The Commandments of COTS:
Still in Search of the Promised Land

David J. Carney and Patricia A. Oberndorf
Software Engineering Institute, Carnegie Mellon University

This article examines current government trends toward using commercial-off-the-shelf (COTS) products. It discusses both the positive and the negative effects of these trends and suggests some high-level issues for policy makers to consider.

Within the past few years, organizations that acquire software-intensive systems have undergone a remarkable change in emphasis toward use of existing commercial products. This shift is especially noticeable in U.S. government procurements, particularly those of the Department of Defense (DoD). Many current Requests for Proposals (RFPs) being issued by the government now include a mandate concerning the amount of COTS products that must be included. Statements from senior officials who support this mandate indicate that increased use of COTS products is intended to be department- or even government-wide. This new interest needs to be examined in terms of its causes and effects, as do the benefits and liabilities of using COTS products.

In this article we offer some observations on all of these aspects and voice some specific concerns and criticisms. We stress that our observations are essentially cautionary, not condemnatory. It is obvious to most observers that the huge growth in software costs will continue, not abate, and that appropriate use of commercial products is one of the remedies that might enable us to acquire needed capabilities in a cost-effective manner.

Where use of an existing component is both possible and feasible, it is no longer acceptable for the government to specify, build, and maintain a comparable product when a commercial solution is available. On the other hand, we also notice many disturbing signals abroad in the land: many statements, whether high-level policies or otherwise, suggest a reluctance to admit that there can be a nontrivial downside to using COTS products. While it may not yet be readily apparent to all observers, many unhappy trade-offs exist when embracing a commercial basis for the government's software systems.

The critical point is that using COTS components in any given circumstance may either be beneficial or it may cause greater problems. Acquisition managers must understand that choosing a COTS component may be a reasonable solution; however, the decision to use COTS should be the product of analysis, reasoning, and engineering decisions, not the desire to jump on the latest bandwagon. Hence we express, using a somewhat whimsical structuring device, some specific issues to consider when planning to use COTS products. Unlike the Commandments of the Exodus, our commandments are not graven in stone. But they are as surely needed. Just as was true in that far-off time, we see abundant evidence that people are still willing to worship before false idols.

The Ten Commandments of COTS
Discussion

I. Do not believe in silver bullets.
There seems to be universal agreement throughout the software community that "there is no silver bullet," and that the lessons found in Brooks' famous paper have been learned [1]. Yet the silver bullet mentality seems to reappear in the government with remarkable frequency. We note a long series of software initiatives, e.g., Ada, computer-aided software engineering (CASE) tools, SEEs, reuse, that placed all-encompassing hopes on particular technologies to answer impending crises. The results seldom matched the expectations; nonetheless, the technologies in themselves were not necessarily failures, e.g., Ada's inability to find broad success was not due to any inherent technical weakness in the language itself; many CASE tools are indeed valuable and useful products.

A technological crisis, including all of the software crises that have come and gone, is generally the result of many factors. While there may be one overriding factor at the base of the problem, there are likely to be many other contributing factors that participate and catalyze. Therefore, an approach that chooses a single factor, isolates it, then hopes for a universally beneficial outcome will usually have little success. While the desire for a simple and uncluttered remedy is understandable, this desire is unlikely to be satisfied. Thus it is the single-mindedness of the response, not whatever "silver-bullet" technology is at hand, that is inherently flawed.

We perceive that the current movement toward COTS products is rapidly acquiring a "silver-bullet" approach to problem solving. Though growth in the cost and complexity of software systems is a difficult problem, to focus only on a commercially based solution is
to misunderstand the problem's intricacies. Therefore, to mandate beforehand that some arbitrary percentage of a system must be COTS products will not have the desired effect. Instead, the real need is for

- thoughtful policies about the types of systems that will or will not benefit from a commercial approach.
- guidelines about the hard trade-offs made when incorporating COTS products into systems.
- handbooks that describe recommended processes and procedures about integrating multiple commercial products.
- upgrade strategies to cover multiple vendors.
- recommendations about when not to use a commercial approach.

