A Review of Component-Based Software Development

Jason Sharp
Tarleton State University

Sherry Ryan
University of North Texas

Follow this and additional works at: http://aisel.aisnet.org/icis2005

Recommended Citation
http://aisel.aisnet.org/icis2005/13

This material is brought to you by the International Conference on Information Systems (ICIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in ICIS 2005 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
A REVIEW OF COMPONENT-BASED SOFTWARE DEVELOPMENT

Jason H. Sharp
Tarleton State University
Stephenville, TX U.S.A.
jsharp@tarleton.edu

Sherry D. Ryan
University of North Texas
Denton, TX U.S.A.
ryans@unt.edu

Abstract

The purpose of this paper is to review and synthesize the current component-based software development (CBSD) literature, to derive a research taxonomy, and to propose an agenda for conducting future research. Due to the rapidly changing nature of technology, increasing development cost, and decreasing development time, researchers and practitioners alike are considering alternatives to traditional development approaches that have been in use and effective for many years. The following research question is examined: Does component-based software development (CBSD) represent a distinct information systems development approach (ISDA), or is it simply an iteration of existing approaches, therefore more accurately representing an information systems development methodology (ISDM)? Based on the review of literature and using Iivari, Hirschheim, and Klein’s four-tiered framework for classifying information systems development as a theoretical lens, a taxonomy is created, six propositions are developed, and future research is recommended.

Keywords: Information systems development, information systems development approach, information systems development methodology, taxonomy, components

Introduction

Due to the ever-changing technology market, current research suggests that component-based software development (CBSD) represents the future of software development (Brereton and Budgen 2000; Fan et al. 2000; Sugamaran and Storey 2003; Vitharana 2003; Vitharana and Jain 2000). However, there is debate among researchers on the degree to which CBSD will impact the development community. Some believe that CBSD represents a revolutionary approach and will totally replace building systems from scratch (Vitharana 2003; Vitharana and Jain 2000). Others argue that CBSD represents an evolutionary change, but recognize the possibilities that CBSD has for reducing the problems associated with more traditional methods (Brereton and Budgen 2000; Sugamaran and Storey 2003).

Both camps agree, however, that CBSD has the potential to assist organizations in the move away from monolithic ERP systems, which tend to be large, rigid, and difficult to modify without a major investment of time and money (Brereton and Budgen 2000; Fan et al. 2000). In fact, flexibility along with portability and reuse are leading factors for the adoption of CBSD over more traditional approaches (Fan et al. 2000; Hong and Lerch 2003). A prime example is Dell Computer, which made the decision to migrate from its SAP project after only 2 years and to adopt the component object model (COM) and the CBSD approach. Why would a major organization choose to drop a system in which millions of dollars had already been invested? Dell’s decision was based on the belief that a large ERP system was not flexible enough to accommodate its future needs and to meet the challenges of ever changing business processes (Fan et al. 2000; Lycett 2001).

Despite the broad interest in CBSD, there has been no integrative review of CBSD research in the IS literature, no taxonomy suggested for future research, nor a theoretical placement as to where CBSD “fits” in the myriad of software development approaches and methodologies. This paper fills these gaps, first, by synthesizing the current CSBD literature and deriving a taxonomy from the existing streams of literature. Second, for each element of this taxonomy, a proposition is formulated, providing an agenda for future research. Third, we provide initial evidence for the research question: Does component-based
software development (CBSD) represent a distinct information systems development approach (ISDA) or is it simply an iteration of existing approaches, therefore more accurately representing an information systems development methodology (ISDM)? As the IS community grapples with classifying advances in software development such as CBSD, differentiating between an ISDA and an ISDM is an important step to help organize thinking so that common understanding can be achieved. It will also provide insight as to the degree of change that CBSD might induce if adopted by software developers.

The remainder of the paper is organized as follows. First we discuss the theoretical basis for classifications of information systems development. We then compare the core features of CSBD with other system development approaches. Next, we synthesize the CSBD literature and define propositions for future research. Finally, we summarize our findings and propositions.

