AN EXAMINATION OF THE CURRENT STATE OF CBSE: A REPORT ON THE ICSE WORKSHOP ON COMPONENT-BASED SOFTWARE ENGINEERING (CBSE)

HELD IN CONJUNCTION WITH THE 20TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE) KYOTO, JAPAN.

Alan W. Brown (alan_brown@sterling.com)
Sterling Software
Applications Development Division
Plano, Texas, 75023

Kurt C. Wallnau (kcw@sei.cmu.edu)
Software Engineering Institute
Carnegie Mellon University
Pittsburgh PA, 15213

INTRODUCTION

On April 25-26 representatives of industry and academia met in Kyoto, Japan, to participate in an international workshop on component-based software engineering (CBSE). This workshop was held in conjunction with the 20th International Conference on Software Engineering (ICSE). The purpose of the workshop was to help the software community develop a better understanding of CBSE, and in particular to identify gaps that exist between industry needs and current academic research in CBSE.

The workshop was organized as a series of panel discussions. Each panelist made a short (15 minute) presentation of their position, after which there was 30-40 minutes of open discussion. The panel presentations were organized by the following broad categories:

- Engineering theory and methodology--what do we mean by CBSE and what is its influence on the way we develop systems?
- Components and object technology--how does CBSE differ from object technology, and how is object technology used to realize CBSE?
- Tools and technology--what are the key technologies, how can they be effectively used, and how might they be extended or otherwise improved?
- Real-life, enterprise-level application of CBSE--why are large enterprises interested in CBSE, what are they doing, and what are their experiences?

In addition to panel discussions, two invited keynote presentations set the tone for each day's session. Dr. Mikio Aoyama's (NIIT, Japan) keynote outlined a large-scale vision for CBSE. Dr. Wojtek Kozaczynski's (SSA, USA) keynote described in detail the notion of *business* component, and

offered a case study in the use of business components in the design of a very large-scale enterprise information system.

The nominal focus of the workshop was on component management infrastructures--the build-time and run-time infrastructures for component-based systems. However, as will be seen, the workshop participants charted a more diverse and comprehensive course.

DIVERSITY OF PERSPECTIVES

CBSE is generating tremendous interest not just in the software community but in numerous industry sectors. Recent technology advances, including the Web, Java Beans, ActiveX and others spur this interest. But CBSE goes well beyond these *technology enablers* as attested by the diversity of perspectives brought to the workshop by its participants. This diversity was also apparent in a related ICSE panel discussion, "CBSE: Can It Change the Way of Software Development?" which generated active debate from a large audience. The debate ranged over wide ranging topics, from the theory of software reuse to the reality of commercial software markets, from available tools to future programming language mechanisms, and from practical testing to rigorous (formal) specification.

In both ICSE panel and CBSE workshop this diversity at times seemed to threaten to blur the conceptual outlines of CBSE beyond all recognition. Does this diversity imply that we are exploring the same basic CBSE concepts from many different points of view? Or are we exploring fundamentally different or unrelated concepts that we are capriciously labeling as CBSE? The results of this workshop suggest that the former is the case: CBSE is a real and emerging engineering practice, and we are making good progress in identifying both the core concepts of CBSE as well as different perspectives on these concepts. In fact, we found that the diversity of perspective, far from diffusing CBSE, often works like stereoscopic vision--it provides "depth of field" to our perception of CBSE concepts.

We illustrate one example of this diversity of perspective. There was general consensus that components act as *replacement units* in component-based systems. However, the concept of replacement unit has different meanings depending upon which of two major perspectives are adopted. The first perspective (CBSE with off-the-shelf components) views component as commercial-off-the-shelf commodity. In this perspective, CBSE requires industrial standardization on a small number of component frameworks. The second perspective (CBSE with abstract components) views component as application-specific core business asset. This perspective places much less emphasis on standard component infrastructures or component marketplaces, and instead emphasizes component-based design approaches. Although this illustration exaggerates the tendencies of each perspective (both of which will operate in any large system development effort), it does reflect real differences that conditioned much of the workshop discussion.

