POWERED BY PRICE Research

The Real Costs of Developing COTS Software

By Arlene F. Minkiewicz, Chief Scientist, PRICE Systems, L.L.C.
17000 Commerce Parkway, Suite A, Mt. Laurel, NJ 08054
1-856-608-7222, Arlene.Minkiewicz@PRICESystems.com[1]

Abstract—Despite the increased use of Commercial Off the Shelf (COTS) software, there has been little increase in the understanding of how to successfully estimate and plan for projects that are COTS based or COTS intensive. This paper describes a research effort focused on identifying the activities, cost or effort drivers and cost estimating relationships (CER's) that apply when planning software projects that are COTS based or COTS intensive.

The approach taken for this study is based on a combination of data collection and analysis with theoretical research and expert knowledge. This work has resulted in the development of a set of cost estimation algorithms suitable for embedded COTS integrations and extensible to enterprise COTS integration efforts. As important as the algorithms, this research has resulted in a methodology designed to guide project planners through a thought process geared toward the issues that require consideration when planning such a project.

TABLE OF CONTENTS

1. INTRODUCTION

Increasingly, the developers of software systems are relying on the incorporation of Commercial Off the Shelf (COTS) software components to decrease the cost and increase the cycle time for delivery of new software systems. While it's fairly obvious that solutions can be delivered faster and with less expense when pre-built components are integrated, the factors that determine how much quicker and less expensive are not universally understood. COTS solutions are not no cost (and sometimes not even low cost) solutions. There are cost issues associated with the use of COTS solutions that need to be understood and managed in order to ensure that the COTS solutions that you choose are the ones that will actually save you time and money. This paper studies the activities associated with the development and sustenance of software systems that incorporate COTS solutions and identifies the cost drivers that must be considered when attempting to make an assessment of the economic feasibility of a COTS software integration.

This is not uncharted territory, although until the mid 90's very little attention was given to the subject of the costs associated with COTS software integration efforts. At that time, several independent exercises were undertaken to develop cost estimating relationships for some subset or generalization of the activities associated with COTS inclusive systems [1],[2],[3]. The most comprehensive work to date has been done at the University of Southern California's Center for Software Engineering (USC CSE), in conjunction with the COCOMO II (Constructive Cost Model) project [4], [5].

The effort discussed differs from previous works in two respects. First, the author takes a more activity-based approach to identifying the sources of costs for individual COTS software components as well as the COTS inclusive system overall. Second, the author extends this analysis to activities that take place during the entire lifecycle of the product. A full and fair analysis of the cost effectiveness of COTS software cannot be accomplished until there is an understanding of all of the activities and costs associated

[1] IEEEAC paper #1159, Version 3, Updated December 10, 2003

with COTS inclusion, integration, and maintenance throughout the viable life of the software system.

The paper is organized as follows. Section 2 discusses the methodology the author has employed to attack the problem of identifying cost drivers and estimating relationships for COTS inclusive systems. Section 3 puts bounds around the solution space. Sections 4, 5, and 6 contain the meat of the paper. Section 4 discusses the activities involved in a COTS based development and relates them to their cost drivers while Section 5 provides details about these cost drivers. Section 6 describes the development of cost estimating relationships and gives the general nature of these relationships. Section 7 discusses other COTS related costs that do not fit neatly into a specific activity but add a sense of completeness to the discussion of costs. Section 8 provides conclusions and further directions for this research effort.

2. SOLUTION METHODOLOGY

The first step in any Operations Research project is to identify the problem being solved. The problem we are attempting to solve is that of identifying all of the activities associated with the inclusion of COTS software into a system and determining the cost drivers and cost estimating relationships for those activities. The plan to accomplish this is to use literature review, expert knowledge, and interviews with practitioners to supplement the cost data that is available for COTS inclusive systems. The cost and technical data for this study came from both military and commercial projects and encompassed both embedded COTS software as well as some commercial Information Systems projects. A summary of the data used for this study can be found in [10].

Once the problem is identified, the next step in constructing a parametric cost estimating solution is to study and understand the subject process and from this construct a mathematical model. Research led to the development of a mathematical model based on assumptions of productivity standards for each activity in the process and adjustments of these 'standard' value to account for values assigned to cost drivers. This mathematical model was than exercised by software developers and project mangers to determine how it fared when applied to real life situations. Once satisfied that the model was useful for practitioners, data from various datasets containing both commercial and aerospace data was applied to determine where it worked well and where further work is required.

