"Adapting Traditional Software Cost Estimating Techniques to New Technology- A PowerBuilder Case Study"

by Jennifer Manring
Thomas Restivo

INTRODUCTION

The ability to estimate the cost of developing software gets increasingly more difficult as new technologies are introduced into an ever changing software environment. New languages, new architecture, design, and development techniques, commercial-off-the-shelf (COTS) integration, legacy code migration, web-based applications, and new code generating tools are just a sample of changes that have recently occurred in the software field. All these changes impact the cost of developing and maintaining software to varying magnitudes.

The software cost estimator needs to understand the technical implications and cost impacts of these technological changes. An estimator can either determine how to best adapt traditional cost estimating techniques or create new estimating mechanisms that are indicative of the "new" software environment. Many of the traditional techniques are predicated on estimating future costs based on historical data or past performance. With changes in the software world occurring with such frequency, the ability to build and maintain a database of historical data that addresses specific new technologies and approaches is limited.

Vendors of proprietary software cost estimating models are attempting to keep their models current with these changes in the software community by collecting data relative to new technologies and approaches and updating their estimating relationships accordingly. However, vendors are hampered by the limited set of actual data that is available in the software community. In cases where a tool appears to have the ability to address a certain new technology, the algorithmic equations behind the model may be based on a very small and limited data set.

In this paper, we discuss how we tailored a commercial parametric cost estimating model to estimate the cost of a software development that was being built using a recently introduced object-oriented (OO) based graphical tool, PowerBuilder. We tailored model inputs to reflect the unique characteristics and aspects of using this new software tool, including translating non-traditional software sizing metrics to source lines of code (SLOC) and calibrating a productivity factor based on data collected from a previous build. This software development cost estimate was recently generated for a Government program.

SYSTEM BACKGROUND

The system being estimated will modernize current transportation systems, include significant new functionality, and provide an open systems infrastructure that will allow new requirements to be easily incorporated in the future. The system provides the customer with an automated management and tracking system to support in-transit visibility over transportation functions during peacetime and contingency operations. This automated system will support fixed and deployed sites, and offer additional and enhanced functionality not present in existing legacy systems

The system's primary functions are to process and track cargo and passenger information, support management of resources, generate standard and ad hoc reports, and to provide message routing and delivery service for virtually all airlift data. This new system will utilize relational databases to disseminate information currently in fielded systems. The system will also provide an integrated set of analysis, reporting, management, and data maintenance tools, and support scheduling and forecasting.

The contractor plans to develop and deliver the software in incremental phases called "builds". Each build has a specific purpose that fits into the overall purpose. For example, Build 1 focuses on transitioning legacy systems off the existing platform, and Build 2 concentrates on modernizing current functionality and incorporating new capabilities. An additional future build (Build 3) will be developed to enhance the system and satisfy new user requirements.

POWERBUILDER

PowerBuilder, developed by Sybase, Inc., is a Personal Computer (PC)-based graphical tool for developing client/server applications under Microsoft Windows 3.x, Microsoft Windows 95, Microsoft Windows NT, Macintosh, and Solaris operating systems. Applications can be built by creating windows, controls (such as listboxes and buttons), and menus within the PowerBuilder development environment. PowerBuilder uses screens, known as painters, to graphically combine the visual pieces of the application. Code is then attached to these visual pieces in a simple BASIC-like language called PowerScript. The PowerScript code that supports a specific event is called an eventscript.

PowerBuilder is a rapid application development (RAD) tool that supports fast development and deployment of complex applications. In PowerBuilder, a working prototype can be developed in a fraction of the time it may take to do the same in a 3rd generation language (3GL) (for experienced PowerBuilder developers).

Using PowerBuilder, front-end applications which access Relational Database Management Systems (RDBMSs) can be developed without coding in a 3GL, such as C or C++. PowerBuilder is known as a 4th generation language (4GL), because it hides the complexities of a 3GL. PowerBuilder supports programming on many database backends including Sybase, Oracle, Informix, and others. Additionally, PowerBuilder is strong in

the area of scalability.

Although not a pure OO language, PowerBuilder supports the major concepts behind OO programming. The objects contained in PowerBuilder are fully extensible via OO-based methods. Windows, controls, and visual objects are generally fully inherited. However, only the properties, variables, and code of the object can be inherited as PowerBuilder does not support multiple inheritance. Additionally, polymorphism is enabled and encapsulation permitted. PowerBuilder allows applications to be structured according to reuse code within and across applications.

This PowerBuilder description and the analysis that follows are based on the PowerBuilder 5.0 version. PowerBuilder 6.0 is now released and includes improved object capabilities and an Internet developer toolkit for creating web applications.

METHODOLOGY

The following methodology section describes the parametric cost estimation technique and outlines the commercial software cost model input tailoring we applied in the process of generating the software development cost estimate.