Classification of Information Systems Development

In an attempt to classify and understand the multitude of information systems development methodologies, Iivari, Hirschheim, and Klein (2001) proposed a four-tiered framework. The framework includes paradigms, approaches, methodologies, and techniques. Paradigms represent the highest level (fourth) of abstraction and are represented by functionalism, social relativism, neohumanism, and radical structuralism. Paradigms are concerned with such issues of ontology, epistemology, methodology, and ethics. The authors suggest that there should only be limited number of paradigms. The information systems development approach (ISDA) comprises the next (third) level and inherits fundamental features from one or two dominant paradigms. The authors define an ISDA as “a set of related features that drive interpretations and actions in information systems development” (p. 186). An ISDA contains four features: (1) goal, the overall purpose, (2) guiding principles and beliefs (i.e., philosophy), (3) fundamental concepts, which define the nature of the approach, and (4) principles for the ISD process. The information systems development methodology (ISDM) makes up the next (second) level. ISDMs consist of concepts, methods, beliefs, values, and principles. Finally, the lowest (first) level represents techniques. A technique “consists of a well-defined sequence of elementary operations that more or less guarantee the achievement of certain outcomes if executed correctly” (p. 186).

According to Iivari et al. (2001) differentiating between levels of this framework (e.g., between ISDAs and ISDMs) is important because it “distinguishes the essential features of methodologies from their less essential ones by associating them with approaches and paradigms” (p. 182). In this way, it transfers the discourse of alternative methodologies and tools to a higher level of abstraction and acts as an “organizing and simplifying vehicle” for the IS community. It also supports the complementary view of methodology engineering that suggests that methodologies are “instantiations of existing ISD approaches” and can, therefore, make software developers more aware of the core attributes of the overarching ISDA. This in turn allows software developers to broaden their available methodology choices. In addition, it provides common terminology so that the precision of discussion can be enhanced. Many authors seem to use information systems development terminology, such as paradigm and approach, almost interchangeably. This causes a significant amount of ambiguity. Consistent with Iivari et al. (2001) we use the term approach throughout this paper.

Iivari et al. (1998) discuss ways of deriving an ISDA. First, one can abstract an ISDA from ISDM’s that have common elements. Next would be to differentiate an ISDA’s core features (goals, guiding principles and beliefs, fundamental concepts, and principles for the ISD process) from other ISDAs. The authors point out that some ISDAs have no concrete methodologies, while others may have one or more ISDMs. As we review the CBSD literature, we extract the core features of the CBSD approach. Although Iivari et al. (2001) identified 11 ISDA’s, we contrast CBSD with two that are frequently used in industry: the structured approach and the object-oriented approach. To briefly illustrate the functionality of the framework, working from the bottom-up, the ISD techniques—data flow diagram, entity relationship diagram, and state transition diagram—are part of the ISD methodology modern structured analysis, which is an instance of the structured approach belonging to the functionalist paradigm. Similarly, the ISD technique use case is a part of the ISD methodology object-oriented software engineering, which is an instance of the object-oriented approach belonging to the functionalist paradigm (Iivari et al. 2001). This framework allows for the classification of business processing reengineering, functional points, and extreme programming as ISD methodologies each consisting of specific concepts, methods, beliefs, values, and principles.