3. BOUNDING THE PROBLEM

While reviewing the literature on COTS software, one of the first things that became clear was that many people mean many different things when discussing COTS software. In order to have a meaningful discussion on the costs of COTS software, it is important to start with a clear understanding of what is and is not included when we discuss COTS software. For this paper, we started with the definition from the USC study that led to the Constructive COTS (CoCOTS) model [4]. The definition of a COTS software product follows:

- Commercially available software product – sold, leased or licensed
- Source code unavailable but documentation provided
- Periodic releases with new features, upgrades for technology, etc.

What we found was that this definition was too limiting for several reasons. First of all, this is not consistent with many of our observations of actual COTS software integration efforts. Customization appears to be quite common, particularly with embedded COTS software. In creating a general-purpose solution it seemed important that we not overlook the issue of customization. Secondly, we found that the activities related to customization of COTS software component aligned with the activities related to traditional software development. The only additional burden on this research effort was to determine what additional productivity should be considered when the customization is accomplished on off the shelf software rather than code developed in house, and how these factors apply. For these reasons the definition was altered to include off the shelf software with source code available.

In addition to the basic definition of COTS software, there is also confusion related to the many different ways a software development organization may plan to use COTS software. There are three basic ways that a COTS software product is used in software development(s) [5], they are either used as tools, infrastructure, or embedded as part of a new application being developed. The major focus of this research was COTS software that was being embedded into software solutions, however, preliminary research suggests that properly applied, this set of algorithms is rich enough to allow for the estimation of projects integrating COTS software solutions into the infrastructure of an organization as well.

The following discussion focuses primarily on the development of new software systems that include COTS software components to deliver some of their required functionality. Other uses of COTS software include migration for legacy systems and selective upgrades to deployed systems. Although most of the activities included in these types of COTS software integration efforts are covered in this paper, they are not the types of processes that were studied.

4. ACTIVITIES

Although in theory a COTS software component should integrate into your application in much the same way that a component you built in-house would, there are issues associated with COTS software products that are not present when all the components are custom made. In order to have a successful COTS software inclusion, the following activities must be added to your project plan or (if already part of your project plan) must be expanded to incorporate COTS software related tasks:

4.1 Analyze Software Requirements
4.2 Evaluate and Select COTS software
4.3 Purchase (or lease) COTS software
4.4 Tailor COTS software
4.5 Design, Code and Test Glue Code and Modifications
4.6 Perform System Level Integration and Test
4.7 Maintain license, subscriptions, and royalty fees
4.8 Evaluate and include COTS software upgrades
4.9 Fix bugs
.

What follows is a description of each of these activities along with those technical and project factors that were found to drive productivity and cost for the activity.

Analyze Software Requirements

Software requirements analysis is an activity that should occur regardless of whether the decision is made to build, buy, borrow, or some combination of the three. In fact, the decision to build or buy should be part of the process of analyzing requirements. Evaluating and selecting the proper COTS software components without first adequately analyzing requirements is impossible. Selection criteria need to relate back to the requirements that the COTS software components are expected to fill.

Cost Drivers— The primary cost drivers for this activity are:

- Functional Size
- Functional Complexity
- Operating Specification
- Project Constraints

Evaluate and select COTS software

It is normally during the concept phase, or early in the requirement analysis activity that the decision is made to investigate the feasibility of using a purchased rather than custom solution to deliver all or part of the required functionality. Once this decision has been made, the first step to be taken is a determination of availability of COTS software components that have the potential to provide needed functionality.

Once availability is established, an evaluation must take place. There are many reasons to perform an evaluation and many criteria against which this evaluation might be performed. The main reasons to evaluate are:

- Determine whether the functionality promised is the functionality delivered
- Determine whether system non-functional requirements (portability, reliability, security, performance) can be met
- Determine whether functional requirements can be met by the functionality delivered
- Determine whether a proposed suite of components are compatible
- Determine whether a COTS solution is an economically feasible solution

COTS software evaluations need to consider not only product characteristics such as functionality, maturity, technology, architecture and long-term viability, but also vendor characteristics such as maturity, stability, cooperation, and ability to provide adequate support, training and documentation.

There are several techniques commonly used for COTS software evaluation. One technique calls for progressive filtering of available COTS software solutions. This requires several iterations of filtering, each one going into progressively more detail until a single solution or set of solutions becomes clear. This is the approach advocated by the CoCOTS group at USC[5]. Another proposed approach is called the ‘puzzle assembly’ process [9]. This approach suggests that it makes little sense to evaluate COTS software solutions in a vacuum – it is better to evaluate a set of components at one time using an evolutionary prototyping approach. In this method, multiple sets may be evaluated in parallel to identify which of them comes closest to solving the entire ‘puzzle’ presented by the system requirements. This approach, while generally more costly than the filtering approach, greatly reduces the risk of getting to the middle (or further) in the software development cycle and realizing that the COTS software components that have been selected are not compatible with each other. A third approach suggests identifying the keystone COTS software components [9]– those for which requirements (whether they be technical, process, functional, vendor, etc.) are unbendable and then basing other component selections on compatibility and ease of interface with these components. The selection process is often a combination of these three techniques.

