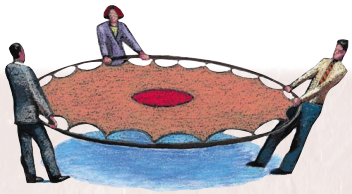
COTS Integration: Plug and Pray?

Barry Boehm and Chris Abts,
University of Southern California



The advantages of COTS products can be undermined by poor support and changing feature sets. Learn the common COTS pitfalls and how to avoid them.

For most software applications, the use of commercial off-the-shelf products has become an economic necessity. Gone are the days when upsized industry and government information technology organizations had the luxury of trying to develop—and, at greater expense, maintain—their own database, network, and user-interface management infrastructure. Viable COTS products are climbing up the protocol stack, from infrastructure into applications solutions in such areas as office and management support, electronic commerce, finance, logistics, manufacturing, law, and medicine. For small and large commercial companies, time-to-market pressures also exert a strong pressure toward COTS-based solutions.

COTS PLUSSES AND MINUSES

However, most organizations have also found that COTS gains are accompanied by frustrating COTS pains. Table 1 summarizes a great deal of experience on the relative advantages and disadvantages of COTS solutions.[1] One of the best COTS integration gain-and-pain case studies[2] summarizes the experiences of David Garlan's group at CMU. Garlan's group tried to integrate four COTS products into the Aesop software architecting environment—the OBST object management system, the Mach RPC Interface Generator, the SoftBench tool integration framework, and the InterViews user interface manager—only to find a number of architectural mismatches among the products' underlying assumptions. For example, three of the four products are event based, but each has different event semantics, and each assumes it is the sole owner of the event queue. Resolving such model clashes escalated the original two-person, six-month project into a five-person, two-year project: a factor of four increase in schedule and a factor of five increase in effort.

Such experiences, and the more general technical and business issues shown in Table 1, indicate that COTS integration differs significantly from traditional software development and requires significantly different approaches to its management. At the USC Center for Software Engineering COTS Integration Affiliates' Workshop, we identified four key COTS integration issues: functionality and performance, interoperability, product evolution, and vendor behavior.

FUNCTIONALITY AND PERFORMANCE

You have no control over a COTS product's functionality or performance. If you modify the source code, it's not really COTS—and its future becomes your responsibility. Even as black boxes, big COTS products have formidable complexity; Windows 95, for example, has roughly 25,000 entry points.

Let the buyer beware

COTS-based projects cannot make blanket assumptions about system requirements or embrace traditional process models. One mistake developers can make is to use the waterfall model on a COTS integration project. With the waterfall model, you specify requirements and these determine the system's capabilities. With COTS products, it's the other way around: The capabilities determine the "requirements," or delivered-system features. If your users have a "requirement" for a blinking cursor and the best COTS product doesn't provide it, you're out of luck.

Another potential danger involves using evolutionary development with the assumption that every undesired feature can be changed to fit your needs. COTS vendors do change features, but in response to the overall marketplace, not to individual users' needs. It's also unwise to assume that advertised COTS capabilities are necessarily real. COTS vendors may have had the best of intentions when they wrote the marketing literature, but that won't help you when the advertised feature isn't there.

Barry Boehm, Computer Science Department, University of Southern California, Los Angeles, CA 90089; boehm@sunset.usc.edu

Table 1. COTS advantages and disadvantages.

Advantages	Disadvantages
Immediately available; earlier payback	Licensing, intellectual property procurement delays
Avoids expensive development	Up-front license fees
Avoids expensive maintenance	Recurring maintenance fees
Predictable, confirmable license fees and performance	Reliability often unknown or inadequate; scale difficult to change
Rich functionality	Too-rich functionality compromises usability, performance.
Broadly used, mature technologies	Constraints on functionality, efficiency
Frequent upgrades often anticipate organization's needs	No control over upgrades and maintenance
Dedicated support organization	Dependence on vendor
Hardware/software independence	Integration not always trivial; incompatibilities among vendors
Tracks technology trends	Synchronizing multiple-vendor upgrades

Use risk-driven process models

Given the vagaries of requirements in COTS-based software development, developers should adopt or modify more dynamic, risk-driven process models, such as risk-driven spiral-type process models. Assess risks via prototyping, benchmarking, reference checking, and related techniques. Focus each spiral cycle on resolving the most critical risks. The Raytheon "Pathfinder" approach—using top people to resolve top risk items in advance—is a particularly effective way to address these and other risks.

