Lognormal	Exponential
Expression:	TRIA(39, 55.7,79)	75 + LOGN(130, 70)	505 + EXPO(293)
Square Error:	0.012190	0.010243	0.004139
Chi Square Test	0.202	0.0501	0.137
Kolmogorov-Smirnov Test [10]	0.119	0.0534	0.0716
Data Summary			
Number of Data Points	133	340	73
Min Data Value	39.1	75.5	506
Max Data Value	78.4	499	1860
Sample Mean	57.9	209	798
Sample Std Dev	8.57	87.7	285
Histogram Summary			
Histogram Range	39 - 79	75-499	505 - 1860
Number of Intervals	12	12	6

Table 1

The selection of distribution functions allow the software engineer quantify verbally that "this task is easy while the next one is complex" as opposed to estimating the hours to implement a correction. This simplifies the estimation process by the engineer and allows the simulation to generate the effort required to implement the changes. [19]

Based on the above abstraction of the data, we formulate a second model:

Model 1:

For any maintenance contract MC, the cost estimate of the contract is produced as follows: classify the changes according to the categories identified from the data analysis , count the number of changes in each category, and compute

$$\text{Cost_estimate}(\text{MC}) = \sum \text{NumberOfChanges}(\text{MC}, i) * \text{MedianCost}(i)$$

Where $\text{NumberOfChanges}(\text{MC}, i)$ is the number of changes of category i , and the $\text{MedianCost}(i)$ is the median cost for the category i .

From the data we have, contracts will be estimated by using the three counts NumberOfSimple , NumberOfTypical , and NumberOfComplex , and computing:

$$\text{Cost estimate}(\text{MC}) = \text{NumberOfSimple} * 57.9 + \text{NumberOfTypical} * 209 + \text{NumberOfComplex} * 798$$

We assess the simplicity and accuracy of this model.

Simplicity of Model 1:

The process of using the model is rather simple. The main issue is in classifying the changes. We are assuming that a simple description of the change would allow its accurate classification. Obviously this is a strong assumption. Further data analysis is needed to identify possible strong correlation between cost and other attributes of the changes. We proceed nevertheless with the testing of this model as is.

Accuracy of Model 1:

Because Model 1 is a refinement of Model 0, it is bound to be more accurate than Model 0, provided changes are classified correctly. We have tested this model with the three ongoing contracts, leading to the following results:

Contracts	Estimate using Model 1	Actual Cost	Percentage Error
Contract 1 with 36 changes	11465	14710	22%
Contract 2 with 30 changes	7115	9424	14%
Contract 3 with 16 changes	6936	8468	18%

This data is definitely too small to allow us to make any general conclusion. However, the results do give us an indication of whether we are on promising path or not. In this case, the data seems to indicate that 1. The model is reasonable since all estimates are within 25% of the actual cost, 2. The model may be underestimating the actual cost of contracts (all three estimates are under by at least 18%). This latter observation sent us back to the data to revise some of the assumptions we were making.

Review of the data and of the General Dynamics software development processes made us retract the assumption of linearity. The cost of a contract is more than the sum of the costs of its changes. After the software changes have been implemented, the software undergoes independent system level testing. Defects identified during system testing were not attributed to an existing change. [6] The cost of correcting these defects is recorded independently at General Dynamics. Therefore, the fulfillment of a contract with 30 changes may require in fact the execution of 36 changes, the 30 requested, and an additional 6. Accounting for this and other business processes would require a more flexible modeling process. This led us to using simulation tools.

4. Contract estimation Using Simulation

4.1 Monte Carlo Simulation Model

Traditionally, simulation methods referred to all methods used materialize and visualize the behavior of systems based on a model. Their definition has been extrapolated to also refer to methods that generate the model from behavioral data. Depending on the nature of data and events, the models are characterized as discrete or continuous, static or dynamic, and stochastic or deterministic. A Monte Carlo simulation is discrete and static and can be either stochastic or deterministic [11,15]. Stochastic simulation relies on the use of random numbers for input where as a deterministic simulation relies on a known set of values. Monte Carlo simulations, as a static simulation, do not require the analysis of data over time.

For any stochastic simulation, the models require a good random uniform distribution between 0 and 1. The generation of such a uniform distribution is beyond the scope of this paper. To produce the models for this paper, the modeling program Arena has been used [17].

