QSM ®

Quantitative Software Management, Phoenix Office
5013 West Vogel Avenue
Glendale, Arizona 85302
(602) 435-9863 (602) 915-3351 FAX
102121.434@compuserve.com or Michael_A_Ross@msn.com

Quoting Small Changes During Software Development with SLIM 4.0 ®

How to Quote Small In-Process Modifications to Software

by Michael A. Ross

Quoting Small Changes During Software Development with SLIM 4.0 ®

How to Quote Small In-Process Modifications to Software

Introduction

Purpose

The purpose of this paper is to describe a process that can be used to estimate the cost and schedule impact that result from a small change to the requirements baseline during software development independent of the project's current completion status (i.e., use of SLIM-Control ® is not required). A small change, within the context of this paper, is defined as a change which both the supplier and the customer agree does not require a complete re-plan of the project. Rather, this change is assumed to be small enough that it can be handled within the project's current committed-to constraints. This process must have the following characteristics:

- The resultant cost, schedule, reliability, and risk impact must be deterministic; i.e., given set of quantified assumptions and constraints, both the supplier and the customer must be able to, in isolation, generate the same result.
- The resultant cost, schedule, reliability, and risk impact must be consistent with the demonstrated capabilities (historical data) of the supplier organization; i.e., it must be within reason for the supplier to be successful given the additional burden.

Scope

This paper describes a process that applies to small changes in software under development from the start of system requirements definition to full-up delivery. This paper assumes a clear distinction between changes and defects. The process described herein applies only to changes. The cost of rework associated with defects is accounted for in the organization's process productivity (QSM Productivity Index).

Background

Historically, it has been very difficult to estimate the cost, schedule, reliability, and risk impact associated with a change that comes in after software development has begun and prior to delivery. Some organizations merely "eat" the changes. Some organizations have tried various cost per line of code metrics to compute cost and assert pricing changes. Other organizations have developed and negotiated the use of tables or matrices that relate the scope of a change to the resultant cost. Still others wait until the project is complete and then enter into a fact-finding exercise with the customer to determine how much was actually spent developing the product.

A further complication to quoting changes arises when the change is relatively large and/or occurs late in the development cycle. Many a project has gotten in serious trouble by accepting a large or late change within its original constraints; the change, subsequently, causing one or more commitments to be broken.

The following assertions, for the purpose of this paper, are assumed to be axiomatic:

- Changes include either or both of new work and rework.
- The function relating cost, size, and time is nonlinear.[1]
- The function relating cost, size, and total defects is nonlinear.[1]

Based on these assertions, we can infer that each change will alter the size-time-effort-reliability solution for the project. Further, we can infer that this impact increases as a function of how much the development has progressed. QSM strongly recommends that if a proposed change impacts the probability of achieving any committed-to project constraint, then the project should be re-planned.

Sizing the Change

Work or Functionality?

There is currently much confusion in the software project management world about what "software size" represents. When we measure software size are we attempting to quantify **work** or **functionality**? It is important that we recognize the difference.

Consider the following simplistic example: Two engineers are given a specification to create a software module that inputs a divisor and a dividend and outputs the corresponding quotient. The target processor is the same for both instances and contains no divide instruction; however, one engineer must implement the module using assembly language while the other must use a high order language. It is not unreasonable to conclude that the assembly language engineer would end up writing, say, fifty statements while the high order language engineer would end up writing, say, five statements; a significant difference. Assuming that the high order language compiler is reasonably optimized, the number of resulting words will be nearly the same for both implementations.

The point this example tries to make is:

- Assuming one statement in either language requires approximately the same amount of brain power, the size of each module measured in statements represents the amount of work that was done by each engineer.
- The size of each module measured in words represents the amount of functionality provided by the resultant module.

[1] Based on QSM's historical data base of over 4000 completed projects of various application types and sizes.

- There is not a one-to-one correspondence between work and functionality.[2]

Whether to measure work or functionality depends on the goal. If the idea is to measure the value produced by an instance of the software development process, then functionality is the answer since it is functionality that the customer is buying. If, on the other hand, the idea is to measure the capability (i.e., throughput, horsepower, productivity, etc.) of an organization's development process, then work is the answer. Since the purpose of this paper is to provide a process for estimating cost and since cost is directly related to work (effort) and not necessarily to functionality, this paper will henceforth assume that size and work are synonymous.