Cost Drivers— The primary cost drivers for this activity are:

- Functional Size
- Number of Components being evaluated
- Operating Specification

- Evaluation Multiplier
- Number of Components selected for detailed evaluation
- Number of Upgrades during development

Purchase (or lease) COTS software

While this task is a relative no-brainer as to what the cost driver is (purchase or licensing price) it is still important to include this in the analysis of a COTS software solution because product cost could have serious implications both up-front and throughout the life-cycle of your software solution. If your software development project is expected to span any length of time, it is important to include in your cost assessment, not only the up front costs, but any subscription renewal or upgrade costs that are likely to occur.

Cost Drivers— The primary cost drivers for this activity are:

- Purchase or license price
- Support fee
- Number of upgrades expected
- Number of copies required

Tailor COTS software

There are certain things that have to be done in or around your software to get the COTS software product configured for your system. Databases and other parameters need to be initialized and loaded, all or part of the components need to be registered with the operating system, security must be activated or initialized, screens and reports need to be scripted, and other scripts may be required. These things require no additional code to be written but they take time and effort. They require reading manuals and/or attending training and then experimenting with the actual tool to reach a level of competency.

Cost Drivers— The primary cost drivers for this activity are:

- Functional Size
- Tailoring Multiplier
- Tailoring Complexity
- Vendor and Product Complexity
- Number of Upgrades expected

Design, Code and Test Glue Code and Modifications

Although it would be nice if the COTS software satisfied all of the functional requirements it was selected to meet completely, this is often not the case. In some cases glue code can be developed to deal with short comings and in other cases actual modification of the delivered software must be considered.

Glue code literally holds the system together. Glue code is any code that needs to be written in order to make the COTS software component function as advertised and/or expected. It is the code that references the interfaces in the COTS software component and needs to interpret return codes from these interfaces. Glue code is often required to convert data and other information from the format in which the system maintains it to the format required by the COTS software component. The glue code in a well-written application will act as a layer between the system and the COTS software component, encapsulating the data in such a way that upgrades and replacements are as painless as possible. Finally, glue code or modification is sometimes required to add functionality that the COTS software components implement inadequately or that logically should be provided by the COTS software component but is not.

In many ways glue code and modification to COTS software is no different than other code developed for a software system but there are some added factors that should be considered. These development activities are complicated by the fact that the source code for the COTS software product is unfamiliar or in some cases unavailable. You can think of the complexity of glue code development as being akin to a situation where an entirely new team of developers is being brought in to integrate custom built components but they are given only limited access to the original development team. The unfamiliarity of the interfaces along with the inability to be able to debug into the components adds complexity, and thus effort to the development exercise.

Another factor that makes glue code and modification development differ from traditional code development is the complete reliance on vendors; to fix bugs when they occur, to ensure that upgrades are upwardly compatible, to release stable products and to fill in where documentation falls short. It is also possible that glue code will have to be written specifically to deal with technical or performance problems in the COTS software components. These factors can add time and effort to the development schedule.

Cost Drivers— The primary cost drivers for this activity are:

- Functional Size
- Amount of Modification
- Functional Complexity
- Operating Specification
- Project Constraints
- Integration Team Complexity
- Number of Upgrades Expected
- Glue Code Size
- Glue Code Language
- Vendor and Product Complexity

Perform System level Integration and Test

Naturally, testing is required of all software not just that which integrates COTS software components. However, system level testing costs should change to account for the

fact that external code is being integrated into the software system. In general, the cost for many system level integration activities is likely to be *higher* for COTS inclusive software than software that is all custom-built components because of the unfamiliarity with the code. The COTS software component should be viewed by the integrator as a "black box". This activity is associated with the testing of the system as a complete unit, verifying that there are not system level problems associated with incompatibilities or competing demands of components. Requirements related to performance, reliability and security could be particularly problematic during this phase as these types of problems are likely to go unnoticed until the whole system is running together. It is best to use an incremental or evolutionary approach when building COTS inclusive systems, as these approaches advocate successive integrations during development rather than a waterfall type process where integration does not happen until the end of the development.