CBSD Versus Other Development Approaches

In comparing CBSD with other development approaches, we revisit our initial research question: Does component-based software development (CBSD) represent a distinct information systems development approach (ISDA) or is it simply an iteration of
existing approaches, therefore more accurately representing an information systems development methodology (ISDM)? In Table 1, we compare CBSD with the object-oriented and structured approaches based on the four features identified by Iivari et al. (1998, 2001): (1) goal, the overall purpose of the approach, (2) guiding principles and beliefs (i.e., philosophy), (3) fundamental concepts, which define the nature of the approach, and (4) principles for the ISD process. The features of the object-oriented and structured approaches were taken from Iivari et al. (2001, p. 192-193). The core features of CBSD were derived from the review of the literature.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Component-Based Software Development</th>
<th>Object-Oriented Approach</th>
<th>Structured Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>To provide an approach that helps to reduce the cost, time, maintenance and problems associated with other approaches through the integration of predefined, pre-built components available from multiple, independent sources, for developing highly flexible systems that will meet the challenges of rapidly changing business processes</td>
<td>To provide an approach that helps to ensure that the products are delivered to the user on time and within budget, that the products meet user requirements, that the user requests to modify the system and/or fix bugs are responded to in a timely fashion, that increasingly sophisticated products are offered to keep a competitive edge, that changes in standards and delivery technology are kept up and the project team feels motivated and successful.</td>
<td>To provide an approach that helps to produce high quality (reliable and maintainable) software in a productive way.</td>
</tr>
<tr>
<td>Guiding Principles and Beliefs</td>
<td>Separation of software element and interface; components as the central focus of the entire system life-cycle</td>
<td>Seamless analysis, design, and implementation.</td>
<td>Separation of the essential model from the implementation model; careful documentation to make the development process visible; graphic notations; top-down partitionable transformation process models to hide complexity; unambiguous, minimally redundant, graphic specification; balancing of models with high cohesion and weak coupling</td>
</tr>
<tr>
<td>Fundamental Concepts</td>
<td>Well-defined interface; component awareness, placing high value on components rather than constructing systems from scratch; communication between components black-box concept; stand-alone use and testing</td>
<td>Problem domain versus implementation domain; object and class, encapsulation; information (implementation) hiding; inheritance; polymorphism; communication between objects</td>
<td>Essential model versus implementation model; transformation (processes); data flow; data store; terminator; module; cohesion; coupling</td>
</tr>
<tr>
<td>Principles</td>
<td>Choosing appropriate components from multiple, independent component market sources; portability; flexibility; reusability</td>
<td>Iterative and incremental development; reuse.</td>
<td>A step by step process at the detailed level of analysis and design activities; situation dependent at the strategic level (waterfall, prototyping, concurrent)</td>
</tr>
<tr>
<td>Examples of Systems</td>
<td>Credit card processing; data conversion; e-mail and messaging; office automation</td>
<td>Interactive; Web-based; expert systems</td>
<td>Transaction processing systems; ERP, management information systems</td>
</tr>
</tbody>
</table>
Although not specifically listed in Table 1, the system development life cycle (SDLC) has been a widely used development methodology that is often associated with the structured approach. The traditional SDLC consists of a series of phases, which assist in the development and support of an information system. Although the terminology may vary slightly, Hoffer et al. (2005) identify the following phases: (1) planning, (2) analysis, (3) design, (4) implementation, and (5) maintenance. The primary advantage of the SDLC is to provide a logical, systematic, methodology for managing the development process. A distinguishing characteristic of the structured approach is its separation of data and processes. Hoffer et al. identify three specific criticisms of this methodology. First, the emphasis on milestones makes it difficult to facilitate changes after the milestone has been reached. Second, the SDLC tends only to involve users in requirements gathering and omits their input from the other phases, often resulting in a system that fails to meet the needs of the user. Third, the maintenance cost of a system developed using the SDLC has a tendency to escalate.

The object-oriented approach, in contrast to the structured approach, is centered on combining data and processes into objects, which represent real-world things in the information system. A key concept is inheritance, which is the ability to create new objects that share the characteristics of an existing object, Hoffer et al. state that the goal of the object-oriented approach is “to make system elements more reusable, thus improving system quality and the productivity of systems analysis and design” (p. 25).

In general, the CBSD approach differs from previous approaches in that it does not assume nor does it require that the developer have knowledge of either the data structures or the processes necessary to make the component function. The concept behind CBSD is the assembling of pre-fabricated components, much like in the area of computer hardware or electronics. Concrete examples of components include credit card authorization and processing, data conversion, spell-checking, interest calculation, and accounts receivable (Jain et al. 2003; a large collection of components can be found at http://www.componentsource.com/). A shopping cart, for instance could serve as a common example of a component if it is a completely stand-alone program that does not require or assume that the developer has a working knowledge of its internal structure.

