Lessons Learned From Using COTS Software on Space Systems

Richard J. Adams and Suellen Eslinger
The Aerospace Corporation

The incorporation of commercial off-the-shelf (COTS) software into software-intensive systems brings promises of reduced cost and schedule and higher reliability and maintainability by using "proven" software. However, the reality of using COTS software can be very different! This article presents the results of a survey of U.S. Air Force Space and Missile Systems Center and National Reconnaissance Office programs on their experiences with incorporating COTS software into their systems. Six major lessons learned derived from the survey are described, along with recommendations for improving the acquisition of COTS-based systems.

The new Department of Defense (DoD) 5000 series of acquisition regulations[1] requires the exploitation of commercial off-the-shelf (COTS) products in DoD acquisitions. The incorporation of COTS software into software-intensive systems brings promises of reduced cost and schedule, and improved reliability and maintainability by using *proven* software. However, the reality is often very different! The use of COTS software, which can provide significant benefits, also poses major new risks in the acquisition, development, and sustainment of software-intensive systems that must be acknowledged and managed.

In support of the U.S. Air Force Space and Missile Systems Center's (SMC's) Directorate of Systems Acquisition, the authors performed an in-depth study of actual COTS-based system (CBS)[2] development and sustainment experiences on SMC and National Reconnaissance Office (NRO) programs. This study was motivated by numerous reports of COTS software-related problems. The purposes of this study were: (1) to synthesize and share lessons learned from actual CBS development and sustainment experiences, and (2) to provide recommendations for mitigating the risks inherent in CBS development and sustainment.

COTS Software Study Process

The study process comprised three steps: gathering information, synthesizing lessons learned, and developing acquisition recommendations. The techniques selected for the first step were focused interviews and documentation reviews. A COTS software experience questionnaire was developed to provide a framework for the interviews. This questionnaire was sent to the interviewees in advance of the interview to enable them to prepare for the type of information being requested. The interview itself was then allowed to proceed as a free exchange of information, with the questionnaire being used to return conversation to the topic, if necessary. The questionnaire was also consulted toward the end of the interview to identify areas that had not been addressed. In addition, interviewees were asked to provide any written documentation previously prepared by their program on COTS software experiences or lessons learned.

The COTS software experience questionnaire requested information on both good and bad experiences with COTS software. For each experience, information was requested on the nature of the experience, where in the life cycle it occurred, the COTS software involved, the functions provided by the COTS software, and their criticality to the system. For good experiences, the interviewees were asked to identify any actions taken that contributed to the good experience. Regarding bad experiences, the interviewees were asked to do the following:

- Identify any actions taken to solve the problem or mitigate further risk.
- Provide information as to what they would have done differently to find the problem earlier and to solve the problem or mitigate the risk, given the benefit of perfect hindsight.

The authors interviewed more than 50 representatives from 18 SMC and NRO program organizations, including personnel from the government, development contractors, and The Aerospace Corporation. In addition, domain experts from The Aerospace Corporation were interviewed concerning their experiences with COTS software in their domains. The domains chosen were considered to be critical to space systems, such as computer security and telemetry processing.

Following the interviews, the authors reviewed the documentation on COTS software experiences and lessons learned that were obtained from the interviewees. Note that the information gathered was from experiences on the ground segments of space systems, due to the still rare use of COTS software in onboard software for satellites and launch vehicles.

During the second step of the study, the authors first identified and documented more than 150 distinct findings from their interview notes and document reviews. The authors then performed an iterative series of analysis and syntheses to derive lessons learned from the findings. Six significant lessons learned were identified that encompassed the collection of findings. The final step of the study was to develop specific acquisition recommendations to help mitigate the risks in CBS development and sustainment. The lessons learned and recommendations are described below.

Lesson Learned 1

Critical aspects of CBS development and sustainment are out of the control of the customer, developer, and user.

This lesson concerns the realities of the commercial marketplace. COTS software vendors are driven by today's fast-paced market, characterized by highly volatile business strategies and market positions. Vendors may go out of business, merge with, or be acquired by other companies. Vendors may also drop or de-emphasize products or hardware platforms, usually without warning.

