Safe and Simple Software Cost Analysis
Barry Boehm, USC

“Everything should be as simple as possible, but no simpler.”
-Albert Einstein

Overview

There are a number of simple software cost analysis methods, but they may not always be safe. The simplest is to base your cost estimate on the typical costs or productivity rates of your previous projects. This will work well if your new project doesn’t have any cost-critical differences from your previous projects. But it won’t be safe if some critical cost-driver has changed for the worse.

Simple history-based software cost analysis methods would be safer if you could identify which cost driver factors were likely to cause critical cost differences, and if you could estimate how much cost difference it would make if a critical cost driver changes by a given degree. In this column, I’ll provide a safe and simple method for doing this by using the recently-published cost estimating relationships in the book Software Cost Estimation with COCOMO II (by Barry Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford Clark, Ellis Horowitz, Ray Madachy, Don Reifer, and Bert Steece, Prentice Hall, 2000). COCOMO II is an updated and recalibrated version of the Constructive Cost Model (COCOMO) originally published in Software Engineering Economics (by Barry Boehm, Prentice Hall, 1981). I’ll also show how the COCOMO II cost drivers can be used to perform cost sensitivity and tradeoff analyses, and discuss how similar methods can be used with other software cost estimation models.

COCOMO II Productivity Ranges

Figure 1 shows the relative productivity ranges (PR’s) of the major COCOMO II cost driver factors, as compared to those in the original COCOMO 81 model. The productivity range for each factor is the ratio of the project’s productivity for the best possible factor rating to the project’s productivity for the worst possible factor rating, assuming that the ratings for all of the other factors remain constant. Here, relative “productivity” is defined in either source lines of code (SLOC) or function points per person-month. The term “part” in Figure 1 reflects the fact that the COCOMO 81 development mode involved a combination of Development Flexibility and Precedentedness.

For example, suppose that your business software group has been developing similar applications for 10 years on mainframe computers, and their next application is to run on a micro-based client-server platform. In this case, the only cost driver factor that is likely to cause significant cost differences is the group’s “Platform experience” (assuming that such factors as “Platform volatility” and “Use of software tools” do not change much). From Figure 1, we see that the productivity range for the “Platform experience” factor is 1.40. This means that changing from the best level of platform experience (over 6 years) to the worst level (less than 2 months) will increase the amount of effort required for the project by a factor of 1.40, or by 40%.
Figure 1. COCOMO-81 and COCOMO II

Software productivity range

- Development flexibility
- Development mode (part)
- Team cohesion
- Developed for reuse
- Precededness
- Development mode (part)
- Architecture & risk resolution
- Platform experience
- Data base size
- Required development schedule
- Language & tools experience
- Language experience
- Process maturity
- Modern programming practices
- Storage constraint
- Platform volatility
- Use of software tools
- Applications experience
- Personnel continuity
- Documentation match to life cycle needs
- Multi-site development
- Turnaround time
- Required software reliability
- Time constraint
- Product complexity
- Personnel/team capability

*Varies by size; see Table 1. Values for 100 KSLOC products
If we compare the productivity ranges for platform experience between COCOMO II and COCOMO 81, they are fairly close (1.40 vs. 1.34). This is the case for most of the cost driver factors. In particular, we can see that personnel and team capability remains the single strongest influence on a software project’s productivity. Its productivity range in COCOMO II is somewhat smaller than in COCOMO 81 (3.53 vs. 4.18), but this may be because COCOMO II has added two more people-related factors (team cohesion and personnel continuity), which may account for some of the variance previously associated with personnel capability.

COCOMO II has some additional new cost drivers besides team cohesion and personnel continuity. These are: software developed for reuse; architecture and risk resolution; software process maturity (somewhat replacing use of modern programming practices); documentation match to life cycle needs; and multi-site development (somewhat replacing turnaround time). Each of these new factors has been determined to have a statistically significant greater-than-1.0 productivity range in the multiple regression analysis of the 161 projects in the COCOMO II database, with one exception. This was the “developed for reuse” factor, which had insufficient dispersion in its rating levels to produce a statistically significant result. Its productivity range of 1.31 (and the productivity ranges of the other COCOMO II factors) have been determined by a Bayesian weighted average of data-determined multiple regression results and expert-determined Delphi results. When the data-determined values are weakly determined, the weighted average will give a higher weight to the expert-determined values. A detailed discussion is provided in Chapter 4 of the COCOMO II book.

