manager

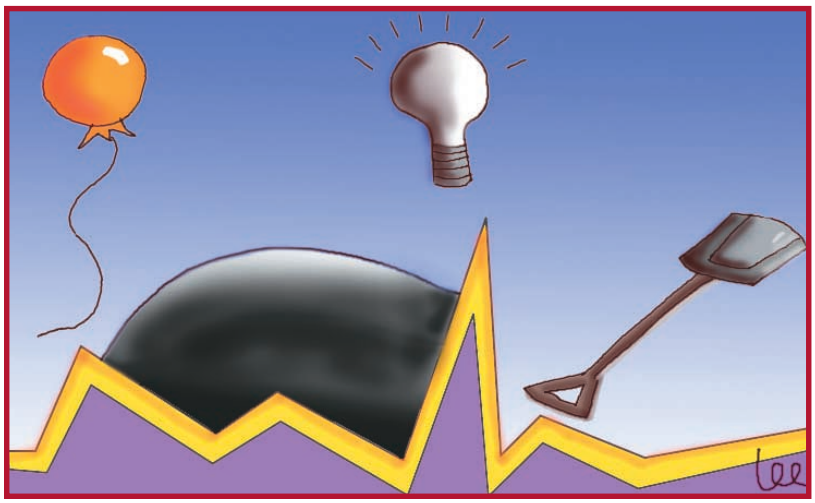
Editor: Donald J. Reifer ■ Reifer Consultants ■ d.reifer@ieee.org

Eight Lessons Learned during COTS-Based Systems Maintenance

Donald J. Reifer, Victor R. Basili, Barry W. Boehm, and Betsy Clark

In the May 2001 issue of *Computer*, two of us published an article called "COTS-Based Systems Top 10 List" (pp. 91–93). The list identified 10 hypotheses that served as challenges for enhancing our empirical understanding of commercial off-the-shelf software. These hypotheses were primarily related to COTS-based systems development.

CBSs remain one of three focus areas (the other two are defect reduction and agile methods) of both our US Federal Aviation Agency-sponsored research and our National Science Foundation-sponsored Center for Empirically Based Software Engineering (CeBASE) efforts. As our original article said, COTS software usage remains relatively immature as it progresses through a peak of inflated expectations, a trough of disillusionment, a slope of enlightenment, and a plateau of productivity. Based on presentations at the recent 2nd International Conference on COTS-Based Software Systems (*ICCBSS 2003 Proceedings*, Springer-Verlag), risks inherent to COTS use are large—as are the potential returns. Pursuing custom solutions remains unattractive primarily because it takes so much time and effort to develop software products.



We have recently extended the Top 10 list of challenges in the original *Computer* article to encompass the life cycle's maintenance phase. During maintenance, COTS products undergo a technology refresh and renewal cycle. As part of this activity, maintainers decide whether to upgrade their COTS products or retain old versions. If they choose to retain old versions, they'll eventually reach the point where the vendor no longer supports those versions. If they choose to update, they must synchronize the associated update with their release cycle and with product updates other vendors are making. They must also coordinate the update of wrappers and glue code so that they will work with the new versions.

Because COTS maintenance is relatively im-

0740-7459/03/$17.00 © 2003 IEEE

mature, we present empirical results gathered to date as lessons learned rather than as either results or hypotheses. We hope these results will help those trying to manage COTS maintenance.

Lesson learned 1

The refresh and renewal process for CBSs must be defined a priori and managed so COTS package updates can be synchronized with each other and the organization's release and business cycle. If they aren't, updates might occur sporadically during the maintenance part of the cycle and the risk of technology obsolescence might increase dramatically.

Source. A recent US Air Force Scientific Advisory Board study (SAB-TR-99-03, April 2000) surveyed 34 COTS-based systems to look at COTS software management within weapon systems.

Implications. Currently, few COTS software lifecycle models address CBS maintenance processes. Guidance is needed to define refresh and renewal process activities. We must also define criteria for making decisions regarding when to incorporate updates within releases, along with those criteria's associated risks and business implications.

Lesson learned 2

COTS software capability and quality evaluation must be managed as a continuing task during the maintenance phase.

Source. Most publications that discuss CBS processes advocate that companies establish a market watch function (see lessons-learned papers in COTS workshops such as ICCBSS 1 and 2 and in research reports by the National Research Council of Canada, the Software Engineering Institute, and the University of Southern California).

Implications. Most COTS software studies recommend that firms not only establish a market watch function to keep track of where their packages are heading but also that they continuously assess their options. A market watch looks at the marketplace as a whole, monitoring a specific vendor's health and viability as well as what competitors are coming out with. COTS evaluation gives you a detailed assessment of package capabilities, quality issues, and future options. It typically involves conducting some form of operational demonstration.

Lesson learned 3

The cost to maintain COTS-based systems equals or exceeds that of developing custom software. Maintenance in this context involves updating CBSs with new releases, modifying wrappers and glue code, and incorporating fixes and repairs into the system.

Source. Reifer Consultants recently studied the cost of COTS software across 16 systems, some of which employ over 40 different packages, across three large firms. Costs average 10 percent of the development cost per year over a projected 10-year life for the system. Although releases occur every year, COTS technology refreshes occur every two years or across two releases. Defect rates per release for CBSs are poorer than for custom-built software, averaging 10 to 40 percent higher.

