Adapting software project estimation to the reality of changing development technologies

Introduction

Estimating software projects where significant amounts of new technology are being used is a difficult and very risky undertaking. In this article we will discuss how to use a macro-estimation technique (SLIM estimation method) to create reasonable estimates that adequately reflect risks and uncertainties.

Change, Change and More Change

Lets face it, we live in world of change. Product development cycles are getting shorter. New tools and development methods are being adopted almost as fast. In the midst of all this change, development managers, product planners and purchasers need to be able to estimate cost, schedules, resources and product quality with investment quality precision.

This is a difficult task but certainly not impossible. When confronted with this job people often feel overwhelmed and find it difficult to step back to objectively evaluate what is really going to be different in the new development paradigm.

Getting Down to Basics

To estimate new technology projects from a macro-estimation perspective there are 2 key variables that we need to get a handle on. The variables are size and efficiency. Lets examine these variables more closely.

Software Size: The first important variable is software size. With the size variable we are simply trying to get a feel for how much product we are going to have to develop.

In the construction industry size is typically measured in square footage of the structure. The square footage gives builders a rough feel for the degree of engineering rigor and discipline required for the job as well as quantity of materials. There is a big difference between building a single family house and a forty story office building. Similarly in software there is a big difference between building a 25,000 SLOC (source line of code) graphics package and a 1,500,000 SLOC CAD/CAM system.

In software there are a number of different ways that the size can be measured. Traditional size measures have been source lines of code and function points. With the introduction of object oriented methods and visual development tools new measures of size might be more convenient to use. The case study presented in this paper will highlight some practical procedures for sizing in these new environments.

Development Efficiency: There are a significant number of factors that affect efficiency of a software development environment. The broad categories are:

- People (skill, motivation, team dynamics, functional knowledge, management, leadership, etc.)
- Tools (operating systems, compilers, planning systems, configuration management tools, data management tools, documentation tools, test tools, etc.)
- Technical complexity (application domain, quantity of new algorithms and logic, processing constraints, memory constraints, platform stability, number of development sites involved, etc.)
- Complexity of code reuse (Complexity of integrating and testing old code with new code, research time required to determine usefulness, experience using the existing software, usefulness of system documentation and customer support, etc.)

We recognize that as we move into new environments that project factors will be significantly different. However, there will be some cultural behavior that will remain essentially the same.

People often feel like their historic data are "no good" anymore, but I always like to look at data from past projects. There are at least two good reasons for looking at past history.

First of all, recent historic data provides a profile of where an organization has been. One can determine the average productivity along with the statistical variation. By looking at the outliers one can determine the productivity impact of unique project factors. The history provides a starting point for us to judge the impact of the new technologies.

Secondly, I find that organizations are always going through periods of significant change. If it is possible to identify technology transition points in the past, then one can learn how this particular organization has been able to assimilate technology. This information can be then be applied to the new situation. Is this approach perfect? - Absolutely not. Is it useful? Without a doubt. Remember, the goal is to reduce risk and eliminate being wrong by a factor of 10. Any meaningful information that we can bring to bear on this situation helps enormously.

Case Study: On a recent consulting engagement we were asked to help a client estimate a billing application. The developer was using the Smalltalk object-oriented (OO) environment. This environment was totally new for them. Our client was being asked by their customer to produce a most likely and a worse case estimate.

Our assignment was to:

1. Estimate the size of the application.

2. Determine a reasonable efficiency level given the complexities of the new environment and our understanding of the technical challenges.

3. Build a baseline estimate and work plans.

4. Build a worse case estimate.

Size Estimate: The first challenge was to estimate the size of the system. My experience has shown that sizing information is either contained in design documents or inside the mind of a technical guru. My technique to get at the sizing information is simple: ask questions and be a good listener. I typically ask for a technical briefing on the system to help facilitate my information gathering. During the briefing I typically ask questions that help me understand what they have to build and easy ways for them to quantify the functionality they must create. Some typical high level probing questions are listed below.