Units of Size

There are many units of measure currently being used to quantify software size. They range from words of physical memory on one end of the spectrum to bang (requirements modeling entities) on the other end and including such measures as units, modules, CSCs, function points, and feature points. The most common unit of measure, however, is the Effective Source Line of Code (ESLOC). There are two major reasons for the popularity of ESLOC as a way to quantify size:

- ESLOC measures size as work and thus provides a key parameter in most software cost estimation equations.
- Given a set of counting rules, total ESLOC for completed projects is deterministic and the counting process lends itself to reasonably easy automation.

Counting ESLOC

Over the years, there has been much effort devoted to standardizing the counting of SLOC and ESLOC. The Software Engineering Institute has published a paper containing a set of counting rules that are becoming a defacto standard. Adjustments are commonly added to account for modified and reused code. The important point here is not the details of the counting rules but rather that an organization consistently uses the same set of rules and, ideally, that these rules are embedded in some sort of automated counting tool.

Estimating ESLOC

Counting ESLOC in a finished product is one thing; estimating ESLOC in a yet-to-be-developed product is quite another. SLIM 4.0 supports several sizing techniques which may be used either in parallel or in combination. Regardless of which method is used, for sizing a small change, the result should be the number of logical source lines of code that will be added or changed.

[2] This is the primary motivation for using high order languages; more bang for the buck.

Computing the Effort, Cost, and Schedule with SLIM 4.0

Once an estimate for the size of the change has been established, invoke SLIM 4.0 with the command line switches /relaxsize and /relaxmbi. Select the Estimation mode. Under the Options pull-down menu, enter the appropriate values in the Reliability and Accounting dialog boxes (specifically the appropriate values for Total Defect Tuning Factor and Labor Rate). Note that in order to facilitate repeatability of the quotation process, these values must be documented as part of the basis of the quotation. Next, select the Project Assumptions dialog box (see Figure 1).

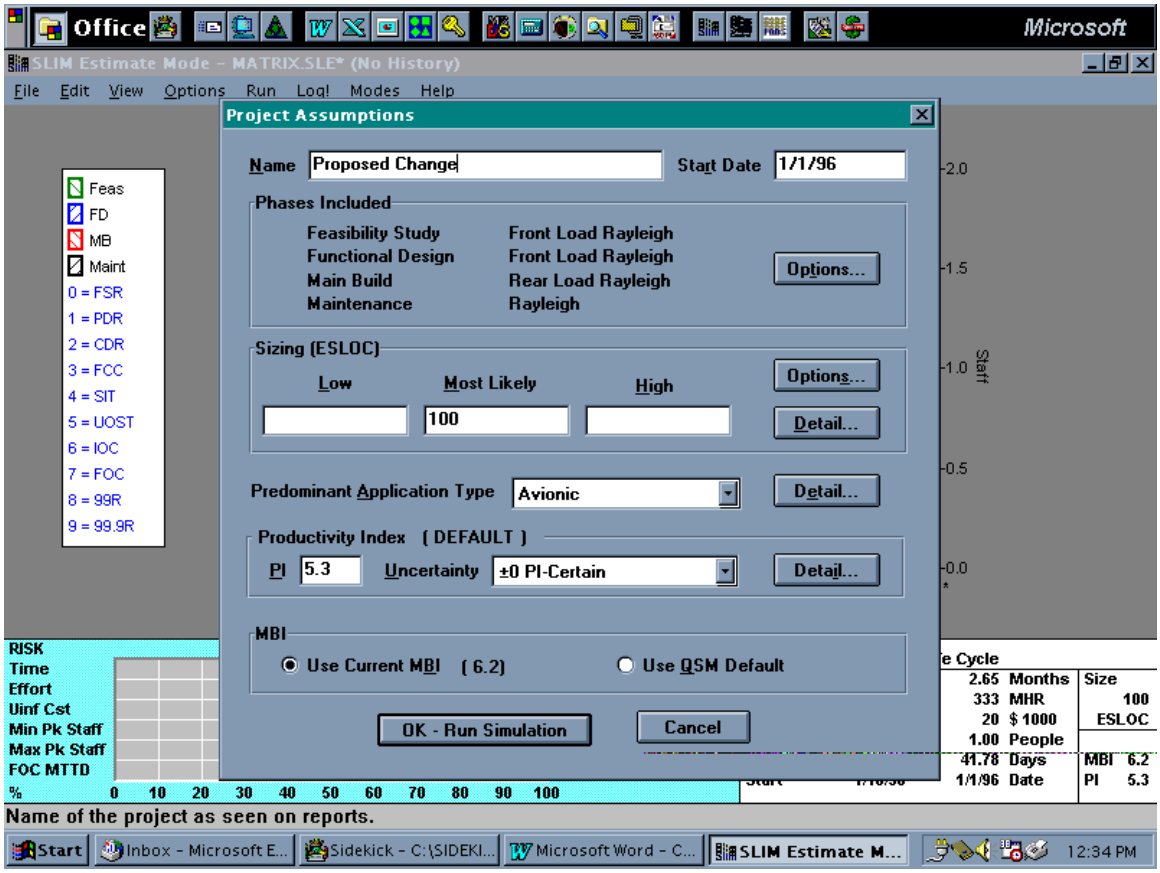

Figure 1 Project AssumptionsDialog Box

Project Assumptions

In the Project Assumptions dialog box, enter the following information:

