Six Steps to a Successful COTS Implementation

Arlene F. Minkiewicz
PRICE Systems

A successful implementation of a commercial off-the-shelf-intensive software system can save programs money if you have the right solution and understand the potential risks involved.

Federal organizations are relying more and more on commercial applications to supplement, enhance, or replace proprietary systems. This dependency is driven by the promise of improved functionality and reduced total ownership cost, as well as concern over the lack of capability to develop and maintain proprietary information technology applications. However, failure to successfully select, control, and implement these critical components continues to result in projects that are delivered late and over-budget or that fail entirely.

The following *six-step methodology* highlights the important activities that should take place during a commercial off-the-shelf (COTS) implementation. Following this methodology throughout the software development life cycle will ensure that significant activities are not being ignored and will increase the chances of planning, executing, and deploying a successful COTS-based software solution.

One of the biggest problems sighted in COTS-based projects is a disconnect between time and cost expectations during planning and those actually realized.

During the planning stages, it is important to plan appropriately for all the major activities necessary to devise a well thought-out solution that will not fall apart with the first upgrade of one of its components. Research [1, 2, 3, 4, 5] has indicated the essential activities that must take place to ensure successful COTS-based projects:

- Analyze software requirements.
- Evaluate and select COTS solution(s).
- Negotiate terms with COTS vendor.
- Implement the COTS-based solution.
- Maintain and upgrade the COTS-based solution.
- Maintain license, subscription, and royalty fees.

Figure 1 portrays an overview of the six steps outlined above and highlights the interactions that may occur throughout the execution of these steps. While this diagram implies a time dependency between these steps, it is important to realize that in certain cases this is neither strictly adhered to nor are all the steps necessarily performed by the organization(s) contracted to deliver a solution. Some requirements analysis and COTS evaluation are likely to occur in very early stages of a project as feasibility and affordability are analyzed.

The following sections provide more details about each of these steps, along with a brief description of the factors to consider when evaluating the affordability and timeliness of a COTS-based solution. Specifics about the quantification and application of these factors can be found in [6].

1. Analyze Software Requirements

Software requirements analysis is a critical part of the software development process, although too often this activity is overlooked or glossed over in the rush to start *building software*. The requirements analysis process is necessary to determine what functionality is necessary to deliver the capability required by the eventual end-user(s).

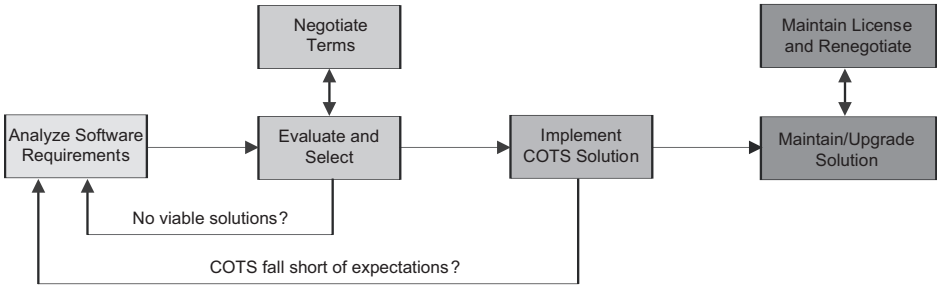
There are two general areas that need to be explored when determining and documenting requirements for a software system: end user requirements and technical requirements. The discovery process for end-user requirements involves business analysts or requirements engineers asking the end-user what they expect from the software. Once end-user requirements have been gathered, an important next step is for the business analysts or requirements engineers to restate those requirements and present them back to the end-user to ensure proper understanding. Technical requirements can be gathered through discussions with engineers who understand the technical nature of the problem being solved.

The question of whether or not COTS solutions are a viable alternative becomes an important factor during the software requirements analysis activity because the software requirements drive the selection criteria for potential COTS solutions. This being said, it is also important not to let the availability of COTS solutions cloud the analysis by obscuring requirements.

Solution providers should be aware that it is unlikely that any COTS solution will be available to satisfy all the requirements for a software system. The requirements analysis process should identify which requirements are the *must have* requirements and which can be somewhat *bendable*. During the evaluation, and possibly during implementation, tradeoffs will be necessary to compensate for functionality not available in COTS solutions (or promised and not delivered with a chosen COTS solution). Decisions should be made during the requirements analysis activity to determine which functions can be subject to such tradeoffs and which cannot. If most or all of the software requirements are determined to be *must haves*, it is wise to revisit the decision to pursue a COTS-intensive solution.

