Developing an Effective COTS Selection Process for Application Systems Integration

Michael D. Moss

Capella University

Abstract

A significant trend within the software industry has been the progressive utilization of commercial off-the-shelf (COTS) applications for complicated, mission critical applications. However, the selection of appropriate COTS components is not necessarily an easy undertaking, as the dependability of the completed system is heavily contingent upon the effectiveness of the selection process (Alves & Finkelstein, 2002). This paper examines the problems and risks associated with selecting the optimum mix of COTS products, the challenges inherent within the decision-making process, and those criteria which managers should utilize during product evaluation. Additionally, this paper investigates how the realities of the marketplace and the new role of engineers as consumers affect COTS-based system development. Finally, this paper proposes several areas which would benefit from future research.

Developing an Effective COTS Selection Process for Application Systems Integration

A significant trend within the software industry has been the progressive utilization of commercial off-the-shelf (COTS) applications for complicated, mission critical system development. The philosophy behind this transition is that the effective use of COTS products decreases development and implementation time (in other words, saving time and money), allowing programmers to focus their attention towards the creation of domain-specific services. In contrast to the more traditional development-centric approach which utilizes in-house software development, those organizations employing COTS-based development utilize a procurement-centric process based upon the acquisition and integration of COTS components (Alves & Finkelstein , 2002).

According to Alves and Finkelstein (2002), the utilization of COTS products requires a significant shift in an organization's approach regarding system design and development, as once familiar processes must be modified. For example, architecture design must be combined with product evaluation and application development must include product adaptation and integration. Additionally, the use of COTS software involves new challenges and risks in that firms have very limited, if any access to a product's internal design and descriptions regarding its capabilities and required operational environment are often incomplete and confusing jargon from the vendor's marketing department. The evaluation process may also be somewhat limited if an organization's only hands-on experience occurs at a vendor sponsored product demonstration, which of course does not accurately simulate actual operational conditions.

Transitioning to a COTS-based development methodology also affects the traditional software development lifecycle (SDLC), as the evaluation of potential COTS packages must be conducted early in the process. And while the accurate gathering of user requirements is always vital to a project's success, this phase becomes even more significant as the selection of COTS products is directly influenced by the customer's stated needs. Given that future modifications to existing requirements normally play havoc with in-house development, significant changes when utilizing COTS products may violate the assumptions by which they were selected, thus rendering them useless and requiring a system redesign.

The selection of appropriate COTS components is not an easy undertaking, as the dependability (e.g., reliability, security, and availability) of the completed system is heavily contingent upon the effectiveness of this process. Unfortunately within the corporate environment, many acquisition decisions are based upon current vendor partnerships, subjective judgments, opinions influenced by vendor marketing departments, and the selection's impact upon profit. The acquisition of COTS components based upon ad hoc decision criteria may significantly increase an organization's risk of making non-optimal decisions. Selecting COTS products which are inherently faulty or do not integrate well with others may easily lead to incidents of unexpected behavior from the completed system, thus placing both the system itself and perhaps even the entire program at risk (Alves & Finkelstein, 2002). Thus, this paper examines the problems and risks associated with selecting the optimum mix of COTS products, as well as the challenges inherent within the decision-making process. It also examines several selection methodologies and evaluates their usefulness.

COTS-Based Development Requires New Ways of Thinking

COTS-based systems development poses unique challenges that traditional developer-centric models cannot anticipate (Yang, Bhuta, Boehm, & Port, 2005). The selection of COTS components represents a critical stage within the SDLC in which development becomes an act of composition. In contrast to custom-based development in which systems are developed according to specific requirements and the software engineers are producers of the code, COTS-based development starts with general requirements and then the requisite products are purchased within the marketplace. In this scenario, the engineers become consumers who integrate the selected components into a system. The nature, timing, order of activities, and procedures involved between the two cases differ significantly, resulting in a fundamental shift with respect to the processes required to develop and maintain COTS-based systems. One must also be aware that this shift not only affects existing processes, but also people as well. Thus, not only must the engineering or technical philosophy evolve, but so too must the organization's business, organization, and culture (Brownsword, Oberndorf, & Sledge, 2000).