Further, these items must be recognized as only part of the solution and must be accompanied by similar considerations of the complementary problems—system distribution, interface standards, legacy system reengineering, etc.—with which a COTS-based approach must be integrated. In short, it is essential that using COTS products be seen as one potential strategy in a complex solution space: no more and no less.

II. Use the term precisely (and demand like behavior from others).
The Federal Acquisition Regulation (FAR) provides a definition of both "commercial item" and "nondevelopmental item." The thrust of these definitions is to clarify the government's understanding of what it means to be a "commercial product." While these definitions are useful, they leave some issues unresolved. For instance, they specify that a product must be either "sold, leased, or licensed to the general public, offered for sale, lease, or license to the general public, or will be available in the commercial marketplace in time to satisfy the delivery requirements." But exactly how far in advance is necessary to satisfy delivery requirements? And in which particular sense must the product have been advertised? These questions are not trivial, for they can have major importance when evaluating proposals for source selection.

A further ambiguity stems from the unfortunate parallelism with the terms "GOTS" (government-off-the-shelf) and "MOTS" (modified-off-the-shelf). Both are considered related to—and sometimes even deemed subsets of—COTS. A GOTS component is one developed by and owned by the government (generally including the source code), and a MOTS component is one in which some alteration has been made to an existing component or product. However, to consider all of these as roughly synonymous is misleading. Given that one prime motivation for choosing COTS is to lower maintenance costs, the term should apply only when source code is unavailable and maintenance impossible (except by its vendor). In that sense, MOTS is fundamentally different from COTS, since any code that is modified for a specific acquisition will necessarily need ongoing maintenance for the life of that system.

Aside from the precise nuance of meaning, there is also the issue of scope. What size component do we include when we say COTS? Is it meant to be all-inclusive, covering any software component from a Web browser to an entire air traffic control system? (If so, then a single technical policy is being applied equally and across the board to systems of 10,000 lines and systems of ten million lines; in short, one size fits all.) The impetus
for the government's current bias toward COTS products is the intuitive belief that in constructing a complex system, there are often some cases where "the item I want already exists on someone else's shelf."

In this general sense, it is moot whether that shelf is commercial or otherwise, since in both cases the acquiring organization desires to delegate the creation of certain functionality elsewhere. The key issue is the government's heightened willingness to consider the classical "make vs. buy" question often faced by industry. But in any sense other than this simple one, there are shades and nuances of meaning that may or may not be significant for a particular acquisition. These subtly different meanings provide a deceptive level of ambiguity.

III. Understand the impact of COTS products on the requirements and selection process.

There is widespread agreement throughout the software community on the importance of a requirements specification. Anecdotal evidence suggests that the requirements specification can be the single most important factor in the success of an acquisition. A COTS bias will have obvious impact both on the specification of requirements and on the evaluation of proposals. This bias necessarily implies either that some software requirements will be written to describe existing products or that the requirements are malleable enough to be implemented with a variety of existing products. Said differently, someone (whether the requirements author or not) must choose which requirements can bend to the exigencies of the marketplace and which cannot.

This is not unrealistic: requirements are not, after all, written in a vacuum, and for any given component of a software-intensive system, the author of its requirements can distinguish absolute from desirable properties of the system. But for those requirements that are expected to be amenable to a commercial solution, the requirements author must have some notion of how bidders will respond. If, for instance, an acquisition of some CASE tools includes a project management tool (and the expectation from most bidders is a COTS item), the author's knowledge of the existing CASE marketplace will guide the description of required functional features. Anything else would be self-contradictory: soliciting bids for a commercial product, yet describing functional capabilities for which no commercial instances exist. Paradoxically, the requirements must also be sufficiently generic, for unless we assume that the author of the requirements has already decided on a specific product, the requirements must be broad enough to accommodate the differences between comparable commercial products.

