

This document is excerpted from the proceedings
of the USC-CSE Focused Workshop #4: COCOMO 2.0
held May 16-18, 1995 at The University of Southern California

Summary of the COTS (Commercial Off-The-Shelf) Software Discussion Group

Leader : Ellis Horowitz (USC/CSCI Dept. Chair)

Participants: Marvin Carr (SEI)
Farzad Davaraya (Hughes, Applied Information Systems)
Jairus Hihn (JPL)
Walt Johnson (Loral Federal Systems)
Milton Lavin (JPL)
Gary Thomas (E-Systems)
Ronald Ulrich (Hughes, Information Technology Center)

Scribe: Christopher Abts (USC)

INTRODUCTION

The discussion focused on five areas related to COTS issues. Initial efforts were devoted to reaching a consensus on what is actually meant by the term "COTS" software. This was followed by an exploration of scenarios in which COTS software might be employed. The group then examined how existing COTS products might be evaluated and selected for use. Discussion of issues related to the testing of COTS software followed next, with the final efforts of the group focusing on the challenge of reflecting COTS software in overall system development cost estimations.

DEFINITIONS

What it is:

It was agreed that a COTS product must be a stand-alone software product that has been commercially sold to at least one customer. However, there are two important caveats related to this definition:

1) the COTS software may not necessarily be commercially available at the time the proposal for the current software system under consideration is prepared, but it must be commercially available at the time of system delivery.

2) the organization developing the new software system may in fact be the first purchaser of the identified COTS product.

A distinction was also made between software that is considered "black box" COTS and "white box" COTS. Black box COTS software is incorporated without any modifications to its internal structure. White box COTS software may be incorporated after minor--sometimes even significant--modification.

Black box COTS software may be procured in three different forms:

1) as executable modules only, in which the COTS product is merely invoked by the larger system and then performs its usual functions.

2) as a product which provides a library and an Application Programmers' Interface or API to the library.

It is expected that developers will make use of the API. Examples are Sybase and Oracle.

3) as source code which is merely bundled in with the developing product.

By its nature white box COTS software requires being procured in a form in which source code is provided by the COTS software vendor.

What it is not:

Any software built in-house by the responsible developing organization is not considered COTS software. This includes software built specifically for the current system under consideration, complete stand-alone software built for other systems, parts of other systems, and in-house standardized component libraries (the latter three items would be considered reusable software). In addition, software built by outside subcontractors specifically for the current system, and purchased outside standardized component libraries which violate the requirement that COTS software be capable of functioning as stand-alone systems is also not considered COTS software (again, the latter item would be considered reusable software).

Related terms:

Black Box COTS - modifications to code internal to COTS product not allowed.

COTS - commercial off-the-shelf.

GFE - government furnished equipment (see GOTS).

GOTS - government off-the-shelf (see GFE).

NDI - not developed in-house.

OTS - off-the-shelf (encompasses COTS and GOTS).

Reuse Software - any reusable software elements built in-house, or software elements purchased from outside that cannot function as self-contained stand-alone systems.

White Box COTS - some modifications to code internal to COTS product permitted.

USAGE SCENARIOS

Three basic scenarios for the use of COTS software were identified. These included COTS products as tools, as infrastructure, and as application components.

As tools:

These would include COTS software products used to build other software products. An example would be a compiler procured for inclusion within a product designed to be a complete software development environment.

As infrastructure:

This would include COTS software incorporated within such products as operating systems, graphical user interface software, and database management systems.

As application components:

These would include COTS products incorporated within a larger system to provide the user with specific application capabilities. An example of a black box COTS application component might be Microsoft Office, desirable for its suite of business productivity aids. An example of a white box COTS application component would be the RSA data encryption system, which is sold as source code designed to be integrated into the source code of the software system for which the customer requires data security.

These usage categories are not absolute. COTS software sometimes provides functionality which

overlaps categories. An example would be SNAP, a tool used at the Jet Propulsion Laboratory to build software, but which also provides run-time systems capability.