Of particular note is the fact that the more numerous commercial customers, not the defense community, drive the market for COTS software. In some instances, the market is diverging from defense needs. One example of this is the empha-

sis by some vendors on Windows NT platforms for commercial users (and the corresponding dropping or de-emphasis of the high performance UNIX workstations extensively used in ground systems for defense space applications). Another example is the targeting of COTS satellite control software to the operational paradigm of commercial communications satellite customers where there is minimal human intervention rather than the person-in-the-loop paradigm of defense satellite operations.

The quality and content of COTS software upgrades are unpredictable. Vendors are market-driven in their upgrades, focusing upon additional features to attract new customers and fixes to problems encountered by their principal customer base. Vendors may not be willing to fix problems experienced by only a few customers, even if the customer is willing to pay the vendor to fix the product. This can be especially applicable to defense space applications, which may use different features or place different stresses upon the COTS software than commercial users of the same software.

Because of time-to-market pressure, vendors perform limited testing on COTS software upgrades, especially regression testing of supposedly unchanged features. In addition to introducing bugs in previously working capabilities, upgrades may decrease performance, increase computer resource utilization, and introduce incompatibilities with other COTS software products. Also, upgrades may eliminate backward compatibility with previous versions, possibly necessitating design and data structure rework. Numerous interviewees stressed the need for developers and sustainers to fully test each upgrade before incorporating it into the system.

The schedule of upgrades, both frequency and release dates, is time-to-market driven. However, the pressure to bring new features to market quickly may cause vendors to drop or slip other promised features, fixes, or upgrades for particular platforms. Sometimes needed upgrades are delayed because of dependencies between COTS software products (e.g., vendors waiting for the next operating system upgrade before issuing their next major upgrade).

The costs of COTS software products and associated services are also market driven. Fees and fee structures of licenses and services may change without warning potentially resulting in a large cost impact if changes occur after developer commitment to a particular COTS software product. One particularly damaging example of this is when a vendor eliminates site licenses and requires a separate copy of the COTS software to be purchased for each operator seat in the ground system. Vendors may also change the type and quality of the customer support they provide. In particular, new vendors trying to gain a market position may initially be very responsive but may lose their responsiveness after establishing a larger customer base.

No program organization interviewed had experienced all of the problems cited above, nor did any organization have problems with all of their selected COTS software. However, every program organization interviewed had some problems with one or more COTS software products. Encountering these realities of the commercial marketplace should be considered the norm, not the exception, in the development and sustainment of CBS. Contingencies (e.g., alternative product choices, and cost and schedule margin) to handle these occurrences need to be built into development and sustainment plans from the beginning of the life cycle.

Lesson Learned 2

Full application of system and software engineering is required throughout the CBS life cycle.

This lesson reflects the fact that using COTS software does not eliminate portions of the system life cycle or the necessity for performing system and software engineering. Using COTS software reduces the scope of software design and implementation activities for that part of the software whose functionality is provided by the COTS software. Software requirements analysis, architectural design, integration and testing, and qualification testing must still be performed along with certain detailed design and implementation tasks. Moreover, system requirements analysis, design, integration and testing, and qualification testing must still be performed for all system functionality, independent of how the functionality is implemented.

Every new COTS software release requires a full application of the system and software engineering life cycle to properly incorporate the new release into the CBS. For example, incorporating each new release can require the following:

- Regression testing and prototyping to determine its behavioral characteristics as compared to the previous release and its compatibility with other COTS soft ware in the CBS.
- Testing of new features and bug fixes.
- Modifications to glue code and user interfaces.
- Modifications to the COTS software data base/file structure and content.
- Full software and system integration testing and requirements verification.
- Training for both the software developers and the operators.

Thorough requirements analysis is especially important with CBS development since it is necessary to understand which requirements can be traded against existing COTS software capabilities versus which requirements are essential to the mission and not in the trade space. This is true for all requirements levels from the highest level system requirements through the software level requirements.

To understand existing COTS software capabilities, hands-on prototyping of the COTS software is necessary since it is not generally possible to determine the true COTS software capabilities from the vendor's marketing demonstrations and literature. Furthermore, due to the possibility of incompatibilities or adverse interactions (e.g., performance degradation) between COTS software products, this prototyping must be performed in a system context where unexpected impacts of integrating multiple COTS software products can be discovered.