A COTS bias can also have an impact on the evaluation process. Selection criteria become difficult to define when choosing between two comparable products such as FrameMaker and Interleaf: given these choices, what clearly distinguishes them? What aspects of selecting one or the other will survive a challenge to an award?

Another difficulty for source selection stems from the looseness of the term "COTS" described earlier. Will each acquisition refine what it means by "COTS"? Will all acquisitions abide by the government-wide definition, e.g., the FARs? Are there minimal yearly sales figures by which some component is acceptably COTS? Or might it be enough that the bidder propose some arbitrary component and state that it is "commercially available"? There have even been suggestions (at least in the presence of the authors) that the COTS directives might eventually be phrased such that "at least 50
percent of the bidder's proposal must be COTS." If so, how is the presence of "50 percent" determined? Half of all Computer Software Configuration Items (CSCIs)? Half the lines of code (that cannot be determined with COTS products)? Half the "shall" requirements? Presuming that the 50 percent figure could be determined, does it also logically follow that with all other things being equal, 65 percent COTS is *de facto* preferable to 55 percent?

**IV. Understand COTS impact on the integration process.**

The current thrust toward COTS products takes place in the context of COTS-based systems. That is, while there are demonstrable benefits in purchasing a stand-alone Oracle database management system (DBMS) rather than creating its functional equivalent from scratch, the current COTS initiatives are actually focused on systems. It is into these systems that the government plans to introduce commercial components, ultimately in the hopes that it will result in lower costs (from lower maintenance requirements) as well as give the system a "plug and play" character (from the expected high degree of integrability). The phrase "plug and play" is significant, because it is the unspoken motivator for much of the current interest in COTS products. It conjures up images of a software environment wherein heterogeneous components can not only be easily inserted or replaced but also seamlessly interconnected and interoperated. The concept is borrowed from the conditions found in the hardware world, where some degree of plug and play does exist: boards from one maker, cables from another, printers, monitors, keyboards—all easily and independently purchased, replaced, and upgraded.

In terms of software, this tempting picture is a myth and will remain so for several years. Software as typically written today shares few of the integrability characteristics of hardware. There are no widely observed functional boundaries of a component, i.e., while a keyboard performs a single bounded role in the hardware world, there is no such analog for a design tool. Even for those software items that do have a relatively clean demarcation of functionality, such as a DBMS, the vendors of such products tend to bundle that functionality with other capabilities, e.g., a DBMS might provide a GUI builder and 4GL processors, etc. Individual products are unique collections of tools and utilities, the internal interfaces of which are generally obscured. These products, though extremely useful, are not readily "plugged" since different products exhibit or obscure different interfaces, and a fatal mismatch is almost certain when one product is replaced by another.

The plug-and-play notion also rests on assumptions about data that different components will share, again by analogy to the hardware world. It is essential to note the nontrivial differences between the two worlds. On the one hand, low-level protocols for hardware tend to be semantically simple (as are agreements on physical things such as pin configurations); on the other, the information that crosses software interface boundaries is far richer semantically. A data item for a graphic design tool will include many levels of information unique to that tool: details of access control, relationships with other data items, ownership, longevity, and so forth. Vendors of such components tend to keep details like these private (a logical outcome of the decision to incorporate multiple functionalities into a single product). The result is that until there are wide agreements from many software vendors about shared data, even if it becomes possible to plug a new component into a software system, it is not likely it will play.
V. Understand COTS impact on the testing process.

It is a truism that testing a system is quite different from testing its constituent parts. The difference is especially pronounced when the system makes use of previously existing software components, a fact dramatically demonstrated in the findings after the recent explosion of the European Space Agency's Ariane-5 rocket. Software from the earlier version of the rocket (Ariane-4) had been reused.

