Selection, Evaluation, Testing, Integration and Implementation of Commercial-Off-The-Shelf(COTS) components in Software

N. Gnanasankaran
MCA Department
KSR College of Arts & Science
Tiruchengode – 637 21, India, sankarn.iisc@gmail.com

K. Iyakutti
Department of Physics & Nanotechnology
SRM University, Kattankulathur
Chennai – 603 203, India, iyakutti@gmail.com

S. Natarajan
School of Physics, Madurai Kamaraj University
Madurai – 625 021, India, s_natarajan50@yahoo.com

K. Alagarsamy
MCA Department, Madurai Kamaraj University
Madurai – 625 021, India, alagarsamy.mku@gmail.com

Abstract

Commercial-Off-The-Shelf(COTS) technology is widely used in many software companies and also in scientific computing. The definition of COTS is furnished. Its virtues and skills needed in using COTS SW are detailed. The methodology used for the selection and evaluation of COTS is given. Some information about the testing of COTS components is also dealt with. The importance of integration and composition of the components is stressed. The technique of two-layer wrapping for integration is also touched upon. The methodology for a successful COTS implementation is given in detail. The “six-step methodology” of Minkiewicz is suggested for successful COTS implementation.

Keywords: COTS, Definition, Selection, Evaluation, Testing, Integration, Composition, Implementation.

1. Introduction

The cost of software development can be significantly reduced by fostering a software component industry. In the past, most software were marketed as complete applications; today we are moving toward an economy where it is hoped that components can perform dual purposes and thus be sold as stand-alone entities. When software components can be marketed in this manner, they are referred to as “Commercial-Off-The-Shelf” (COTS).

It’s interesting to note that more than 99% of computer instructions come from COTS products[1]. It’s inevitable that your SW for any activity will involve COTS components. Your project team simply cannot write every component needed to make an effective SW-related decisions and analyses. But, your evaluation of COTS is important so as to minimize risks and maximize benefits within the scope of your projects.

Software practitioners today are very familiar and comfortable with custom system development. However, fewer are familiar with COTS-based system development involving the purchase and integration of a dozen or more COTS products that provide system functionality, in which the only custom software is that necessary to “glue” the pieces together.

This scenario calls for a different philosophy and process from that familiar to those who acquire systems the traditional way, using custom development. It requires new understandings about the COTS marketplace and how all engineering, business, and management activities must work together harmoniously to accommodate it. There are new rules of engagement that must be recognized and observed, and those who fail to appreciate the differences risk disappointment with COTS-based systems. The new rules flow both from the definition of a “COTS product” and from the consequences of assembling things from
purchased parts. The new rules apply to all COTS-based systems, whether they consist basically of one large product or product suite (called COTS-solution systems) or are made up of many COTS products from a variety of vendors (called COTS-aggregate systems).

Due to our interest in studying COTS components and their usage, the application of COTS components in crystallographic software was evaluated by us earlier [2]. In another contribution, the case of a molecular modeling software, viz., GROMACS was taken up as a case study [3-4]. In another paper, one of the software used for mathematical computations, viz., SCILAB was considered for detailed study[5]. In this paper, the definition of COTS, its virtues, skills needed to use it, selection, evaluation, testing, integration and composition of COTS components and the methodology for a successful COTS implementation are considered in detail.

2. COTS - Definition

According to Brownsword et al.[6],

- A COTS product is sold, leased, or licensed to the general public and offered by a vendor trying to profit from it
- Supported and evolved by the vendor, who retains the intellectual property rights
- Available in multiple, identical copies and
- Used without source code modification

2.1 Virtues of COTS SW

- Costs are lower than custom development, because product development costs are shared over many users
- Many others participate in finding bugs and limitations to the product (and the producer may actually fix these bugs)
- You can incorporate new technology more quickly because you use products containing it without having to learn all about it yourself
- Development time and risk are avoided when the COTS product provides all the features you need

3. Key Skills in Using COTS SW

Different skills are needed for using COTS SW effectively. Most COTS SW packages could be designed/implemented better by the SW professionals. Designing and writing codes are more fun for most software engineers than integrating the (suboptimal) packages available off-the-shelf. Integration skills are different from development skills. Problem-resolution is different when you must work around limitations/characteristics of a COTS SW package.