The rest of this summary will focus on the results of the last session of the workshop--a session devoted to synthesis of the major themes from the panel discussions. It was in this session that many of the different perspectives on CBSE were brought into focus. An online version of the workshop proceedings is in production[1].

[1] Check www.sei.cmu.edu/cbs/icseworkshop.html for the availability of these proceedings and selected panel presentations.

SYNTHESIS OF WORKSHOP RESULTS

The closing session was organized as a facilitated, large-group brainstorming session. The objective was to identify the major results that participants could take away from the panel discussions. The discussion ranged from the conceptual--what is CBSE? to the skeptical--why will it work now if it didn't before? and finally to the practical--what will it mean to organizations if CBSE does work? Each of these will be discussed in turn.

WHAT IS CBSE?

Predictably, some discussion time focused on definition of terms--notably, the term "component." Just as predictably, these definitions only sketched the contours of this complex concept. Fortunately, the workshop was able to make use of several established definitions, and to their credit the participants used these definitions as a basis for exploring additional characteristics of components rather than arguing for or against the validity of any particular definition.

The following (different) definitions of *software component* are representative of those that are emerging in the software industry.

1) *A component is a non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces*[2].

2) *A run-time software component is a dynamically bindable package of one or more programs managed as a unit and accessed through documented interfaces that can be discovered at run-time*[3].

3) *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party.*

4) *A Business Component represents the software implementation of an "autonomous" business concept or business process. It consists of all the software artifacts necessary to express, implement and deploy the concept as a reusable element of a larger business system.*

A close inspection of these definitions is revealing in that they seem to describe approximately the same concept, but there are sufficient differences to make them non-substitutable. For example, definitions 1 and (especially) 4 are explicit about the large-grained nature of components; this *might* be inferred from definition 2, but can *not* be inferred from definition 3. Definitions 1 and 3 are explicit about the need to accommodate context dependencies, but *only* definition 3 requires explicit description of context dependencies. Explicit context dependencies *might* be inferred from definition 4 (which uses a convenient if not descriptive "catch all" phrase about what is needed to "express, implement and deploy" components), but can *not* be inferred from definition 2. Similar variation can be observed concerning the notion of component autonomy--the ability to of components to be deployed independently, or execute independently, and so forth.

[2] As presented in Philippe Kruchten's (Rational) position statement.

[3] This and the remaining definitions were presented in Wojtek Kozaczynski's keynote. The first is from the Gartner Group, the second from Szyperski's "Component Software," while the last is the one adopted by Dr. Kozaczynski's for his work.

Each of these definitions has merit. Rather than debate these merits in detail, the workshop participants decided that two additional characteristics of components need to be noted--the first deals with the relationship between components and object technology, the second deals with the relationship between components and software architecture.

Object technology is neither necessary nor sufficient for CBSE

It is curious that such a strong statement--as obvious as it may seem to some--should have passed without more vigorous contention. Instead, this assertion was generally acknowledged to be a natural consequence of the panel presentations and discussions--and this notwithstanding the fact that most available technologies for component-based development clearly *are* object-oriented. For example, Java Beans and Enterprise Java Beans are prime exemplars of component-based technology. From the methodological arena UML, itself an outgrowth of OOA/OOD, actively addresses component concepts. In this context it seems strange to assert the independence of CBSE from OT. How can this apparent incongruity be justified?

To state the conclusion first, the workshop participants agreed that object technology (OT) was a useful and convenient starting point for CBSE, but a) by itself, OT did not express the full range of abstractions needed by CBSE, and b) it is possible to realize CBSE without employing OT. Thus, OT is neither necessary (b) nor sufficient (a). Moreover, as discussed later, CBSE might induce substantial changes in approach to system design, project management, and organizational style--changes that go well beyond those implied by a large and growing base of industry experience with OT.