Parametric Estimation

For this program estimate, we elected to use the parametric modeling technique to estimate software development costs based on the availability of technical and cost data. Parametric modeling is a cost-estimating method that employs multivariate models, or equations, which have been derived from a combination of expert opinion and statistical fitting to historical data. The cost analyst, as a user of a parametric model, inputs a set of attributes that characterize the software development: product characteristics such as size and complexity, personnel attributes such as experience and skill level, and environmental factors such as tools availability and number of subcontractors. The typical parametric software cost model outputs development and maintenance effort, cost, and schedule estimates. Some commercial cost models also compute probability distributions, which reflect estimation uncertainty that is common in software cost estimates.

Specifically, we used the Lockheed-Martin commercial parametric cost model, Parametric Review of Information for Costing and Evaluation Software (PRICE S ™). PRICE S is a widely accepted software cost estimating tool both within the commercial community and in Government Agencies. PRICE S provides software development estimates based on project scope, program composition, processor loading, and developer performance.

PRICE S Input Tailoring

The software development cost estimate reflects the costs to develop Builds 2 and 3. Build 1 software development costs were sunk costs, therefore not included in the estimate. Actual Build 1 size metrics were provided by the Program Office and software developer and used to convert the non-traditional sizing estimates given for Builds 2 and 3 to traditional SLOC estimates. SLOC translations that could not be derived from calibrating Build 1 data were calculated using engineering judgment. The tailoring of specific PRICE S model inputs used to generate Build 2 and 3 software development cost estimates are described in detail below.

Build 2 and 3 Model Inputs

The principal PRICE S input variables may be grouped into seven categories: (1) project magnitude; (2) productivity; (3) level of new design and new code; (4) customer specifications and reliability requirements; (5) program application; (6) utilization; and, (7) development environment. We tailored the PRICE S inputs to best reflect Builds' 2 and 3 software development environment and the impact of using PowerBuilder. Only the parameters that were uniquely tailored for this PowerBuilder and structured query language (SQL) software cost estimate are described below, specifically project magnitude and productivity.

Project Magnitude

The project magnitude inputs specify the software size and languages to be used. Detailed sizing data, metrics, and SLOC translations for each build are discussed below. The detailed SLOC counts shown are also broken out by the software language used.

Software Size

Software size is a key cost driver in any software development estimate. Traditional sizing metrics, such as SLOC and function points, are the primary sizing inputs used in commercial software cost models like PRICE S. However, the Program Office and software developer could only partially provide these traditional metrics. Instead, non-traditional sizing metrics, such as numbers of windows and stored procedures, were provided because this is how the software developer had been collecting and tracking sizing metrics to-date.

Therefore, we had to develop a process for translating these non-traditional sizing metrics to SLOC for input compatibility with the PRICE S model. The major steps in this translation process are shown below and described in more detail in this section.

Step 1: Collected software sizing data (SLOC and OO metrics) for all Builds

Step 1 describes the methodology and reasoning behind the initial data request to the Program Office and software developers. Actual data collected for Build 1 is shown in detail below. However, not all the requested data was able to be provided for Build 2.

Instead, the software developer provided non-traditional sizing metrics for Build 2 which are also shown in detail below.

Step 2: Analyzed and determined relationships (translation factors) with Build 1 software metrics

Step 2 describes how translation factors were calculated using the Build 1 data. These factors were used to translate the non-traditional software sizing inputs shown in Step 1 to traditional SLOC estimates for Builds 2 and 3. The actual translation factors calculated from Build 1 are shown in detail below.

Step 3: Adjusted SLOC to account for automated code generation

Step 3 describes how the sizing translation factors (calculated in Step 2) were adjusted to account for the use of automated tools used by the software developer in the development process. The sizing adjustments were performed to accurately reflect the amount of estimated software to be developed in Build 2. Each automated code generating tool and corresponding adjustment factor are discussed in detail below.

Step 4: Calculated SLOC for Builds 2 and 3 using calibrated Build 1 translation factors.

Step 4 shows the results of the estimated SLOC calculated for Builds 2 and 3 based on the methodology described in Steps 1 through 3. Additionally, software size for development unique to Build 2 was also estimated. The size of the unmodified software from previous builds was factored into the Build 2 and 3 estimate to account for integration with the respective earlier builds. Results from Step 4 are shown in detail below.

The following descriptions of each of the four Steps provide greater detail and results.

Step 1: Collected software sizing metrics (SLOC and OO metrics) for all Builds

The software development effort was primarily comprised of two efforts: 1) a PowerBuilder and C++ development; and 2) a Database development. The PowerBuilder and C++ effort is divided into five major components based on functionality called Computer Software Configuration Items (CSCIs). All five CSCIs are shown together as the translation factors used are consistent across all five CSCIs. The Database effort represents the sixth CSCI and is shown separately because of the unique translation factors used for stored procedures. Total size per build is the sum of both of these efforts.