The error occurred in a software module [that] computed meaningful results only before liftoff. As soon as the launcher lifts off, this function serves no purpose. [The purpose of this function] is based on a requirement of Ariane-4. ... The same requirement does not apply to Ariane-5, which has a different preparation sequence and it was maintained for commonality reasons, presumably based on the view that, unless proven necessary, it was not wise to make changes in software that worked well on Ariane-4. [2]

We cite this instance because it provokes highly relevant questions that should be asked when deciding to use any previously written component in a system: What types of testing, both at the unit level and at the system level, are possible? What types of testing are necessary? In the example cited above, the requirement that drove the creation of a module for the earlier system was overlooked (or, more likely, forgotten) when the module was later reused, and whatever testing was performed made no provision for it.

So when system designers today are faced with the choice of using a COTS component, how are they to determine the testing that will be necessary? What requirements drove the creation of that component? Are they documented? Are they complete? Even at the subsystem level, how does one do unit testing for a COTS component? Current research is only beginning to grapple with these questions; it is vital that we have some better notions of how they might be answered. Widespread incorporation of COTS components into systems may not result in failures as spectacular or as costly as that of Ariane-5, but for at least some systems, similar misunderstanding about requirements will occur, and there will most certainly be failures.

VI. Realize that a COTS approach makes a system dependent on the COTS vendors.

The role of the components vendors can be a decisive factor in successfully creating and maintaining a commercially based software system. The following aspects are especially significant:

- What standards govern the product in question? In spite of the current interest in the role of standards for commercial products, there is still the danger of a system becoming overly dependent on a particular vendor's product. This commonly occurs when a product has a number of desirable but nonstandard features. Such features are often attractive, but they exact a price, since a system that relies on them will result in what is commonly called "vendor lock"—becoming captive to the products of a single vendor or set of vendors.

- Does the vendor supply adequate documentation for the component in question? Some products offer extensive and useful documentation, but this is by no means normative. (This point can be easily verified by any system administrator who has
tried to follow installation instructions.) Installation documentation is usually insufficient: do programmer reference manuals or maintenance manuals exist? More to the point: are they well written? Are they accurate? Documentation is not the only issue: what kind of user support is available? Is the company responsive to user inquiries? If the component is to be used in a critical system, what is the availability of that support, i.e., is it 24-hour support?

- If the component is to be part of a system that will operate for several years, what are the probabilities that the company will exist for that time? There are many anecdotes throughout the software community about products' vendors going out of business, leaving clients with nonfunctional tools and unretrievable data. Such anecdotes are, unfortunately, not entirely legendary.

- Even assuming that the company exists, how long will it support the COTS product in question? Vendors often phase out their support for any given product. Would such an occurrence have an impact on the maintenance of the system in question? These questions are neither new nor profound. To a large extent, they have been relevant in many acquisitions for years, e.g., questions like these are always in the mind of a buyer of CASE tools. But they demand reasonable answers, and as they become applicable to more and more parts of a complex system, the need for those answers grows as well.

VII. Realize that maintenance is not free.

Since much of the motivation for the movement toward COTS-based systems is the expectation of easier and cheaper system maintenance, it is necessary to examine some aspects of how maintenance and upgrade activities can be affected in a system with numerous commercial components. Upgrading a COTS-based software system means that as new releases of the commercial components are made by the various vendors, the system will incorporate them. Though it is possible to occasionally skip over a release, e.g., ignore a release of something like 3.03A, it is still the case that vendors tend to support only a limited number of versions, and a program that ignores a vendor's new releases cannot survive in the long term.

A system's commercial components should be as up-to-date as possible. A system with several commercial components has an extremely heavy dependency on the various release cycles of the COTS vendors. Different pieces of the system will be upgraded at widely varying intervals, which will require that licenses be renewed periodically. It should be kept in mind that component upgrade can result in numerous unforeseen problems: incompatible files and databases, different naming conventions, and introduction of new conflicts between COTS components. Depending on the number of COTS components and different COTS vendors, the effect of these multiple dependencies can vary from short-term user inconvenience to total system instability.