Cost Drivers— The cost drivers for this activity are:

- Total Functional Size of all software components
- Functional Complexity
- Operating Specification
- Project Constraints
- Integration Team Complexity
- Number of Upgrades Expected
- Glue Code Size
- Glue Code Language
- Vendor and Product Complexity
- External Integration Complexity

Maintain license, subscription, and royalty fees

The license fee is of course the cost driver for licensed COTS software. If you are paying royalties, that cost is most probably driven by number of installations as well. Once again, this is a simple yet necessary part of any cost analysis of your COTS solutions. It is important to understand the upgrade policies of your vendor. It is wise to do a long-term analysis of the differences between subscription (if this is an option) versus paying for upgrades on an individual basis. Depending on the volatility of the vendor and the frequency of releases, subscription is often the best option.

Evaluate and include COTS upgrades

Just as assessment was needed during development to determine if COTS software upgrades should be included in a software system, it is also necessary to evaluate updated versions of the product(s) after the system is deployed. In some cases these evaluations will result in a determination to refresh the system with the upgraded COTS software products. There are countless reasons why an upgrade would be desirable once the product is deployed. If there are bugs in the COTS software that affect your system operation, obviously you need to upgrade to correct these. Beyond this, the vendor should be upgrading the product to keep up with rapidly changing technology. In order to maintain a software system that meets market expectations with respect to performance, look and feel, operating platform, etc., updates of the COTS software product are expected. In addition to maintenance and adaptation, COTS software upgrades will also offer new and improved functionality that may increase the value of your COTS inclusive software system. Certainly, a benefit of a well developed, well maintained, and well integrated COTS software component should be decreases in maintenance costs and upgrade cycle time.

While this paper has already talked extensively about evaluation and re-evaluation costs, what needs to be addressed here is the costs of upgrades and the risks associated with a COTS software upgrade. Sometimes upgrades change the interfaces to the COTS software components so that with the upgrade, existing functionality ceases to work correctly requiring rewrite of some of the glue code. Even the vendors that do an excellent job of maintaining interfaces and keeping their eye on upward compatibility find that in order to offer new features or to keep up with technology they must change interfaces and rework functionality. In order to get access to the new features and technology in an update, the software developer may be forced to accept changes that they don't want or need as well – creating additional cost with no added value to the software system. Every upgrade also carries with it the possibility of incompatibilities with your existing software system, other COTS software components, or even the operating platforms on which you expect the system to work. For this reason each upgrade should include a repeat of system operational testing and system regression testing to ensure that the effects on system operation are understood and desirable. While it is important to understand all this when evaluating whether or not to upgrade it is also important to understand that there is a risk associated with failure to upgrade. At some point the vendor may decide to discontinue support on older versions of the COTS software. This possibility should also be considered.

Cost Drivers— The cost drivers for this activity are:

- Functional Size
- Evaluation Multiplier
- Functional Complexity
- Operating Specification
- Number of Components being evaluated
- Number of component for detailed evaluation
- Total Functional Size of System
- Vendor and product Complexity
- External Integration Complexity
- Integration Team Complexity

When estimating life cycle costs a certain amount of enhancement should be assumed as part of that calculation so additional tailoring and glue code of new functionality may be part of the overall cost picture as well.

Fix bugs

COTS software, just like the code you build in house, is not defect free. And the cost to correct bugs related to your COTS software may differ from typical repair costs significantly because of the lack of available source code and a related lack of understanding of exactly what's going on inside. Additionally, you may identify bugs in the COTS software component that the vendor is unable or unwilling to fix for which you will be required to develop workarounds in the glue code or with modifications.

Cost Drivers— The primary cost drivers for this activity are:

- Functional Size
- Amount of Modification
- Functional Complexity
- Operating Specification
- Project Constraints
- Integration Team Complexity
- Number of Upgrades Expected
- Glue Code Size
- Glue Code Language
- Vendor and Product Complexity
- Length of time software will be deployed

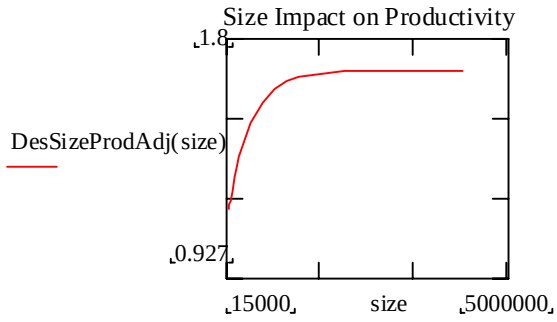
5. COST DRIVERS

This section describes each of the cost drivers mentioned above and shows how these cost drivers are applied to impact the productivity of the various COTS activities. Productivity in this context is measured as hours per size unit, so increasing relationships actually represent a decrease in productivity. Because of the complexity of some of the relationships, these productivity impacts and the proprietary nature of some of the relationships they are provided in the form of charts rather than the actual relationships. More detail on the non-proprietary relationships can be found in [10].