Although the focus of this article is on COTS software solutions, it is important to mention here that when entire systems are being constructed, the COTS decision may need to be visited even before soft-

Figure 1: *Overview of the Six Steps*



ware requirements analysis commences. During the analysis of system requirements, decisions may be required to determine whether certain functionality should be addressed with hardware or software. The availability of software COTS solutions could be a factor in the determination of the affordability of such tradeoffs.

As with all activities in a software development process, successful execution of the requirements analysis process takes time and effort. The major driver in determining time, cost, or effort for the software requirements analysis activity is a measure of the amount of functionality the software system is intended to deliver. The *measurement* of software size is always a challenge. A measure that quantifies functionality delivered such as function points or analogies to known systems is best.

Whether the plan is for that functionality to be delivered primarily with COTS solutions, newly developed solutions, or some combination of the two, the time and effort devoted to requirements analysis should be fairly consistent. The technical complexity of the functionality as well as the software's operational platform will also drive the requirements analysis effort because more complex solutions require more time to understand and communicate. Additionally, the presence of project constraints, timing, memory, or schedule will impact the effort required for this activity.

2. Evaluate and Select COTS Solution(s)

Once a decision to pursue a COTS alternative is made, the first step is to determine the availability of COTS solutions that have the potential to provide needed functionality, then evaluate these solutions. The main reasons to evaluate are the following:

- 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 can operate successfully in the environment(s) where the system is intended to operate.
- Determine whether a proposed set of components can operate successfully in an integrated fashion.
- Determine the stability and viability of the vendor.
- Determine the willingness of the vendor to cooperate and help make the project successful.

The evaluation needs to be focused on more than just product characteristics such as functionality, maturity, technology, architecture, and long-term viability. There should also be a focus on vendor characteristics such as maturity, stability, cooperation, and ability to provide adequate support, training, and documentation. The evaluation should also be used to ensure that there are no compatibility issues associated with using COTS solutions from multiple vendors.

The selection process is often a combination of the following three evaluation techniques:

- **Progressive filtering of available COTS components.** This requires several iterations of filtering, each one going into progressively more detail until a single solution or set of solutions emerges as the best answer.
- **Puzzle assembly process.** This approach suggests that 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.
- **Identifying the keystone COTS software components.** This is identifying those components 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 the keystone components.

"When performing a COTS evaluation, it is valuable to obtain references from the COTS vendors in an effort to speak with developers and end users who have worked with a particular COTS solution."

A commonly cited challenge by system integrators is that COTS products often fail to deliver the functionality or other requirements promised during evaluations. It is prudent to get some hands-on time with the components being evaluated where possible; this may be problematic as it most likely requires a great deal of cooperation and support from both the vendor and the integrating organization.

When performing a COTS evaluation, it is valuable to obtain references from the COTS vendors in an effort to speak with developers and end users who have worked with a particular COTS solution. This not only provides valuable feedback about vendors' responsiveness and dependability, it also aids the planning process by highlighting problems or pitfalls other integrators may have experienced.

The evaluation and selection activity not only facilitates identification of available COTS solutions, it also points to those pieces of functionality that cannot be satisfactorily implemented by existing off-the-shelf solutions. An important byproduct of this investigation may be an examination of the cost, schedule, and effort associated with developing custom code to make up for required functionality missing in COTS solutions. This examination may require revisiting the cost/benefit analysis leading to the decision to build a COTS-based solution.

Generally, the time and effort devoted to the selection and evaluation activity is often based on a predetermined level of effort. The determination of this level of effort should consider the amount of functionality being implemented with COTS solutions (based on functional size or analogy), the number of solutions that will be evaluated, the type(s) of evaluations being performed, and the number and criticality of evaluation criteria.

3. Negotiate Terms With COTS Vendors

Certainly it is important to negotiate the best deal possible when working with one or more vendors to craft a solution. It is even more important to understand the impact of these negotiations and their timing on the eventual success or failure of your project. Several of the most commonly cited challenges of those building software solutions with COTS components involve vendor forthrightness and cooperation [4].

Vendors are much more likely to address customer concerns with missing

or incomplete functionality and/or bugs in the software before signing on the dotted line. During the negotiation process, it is important to address and resolve any known issues and establish expectations for issues that emerge during the integration process and throughout the product life cycle. Clearly, the size of the vendor and the size of the purchase may be factors in determining how demanding any particular customer can be, but it is important to set expectations with all of the project stakeholders.