- How many pieces (subsystems - modules) are there likely to be in the system?

- Which pieces will have to be newly developed, modified, reused internally with no change, or purchased?

- What languages are going to be used? How much experience do they have with this language and operating system environment(s)?

- What are the primary construction units that the software engineers will be building (objects, screens, modules, reports application views, etc.)?

- Has any proto-type construction been done using the new development paradigm. If yes, do they have any basic statistics related to size, time and effort?

As we discussed the architecture of the billing application we identified the following subsystems.

User Interface
Business Model
Access Relational Mapper
Access Transaction Manager
Data Management Oracle
Data Management Versant
Invoice - Reports
Use Cases
Use Case Framework
Other Classes

Table 1. Subsystems to be built.

After some discussion it became clear that some development work had already begun. We collected the following statistics on the completed software.

1. The number of Classes (objects)

2. The number of Methods

3. The number of Smalltalk source lines of code

Table 2 shows a summary of the statistics. In total 17,313 Smalltalk source lines of code had been written. The average Smalltalk SLOC per class was 100.66. The SLOC per class for individual sub-systems ranged from 58 to 188.

Sub-system	SLOC	Classes	Avg SLOC/Class
Business Model	4312	74	58.27
User Interface	3200	17	188.24
Use Case Framework	1893	13	145.62
Use Cases	6585	54	121.94
Other Classes	1323	14	94.50
Total	17313	172	
Average SLOC/CLASS			100.66
Small Talk Class Library	107556	911	118.06

Table 2 Sizing Statistics on completed work.

The average size of Smalltalk classes were significantly smaller than what I had seen for object oriented C++ applications. The C++ applications have averaged around 250 SLOC per class. So I was looking for some information to verify that the average class size of 100.66 SLOC was reasonable.

We turned to the Smalltalk class library. These are reusable classes that are provided as part of the Smalltalk development environment. There was a total of 107,566 Smalltalk SLOC and 911 classes. The average size of a class is 118.06. We felt comfortable with our statistics but were a little concerned that the average size might grow 5% to 10% as objects mature through the development process.

To estimate the size for the rest of the system we needed to estimate the total number of objects for each of the major subsystems.

The data management and reporting subsystems were going to be built using visual programming tools. Somehow we needed to find a reasonable way to size tables and reports. We decided to use visual programming primitives as our size measure.

The approach for determining a primitive is:

1. Identify the product construction elements (what are we constructing? In this case it was Tables and Reports.

2. Identify the specific programming units for each construction element. (the smallest units of work required to build the construction element - placing fields on reports and setting properties etc.)

3. Determine the typical number of programming units for a simple, average and complex construction element.

4. Build an algorithm to transform construction elements into primitives.

5. Estimate the number of primitives to implement the functionality.

Performing this process for tables was quite simple. Tables are comprised of columns. For this system we figured a simple tables required 2 columns, an average table requires 5 columns and a complex table would require up to 10 columns. The algorithm for determining table primitives is:

Table Primitive Complexity Matrix and Estimation Algorithm

Programming Units	Simple	Average	Complex
Columns	2	5	10

Table Primitives = (# of Simple tables * 2) + (# of Average Tables * 5)+(# of Complex Tables * 10)

Table 3. Table primitive table

Determining the report primitives was more involved. Report creation requires the programmer to identify specific tables and fields within the tables that will be used in the report. To build the report the developer must place fields on the report and set specific properties for each data field. Some reports may have multiple sections and some unique "behind the scene" source code. Table 4 shows our mapping for simple average and complex reports.

Report Primitive Complexity Matrix and Estimation Algorithm

Programming Units	Simple	Average	Complex
Tables	1	5	10
Fields	6	10	25
Properties	6	6	6
Sections	3	5	8
Unique code	0	0	100
Report Primitives	45	115	508

Report Primitive Definition (number of tables * fields) + (number of fields * properties) + number of sections + lines of code

Report Prim1tive = (# of Simple reports * 45) + (# of Average Reports * 115)+(# of Complex Reports * 508)

Table 4. Report primitive table.