Functional Size

Software size measurement continues to be an issue, whether the software is being built in house or is purchased off the self. The best way to measure this size continues to be an issue. Sometimes a measure like source lines of code (SLOC) makes sense, especially if the source code is available. Even when this is the case, it is not always a desirable measure. Because the software is off the shelf the total size may not be a good measure of the amount of functionality that is actually being utilized in the system being developed. Function Points are often better because they represent the actual functionality being implemented but Function Point counting is problematic because it is subjective and hard to automate. In our research we developed a measure called Functional Size that relates to the 'amount' and types of functionality being implemented. Functional size provides a way to consistently quantify the amount of functionality being integrated with COTS software. Through the calibration process we were able to relate this measure to the cost of evaluation, modification, tailoring and integration activities. Although the impact of size on productivity varies for somewhat for each of the activities, the basic relationship is represented in Figure 1.

Figure 1



Functional Complexity

This is a quantification of the complexity of the functions that are being integrated into the system. This driver basically asks the question 'On a scale from 1 to 10, how complex is the functionality?' The values for functional complexity were determined by studying a set of projects with similar characteristics, which delivered functionality in different categories. As it is impossible to find two projects that are alike in all ways but one, expert knowledge supplemented actual data for this analysis. Clearly software that implements simple calculator functions is less complex to modify, integrate and test than software doing complex real time functions. Figure 2 shows how functional complexity impacts project productivity. Figure 3 shows the rankings for some typical applications.

Figure 2

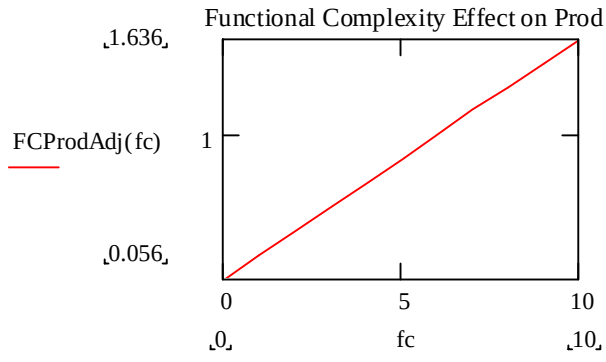


Figure 3

Application type	Functional Complexity
Accounting Package	3.02
Customer Relationship Management Systems	3.70
Data Processing	3.05
Database Systems	3.95
Encryption Applications	7.72
Expert or Decision Support Systems	4.14
Fleet Management	3.97
Flight Management	6.20
Guidance Control	7.85
Health Monitoring Systems	4.52
Human Resource Applications	3.89
Imaging, Sensing and Mapping	4.64
Management Information Systems	3.56
Material Requirement Planning	3.18
Office Automation	3.68
Operating System - Command Line Interface	7.41
Operating System - Graphical User Interface	7.60
Purchasing/Inventory Control	3.80
Satellite Data Link	7.09
Telecommunications	5.95
Text Processors	2.89
Weapon Control	7.76
Weapons Management	7.58

Amount of Modification

This metric simply refers to the percent of the functionality that will need to be modified during the integration process in order to make the off the shelf components meet the requirements for the system being developed. This percentage is applied to the Functional Size to quantify, along with the amount of glue code being developed, the design, code and test activities.

Operating Specification

This driver quantifies the complexity added to the project based on the environment where the software is expected to operate. The values for complexity were determined by studying a set of projects with similar characteristics which were designed to operate in different environments and by studying the demands of these various environments on software productivity in general. As it is impossible to find two projects that are alike in all ways but one, expert knowledge supplemented actual data for this analysis. Figure 4 shows the impact of Operating Specification on productivity. Figure 5 shows Operating Specification rankings for typical environments.