1) Name: Enter the name of the change.

2) Start Date: Enter the date that work is expected to begin on the change.

3) Phases Included: Select and enter the same phases and phase options as were used in the baseline project estimate.

4) Sizing: Enter the estimated size of the change. If the constituent Options… and Detail dialog boxes have been used then a value will already appear here that represents the result of the chosen sizing methodologies. The size should include new code and modified code. Use actual sizes; do not apply any adjustment factors or percentages.

5) Predominant Application Type: Enter the application type of the change. This will generally be the same as the application type used in the baseline project estimate.

6) Productivity Index: Select Detail… (see Figure 2).

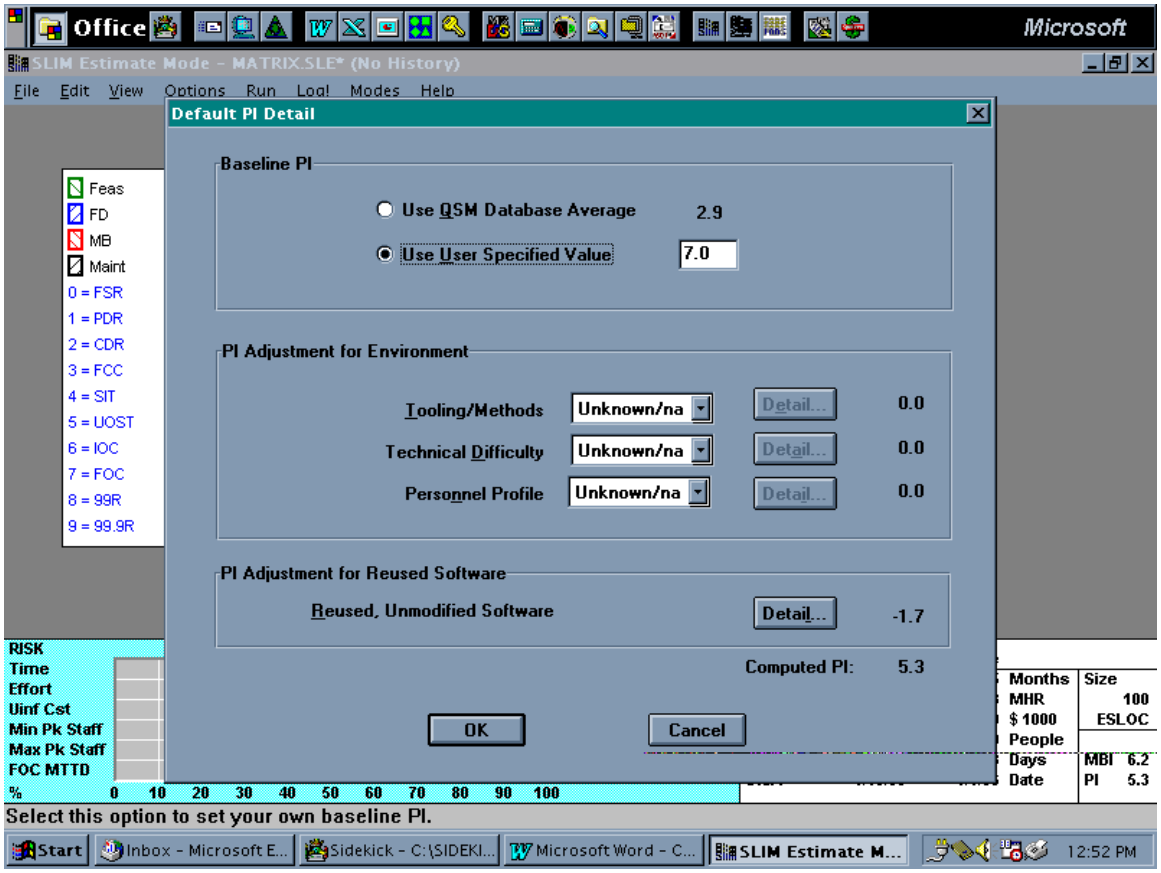

Figure 2 Default PI Detail Dialog Box

7) Baseline PI: Select User Specified Value and enter the PI that was assumed in the baseline project estimate.

