Issues and Methods for Assessing COTS Reliability, Maintainability, and Availability

Norman F. Schneidewind
Code IS/Ss
Naval Postgraduate School
Monterey, CA 93943, USA

Allen P. Nikora
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109-8099, USA

Introduction

Many vendors produce products that are not domain specific (e.g., network server) and have limited functionality (e.g., mobile phone). In contrast, many customers of COTS develop systems that are domain specific (e.g., target tracking system) and have great variability in functionality (e.g., corporate information system). This discussion takes the viewpoint of how the customer can ensure the quality of COTS components. In evaluating the benefits and costs of using COTS, we must consider the environment in which COTS will operate. Thus we must distinguish between using a non-mission critical application like a spreadsheet program to produce a budget and a mission critical application like military strategic and tactical operations. Whereas customers will tolerate an occasional bug in the former, zero tolerance is the rule in the latter. We emphasize the latter because this is the arena where there are major unresolved problems in the application of COTS. Furthermore, COTS components may be embedded in the larger customer system. We refer to these as embedded systems. These components must be reliable, maintainable, and available, and must interoperate with the larger system in order for the customer to benefit from the advertised advantages of lower development and maintenance costs. Interestingly, when the claims of COTS advantages are closely examined, one finds that to a great extent these COTS components consist of hardware and office products, not mission critical software [1].

Obviously, COTS components are different from custom components with respect to one or more of the following attributes: source, development paradigm, safety, reliability, maintainability, availability, security, and other attributes. However, the important question is whether they should be treated differently when deciding to deploy them for operational use; we suggest the answer is *no*. We use *reliability* as an example to justify our answer. In order to demonstrate its reliability, a COTS component must pass the same reliability evaluations as the custom components, otherwise the COTS components will be the weakest link in the chain of components and will be the determinant of software system reliability. The challenge is that there will be less information available for evaluating COTS components than for custom components but this does not mean we should despair and do nothing. Actually, there is a lot we can do even in the absence of documentation on COTS components because the customer will have information about how COTS components are to be used in the larger system. To illustrate our approach, we will consider the reliability, maintainability, and availability (RMA) of COTS components as used in larger systems.

Finally, COTS suppliers might consider increasing visibility into their products to assist customers in determining the components' fitness for use in a particular application. We offer ideas of information that would be useful to customers, and what vendors might do to provide it.

Reliability

There are some intriguing questions concerning how to evaluate the reliability of COTS components that we will attempt to answer. Among these are the following: How do we estimate the reliability of COTS when there is no data available from the vendor? How do we estimate the reliability of COTS when it is embedded in a larger system? How do we revise our reliability estimates once COTS has been upgraded? A fundamental problem arises in assessing the reliability of a software component: a software component will exhibit different reliability performance in different applications and environments. A COTS component may have a favorable reliability rating when operated in isolation but a poor one when integrated in a larger system. What is needed is the operational profile of COTS components as integrated into the larger system in order to provide some clues as to how to test COTS components. We will assume the worst case situation that documentation and source code are not available. Thus, inspection would not be feasible and we would have to rely exclusively on testing and reliability calculations derived from test data to assess reliability.

The operational profile identifies the criticality of components and their duration and frequency of use. Establishing the operational profile leads to a strategy of what to test, with what intensity, and for what duration. We must recognize that a COTS component must be tested with respect to *both* its operational profile and the operational profile of the larger system of which it is a part. The COTS component would be treated like a black box for testing purposes similar to a custom component being delivered by design to testing but without the documentation. Testing the COTS components according to these operational profiles will produce failure data that can be used for two purpose: 1) make an empirical reliability assessment of COTS components in the environment of the larger system and 2) provide data for estimating the parameters of a reliability model for predicting future reliability [2]. A comprehensive software reliability engineering process is described in [3]. As pointed out by Voas, black box and operational testing alone may be inadequate [4]. In addition, he advocates using fault injection to corrupt one component (e.g., COTS component) to see how well other components (e.g., the larger system) can tolerate the failed component. While this approach can identify problems in the software, it cannot fix them without documentation. Thus there must be a contract with the vendor that allows the customer to report problems to the vendor for their resolution. Unfortunately, from the customer's standpoint, vendors are unlikely to agree to such an arrangement unless the customer has significant leverage such as the Federal Government. In the case where documentation is available, it would be subjected to a formal inspection of its understandability and usability. If the documentation satisfies these criteria, it would be used as an aid to inspecting any source code that might be available.

Maintainability

In the case of maintainability, there are more intriguing issues. Suppose a problem occurs in an embedded COTS system. Is the problem in COTS or in the larger system? Suppose it is caused by an interaction of the two. The customer knows the problem has occurred, but does not know how to fix it if there is no documentation. The vendor, not being on site, does not know the problem has occurred. Even the vendor may not know how to fix the problem if the source of the problem is the larger system or an interaction between it and COTS components. In addition, suppose the customer needs to upgrade the larger system and this upgrade is incompatible with COTS components. Or, conversely, the vendor upgrades COTS components and they are no longer compatible with the larger system.

