Calculating for Reuse

By William H. Roetzheim

, Software Development

Dec 01, 2000
URL:http://www.ddj.com/architect/184414680

Several companies that I've worked with while at the CostXpert Group have increased their return of investment (ROI) by implementing formal software cost estimating throughout their organization. The increased cost savings and revenue enhancement are primarily the result of greater project success and increased control of project scope. Our three year ROI observations for client companies of the CostXpert Group has ranged from 29:1 to 37:1. These results are similar to results obtained independently by Capers Jones (see *Estimating Software Costs*, McGraw Hill, 1998); yet few companies and individuals really understand that software estimating isn't just art, but a crucial skill that can be learned. It really is possible to accurately and consistently estimate development costs and schedules for a wide range of projects. This series of articles provides you with the tools you need to understand step-by-step approaches to estimating the cost and schedule for your projects. Although there are a wide range of software cost estimating tools on the market to help with this process, this series focuses on understanding the fundamental concepts. You will be able to implement these concepts using nothing more complicated than a spreadsheet.

In the first article, "Estimating Software Costs" (Project & Process Management, Oct. 2000), I covered the various methods of estimating the size of a program (called program volume). I discussed the traditional measures of Lines of Code and Function Points, in addition to introducing other approaches. In the second article, "Project Cost Adjustments" (Project & Process Management, Nov. 2000), I discussed the concept of project cost adjustments for variations in the project environment. "Calculating for Reuse" builds on these concepts and explains how you can use them to quantify the impact of software reuse and commercial components and libraries on your estimate. Finally, the last article, "Creating the Project Plan," will describe how you can use your new insight into project cost and schedule to create a complete project plan.

Dealing with Code Reuse
The basis of code reuse is that code already exists that may be used in any given project. Reuse of code, objects or even entire applications is becoming de rigeur for most software projects. To accomplish this, begin by actually counting the values measured in the existing application(s). For example, you should use a code-counting utility to physically count the lines of code in the existing program modules if you're using Lines of Code. If you're using Function Points, count the existing reports, screens, tables and so on.

Calculating the Equivalent Volume
Your goal is to convert from the known value for the volume of reusable code to an equivalent volume of new code. Think about it this way:

- If you have 100 function points worth of code that is completely unusable, then you won't save any effort by reusing it, and the equivalent amount of new code equals 100 function points.
- If you have 100 function points worth of reusable code, and you can reuse it without any changes, re-testing or integration, then there is no cost to reusing the code, and the equivalent amount of new code equals 0 function points.
- If you have 100 function points worth of reusable code, but it only saves you half of the effort relative to new code, then the equivalent amount of new code equals 50 function points.

You convert reused volume values to equivalent new volume values by examining three factors: percent design modification, percent code modification and percent integration and testing.

Percent design modification measures how much design effort the reused code will require. Basically, a low percent value indicates high code reuse, whereas a high percent value indicates low code reuse and increases the requirement to develop new code:

- A value of 0 percent means that the reused code is perfectly designed for the new application, and you won't have to schedule time for design.
- A value of 100 percent tells you that the design is completely wrong, and the existing design won't save you any time.
- A value of 50 percent means that the design requires some changes and that the effort involved in making these changes is 50 percent of the effort of doing the design from scratch.

For typical software reuse, the percent design modification will vary from 10 to 25 percent.

Percent Code Modification measures how much the physical source code must be changed:

- A value of 0 percent means that the reused code is perfect for the new application, and the source code can be used without change.
- If the reused code was developed in a different language, and you need to port the code to your current language, the value would be 100 percent.
- Numbers in between 0 and 100 percent imply varying amounts of code reuse.

The percent code modification should always be at or higher than the percent design modification. As a rule of thumb, I've found the percent code modification is often twice the percent design modification.

Percent integration and testing measures how much integration and testing effort the reused code will require:

- A value of 0 percent would mean that you don't anticipate any integration or integration test effort.
- A value of 100 percent means that you plan to spend just as much time integrating and testing the code as you would if it was developed from scratch.

The percent integration and testing should always be at or higher than the percent code modification. I recommend that you set the percent integration and testing to at least twice the percent code modification. It's not unusual for this factor to be 100 percent, especially for mission-critical systems where the risk of failure is significant. For commercial off-the-shelf components (purchased libraries), where the percent design modification and percent code modification are often 0, it's not unusual to see a number of 50 percent here to allow for the integration effort and time spent testing the application with the commercial component.

Finally, after measuring your existing code's volume and estimating your percent design modification (DM), percent code modification (CM) and percent integration and testing (I & T), you would calculate the equivalent volume of new code (EV) as follows:

EV = AAF * Reused Volume = [(0.4 * DM) + (0.3 * CM) + 0.3 * I & T)] * Reused Volume

Suppose that you need to implement a new e-commerce system consisting of 15,000 source lines of code (SLOC). Let's ignore environmental adjustments for the moment. Using the formula from our first article, the predicted effort will be:

Productivity * KSLOC$^{\text{Penalty}}$ = 3.60 * 15$^{1.030}$ = 3.60 * 16.27 = Effort = 58.6 Person Months

Now, suppose that you identify a similar application that you can purchase with source code. You have 15,000 SLOC that you want to reuse. If you assume that the correct value for design modification is 25 percent, the correct value for code modification is 50 percent and the correct value for integration and test is 100 percent, what would be the new effort required?