Figure 4

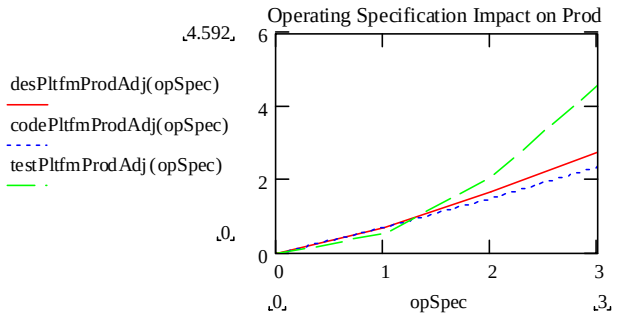


Figure 5

Environment	Operating Specification
Commercial Proprietary Software - Low Reliability	0.70
Commercial Proprietary Software - Nominal Reliability	0.80
Commercial Proprietary Software - High Reliability	0.90
Commercial Production Software - Nominal Reliability	1.00
Commercial Production Software - High Reliability	1.20
Commercial Production Software - Airborne	1.70
Military Software - Ground	1.20
Military Software - Mobile	1.40
Military Software - Airborne	1.80
Space Software - Unmanned	2.00
Space Software - Manned	2.50

Occasionally, the COTS software component will be developed for an Operating Specification different than the one the software system is being developed for. When this occurs, additional qualification activities may need to occur to ensure that it meets the requirements of a higher Operating Specification. For this reason our model looks at both the Delivered Operating Specification and the Target Operating Specification and adjusts productivities for certain activities accordingly.

Glue Code Size

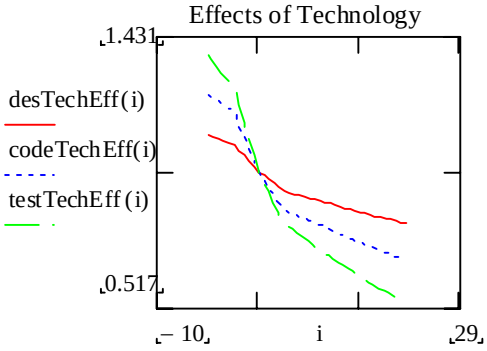
In addition to making modifications to the delivered COTS software, it is often necessary to develop glue code to build bridges between COTS software within the system or between the COTS software components and other components developed in the system. Glue code also may be required to work around limitations in the COTS software. This size is measured using standard software size measures such as Source Lines of Code or Function Points.

Glue Code Language

Because different programming languages and development tools have different inherent productivities, this is an important cost driver, particularly in cases where a great deal of glue code development is expected. The language is an important productivity driver for two reasons. The first is the inherent productivity of a

programming language based on features of the language itself and features of the development environment that delivers it. The second relates to the maturity of that language. Figure 6 gives a generalization of this technology improvement productivity relationship. More details on the productivities of specific programming languages can be found in [10].

Figure 6



Number of COTS components to be evaluated

Generally, there is more than one off the shelf component that delivers the functionality, or some facsimile of the functionality being relegated to COTS software components. Evaluating which one is best for this particular software project is often a non-trivial exercise. The number that will be evaluated is a cost driver for the evaluation activity

Number of components for detailed evaluation

Ideally evaluation occurs in two steps. Many components are evaluated broadly to determine the best candidates for further evaluation. Then several are selected for a more detailed evaluation. This number indicates the number to be selected for this detailed evaluation.

Evaluation Multiplier

Research shows that the functionalities that the COTS software component will be delivering is a good predictor of the cost and effort for the evaluation activity. The selection of a value for this factor is based on an evaluation of the different types of functions the COTS software components are delivering to the software system as well as the method selected for evaluation. This factor adjusts the amount of evaluation required per functional size unit. The productivity for the evaluation is a function of the number of components being evaluated, the number of detailed evaluations expected and the evaluation multiplier. Figure 7 details this relationship.

Figure 7
Evaluation Productivity

$$InitialEval\ Prod = EvaluationMultiplier * CompUnderEval * desPltfmProdAdj$$

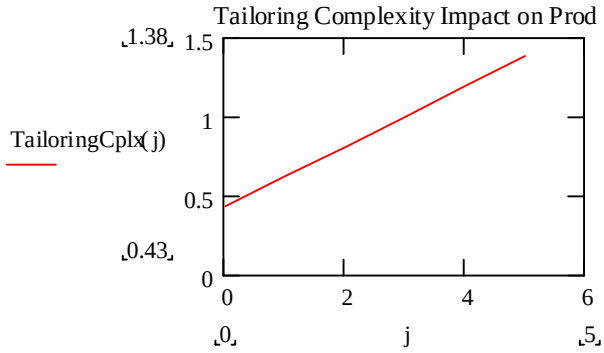
$$DetailedEval\ Prod = 3 * EvalMultiplier * CompUnderEval * \frac{PercforDetailedEval}{100}$$

$$DetailedEval\ Prod = DetailedEval\ Prod * (1 + NbrUpgrades * 0.2)$$

Tailoring Complexity

This is a quantification of the nature of the tailoring that will be required to get the COTS software functional within the context of the software being built. This value is based on the amounts and types of tailoring that will be required along with the amount of support that can be expected from the documentation and the vendor(s). This is a value from 1 to 5 with 1 indicating a very limited tailoring exercise and 5 a very complex one. Figure 8 shows the relationship between tailoring complexity and productivity