Review of the CBSD Literature

CBSD is not simply the addition of a few new features to traditional development approaches. Hopkins (2000) and Fan et al. (2000) argue that it represents a major shift and a huge advance in software development. Many researchers believe that CBSD truly is the future of the discipline (Fan et al. 2000; Vitharana 2003). The general concept behind CBSD is that of building systems through the use of predefined components from multiple, independent sources. These components, consequently, form a unique system, which allows for rapid change and modification (Hopkins 2000; Sugumaran and Storey 2003). According to Hopkins (2000) and Waguespack and Schiano (2004), CBSD reveals the maturation of the field and the fact that the focus is no longer on developing large, customized, in-house systems, but rather on the integration of pre-developed components intended to stand-alone to address organizational needs; consequently, requiring its own distinct life-cycle process. Hopkins states “ideally a new system can be built using mostly predefined parts, with only a small number of new components required” (pp. 27-28).

The methodology for examining the CBSD-related literature consisted of keyword searches on four information systems related databases as well as a year-by-year review of three top IS journals (Kohli and Devaraj 2003; Schultze and Leidner 2002). Utilizing Computer Source, Business Source, ABI Inform, and IEEE Express, searches were conducted on the following keywords: component-based software development, CBSD, component-based development, CBD, component-based software engineering, CBSE, business components, software reuse, service oriented application development, asset based software development, and asset based software engineering. A broad search of the keywords listed above indicated that CBSD research was limited before 1990, but began to appear with increasing regularity in the late 1990s. We made the decision to examine the literature from 2000 through 2005 in an effort to present the most current research while attempting to meet the objectives of this paper. In addition to the searches via the online databases, each issue of MIS Quarterly, Information Systems Research, and Journal of Management Information Systems, from 2000 through 2005 was reviewed by title to ensure that related research was not omitted due to the keywords chosen.

The exclusion of specific articles was determined based on the objective of the paper, which was to examine whether or not CBSD represents an ISD approach rather than an ISD methodology. Articles that focused on components from a purely scientific or engineering perspective, rather than from the context of information systems, were excluded. The strength of the journal and the publication date was also a determining factor in exclusion. As mentioned previously, publication date was established as a criterion to ensure that the most recent CBSD research was examined as well as allowing for the maturation of the CBSD field. An iterative process was employed for grouping similar articles together, allowing the development of categories rather than forcing them into a previously defined classification scheme (Strauss and Corbin 1968). Communications of the ACM had the
Influence of the Object-Oriented Approach on CBSD (10%)

Differences between CBSD and Traditional Approaches (15%)

Component Definition (10%)

Component Models (5%)

Storage and Retrieval of Components (15%)

Socio-technical effects of CBSD (15%)

Requirements Identification (5%)

Design (10%)

Benefits and Challenges of CBSD (15%)

Note: (nn%) represents the percentage of articles in this category.

Figure 1. CBSD Research Taxonomy

The highest number of articles reviewed, with 25 percent of the articles, followed by IEEE Transactions with 20 percent. This is not surprising due to the technical orientation of the subject.

The grouping process resulted in six major categories as shown in Figure 1 and the number of articles reviewed in each category. A proposition, derived from the literature, is presented for each major category followed by a discussion of that category. The greatest amount of the research was grouped under the “Differences between CBSD and Traditional Approaches” category and its subcategories, totaling 45 percent of the articles. The least amount of research literature was found under the “Component Models” and “Component Definition” categories. The literature indicates that there is a significant need for research on models that make components the central focus throughout the development of a system (Dahanayake et al. 2003; Seker and Tanik 2003). These categories provide fertile ground for future research.

Influence of the Object-Oriented Approach on CBSD

Proposition 1: Although the majority of research supports object-orientation as the antecedent of CBSD, the principles of the object-oriented approach are insufficient to fully characterize the component concept.

Several researchers trace the origin of CBSD back to the object-oriented model, which in and of itself has been considered a significant revolution in software development in contrast to more traditional structured approaches (Fan et al. 2000; Hopkins 2000; Waguespack and Schiano 2004). In one sense, a component can be thought of as an object, but the differentiation goes back
to the basic definition, which states that the component also has a well-defined interface and a specific use (Hopkins 2000). A further distinction, however, can be made between the two. The object-oriented approach is primarily concerned with the development of objects within the software program, whereas CBSD is concerned with building “software systems by combining and matching pre-developed software objects” (Fan et al. 2000, p. 28).