Some of the factors in Figure 1 have an asterisk, indicating that their productivity ranges vary by size, since they enter the COCOMO II estimation formula as exponential functions of size rather than as effort multipliers. The productivity ranges for these size-sensitive variables in Figure 1 are for a product whose size is 100,000 source lines of code (100 KSLOC). Table 1 shows how the productivity ranges for these factors vary by size.

<table>
<thead>
<tr>
<th>Table 1. Size-Dependent Productivity Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factor</td>
</tr>
<tr>
<td>Development Flexibility</td>
</tr>
<tr>
<td>Team cohesion</td>
</tr>
<tr>
<td>Precedentedness</td>
</tr>
<tr>
<td>Architecture and risk</td>
</tr>
<tr>
<td>resolution</td>
</tr>
<tr>
<td>Process maturity</td>
</tr>
</tbody>
</table>

Thus, for example, the effect of varying a project’s process maturity from a low Level 1 to a Level 5 on the SEI Capability Maturity Model scale will typically improve productivity by only 20% on a 10 KSLOC project, but it will improve productivity by 71% on a 1000 KSLOC project, mostly by reducing rework. These ranges probably under-estimate the effect of high maturity levels, as they normalize out the effects of their productivity factors such as the use of

Safe and Simple Software Cost Analysis

To return to our Platform Experience example, it may turn out that you can do better than a cost increase of 40% if you can find a few people with some microprocessor-based client-server platform experience to put on your project. For each cost driver factor, COCOMO II provides rating scales and tables showing how productivity will vary for each rating level. Table 2 shows the resulting effort multipliers by rating scale for each of the personnel-experience cost factors in COCOMO II.

**Table 2. COCOMO II Effort Multipliers vs. Length of Experience**

<table>
<thead>
<tr>
<th>Average Length of Experience</th>
<th>≤ 2mo</th>
<th>6mo</th>
<th>1year</th>
<th>3 years</th>
<th>≥ 6 years</th>
<th>PR</th>
</tr>
</thead>
<tbody>
<tr>
<td>With Platform</td>
<td>1.19</td>
<td>1.09</td>
<td>1.00</td>
<td>0.91</td>
<td>0.85</td>
<td>1.40</td>
</tr>
<tr>
<td>With Languages &amp; Tools</td>
<td>1.20</td>
<td>1.09</td>
<td>1.00</td>
<td>0.91</td>
<td>0.84</td>
<td>1.43</td>
</tr>
<tr>
<td>With Application Area</td>
<td>1.22</td>
<td>1.10</td>
<td>1.00</td>
<td>0.88</td>
<td>0.81</td>
<td>1.51</td>
</tr>
</tbody>
</table>

These tables can be used to perform the following safe and simple three-step cost estimation method, illustrated by the personnel-experience cost factors above.

Step 1. Estimate the new project’s productivity and effort based on previous experience. The new project is sized at 20,000 SLOC. Productivity for previous mainframe applications was 500 LOC/person month (PM), yielding an effort estimate of 40 PM. Multiply by your labor rate/PM to estimate the cost.

Step 2. Determine which COCOMO II cost driver factors will change for the new project and by how much. With the current staff, platform experience will go from ≥ 6 years to ≤ 2 months. By going to an Option A, with a couple of fairly experienced client-server developers added to the team, the average length of platform experience on the new project would be 6 months.

Step 3. Use the COCOMO II effort multipliers to revise the original estimate. Going from ≥ 6 years of platform experience to ≤ 2 months changes the effort multiplier from 0.85 to 1.19, an increase of a factor of 1.19/0.85 = 1.40. This leads to a revised effort estimate of 40 PM (1.40) = 56 PM. Going to a platform experience of 6 months (Option A) changes the effort multiplier from 0.85 to 1.09, an increase of a factor of 1.09/0.85 = 1.28. The revised effort estimate is then (40 PM) (1.28) = 51 PM.
The COCOMO II book has the full set of these cost driver effort multiplier tables on its inside cover for easy reference.