To accurately estimate the software size of the PowerBuilder effort, we needed to collect OO type sizing metrics. With having limited experience in estimating OO type developments, we were uncertain to what were the best kind of OO sizing metrics to collect. Therefore, we contacted PRICE S representatives who recommended collecting OO metrics such as number of objects, average depth in inheritance tree,

average number of children, and a weighted method per class (weighted on complexity). These OO metrics could then be input into the new PRICE S feature (Predictive Object Points (POP) Sizer) to reflect the unique characteristics of an OO development that may be lost in SLOC counts. Also, we requested SLOC counts to account for the non-OO software development efforts, such as the C++ and SQL (database) development. SLOC counts were also requested to serve as a backup in case the developer could not provide the OO metrics and to provide insight into differences between the two sizing techniques. Both OO metrics and SLOC counts were requested for Builds 1 through 3.

Unfortunately, the Program Office and software developer were not able to provide OO metrics and SLOC for all the Builds. For Build 1, SLOC could be provided but only a partial set of OO metrics, specifically, number of objects, could be provided. Neither the Program Office nor the software developer were able to provide estimated SLOC or OO sizing data for Builds 2 and 3. Instead, the Program Office provided sizing data based on an estimate of the number of windows and stored procedures planned for Build 2.

Tables 1 and 2 show the actual metrics, SLOC and number of objects respectively, for Build 1. In Table 1, CSCI 1 shows no SLOC counts as no software development was required in Build 1. Table 2 however, shows a computed average for CSCI 1 as this information was needed to compute SLOC translations for Builds 2 and 3. No actuals were available from Build 1 as no development for CSCI 1 was performed for Build 1.

Table 1. Build 1 Actual KSLOC per Language

Language	CSCI 1	CSCI 2	CSCI 3	CSCI 4	CSCI 5	Total
PowerBuilder	0	16	42	22	105	184
C++	0	0	0	0	44	44
Build 1 KSLOC per Language	*0*	*16*	*42*	*22*	*149*	***228***

Table 2. Build 1 Actual Number of PowerBuilder Objects

	CSCI 1	CSCI 2	CSCI 3	CSCI 4	CSCI 5
PowerBuilder Objects	*235**	134	319	253	580

*An average of CSCIs 2 through 4 was used for future translations.

In addition to the PowerBuilder and C++ effort, the software developer estimated that there were 842 stored procedures developed for CSCI 6 for Build 1. Table 3 shows

the estimated numbers of new and modified windows and stored procedures for Build 2. Modifications to Build 1 and new development for Build 2 are broken out as both of these efforts comprise the total Build 2 development effort. Build 3 metrics were calculated as a percentage of growth over Build 2 due to the uncertainty and changing requirements associated with Build 3. Build 3 calculations are discussed in greater detail and shown in Step 4.

Table 3. Build 2 Windows and Stored Procedure Estimated Counts

	CSCI 1	CSCI 2	CSCI 3	CSCI 4	CSCI 5	Total
Build 1 Windows Modified	0	0	6	10	4	**20**
New Build 2 Windows	8	0	36	137	12	**193**
Total Build 2 Windows	**8**	**0**	**42**	**147**	**16**	**213**
Build 1 Stored Procedures Modified	0	36	149	87	231	**503**
New Build 2 Stored Procedures	40	0	105	375	260	**780**
Total Build 2 Stored Procedures	**40**	**36**	**254**	**462**	**491**	**1283**

Step 2: Analyzed and determined relationships (translation factors) with Build 1 software metrics

Since PRICE S uses either SLOC or function points as a sizing input, we needed a mechanism or relationship that translated the Build 2 estimated windows and stored procedures to SLOC or function points. SLOC was the logical choice because we already had Build 1 actual SLOC metrics. As shown previously in Table 3, Build 2 metrics were provided in numbers of windows (both windows and tabs were included) and stored procedures to be modified from Build 1 as well as new development for Build 2. The window counts were mapped to the PowerBuilder and C++ effort (CSCIs 1 through 5), and the stored procedure counts were summed together and designated as the database (SQL) effort (CSCI 6). Because of the distinct differences in language and metrics, these efforts required separate translation factors.

Based on feedback from the software developer, we knew there existed a relationship between number of objects per window. However, we needed a link to SLOC to estimate the PowerBuilder and C++ effort. The linking factor used between these metrics was the PowerBuilder logical code grouping called an eventscript. An eventscript is essentially a grouping of SLOC that supports a specific event (e.g., open window, select option, etc.).

To compute the translation factors, we needed the Program Office to provide Build 1 number of windows, objects, eventscripts, and KSLOC per CSCI. These metrics are shown in Table 4.

Table 4. Build 1 Windows, Objects, Eventscripts, and KSLOC Metrics

	CSCI 1	CSCI 2	CSCI 3	CSCI 4	CSCI 5
PowerBuilder Number of Windows	16*	8	26	14	10
PowerBuilder Number of Objects	251*	134	319	253	580
PowerBuilder Number of Eventscripts	1,809*	1,321	1,765	1,537	1,031
PowerBuilder KSLOC	30*	16	42	22	105

*Computed from averages of CSCIs 2 through 4.

The calculated relationships between objects/window, eventscripts/object, and SLOC/eventscripts were used to estimate Build 2 SLOC per CSCI. These translation factors are shown in Table 5. These factors will translate the number of windows to SLOC.