4.1 Generic Model with Simulation Tool Mg

We have opted to upgrade Model 1 to account for the self-generating effect of changes. Data analysis indicated a 20% incidence of this phenomenon. Rather than building a model that explicitly captures this feature, we have preferred to build a generic model that would allow us to experiment with this and other features as needed. Thus, the use of simulation models.

The model was produced using the simulation tool Arena. Arena was selected on several factors the least of which was the author's level of understanding of the tool and to Arena's ability to support input and output analysis as well as the developing the simulation models. The Arena graphics simulation system is a flexible modeling environment combined with a graphical user interface. Since the model is build graphically, the modeler is not required learn a simulation programming language.

Arena is designed for building computer models that accurately represent an existing or proposed application. Arena was developed as a tool to simulate manufacturing processes. In simulating a manufacturing process, material or parts arrive at the facility and wait to be used. The parts are then processed by one more resources or machines and prior to being processed, the part waits in a queue. The modeling of this process can evaluate the amount of time(s) required to process a part(s). It is this concept of moving software correction entities through a set of developer resources that applies to modeling the effort required to develop software.

Some of the processes that can be accounted for by the modeling tool are

- The assignment of the time required to implement a change based on a random distribution function(s)
- The self-generating feature of changes, where a task can trigger the creation of other tasks. This recursive process may be hard to formalize and compute manually, but can be easily and visually modeled using Arena.
- The number of developers assigned to execute a contract. Different characteristics about the parallel processing of changes can be graphically represented within the model and provide an estimate of the project duration.

4.2 Illustration of Model Mg Simulation

We have used the Arena simulation tool to account for the 20% self-generating effect of changes. The input to the model consists of

- The three random distribution functions generated from table 1,
- The count of changes from each of the three categories whose distributions are captured by the random functions above.

The model then simulates the maintenance process. Changes are placed in a waiting queue with a specification of their associated random distribution function. Each change waits in the queue until a developer is available to execute it. Once a developer is available, the time it takes to execute the change is calculated using the corresponding distribution function. Upon completion of the change, the time to implement the change is added to the total time for the developer.

In order to account for the self-generating effect of changes, we add an extra step to the process, namely a pass/fail distribution with a failure probability of 20%. In the case of failure, we generate a change and place it in the queue. We have not created any correlation between the category of the original change and that of the generated one (a typical change may trigger a simple, typical, or complex one for example). Such refinement can be added if desired. The process of change execution continues recursively until the queue is empty and all changes have been executed. The total cost is calculated by summing the costs for all developers. The estimated duration of the contract is defined as the maximum single developer time. As a comparison, the model also uses the COCOMO II duration equations based on the total time.

By the very nature of simulation, a single execution of the model does not provide a valid result. The value of each result will vary due to the random distribution functions. A number of runs is required in order to achieve confidence that the mean of the results from the simulation is valid. The verification of the simulation is determined using a confidence interval based on the relationship of the half-width value and the mean. In order to achieve a confidence interval greater than 95% [13], we need to ensure that the half-width is less than 5% of the mean. We need to run the simulation model 1000 times [14] to achieve the desired confidence.

$t_{N-1,(1-\alpha)/2} * \sigma \sqrt{N}$ with $\alpha = 0.95$ and $(1-\alpha)/2 = 0.0,52$ and $t_{N-1,(1-\alpha)/2} = 2.78$ for a 95% confidence interval with a sample size of 5. The value of t is determined using a standard t-distribution table.

Using the above simulation model, we obtain the following results:

Contracts	Estimate using Model Mg	Actual Cost	Percentage Error
Contract 1 with 36 changes	14369	14710	2.5%
Contract 2 with 30 changes	9848	9424	4.5%
Contract 3 with 16 changes	8023	8468	5.0%

4.3 Assessment of Model Mg

The Model Mg meets the criteria of simplicity and accuracy. It encompasses both Models M0 and M1 but allows users to graphically upgrade these models and also to visualize the interactions between different features of the software development and maintenance processes and the software development environment.

As with M1, the accuracy of Mg is based on the assumption that users can accurately classify changes without prior analysis. The three contracts we have used for testing these three models do not give us a good indication as to how realistic this assumption is. These three contracts were well underway before we started the testing of the models. In allocating the changes to simple, typical and complex, the software developers may have had a better understanding of the size of the change (in hours) than they would have had at the project start. Further work is required to determine if this approach is valid for new functionality, assuming there is no significant changes in the development environment. Significant changes (i.e., new methodology, and different platform) are probably not suitable to this estimation approach.

