Software Made Simple: Effort Adjustment Factors and the Accuracy of the Estimate

Jeremy Goucher

MCR, LLC

June 2018

Abstract

This research investigates the sensitivity of estimated hours for software development in relation to effort adjustment factors (EAFs). The analysis highlights the importance of performing original data analysis for new estimates, rather than relying on rules of thumb or industry standards. Further, a regression method is proposed as an alternative to traditional equivalent source lines of code (ESLOC) methods and a proof is provided to show the relationship between the regression method and ESLOC methods. Finally, the analysis is supported with data from over thirty historic programs.

Table of Contents

Introduction

Software development cost are particularly difficult to estimate. Further, a single software intensive project can exceed $1B. GAO reported in 2013 that "IT projects too frequently incur cost overruns and schedule slippages…" Further, four recent DoD programs reported a combined $1.3B in cost overruns largely due to poor cost controls, according to the GAO (Powner, 2013)[1]. Nearly all modern DoD projects include some sort of software development. This research investigates the sensitivity of estimated hours for software development in relation to effort adjustment factors (EAFs). The analysis highlights the importance of performing original data analysis for new estimates, rather than relying on rules of thumb or industry standards. An alternative regression based method is proposed to overcome the subjective bias that is incurred naturally from the ESLOC method. The regression method is supported with data from over thirty historic programs.

Data Set

This study leverages historic data from 33 Department of Defense programs spanning 2001 to 2014 across all four major branches of armed services. There are a total of 212 computer software configuration items (CSCIs) covered by the data. They include initial software lines of code (SLOC), final SLOC, initial hours, and final hours. The SLOC is reported by new code, modified code, reused code, and autogenerated code. For the purposes of this study, autogenerated code is assumed to be zero effort and is ignored. The amount of autogenerated code reported is minimal, only 10 of 212 CSCIs reported autogenerated code. A single test case was conducted using autogenerated code and the inclusion did not impact the results.

The size, required hours, maturity, and coding languages used varied across the programs. The largest program required over 3 million person-hours to complete. The smallest required only 22K hours. Approximately 50% of the observations were considered "new" efforts, the remainder were considered "upgrade" efforts. The coding languages included Java, C, and Ada type languages, with multiple subtypes (ex: C++ or Ada95).

The programs also include a wide array of system types. There are radar systems, satellite systems, command and control systems, user interface systems, communication systems, among others. Further, the data set includes systems developed by many different companies or organizations. This robust data set is ideal for a software analysis because any programmatic bias is removed and the analysis focuses strictly on quantitative data.

Basic ESLOC Method

Equivalent Source Lines of Code (ESLOC) is a normalized measure of the amount of code that needs to be or has been written. The most basic ESLOC method uses effort adjustment factors to generate ESLOC, then computes an ESLOC growth rate and productivity rate to estimate hours required. An effort adjustment factor (EAF) is the normalized effort required to develop a line of code. EAFs are typically categorized by new, modify, and reuse and apply to the associated type of code. The rationale behind

1. Powner, D. A. Information Technology: OMB and Agencies Need to More Effectively Implement Major Initiatives to Save Billions of Dollars. District of Columbia: United States Government Accountability Office, 2013. Retrieved from GAO.gov: https://www.gao.gov/assets/660/656191.pdf

EAFs is that it takes less effort to incorporate previously developed code (aka reuse code) than it does to modify code, and less to modify code than it does to develop new code.

Growth rate is an adjustment made to an ESLOC estimate based upon the historical accuracy of the initial estimate. It is measured by comparing the initially estimated ESLOC to the final ESLOC that is developed on a given program. Measurements assume the same EAFs are used to compute initial and final ESLOC values for all programs being analyzed. Though there is no inherent requirement for the growth rate to be a positive number, it typically is a positive number.

Productivity is the number of ESLOC that can be developed in an hour. The measurement is based on final developed ESLOC and final hours upon program completion. The measurement assumes the same EAFs are used for all programs being analyzed.