Not all are in agreement that the principles of the object-oriented approach sufficiently characterize the component concept. Dahanayake et al. (2003) argue that CBSD is not simply an alternative way of using prior technology to produce functionality in object-oriented software; rather, CBSD represents “an opportunity to tighten the loose ends left dangling from the OO era” (p. 22). To go a step further in this direction, a case study conducted by Lycett (2001) challenges the generally held view that object orientation provides a suitable foundation for CBSD. The results of this case study revealed a lack of convincing evidence “that suggest that component-based development is object-oriented development in essence” (p. 206).

**Differences between CBSD and Traditional Approaches**

**Proposition 2:** The characteristics of CBSD in relation to overall philosophy, requirements identification, design, benefits, and challenges provide a basis for adequately distinguishing it from traditional approaches.

CBSD differs from traditional approaches in the following ways. First, systems are constructed from predefined and pre-built components rather than from scratch (Fan et al. 2000; Hopkins 2000; Sugumaran and Storey 2003; Waguespack and Schiano 2004). Waguespack and Schiano use the term component awareness to identify this concept. Component awareness emphasizes the placement of a high value on using components instead of building a new system from the ground up.

Second, components are constructed as stand-alone projects, whereby, development and testing may be conducted in isolation from other components (Sparling 2000; Waguespack and Schiano 2004). In relation to this second difference, Hopkins uses the term evolution to describe a significant driver for CBSD. Specifically, evolution has to do with system maintenance, whereby components can be replaced or upgraded with little or no impact on the other components.

Third, a major emphasis is placed on the reuse of components (Hopkins 2000; Seker and Tanik 2004; Sparling 2000; Waguespack and Schiano 2004). In addition to evolution, Hopkins identifies reuse as a second predominant driver for CBSD and concurs with Waguespack and Schiano that reuse is one of the central premises of CBSD. Ideally, once a component has been developed and thoroughly tested, it can then be used in multiple applications. Sparling stresses the importance of reuse by stating that it is often a key to selling the idea of components to management. Lycett (2001) reiterates that the concept of reuse should underlie every phase of the system life cycle. Finally, the separation of the interface from the software itself or physical packaging is also an important distinguishing characteristic of CBSD from traditional approaches (Hopkins 2000).

**Requirements Identification**

Jain et al. (2003) emphasize the fact that requirements analysis is a critical aspect of systems development. Unfortunately, many systems fail because requirements are not correctly identified. The authors contend that traditional methods for eliciting requirements are actually a part of the problem. It is argued, therefore, that CBSD in comparison to traditional methods improves the requirements identification process. This shift affects both the traditional phases of the system development life cycle as well as the role of the user in development. First, in relation to the traditional life-cycle phases, which typically include requirements identification, analysis, design, development, and testing, CBSD entails requirements identification, component selection, component assembly, and integration testing. A major difference in the two approaches is that the developer is no longer the prime participant in design; rather the component builder takes on this role. Second, in regard to the user, CBSD emphasizes the shift from passive users to active participants.

**Design**

Another shift involves the design of components. Vitharana, Jain, and Zahedi (2004) and Vitharana et al. (2003a) propose that business strategy should guide component design and propose a design model called business strategy-based component design (BusCod). The two key assumptions of the model are as follows: (1) a clean-slate approach and (2) a thoroughly analyzed domain based on the object-oriented approach. The concept behind the model is the mapping of managerial goals to technical
features. In a review of the literature across multiple disciplines, the authors identified five specific managerial goals: cost effectiveness, ease of assembly, customization, reusability, and maintainability. Drawing from the object-oriented approach, the authors identified coupling, cohesion, number of components, component size, and complexity as important technical features. In order to map the two pieces of the model, the sign of association, either positive or negative, and the extent of association must be determined. The authors stress the importance of the impact of the technical features on each of the managerial goals. In order to provide initial benchmark values to users of the model, a survey was conducted among experts in the fields of CBSD and object-oriented design.