3.1 Selection of COTS

Mohamed, Ruhe and Eberlein[7] give the details of the COTS selection process in a recent research paper. This paper explores the evolution of COTS selection practices and surveys eighteen of the most significant COTS selection approaches. This paper also traces how each approach contributed to the improvement of current COTS selection practices and then compares them highlighting some open research issues relevant to the selection process, and concludes with a discussion of possible future directions to address these issues. Martínez[8] says that one of the most critical activities in COTS-based development is the selection of the components to be integrated into the system under development. Selection is basically composed of three activities: searching of candidates from the marketplace, evaluating them with respect to the system requirements and deciding the COTS to be used. Most of the different existing methods for COTS selection focus their efforts on evaluating and deciding, letting aside the problem of searching components.
in the marketplace. This lack of proposals is a serious drawback that makes the whole selection process highly risky, and often expensive and inefficient.

Martínez[8] has introduced the GOThIC (Goal-Oriented Taxonomy and reuse Infrastructure Construction) method that provides an integrated strategy for facilitating COTS components searching and reuse. It has been elaborated following an iterative process based on action-research premises to identify the actual challenges related to COTS searching. Then, possible solutions were envisaged and implemented by several industrial and academic case studies in different domains. Successful results were recorded to articulate the synergic GOThIC method solution.

3.2 Evaluation of COTS

An approach to defining the evaluation criteria of COTS was offered by Torchiano and Jaccheri[9]. They present a set of attributes to characterize COTS SW and aid in a selection process. Another approach to COTS evaluation was presented by Oberndorf [10]. The emphasis here is on the process of evaluating products. It is known as the PECA process and its functions are planning and evaluation, establishing evaluation criteria, collecting data and analyzing results. This process is not always sequential. That is, as you begin establishing criteria, you may recognize that you have not included a necessary domain expert. Or, your preliminary analysis of results may indicate missing evaluation criteria. With this process, you will amend your process and repeat the steps with the additional definitions. Any product evaluation is a complex process and must not be considered in isolation. The evaluation of the company that provides the product as well as how the product will integrate with other products in the organization is critical to a successful evaluation.

Dubois and Franch[11], opine that models for representing the evaluation criteria and the evaluations themselves, as well as processes to conduct the evaluation activity, are very important. Rawashdeh and Matalkah[12], feel that a model which ensures quality characteristics of such systems becomes a necessity. There are several existing quality models used to evaluate software systems in general; however, none of them is dedicated to COTS-based systems. An analysis study has been carried out by them[12] on several existing software quality models, namely: McCall’s, Boehm, ISO 9126, FURPS, Dromey, ISO/IEC TR 15504-2 1998(E), Triangle and Quality Cube, for the purpose of evaluating them and defining a ground to build a new model specializing in evaluating and selecting COTS components. The study also outlines limitations found in the existing models, such as the tendency to ignore a certain quality feature like functionality or the failure to describe how the quality measurement in these models has been carried out. As a result of this analysis, a new model has been built that supports a standard set of quality characteristics suitable for evaluating COTS components, along with newly defined sets of sub-characteristics associated with them. The new model avoids some of the limitations found in the existing models. The new model ignores quality characteristics that are not applicable to COTS components and is empowered with new ones that are. In addition, it matches the appropriate type of stakeholders with corresponding quality characteristics; such a feature is missing in all existing models. The objective of the new model is to guide organizations that are in the process of building COTS-based systems to evaluate and choose the appropriate products and that is essential to the success of the entire system.
3.2.1 Benefits and Pitfalls

Whatever evaluation approach you take in choosing a COTS-based product, you can learn from the experiences of others. There are services and tools that can guide you in a COTS evaluation. Software Engineering Institute of the Carnegie Mellon University has developed CURESM (COTS Usage Risk Evaluation) and also conducts product evaluation workshops to assist organizations in appropriately staffing and conducting COTS assessments.