Lastly, suppose there are no incompatibilities, but the customer may be forced to install the latest COTS components upgrade in order to continue to receive support from the vendor. None of these situations can be resolved without either documentation that the customer can use to attempt to fix the problem, or a contract with the vendor of the type mentioned above. As in the case of reliability, when neither of these remedies is available, problems can only be identified but they cannot be fixed. Thus the software cannot be maintained. An additional factor that impacts both reliability and maintainability is that the vendor is unlikely to continue to support the software if the customer modifies it. Thus the situation degenerates to one in which the customer is totally dependent on vendor support to achieve reliability and maintainability objectives. This may be satisfactory for office product applications but it is unsatisfactory for mission critical applications.

Availability

High availability is crucial to the success of a mission critical system. What will be system availability using COTS? To attempt to answer this question, it is useful to consider hardware as a frame of reference. The ultimate COTS is hardware; it has interchangeable and replacement components. Maintenance costs are kept low and availability is kept high by replacing failed components with identical components. Unlike hardware, availability cannot be kept high by "replacing" the software. A failed component cannot be replaced because the replacement component would have the same fault as the failed component. Fault tolerant software is a possibility but it has had limited success.

We see that availability is a function of reliability and maintainability as related by the formula:

$$\text{Availability} = \text{MTTF}/(\text{MTTF}+\text{MTTR}) = 1/(1+(\text{MTTR}/\text{MTTF})),$$

where MTTF is mean time to failure and MTTR is mean time to repair. MTTF is related to reliability and MTTR is related to maintainability. For high availability, we want to drive *time to failure* to infinity and *repair time* to zero. However, we have seen from the discussion of reliability and maintainability that achieving these objectives is problematic. Thus to achieve high availability, either the COTS software must be of high intrinsic reliability – probably a naive assumption – or there must be in place a strong vendor maintenance program (this assumption may be equally naive).

Improved Visibility into COTS

Major drawbacks of including COTS in a software system are the lack of visibility into how the COTS components were developed and an incomplete understanding of the components' behavioral properties. Without this information, it is difficult to assess COTS components to determine their fitness for a particular application. As suggested by McDermid in [5], a partial solution might be for COTS vendors to identify a set of behavioral properties that should be satisfied by the software, and then certifying that those properties are satisfied. For instance, an operating system supplier might certify that a lower-priority task does not interrupt a higher priority task as long as the higher priority task holds the resources required to continue processing. COTS vendors might also include the specifications of those components as well as details of verification activities in which those specifications had been used to show that specific behavioral properties of the software were satisfied. For instance, an effort in progress at the Jet Propulsion Laboratory [6] involves developing libraries of

reusable specifications for spacecraft software components using the PVS specification language [7]. The developers of the libraries work cooperatively with anticipated customers to develop the specifications and identify those properties that the components should satisfy. As they develop the libraries, the component developers use the PVS theorem prover to show that the behavioral properties are satisfied by the specification. These proofs are intended to be distributed with the libraries. When customers modify the libraries, perhaps to customize them for a new mission, they will be able to use the accompanying proofs as a basis for showing that the modified specification exhibits the desired behavioral properties. Similarly, commercial vendors could work with existing and potential customers through user groups to discover those behavioral properties in which users are the most interested, and then work to certify that their components satisfy those properties.

Conclusion

The decision to employ COTS on mission critical systems should not be based on development cost alone. Rather, costs should be evaluated on a total life cycle basis and RMA should be evaluated in a system context (i.e., COTS components embedded in a larger system). COTS suppliers should also consider making available more detailed information regarding the behavior of their systems, and certifying that their components satisfy a specified set of behavioral properties.

References

[1] Clemins, Archie, "IT-21: The Path to Information Superiority." CHIPS Jul 1997, http://www.chips.navy.mil/chips/archives/97_jul/file.htm, p. 1.

[2] Norman F. Schneidewind, "Reliability Modeling for Safety Critical Software", IEEE Transactions on Reliability, Vol. 46, No.1, March 1997, pp.88-98.

[3] Recommended Practice for Software Reliability, R-013-1992, American National Standards Institute/American Institute of Aeronautics and Astronautics, 370 L'Enfant Promenade, SW, Washington, DC 20024, 1993.

[4] Jeffrey M. Voas, "Certifying Off-the-Shelf Software Components", IEEE Computer, Vol. 31, No. 6, June 1998, pp. 53-59.

[ISO97] ISO 9000-3, "Quality management and quality assurance standards – Part 3: Guidelines for the application of ISO 9001:1994 to the development, supply, installation, and maintenance of computer software", Second edition, 1997-12-15, International Standards Organization, 1997.

[5] Nancy Talbert, "The Cost of COTS", IEEE Computer, Vol. 31, No. 6, June 1998, pp. 46-52.

[6] "Reusable Libraries of Formal Specifications", NASA Formal Methods web site, http://eis.jpl.nasa.gov/quality/Formal_Methods/library.html

[7] "The PVS Specification and Verification System", SRI International Computer Science Laboratory, http://www.csl.sri.com/sri-csl-pvs.html.