Table 5. Build 1 Calibrated Translation Factors

	CSCI 1	**CSCI 2**	**CSCI 3**	**CSCI 4**	**CSCI 5**
PowerBuilder Objects/Window	*16**	17	12	18	58
PowerBuilder Eventscripts/Object	*7**	10	6	6	2
PowerBuilder KSLOC/Eventscript	*.02**	.01	.02	.01	.10

* An average of CSCIs 2 through 4 was used for CSCI 1 for translation factors.

We researched and found a limited amount of data involving non-traditional sizing ratios and translation factors from a commercial software cost model vendor. Metrics of four to five eventscripts/object and .050 KSLOC/eventscript have previously been seen by the vendor with limited data points (less than three - all from the same company). However, when these metrics were input into our translation model, the results were deemed unreliable for this software development effort. The translation model estimated the Build 1 modifications to be significantly greater than the original (actual) SLOC metrics provided by the Program Office for Build 1. Hence, we used the translation factors that were computed directly from our own Build 1 data.

C++ sizing for Build 2 was prorated based on the ratio of C++ SLOC to PowerBuilder SLOC from Build 1. The relationship between C++ and PowerBuilder code was

determined to remain constant from Build 1 to Build 2 based on engineering judgment by the Program Office and software developer, as the C++ code is the supporting link between the PowerBuilder windows and the SQL database. For Build 1, C++ code comprised 42 percent of the total SLOC (PowerBuilder and C++ combined). The 42 percent factor was then applied against the total estimated Build 2 SLOC to estimate the C++ portion of Build 2 SLOC. .

The Database effort is categorized into three development types: 1) manual stored procedure development, 2) CAST Workbench generated stored procedure code, and 3) ERWin generated trigger code. The CAST Workbench and ERWin automated code generator tools and corresponding adjustments to SLOC to reflect the use of automated code generators, are discussed in detail in Step 3. The Database vendor working with the software developers broke the 842 Build 1 stored procedures out by development type. The Database vendor also ran queries on the stored procedures developed for Build 1 and provided an average of .18 KSLOC per stored procedure (manual and CAST Workbench) as shown in Table 6. However, the vendor's expert opinion, , concluded that the average SLOC calculated was too high because Build 1 figures included many large stored procedures (approximately six data points ranging from 1.0 to 6.0 KSLOC per stored procedure). The stored procedures with such high SLOC counts were unique stored procedures that most likely would not need to be reworked for Build 2 and therefore skewed the true average. Based on this opinion, we subtracted out the outlying datapoints and reran the queries, which yielded an average of .13 KSLOC per stored procedure. This translation factor was then used to compute the estimated Build 2 and 3 SQL SLOC. The ERWin trigger KSLOC/stored procedure of .10 was deemed valid by the Database vendor. The percent breakout by database development type was also assumed to be constant for future Builds.

Table 6. Build 1 Database Actual KSLOC per SQL Stored Procedure

	Number of Stored Procedures	KSLOC	Avg. KSLOC/ Stored Procedure	Percent Breakout
SQL Stored Procedures - Manual	399	71	*0.18*	57%
SQL Stored Procedures - CAST Workbench	131	23	*0.18*	19%
SQL ERWin - Triggers	312	30	0.10	24%
Total	*842*	***124***	-	-

Step 3: Adjusted SLOC to account for automated code generation

The software developer used automated code generating tools in both the PowerBuilder/C++ and SQL (database) efforts. The SLOC translations calculated

from Build1 data needed to be adjusted to better account for actual software development when using an automated code generating tool. These adjustments were done so that the SLOC inputs into the PRICE S model would reflect a true measure of actual software development effort. Otherwise, the software development effort would factor in work done automatically by the code generating tools. These adjusted SLOC estimates are called *equivalent* SLOC.

PowerBuilder has a built-in code generator capability. The code generator partially translates window creation, design, and properties developed in a graphical user interface (GUI) environment into SLOC. The Build 2 PowerBuilder SLOC estimate was adjusted down to 50 percent of its total estimated size (based on Program Office input) to better account for actual PowerBuilder software development effort.

As described previously, the Database effort is categorized into three development types. The manual stored procedure estimates required no adjustment because this aspect of the database development effort did not employ any code generating tools. The CAST Workbench effort involved the CAST Workbench code generating tool automatically creating SQL stored procedures. The CAST Workbench SLOC estimate was adjusted down by 50 percent (based on vendor input). The stored procedures generated by CAST Workbench required some manual adjustments and modifications, which were captured by the remaining 50 percent of estimated SLOC. The ERWin code generator tool creates database triggers. The ERWin estimated SLOC was adjusted down by 90 percent (based on vendor input) because this software is almost completely automated. Again, minor manual adjustments were required, which were captured in the remaining 10 percent of estimated SLOC.

These adjusted automated code generated SLOC factors were applied to the actual Build 1 CSCI SLOC totals as shown in Tables 7 and 8, respectively. These factors were then applied to the Builds 2 and 3 computed SLOC after the translation factors were applied as shown in Step 4.