Increase design flexibility by changing requirements into goals.

The selection of appropriate COTS products begins with a reevaluation regarding the nature of requirements, as they play a vital, controlling role within the system development process. Traditionally, the primary goal of gathering requirements is to translate the desires of the stakeholders into a succinct and clear description with respect to the system being developed. While the degree of specificity required for the development-centric approach provides the requisite level of detail for system design, development, and implementation, it serves as an impediment when utilizing COTS-based development. As requirements become more restrictive, they tend to reduce the pool of applicable products, exclude the use of COTS altogether, or require significant product modification. Basically, COTS-based development requires more flexibility (Alves & Finkelstein, 2002).

To achieve the requisite flexibility and thereby improve the COTS selection process, Alves and Finkelstein (2002) suggest the adaptation of a goal-orientated approach with respect to defining requirements. The authors argue that utilizing goals provides an appropriate level of abstraction, allowing for the evaluation of alternative requirements which achieve the same, desired outcome. Additionally, as requirements have been known to evolve during a system's development, goals (which are established at a higher level than requirements) provide a more stable base from which to make decisions related to COTS acquisition. Thus, the acquisition process becomes an exercise matching a system's goals with the features and capabilities provided by various COTS products. In conjunction with an assessment of each product's features, the risks of each alternative must also be carefully considered, to include the ability to modify the product, the level of vendor support, and (of course) the number and relative importance of those goals which required compromise.

By replacing the tradition method of itemizing specific requirements with a goal-centric approach, the authors introduce flexibility into a process which normally produces highly constraining results. This new degree of freedom allows for negotiation and compromise among the various stakeholders, as they search for an appropriate solution.

Realities of the marketplace affect COTS development.

In contrast to traditional development activities in which system engineers and software developers are cloistered in their cubicles designing systems and writing code, COTS development does not occur within a vacuum. As Brownsword et al. (2000) remind us, COTS development is shaped by the realities of the marketplace which directly affect the nature and evolution of a COTS-based system. So what are the characteristics of this marketplace? The foremost is frequent and continuous change. Another is perspective, as the development and evolution of COTS products are based upon the needs of the marketplace, not an individual system. Products are developed based upon a general set of assumptions regarding their use, not a specific set of requirements. Licensing and proprietary rights may significantly affect both system cost and how a product may be modified to fill specific requirements. The frequency of version releases and patches rests at the discretion of the vendor, not the customer. Software engineers have limited, if any, access to the product's source code. The design of COTS products is based upon architectural assumptions which may not only be inappropriate for a specific system, but could also result in conflicts with existing systems. And finally, COTS products may require interdependencies with software which is not present on the host system.

Organizations must adopt new perspectives and procedures.

Indeed, the utilization of COTS components requires system developers to simultaneously consider requirements, architecture, license pricing, and the marketplace, as any one of these may easily affect the other three. This fundamental change in systems design requires an evolution in existing development and maintenance processes, as well as the adoption of new ones. And when processes change, people must change also. As such, the transition to COTS-based development is not marked by merely engineering or technical changes, but also by changes to the very fabric of an enterprise, affecting its business, organization, and culture.
So what are some of the changes an organization might expect? Brownsword et al. (2000) notice these changes usually occur within three major activity areas: engineering, business, and project-wide. As the engineering activities are directly associated with a system's technical conceptualization, construction, and maintenance, this area is significantly affected. Engineers must now understand the manner in which requirements, the marketplace, and design are interconnected with each other. Engineers must determine and prioritize their user's requirements, determine which elements are negotiable, generate alternative designs and select COTS technologies based upon in-house evaluations, maintain current knowledge regarding available and emerging technologies, and perform any modification required to successfully

integrate COTS products. Given that decisions regarding the acquisition of COTS products are no longer simply based upon technologies but also upon sound business practices, activities within the business area must now include the establishment of business case and cost estimates, negotiation of licenses, and management of vendor relationships. As the project-wide activity areas spans across an organization's engineering and business activities, the organization must now identify, prioritize, and mitigate COTS associated risks, identify the required COTS skill sets, and secure the by-in of senior leadership and project staff.

Implementing a COTS Selection Process

Establish a formal selection and acquisition process.