5. Summary, Conclusions, and Future plans

In this paper, we have addressed the need for accurate methods for maintenance contract negotiations. Even though maintenance is *the* predominant activity in software development, it did not receive the needed level of attention in terms of cost and time estimates. We have started exploring experimental approaches to software maintenance that are specifically tailored to the nature and needs of this activity. In particular, we have used the fact that:

- Software maintenance contracts are collections of changes. Methods based on approximate estimates of individual changes could lead to accurate estimates for the overall contract.
- Software maintenance contracts are negotiated without the benefit of prior analysis. This often rules out parametric models.
- Most software houses do maintain extensive records about previous maintenance contracts of the same software system as well as maintenance contracts of very similar systems.

The paper describes three different models of increasing accuracy that have been tested in our context. Although we are still in the early stages of testing and refining our own estimation models, we feel that the approach is promising and can already be tried in other environments. The main lessons that we have drawn so far are:

- Very simple models based on historical data are very un-expensive and can give very good results.
- It is important to have models that can be easily tailored to specific development environments in a simple and intuitive manner.
- Simulation tools available provide a invaluable resource that can be used to analyze historical data, detect patterns, test correlations, and model processes. Arena is an excellent to produce prototype simulation models and is definitely easier to develop than using programming languages such as SIMAN or C++. Arena also provides an excellent mechanism to analyze both input and output. However, the model produced by Arena is not readily modified.
- Experience with using historical data for estimation gives indication as to what data is most useful and needs to be collected.

In our context, we have started exploring the use of other dimensions of the data in order to facilitate the categorization of errors and make better assignments of changes to developers.

Acknowledgements:

We thank Professor Sankar Sengupta from Oakland University for his invaluable input concerning the simulation aspects of this work.

References:

1. Leintz, P.B. and Swanson, E.F., *Software Maintenance Management*, Addison-Wesley, 1980.
2. Strak, G.E., *Measurements for Managing Software Maintenance*, Proceedings of the 1996 International Conference of Software Maintenance.
3. Agawal, R, et al, *Estimating Software Projects*, ACM SIGSOFT Software Engineering Notes, Vol. 24, No. 4 July 2001.
4. Bohem, Barry, et al., *Software Cost Estimation with COCOMO II*, Prentice-Hall, 2000.
5. Hulkower, Neal D., Vitkevich, John A, *A Framework for Estimating the Cost of Fixing the Year 2000 Problem*, Mitre, 13 February 1997.
6. McConnel. S., *Software Project Survival Guide*, Microsoft Press, 1998.
7. Jones, T.C., *Estimating Software Costs*, McGraw-Hill, 1998, Section 6.
8. Paulk, M.C., et. al., *Capability Maturity Model for Software, Version 1.1.*, CMI/SEI-93-TR-24, Pittsburgh, PA., Software Engineering Institute, Carnegie Mellon University, 1993.
9. Royce, W., *Software Project Management, A Unified Framework*, Addison-Wesley, 1997.
10. Hines, W.W., and Montgomery, D.C., *Probability and Statistics in Engineering and Management Science*, 3rd Edition, Wiley, 1990.
11. Banks, J. et al, *Discrete-Event System Simulation*, 2ed Edition, Prentice Hall, 1999.
12. Ritter, Terry, 24 June 1998, <http://www.io.com/~ritter/JAVASCRP/NORMCHIK.HTM#KolSmir>
13. Pegden, C.D., R Shannon, and R. Sadowski (1995), *Introduction to Simulation Using SIMAN*, 2nd Edition, McGraw Hill, Inc, New York.
14. Raatikainen, K., *A Sequential Procedure for Simultaneous Estimation of Several Means*”, ACM Transactions on Modeling and Computer Simulation, Vol 8, No, 2 April 1993, pages 108-133
15. Law, A. M. and Kelton, W. D., *Simulation Modeling and Analysis*, 3rd ed., McGraw Hill, 2000.
16. Bennatan, E.M., *On Time Within Budget*, 2nd ed., McGraw Hill, 1995
17. Kelton, W.D., et. al., *Simulation with Arena*, McGraw-Hill, 1998
18. Roetzhein, W., *Estimating Software Costs*, Software Development, Oct 2000, Vol. 8, No.10.
19. Miranda, E., *Improving Subjective Estimates Using Paired Comparisons*, IEEE Software, January/February 2001, Vol. 10, No. 1.