Numerous interviewees emphasized the importance of designing CBS architectures to support the evolution or replacement of COTS software. Since true *plug and play* among COTS software products does not yet exist in the commercial marketplace, architectural features that help minimize the impact of upgrading to new releases of COTS software or replacing one COTS software product with another of similar functionality are essential. The CBS architecture must also have a suffi-

cient computer resource margin and growth path to accommodate increases in resource utilization by COTS software upgrades.

Security, safety, and supportability must be designed into the CBS at the system level. COTS software capabilities in these areas are aimed at commercial applications having different, and frequently less stringent, security, safety, and supportability requirements than defense applications. Furthermore, each COTS software product is designed independently, as a stand-alone package, not as part of an integrated system. The CBS design needs to provide for integrated security, safety, and supportability features across COTS software products and newly developed or reused code. This is especially important for security since each COTS software product has its own vulnerabilities. Many of these vulnerabilities are well known in the industry, and new vulnerabilities are continually being identified. Without integrated security features being designed into the CBS, the system's vulnerabilities can be determined simply by knowing which COTS software products are in use.

Both initial evaluation of COTS software for product selection, and subsequent periodic re-evaluations of new COTS software releases for product evolution are necessary throughout the development and sustainment life cycle. The system-engineering viewpoint must be applied simultaneously to the selection of the computer hardware and COTS software. Selection of a computer hardware platform without concurrent consideration of the availability of COTS software for that platform can result in more newly developed software being required than expected, and thus can increase the system development and life-cycle costs.

The initial evaluations and periodic re-evaluations of COTS software must be based upon multi-dimensional evaluation criteria, not just upon the functionality provided by the COTS software. Examples of such evaluation criteria include the reliability of the COTS software, its ability to interface with other parts of the CBS and with legacy systems, the COTS software's implied operations concept, the vendor's characteristics, and the cost.

Finally, it is always necessary to have backup strategies and contingency plans for each COTS software product in case unforeseen problems arise that require its replacement.

Lesson Learned 3

CBS development and sustainment require a close, continuous, and active partnership among the customer, developer, and user.

This lesson concerns the need for the customer, developer, and user to be prepared to trade cost, schedule, performance, and operations and maintenance concepts to achieve the maximum benefits from using COTS software. The customer and user must understand their requirements sufficiently well to know which requirements can be relaxed to achieve a COTS-based solution and which are essential to the mission and cannot be traded. To facilitate the trades of requirements versus COTS software capabilities, the customer and user must be willing to prioritize their requirements initially and re-prioritize them as necessary throughout the life cycle. Merging of an intimate understanding of the requirements (as held by the customer and user) and an intimate knowledge of the COTS software capabilities (as held by the developer) is necessary to ensure the adequacy of these trade decisions.

Each COTS software product has its own world view that, when incorporated into the CBS, may force a particular operations concept upon the user. Frequently the COTS software operational paradigm is at odds with the user's existing operational procedures. As such, accommodating a COTS-based solution may require the user to be able and willing to reengineer existing operational procedures. Similar reengineering may be needed for existing on-site maintenance procedures. Ensuring the eventual acceptability of the CBS in the user's operational environment requires close cooperation between the user and developer.

At any time during the CBS life cycle, decisions may need to be made due to issues such as new COTS software limitations or incompatibilities being discovered, COTS software upgrades diverging from needs, or COTS software needing to be replaced due to withdrawal of vendor support. A close, continuous, and active partnership among the customer, developer, and user (e.g., via the application of integrated product and process development) will help to ensure the adequacy of the major COTS software-related decisions and the acceptability of the delivered CBS. Without such a partnership, the customer and user will not gain a full understanding of the evolving CBS capabilities and may experience unpleasant surprises when finally exposed to the capabilities of the delivered CBS in the operational environment.

Lesson Learned 4

Every CBS requires continuous evolution throughout development and sustainment.

This lesson reflects the fact that maintaining currency with COTS software upgrades is essential during both development and sustainment. Because vendors support only a limited number of past releases, delaying implementation of upgrades can result in unsupported versions of COTS software products in the CBS. When this happens, the vendor will not provide fixes to bugs and will not provide consultation services.

Delaying the implementation of upgrades can exacerbate system impacts. Upgrading from one major release to the next consecutive release does require time and effort. However, the upgrade can be considerably more expensive when attempting to skip major releases, which generally occur every 12 to 18 months. Delaying upgrades longer than that time interval can result in significant paradigm changes in the COTS software.