To illustrate the *insufficiency* of OT for CBSE, consider the role of component as replacement unit in a system. The earlier definitions of component addressed at least one characteristic that relates to replaceability--explicit context specification. Concretely, explicit context specification might be implemented via a "uses" clause on a specification[4], i.e., a declaration of what system resources are required for the component to work. OT does not typically support this concept. This is *not* to suggest that OT should adopt "uses" clauses--there are strong arguments why it should not. However, these same arguments lose some of their force when applied to design-level (as distinct from programming-level) abstractions, and especially where a compositional style of development from *existing* components is desired.

To illustrate the *non-necessity* of OT we (ironically) draw on the experiences of workshop participants in attempting to use OT to implement CBSE. Put bluntly, some practitioners are attempting to find ways to insulate their approaches to CBSE from OT. Why? Because the OT technology market--in particular, distributed OT such as Java, CORBA, Active/X--is far too unstable and contentious, and will, in the opinion of many, continue to be unstable and fragmented. The tendency of workshop discussion was to treat distributed OT as infrastructure "plumbing" and to treat components as larger-grained abstractions and implementations that can be applied to a variety of different infrastructures. Whether a complete separation of components from infrastructure is feasible led to the following discussion on the relationship of component and architecture.

[4] This is actually a contentious issue even in CBSE: a "uses" clause would seem to imply that the interface describes an *implementation* rather than an *abstraction* over many *possible* implementations. Once again we can see the perspectives of component as off-the-shelf implementation vs. component as design abstraction at work.

Components are inseparable from architecture

If one of the motivations for CBSE is to improve system flexibility through a compositional style of development, a natural question is what makes composition possible? It has to be more than our ability to describe abstractions via abstract interfaces--if that were all that was needed there would be no need for CBSE. Instead, it is apparent that the degree to which we are able to "plug in" components--the operative phrase in compositional development--is directly related to the degree to which components adhere to some set of pre-defined constraints or conventions. The most prominent component technologies--(Enterprise) Java Beans, ActiveX, and CORBA[5] all impose constraints on components. To illustrate with a simple example, the ability of a component infrastructure to inquire into the interfaces of a component (see definition 2 for component, above) requires the component to implement some service or obey some convention that is *required by its underlying component infrastructure*.

With this in mind, it was suggested that components implement two kinds of interface: a functional interface that reflects the role of a component in the system, and another *extra*-functional interface that reflects the component model imposed by some underlying component framework. These extra-functional interfaces express the architectural constraints that enable composeability and other desirable properties of component-based systems. Therefore, our understanding of what makes a component a component is inextricably linked to our understanding of the architectural constraints imposed on components by a component framework-cum-object model.

After some discussion, however, the workshop participants decided that although it is valid to assert that components and architecture go hand in hand, the "two interface" suggestion outlined above places too much emphasis on the role of component framework in software architecture. Indeed, as noted, some participants were looking for a clean separation between software architecture and component framework. A more general conception avoids this problem but still preserves the notion of component/architecture duality by recognizing three different views of architecture:

- Run-time: This includes component frameworks and component models that provide run-time services for component-based systems.
- Design-time: This includes the application-specific view of components, such as functional interfaces and component dependencies.
- Compose-time: This includes all that is needed to assemble a system from components, including generators and other build-time services (a component framework may provide some of these services).

These additional characteristics of components that emerged from the workshop discussion suggest that components are complex design-level entities (i.e., both abstractions and implementations). The question is whether this complexity serves to solve enterprise-level problems. This leads to the discussion on the motivators behind CBSE.

WHY CBSE NOW?

Over the past decade there have been many attempts to improve software development practices by improving design techniques, by developing more expressive notations for capturing a system's

[5] Assuming the Object Management Group adopts a component model.

intended functionality, and by encouraging reuse of pre-developed pieces of a system rather than building from scratch. Each of these has had some notable success in improving the quality, flexibility, and maintainability of application systems, with many organizations having developed complex, mission-critical applications deployed on a wide range of platforms.

Despite this success, tremendous problems must be faced by any organization engaged in developing, deploying, and maintaining large-scale software-intensive systems. Furthermore, the past few years has seen a number of significant changes in the requirements, tactics, and expectations of application developers. These provide the context within the question "why CBSE now?" was examined, and various CBSE solutions discussed.

