On Software Components and Commercial ("COTS") Software

Kurt C. Wallnau
Software Engineering Institute
Pittsburgh, PA, 15206

Abstract: The software industry is struggling to understand the meaning and implications of component-based software. A number of different perspectives have emerged concerning the nature of "software components", but one perspective that is particularly strong is that software components will be (*or are already*) commercial software products. In this paper I argue that if this commercial perspective is valid, then the goal of replaceable, standard components will never be achieved because component providers will resist might and main the emergence of the commodity-style markets implied by component substitutability. However, this strong assertion must be reconciled with manifest evidence that a marketplace of commercial software "components" is emerging. This reconciliation takes the form of a reference model that describes and relates different market niches for software components and engineering skills in an emergent component-based software paradigm.

Motivation

In reviewing the literature for component-based software engineering (CBSE) a.k.a. component-based software (CBS) a.k.a. component-based development (CBD), I have been struck by the recurring theme of *component marketplace*. This is true of submissions to the ICSE'99 workshop on CBSE, at least eight of which assume without comment that components are purchased "commercial off-the-shelf" (COTS). The correlation of component concepts with marketplace concepts is also a fundamental precept of Szyperski's influential (if not seminal) book on software components [szyperski-98].

Unfortunately, there is an inherent mismatch between one frequently stated key objective of software components, the ability to easily replace components, and the interests of COTS software vendors. Substitutability implies that one vendor's product can be substituted for another, and this, in turn, implies a commodity market of software products. But commodity markets are dominated by price competition, which is not in the interest of software product vendors. Why? Because software production is just as difficult, expensive and risky for producers of software products as it is for their consumers. These and other software production factors are not consistent with the emergence of a commodity market for software components.

The problem is that component substitutability expresses the interests of consumers without taking consideration of the interests of producers. One frequently encounters claims such as how software components will provide investment protection for consumers, allowing a measure of isolation and insulation from a fast changing technology marketplace. Of course, component providers have an interest in protecting *their* investments, and these interests take the form of non-standard and innovative product features. Of course, non-standard and innovative features encourage the opposite of product substitutability--*vendor lock*, the ideal condition for protecting a vendor's investment in their product line.

Consumers appear all too willing to participate in this form of component market. If the reader doubts this, then perhaps a few thought experiments might be convincing. First, is it more reasonable to assume that product selection decisions are made on the basis of the *unique* capabilities provided by the selected product, or on the basis of features that a selected product provides that are also provided in equal measure by all competing products? Second, is it reasonable to assume that the software market is behaving *irrationally* in producing software products that are difficult to integrate or substitute? Is it reasonable to ascribe these difficulties to *technology* deficiencies? The author is comfortable that performance of these experiments will sustain the premise that the component marketplace does not--and will not--result in replaceable software components.

And yet the arguments by Szyperski and others that commercialization of software components is ongoing and inevitable are compelling, and the evidence of this emergence is manifest to any practicing software engineer. It is, therefore, self evident that a thorough understanding of the relationship of software component technology to the component marketplace is essential if we are to fully understand the meaning and implications of CBSE. To this end I propose a reference model for relating software components and CBSE engineering skills to various market niches. This model is intended to complement Jim Ning's excellent reference model that describes and relates various technical aspects of software components [ning-99].

Concept Diagram

The component/marketplace reference model is depicted in Figure 1. The iconography is as follows. Things that appear in *solid* boxes are components, in the sense that they are *binary units of independent production, acquisition and deployment* [syperski-98]. The hammer-in-circles are engineering skills. Both have *value* in the marketplace that is related to technical aspects of software components. There is some risk in combining skills with components in the same reference model. However, the overall theme of this position paper is that the marketplace has its own internal logic that actively resists the emergence of plug-replaceable components. This necessarily requires the intervention of an external agency, i.e., skilled engineers, to overcome obstacles to plug-replaceability introduced by component vendors. Thus, it is difficult to disentangle components from the engineering skills needed to integrate them.

The positioning of icons suggests relationships between these marketable components and skills; these relationships are made more explicit in the prose description that follows. The description is structured as a glossary of the acronyms used to label the icons. In this description it is assumed that the overall market context is enterprise information systems. This is not an arbitrary choice, but reflects another conjecture (not justified in this position paper) that enterprise-level computing is and will continue to drive software component technology, and that the application of CBSE to other domains is possible but will nevertheless represent a specialty market.

