COTS-Based Systems Top 10 List

Victor R. Basili, University of Maryland
Barry Boehm, University of Southern California

In the January 2001 issue of *Computer* (pp. 135-137), we published the Software Defect Reduction Top 10 List—one of two foci pursued by the National Science Foundation-sponsored Center for Empirically Based Software Engineering (CeBASE).

COTS-based systems (CBS) provide the other CeBASE focus. For our intent, COTS software has the following characteristics: The buyer has no access to the source code; the vendor controls its development; and it has a nontrivial installed base (that is, more than one customer; more than a few copies).

Criteria for making the list are that each empirical result has

- significant current and future impact on software dependability, timeliness, and cost;
- diagnostic value with respect to cost-effective best practices; and
- reasonable generality across applications domains, market sectors, and product sizes.

These are the same criteria we used for our defect-reduction list, but they are harder to evaluate for CBS because it is a less mature area.

CBS's roller-coaster ride along Gartner Group's visibility-maturity curve (http://gartner11.gartnerweb.com/public/static/hotc/hc00094769.html) reveals its relative immaturity as it progresses through a peak of inflated expectations (with many overenthusiastic organizational mandates to switch to CBS), a trough of disillusionment, a slope of enlightenment, and a plateau of productivity.

We present the CBS Top 10 List as hypotheses, rather than results, that also serve as software challenges for enhancing our empirical understanding of CBS.

This list offers hypotheses for examining CBS project decisions in light of empirical data samples.

HYPOTHESIS ONE

More than 99 percent of all executing computer instructions come from COTS products. Each instruction passed a market test for value.

- *Source.* The more than 99 percent figure derives from analyzing Department of Defense data (B. Boehm, "Managing Software Productivity and Reuse," *Computer*, Sept. 1999, pp. 111-113).
- *Implications.* Economic necessity drives extensive COTS use. Nobody can afford to write a general-purpose operating system or database management system. Every project should consider the CBS option, but carefully weigh CBS benefits, costs, and risks against other options. "Market test" means that someone willingly pays to install the COTS component, not that every instruction is used or proves valuable.

HYPOTHESIS TWO

More than half the features in large COTS software products go unused.

- *Source.* According to Torii laboratory data reported at the 2000 International Software Engineering Research Network Workshop, individuals working alone used 12 to 16 percent of Microsoft Word and PowerPoint measurement features, whereas a 10-person group used 26 to 29 percent of these features.
- *Implications.* Adding features is an economic necessity for vendors, but it introduces complexity for COTS adopters. This added complexity can require expanded computer resources, such as speed and memory, to provide functionality that is not used or needed. These unused features are, however, available gratis, when you need them.

HYPOTHESIS THREE

The average COTS software product undergoes a new release every eight to nine months, with active vendor support for only its latest three releases.

- *Source.* Ron Kohl's surveys, presented at Ground System Architectures Workshops 99, 00, and 01, show average release frequencies of 6.3, 8.5, and 8.75 months, respectively. The number of supported releases is anecdotal. However, wide variations occur around these averages.
- *Implications.* Using COTS products will solve many infrastructure change adaptation problems. Vendors must evolve their products to sustain the market's competition, depending on their perceptions of fluctuating market demands. If your product's development is inconsistent with the market's evolution, your applications will have nontrivial adaptive maintenance costs, even during production, and can introduce new risk.

Table 1. Effort required to complete various COTS-directed development activities.

Activity	Average effort (%)	± Standard deviation (%)
Glue code	37	± 36
Tailoring	26	± 30
Assessment	24	± 20
Volatility	13	± 11

HYPOTHESIS FOUR

CBS development and postdeployment efforts can scale as high as the square of the number of independently developed COTS products targeted for integration.

- *Source.* Integrating n COTS products involves potentially $n(n-1)/2$ interfaces. The theoretical justification for this relationship stems from the architectural incompatibilities that pose difficulties in integrating any two COTS products. Although most empirical evidence is anecdotal so far, projects required to integrate various COTS products—such as NASA's EOSDIS project—have experienced effort expenditures not inconsistent with this relationship.
- *Implications.* Beware of proliferating excessive COTS products in your system—four can be too many (D. Garlan et al., "Architectural Mismatch: Why Reuse Is So Hard," *Software*, Nov. 1995, pp. 17-26). Making the scaling law approximately linear involves using sound interface standards, modular domain architectures, wrappers around COTS products, and well-planned multi-COTS-product refresh cycles. Check for architectural incompatibilities and monitor the products' technical efficacy and business health.

HYPOTHESIS FIVE

CBS postdeployment costs exceed CBS development costs.

- *Source.* We are only beginning to accumulate sufficient empirical data to support this hypothesis. Consider the adaptive maintenance costs for legacy systems: These costs relate to the need to adapt your applications to changes in the COTS platform. Considerable anecdotal evidence supports the hypothesis, except for short-lifetime systems.
- *Implications.* Although similar to the life-cycle cost distribution for non-CBS software, this relationship comes as a surprise for CBS developers and policymakers who expect to pay just acquisition costs. There's an added hazard: A cost-constrained developer on a 27-month project might be tempted to present a maintainer from a different organization with about-to-become-unsupported COTS products. Not adapting to three release cycles during development saves development costs but causes the system to deplete COTS vendor support by month 27.

HYPOTHESIS SIX

Although glue-code development usually accounts for less than half the total CBS software development effort, the effort per line of glue code averages about three times the effort per line of developed-applications code.