Benefits

Based on the characteristics of CBSD as outlined above, advocates of the approach identify parallel development and prototyping as two significant advantages (Sparling 2000). Parallel development refers to the ability to have multiple teams working simultaneously on different components of the same project. Parallel development is enabled by the ability to develop and test components completely separate from other components. Prototyping allows for quickly identifying potential problems, easing into the CBSD process, and selling CBSD to stakeholders. Additional benefits include rapid development and reduced lead-time, leveraged costs, enhanced quality, increased customization and maintainability (Fan et al. 2000; Vitharana 2003). The bottom line is that “components with well-defined interface can be quickly combined and assembled to form a complex application system” (Fan et al. 2000, p. 28).

Challenges

This is not to say that CBSD is not without disadvantages and challenges. One of the major concerns involves the growth of the component market and its impact on the future of CBSD as the next predominant stage for software development (Hopkins 2000; Ravichandran and Rothenberger 2003). Fan et al. (2000) contend that a strong market is indispensable for the growth of the component framework due to the fact that it will drive competition, which in turn generates better products at lower prices. Factors contributing to the success of the component market include standardization, customization, certification, quality assurance, and cost.

A specific concern of researchers is that there are currently not enough components available on the market to meet the needs of users (Fan et al. 2000; Hong and Lerch 2003).

Ravichandran and Rothenberger (2003) assert that the current market place is a major stumbling block to the success of CBSD due to its immaturity and current inability to find components that meet the specific requirements of users. Hong and Lerch (2003) suggest that the sluggishness of component manufacturers is simply that they do not have a clear understanding of the software needs of users.

Ultimately, unless component manufacturers seriously address the above-mentioned issues, “component markets could go the way of many other innovations in software development—a lot of hype and no discernable improvement in software development practices” (Ravichandran and Rothenberger 2003, p. 114).

Component Definition

Proposition 3: Although modularization is a common element in structured and object-oriented approaches, a component in CBSD includes distinctly different characteristics than the other approaches.

Seker and Tanik (2004) assert that CBSD is a “software methodology in which all life-cycle processes are based on integration of components” (p. 475). So what exactly is a component? Although a standard definition of a component has not necessarily been agreed upon, the general consensus it that a component is made up of a software element and a well-defined interface (Hopkins 2000; Seker and Tanik 2004; Waguespack and Schiano 2004). As noted previously, it is this separation of the physical packaging and the interface that constitutes an important distinguishing factor of CBSD (Hopkins 2000). Two representative definitions that appear to capture the essence are provided by Hopkins (2000), who defines a component as “a physical packaging of executable software with a well-defined and published interface” (p. 27), and Waguespack and Schiano (2004), who identify a component as “an element of software that is clearly defined and separable from the system” (p. 54).
A central concept of a component is the fact that it is not necessary for the user to have knowledge of its internal structure (i.e., the user does not have to understand the underlying code) (Seker and Tanik 2004; Waguespack and Schiano 2004). The user simply needs to understand the functionality of the component and have access to the interface. The concept of the black-box has been employed to characterize a component. The user is not concerned with what happens inside the box, only with the inputs that enter it and the outputs that are produced (Ravichandran and Rothenberger 2003). Seker and Tanik consider this characteristic a significant shift from both structured systems analysis and object-oriented development. In sum, Vitharana and Jain (2000) assert that “the traditional built-from-scratch software ideology is behind us, and the trend is in the CBSD involving component fabrication and component assembly” (p. 302).

**Component Models**

*Proposition 4: There is a need for models that incorporate components as the central focus throughout the entire system development life-cycle, due to the insufficiency of current models based on object orientation.*

The development of a component model is a critical aspect of CBSD (Hopkins 2000; Sparling 2000). The major component models include distributed component object model (DCOM) from Microsoft, common object request broker architecture (CORBA) from the Object Management Group, and enterprise Java beans (EJB) from Sun Microsystems (Hopkins 2000). More recently developed models include Microsoft ActiveX, Microsoft .NET, and Sun’s J2EE (Waguespack and Schiano 2004). It is the model that serves as the foundation for communication between components. Although a central premise is that components are self-contained, self-functioning units, they must still be able to “talk” to each other. Not only should a standardized component model be employed, but Sparling (2000) suggests that goals should be established as well as the standardization of design principles for analysis, design, development, documentation, and testing tools.