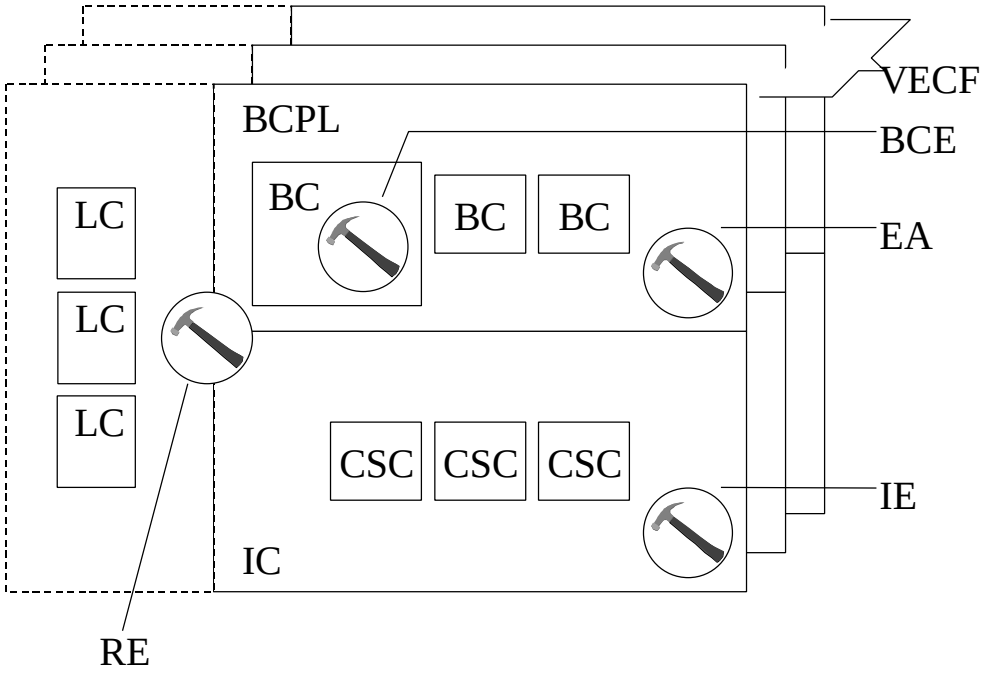

Figure 1: Components and Marketplace

Reference model glossary, organized to facilitate the discussion of related issues:

- **CSC--COTS Software Component**. Modern enterprise systems are composed from these types of components, which include relational database, transaction monitor and other middleware, web/intranet, security and a host of others. This partial extensional definition may not be theoretically satisfying, but it will satisfy engineers who are experienced in building enterprise systems. This is a robust and unstable market that will likely continue to behave in the future as it has in the past--momentary but only partial convergence on integration standards, followed by periods of technology disruption.
- **IC--Infrastructure Component**. The cost and complexity of integrating CSCs, and the high-level of sophistication required of application programmers who use integrated CSCs, has resulted in a new class of component--the off-the-shelf enterprise architecture and integration framework, generically referred to as IC. A class of *component-friendly* ICs is now emerging, specifically Microsoft's COM+/MTS and Sun et.al.'s Enterprise JavaBeans (EJB). This is an

emerging market whose success depends upon evidence that the *switchover* costs from conventional ICs to component-friendly ICs is justified by production efficiencies (e.g., faster time to market, decreased maintenance costs, easier incorporation of innovation). Note that this market will remain fragmented even if EJB succeeds, for EJB vendors are no more inclined to produce plug-replacable *servers* and *containers* than any other vendor of commercial software, and there is evidence to support this assertion [dorda-99]. In the remainder of this paper it is assumed that ICs are component friendly.

- **IE--Infrastructure Engineering**. The skills required to integrate "best of breed" CSCs is daunting. Not only is each CSC complex in its own right (as it must be if it will be "bought" rather than "made") and unstable (as it must if the vendor is to add innovative features to outpace competitors who invariably adopt the successful features of competing products), but their integration into *ensembles* of products is also quite complex [seacord-99]. The engineering skills and capacity for keeping abreast of changing technologies are therefore quite specialized, and very distinct from those skills often associated with the production of business applications.
- **BC--Business Component**. Assuming the emergence of a market for component-friendly ICs, *business* components will be developed. The definition of business component provided by Wojtek Kozaczynski [brown-98] as a "software implementation of an autonomous business process" is sufficient for the purposes of this paper. Actually, it is the author's opinion that there will *never* emerge a marketplace of individual business components because a) business processes tend to be unique, and, b) even if they could be made generic they would not become plug-replacable with other business components for precisely the same market reasons that inhibit plug-replacable CSCs. Thus, a market of individual BCs is highly unlikely.
- **BCPL--Business Components Product Line**. More plausible is the emergence of a *product-line* of business components--at least, there is at least one vendor who thinks this is a plausible scenario [theory-99]. Note that this discussion finesses two issues. First, there is the distinction between a component and configuration of components. Second, there is the distinction between a product line, and the specific product configured from the product line. These are important distinctions, but need not detain us as they do not invalidate the overall thrust of the observation, which is that the success of this market, if it is decoupled from *vertically-integrated components* (see next), depends on whether BCPLs can be deployed into a variety of ICs. This, as noted earlier, is dubious if the IC market continues to fracture, for example if EJB proves to be insufficient as a basis for any reasonable level of portability of Enterprise JavaBeans..
- **VECF--Vertically-Integrated Component Framework.** This corresponds to systems such as Baan, PeopleSoft, OracleFinancial and other well-known enterprise resource management (ERM) systems. These are "vertically integrated" in the sense that the frameworks package off-the-shelf business processes as well as all the software needed to adapt, maintain, deploy and execute these processes. The current generation of VECFs are decidedly not component-based, and if anything these large-scale systems are aggressive in their pursuit of "vendor lock." However, these vendors are, bit by bit, being forced to make their systems more adaptive to 3rd-party integration (and therefore component-level substitutability) in order to make their products more cost competetive in "mid-level" enterprises.
- **LC--Legacy Component**. Legacy components are existing business applications, whether they are unique to an enterprise (in which case there is no market), or whether they are COTS business applications. The marketplace for these components is not of interest to CBSE except