Simple Sensitivity and Tradeoff Analysis

This approach can be extended to cover sensitivity and tradeoff analysis among several cost driver factors. For example, suppose you also had an Option B to add a couple of extremely experienced client-server developers, increasing your average platform experience to 1 year. However, they have very little business applications experience, decreasing your average applications experience from ≥ 6 years to 3 years. In this case, the comparison to the previous Option A is:

<table>
<thead>
<tr>
<th>Effort Multipliers</th>
<th>Person - Months</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform x Application = Product</td>
<td></td>
</tr>
<tr>
<td>Option A</td>
<td>1.09</td>
</tr>
<tr>
<td>Option B</td>
<td>1.00</td>
</tr>
</tbody>
</table>

This indicates that Option A and B are roughly equivalent in effort and cost (unless there are major differences in salary levels), in which case the project can use other criteria to choose between A and B (e.g., the opportunity for client-server-expert mentoring and risk reduction with Option B).

As a further example, the comparison between Options A and B would be different if the new project involved a change in programming languages and tools (e.g., from COBOL to Java), causing the average experience for this factor also to be 6 months for Option A and 1 year for Option B. Now the comparison is:

<table>
<thead>
<tr>
<th>Effort Multipliers</th>
<th>Person - Months</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform x Application x Lang. &amp; Tools = Product</td>
<td></td>
</tr>
<tr>
<td>Option A</td>
<td>1.09</td>
</tr>
<tr>
<td>Option B</td>
<td>1.00</td>
</tr>
</tbody>
</table>

In this case, the added 9% benefit in Option B makes it look more attractive from a cost and effort standpoint.

Summary

Thus, we can see that the COCOMO II rating scales and effort multipliers provide a rich quantitative framework for exploring software project and organizational tradeoff and sensitivity analysis. The framework would enable the project manager to explore alternative staffing options involving various mixes of application, platform, and language & tool experience. Or, an organization-level manager could explore various options for transitioning a portfolio of applications from their current application/platform/language configuration to a desired new configuration (e.g., by using pilot projects to build up experience levels).
Note that these analyses can be done either by running the COCOMO II model, or by doing calculator- or spreadsheet-based analysis using the published multiplier values in Table 2 or in the COCOMO II book. Similar tradeoff analyses can be performed by running commercial COCOMO II tools or alternate proprietary cost models (see box for web sites of major software cost models).

It is generally a good practice to compare analysis results between two or perhaps more software cost models, as each reflects a somewhat different experience base. Some particular advantages of having one of these models be COCOMO II are that all of its detailed definitions and internals are available for examination; the Bayesian methods by which it combines expert judgement and data analysis are well defined; and the evidence of statistical significance of each cost driver factor is provided. From an expectations-management standpoint, however, there is no guarantee that COCOMO II can fit every organization’s style of development, data definitions, and set of operating assumptions. For example, it is only calibrated to organizations and projects that collect carefully defined data which can be mapped to COCOMO II’s definitions and assumptions. If your organization operates in a different mode than this, the COCOMO II-based analyses will likely provide useful relative guidance, but less precise cost estimates and payoff factors. And if your organization does collect carefully defined data, the analyses based on COCOMO II or other models will be stronger if you use the data to calibrate the models to your experience.

Box: Leading Software Cost Estimation Tools

Below are the URL’s providing descriptions of leading software cost estimation tools. Each of the tools cited has been backed by a lot of effort to relate the tool to a wide variety of software project experiences. My apologies if the list fails to include your favorite tool.

Knowledge PLAN (Software Productivity Research/Artemis). http://www.spr.com
SEER (Galorath, Inc.). http://www.galorath.com
SLIM (Quantitative Software Management). http://www.qsm.com

COCOMO II-based tools:
COSTAR (Softstar Systems). http://www.softstarsystems.com
CostXpert (Marotz, Inc.). http://www.costxpert.com
Estimate Professional (Software Productivity Center). http://spc.ca/products/estimate
USC. COCOMO II. 2000 (USC Center for Software Engineering). http://sunset.usc.edu/research/COCOMOII