Table 7. Build 1 Equivalent PowerBuilder and C++ SLOC

Language	CSCI 1	CSCI 2	CSCI 3	CSCI 4	CSCI 5	Total
PowerBuilder	0	16	42	22	105	184
PowerBuilder Equivalent KSLOC (50%)	0	8	21	11	52	92
C++	0	0	0	0	44	44
Equivalent Build 1 KSLOC per Language	*0*	*8*	*21*	*11*	*96*	***136***

Table 8. Build 1 Database Equivalent KSLOC per SQL Stored Procedure

	KSLOC
SQL Stored Procedures - Manual	71
SQL Stored Procedures - CAST Workbench	23
CAST Workbench Equivalent KSLOC (50%)	*12*
SQL ERWin - Triggers	30
ERWin Equivalent KSLOC (10%)	*3*
Equivalent Build 1 KSLOC per SQL Stored Procedure	***86***

Step 4: Calculated SLOC for Builds 2 and 3 using calibrated Build 1 translation factors.

Once the SLOC translation factors and adjustment factors were calculated, we could compute SLOC estimates for both the Builds 2 and 3 PowerBuilder and C++ effort as well as the SQL (database) effort. The translations for each of these efforts are shown separately. Table 9 shows the Build 2 SLOC translations, Build 1 translation factors, and predicted Build 2 metrics (both for Build 1 modifications and Build 2 new development). These translations calculate unadjusted PowerBuilder SLOC. As described earlier, a 50 percent SLOC adjustment was given to calculate equivalent PowerBuilder SLOC.

Table 9 also shows other software development effort that consisted of C++, UNIX scripts, and C routines that link the PowerBuilder windows to the Database. Since no automated code generating tools were used in these efforts, SLOC is equal to equivalent SLOC. The UNIX scripts and C routines were calculated using engineering judgment provided by the Database vendor. In Build 2, there were 54 UNIX scripts and 34 C routines that resulted in 2.7 KSLOC and 1.7 KSLOC, respectively. Based on these figures, an estimate of .05 KSLOC per script or routine was applied. The sum of these efforts combined with the PowerBuilder equivalent SLOC is the total Build 2 estimated SLOC for CSCIs 1 through 5.

Table 9. Build 2 PowerBuilder and Other Effort Estimated Equivalent SLOC Count

	CSCI 1	CSCI 2	CSCI 3	CSCI 4	CSCI 5	Total
Build 1 PowerBuilder Objects/Window	16	17	12	18	58	-
Predicted Build 2 Objects - Build 1 Mod	0	0	73	181	232	486
Predicted Build 2 Objects - New Build 2	126	0	439	2480	696	3,741
Build 1 PowerBuilder Eventscripts/Object	7	10	6	6	2	-
Predicted Build 2 Eventscripts - Build 1 Mod	0	0	405	1100	412	1,917
Predicted Build 2 Eventscripts - New Build 2	898	0	2430	15064	1237	19,629
Build 1 PowerBuilder KSLOC/Eventscript	.02	.01	.02	.01	.10	-
Predicted Build 2 KSLOC - Build 1 Mod	0	0	10	16	42	67
Predicted Build 2 KSLOC - New Build 2	15	0	57	214	126	412
PowerBuilder SLOC Total	15	0	67	230	168	480
PowerBuilder Equivalent SLOC	*8*	*0*	*34*	*115*	*84*	*240*
Other Effort						
C++ SLOC	0	0	0	0	70	70
UNIX Script SLOC	0	0	0	0	3	3
C Routine SLOC	0	0	0	0	2	2
Other SLOC Total	0	0	0	0	74,318	74,318
Equivalent SLOC Total	*8*	*0*	*34*	*115*	*158*	*314*

The Database effort also needed to be broken out by development type to accurately calculate equivalent SLOC. The Database vendor (working with the Program Office) concluded that the numbers of stored procedures provided for Build 2 did not include triggers. A relative percentage of triggers calculated from Build 1 (59 percent) was used to increase the number of stored procedures estimated to account for the triggers. The original number of stored procedures was divided equally into manual generation and CAST Workbench tool generation (based on Database vendor expertise and expected increase of tool usage). The appropriate adjustments were made to calculate equivalent SLOC from the translations. Table 10 shows the SLOC translations for the SQL (database) effort.

Table 10. Build 2 Database Translated Equivalent KSLOC Count

	B1 Mod		B2 New	
	# Stored Procedures	KSLOC	# Stored Procedures	KSLOC
SQL Stored Procedures - Manual	252	31	390	49
SQL Stored Procedures - CAST Workbench	252	31	390	49
CAST Workbench Equivalent KSLOC (50%)	-	*16*	-	*24*
SQL ERWin - Triggers	296	28	459	44
ERWin Equivalent KSLOC (10%)	-	*3*	-	*4*
Total SQL Equivalent KSLOC	***799***	***50***	***1,239***	***78***

Build 2 also required some additional SQL to account for the replication process required to disseminate information from the central data repository to the individual sites. The replication process is based on the publish/subscribe philosophy. A particular location publishes all that is available to be replicated and receiving locations subscribe to the particular information they are interested in. The replication definitions are the publications in this scenario. This program's replication strategy was the replication of stored procedures. So, a replication definition for each stored procedure to be replicated needs to be created at the primary site. A subscription would be created for each site wishing to receive that information.