In spite of the fact that the use of CBSD is rising, a comprehensive component-based concept structure or model is considered by some to be insufficient or missing (Dahanayake et al. 2003; Seker and Tanik 2004). Dahanayake et al., argue that in order for true CBSD to exist, components must not simply be considered an element of the implementation and deployment phases. A shift is necessary which makes components the central point of focus throughout the entire system development lifecycle. A further contention is that current models are too highly influenced by object-oriented methodologies and that “fully component-oriented and even component-centered methods are needed, starting and ending with the component concept” (Dahanayake et al. 2003, p. 18). Consequently, in relation to the existing models such as CORBA, COM, and EJB, Dahanayake et al. suggest that only when components become the central focus will these infrastructures be fully utilized for the development of complex systems.

**Storage and Retrieval of Components**

*Proposition 5: Storage and retrieval of components needs enhancement and standardization because there are unique challenges not addressed in traditional approaches.*

Although the ability to effectively store and retrieve components is considered a key element in the success of CBSD, current approaches are not considered adequate to meet the needs of users (Sugumaran and Storey 2003; Vitharana et al. 2003b). Vitharana et al. (2003b) contend “no clear guidelines for classifying, coding, and storing business components exist” (p. 650). In order to remedy this problem, they proposed a storage and retrieval approach rooted in group technology and software reuse paradigms. One primary element taken from group technology is the C&C scheme, which stands for classification and coding. Classification places similar parts into groups and coding assigns symbols to each part. The authors made a comparison between the coding and classification of business components to that of the well-developed scheme used in the manufacturing industry. Overall, the authors adopted a facet-based scheme and identified seven descriptor facets for representing business components on which the authors developed a prototype component knowledge-based repository (CKBR).

Sugumaran and Storey (2003), on the other hand, developed a semantic-based approach to retrieval. In the related research, the authors pointed out the limitations of the current methods (keyword and string search, faceted classification, and signature and behavioral matching) for retrieving components. In contrast, the authors claimed that their proposed approach “provides a natural and flexible way for the user to specify requirements for component search” (p. 12). The approach is based on three steps: (1) initial query generation, (2) query refinement, and (3) component retrieval and feedback. The approach has been used to develop a Web-based system of component search and retrieval using Jess, JavaBeans, and JSP.
Socio-Technical Effects of CBSD

Proposition 6: Implementation of CBSD brings its own set of unique challenges to the social dimension of an organization.

Although a significant amount of literature has been written in regard to the technical aspects of CBSD, there is a stream of research that examines its affect on the social dimension of organizations as well (Huang et al. 2003; Kunda and Brooks 2000; Robey et al. 2001). Kunda and Brooks (2000) make an excellent point by stating “software systems do not exist in isolation, they are used in social and organizational contexts” (p. 716). Huang et al. (2003) affirm this statement by arguing that many times organizations view technology in isolation of the established subcultures, failing to take into consideration how that technology might affect the people working in the organization. Another common assumption of many organizations is that certain technologies, such as CBSD, will foster organizational renewal (Huang et al. 2003, p. 89) in and of itself apart from the human factors.

In a series of case studies, Kunda and Brooks sought to examine the barriers to the successful application of CBSD. In doing so, three behaviors affecting the successful deployment of CBSD were identified: (1) individual, (2) group, and (3) organization. By conducting face-to-face interviews and documentation review, the findings revealed that participation and interaction, as well education about the CBSD concept, are important. It was also recommended that an incremental approach be utilized for the implementation of CBSD. Overall, the findings suggested that incorporating CBSD, like adopting other technologies, can be difficult, but with a consideration of the human factors it can be implemented successfully.

Huang et al. (2003) also utilized a case study approach to examine the implementation of CBSD at a multinational banking corporation. The data for the study was collected via on-site observation, interviews, formal dialogue, and documentation review. The observation took place over a 2-month period, followed by 3 rounds of interviews with 32 employees. Not surprisingly, those employees considered to be on the cutting edge of the banking industry and technology considered CBSD a priority, whereas those employees working in support roles did not.