The end result of this negotiation should be a clear picture of the non-recurring and recurring costs associated with the system being developed. A nonrecurring cost is a one-time fee associated with acquiring the COTS solution such as the purchase price of shrink-wrapped software. Recurring costs are those generally based on usage or time of use such as annual licensing fees. Negotiations should also result in a common understanding between parties of update and upgrade policies, as well as expectations with regard to vendor responsiveness and cooperation. It may also be necessary during this step to develop a plan with the vendor to ensure that maintenance of the deployed solution be possible even if the vendor goes out of business. This plan generally involves the escrowing of source code to be made available only under terms of the agreement such as bankruptcy or company demise.

The cost and effort drivers for this activity should be broken into two parts. The first part is the actual dollar value that is determined for delivery of the COTS component after negotiations and any promised royalties and other fees. The second part relates to the amount of time and resources that must be devoted to the negotiation with the vendor.

4. Implement the COTS-Based Solution

Once analysis, evaluation, and selection of a COTS-based solution are complete, implementation can commence. The following activities may be required to ensure successful implementation.

Tailoring of a COTS Solution

There are certain necessities that should be performed in or around software to get the COTS software components configured for the system and its requirements. 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 script development may be required.

Although these activities are unlike traditional coding exercises, they do take time and effort to complete. The results of these activities require testing and verification. These tasks also require a significant level of understanding as to how the COTS component(s) work and how to work with them. This requires reading manuals and/or attending training and then experimenting with the actual components to reach a level of competency.

The major cost, effort, and schedule drivers for the tailoring activity include the amount and complexity of scripts, database parameters, reports, and screens being customized. Additionally, as security and access control requirements increase in rigor, they will drive up time and cost. It is important also to consider the ease with which the COTS solution can be understood, the quality of training and documentation, and the integration team familiarity with the COTS solutions being used and the system being implemented.

Modification of COTS Software

Generally the definition of COTS software precludes modifications because COTS software does not have source code available. This is the case when the COTS software is a shrink-wrapped commercial product. Sometimes solutions call for the integration of a series of off-the-shelf components that do not meet this traditional definition of COTS but rather are components with source code available that are either furnished by the customer or otherwise obtained. While these projects are not strictly *COTS* projects, they do happen and are mentioned here for completeness.

It would be nice if the COTS software completely satisfied all the functional requirements it was selected to meet, but this is often not the case. In situations where the source code can be made available, the project team needs to make a determination whether or not modification is an option. It is generally a bad idea to make modifications to COTS software because this negates much of the productivity increase obtained from using COTS components and is likely to jeopardize any likelihood of supplier maintenance of the COTS components. If COTS modifications are being considered, care should be taken to ensure that these modifications are very modular in nature. The developers need to learn a great deal about the architecture and basic structure of the solution before any modifications can be made. It is important to understand that the project productivity hit can be substantial even when the slightest modifications are made to a COTS component.

The major cost, effort, and schedule drivers for modifications to COTS software are the same factors that drive costs for any software modification project (functional size, extent of modification, technical complexity, eventual operating platform, productivity, and efficiency of development organization). These factors must then be augmented to account for unfamiliarity of the COTS solution code, quality of COTS training and documentation, vendor cooperation, development team experience with COTS solutions, and the COTS integration process. It is also important to include maintenance of the entire COTS component in the affordability analysis as once modifications have been made it is unlikely that the COTS vendor will continue to support their solution.

Design, Code, and Test of Glue Code

Glue code literally holds the system together. Glue code is any code that needs to be written to make the COTS software components function as advertised and/or as required. 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 data to the format required by the COTS component. In a well-written application, the glue code acts as a layer between the system and COTS components, encapsulating the data in such a way that upgrades and replacements are as painless as possible. Finally, glue code is sometimes required to add functionality that the COTS software component implements inadequately or that should be provided by the COTS component but is not.

These development activities are complicated due, primarily, to unavailable source code. The complexity of glue code development is akin to a situation where an entirely new team of developers is brought in to integrate custom built components, but is given limited or no access to the original development team and the source code. The unfamiliarity of the interfaces, along with the inability to debug the components, adds complexity to the development exercise.

Another factor that makes glue code development differ from traditional code development is the complete reliance on vendors to fix bugs when they are discovered, ensure that upgrades are upwardly

compatible, release stable products, and fill in where documentation falls short. Bug fixing can be particularly problematic in a COTS project, especially when there are multiple vendors involved or when modifications have been made to the COTS components. With multiple vendors, especially with unavailable source code, the integrator must rely on the vendors not only to fix the bugs but also to cooperate with each other in the identification of where a bug actually resides. When the COTS components have been modified, the integrator often must struggle to convince the vendor that the bug is in the original code and not a side effect of changes they have made. Careful modularization and documentation of any COTS modifications may help alleviate this problem.