Table 5 is the best estimate of the size at completion. The table shows a low, most likely and a high estimate for the number of objects, tables and reports in each of the subsystems. The column ESLOC/component are the transforms that convert our estimates into Smalltalk source code. The expected size of the system is 26, 899 SLOC but it could range from 24,000 to 30,000 when uncertainty is taken into account.

Overall Size Estimate

GUI Sizing

			Components		
Component Name	Included	ESLOC/Component	Low	Most Likely	High
User Interface	Yes	188.0	28	32	38
Business Model	Yes	58.0	74	92	105
Access Rel Mapper	No	121.0	10	25	30
Access Trans Mgr	Yes	121.0	1	1	2
DM Oracle	Yes	5.0	53	106	289
DM Versant	Yes	7.0	3	10	15
Invoice Reports	Yes	115.0	12	18	27
Use Cases	Yes	121.0	54	75	100
Use Case Framework	Yes	146.0	12	13	14
Other Classes	Yes	94.0	14	16	18
Total ESLOC			23644	26899	30153

BS12

Table 5. Overall size estimate

Productivity Assumptions: In an ideal case one would like to have some basic historic data on size, time and effort from some completed projects. However, in this case, my client was a startup company and this was their first set of products. There was no historic data!

The SLIM estimation method requires an input for productivity in the form of an index which is called a productivity index (PI). The scale ranges from 1 to 40. Higher values denote a more efficient environment and / or a simpler application domain.

Our approach for determining the productivity index was to assess 5 general categories at a fairly high level of abstraction. The assessment categories were: Application domain, tools and methods capability, technical complexity, personnel capability, and code reuse complexity.

The application code was 60% business information system, 30% layered software and 10% scientific. The application complexity category resulted in a starting Productivity Index of 15.

Tools and methods were assessed on a scale of 1 to 10 where 1 means tools are primitive or non-existent and methods are immature or non-existent. This category was assessed at a conservative 2, based on unfamiliarity with the tools and the absence of formal methods. This rating reduced the productivity index by 1.7 .

Technical complexity was assessed on a scale of 1 to 10 were 1 is simple and 10 is extremely challenging technically. This category was assessed at an conservative 8, based on the newness of the product line, client server complexities, completely new design and new logic. This rating reduced the productivity index by 1.7.

Personnel was assess on a scale of 1 to 10 were 1 is inexperienced, unskilled people with no functional knowledge of the application domain and 10 is the other end of the spectrum. This category was conservatively assessed as a 6. It probably could have been assessed as an 8 or 9 but it is the first time this team has worked together so we down graded the assessment to a 6. This rating increased the productivity index by .7.

Complexity of reusing code was assessed by answering a series of questions related to how much of the total system will be commercial packages and unmodified software, experience using the unmodified code in building systems, number and complexity of interfaces to the unmodified software, testing rigor required and usefulness of documentation and customer support. For this application there were two sources of reused code. The first was the class library that was supplied as part of the Smalltalk environment. The second was a product that they planned to procure to do data transformation. Based on our assessment of the reuse factors the productivity index was decreased by .9.

The aggregate of these adjustments produced an overall productivity assessment of 11.5. This assessment is 3.5 values lower than our starting point of 15. The assessment summary is shown in Figure 1 below.