Implications. Even though firms can save time and effort during development using CBSs, they should evaluate the total lifecycle cost of options prior to making commitments. Such analysis could identify risks that negate many of the advantages that CBSs bring to the table. For example, firms must coordinate glue code updates along with package improvements. Considering that a line of glue code costs, on average, three times that of a line of custom code to develop and maintain, maintenance effort can get quite expensive. In situations where CBSs have a long life, custom solutions might work out to be cheaper than COTS alternatives. Project managers whom RCI interviewed also said that, unlike custom systems, COTS-based systems need a continual stream of funding throughout their life cycle. Such funding is necessary to keep up with a dynamic marketplace in which vendors are continually releasing new versions. Funding was an issue with several of the projects in this study because their maintenance budgets often get cut. The managers believe that this hurts a CBS more than a custom system because they can delay maintenance on the latter if they have to because of budget limitations.

The cost to maintain COTS-based systems equals or exceeds that of developing custom software.

Lesson learned 4

The most significant variables that influence the lifecycle cost of COTS-based systems include the following (in order of impact):

- Number of COTS packages that must be synchronized within a release
- Technology refresh and renewal cycle times
- Maintenance workload (the amount of effort software engineers expend to handle the task at hand) for glue code and wrapper updates
- Maintenance workload to reconfigure packages
- Market watch and product evaluation workload during maintenance
- Maintenance workload to update databases
- Maintenance workload to migrate to new standards
- COTS maintenance license costs

Source. The RCI study mentioned earlier was a source here also. The study identified these parameters using

a survey that asked those responsible for maintenance for insight. The number of packages requiring synchronization was twice as sensitive as the need to migrate to new standards.

Implications. Cost models like USC's COCOTS (see http://sunset.usc.edu for information) should be updated to encompass the full CBS life cycle. Currently, they focus on estimating the costs associated with evaluating, adapting, and deploying COTS software packages during development and maintenance. In the future, such models should incorporate additional variables such as the last three bullets on our list to permit those assessing lifecycle costs to estimate the full cost of the maintenance portion of the CBS life cycle.

Lesson learned 5

Maintenance complexity (and costs) will increase exponentially as the number of independent COTS packages integrated into a system increases.

Source. USC's COCOTS team has initial results of a study of 20 projects.

Implications. Projects should understand the maintenance implications of integrating a large number of COTS products into a system. In addition to the effort involved in the initial integration, they should consider that each product will evolve in its own way, according to different timetables, at the vendors' discretion. They will have to expend considerable effort to handle these products' continuing evolution (for example, understanding the impact of an upgrade on the rest of the system or making changes to glue code.

Lesson learned 6

Software engineers must spend significant time and effort up front to analyze the impact of version updates (even when the decision is made not to incorporate the updates).

Source. Initial results of the COCOTS team's 20-project study suggest that analysis efforts during maintenance directed toward updates can tax the organization severely. This is particularly true for safety-critical systems.

Implications. Maintenance modeling must assume that CBSs incur fixed and variable costs. Fixed costs are those associated with market watch and continued product evaluation. Variable costs are a function of the work performed to incorporate updates, fixes, changes, and optimizations into the impending release. The workload performed by the fixed staff must be optimized (balanced) as part of this process.

Lesson learned 7

Flexible CBS software licensing practices lead to improved performance, reliability, and expandability.

Source. RCI performed surveys in 2000 and 2001 on best acquisition practices for the US Army (see www.reifer.com for a paper on innovative licensing).

Implications. The studies identified partnering instead of conflict management as the preferred approach to licensing. Shared goals lead to products with improved "goodness of fit" and "functionality" for the buyer. Leveraging relationships to achieve shared goals is highly desirable. Innovative contracting under such arrangements lead to deep volume discounts and priority service and bug fixes. Traditional approaches to licensing, where contracts instead of relationships govern, lead to distrust and poor results.

Traditional approaches to licensing, where contracts instead of relationships govern, lead to distrust and poor results.

Lesson learned 8

Wrappers can be effectively used to protect a CBS from unintended negative impacts of version upgrades.

Source. Several projects were interviewed for the COCOTS database. One project successfully used wrappers for information hiding so that different versions of COTS products (or different products) could be swapped without affecting the rest of the system.

Implication. CBS architectures should accommodate COTS changes throughout the system life cycle.

To make better decisions relative to CBSs, we need empirical knowledge. To gain this knowledge, we must more fully understand the lifecycle processes people use when harnessing COTS packages. The initial findings reported here are but the first step in our attempts to capture this empirical knowledge. We plan to continue collecting data and investigating the phenomenology of COTS-based systems. This work complements the more general results available at our CeBASE Web site (http://cebase.org/cbs) and the SEI Web site (www.sei.cmu.edu/cbs). We welcome your comments, input, and contributions.

Donald J. Reifer is a visiting associate with the Center for Software Engineering at the University of Southern California and president of Reifer Consultants Inc. Contact him at dreifer@earthlink.net.

Victor R. Basili is a professor in the Computer Science Department at the University of Maryland and director of the Fraunhofer Center–Maryland. Contact him at basili@cs.umd.edu.

Barry W. Boehm is director of the Center for Software Engineering at the University of Southern California. Contact him at boehm@sunset.usc.edu.

Betsy Clark is president of Software Metrics and works with USC on several CBS projects. Contact her at betsy@software-metrics.com.