

# Open Source Software: The Other Commercial Software

**Scott A. Hissam**

Software Engineering Institute  
Carnegie Mellon University  
4500 5<sup>th</sup> Avenue  
Pittsburgh, PA, 16066 USA  
+1 412 268 6526  
shissam@sei.cmu.edu

**Charles B. Weinstock**

Software Engineering Institute  
Carnegie Mellon University  
4500 5<sup>th</sup> Avenue  
Pittsburgh, PA, 16066 USA  
+1 412 268 7719  
weinstock@sei.cmu.edu

## ABSTRACT

Open source software is a source of components from which systems can be built. Components can also be acquired in the marketplace. There are a myriad of issues that affect software that incorporates commercial components. These issues (and others) also affect open source software. In this paper we examine issues that an organization using open source software is likely to face in light of lessons learned from the world of commercial components

## Keywords

Open source software, commercial off-the-shelf software, COTS software

## 1 INTRODUCTION

Open Source Software (OSS) provides organizations a new source for viable software components. Before OSS, there were basically three choices for software components. The component could be built, a similar component could be reused, or it could be purchased.

If the component was to be built, it could be developed in-house or contracted out. Source code was available for this software.

If a suitable component could be located and adapted to the organizations' needs reuse was possible. Typically software to be reused would come in binary and source code form.

Components purchased commercially are typically not delivered with source code.

## The World After OSS

With the advent of OSS the community has an additional source of components which is actually a combination of

all three of the above choices. OSS is very much like reusable components in that they both are developed by others and often come in binary and source code form. But like reusable components, it can be challenging to understand what the component does [1].

Because it comes with the source, OSS is similar to custom-built software. However, it lacks the design, architectural and behavioral knowledge inherent to custom-built software. This is also a problem with software purchased commercially. This lack of knowledge allows us to draw a strong analogy between OSS and COTS software in spite of source being available for the former and not for the latter.

The Software Engineering Institute has been studying COTS-based systems for a number of years and has learned some important lessons about them. Many of these lessons apply directly to OSS.

Organizations adopting an OSS component have access to the source, but are not required to do anything with it. If they choose not to look at the source they are treating it as a black box. Otherwise they are treating it as a white box. We discuss both of these perspectives in the following sections.

## 2 OSS AS A BLACK BOX

Treating OSS as a black box is essentially treating it as a COTS component; the same benefits and problems will apply. For instance an organization adopting COTS products should know something about the vendor (e.g., its stability, its responsiveness to problems, etc.), and an organization adopting OSS should know something about its community.

If the community is large and active the organization can expect that the software will be frequently updated, that there will be reasonable quality assurance, that problems are likely to be fixed, and that there will be people to turn to for help. If the community is small and stagnant it is less likely that the software will evolve, that it will be well tested, or that there will be available support.

Organizations that adopt COTS solutions are often too small to have much influence over the direction in which

the vendor evolves the product [2]. Black box OSS is probably worse in this regard. A COTS component will change due to market pressure, time to market considerations, the need for upgrade revenue, etc. OSS components can change for similar market reasons, but can also change for political or social reasons (factions within the community), or because someone has a good idea—not necessarily in a direction suitable to the organization.

Organizations that adopt COTS products can suffer from the vendor driven upgrade problem: the vendor dictates the rate of change in the component and the organization must either upgrade or find that the version they are using is no longer supported. This same problem exists with OSS. The software will change, and eventually the organization will be forced to upgrade or it will not be able to benefit from bug fixes and enhancements. The rate of change for an eagerly supported OSS component can be staggering.

Organizations that adopt COTS solutions often find that they either have to adapt to the business model assumed by the component or pay to have the component changed to fit their business model [3]. We have found that adapting the business model usually works out better than changing the component [4]. Once you change a component you “own the solution”. If your changes are not accepted by the vendor you’ll be faced with making your changes to all future versions of the software.

For black box OSS it may be easier for a change to make its way back into the standard distribution. However, the decision is still out of the organization’s control. If the community does not accept the change, the only recourse is to reincorporate the change into all future versions of the component.

Because of a lack of design and architectural specification, undocumented functionality, unknown pre- or post-conditions, deviations from supported protocols, and environmental differences, it is difficult to know how a COTS component is constructed without access to the source code. As a consequence it can be difficult to integrate the component. With OSS the source is available but consulting it means that the component is no longer being treated as a black box.

### 3 OSS AS A WHITE BOX

Because the source is available, it is possible to treat OSS as a white box. It therefore becomes possible to discover platform specific differences, uncover pre- and post-conditions, and expose hidden features and undocumented functionality. With this visibility comes the ability to change the components as necessary to integrate them into the system.

However, sometimes the source is the only documentation provided. Some consider this to be enough. Linus Torvalds, creator Linux, has said, “*Show me the source.*” Yet, if this were the case then there would be no need for UML, use

cases, sequence diagrams, and other sorts of design documentation. Gaining competency in the OSS component without these additional aids can be difficult.

An organization that treats OSS as a white box has a few key advantages over one that treats it as a black box. One is the ability to test the system knowing exactly what goes on inside the software. The other is the ability to fix bugs without waiting for the community to catch up. A seeming advantage is the ability to adapt the system to the organization’s needs. But, as we’ve already discussed, a change not accepted by the community means you own it and have given up many of the benefits of OSS.

### 4 SUMMARY

Just as for COTS components, OSS components require substantial skills to understand, evaluate, use and integrate. Open Source “improves” upon COTS products because access to the source lets an organization gain insight into the component and, if warranted, repair by making adaptation directly to the software source code.

In this paper we have attempted to show the similarities between OSS and COTS components. Neither are the silver bullets that commercial vendors and OSS pundits have made them out to be. While both offer the promise of substantial savings over in-house or custom-built development, there is still an engineering overhead to be paid.

### REFERENCES

1. Shaw, M., Truth vs. Knowledge: The Difference Between What a Component Does and What We Know It Does. *Proceedings of the 8<sup>th</sup> International Workshop on Software Specification and Design*, March 1996.
2. Hissam, S., Carney, D., Plakosh, D., SEI Monograph Series: DoD Security Needs and COTS-Based Systems. Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA., September 1998.
3. Oberndorf, P., Foreman, J., Lessons Learned from Adventures in COTS-Land. *Proceedings of the 11<sup>th</sup> Annual Software Technology Conference*, Utah State University, Salt Lake City, UT, May 1999.
4. Brownsword, L., Place, P., Lessons Learned Applying Commercial Off-the-Shelf Products: Manufacturing Resource Planning II Program. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. CMU/SEI-99-TN-015, June 2000.