8) PI Adjustment for Environment: Select the same values and Detail… as was assumed in the baseline project estimate.

9) PI Adjustment for Reused Software: Select Detail… (see Figure 3).

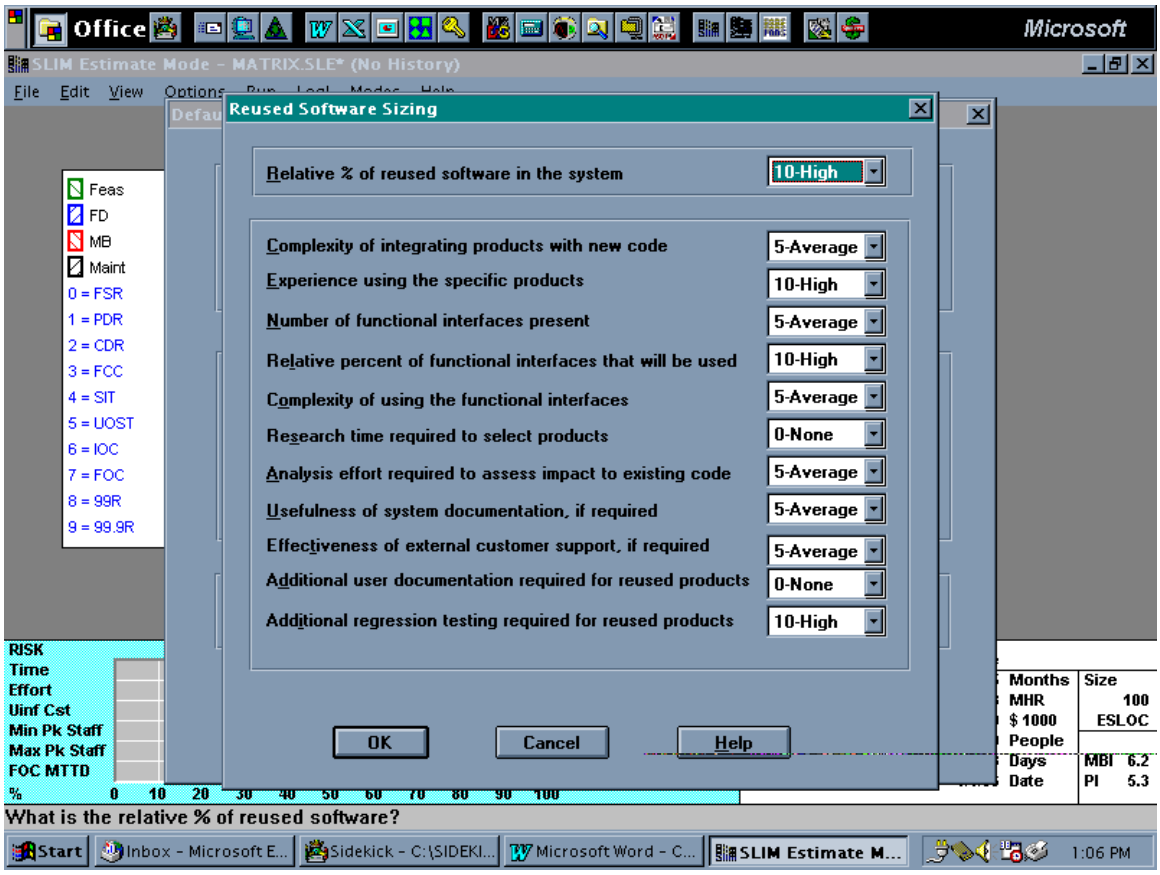

Figure 3 Reused Software Sizing Dialog Box

10) Reused Software Sizing: Select settings that most-closely characterize the maturity of the system being changed. Note that within this context "system being changed" and "reused system/products" are synonymous. Note that in order to facilitate repeatability of the quotation process, these values must be documented as part of the basis of the quotation.

11) Return to Project Assumptions Dialog Box: OK out of Reused Software Sizing and Default PI Detail.

Run the simulation by selecting OK - Run Simulation. Note that for very small changes the simulation may have to be run initially with a dummy large size in order to have access to the Staffing view so that a sufficiently high MBI can be induced and then the simulation run again with the correct size.