3.2.2 Vendor buildup

Success is dependent upon a good relationship with your vendor. At a minimum, the vendor must view the component you purchase as core to their product line. You should also look at the vendor’s history in supporting prior releases simultaneously with the current product release. Determine your ability to influence your vendor’s product direction. While you might like to have every requirement met exactly, it is to your advantage to shape the COTS product with the support of the vendor. You do not want to receive special versions of the product that are only provided to you. Significant benefits are gained when many customers are using the same software. More thorough testing increases product stability and proper functioning.

According to Carney[13], COTS-based system does not eliminate the need to engineer your application integration. Similarly, COTS-based systems do not mean there is no development on your part. You need to be confident that your vendor has clearly identified the integration requirements and who is responsible for performing (and paying for) the integration.

Consider all aspects of the product when measuring product maturity. You may get what you want in the latest release, but has that release involved significant architectural changes that would reduce product stability – even when the vendor has a good history? Are you sold on buzzwords? Consider your reasoning when requiring adherence to any new technology. Is that technology really proven in a production environment? How will it benefit your users?

3.3 Testing of COTS components

COTS-based systems must still be tested within the context of your usage. No one will have the exact same configuration that you do. And, changes in COTS-based products will have a ripple effect on your configuration. Plan and budget for some change with each release. Understand your vendor’s policy for compatibility of APIs to prior releases. You can reduce the time you budget for testing because you have purchased COTS software, but you cannot eliminate test periods.

While “plug and play” has been fairly successful for hardware devices, it really hasn’t happened yet for software. It takes work to make the pieces come together. More importantly, if you determine that the component you are using is not sufficient, it will take still more work to plug in an alternate component. Then, that component may exhibit significantly different semantic and presentation behaviors. If your COTS choice is based on the belief that you can swap out the component at a later time, you probably have not made the right choice.

Vendors will always emphasize the significant number of features that their software encompasses. But, in reality, more than half of the features in COTS products go unused. The wise consumer will look beyond the length of the list to the actual usability and applicability of those features.

Pandeya and Tripathi[14] had stated the following in their recent paper: In a component-based software development life cycle, selection of preexisting components is an important task.
Every component that has to be reused has an associated risk of failure of not meeting the functional and non-functional requirements. A component’s failure would lead a developer to look for some other alternative of combinations of COTS, in-house and engineered components among possible candidate combinations. This means that design itself can readily change. The very process of design of a software system and component selection seems to be heavily dependent on testing results. Instability of design, further, becomes more severe due to requirements change requests. Therefore, this instability of design has to be essentially mitigated by using proper design and testing approaches, otherwise, it may lead to exorbitantly high testing cost due to the repeated testing of various alternatives. How are these three activities: component-based software design, component selection and component-based software testing interrelated? What process model is most suited to address this concern? This work explores the above questions and their implication in terms of nature of a process model that can be convincing in case of component-based software development.

4. Integration/Composition of COTS Components

When we look at developing software systems using components, it is the work of integrating the components with each other and the rest of the system that is the most important part of the component-based development process. Depending on the component model used and the actual components that are to be integrated, more or less work will have to be done in order to get all the parts of the system to function correctly together. It is this composition of different parts that make reasoning about quality attributes in a component-based system more complex than in a less modular system. Each component will have its own quality attribute profile, but when interfaced and used together with other components, the resulting composition may show a different quality attribute profile altogether.

When integrating components into a system assembly, it would be useful to be able to predict how the quality attributes for the whole system will be. The predictability of a composed system depends on many variables, not just the components or the component framework. The quality of the extra code needed to integrate the components into a functioning system (the glueware) will also affect the system, and perhaps even more importantly, process and organizational factors will also affect the system.

4.1 Two-layer wrapping for COTS Integration

Garcia-Rosello et al.[15] have furnished the details regarding "Two-layer wrapping for COTS SW integration", in the recent paper published in “IEEE Software”. They had written as given below: “Although COTS product integration presents clear advantages in a variety of engineering fields, several problems can arise in part due to their heterogeneous nature. Little research addresses the integration of particular COTS to specific domains. A University of Vigo project for reusable software development in the engineering domain adopted this approach. The underlying hypothesis was that making COTS integration easier would facilitate the development of domain-specific applications. The solution consists of a two-layer wrapping approach. The first layer captures the COTS domain model, facilitating integration of general-purpose functionality. A second layer provides better integration of more domain-specific functionality. Several real software development projects have used the proposed solution and the results yielded notable
effort savings, showing the approach's utility in reducing COTS integration efforts.