First, calculate the equivalent source lines of code (ESLOC):

ESLOC = AAF * 15,000 = [(0.4 * 0.25) + (0.3 * 0.50) + (0.3 * 1.00)] 15,000 = 0.55 * 15,000 = 8,250

And the new calculated effort is now:

Productivity * $\text{KSLOC}^{\text{Penalty}} = 3.60 * 8.25^{1.030} = 3.60 * 8.79$ = Effort = 31.6 Person Months

There are three additional factors that need to be considered to complete the reuse picture: assessment and assimilation (AA), software understanding (SU) and unfamiliarity with software (UNFM). Assessment and assimilation indicates how much time and effort will be involved in testing, evaluating and documenting the screens and other parts of the program to see what can be reused. Values should range from 0 to 8 percent.

Software understanding estimates how difficult it will be to understand the code once you begin modifying it, and how conducive the software is to being understood. Is the code well-structured? Is there good correlation between the program and application? Is the code well-commented? A numeric entry should fall between 10 and 50 percent; the default is 30 percent.

Unfamiliarity with software indicates how much time your team has spent working with this reusable code before. Is this their first exposure to it, or is it very familiar? The range of possible values is between 0 and 100 percent; the default is 40 percent. These three factors add a form of tax to software reuse, compensating for the overhead effort associated with reusing code.

For projects where the amount of reuse is small-the adaption adjustment factor (AAF) is less than or equal to 50 percent-the following formula applies:

ESLOC = ASLOC {AA + AAF [1 + 0.02 (SU) (UNFM)]}

Let's take the earlier example involving 15,000 reused SLOC. Suppose you found that you could get by with 10 percent design changes, 20 percent code changes, and 40 percent integration and test effort. AAF would then be calculated as:

AAF = (0.4 * 0.1) + (0.3 * 0.2) + (0.3 * 0.4) = 0.22

Because AAF is less than or equal to 50 percent we can use the formula just presented. Now, suppose that assessment and assimilation was four percent, software understanding was 30 percent and unfamiliarity with software was 40 percent.

The equivalent source lines of code would now be:

ESLOC = 15,000 {0.04 + 0.22 [1 + 0.02 (0.3 * 0.4)]} = 15,000 [0.04 + 0.22] = 3,900

Using our earlier assumptions, the effort required to build this software would be:

Productivity * $\text{KSLOC}^{\text{Penalty}} = 3.60 * 3.9^{1.030} = 3.60 * 4.06$ = Effort = 14.6 Person Months

When reuse is low and AAF is greater than 50 percent changes, you would use this formula:

ESLOC = ASLOC [AA + AAF (SU) (UNFM)]

Let's work through the same example of 15,000 lines of reused code, but let's assume that the design modification was 50 percent, the code modification 100 percent, the integration and test was 100 percent and the correct values for assessment and assimilation, software understanding and unfamiliarity with software were 8 percent, 50 percent and 100 percent respectively.

AAF is now calculated as:

AAF = (0.4 * 0.5) + (0.3 * 1.0) + (0.3 * 1.0) = 0.80

Because AAF is over 50 percent, we use the second formula as follows:

ESLOC = 15,000 [0.08 + 0.80 (0.50 * 1.0)] = 15,000 * 1.2 = 18,000

Effort is now calculated as:

Productivity * $\text{KSLOC}^{\text{Penalty}} = 3.60 * 18^{1.030} = 3.60 * 19.6$ = Effort = 70.6 Person Months

In this case, reusing those 15,000 lines of code actually takes you 12 person months more effort than writing the same code from scratch!

In fact, this phenomenon is even more pronounced than shown in the example above. If you need 15,000 lines of new functionality, you will seldom find a reusable block of code that exactly matches the functionality you need. More often, the reused code will be significantly larger than the new code because it will do many unnecessary functions. Perhaps you will be reusing a piece of code that is 25,000 lines of code in size to get the 15,000 lines of code that you care about. Well, the entire 25,000 lines of code will typically need to be assessed, understood and tested to some degree. The end result is that somewhere between 15 and 30 percent of the design change is the crossing point beyond that, at which point you are better off rewriting the code from scratch. The correct value in this range will depend largely on how well matched the reused code is to your requirements and the quality of that code and documentation.

Dealing with Components

Another type of reuse involves reusing components or business objects where you will not be changing the design or the code. In this case, your testing of the reused components will vary between 0 and 100 percent, with numbers in the 20 to 50 percent range being typical. This means that AAF will always be below 50 percent in this case.

The correct formula for equivalent volume associated with incorporating these components into your application is thus:

ESLOC = ASLOC {AA + AAF [1 + 0.02 (SU * UNFM)]}

Suppose that we are reusing a component library that is 20,000 SLOC in size. Test effort is estimated at 25 percent, assessment and assimilation is 6 percent, software understanding is 10 percent and unfamiliarity with software is 100 percent). What will be the effort associated with this work?

Start by calculating AAF:

AAF = (0.4 * 0.0) + (0.3 * 0.0) + (0.3 * 0.25) = 0.07

ESLOC is calculated as follows:

ESLOC = 20,000 {0.06 + 0.07 [1 + 0.02 (0.10 * 1.0)]} = 20,000 (0.06 + 0.07) = 2,600

And effort to integrate this library would be calculated as:

Productivity * $\text{KSLOC}^{\text{Penalty}}$ = 3.60 * $2.6^{1.030}$ = 3.60 * 2.7 = Effort = 9.7 Person Months

Of course, a typical application consists of a combination of new development, reused code and off-the-shelf components and libraries. In this case, you simply calculate the equivalent SLOC for all portions of the application, then plug this total number into the effort equation to get a total effort in person months.

See you next month when I show you how to use your new-found insight into project cost and schedule to create a complete project plan.