Project Constraints

If a risk assessment for the proposed change is desired, then complete the Project Constraints dialog box with the appropriate values and desired probabilities. Generally a risk assessment is not necessary unless the change is fairly large and/or occurs late in the development cycle.

Effort and Time Tradeoff

Select the Staffing view (see Figure 4).

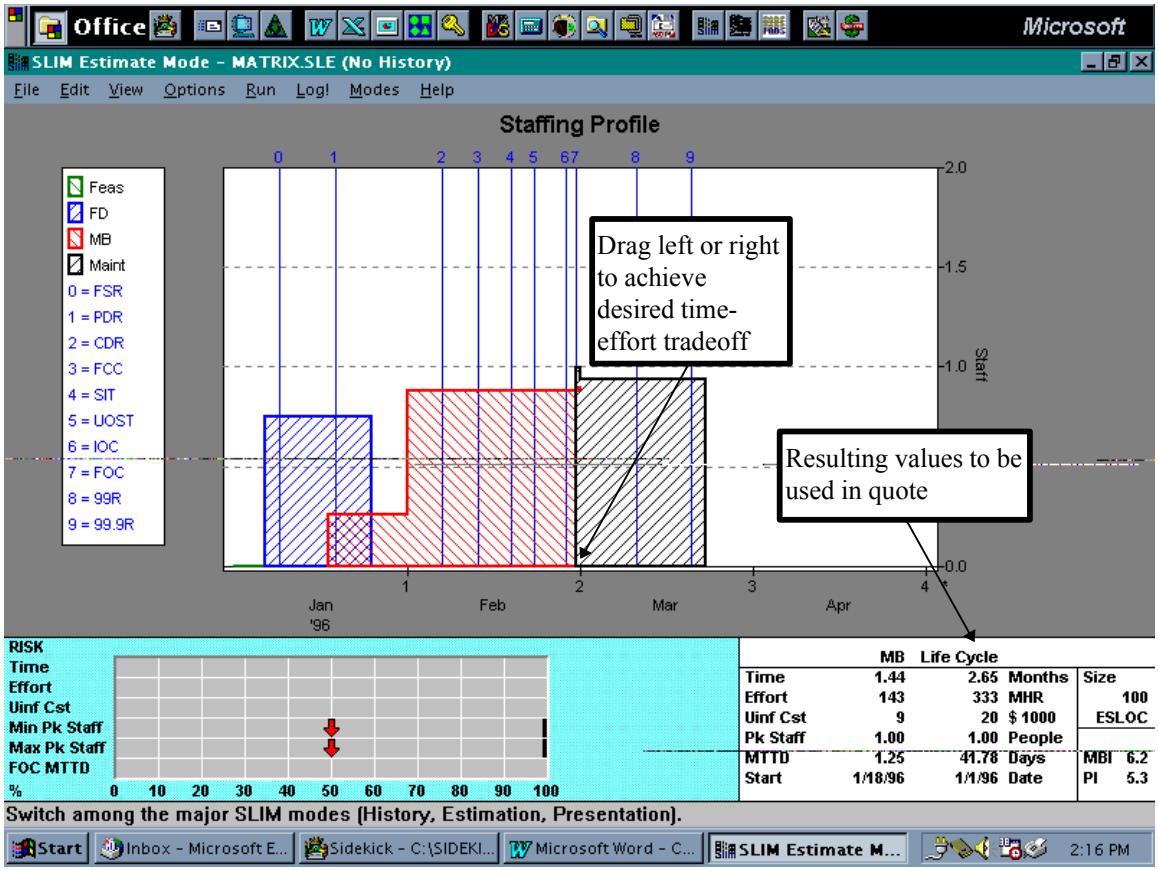

Figure 4 Staffing View

Drag the Main Build schedule left or right as necessary in order to satisfy all known constraints associated with the change. The resulting values in theLife Cycle column of the metrics box in the lower right corner of the Staffing view represent the numbers that should be used in the quote.

Conclusion

This paper documents a fairly quick, simple, and deterministic method for computing the management numbers associated with quoting small software changes during the development cycle. Keys for successful use of this process include:

- Limit the use of this process to small changes.
- Avoid using this process for changes that occur near the end of the development cycle.

- A change that increases the risk of meeting a commitment to the customer should trigger a project re-plan.

- Do not quote a group of changes as the sum of quotes developed by this process for each individual change (i.e., changes are not additive).

- Document all assumptions input into SLIM as part of the basis of the quotation and communicate this information to all parties concerned. This may involve a negotiation between the supplier and the customer to establish standardized assumption values.