In their article “An architectural approach to building systems from COTS software components”, Vigder and Dean[16] had written the following: As software systems become increasingly complex to build, developers are turning more and more to integrating pre-built components from third party developers into their systems. This use of COTS software components in system construction presents new challenges to system architects and designers. Their[16] paper is an experience report that describes issues raised when integrating COTS components, outlines strategies for integration and presents some informal rules they have developed that ease the development and maintenance of such systems.

Guerra, Rubira, Romanovsky and Lemos[17], in their paper on "Integrating COTS Software Components into Dependable Software Architectures", observe the following: The COTS components, by their very nature, are built to be reused as black boxes that cannot be modified. Instead, the system architect has to rely on techniques external with respect to the component for resolving mismatches of the services required and provided that might arise in the interaction of the component and its environment. This paper proposes an architectural solution to turning COTS components into idealized fault-tolerant COTS components by adding protective wrappers to them.

In his paper on COTS Components in Software Development, Hardy[18] has written the following: The increasing need for efficient methods of software reuse and interoperability between applications has led software engineers into the realm of component-based software engineering (CBSE). COTS components provide a means by which the component development paradigm can be achieved. COTS components are designed by a third party and allow engineers to build software applications out of preexisting elements. The use of COTS components in software engineering allows for gains in developmental productivity, efficiency and reuse. The reliability, functionality and quality of COTS software components must be assessed before integration into a system. During integration, careful attention must be paid to the level of security achieved between interacting components. Software-wrappers can be used to control a component’s functionality as well as limit dependencies between components. Glass-box, black box and fault-injection techniques can be used to test the operability of a COTS system. Once a system has been designed and integrated, it must be maintained. Problems associated with individual component upgrades as well as vendor liquidation play a large role in the long-term maintenance of a COTS system.

4.2 A framework to address the impact of system of systems integration using COTS technology

In their recent paper published in the “International Journal of System of Systems Engineering”, Van Leer and Jain[19] writes the following: For systems engineering, systems integration (SI) establishes linkages between hardware (HW), software (SW), products, services, processes and humans. Over the last decade the world of systems development has evolved rapidly particularly in the use of COTS products as elements of larger systems. The growing trend toward COTS-based systems (CBS) architectures is based on modular components available within the market. This trend has presented various challenges for systems engineering practitioners attempting to understand the implications of
using COTS products within these large and complex projects. This paper analyses those unique aspects of COTS products that influence the SI process differently than the integration of ‘in-house’ custom developed products.

5. Methodology for Successful COTS Implementation

A successful implementation of a COTS 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 the concern over the lack of capability to develop and maintain proprietary IT 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.

5.1 Steps to a Successful COTS Implementation (Minkiewicz, 2005)[20]

The following “Six Step Methodology” proposed by Minkiewicz (2005)[20] highlights the important activities that should take place during a COTS implementation. Following this methodology throughout the software development lifecycle 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 of the major activities necessary to devise a well thought out solution that will not fall apart with the first upgrade of one of it’s components. Research [21-24] has indicated that the essential activities that must take place to ensure successful COTS based projects are:

- Analyze Software Requirements
- Evaluate and Select COTS Solution(s)
- Negotiate purchase / lease arrangement with vendor
- Implement the COTS based solution
- Maintain and upgrade the software solution
- Maintain license, subscription and royalty fees

Fig. 1 Overview of the six steps[20]

An overview of the six steps outlined above(Fig. 1) highlights the interactions that may occur throughout the execution of these steps. While this process implies a time dependency between these steps, it is important to realize that in certain cases this is not 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 is likely to occur in very early stages of a project as feasibility and affordability are analyzed.

Specifics about the quantification and application of the above factors can be found in another paper of Minkiewicz[25].
6. 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 well thought out or well 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 that results 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 the “Six Step Methodology” will ensure that important activities and decision points are properly executed, reducing many of the risks associated with such developments.

7. Acknowledgements

One of the authors (NG) thanks the Management and the Principal of the K.S.R. College of Arts and Science, for the encouragements.

8. References

(ICSE’04), http://www.lsi.upc.es/events/mpec/