As with modifications to COTS components, the major cost, effort, and schedule drivers for glue code development are not unlike those drivers associated with any custom development, but the productivity of the development team needs to be adjusted to account for unfamiliarity and unavailability of COTS components along with the requirement for vendor support and cooperation in solving integration problems. As the number of different COTS components and the number of different vendors involved increases, so increases the complexity (thus effort and schedule) of this activity.

Integration and Test of COTS Components With Other COTS or Custom Components

The system needs to be integrated and tested to ensure that all functional and non-functional requirements are met. This activity tests the entire system (or subsystem) as a complete unit, verifying that there are no system-level problems associated with incompatibilities or competing demands of components for limited resources.

Requirements related to performance, reliability, and security could be particularly problematic during this activity 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.

A well-designed integration and test approach would focus early releases on those areas that are high risk or most likely to lead to incompatibilities. If multiple COTS components from multiple vendors are being integrated, it is important to have all of them running together at the same time as early as possible, even if each is functioning in very limited capacity with respect to its intended feature set. This helps identify potential contention and incompatibilities between components.

When assessing the effort and schedule for glue code development, tailoring, or integration activities, an important factor is the update cycle for the COTS solutions being integrated. If updates/upgrades are frequent, additional time and effort may be required to evaluate and possibly incorporate these updates.

"It is generally a bad idea to make modifications to COTS software because this negates much of the productivity increase obtained from using COTS components and is likely to jeopardize ... supplier maintenance ..."

Upgrades and updates may be included if they contain required bug fixes, improvements to keep the look and feel current with user expectations, or features that relate to missing or incomplete requirements in earlier versions. Vendor quality and stability, along with vendor(s) regular release schedules, are important factors in assessing effort associated with updates and upgrades.

In general, the effort, cost, and schedule for many system-level integration activities is likely to be *higher* for COTS-inclusive software than software that is composed of custom-built components because of the unfamiliarity with the code. The COTS software component should be viewed by the integrator as a *black box*. The factors that drive the effort and schedule for integration activities include the amount of functionality being integrated (including functionality provided by homegrown development as well as that coming from the COTS components), the complexity of the functionality, the operating platform for the system, glue code size, and vendor-related issues such as cooperation, support, and upgrade policies.

5. Maintain and Upgrade the COTS-Based Solution

Once the software is deployed, the following two ongoing activities are required to keep it operational and keep end-users happy.

Evaluation and Inclusion of Updates and Upgrades From the Vendor

There are countless reasons why the inclusion of updates and upgrades are desirable once the product is deployed. If there are bugs in the COTS software that affect system operation, then an upgrade is obviously needed. Beyond this, the vendor should be upgrading the product to keep up with rapidly changing technology. To maintain a software system that meets market expectations with respect to performance, look and feel, operating platforms, etc., updates of the COTS software components will be likely.

Whether including an update or upgrade, each refresh of the COTS components poses potential risk. Assessment is required with each release to determine whether it makes sense to include it in the software system. Sometimes upgrades change the interfaces to the COTS software components so that with the upgrade, existing functionality ceases to work correctly, requiring a rewrite of some of the glue code. Vendors find that in order to offer new features or to keep up with technology, they must change interfaces and databases or rework functionality. To get access to the new features and technology in an upgrade, the software developer may be forced to accept changes he or she does not 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 the existing software system, other COTS software components, or even the operating platforms on which the system runs. 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 of 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 will discontinue support of older versions of

the COTS software.

The major factors that drive cost, effort, and schedule for the evaluation of COTS upgrades include the amount of functionality delivered by COTS solutions, the number of COTS solutions in the system, the upgrade/update frequency of the vendor(s), and the quality and stability of the COTS solutions. When (and if) a decision has been made to include upgrade/updates of COTS solution(s), the drivers for the integration and test are the same as those for integration and test during development, although the extent of effort should be tempered by the extent of functionality changes in the upgrades/updates.

Bug Fixes

Software, whether it is homegrown or acquired externally, is likely to have defects. Bug fixing efforts for COTS-intensive systems will differ significantly from typical repair efforts. Additionally, bugs may exist in the COTS software component that the vendor is unable or unwilling to fix for which workarounds in the glue code need to be developed.