In examining the context and maturity of CBSE, two important aspects of the question "why CBSE now?" quickly emerged. The first aspect is the maturing of a number of underlying technologies from which to build components and assemble applications from sets of those components. The second aspect is the change in business and organizational context within which applications are developed, deployed, and maintained.

MATURING COMPONENT TECHNOLOGIES

A number of workshop participants voiced the opinion that CBSE is happening now. The highlighted the fact that the past few years have seen changes in the way systems are developed. Development environments such as Visual Basic, and languages such as C++ and Java dominate new application development. These languages, and the tools supporting them, have brought with them the ability to share and distribute pieces of applications through approaches such as Visual Basic Controls (VBXs), ActiveX controls, class libraries, and Java Beans. As these technologies have matured, so has understanding about how to develop pieces of applications following these approaches. Notions of component-oriented development are no longer foreign to many of today's application developers.

Each of these component approaches relies on some underlying services to provide the communication and coordination necessary to make applications from components. The component infrastructure acts as the "plumbing" that allows communication among components. In order for components to communicate they must share an understanding of how to use the infrastructure. This could be as simple as a set of naming standards for operations, a standard place to put information about the components, and in particular a set of conventions about how to make use of other components using the infrastructure.[6] In addition, this infrastructure may provide functionality in a number of areas to allow components using the infrastructure to do so effectively and efficiently. This may include services to:

- find out what components are currently connected to the infrastructure;
- make reference to other components via some meaningful naming scheme;
- guarantee once-only delivery of messages between components;
- manage transactions consisting of multiple interactions among components;
- allow secure communication between components.

[6] Such a standard is sometimes referred to as a *Component Model.*

A number of component infrastructure technologies have been developed. There seem to be three specific infrastructures on which some measure of standardization is beginning to occur, and for which many components, tools, and methods are now available – the Object Management Group's (OMGs) Common Object Request Broker Architecture (CORBA), Sun's Java Beans and Enterprise Java Beans, and Microsoft's Component Object Model (COM) and Distributed COM (DCOM). Each of these component infrastructure technologies was discussed at the workshop, with participants relating examples of their use in various operational contexts.

Tools and environments supporting each of these technologies are widely available, and in use in a large number of organizations. Many benefits of using these tools are being realized. However, as highlighted by a number of workshop participants, it is also being found that many challenges must be faced when using these tools for the development of larger applications, in the management of multiple versions of components, and when integrating components developed by different people using a variety of technologies.

EVOLVING BUSINESS AND ORGANIZATIONAL CONTEXT

Far from concentrating exclusively on component technology issues, a number of workshop participants broadened the scope of the discussion to consider the business and organizational context within which CBSE must operate. A number of important recent developments in this regard were suggested.

First, the style and architecture of the applications being developed has significantly changed. Over the past few years there has been a major shift from centralized mainframe-based applications accessed via terminals over proprietary networks toward distributed, multi-tiered applications remotely accessible from a variety of client machines over intranets and the internet. Building such applications requires application development tools and techniques to evolve to support new methods and approaches for application development. Where organizations used to be involved in a small number of large projects, they now are typically involved in a larger number of smaller projects whose results must be shared.

Second, organizations have made significant financial and intellectual investment in the applications they have built over the past two decades. The resources required for developing new applications from scratch are typically not available, with the result that organizations must look for ways in which they can leverage and reuse their existing investment across a range of their applications. It has become of strategic importance to be able to reuse existing knowledge to enable new applications to be assembled quickly and reliably. To achieve this developers require greater support and guidance for decomposing applications into meaningful pieces, and for assembling new applications from a mixture of new and existing pieces.