Basically, the COTS selection process involves four activities: gathering and defining the requirements, understanding what products are available, assessing which products meet the requirements, and choosing the "best" package (Carvallo, Franch, & Quer, 2007). Gingras (2006) expands this philosophy by proposing that organizations adopt the following 11 step process for COTS acquisition:

1. The Business Case: A project should not be approved without an appropriately documented business case which justifies the project and specifies its anticipated, quantifiable benefits.
2. Governance and Control: An oversight structure must be established to keep the project on course.
3. Constraints: Budgetary, technology, and time constraints must be established.
4. Requirements Definition: Requirements must be understandable (even to nontechnical personnel) and mapped to business processes.

5. Market Intelligence and Survey: Investigate and rank vendors according to such metrics as market share and financial stability.
6. Selection Parameters: Draft a Request for Proposal (RFP) based upon the requirements and forward it to the vendors selected in Step 5. Additionally, rate the requirements in terms of importance, thereby allowing for a calculated weighted average based upon the vendors' responses. The vendors responses should address whether a specific feature is currently available scheduled for a future release, available as a third party add-on, or not available.
7. Implementation Considerations: Select capable system integrators. If necessary, requite a "short list" of each vendor's top five integrators.
8. Negotiation and Contracts: Receive each vendor's standard contract early in the process for review by your legal team.
9. Implementation Issues and Project Management Office (PMO) Formation: As a general guideline, 80% of the PMO staff should be recruited from business departments, not IT.
10. Training Considerations: Develop a three-part training program consisting of pre-implementation, implementation, and post-implementation courses.
11. Post-Implementation Analysis: After the project is completed, review the process. Conduct an independent audit with respect to the effectiveness of the processes and the benefits derived from the new system.

Develop standards for COTS evaluation.

However, according to Li, Conradi, Bunse, Torchiano, Slyngstad, and Morisio (2009), a significant number of companies do not employ a formal COTS selection process. These organizations usually justify their ad hoc approach by citing the limited number of COTS products available or the per-existing relationship with a long-term vendor. But the disturbing attitude among many integrators is their belief that formal processes are not cost-effective and increase their delivery time. Thus, any proposed COTS selection process must meet several standards before it may be considered for use within the “real world.” For example, the selection process should ensure the COTS product is (1) functional, (2) reliable, (3) cost effective, (4) easy to customize, (5) easy to implement, and (6) fulfills the system’s requirements.

From their survey of middle to senior level IT management, Keil and Tiwana (2005) identify seven criteria which the managers use when evaluating COTS products: functionality, reliability, cost, ease of use, vendor reputation, ease of customization, and ease of implementation. The authors suggest that by using these criteria as a foundation, adding relative weights, and incorporating scoring (for example, 1-10 possible points for each attribute), managers could develop a process for quickly evaluating and comparing COTS applications. This process could also allow managers to perform what-if analysis by determining the sensitivity of the scores to modifications with respect to the software’s attributes.

Another relevant issue involves system dependability (reliability, security, and availability). As the dependability of a COTS-based system is a function of those products from which it is comprised, failure may occur if those components do not behave as anticipated. To mitigate this risk, engineers must utilize a process which provides them with empirical evidence demonstrating a component’s’ dependability. Donzelli, Zelkowitz, Basili, Allard, and Meyer (2005) developed such a procedure, which they term the Unified Model of Dependability (UMD). Basically, the UMD translates a system architect’s high-level requirements into a comprehensive model for each component which specifies those quantifiable properties the component must have to be considered dependable. Once developed, this model acts as a point of reference from which experiments may be designed to compare the dependability of similar COTS products. This process allows engineers to assess the strengths and weaknesses of individual components within a specific environmental context. The knowledge gained from this exercise not only helps engineers select products which optimally match the system's requirements, but also identifies any inherent weaknesses which must be addressed to improve both theirs and the system's dependability.

Choose an appropriate time within the SDLC for COTS selection.