The effort associated with bug fixes for COTS software, as with any software, is a direct function of the quality of the initial software offering, as well as the quality with which bugs are fixed. This quality is generally a function of the amount of functionality, the complexity of the system, and the development processes and practices employed by the initial software developer. For COTS software, some of these factors are apparent and some are hard – if not impossible – to find out. Also, how and when the bugs in the COTS component get fixed is for the most part out of the integrators' control.

These factors complicate the process of planning for maintenance of COTS-based systems. Additionally, the maintenance process is plagued with the same issues cited earlier for the glue code design, integration, and test: It is not always obvious where the bugs actually occur. Despite the hurdles mentioned, it is still possible to plan proactively for the maintenance of a COTS-based system based on what is known. Functional size of the entire system (including not just COTS components but homegrown components as well), system complexity, operating platform, glue code size, amount of modification, and maintenance team productivity can be used to baseline the effort. This effort should then be adjusted for the loss of productivity associated with debugging through *black box* code and interfacing with multiple vendors.

6. Maintain License, Subscription, and Royalty Fees

License or maintenance fees need to be paid in order to ensure updates and upgrades as well as continuing support of the COTS components. It is important to understand vendor(s) upgrade policies. It is also wise to do a long-term analysis of the differences between annual subscription fees (if subscription is an option) versus paying for upgrades on an individual basis. This analysis should include upgrade policies, vendor stability, and frequency of releases. License and royalties should be an important part of the initial negotiation process. Renewal periods are an opportunity to revisit the terms of the negotiation to determine whether the vendor is meeting support and upgrade commitments.

Conclusions

A well thought-out and well-executed software project that incorporates one or many COTS solutions can happen more quickly and be more cost effective than the same system implemented with custom developed components. Too often, COTS projects are not thought out or planned, running on the incorrect assumption that every COTS solution is a small integration project without the issues and complexities cited above. This way of thinking leads to unrealistic and poorly managed expectations, resulting in failed projects. These types of failures occur when projects fail to plan for or incorporate the additional activities unique to COTS-intensive developments. Following this *six-step methodology* will ensure that important activities and decision points are properly executed, reducing many of the risks associated with such developments.◆

References

1. Ellis, T. "COTS Integration in Software Solutions – A Cost Model." INCOSE Symposium, St. Louis, MO, July 1995.
2. Center for Software Engineering. "COCOTS." Los Angeles, CA: University of Southern California, June 1997 <http://sunset.usc.edu/research/COCOTS/cocots_main.html>.
3. Abts, Christopher. "COTS Software Integration Cost Modeling Study." Los Angeles, CA: Center for Software Engineering, June 1997.
4. Brownsword, L., et al. "Lessons Learned Applying Commercial Off-the-Shelf Products." Pittsburgh, PA: Software Engineering Institute, June 2000 <www.sei.cmu.edu>.
5. Oberndorf, P., et al. "Workshop on COTS-Based Systems." Software Engineering Institute, Nov. 1997 <www.sei.cmu.edu>.
6. Minkiewicz, A. The Real Costs of Developing a COTS-Based System. Proc. of IEEE Conference on Aerospace and Defense, Big Sky, MT., Mar. 2004.

About the Author

Arlene F. Minkiewicz is chief scientist of the Cost Research Department at PRICE Systems. She is responsible for the research and analysis necessary to keep the suite of PRICE estimating products responsive to current cost trends. In her 20-year tenure with PRICE, Minkiewicz has researched and developed the software cost estimating relationships that were the cornerstone for PRICE's commercial software cost estimating model, ForeSight, and invented the Cost Estimating Wizards originally used in ForeSight that walk the user through a series of high-level questions to produce a quick cost analysis. As part of this effort she has invented a sizing measurement paradigm for object-oriented analysis and design that allows estimators a more efficient and effective way to estimate software size. She recently received awards from the International Society of Parametric Analysts and the Society of Cost Estimating and Analysis for her white paper "The Real Cost of COTS." Minkiewicz contributed to a new parametric cost estimating book with the Consortium for Advanced Manufacturing – International called "The Closed Loop: Implementing Activity-Based Planning and Budgeting," and she frequently publishes articles on software estimation and measurement. She has also been a contributing author for several books on software measurement and speaks frequently on this topic at numerous conferences.

17000 Commerce PKWY STE A
Mt. Laurel, NJ 08054
Phone: (856) 608-7222
Fax: (856) 608-7247
E-mail: arlene.minkiewicz@pricesystems.com