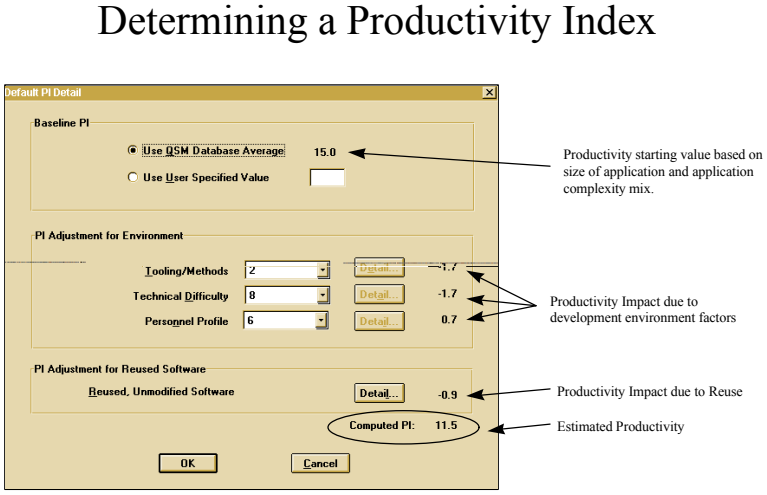

Figure 1. Productivity Assessment Selections

A productivity index which is 3.5 below the starting point is conservative. However, the developer wanted a workable plan that was completely realistic. They felt that it was better to set realistic expectations that were achievable so they could deliver on their commitments.

Base Estimate: The developer needed to keep the staffing under 8 people at peak loading. The current staffing was at 6 people. The customers goals were to have a system in place by September 15, 1996 . The product needed to be able to get through a 8 hour billing cycle without encountering any critical, serious or moderate defects.

Based on a size of 26,899 Smalltalk and a productivity index of 11.5 and subject to the staffing, schedule and reliability goals we produced our initial estimate. It is shown in Figure 2 in the form of a project staffing profile. This estimate could satisfy all of the criteria except the schedule. The schedule was about 2 weeks over the desired date of September 15, 1996. The effort estimate is 11,146 person hours and the reliability estimate is 2 days average time to discover a defect at the delivery date.

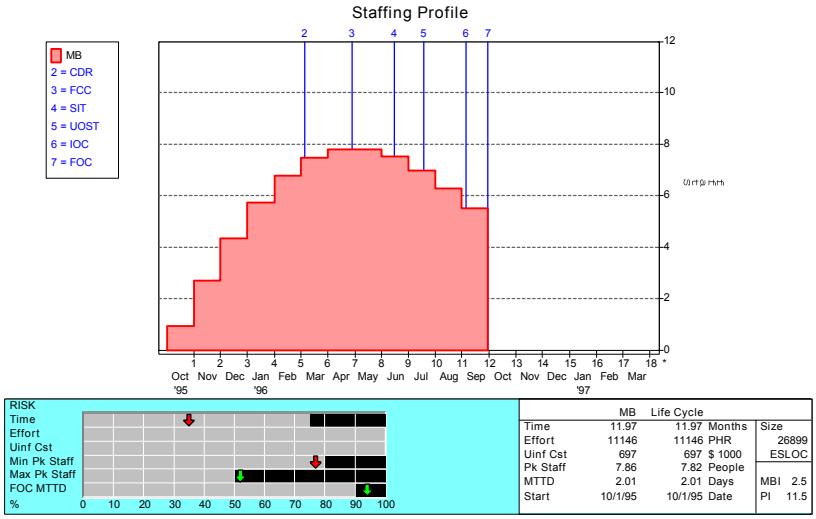

Figure 2. Base estimate of the project with the peak staffing constraint of 8 people.

We performed a probability analysis on the schedule. The probability analysis used simulated variations from our size estimates and productivity assumptions. The probability analysis showed that there was only about a 35% probability of meeting the September 15 date. If we could negotiate a two month extension to November 15 then the probability would increase to 90%. The schedule probability analysis is shown in Figure 3.