Table 11 shows the total equivalent SQL SLOC (same as SLOC in this case) for the replication definitions, unique subscriptions, and subscription modifications. Once a generic subscription is created, it can be modified for different sites by simply changing name and identifier of the site. This SLOC total is added to the SQL SLOC total calculated from the stored procedures to calculate total Build 2 SQL SLOC. This total is 140 KSLOC.

Table 11. Build 2 Additional SQL SLOC

	Number of Def/Subscriptions	SLOC/Def/Subscription	Total KSLOC
Replication Definitions	106	8	.80
Unique Subscriptions	73	10	.73
Subscription Modifications	7,649	2	12
Total Equivalent SQL SLOC	**-**	**-**	**12.997**

Build 2 also had to account for integration of the Build 1 software that was not modified. Table 12 shows the calculations for Build 1 unmodified minus Build 1 modified software. The reused SLOC counts were used as direct input into PRICE S. No equivalent SLOC was calculated for this effort as all SLOC (regardless of how it was generated) will need to be integrated together with the new Build 2 SLOC.

Table 12. Build 2 Reuse Metrics (KSLOC)

	CSCI 1	CSCI 2	CSCI 3	CSCI 4	CSCI 5	CSCI 6 (DB)	Total
Build 1 PowerBuilder SLOC	0	16	42	22	105	0	184
Build 1 Mod PowerBuilder SLOC	0	0	10	16	42	0	67
PowerBuilder REUSE	*0*	*16*	*32*	*6*	*63*	*0*	**117**
Build 1 SQL SLOC	-	-	-	-	-	124	124
Build 1 Mod SQL SLOC	-	-	-	-	-	91	91
SQL REUSE	-	-	-	-	-	*33*	**33**
Build 1 C++ SLOC	0	0	0	0	44	0	44
Build 1 Mod C++ SLOC	0	0	0	0	0	0	0
C++ REUSE	0	0	0	0	44	*0*	**44**

Table 13 shows Build 3 metrics. Build 3 SLOC was computed using a growth factor of 33 percent, based on Program Office input, over the unadjusted SLOC counts from Build 2 SLOC. Build 3 SLOC was estimated over the final estimated SLOC of Build 2, which represented new development and modifications of Build 1 as well as the reused SLOC from Build 1. The appropriate adjustments were made to calculate equivalent SLOC. These equivalent SLOC counts were used to estimate the software development of Build 3.

Table 13. Build 3 Metrics (KSLOC)

	CSCI 1	CSCI 2	CSCI 3	CSCI 4	CSCI 5	CSCI 6 (DB)	Total
Build 2 PowerBuilder SLOC Total	15	16	99	236	230	0	597
Build 3 PowerBuilder SLOC	5	5	33	78	76	0	197
Build 3 PowerBuilder Equivalent SLOC	3	3	16	39	38	-	99
Build 2 SQL SLOC Total	-	-	-	-	-	279	279
Build 3 SQL SLOC	-	-	-	-	-	92	92
Build 3 Stored Procedure (Manual)	-	-	-	-	-	35	35
Build 3 Stored Procedure (CAST)	-	-	-	-	-	35	35
Build 3 ERWin Triggers	-	-	-	-	-	22	22
Build 3 SQL Equivalent SLOC	-	-	-	-	-	55	55
Build 2 C++ SLOC Total	-	-	-	-	118	0	118
Build 3 C++ SLOC	-	-	-	-	39	0	39
Build 2 UNIX Scripts SLOC Total	-	-	-	-	3	-	3
Build 3 UNIX Script SLOC	-	-	-	-	1	-	1
Build 2 C Routines SLOC Total	-	-	-	-	2	-	2
Build 3 C Routine SLOC	-	-	-	-	1	-	1

Programming Language

The specific programming languages used to develop each Build include PowerBuilder, C++, UNIX scripts, C, and SQL (generated through manual coding, CAST Workbench generated stored procedures, and ERWin generated triggers). Further breakout was described previously.

Productivity

Productivity reflects the experience, skill, and knowledge of the assigned individuals or team, as applied to the specific software development. The PRICE S input that captures the overall capability of an individual or team is the "Productivity Factor." PRICE S recommends that the Productivity Factor (PROFAC) be calibrated from historical data when the developing organization is known. Performance characteristics on recent projects tend to hold for future projects.

Our ability to tailor the productivity input parameter was affected by the limited historical data available associated with developing software using PowerBuilder. In discussions with one commercial model vendor, they could only provide five unique

PowerBuilder data points, three of which came from the same company and were somewhat dissimilar in nature to this system. A second model vendor could not provide any productivity guidance specific to PowerBuilder. Additionally, we searched the web extensively and talked directly to the PowerBuilder vendor, but were still unable to gather sufficient productivity information or data specific to PowerBuilder. Therefore, we did not feel comfortable relying on the PRICE S default values or recommended productivity values because the model database did not contain data from software developments that used PowerBuilder.