as they introduce *repair engineering* as a necessary and marketable skill. Such repairs are implied by the need for new component-based systems to interoperate with existing systems.

- **RE--Repair Engineering**. Assuming the emergence of a market of component-friendly ICs, a substantial problem remains about how to integrate LCs using component technology. This integration requires the removal of various mismatches between LCs, BCs and component-friendly ICs. There are two classes of mismatches, both of which are non-trivial to diagnose and repair. The first class is *semantic* mismatches at the business process level. Diagnosing and repairing these mismatches requires a thorough understanding of the application domain. The second class is *mechanical* mismatches between the LC and the IC. Diagnosing and repairing these mismatches requires a thorough understanding of operating system level primitives (processes, protocols, etc.) and the IC component model (component life cycles, state transitions and other component-level protocol issues). Engineers skilled in dealing with both kinds of repairs are a rare breed.
- **BCE--Business Component Engineering**. Desinging and implementing BCs involves new skills. In addition to the ability to map an understanding of business processes and concepts to software abstractions (a skill that is needed to build even current-generation enterprise applications), new skills include understanding and management of modular requirements, familiarity with component concepts and infastructures, fault diagnosis in distributed systems executing "black box" components, and the ability to track and accommodate fast changing and frequently non-robust products that underlie the IC.
- **EA--Enterprise Architectecting.** Enterprise architectures enbody large-scale corporate decisions regarding business objectives, and the information technology strategies needed to attain these objectives. The challenges of enterprise architecting have always been substantial, requiring a deft combination of business and technology savvy. A number of additional skills are required if architects are to accommodate the market imperatives implied by component technology, including designing for resiliance in the face of change, maintenance of technology competency, managing uncertainty and loss of control of production factors, and so forth [wallnau-99].

The above discussion is meant to be suggestive rather than exhaustive. It is intended to describe the large-scale implications of market imperatives on software component technology. It also indulges in a bit of prognostication.

Summary

If the premise is valid that the benefits of component technology are derived from a component marketplace, as is eloquently argued by Szyperski and tacitly assumed by others, then the premise that the benefits of software components are derived from their plug-replaceability is surely at risk. This literal dilemma suggests that software researchers need to better understand and relate market factors to the technical concepts underlying component-based software. To this end I have proposed a reference model--doubtless incomplete and flawed--as a starting point for building this understanding.

References

[brown-99] Alan Brown, Kurt Wallnau, "The Current State of CBSE," IEEE Software, September/October 1998, pp.37-46.

[dorda-99] Santiago Comella Dorda, John Robert, Robert Seacord, "Theory and Practice of Enterprise JavaBean™ Portability," SEI Technical Note, in preparation.

[ning-99] Jim Ning, "A Component Model Proposal," in proceedings of the 2nd Workshop on Component-Based Software Engineering, in conjunction with ICSE99 http://www.sei.cmu.edu/cbs/icse99/papers/index.html

[seacord-99] Robert Seacord, Kurt Wallnau, John Robert, Santiago Comella Dorda, Scott Hissam, "Custom vs. Off-The-Shelf Architecture," SEI Technical Note, in preparation, submitted to EDOC'99.

[szyperski-98] Clements Szyperski, Component Software Beyond Object Oriented Programming, Addison-Wesley, 1998.

[theory-99] Web site for TheTheoryCenter, www.theorycenter.com.

[wallnau-99] Kurt Wallnau, "COTS Software: Five Key Implications for the System Architect," in Software Tech News, DoD Data and Analysis Center for Software, http://www.dacs.dtic.mil/awareness/newsletters/technews2-3/toc.html