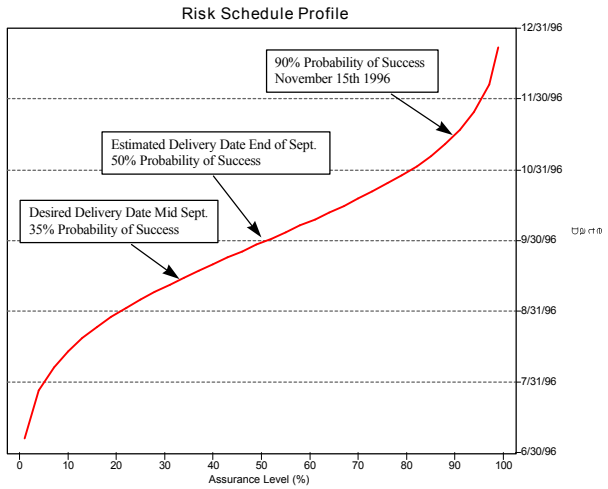

Figure 3 Schedule probability analysis for our most likely estimate.

Assess work to date against plan to verify estimation assumptions: Estimation is often performed as a one time exercise but, there are good reasons to re-estimate throughout the development. First, re-estimation allows one to make adaptive forecast when the nature of the project changes (growth in requirement etc.), and second, it allows one to confirm the assumptions the estimate is based.

This practice of confirming assumptions is especially important in estimates of new technology projects because, you have the most uncertainty about your size and productivity assumptions. You want to quickly identify poor assumptions and make practical adjustments early in the projects when you have the most leverage to implement positive changes or re-negotiate with your customers.

In this instance, we wanted to confirm that the project team was performing at or better than the productivity index of 11.5. To confirm our assumptions we used the project plans for staffing, schedule, effort and integrated code.

It is important to establish some control bounds around the baseline project plan. We use 3 control bound regions. The Green region is normal acceptable variation, amber is marginally acceptable performance (to be watched carefully) and the red zone is unacceptable performance requiring action.

If our estimation assumptions were realistic then our actual performance should be in the green or yellow zone on the favorable side of production curves. Figure 4 shows the actual data compared the project estimate. The key points are that code is being built and integrated at a faster rate than the plan based on a PI of 11.5 (code production is approximately 4000 SLOC higher than the plan - approximately 30% better) . The staffing and effort are higher than planned (Cumulative effort overrun to date is 486 person hours approximately 22 % higher than planned). The actual performance made us feel comfortable with our estimation assumptions.

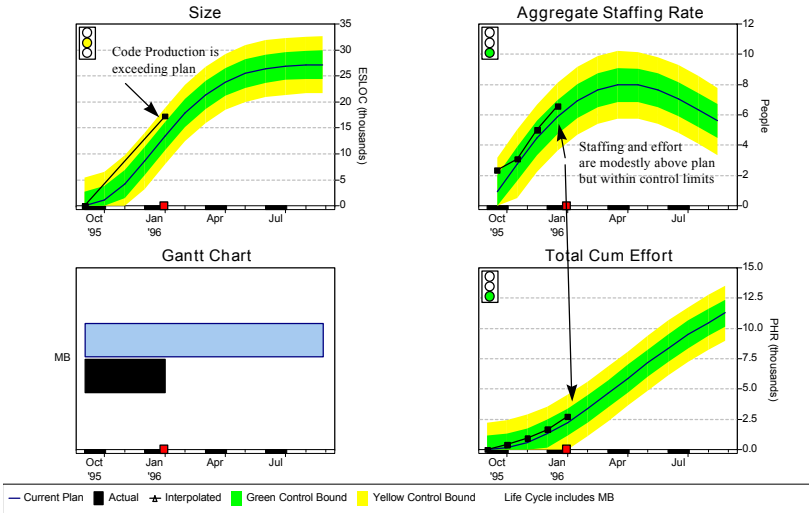

Figure 4. Actual performance vs. estimate based on a size of 26,899 SLOC and a PI of 11.5. The actual performance so far indicates that this project is doing fine. It is probably achieving a productivity index better than 11.5

Worse case scenarios: For the worst case scenario we assumed that the purchased software had to be built from scratch and the project staffing would not increase above the current staff of 6.

Our best estimate showed the size would increase by about 25 objects (2,823 Smalltalk SLOC). The new expected size would be 29,722 SLOC. The worse case estimate is shown in Figure 5. It extends the schedule by about 7 weeks to the mid November time frame.