Therefore, we interviewed Program Office personnel to gather as much actual data on the efforts associated with the Build 1software development, since the contractor used PowerBuilder in the Build 1 software development process. Build 1 metric data (size, cost, and complexity variables) was then calibrated to determine the contractor's actual PROFAC. Based on the platform (a measure of the portability, reliability, structuring, testing and documentation required for acceptable contract performance) for this system, the calibrated PROFAC resulted in a value slightly below the mid-point of the PRICE S recommended PROFAC values. If we had used the industry average (default value), our final estimate would have underestimated the effort required. The calibrated Build 1 PROFAC value was then used as the productivity inputs to PRICE S for Builds 2 and 3.

RISK-UNCERTAINTY

Statistical range estimates were calculated to account for the level of uncertainty and inherent error with using cost models and parametric estimating techniques to estimate a software development effort. We determined that two major areas of uncertainty in this estimate were: (1) software sizing; and (2) productivity levels. The PRICE S risk analysis feature was used to apply risk factors to these parameters and generate cumulative probability distributions. The detailed methodology of the risk analysis and results are described below.

The first major risk area was the total amount of software to be developed. We tried many techniques to estimate a range of software size including using analogous industry translation factors and varying our own translation factors. However, none of these techniques were deemed completely reliable methods as all contained shortcomings. Hence, we used engineering judgment and estimated that the "true" SLOC count was within a range of +/- 20 percent of our computed SLOC estimates. This range was based on the level of confidence in the Program Office data, level of confidence in the SLOC translation factors, and insight into the development environment. This range was modeled using a triangular distribution with the calculated SLOC count being the most likely value, 80 percent of the SLOC count being the low value, and 120 percent of the SLOC count being the high value.

The second major area of risk was the software development PROFAC value used. Although this factor was calibrated directly from actual Build 1 data, changes in level of complexity and developer experience between Build 1 and Builds 2 and 3 make the calibrated PROFAC value questionable as a point input. Again, a triangular distribution

was used to model a range of PROFAC values. The calibrated PROFAC from Build 1 was used as the most likely, since it was based on actual project data. For the high end of the effort range, a one level lower PROFAC value was used to account for an increased level of complexity in future Builds. For the low end of the effort range, a one level higher PROFAC value was used to credit the development team with experience gained from the Build 1 software development.

Both of these risk input range factors were incorporated into the PRICE S risk analysis feature, and the results of this analysis are shown in a cumulative distribution graph in Figure 1. A monte carlo simulation was used to forecast results that ensured the entire range of possible outcomes was included as well as the likelihood of achieving. One thousand iterations were run. For purposes in this analysis, the cumulative distribution and its mean were used to derive the statistical ranges. The mean is the point estimated to have a 50 percent chance of achievement. The 95th percentile from the cumulative distribution is used for the high estimate (95% chance that costs are less than or equal to the high estimate) and the 5th percentile for the low estimate (5% chance that costs are less than or equal to the low estimate).

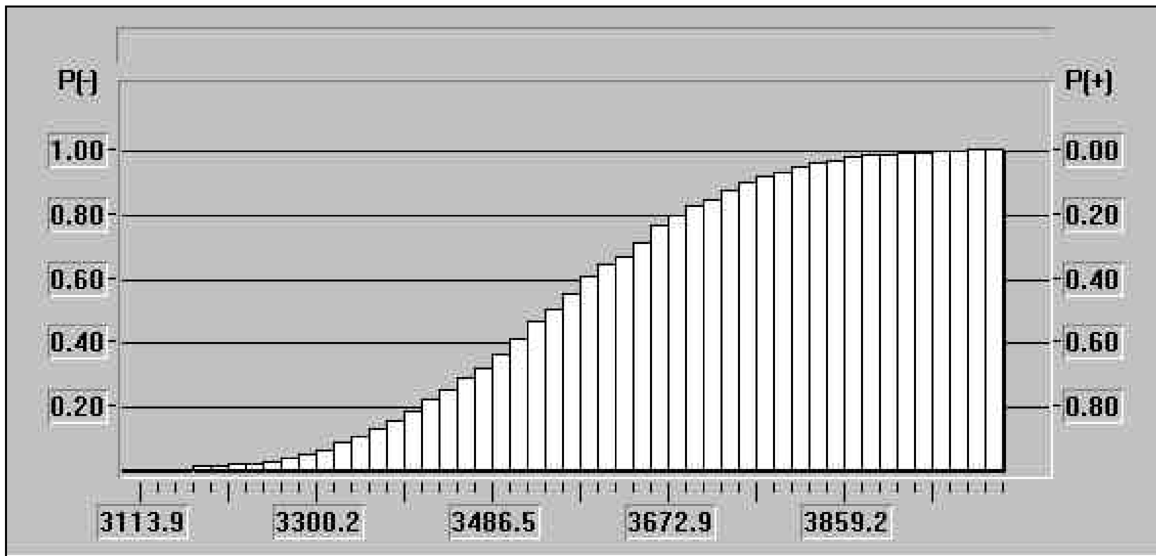

Figure 1. Cumulative Distribution Graph for Build 2 (Effort in Person Months)

The range span from the low to high estimate gives a 90% confidence interval (90% confidence that costs are in this range). The results of the software development effort estimates for the 5 percentile value, mean value, and 95 percentile value are presented in Table 14. The mean standard error is also shown to provide insight into the probability that the true mean of the forecast lies within the estimated mean. The probability that the

true mean is within the estimated mean plus or minus the mean standard error is 68 percent. The software development effort includes system requirements through HW/SW integration as well as system integration.