The equations needed to compute ESLOC and estimate the growth and productivity rates are given below.

$$EstESLOC = EAF_{New} * New\ SLOC + EAF_{Mod} * Modified\ SLOC + EAF_{Reuse} * Reuse\ SLOC$$

$$GrowthRate = \frac{ESLOC_{final}}{ESLOC_{initial}}$$

$$Productivity = \frac{ESLOC_{final}}{Hours_{final}}$$

The growth rate and productivity should be computed for each program in the data set being analyzed. To compute estimated hours required for a new program, the estimated ESLOC for a new program will be multiplied by the mean growth rate and the mean productivity as shown below. Keep in mind the growth rate and productivity variables are distributed random variables with a mean and variance and therefor these metrics introduce quantitative risk into the estimated hours.

$$EstHours = \frac{EstESLOC * E[GrowthRate]}{E[Productivity]}$$

EAFs have been around for a long time. One common method for developing EAFs is to determine the amount of code that needs to be designed, written, and tested, then multiplying these by the relative effort required for each type of effort (ie COCOMO method)[2]. For example, suppose it is determined that a software developer spends 40% of their time designing code, 30% writing code, and 30% testing code. New code requires all of this effort. Suppose the developer determines that 50% of the modified code requires designing and coding, and 100% requires testing. Then the ESLOC equation for modified code would be:

$$ModifiedESLOC = ModifiedCode * (.4 * \%Design + .3 * \%Write + .3 * \%Test)$$

$$= ModifiedCode * (.4 * .5 + .3 * .5 + .3 * 1)$$

$$= ModifiedCode * 0.65$$

2. Barry Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Bert Steece. *Software Cost Estimation with COCOMO II* (with CD-ROM). Englewood Cliffs, NJ:Prentice-Hall, 2000.

Once the equation is simplified, the resulting factor is a single factor and therefor this method reduces to the generic method described above. There is no standard way to create an EAF. One rule of thumb commonly used includes values of 1 for new code, 0.5 for modified code, and 0.08 for reused code. Regardless of the chosen EAFs, the subjectivity of the EAFs is a source of bias error in the cost model and this error is demonstrated below.

ESLOC Error

The first step in measuring the error is to define the data sets for new, modified, and reused code in order to generate growth and productivity metrics, as shown below.

$EstNewSLOC$, $EstModSLOC$, and $EstReuseSLOC$ are the engineering predictions for the required SLOC to complete the program. These values are plugged into the ESLOC equation above using the chosen set of EAFs and the resulting estimated ESLOC is computed and denoted $EstESLOC$. Recall the growth rate computation above will capture the estimating error in the estimated SLOC provided by the engineer, therefor the estimated ESLOC is treated as a constant. Substituting the productivity and growth rate equations into the $EstHours$ equation above gives the below simplification:

$$EstHours = \frac{EstESLOC * E[Growth]}{E[Productivity]}$$

Substituting and attempting to simplify the above equation is a non-trivial effort. For example, take the Growth term in the above equation: $[Growth] = E\left[\frac{ESLOC_{final}}{ESLOC_{initial}}\right]$. It can be shown that $E\left[\frac{ESLOC_{final}}{ESLOC_{initial}}\right] \neq \frac{E[ESLOC_{final}]}{E[ESLOC_{initial}]}$; at least not as a matter of fact. So any attempt to simplify the estimated hours equation becomes difficult right from the start because the expectation operator cannot be distributed to each term in the quotient. Therefore when trying to assess the error associated with the selection of EAFs, it's easier to look at a delta type equation as shown below, rather than looking to simplify the larger equation.

The error from using EAFs can be defined as $EAFerror = EstHours_l - EstHours_k$ where l and k represent two sets of EAFs. The right hand side of this equation will look as follows:

$$\frac{EstESLOC_l * E[Growth_l]}{E[Productivity_l]} - \frac{EstESLOC_k * E[Growth_k]}{E[Productivity_k]}$$