Figure 8



Tailoring Multiplier

Research shows that the functionalities that the COTS software will be delivering are a good predictor of the cost and effort for the tailoring activity. The selection of a value for this factor is based on an evaluation of the different types of functions the COTS software components are delivering to the software system. This input affects the amount of tailoring that occurs per functional size unit. The productivity impact for the tailoring activity is a function of the tailoring complexity, tailoring multiplier, vendor complexity and number of upgrades. Figure 9 shows the basic relationship for this effect.

Figure 9
Tailoring Productivity

$$Tailor\ Prod = TailorMult * TailorComplexity * VendorComplexity$$

$$Tailor\ Prod = Tailor\ Prod(1 + NbrUpgrades * 0.25)$$

Vendor and Product Complexity

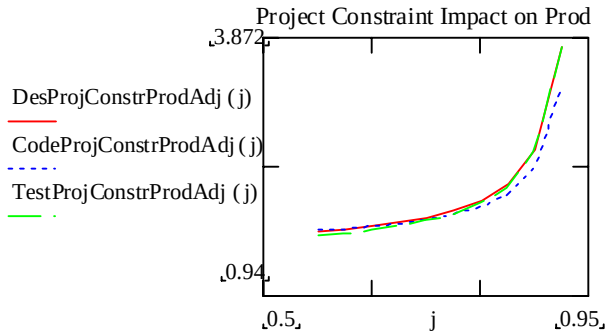
This is a quantification of productivity affects of the stability and maturity of the COTS software products used and the support and cooperation expected from COTS vendors. The vendor and product complexity impacts the

productivity of the tailoring activity as well as the design, code and test activities.

Project Constraints

This driver is a quantification of memory or timing constraints imposed by requirements of the software system. Figure 10 shows how project constraints impact productivity.

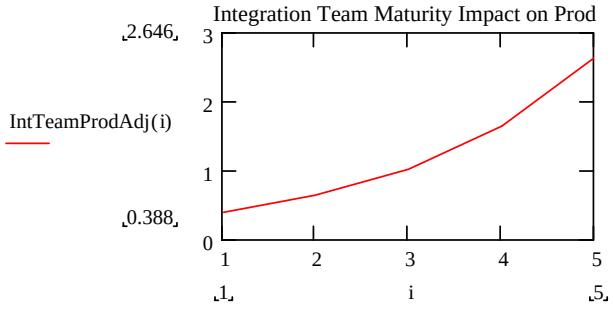
Figure 10



Integration Team Maturity

This cost driver indicates the benefits or costs of the level of experience and knowledge that the integration team has with the COTS software products, the software system being built and the COTS integration process. This cost driver applies to the integration that occurs at the component level as well as integration to the system. This driver is a value from 1 to 5 with 5 being a very immature team and 1 being a very experienced one. Figure 11 shows how integration team maturity affects productivity.

Figure 11



External Integration Complexity

This cost driver ranks the complexity or simplicity of the integration of specific components into the complete system. This depends on the experience of the team with the individual components as well as the number of points of interface a particular component has with the system.

Number of Upgrades during development

If the development of the software systems spans more than nine months, it is quite likely that the COTS software may be upgraded by the vendor. When this occurs it may be necessary to perform additional evaluation activities to determine whether or not this upgrade should be used rather than the component delivered initially. If it is to be incorporated, additional development and integration tasks may be required to keep the system functional or include additional features supplied with the upgrade.

6. Cost Estimating Relationships

Armed with an understanding of all the relevant activities along with knowledge of cost drivers and a means for measuring these cost drivers, the next step in building a cost model involves the development of cost estimating relationships. These come from data collection and analysis, heuristics and expert knowledge.

The approach taken was a rather simple approach. For each activity a 'standard' productivity was established by asking the question 'For an average organization with average people and processes performing an average COTS integration how many hours per unit of Functional Size is spent?' Then, based on the data collected for each of the cost drivers, how does this cost driver improve or inhibit that productivity. Additional analysis was required for those cost drivers that in combination impact productivity differently than when viewed alone. From this process a relationship of this general form evolved:

Effort = Size*BaselineProd*ProdAdjustments

Specific details of these relationships and how they apply to each of the activities can be found in [10]. The data set that was used to derive these relationships contains a fairly balanced representation of aerospace and defense projects and military and commercial information systems and business applications. Since the data set contains such a rich mix of projects, the baseline productivities really do represent a 'typical' organization. Clearly, an organization wishing to use such relationships would get the best results by calibrating their own organization's productivity for activities then applying the productivity impact relationships to that.