Sometimes upgrading to a later major release requires upgrading through each of the intermediate major releases. This is especially true if the vendor has changed the structure of the COTS software's databases or files. When this occurs, vendors frequently provide automated tools to assist their customers in conversion from one release to the next consecutive release. Such tools are not provided to assist in skipping major releases.

Many factors, both internal and external to the CBS, can drive the need to maintain currency with upgrades to the CBS' COTS software. Organizations or systems external to the CBS can require

COTS software upgrades. Examples of this include upgrades to government off-the-shelf software incorporated into the CBS, to legacy systems to which the CBS must interface, or to the Defense Information Infrastructure Common Operating Environment (if used by the CBS).

Another factor influencing the need to maintain currency with COTS software upgrades is the limited life span of computer hardware platforms (e.g., workstations and servers). Most programs plan on hardware upgrades every four to five years during sustainment. Maintaining currency with COTS software releases is essential for upgrading to new hardware since it is not usually possible or desirable to execute old versions of the operating system and other COTS software on new hardware platforms.

Furthermore, COTS software may need to be replaced or added at any time due to factors such as: elimination of vendor support, divergence from system needs, identification of unacceptable limitations or vulnerabilities, increased costs for licenses or support services, and new or modified user needs requiring changes in functionality or performance. Incorporating new COTS software usually requires the latest version of the operating system and other related COTS software to be in place.

One of the most damaging decisions frequently made in CBS development is to freeze the versions of the COTS software products throughout the development period. Due to the length of the development period for large software-intensive defense systems, this decision can result in the delivery of a system that is obsolete because its COTS software products are no longer supported. A major upgrade effort with associated cost and schedule impacts is then necessary before or shortly after the system becomes operational. Since maintaining currency with COTS software upgrades is necessary throughout development as well as sustainment, upgrading COTS software needs to be built into both development and sustainment plans from the beginning of the life cycle.

Numerous interviewees emphasized the folly of modifying COTS software, which can constrain the CBS evolution path and increase life-cycle costs. Modifying COTS software should always be a solution of last resort in CBS design. Incorporating a modified COTS software product into the CBS requires the developer and government to engage in a long-term relationship with the vendor to ensure that the unique modifications will be made to future releases. Attaining such a relationship is not always possible.

Lesson Learned 5

Current processes must be adapted for CBS acquisition, development, and sustainment.

This lesson concerns the need to modify existing processes to be suitable for the acquisition, development and sustainment of CBS. The developer's software and system engineering processes must be adapted to handle the integration of COTS software into the system. New processes must be added and existing processes updated to handle such activities as performing requirements trades against COTS software capabilities, evaluating COTS software against robust evaluation criteria, accounting for COTS software in safety, security and supportability analysis and design, and incorporating COTS software upgrades during development.

CBS development works best when iterative life-cycle models (e.g., spiral or evolutionary) and extensive prototyping of the COTS software in the system context are used together, and when the understanding gained from COTS software prototyping is integrated with the software architecture and design models. Furthermore, the time and effort distribution for development tasks need to be reallocated. Additional time and effort need to be spent on evaluation, prototyping, and analysis (the front end), and on integration and testing (the back end), and less needs to be spent on software implementation (the middle).

Numerous interviewees stressed the need for enhanced configuration management processes to handle the complexities of COTS software during both development and sustainment. The configuration management system must be able to manage multiple releases and patches to each release for each COTS software product. It must also be able to manage different configurations of COTS software at each development, sustainment, and operations facility (including mobile units), and even different configurations of COTS software on each computer hardware platform within each facility. For COTS software incorporated into firmware, configuration management cannot be performed at the board level but must be performed for the contents of the chips on the board.

Customer and user processes also need to be created or adapted to be suitable for the acquisition and sustainment of CBS. Examples include prioritizing user requirements, providing flexible and efficient responses to unexpected impacts due to problems encountered with COTS software, and handling the schedule variability of COTS software upgrades. Other examples include developing contracts compatible with the acquisition of CBS, and ensuring program milestones are compatible with the reallocation of time needed for CBS development schedules.