An important issue addressed by the Li et al. (2009) involves the timing of the COTS selection process. As the traditional SDLC lacks a phase specifically for COTS selection, many companies acquire their products relatively early in the process. Integrators appear to make their selections early in the belief that any issues should be identified as soon as possible. This practice results in both benefits and challenges. As reported by Li et al., those integrators who select COTS products early cite the following benefits:

- Once the functional requirements and architecture have been determined, they have solid criteria by which to select their components.
- They may easily configure the system to the component's specification, allowing for seamless integration.
- Once the architecture has been defined and the COTS products selected, they may easily develop their test cases and quality assurance plans. Additionally, they develop more accurate project cost estimates.

However as Li et al. (2009) discover, the selection of COTS products early in the SDLC also involves several challenges for integrators as well:

- In these early stages, they are usually not very concerned regarding the product's technical details; an attitude which could lead to future implementation and integration issues.
- If the project's duration is rather long, a COTS vendor could release a new version, potentially leading to a component revaluation or even system redesign.
- If the agile development process were being used, the identification and documentation processes are usually established to facilitate rapid changes. As such, the earliest point at which time they could consider component selection would be during either the detailed-architectural-design or the development iterations. As such, extra effort could be required to refactor the system.

While the early identification and evaluation of COTS products may lessen the risk of mismatching components with requirements, integrators should be aware regarding the dangers of making their selections too early.

New Developments Regarding COTS Selection and Maintenance

The implementation of a COTS-based methodology results in new challenges regarding the identification of those components which are most suitable for building the system. This process is generally carried out in a relatively ad hoc manner via searching through unstructured information across the Internet. This method becomes more expensive and inefficient as the evaluation criteria become more complex.

Increase the efficiency of COTS selection via standardized search criteria.

As a potential solution to this problem, Réquilé-Romanczuk, Cechich, Dourgnon-Hanoune, and Mielnik (2005) support the development of a standardized structure for applying a knowledge-based process to the identification of COTS components. The authors approach this issue from both the vendors' and customers' perspectives by proposing a system in which the information provided by vendors when describing their products and that utilized by customers as their search criteria becomes standardized. Réquilé-Romanczuk et al. suggest the establishment of such a standard with respect to the description of a product's properties and capabilities would allow significantly more efficient matching between vendors and customers. However, as the authors also note, many difficulties exist before such a system may come to fruition.

While utilizing COTS components may well reduce cost and time-to-market while improving productivity with respect to system development, selecting the appropriate components is a very challenging task (Maiden & Ncube, 1998). In the real world, finding the appropriate product that successfully satisfies a system's requirements is difficult. And, incorporating a wrong choice could well result in a mismatch with other COTS products, thus degrading the entire system. So while waiting for Réquilé-Romanczuk et al.'s (2005) standardized, classification system to become available, Dong, Yang, Chung, Alencar, and Cowan (2005) offer a detailed architectural specification template which they believe will assist system architects in both selecting the right COTS product and detecting current mismatches within the system.

Basically, the implementation of a COTS-based design involves looking for suitable components which meet requirements, testing and evaluating those components, modifying the selected components to fit into the system's infrastructure, integrating the components into a coherent whole, and finally maintaining the completed system. However as COTS products are normally treated as black-boxes, component selection has become problematic. While a more grey-box approach has been called for to expose more of a component's functionality (e.g., external calls and call sequences), Dong et al. (2005) believe this exposure should be extended to include a component's functional interfaces, assumptions, non-functional properties, applicability, standards, related components, and sample uses.

Specifically, several characteristics regarding COTS products should be available to system engineers as they evaluate candidates for their solution. Information concerning the functional interfaces exposed by the product should include the component's structural information (such as names, types, and number of parameters, and UML diagrams) and behavioral aspects (to include state transition, data flow, and collaboration diagrams). Additionally, a component's assumptions should be clearly stated so that integrators are aware of both its pre-conditions and post-conditions. These assumptions should include suppositions regarding the system's architectural design, characteristics regarding the components and their connections, and the assembly process. Knowledge regarding a component's non-functional properties (performance, security, reliability, and concurrency) is vitally important to engineers as they develop strategies for connecting various COTS products. For example, the unknowing combination of a single-thread Web agent with a multi-thread Web server would in effect transform a concurrent system into a sequential one, resulting in significant and unexpected performance degradation. The component's applicability should be well documented, as COTS products are usually limited to specific operating systems (UNIX or Windows), languages (C, C++, Java), and environments. Engineers must be able to validate a component's functionality within a specific environment. A component's specifications should list the standards to which it adheres, whether it is COBRA, DCOM, or .NET. Knowing the standard gives engineers a degree of confidence with respect to its compatibility with other COTS products during integration. The inclusion of information comparing one component with its competitors provides engineers with valuable information, increasing their changes of finding the best product. These data are also very useful when integrators wish to know how well one component collaborates with another, thus obtaining the optimal product mix. The inclusion of examples and sample use cases assists engineers in using the product correctly and to its full potential. These examples also provide templates by which engineers may test the components against their system's requirements (Dong et al., 2005).