You should also perform the equivalent of a "receiving inspection" upon initial COTS receipt. This practice ensures that the COTS product really does what it is expected to do. Keep requirements negotiable until the system's architecture and COTS choices stabilize. This prevents you from promising features and capabilities that the system cannot support easily—or at all. Finally, involve all key stakeholders in critical COTS decisions. These stakeholders can include users, customers, developers, testers, maintainers, operators, or others as appropriate.

INTEROPERABILITY

Most COTS products are not designed to interoperate with each other. The Garlan experience[2] provides a good case study and explanation for why interoperability problems can cause COTS integration cost and schedule overruns.

Roadblocks to interoperation

If you commit prematurely to incompatible combinations of COTS products, you lessen the chance they will operate with your own software. This situation can happen in many ways: through haste, desire to show progress, politics, short-term emphasis on rapid application development, or an uncritical enthusiasm for features or performance.

Trying to integrate too many incompatible COTS products can also cause problems. As Garlan[2] shows, four such products can be too many. In general, trying to integrate more than a half-dozen COTS products from different sources should place this item on your high-risk assessment list. Nor should you defer COTS integration till the end of the development cycle. Doing so puts your most uncontrollable problem on your critical path as you approach delivery.

Finally, avoid committing to a tightly coupled subset of COTS products with closed, proprietary interfaces. Such products restrict your downstream options; once you're committed, it's hard to back out.

Ensure smooth interconnections

To make sure your in-house software works with the COTS products you purchase, use the Life Cycle Architecture milestone[3] as an anchor point for your development process. In particular, include demonstrations of COTS interoperability and scalability as risks to be resolved and documented in the feasibility rationale. Use the AT&T/Lucent Architecture Review Board (ARB) best commercial practice[4] at the Life Cycle Architecture milestone. Over a 10-year period, AT&T documented at least a 10 percent savings from using ARBs.

Finally, strive to achieve open architectures and COTS substitutability. In the extremely fast-moving software field, the ability to adapt rapidly to new best-of-breed COTS products is critical.

PRODUCT EVOLUTION

You have no control over a COTS product's evolution, which responds only to the overall marketplace. Upgrades are frequently not upwardly compatible; old releases become obsolete and unsupported by the vendor. If COTS architectural mismatch doesn't get you initially, COTS architectural drift can easily get you later. Our Affiliates' experience indicates that complex COTS-intensive systems often have higher software maintenance costs than do traditional systems—for example, when the application is relatively stable and the COTS products are relatively volatile. Good practices, such as batching COTS upgrades in synchronized releases, can lower these costs.

Evolutionary dead ends

"Snapshot" requirements specifications and corresponding point-solution architectures make it too difficult to evolve your software. These are bad practices for traditional systems; with uncontrollable COTS evolution, the maintenance headaches become even worse. If you understaff maintenance personnel or undertrain them in COTS adaptation, you invite long-term problems. Integrating COTS products with your own software, then maintaining the combined system, is a challenging task under the best circumstances; without proper training, that challenge becomes significantly greater.

Tightly coupled, independently evolving COTS products cause an increase in maintenance overhead. Using just two such products will make your system's maintenance difficult; using more than two will make the problem much worse. Finally, it's wrong to assume that uncon-

trollable COTS product evolution is just a maintenance problem: It can attack your development schedules and budgets as well.

Nurture beneficial mutations

You can offset the costs and risks associated with long-term COTS product evolution by sticking with dominant open commercial standards. These standards make COTS product evolution and substitutability more manageable.

For COTS product selection criteria, use likely future system and product line needs (evolution requirements) as well as current needs. These evolution criteria can involve portability, scalability, distributed processing, user interface media, and various kinds of functionality growth. Use flexible architectures that facilitate adaptation to change. Strong choices include software bus, encapsulation, layering, and message- and event-based architectures.

Carefully evaluate COTS vendors' track records with respect to product-evolution predictability. Widely varying feature sets, too-frequent updates, and dramatic shifts in product capabilities can cause problems in the long term.

Finally, establish a proactive system-release strategy, synchronizing COTS upgrades with system releases. Planning the ongoing integration of evolving COTS products with your own internally developed software helps ensure that both continue to function harmoniously.

VENDOR BEHAVIOR

COTS vendor behavior varies widely with respect to support, cooperation, and predictability. Given the three major sources of COTS integration difficulty we've already cited, an accurate assessment of a COTS vendor's ability and willingness to help with these areas is tremendously important. The workshop identified a few assessment heuristics that proved helpful, such as that the value of a COTS vendor's support follows a convex curve with respect to the vendor's size and maturity. Specifically, according to our Affiliates' experience, mid-sized companies occupy the highest arc of the curve and usually provide the best support because, as a recent radio commercial generalizes, "Small companies are too small to help; big companies are too big to care."