Table 14. Range Estimate for Build 2 (Effort in Person Months)

	5%	Mean	95%	Mean Std. Error
Acquisition (Investment)	3,307	3,562	3,829	5.0

Build 3 used the same risk inputs and methodology as Build 2 with one exception: reuse (or unmodified) SLOC from Build 2. Build 2 reuse of Build 1 SLOC was an actual SLOC count, so no risk was attached. However, since Build 3 reuse of Build 2 SLOC was not based on an actual, a range of reuse SLOC was modeled using a triangular distribution. Again, a +/- 20 percent was used to estimate the low and high values, and the calculated SLOC value was used as the most likely. Figure 2 shows the cumulative distribution graph for Build 3.

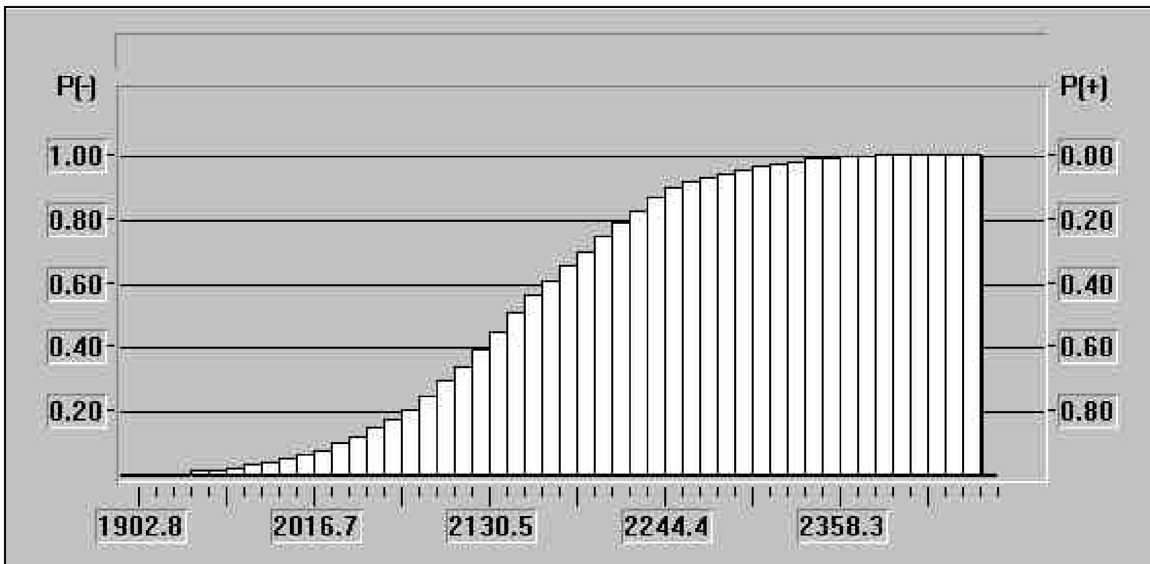

Figure 2. Cumulative Distribution Graph for Build 3 (Effort in Person Months)

Table 15 shows the results of the software development effort estimates for Build 3. Again, the Build 3 software development effort includes system requirements through hardware/software integration as well as system integration.

Table 15. Range Estimate for Build 3 (Effort in Person Months)

	5%	Mean	95%	Mean Std. Error
Acquisition (Investment)	2,009	2,154	2,300	2.7

SUMMARY

With the rapid changes occurring in the software engineering community, software cost estimators need to understand the technological implications and cost impacts of these new technologies and development approaches. New and innovative software language, design, and architecture implementations are bringing traditional software cost estimating techniques into question. We experienced these types of issues and questions on a recent software development cost estimate effort that involved a new OO based graphical tool called PowerBuilder.

In the process of estimating this software development effort, we adapted non-traditional metrics to work within the traditional input requirements of a commercial parametric software cost estimating tool, PRICE S. We had to be very creative in translating nontraditional sizing metrics, like objects, windows, eventscripts, triggers, and stored procedures to traditional SLOC estimates. Also, commercial vendors had limited historical data on PowerBuilder developments; however, we were fortunate to have actual data from the software developer's previous build to calibrate a productivity value rather than use the model's default value.

As the software community continues to grow and incorporate new technologies, the software cost estimating community is going to have to adapt traditional cost estimation techniques or develop new methods that account for these changes. We were fortunate on this effort to have actual data that was applicable to the effort being estimated, but this will not always be the case.