Another costly item for maintenance of COTS-based software systems results from the degree to which those systems exhibit an integrated character. We earlier suggested that the realities of the software world are quite different from those of the hardware world, and COTS software components are seldom built to plug into an existing system easily. The usual way to overcome this deficiency involves "wrappers," "bridges," or other "glueware." These terms refer to third-party software that performs whatever integrating functions are necessary: trapping output from one component and
reformatting it for input to another, sending notification messages about one tool's completion to another for start-up, and so forth. Most heterogeneous software systems that exhibit "integratedness" are built precisely this way. However, this technique is not one that necessarily leads to lower maintenance costs: writing wrappers can be a complex activity, requiring expertise both at the detailed system level and in the COTS components being wrapped. Wrappers are often "point-to-point" solutions, which means that when a vendor releases a new version, any wrapping that involves the new component will potentially need to be upgraded. Given the random vendor release cycles described above, keeping the glueware current and up-to-date for an integrated system of any complexity can become a maintenance nightmare.

VIII. You are not absolved of the need to engineer the system well.
The discipline of engineering is no less critical to a COTS-based system than to any other type of system; in some particulars it could be even more critical. The reality of today's available COTS products is that few of them are designed to work together. Many have been created to be used stand-alone and to require no collocation (let alone interaction) with any other product or component. Even when they have been designed to cooperate with another product, it is most often another product from the same vendor or from another vendor with whom the first vendor shares some special interest. A COTS-based system is still a system with its own requirements, both developmental and lifecycle. This system will need to be designed, brought together, tested, and managed the same as any other system that has been built or acquired in the past.

IX. Just "doing COTS" is not an automatic cost saver.
Currently, there is little government experience with building and fielding significantly COTS-based systems over any meaningful part of an overall system lifecycle. So far, we have more promises and wishful thinking than facts or verified cost models on which to base our enthusiasm. And what is behind this enthusiasm? In part, it is the observation of COTS-based revolutions in other industries, e.g., automobiles, which appear to hold out the hope and expectation that using COTS products will similarly lower software system costs. The analogy is compelling. It makes sense that the proration of development and upgrade costs across multiple users and competition between suppliers should lead to cost savings for each part of any system that derives from the COTS world. But note that most of this argument is based on per-unit component cost considerations. What about the system costs? The process involved in understanding the COTS products as (potential) system components is far too complex to be satisfied by a glossy brochure and vendor promises. And sometimes all the available documentation together is not enough to describe the full behavior of a product or the implications of that behavior for the integration of the component into the system. Other hidden costs include

- market research to find the COTS products that are suitable.
- product analyses to select among alternatives.
- licenses and warranties, especially if the warranty available to the general public does not suit your needs.
In addition to these items, many of the aspects of COTS-based systems previously cited in this article can have still greater costs:

- integration of all those diverse products into a system.
- keeping up with the asynchronous upgrades of the various products (this includes reacquiring the knowledge needed to be sure each will continue to be a contributing member of the system, not to mention individual upgrade costs—vendors do not hand out new releases for free!).
- coordination of the many support vendors during the system lifecycle.
- recovery when a vendor discontinues a product or goes out of business.

In any given situation, the costs that prevail depend on the individual system and circumstance, the specific risk-mitigation strategies, and the management skills at hand.

X. Just "doing COTS" must be part of a large-scale paradigm shift.
In most college curricula, the introduction to software and computer science still consists of learning one or more programming languages. Students learn to write whole systems and subsystems, designing them from a blank piece of paper, then coding, debugging, and testing them. In contrast, when using existing products as components in a system, one must determine how to get them to cooperate with one another. This will often necessitate writing wrappers to achieve the desired cooperation and integration, a process that will undoubtedly need to be repeated to keep the components cooperating as each product changes (usually independently).