Finally, it was noted that when incorporating COTS software, black box components can turn gray or white. It was suggested that within commercial systems, black box components rarely go white due to budgetary constraints. The end-user will likely just live with any reduced functionality which results from being unable to modify the COTS component. Within government and defense systems, however, black box components frequently will turn white, often with the system requirements leveraged against the capabilities potentially offered by the COTS component.

SELECTION

Selection of COTS software for inclusion within a larger system must be based upon three familiar criteria:

- 1) Functional requirements (what capability does the COTS element provide?)
- 2) Performance requirements (can the COTS element provide that capability within given sizing and timing constraints?)
- 3) Nonfunctional requirements (what cost, training, installation, maintenance, and reliability requirements are associated with that COTS element?)

In addressing these criteria, it was agreed that the considerations represented by the cost drivers found within Loral's COTSCOST model (see p??/appendix??) provide a sound basis on which to begin an evaluation of COTS software.

The Loral drivers:

- Vendor Maturity.
- End User & Administrative Training.
- Configurability/Customization Administrative Effort.
- Installation Ease.
- Portability.
- Ease to Upgrade.
- Previous Product Experience.
- Product Support Services.
- Ease of Use for Administrator.
- Product Support Quality.
- Expected Release Frequency.
- Product Maturity.
- Application or System type of COTS package.
- User, Administrator, & Install Documentation.
- Ease of Use for End User.

(Note that in evaluating COTS software, in the COTSCOST model these drivers apply only to a given COTS product itself, not to the larger system under consideration nor to the interface software which must be written to fold the COTS element into that larger system.)

Other important considerations include the ability of the COTS element to meet contracted system requirements, interoperability of the COTS element with other elements of the overall system, experience of the organization planning to incorporate the COTS product, and experience of the end-user with the COTS product.

Another important consideration is whether the overall design of the system will be negatively impacted by the COTS product.

This is related to the issue of interoperability, and addresses whether the functioning of the COTS product could potentially

interfere with the rest of the system. How are the system builders assured that the COTS product will not affect/modify/destroy/make impossible the functioning of some other aspect or element of their system?

Finally, it was observed that selection and use of COTS products implies a prototyping software development lifecycle.

TESTING

It was agreed the use of COTS software does not reduce testing. To the contrary, testing is now a major activity. Issues to be addressed by testing include

- 1) Testing the functionality of the COTS product (in this instance care must be taken not to confuse characterization of the purported functionality offered by the COTS software with verification of that functionality.)
- 2) Testing the performance of the COTS product.
- 3) Testing the integrated system which employs the COTS product, by both functionality and performance.

In current practice there is a range in the formality of the standards used to test COTS software. These include the familiar government standard 2167A (which demands the most formality and requires the software be tested against "shall/will" type specifications), government standard 7935A (which requires software be tested against "loaded" scenarios, taking a stress test approach and treating software almost as a black box), and finally the "commercial" standard (which has been characterized as requiring software be tested "catch as catch can").

The formality of the testing depends upon the use intended for the COTS product. Mission critical/safety critical applications demand the most formality, with the accompanying increase in testing costs.

It was suggested that establishment by upper management of formal policies within an organization regarding the testing of COTS software can assist in ensuring the consistency and dispersal of the results of such testing throughout an organization. This reduces duplication of effort and affords a common understanding of the meaning of COTS software test results across internal organizational boundaries. In particular, it is recommended that a repository which contains information and test results with respect to given COTS products be maintained and made available to internal organizational entities as needed. Depending upon the needs of the organization, this repository could be as formal as a COTS product database, or as informal as a bulletin board on an internal usenet.

COST ESTIMATION

The group used the existing Loral COTSCOST model as a springboard for examining COTS software cost estimation considerations.

The Loral model:

1) Uses function point analysis to estimate the amount of interface code which will be required to integrate a COTS product into the larger system being considered.

2) Uses an augmented subset of the standard COCOMO cost drivers to estimate the effort required for the integration of that COTS product (augmented in that COTSCOST uses the COSTAR implementation of COCOMO, which uses non-standard COCOMO drivers such as security requirements).