Interviewees also stressed the need for standardization of certain government processes as they relate to COTS software. Areas needing standardization include safety certification and security accreditation so that all parties understand the safety or security requirements that must be fulfilled when the system contains COTS software. In addition, standardized government processes for COTS software licenses need to be implemented to ensure that COTS software license currency is maintained, and that the COTS software licenses agreed to by the government are suitable for defense needs. The license for a COTS software product to be used by operational forces in the field, for example, should prohibit any expiring keys in the COTS software and should not contain any export restrictions.

Lesson Learned 6

Actual cost and schedule savings with CBS development and sustainment are overstated.

This lesson concerns the universal tendency to overestimate the cost and schedule savings due to COTS software usage (i.e., to underestimate the required cost and schedule for CBS development and sustainment). There are two principal components to this underestimation. First is completely overlooking or significantly underestimating tasks that must be performed in CBS development and sustainment, or costs of COTS software license

fees and other services. Second is not allowing enough cost and schedule margin to handle unexpected impacts that can occur due to problems with COTS software at any time during the life cycle.

Here are examples of frequently overlooked tasks: hands-on prototyping of COTS software (especially in a system context); acquisition of in-depth knowledge of COTS software (e.g., training mentors and tool-smiths and purchasing vendor support); installation and configuration of the COTS software in the development and operational facilities; and preparing integrated system training and documentation in addition to the vendor-supplied training and documentation.

Also, tasks to incorporate COTS software upgrades are almost always overlooked. Examples of such tasks include performing COTS software and system regression tests for each COTS software upgrade; implementing and testing software changes needed to support the upgrades (e.g., additions or changes to glue code, databases, or configuration files); and training developers and operators for each COTS software upgrade. The time and effort for these overlooked tasks generally cannot be obtained from software cost models, but must be estimated bottom-up and included in the total cost and schedule estimates.

One area where time and effort are significantly underestimated is software engineering. Software development time and effort are generally obtained by estimating the number of source lines of code and applying a software cost model. When the decision is made to use COTS software to obtain certain system functionality, the total number of lines of code is reduced by the number of lines of code that would have been needed to provide that functionality. Using this technique with a software cost model causes the elimination of all software development activities for that functionality from the cost and schedule estimates.

However, use of COTS software to provide functionality reduces only the amount of software design and implementation effort, not all software development activities. Software requirements analysis, architectural design, integration and testing, and qualification testing must still be performed, along with certain detailed design and implementation tasks. Even if the number of lines of glue code for integrating the COTS software is added to the total software size estimate, the resulting cost and schedule estimates are not sufficient to cover all of the necessary software development activities.

While some of the newer software cost models do have features available for estimating costs associated with COTS software, these models are not yet in widespread use, and the accuracy of the resulting estimates has not yet been calibrated in the defense software environment.

Other areas where time and effort are significantly underestimated are system engineering and system integration and testing. The system models and tools currently in use for cost and schedule estimation do not adequately address incorporating COTS software. The effort for system engineering and system integration and testing is commonly estimated as a percentage of the total development cost. When COTS software is used to provide some system functionality, the reduction in the software development effort causes a corresponding reduction in the system engineering and system integration and test effort. This reduction is not warranted since the same system engineering and system integration and testing for that functionality must still be performed, independent of whether COTS software or developed code provides the functionality.

Costs of COTS software license fees are also frequently overlooked or underestimated. The number of different COTS software products required to implement the CBS and the number of individual licenses required to be purchased are difficult to estimate, especially early in the life cycle before the design is known. Additionally, vendors usually charge for services not included in their standard licenses. Examples of such services are on-site vendor assistance during development or operations, and escrowing source code to protect against the possibility of the vendor going out of business.

CBS cost and schedule estimates almost never contain enough margin to handle the COTS software problems encountered in CBS development and sustainment. As described above, unexpected impacts can occur with COTS software at any time during the life cycle. The lack of appropriate margin results in cost and schedule overruns when COTS software-related problems occur. When cost as an independent variable (CAIV) is applied, the cost of handling unexpected COTS software problems can mean that system capabilities must be deleted to balance the cost. Cost and schedule estimates for CBS development and sustainment should always contain a planned margin (i.e., management reserve) for handling the unexpected COTS software problems that are certain to arise.