The shift from the first process (building from scratch) to the next (integration of ready-made components) constitutes a significant paradigm shift for programmers and system developers. And by extension, the testing, quality assurance, and maintenance employees have to cross over to that same new mind-set. Changes to all these positions also require that managers modify the expectations they have and the techniques they employ.

In other words, the switch to COTS-based systems is not just a technological change; it profoundly affects many people in their several roles. Organizations can be equally impacted, experiencing modifications in the activities they undertake, their structures and relationships, their required training, corporate policies, relationships between government and contractors, and relationships across the marketplace. This paradigm shift toward integration of others' products—from a producer to a consumer mentality—has widespread effects. The worst thing one can do is consider it a mere change in technology.

Afterthought
Much has been made, and will continue to be made, of "the software crisis." It periodically reasserts a presence in the government's consciousness, and its symptoms—spiraling software costs, growing system complexity—are valid causes of concern. But perhaps this phenomenon is less a crisis than is generally thought. Perhaps it is simply the case that the growth of software's cost and complexity is (at least partially) the natural result of the engineering decisions being made; that is, given that a greater and greater proportion of a system's functionality is being allocated to software, while the system's
functional capabilities are increasing, it is inescapable that the cost and complexity of the software will proportionately grow as well. That this reallocation engenders a crisis is too often caused less by the software than by key system decisions being made without proper understanding of the constraints and limitations of software or without appropriate input from software (as opposed to system or hardware) experts.

There is another factor that relates to this impending crisis; namely, the belief that more complex systems can be had for less cost. Although it is sometimes the case, certainly with computer hardware, that capability grows as cost diminishes, it is doubtful that this is always a reasonable expectation. In the case of software, it flies in the face of numerous bitter experiences and contradicts plain folk wisdom—"you get what you pay for." Thus, while such slogans as "better, cheaper, faster" represent attractive ideals, it is naive in the extreme to believe that to merely state the slogan will make it come true.

A more realistic goal should be to demand awareness, on the part of all high- and mid-level employees, of the pragmatic realities of software-intensive systems. It is not reasonable to speak of buying fully integrated CASE environments when there is no agreement among environment experts as to what "fully integrated" means. It is not reasonable to expect that completely interoperating software systems appear on demand when in spite of recent advances in gluing disparate systems together, Apple users and Sun users still find themselves at an impasse when trying to share a word processor file. And it is not reasonable for the government to continue the cycle of large-scale procurements, initiatives, thrusts, plans, or whatever else (most of which have consumed many precious dollars and provided precious little benefit) without a more solid grounding in the technical realities of software practice.

This grounding must be based on knowledge, not slogans. It is all well and good to hope that using commercial software components will save money in the long run. As we stated on the first page of this article, we fully concur with the idea of using COTS products when appropriate and when warranted. But the savings will not come automatically, and they will not be the result of applying a simplistic solution to a complex problem. Anyone who promises otherwise is offering us a golden calf to worship. Following that path did not work in years gone by, and it is not likely to work today.

About the Authors

David Carney is a member of the technical staff in the Dynamic Systems program at the Software Engineering Institute (SEI). Before joining the SEI, he was on the staff of the Institute for Defense Analysis in Alexandria, Va., where he worked with the Software Technology for Adaptable, Reliable Systems program and with the NATO Special Working Group on APSE. Prior to that, he was employed at Intermetrics, Inc., where he worked on the Ada Integrated Environment project.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Voice: 412-268-6525
Fax: 412-268-5758
E-mail: djc@sei.cmu.edu
Tricia Oberndorf is a member of the technical staff at the SEI. She is a part of the Dynamic Systems program and concentrates on the investigation of integration and open system issues. She has spent much of her career investigating a number of other integration and open systems questions in the context of CASE environments and other kinds of systems. Prior to joining the SEI, she worked for the Navy for over 19 years.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Voice: 412-268-6138
Fax: 412-268-5758
E-mail: po@sei.cmu.edu

References