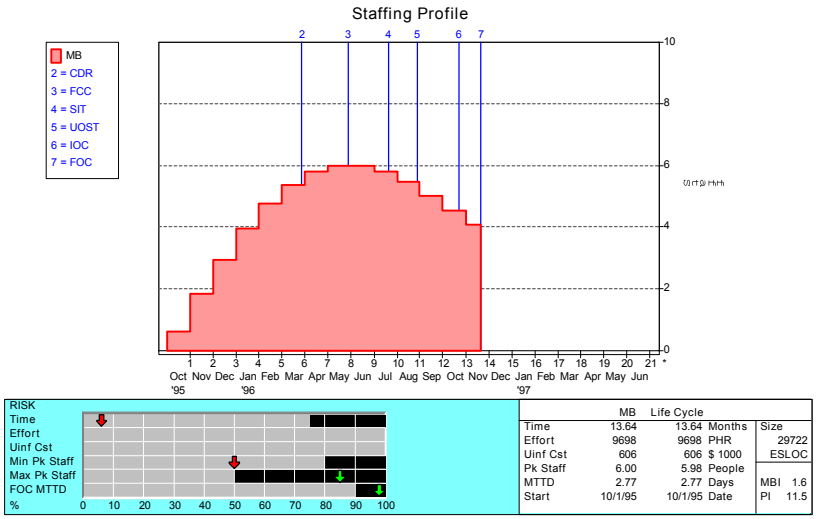

Figure 5. Worse case scenario. Assumes purchase software will have to be built and peak staffing will not exceed 6 people.

The schedule probability analysis show that there is only a 7% chance that the project would be completed by the September 15 date. There is a 90% probability that the schedule will not extend beyond January 10, 1997.

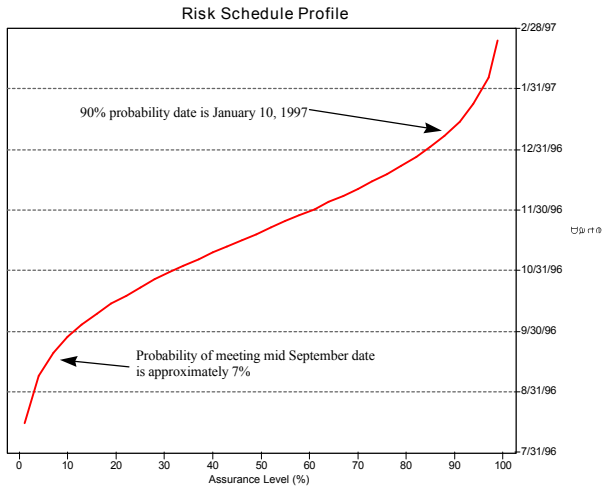

Figure 6. Schedule probability analysis

Summary

Estimating projects that have new technology are full of unknowns. To be successful you should:

- Ask specific questions of the developer to find out what software will be newly developed, what will be reused with no changes and how much software will be modified.
- In sizing the application be flexible so you are able to accommodate any new technology.
- Use your historical data to provide insights about how the organization has assimilated change in the past and what your current capability is today. Look at your new situation and apply some professional judgment backed up by empirical evidence.
- Set realistic expectations - be on the conservative side in your choice of estimation assumptions.
- When appropriate present alternative estimating scenarios (add more people, reduce function, etc.).
- Present best case and worse case scenarios - commercial managers and customers like to know their options.
- Bound your estimates with probability analysis - This quantifies the amount of uncertainty in the estimate consistent with the quality of the input assumptions.
- Track progress against the base line plan . This allows you to confirm the validity of your initial estimation assumptions and make adaptive forecast if your assumptions were incorrect, or if the scope of the project changes. By using this disciplined measurement approach you build a metrics database as you guide the project through to successful completion.

Give these ideas a try. I think you will find that they take a lot of risk out of a risky business.