This equation again becomes difficult to work with, but with a few assumptions there are important inequality metrics that can be determined. Suppose all of the EAFs in set k are larger than the EAFs in set l, with the exception of EAF_{new} which is equal to one in both cases. Then all ESLOC calculations are necessarily larger, whether predicted or historical. This is because all of the SLOC terms are the same value and positive, and the factors being multiplied are larger, therefore each term is larger and the sum is larger. We can also say that productivity is higher, since the numerator, ESLOC, in the productivity calculation gets bigger while the denominator, hours, stays the same. We can summarize it as follows:

$$For\ EAF_l < EAF_k;\ ESLOC_l < ESLOC_k\ and\ Productivity_l < Productivity_k$$

But, since both ESLOC and Productivity are increasing, it's impossible to state whether or not $\frac{ESLOC}{Productivity}$ is getting larger or smaller since it depends on the size of the increase of ESLOC and productivity relative to each other. If ESLOC increases by more than productivity, then the ratio increases, but if productivity increases by more than ESLOC, then the opposite happens.

However, we can add an assumption to help further our progress. In typical cost estimating, the relative size and complexity of the historical program or project from which data is drawn should be similar to the thing we are trying to estimate. Let's say that estimated ESLOC is very similar in size to the mean historical ESLOC. More specifically that the new, modified, and reused SLOC components of the E[ESLOC] equation is very similar to the new, modified, and reused SLOC components of the EstESLOC. Then we know the estimated ESLOC term is likely increasing at a rate faster than productivity. This is because the estimated ESLOC term is increasing at a rate very similar to historic mean ESLOC. But final hours, which is the denominator in the productivity calculation is staying the same, therefore productivity itself is growing at a slower rate than estimated ESLOC. Therefore we can make the following statements given the assumptions outlined:

$$For\ EAF_l < EAF_k\ and\ EstESLOC\ \cong E[ESLOC]$$

$$EstESLOC_l < EstESLOC_k$$

$$E[Productivity_l] < E[Productivity_k]$$

$$\frac{EstESLOC_l}{E[Productivity_l]} < \frac{EstESLOC_k}{E[Productivity_k]}$$

However, to determine whether or not the delta equation above is positive or negative, we still must address the growth term. Recall that growth is measured by comparing the initial estimated ESLOC form a historic project to the final ESLOC that was developed. We know that when the (k) set is larger than the (l) set of EAFs, then initial ESLOC(k) is larger than initial ESLOC(l). We also know the same holds for the final ESLOC equations. However the question remains to whether or not the ratio of initial to final is getting bigger, which is to say whether or not final ESLOC is increasing at a rate faster than initial ESLOC.

To gain an understanding of the change in the growth rate relative to a change in the assumed EAFs, surface plots were created for each of the 33 programs studied with the modify EAF on the X1 axis, reuse EAF on the X2 axis, and growth on the Y axis. Again the new code EAF is left out of the analysis because it is deterministically equal to one in all cases. Below is an example of one of these surface plots. In this plot, a cord line has been drawn along the diagonal which represents the positively correlated values of the modify and reuse EAF. This cord line depicts the direction of the change in the growth rate when both EAFs increase or decrease at the same time. In order to assess the change in the growth rate relative to a change in the assumed EAFs, the point values along this cord line was collected for all programs and the slope of the line, relative to the modify EAF. See figure 1 for a sample plot. A bar chart showing these slopes is shown immediately to the right of the surface plot (figure 2).

Figure 1: Cord Line Sample Plot

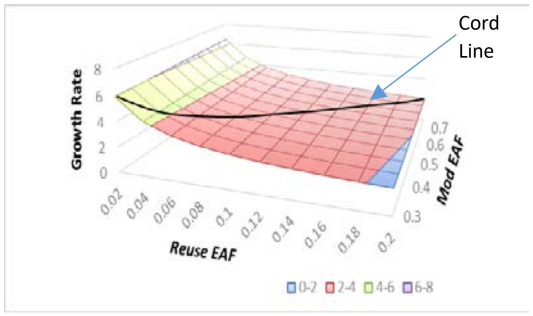
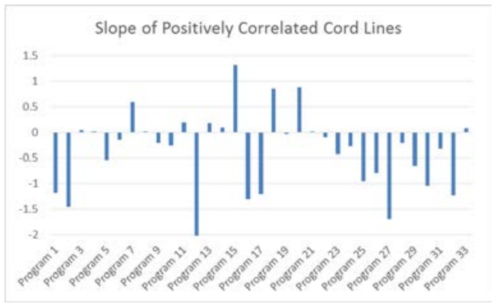