3) Estimates administrative costs associated with the integration of the COTS product.

4) Performs a final evaluation of the costs related to the use of a given COTS product within the proposed larger system using a table of items which include the associated interface code sizing based upon function points and the estimated effort based upon the cost drivers.

5) Provides access to a combined list of the comparative data of all the COTS products evaluated which meet the needs of the larger system, linked by the system function being met by those COTS products.

The current approach to estimating COTS software costs using COCOMO would be as follows:

1) If COTS software is on the platform, use Platform Volatility/Experience cost drivers.

2) If COTS software is part of a tool suite, use Language/Tool Experience and Use Of Tools cost drivers.

3) If COTS software consists of application components, use interface component size determined by function points, and the reuse features of the COCOMO model to estimate COTS software assessment and assimilation effort, accounting for these risk factors:

COTS software immaturity, staff inexperience with respect to COTS software integration in general and with the given COTS software in particular, incompatibility with the larger system and with other COTS products used in the system, the computing platform, performance, licensing, tailoring, training, and lack of control over future evolution of the given COTS product by the vendor.

Other considerations included the appropriate complexity/granularity of a COTS software cost estimation model (e.g., COCOMO uses fifteen cost drivers, Loral's COTSCOST uses five drivers, while one software cost model at JPL requires only three drivers).

LIFECYCLE

Use of COTS software implies a prototyping software development lifecycle. The nature of this prototyping lifecycle can differ, however, depending upon whether the COTS component is considered white box or black box.

(insert figure ??? -- effort vs time using white box/black box COTS products in a software system development.)

Though conceptually not the same, white box COTS software and reusable software are similar in nature (they both allow modifications to existing code, though the former can operate as stand-alone

elements while the latter cannot). Therefore, they might be treated the same in the development lifecycle in order to maintain simplicity in any proposed cost model. A graph of effort over time using white box COTS software in a system development would then tend to follow a familiar unimodal model.

Black box COTS software, however, should be treated as a separate entity. The reason is that testing and qualification of black box COTS products might be carried out independently of any given system development, or at least before system requirements are defined. This might often be the case if an organization has in fact established a COTS product knowledge repository. This implies that an appropriate representation of the required effort over time using black box COTS products in a software development could in fact be bimodal, capturing a smaller but none-the-less significant rise in activity appearing at the start of the development cycle.

A proposed lifecycle for the handling of black box COTS software based upon the experience of Loral with its COTSCOST model is as follows:

- 1) Qualify COTS products.
- 2) Perform system requirements.
- 3) Administer COTS software acquisition.
- 4) Prototype the system including COTS software.
- 5) Fully integrate the COTS software and the interface code.
- 6) Test completed prototype.

The black box COTS products are qualified ahead of time by a group specifically tasked to study COTS products. Again, however, the order of these activities is not rigid. Steps 1 and 2 are sometimes reversed.

For white box COTS software steps 1 and 2 are always reversed.

Finally, it was agreed that the current COCOMO model captures the reusable software lifecycle--and therefore the white box COTS software lifecycle--adequately. It does not, however, adequately model the black box software lifecycle when black box COTS product integration follows the bimodal path.

FUTURE DIRECTIONS

The following were suggested as questions to be explored or activities to be pursued in the future to further the understanding of COTS software modeling issues:

-- the establishment of a World Wide Web site by some organization (USC perhaps?) to serve as a clearing house/newsgroup for COTS products information.

-- a focused workshop sponsored by USC to address specifically COTS software.

-- the identification of what data should be collected from the affiliates to validate the new suggested COTS software lifecycle, particularly the black box approach.

--determination of the appropriate granularity of a COTS software cost model; of the appropriate cost drivers needed to support that granularity; and of the linearity/nonlinearity of

those cost drivers.

CONCLUSION

When considering the use of COTS software in a system development, the main advantages offered by the COTS software is that 1) it is usually (though not necessarily) available very quickly, and 2) it is likely (though again, not necessarily) stable. The main disadvantage is the testing required, both in the qualification of the COTS product itself, and in the integration of the COTS product with the overall software system.