- *Source.* The mean and standard deviations of major CBS effort sources—excluding functional development effort—for COTS-directed software development activities consume the resources shown in Table 1. The large standard deviations indicate that no one-size-fits-all distribution of these CBS cost sources exists. The median CBS productivity rate for the 20-project COCOTS database is about 100 glue-code source lines of code per person-month of CBS effort. The median applications productivity rate for the 161-project Cocomo II database is about 270 SLOC per person-month. The productivity rate for developing glue code depends on at least the 13 cost drivers in the COCOTS glue-code model including COTS supplier and integrator capabilities, COTS product maturity and interface complexity, and degree of mismatch between desired system qualities and the qualities the COTS products provide. Similarly, applications-code productivity varies as a function of several often different cost drivers.
- *Implications.* First, don't assume that the bulk of the CBS effort involves glue code. COTS assessment investments can reduce glue-code and overall CBS costs and risks. Risk considerations—benchmarking, prototyping, trial use, and reference checking—can determine adequate assessment. Second, the factor of 3 for developing glue code over application code is a general range-checking factor. Although it provides important input for cost estimation, we don't apply this factor to a project's application-productivity rate to determine its CBS productivity rate.

HYPOTHESIS SEVEN

Nondevelopment costs, such as licensing fees, are significant, and projects must plan for and optimize them.

- *Source.* A number of CBS nondevelopment effort costs need to be factored into the cost-benefit-risk analyses. The SEI identified three significant CBS activity areas—vendor relationships, license administration, and training and cultural transition. Technical papers report the expenditure and savings of millions of dollars for COTS licensing. For example, one opportunity to optimize costs is enterprise-wide COTS licensing. Northrop saved more than $1 million annually by pooling its tool licenses and using basic ordering agreements.
- *Implications.* Organizations should modify their technical and management processes to address CBS issues and opportunities and plan for nondevelopment activities. For example, most object-oriented analysis and design methods don't address COTS.

HYPOTHESIS EIGHT

CBS assessment and tailoring efforts vary significantly by COTS product classes—

operating system, database management system, user interface, device driver, and so forth.

- *Source.* Table 2, drawn from the COCOTS database, reveals the median amount of tailoring efforts and number of tailored-product data points for various COTS product classes in the COCOTS database. Each class has at least a factor of 4, ranging from maximum to minimum, indicating the wide distribution of CBS effort.
- *Implications.* The small samples and large variations in the COCOTS data do not offer precise boundaries between the assessment, tailoring, and glue-code development activities. Thus, the values are only appropriate for range-checking and as starting points for a more robust basis of estimation.

Table 2. Tailoring effort for various COTS product classes.

Product class	Person-months	Products
Database management systems	38	11
GUIs	14	6
Networking	13	8
Device drivers	3	11
Operating systems	2	37

HYPOTHESIS NINE

Personnel capability and experience remain the dominant factors influencing CBS-development productivity.

- *Source.* Both expert Delphi consensus on COCOTS glue-code cost drivers and anecdotal experience support this assertion. The largest COCOTS Delphi productivity ranges are for COTS product and integration experience, personnel capability, and personnel continuity, each with about a factor of 2 productivity range. A strong correlation (R^2 = 0.89) exists between COTS glue-code productivity estimates and actual values using these factors in the medium range—2,000 to 100,000 SLOC of glue code—in the COCOTS database.
- *Implications.* Don't tackle difficult COTS integration challenges without skilled support. If you contract for software acquisition, don't saddle your organization with COTS products that it has neither the capability nor the experience to maintain.

HYPOTHESIS TEN

CBS is currently a high-risk activity, with effort and schedule overruns exceeding non-CBS software overruns. Yet many systems have used COTS successfully for cost reduction and early delivery.

- *Source.* The Standish Group's CHAOS report (http://www.scs.carleton.ca/~beau/PM/Standish-Report.html) cited median applications software cost overruns of about 50 percent, with 4 percent of the projects overrunning beyond 400 percent. The report documented median application software schedule overruns of about 100 percent, with 1 percent of the projects having schedule overruns beyond 400 percent. CBS anecdotal overrun data seems similar for COTS applications serving as the applications platform, but with more overruns exceeding 400 percent for systems in which COTS products comprise the applications software. The lack of visibility in COTS products, vendors' temptations to overpromise on vaporware, and the difficulty of estimating glue-code size in source code lines combine with overoptimism and desire to please to produce unrealistic CBS cost and schedule commitments. On the other hand, the US Air Force Scientific Advisory Board provided examples of 38 large software-intensive systems, such as AWACS, that managed the risk associated with COTS (*SAB-TR-99-03*, Apr. 2000). Many such systems have integrated COTS packages to provide the required functionality.
- *Implications.* Validate CBS cost and schedule estimates frequently, using expert judgment, task breakdowns, emerging estimation models, and analogies with previous CBS projects. Treat estimation accuracy as another risk to be mitigated by prototyping, benchmarking, reference checking, and so forth. Substantiate estimates and improve future predictions by gathering data. CBS projects are excellent for applying Experience Factory approaches (V. Basili et al., "The Experience Factory," *Encyclopedia of Software Engineering*, John Wiley & Sons, New York, 1994) to empirically based software-project management and organizational improvement.

Empirical knowledge of CBS is at an early stage of maturity compared with software-defect reduction. The initial quantitative values or ranges in the CBS Top 10 List represent preliminary hypotheses for validity-checking CBS project decisions against small empirical data samples with explained variability sources. Although not ideal, it is better than having one or zero data points with no explained variability sources.

This list provides a starting point from which the software community can develop solid empirical methods for coping with COTS-based data. It complements the more general SEI CBS Web site at http://www.sei.cmu.edu/cbs. We welcome your comments and contributions to the CeBASE CBS Web site at http://www.cebase.org/cots. ✷

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