Perishable promises

Beware of uncritically accepting vendors' statements about their COTS products' capabilities and support. Shifting markets, mergers and buyouts, or unforeseen technological developments can convert a vendor's best intentions into broken promises. A lack of fallbacks or contingency plans can also derail your project. Any project that doesn't allow for such contingencies as product substitution or escrow of a failed vendor's product is one that courts disaster.

Foster realistic expectations

To ensure that you establish the best vendor relationships possible, you must perform extensive evaluation and reference-checking of a COTS vendor's advertised capabilities and support track record. Web searches, interviews with the vendor's other clients, and industry publications can all help determine a given vendor's credibility.

Because COTS products are likely to remain an essential component of your software for a long time, it can help to establish strategic partnerships or other incentives for COTS product vendors to provide continuing support. These incentives can include financial assistance, early experimentation with a new COTS vendor's capabilities, and sponsored COTS product extensions or technology upgrades. Further protect yourself by negotiating and documenting critical vendor support agreements. You can establish a "no surprises" relationship with vendors by determining, in advance, exactly what's expected of both parties.

Sources and Resources

The best source I know for COTS integration information is the CMU Software Engineering Institute's Web page on its COTS-Based Systems (CBS) initiative at http://www.sci.cmu.edu/cbs/cbs_description.html. The USC-CSE Web page for the Constructive COTS Integration (COCOTS) cost estimation model has pointers to numerous COTS integration information sources. It's at http://sunset.usc.edu/COCOTS/cocots.html.

The following five major recent books on software reuse offer valuable perspectives on integrating reusable components in general, of which COTS is a special case.

I. Jacobson, M. Griss, and P. Jonsson, *Software Reuse*, Addison Wesley Longman, Reading, Mass., 1997.

W. Lim, *Managing Software Reuse*, Prentice Hall, Upper Saddle River, N.J., 1998.

J. Poulin, *Measuring Software Reuse*, Addison Wesley Longman, Reading, Mass., 1997.

D. Reifer, *Practical Software Reuse*, John Wiley & Sons, New York, 1997.

W. Tracz, *Confessions of a Used Program Salesman: Institutionalizing Software Reuse*, Addison Wesley Longman, Reading, Mass., 1995.

The Lim and Reifer books offer the most COTS-specific insights.

The upcoming 1999 International Conference on Software Engineering (ICSE '99) in Los Angeles, May 16-22, 1999, includes a COTS Integration industry experience session that features Lockheed Martin's Dorothy McKinney and Texas Instruments' Marie Silverthorn, with the SEI's Tricia Oberndorf as discussant. Among ICSE '99's architecture experience case studies is an insightful paper by David Barstow on how his sports information software's architecture evolved in response to the rapid evolution of Netscape and related COTS products. The associated Symposium on Software Reuse, May 21-23, will have a specific reuse focus that includes COTS integration. See the ICSE '99 Web site at http://sunset.usc.edu/icse99/index.html.

By accelerating the speed with which new software products can reach their users, while simultaneously offsetting up-front development costs, COTS products have become an increasingly attractive option for budget-conscious software developers. But these advantages bear a sometimes hidden price tag. In the short term, differences between the underlying design of an organization's internally developed software and that of the COTS products it chooses to integrate with that software can cause unforeseen problems. In the long term, fluctuating COTS developer support and the unpredictable evolution of the COTS product itself can hobble any organization that incorporates that product into its software, crippling the hybrid system or even rendering it completely unusable.

To guard against such grim situations, approach any COTS integration project clear-eyed and wary. By following the recommendations we've given in this article, you can go a long way toward maximizing COTS' advantages and protecting yourself against its more expensive dangers. ❖

Barry Boehm developed the Constructive Cost Model (Cocomo), the software process Spiral Model, and the Theory W (win-win) approach to software management and requirements determination.

Chris Abts is developing the constructive COTS Integration (COCOTS) cost model. Contact him at cabst@sunset.usc.edu.

References

1. National Research Council, *Ada and Beyond: Software Policies for the Department of Defense*, National Academy Press, Washington, D.C., 1997.
2. D. Garlan, R. Allen, and J. Ockerbloom, "Architectural Mismatch: Why Reuse is So Hard," *IEEE Software*, Nov. 1995, pp. 17-26.
3. B. Boehm, "Anchoring the Software Process," *IEEE Software*, July 1996, pp. 73-82.
4. J. Marenzano, "System Architecture Review Findings," D. Garlan (ed.), *Proc. ICSE 17 Architecture Workshop*, Carnegie Mellon Univ., Pittsburgh, 1995.