Figure 2: Cord Line Slopes



The purpose of this exercise is to demonstrate the difficulty in measuring the change in growth rates due to a change in the assumed EAFs. Nine of the 33 programs studied had a positive cord line slope, implying that when both the modify and reuse EAFs increases, the growth rate also increases. For these nine programs, when combined with the ESLOC and productivity statements above and the restrictive assumptions put in place, we can ascertain that the estimated hours will be overstated if the assumed EAFs are also overstated. For the remaining 24 programs, we cannot be sure whether or not the estimated hours will be overstated, understated, or will remain approximately the same due to an overstatement of the assumed EAFs. The issue of measurement error due to erroneously assumed EAFs is further complicated by the cases in which the modify and reuse EAFs are wrong in opposite directions, for example the modify EAF is overstated while the reuse EAF is understated. And all of this analysis relies on the assumption that the expected value of the final ESLOC in the historic data set is similar to and behaves similarly to the estimated ESLOC for the new program. Therefor the next step is to compare residuals of estimated hours across the programs in our data set.

To assess the quality of the residuals, five sets of EAFs were used to compute growth, productivity, estimated ESLOC, and, ultimately, estimated hours. The five sets of modify and reuse EAFs were (0.3, 0.02), (0.5, 0.1), (0.75, 0.2), (0.3, 0.2), and (0.75, 0.02). The new EAF was set to 1 as per usual. Growth rates and productivity were computed for each of the 33 programs, using each of the five sets of EAFs. Then estimated hours was computed for each program and each set of EAFs, using the average productivity and average growth rate across all programs relative to each set of EAFs. The result is a set of five predicted hours for each program which can then be compared to actual hours and residuals computed for each set. The below graph shows these residuals as a percentage of the actual hours. The candlestick pattern shows the highest and lowest residual factor at each end as well as the second highest and second lowest at the top and bottom of each bar. Red bars represent cases where the residual factor decreases as the EAFs increase and green bars represent cases where the residual factor increases as the EAFs increase. The yellow line is the zero line which is the case in which the predicted hours equals the actual hours and the residual is zero.

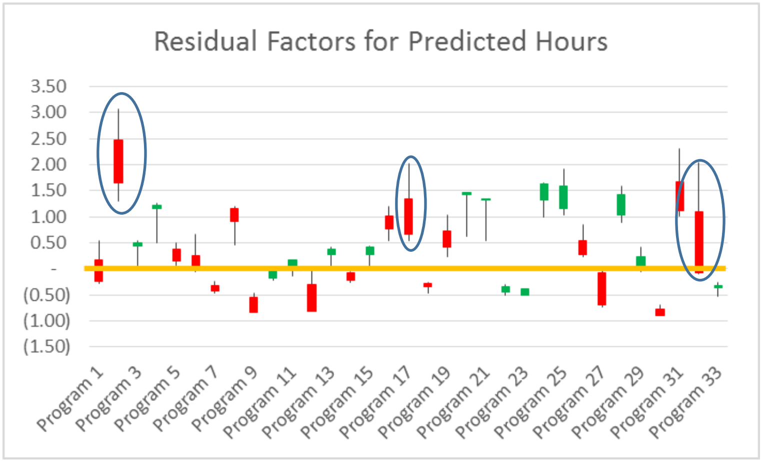


Figure 3: Residual Factors

In figure 3 above, a positive residual factor implies actual hours were larger than the residual hours and the program would have overrun. The y-axis denotes percent over/underrun when comparing actual to predicted values. A value of 1 implies a 100% overrun, which is to say the actual cost was twice as much as the predicted cost. Using this methodology 21 of 33 programs would likely have had an overrun and nine would likely have doubled their predicted hours.

Not only are there large residuals between the predicted hours and the final hours, but there is also large variance within programs across various EAF assumptions. Programs 2, 17, and 32, circled above, are examples of particularly large variance in the residual across various sets of EAFs. This sample data demonstrates that assuming a set of EAFs based on industry standards or rules of thumb often results in largely inaccurate predicted hours and relies on subjective factors.