Third, organizations began to realize the strategic impact on their business practices held by software-intensive systems supporting their organization. Significant costs were experienced by a number of organizations that found themselves locked into proprietary software solutions. Typically this arose from two sources. On the one hand, some organizations had attempted to develop large parts of their software infrastructure (both systems software and application software) themselves. Often they found themselves responsible for a growing, and very expensive, software maintenance backlog. Frequently they found themselves at a disadvantage against more agile organizations, which could quickly respond to customer and market changes by updating or replacing their computer infrastructure. On the other hand, some organizations found that they had relied too much on a single product from single vendor for their computing infrastructure. The resultant "vendor lock-in" made it difficult to take advantage of a free market of computing suppliers, left important decisions about computing infrastructure in the hands of third parties, and often significantly reduced the ease

with which information could be shared among partner organizations. A particular example of this has been the move toward complete, packaged applications that are frequently found to be without the flexibility required to be readily customized for specific organizational needs. Organizations began to look for an appropriate balance between writing everything from scratch each time, and being able to evolve existing systems at a sufficient rate to meet changing business needs.

Finally, and perhaps most importantly, a revolution has occurred in the business environment in which organizations operate. A key to the success of many organizations is to maintain some measure of stability and predictability in the markets in which they operate, in the technology employed to support its core businesses, and in the structure of the organization itself. Unfortunately, the past few years has seen an inexorable rise in rate of change faced by organizations in all of these areas. Strategic advantage can be gained by those organizations that can deal with these changes most effectively. The ability to manage complexity and rapidly adapt to change has become an important differentiator among competing organizations.

The solution to these problems seemed to lie in an approach to software development that addresses each of these requirements. The workshop highlighted the goals of CBSE as the need to:

- embrace the opportunities offered by new technologies in the delivery and deployment of software systems;
- encourage reuse of core functionality across applications;
- enable flexible upgrade and replacement of pieces of a system whether developed in-house, by third parties, or purchased off-the-shelf;
- encapsulate the best practices of organizations in a way that allows them to be replaced in the face of a variety of different change scenarios.

LARGE SCALE IMPLICATIONS OF CBSE ON ORGANIZATIONS

Workshop participants experienced with CBSE development projects were quick to point out that CBSE is much more than simply using object request brokers, setting up a library of useful code, or acquiring Visual Basic controls over the internet. While these may all be tactical approaches to realize CBSE, there is more strategic thinking and planning that must take place for an organization to successfully apply CBSE as key approach to software development. In particular, there are very broad implications of CBSE as a new approach to building, acquiring, assembling, and evolving systems. As a result, for CBSE to be successfully adopted within an organization many concerns must be addressed.

Two classes of concerns can usefully be distinguished, based on whether components are used as a design philosophy independent from any concern for reusing existing components, or whether components are off-the-shelf building blocks used for the design and implementation of a component-based system. (This distinction was illustrated earlier as a major difference in perspective held by workshop participants). For discussion purposes we will denote these classes as CBSE with abstract components and CBSE with off-the-shelf components, respectively. As noted earlier this distinction is useful despite the fact that these classes of issues rarely exist in isolation of each other.

CBSE with abstract components.

CBSE with abstract components involves radically rethinking the relationship between design, requirements, and components. Fundamentally, it requires new methods for software development to be developed and applied, new processes, and the availability of powerful tools to automate generation and management of components and interfaces. These CBSE-oriented methods and tools are currently under development and provide an interface-based design focus to development concentrating on the basic component architecture of a solution. As experienced by workshop participants, such an approach provides stability of a system design at the interface level, concentrates attention on collaborations among interfaces as the basis for understanding a system architecture, and enables reuse and replacement of implementations conforming to the interface specifications. A number of CBSE methods are beginning to emerge [1, 2, 3] and are being tracked or applied in current projects. However, workshop participants were eager to point out that a great deal of further experience with these approaches is required.

One particular consequence of this revolution in design approaches was pointed out by a number of workshop participants: a dramatic change in the primary roles of software engineers, and the skills they require to be effective. A number of organizations moving toward a CBSE approach have found that the must re-think the organization of their teams to concentrate on component provisioning within a well-defined component architecture. Finding people who can operate in this environment is proving to be a major challenge. The lack of the appropriate skills within an organization could severely hamper the adoption of CBSE.