7. Other Costs to Consider

Other costs may be associated with COTS integration that bear consideration when doing a cost/benefit analysis of a COTS solution:

1. Costs to keep COTS source code in escrow
2. Changes in license or service fees

Costs to keep COTS code in escrow
Depending on the type of software system you are building, there may be a requirement that a copy of the code for the COTS solution be held in escrow in case the vendor goes out of business or drops support of the COTS solution during the lifecycle of your product. Many vendors charge for this service.

Changes in license or service fees
Often changes in the product, vendor ownership, or the market place will result in changes in the way a vendor determines licensing and subscription fees. This is generally not something that can be anticipated during the selection process but it is an eventuality that should be recognized and understood in the context of the software system under construction.

8. CONCLUSION

There is much evidence that a well thought out, well implemented COTS solution will improve the speed and reduce expenses of your software development projects. There is further evidence that these benefits are often not as great as most of us would like them to be. It is certainly tempting to think, when planning a project, that the reports and charts will be a breeze because they will all be implemented with COTS software. Care should be taken when planning a project to consider all of the activities that should be performed and to schedule time and resources to complete these activities adequately. It is also important to look at the long term issues associated with the solutions you select.

The relationships developed through this research have fared well against our data sets. Results are within 20% of actual costs for more than 80% of the data studied when comparing at a system level. Because data collection varies from organization to organization, putting different costs in different buckets – or not even collecting data at an activity level, comparisons at this level are often problematic.

This research has resulted in the development of a cost model that can help project planners and estimators plan for software projects that include the integration of some or many COTS software. Just as valuable, it has resulted in the development of a repeatable thought process that a project planner should follow each time they attempt to estimate the costs of their software projects, considering each of the activities outlined above and quantifying the impact that their particular process, people, and projects will have on the overall productivity of their development and integration efforts.

8. REFERENCES

[1] Richard Stuzke, "Costs Impact of COTS Volatility", *Knowledge Summary: Focused Workshop on COCOMO 2.0*,USC Center for Software Engineering, May 16-18, 1995

[2] Randall Jensen, "Estimating the Cost of Software Reuse", *Crosstalk*, May 1997, pp17-21

[3] T. Ellis, "COTS Integration in Software Solutions – A Cost Model", *Systems Engineering in the Global Marketplace*, NCOSE Symposium, St. Louis, MO July 24026, 1995

[4] University of Southern Californai, Center for Software Engineering, "CoCOTS Whitepaper", (available at http://sunset.usc.edu/research/COCOTS/cocots_main.htl, June 1997

[5] Christopher Abts, "COTS Software Integration Cost Modeling Study", University of Southern California, Center for Software Engineering, June 1997

[6] Vigder, et.al. "COTS Software Integration State of the Art, Natioanl Research Council, Canada, Software Engineering Group, January 1996

[7] Brownsword, L, et.al., "Lessons Learned Applying Commercial Off-the-Shelf Products", Software Engineering Institute, Carnegie Mellon, June 2000 (available at www.sei.cmu.edu)

[8] Carney, D., "Assembling Large Systems from COTS Components: Opportunities, Cautions and Complexities", Software Engineering Institute, Carnegie Mellon University, June 1997 (available at www.sei.cmu.edu)

[9]Oberndorf, P., et al, "Workshop on COTS-Based Systems", Software Engineering Institute, Carnegie Mellon University, November 1997, (available at www.sei.cmu.edu)

[10] Minkiewicz, A, "True S White Paper", PRICE Systems, L.L.C., November 2003, internal document available on request, www.pricesystems.com

Arlene Minkiewicz *is the Chief Scientist for PRICE Systems, L.L.C. In this role, she leads the effort to provide Cost Research in support of PRICE's entire suite of cost estimating products, including the PRICE Hardware and Software Models and the TruePlanning Suite of Products. Prior to this assignment, Ms. Minkiewicz functioned as the Director of Product Development, responsible for the maintenance and enhancement of the PRICE products. During her 19 years with PRICE she has become a subject matter expert in the area of software cost estimating and software measurement and has developed a software cost estimating model based on an object-oriented software measurement system she invented. Ms. Minkiewicz has a BS in Electrical Engineering from Lehigh University and an MS in Computer Science from Drexel University.*

1-856-608-7222, Arlene.Minkiewicz@PRICESystems.com
17000 Commerce Parkway, Suite A, Mt. Laurel, NJ 08054