The Regression Method

An alternative method, and one which is seemingly much simpler over traditional methods of EAFs, is to regress hours on the three types of SLOC in a multivariate linear regression. Regression modelling is one of the most common forms of data analysis in use for physical systems. Typically the variable of interest is a size type, such as weight. In software the SLOC is the size and SLOC is also arguably the number one cost driver. The proposed equation is as follows:

$$Hours = \beta_0 + \beta_1 * NewSLOC + \beta_2 * ModSLOC + \beta_3 * ReuseSLOC + \varepsilon$$

When performing the regression, it is important to use only final hours regressed on final code counts. This ensures only completed projects with actuals are included in the regression. DCARC identifies initial data as being estimates at the time of contract award and therefor these values would not be appropriate to include in a CER.

The regression method leaves the problem of addressing the growth rate. The data studied includes both initial and final hours. The initial hours reported are the hours bid or the estimated hours at the time of contract award. Since the proposed regression based CER is a direct measure of hours, the best way to capture growth is by directly comparing initial hours to final hours. This ignores the errors associated with converting from SLOC to ESLOC and focuses on the growth strictly from the initial proposed hours to the final hours reported. In this way, the growth rate becomes a distributed random variable which can be applied directly to the estimated hours for the new program. The interpretation of growth remains the same - it is still a measure of estimating error in a software program.

$$Growth = \gamma = E\left[\frac{Hours_{initial}}{Hours_{final}}\right] \sim N(\mu_H, \sigma_H^2)$$

$$FinalHours = \widehat{Hours} * \gamma$$

The multivariable linear regression results and gamma analysis are provided below in figures 4 and 5.

Model Form:	Weighted Linear model
Number of Observations Used:	33
Equation in Unit Space:	Hours = 2.255e+004 + 1.173 * New + 0.3617 * Mod + (-0.03106) * Reuse
Error Term:	MUPE (Minimum-Unbiased-Percentage Error)

Coefficient Statistics Summary

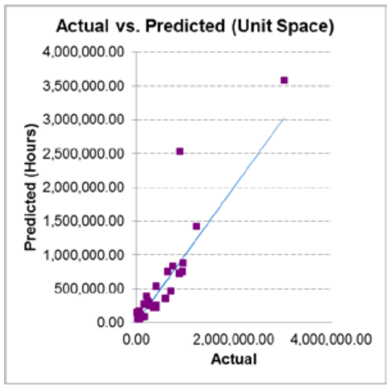
Variable	Coefficient	Std Dev of Coef	Beta Value	T-Statistic (Coef/SD)	P-Value	Prob Not Zero
Intercept	22547.7943	15474.2763		1.4571	0.1558	0.8442
New	1.1731	0.1573	0.8035	7.4574	0.0000	1.0000
Mod	0.3617	0.1877	0.2068	1.9270	0.0637	0.9363
Reuse	-0.0311	0.0190	-0.1787	-1.6376	0.1122	0.8878

Goodness-of-Fit Statistics

Std Error (SE)	R-Squared	R-Squared (Adj)	Pearson's Corr Coef
0.4806	68.98%	65.77%	0.8306

Analysis of Variance

Due To	DF	Sum of Sqr (SS)	Mean SQ = SS/DF	F-Stat	P-Value	Prob Not Zero
Regression	3	14.8991	4.9664	21.4987	0.0000	1.0000
Residual (Error)	29	6.6992	0.2310			
Total	32	21.5983				



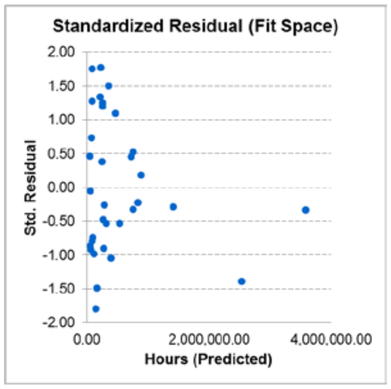


Figure 4: Regression Results

Growth Rate (Hours Basis)		
Number of Observations	33	
μ_γ	σ_γ^2	$C.V._\gamma$
1.3699	.4671	.34

Figure 5: Growth Rate Analysis

It is interesting to note that the coefficient on reuse code is negative 0.03. This implies the use of reuse code reduces total required hours by 0.03 hours per line of reuse code. The value has a greater than 88% probability that the true mean is not zero which also means there is a greater than 88% probability that the coefficient is a negative number. This is likely because the use of reuse code results in efficiencies throughout the software development process.