An important issue raised by Dr. Kozaczynski was that the major day-to-day challenge organizations face in moving to a CBSE approach will be in the management of component-based applications as they are deployed, and maintained, and continue to evolve. It is expected, for example, that multiple components will be available providing similar functionality, many versions of the same component will emerge, multiple configurations of sets of components will be in use, and so on. Traditional configuration management and version control techniques provide an important starting point to manage some of the issues that arise. However, as the emphasis moves from monolithic development and deployment to component-oriented approaches, it has often been found that new management methods and tools are essential. In particular, high composeability in a product line setting amounts to mass customization and this introduces tremendous configuration management challenges and support challenges. There are many opportunities for new tools and techniques in this area.

CBSE with off-the-shelf components.

CBSE with off-the-shelf components moves organizations from the task of application development to one of application assembly. The primary approach used to construct an application is now the use of pre-developed pieces. In many cases these pieces have been developed at different times, by different groups of people, and with many different styles of use in mind. There are many implications of making such a change, which we illustrate by considering one particular scenario highlighted in detail at the workshop: black box assembly of commercial off-the-shelf (COTS) components.

During the workshop some details were discussed regarding the experiences of one U.S. Government command and control application developed using a large number of commercial off-the-shelf packages. In assembling COTS components the organization was placed in a situation in which they had limited access to the internal design of the component, pre-defined options for

customizing the component's behavior, no ability to influence the release cycle of new versions of the component, and total reliance on the long-term viability, integrity, and ability of the packages' maintainers. As a result, many aspects of the design, assembly, testing, deployment, and maintenance of a system were effected. As well documented in [4], in such cases the development effort becomes one of gradual discovery about the components, their capabilities, their internal assumptions, and the incompatibilities that arise when they are used in concert. In fact, as experienced in this example, the architecture of a COTS-based system often degenerates into a series of contingency and risk mitigation strategies based on this discovered information. The workshop concluded that as the emphasis on out-sourcing of systems and increased use of COTS components continues to grow, many improvements are needed in the ways in which such components are documented, assembled, adapted, and customized.

SUMMARY

The adoption of software technology as key to business enterprises in all market sectors is generating tremendous demand for more flexible enterprise systems. This demand coincides with a maturing software technology infrastructure for building distributed enterprise systems. CBSE is a new style of software system development that is emerging from this growing demand and maturing technology. While CBSE is still evolving, and while there is a diversity of perspectives about what CBSE is all about, there is little doubt that *something* is happening, we are calling that something CBSE, and that the outlines of CBSE are becoming clearer all the time.

The ICSE Workshop on CBSE provided an opportunity for researchers and practitioners to share their differing perspectives on CBSE. A synthesis of these different perspectives helped the workshop participants to understand better the nature of components, what is motivating the emergence of CBSE, and what potential implications of CBSE are on organizations. The ICSE CBSE Workshop is not the first or only forum for exploring CBSE concepts. However, as a premier conference on software engineering, ICSE provides a superb context for this type of workshop.

ACKNOWLEDGEMENTS

Thanks are due to the ICSE workshop organizers and Dr. Mikio Aoyama in particular, for the excellent workshop arrangements. Thanks also to Wojtek Kozaczynski (SSA) for his detailed recording of the summary session, and to Wojtek Kozaczynski, Philippe Kruchten (Rational), Chris Dellarocas (MIT), David Carney (SEI) and Mikio Aoyama for their comments on this summary.

REFERENCES

[1] D. D'Souza and A.C. Wills, "Objects, Components, and Frameworks with UML: The Catalysis Approach", Addison-Wesley, 1998.

[2] P. Coad, "Java Design: Building Better Applications and Applets", Prentice Hall, 1997.

[3] P. Allen and S. Frost, "Component Based Development for Enterprise Systems: Applying the Select Approach, Cambridge University Press," 1997.

[4] D. Garlan et al., "Architectural Mismatch: Why it is Hard to Build Systems Out of Existing Parts", IEEE Software, November 1996