Acquisition Recommendations

The government needs to be an intelligent CBS buyer. Accomplishing this requires appropriate planning and contracting for CBS acquisition to handle the issues described in the lessons above. In particular, it should be noted that defense CBSs are almost never commercial items in themselves, but are large, complex, software-intensive systems, some of whose components contain COTS software. What is desired is a balanced solution among COTS, reuse, and newly developed software to meet the CBS cost, schedule, and performance objectives. Therefore, commercial item procurements (i.e., FAR 12 acquisitions) are almost never appropriate vehicles for acquiring defense CBSs.

To support defense programs in acquiring CBSs, it is recommended that several cross-program horizontal engineering initiatives be established. First, guidance for CBS life cycle cost and schedule estimation needs to be developed to address the problems described in Lesson Learned 6. Second, a repository for actual development and sustainment experiences with COTS software products (as opposed to vendor marketing information) needs to be developed and made accessible to CBS acquirers, developers, and sustainers. Lastly, specific CBS acquisition guidance that can be tailored to individual programs is needed, such as recommended contract structures, language for incorporation into contracts, and guidance for applying evolutionary acquisition.

Conclusion

The potential benefits of using COTS software in defense systems are extensive. Today's complex defense systems require

the leverage provided by COTS software, that is, enhanced system capabilities with reduced cost and schedule. The use of COTS software enables the government and developers to focus on providing the defense-unique needs.

This study demonstrated, however, that only careful acquisition, development and sustainment preparation and execution achieve the potential CBS benefits. CBS success depends upon preparing for a complex development and sustainment effort; preparing for inherent cost, schedule, and performance risks beyond government or developer control; and preparing to make adjustments to current acquisition, development and sustainment processes. While this study was conducted on defense space systems, the authors believe that the lessons learned are not limited to that domain, but are widely applicable to the use of COTS software in any large, software-intensive system.◆

NOTES

1. See, for example, DoD Directive 5000.1, Oct 23, 2000, paragraph 4.2.3.
2. For this paper, a COTS-based system (CBS) is defined to be a system that contains commercial-off-the-shelf *software* products as elements of the system.

About the Authors

Richard J. Adams is a senior engineering specialist at The Aerospace Corporation with more than 30 years experience in software engineering, software project management and software acquisition. Previously he worked for TRW, where he developed software and managed software development projects for DoD software-intensive systems. He has a bachelor's degree in mathematics from California State University at Long Beach.

The Aerospace Corporation
Mail Station M1/112
P.O. Box 92957
Los Angeles, CA 90009-2957
Phone: 310-336-2907
Fax: 310-336-4070
E-mail: Richard.J.Adams@aero.org

Suellen Eslinger is a distinguished engineer at The Aerospace Corporation with more than 30 years experience in software engineering and software acquisition. Previously she worked at Computer Sciences Corporation and General Research Corporation where she developed software and managed software development projects for DoD and NASA software-intensive systems. She has a bachelor's degree from Goucher College and a master's degree from the University of Arizona, both in mathematics.

The Aerospace Corporation
Mail Station M1/112
P.O. Box 92957
Los Angeles, CA 90009-2957
Phone: 310-336-2906
Fax: 310-336-4070
E-mail: Suellen.Eslinger@aero.org

Coming Events

June 11-13
E-Business Quality Applications Conference
qaiusa.com/conferences/june2001/index.html

June 18-22
ACM/IEEE Design Automation Conference
www.dac.com

June 25-27
2001 American Control Conference
acc2001.che.ufl.edu

July 1-5
Eleventh Annual International Symposium of the International Council on Systems Engineering
incose.org/symp2001

July 19-21
2nd Int'l Symposium on Image and Signal Processing and Analysis ISPA'01
ispa.zesoi.fer.h

August 5-10
HCI International 2001: 9th International Conference on Human-Computer Interaction. (UAHCI 2001)
hcii2001.engr.wisc.edu

August 27-31
Fifth IEEE International Symposium on Requirements Engineering
www.re01.org

August 27-30
Software Test Automation Conference
http://www.sqe.com/testautomation

September 10-14
Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-9)
www.esec.ocg.at

October 29-November 2
Software Testing Analysis and Review
http://www.sqe.com/starwest

February 4-6, 2002
International Conference on COTS-Based Software Systems (ICCBSS)
At the Heart of the Revolution
http://www.iccbss.org