On a final note relating to the regression method, at its core this is not particularly different from the ESLOC method. The EAFs in the ESLOC method can be interpreted as normalized coefficients. If each coefficient in the regression equation were divided by the coefficient on new code, the resulting factors would be unit-less EAFs. This is inherently what the EAF is doing. The ESLOC method then adds hours back in by dividing by productivity, which is actually just the coefficient on new code. See below equations for proof.

$$\widehat{Hours} = \widehat{\beta_0} + \widehat{\beta_1} * NewSLOC + \widehat{\beta_2} * ModSLOC + \widehat{\beta_3} * ReuseSLOC$$

$$FinalHours = \gamma * \widehat{Hours} = \gamma * \left(\widehat{\beta_0} + \widehat{\beta_1} * NewSLOC + \widehat{\beta_2} * ModSLOC + \widehat{\beta_3} * ReuseSLOC\right)$$

$$\gamma * \widehat{Hours} * E[Productivity] = \gamma * \left(\widehat{\beta_0} + \widehat{\beta_1} * NewSLOC + \widehat{\beta_2} * ModSLOC + \widehat{\beta_3} * ReuseSLOC\right) * E[Productivity]$$

But $E[Productivity] = E\left[\frac{ESLOC_{final}}{Hours_{final}}\right] = E\left[\frac{NewCode}{Hours_{final}}\right] = \frac{1}{\widehat{\beta_1}}$. Recall that the purpose of the EAFs are to normalize code to represent the same effort required to develop a single line of new code. This provides the result shown. Therefor,

$$\gamma * \widehat{Hours} * E[Productivity] = \gamma * \frac{\left(\widehat{\beta_0} + \widehat{\beta_1} * NewSLOC + \widehat{\beta_2} * ModSLOC + \widehat{\beta_3} * ReuseSLOC\right)}{\widehat{\beta_1}}$$

$$= \gamma * \left(\frac{\widehat{\beta_0}}{\widehat{\beta_1}} + NewEAF * NewSLOC + ModEAF * ModSLOC + ReuseEAF * ReuseSLOC\right)$$

$$= \gamma * \left(\frac{\widehat{\beta_0}}{\widehat{\beta_1}} + EstESLOC\right)$$

$$\gamma * \widehat{Hours} = \frac{\gamma * \left(\frac{\widehat{\beta_0}}{\widehat{\beta_1}} + EstESLOC\right)}{E[Productivity]}$$

Finally, consider the case where the regression model is forced through the origin, as is the case with the ESLOC method. Then $\frac{\widehat{\beta_0}}{\widehat{\beta_1}} = 0$ and the desired result is achieved. This reinforces the idea that the regression method does not actually deviate from the ESLOC method, however it is objective in measuring the coefficients applied to new, modified, and reused SLOC as well as the sensitivity of the equation.

Summary

Software development has historically been difficult to estimate. Traditional EAF methods rely on effort adjustment factors which are inherently subjective. This results in unmeasurable bias error and can result in unnaturally large variance. GAO reports and the CEBoK highlight software estimating as being a driver of risk in modern programs. This study investigated the error associated with using EAFs leveraging data from 33 historic programs. The research shows that using a regression method rather than an EAF method removes a significant portion of bias error and subjectivity in the software cost estimate. The demonstrated regression shows promise because all of the parameters in the ANOVA are within estimating norms and statistically significant. The next step in this research is to expand the data set and begin to look at specific groups of data which are of common size, common coding languages, or families of software products make a difference to these results. Additionally an analysis of development methods, such as waterfall versus agile, should be pursued as more agile data becomes available.