To assist integrators in their search for and selection of suitable COTS products, Dong et al. (2005) provide a component specification stencil written in XML, which allows for an automated searching process, as well as the detection of mismatched components. By using their XML specification for COTS components, integrators may build a repository from which information regarding component selections, assessments, and integration may be warehoused for future queries. The authors cite several advantages of using their stencil for specifying COTS components. It reduces costs and errors during the selection process. And, the explicit declaration of a component's properties assists engineers in discovering inconsistencies between COTS products during integration, resulting in fewer surprises.

Using the XML generated database greatly simplifies the COTS selection process, as engineers may take full advantage of its searching and matching functionalities. To select a product, an engineer conducts a search within the database using specific criteria, such as functional properties, non-functional properties, applicability, and standard. Those components which match the query specifications are then further reviewed based upon the data presented on their XML stencil. Integrators then select those components which best fit the system's requirements. Additional information which is returned with each query include the component's assumptions and related components, allowing engineers to know those conditions which the component requires to function properly and with which other components it may be successfully integrated. Additionally, as vendors adopt this standard to describe their products, integrators will be able to conduct COTS searches across the Internet (Dong et al., 2005).

COTS updates require regression testing: the I-BACCI *alternative.*

Unfortunately, the COTS environment is not static, as products are continuously updated. Once product decisions have been made, patches, minor revisions, and major upgrades, continue their evolution. When these new versions are released, they must be tested before replacing those components currently residing within the system. Organizations rely on regression testing to verify that any modification made to the product has not resulted in unintended consequences and the new component still meets the requirements for which the older version was originally acquired. However, many techniques utilized in regression testing rely on access to the component's source code, which is not supplied. As their code is proprietary, companies usually furnish only the binary files and documentation. Therefore, the default regression testing strategy is usually to retest each function which interacts with the glue code connecting the various COTS products. While this retest-all linkage strategy is rather clear-cut, the process may become prohibitively expensive in terms of both time and resources. As such this regression technique is usually considered to be neither feasible nor efficient (Zheng, Robinson, Williams, & Smiley, 2006).

To alleviate this issue, Zheng et al. (2006) offer a regression process based upon their premise that "when components change and only binary code and documentation are available, regression test selection can safely be based upon the glue code that interfaces with sections of the component that changed" (p, 512). Referred to as the Integrated - Black-box Approach for Component Change Identification (I-BACCI), their six-step process accepts the binary code of both the new and revised components and produces a set of test cases which focus on the deltas between the two versions.

The first step of the I-BACCI process involves the decomposition of the component's binary files and the filtering of inconsequential information (usually performed by such tools as Decomposer and Trivial Information Zapper) to expedite the comparison between the old and new versions by utilizing differencing tools. During the second step, these differencing tools (such as TID-BITZ, custom software created by the authors) identify deltas between the two releases and generate a report indicating the modified functions. The third and fourth steps produce function call graphs. The inputs for these steps are the calling functions within the components and those within the glue code, respectively. Results from these steps may be integrated and their graphs drawn by such applications as GraphViz10. These graphs are reviewed to identify those glue code functions which have been affected by the version changes. During the fifth step, directed graph theory algorithms are utilized to identify those functions within the glue code that have been affected by the version change. These are the only functions which require testing. The sixth step involves the development of test cases which are mapped to their respective glue code functions (discovered in the previous step) and their subsequent testing.

According to Zheng et al. (2006), their I-BACCI process may significantly reduce the number of required test cases, as it concentrates solely on those glue code functions which have been affected by a version change, while ignoring the remaining areas of the application. Results from their testing indicate a potential 70% reduction in the number of required test cases without increasing the level of fault exposure.

Potential for Further Research

Given this paper as a starting point, several avenues exist which would benefit from future research. One area involves an investigation into the balancing of goals in terms of addressing prioritization and possible tradeoffs between desired requirements for a system and available features provided by the proposed COTS solution (Alves & Finkelstein, 2002). Another area of interest could be the degree to which personalization and adaptation of a mixture of COTS components could be predicated before their acquisition (Réquilé-Romanczuk et al., 2005). As previously noted, many companies conduct generalized, ad hoc searches for COTS products. As such, research regarding the existence of component specification repositories (to include their search and matching mechanisms) on the Internet could be of value (Dong et al., 2005). Matching future research considerations against the previously mentioned selection criteria should provide a gauge with respect to their potential for making a relevant contribution to the body of knowledge regarding those issues surrounding COTS selection.

References

Alves, C., & Finkelstein, A. (2002). Challenges in COTS decision-making: A goal-driven requirements engineering perspective. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, 789-794. Ischia, Italy: ACM Press. doi:10.1145/568760.568894

Brownsword, L., Oberndorf, T., & Sledge, C. (2000). Developing new processes for COTS-based systems. *IEEE Software, 17*(4), 48-55. Retrieved February 4, 2010, from ABI/INFORM Global.

Carvallo, J., Franch, X., & Quer, C. (2007). Determining criteria for selecting software components: Lessons learned. *IEEE Software, 24*(3), 84-94. Retrieved February 4, 2010, from ABI/INFORM Global.

Dong, J., Yang, S., Chung, L., Alencar, P., & Cowan, D. (2005). A COTS architectural component specification stencil for selection and reasoning. In *Proceedings of the Second International Workshop on Models and Processes For the Evaluation of off-the-Shelf Components*, 1-4. St. Louis, Missouri, United States: ACM Press. doi:10.1145/1082948.1082959

Donzelli, P., Zelkowitz, M., Basili, V., Allard, D., & Meyer, K. (2005). Evaluating COTS component dependability in context. *IEEE Software, 22*(4), 46-53. Retrieved February 4, 2010, from ABI/INFORM Global.

Gingras, D. (2006, February). Effective use of software stems from standardized selection process. *Pulp & Paper, 80*(2), 46-49. Retrieved February 4, 2010, from ABI/INFORM Global.

Keil, M., & Tiwana, A. (2005). Beyond Cost: The drivers of COTS application value. *IEEE Software*, *22*(3), 64-69. Retrieved February 4, 2010, from ABI/INFORM Global.

Li, J., Conradi, R., Bunse, C., Torchiano, M., Slyngstad, O., & Morisio, M. (2009). Development with Off-the-Shelf Components: 10 Facts. *IEEE Software*, *26*(2), 80-87. Retrieved February 4, 2010, from ABI/INFORM Global.

Maiden, N., & Ncube, C. (1998). Acquiring COTS software selection requirements. *IEEE Software*, *15*(2), 46-56. Retrieved February 4, 2010, from ABI/INFORM Global.

Réquilé-Romanczuk, A., Cechich, A., Dourgnon-Hanoune, A., & Mielnik, J. (2005). Towards a knowledge-based framework for COTS component identification. In *Proceedings of the Second International Workshop on Models and Processes For the Evaluation of off-the-Shelf Components*, 1-4. St. Louis, Missouri, United States: ACM Press. doi:10.1145/1082948.1082951

Yang, Y., Bhuta, J., Boehm, B., & Port, D. (2005). Value-based processes for COTS-based applications. *IEEE Software*, *22*(4), 54-62. Retrieved February 4, 2010, from ABI/INFORM Global.

Zheng, J., Robinson, B., Williams, L., & Smiley, K. (2006). Applying regression test selection for COTS-based applications. In *Proceedings of the 28th International Conference on Software Engineering*, 512-522. Shanghai, China: ACM Press. doi:10.1145/1134285.1134357