In the final study, Robey et al. (2001) conducted a research study to examine what types of conflict arise within three different ways of approaching software development: traditional, iterative, and component-based. The authors then sought to determine which types of conflict management were appropriate for each. The findings indicated that the methodology itself significantly affected social interactions as well as contributed to the type of conflict and conflict resolution measure taken.

Development of Research Taxonomy

Based on the definitions provided by Iivari et al. (1998, 2001), the review of the extant literature, and the identification of the core features described in Table 1, an argument can be made that CBSD does indeed represent a distinct ISDA. In addition to the literature, the core features that support this assertion include its goal, guiding principles, and beliefs. In stark contrast to structured and to a lesser but still discernable degree, in relation to object-oriented, the focus on developing software from predefined, pre-built components available from multiple, independent sources, not simply from internally developed objects or modules, does represent a departure from the two existing approaches. The centrality of the component is also a distinguishable characteristic.

However, additional research is recommended before concluding that CBSD represents a distinct ISDA. At the level of fundamental concepts and principles there appears to be considerable overlap with concepts and principles, in the object-oriented approach. This overlap includes the emphasis on flexibility, reusability, and portability, all of which are terms commonly used to describe both approaches. Even the issue of component markets finds common ground in both approaches with the availability of multiple libraries for object-oriented programming languages. The propositions posited in this research (see Table 2) form a future research agenda that focuses on assessing the degree to which CBSD differs from other approaches.

Conclusion

This paper significantly contributes to the information systems field in several ways. First, we integrated and synthesized CBSD research in the information systems literature resulting in a taxonomy derived from the existing streams of literature. Next, for
Table 2. CBSD Research Taxonomy with Propositions

<table>
<thead>
<tr>
<th>Category</th>
<th>Studies</th>
<th>Proposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Influence of the Object-Oriented approach on CBSD</td>
<td>Fan et al. 2000; Lycett 2001</td>
<td>P1: Although the majority of research supports object-orientation as the antecedent of CBSD, the principles of the object-oriented approach are insufficient to fully characterize the component concept.</td>
</tr>
<tr>
<td>Differences between CBSD and Traditional Approaches</td>
<td>Brereton and Budgen 2000; Hong and Lerch 2003; Hopkins 2000; Jain et al. 2003; Ravichandran and Rothenberger 2003; Sparling 2000; Vitharana 2003; Vitharana and Jain 2000; Vitharana et al. 2003a</td>
<td>P2: The characteristics of CBSD in relation to overall philosophy, requirements identification, design, and benefits and challenges provide a basis for adequately distinguishing it from traditional approaches.</td>
</tr>
<tr>
<td>Component Definition</td>
<td>Seker and Tanik 2004; Waguespack and Schiano 2004</td>
<td>P3: Although modularization is a common element in structured and object-oriented approaches, a component in CBSD includes distinctly different characteristics than the other approaches.</td>
</tr>
<tr>
<td>Component Models</td>
<td>Dahanayake et al. 2003</td>
<td>P4: There is a need for models that incorporate components as the central focus throughout the entire system development life-cycle, due to the insufficiency of current models based on object-orientation.</td>
</tr>
<tr>
<td>Storage and Retrieval of Components</td>
<td>Sugumaran and Storey 2003; Vitharana, Jain, and Zahedi 2004; Vitharana et al. 2003b</td>
<td>P5: Storage and retrieval of components represents a dimension of CBSD not identically addressed in traditional approaches.</td>
</tr>
<tr>
<td>Socio-Technical Effects of CSBD</td>
<td>Huang et al. 2003; Kunda and Brooks 2000; Robey et al. 2001</td>
<td>P6: Implementation of CBSD brings its own set of unique challenges to the social dimension of an organization.</td>
</tr>
</tbody>
</table>

each element of this taxonomy, we formulated a proposition to provide guidance and an agenda for future research. Third, we initiated the discussion and provided initial evidence as to whether CBSD is a distinct information systems development approach or merely another information systems development methodology. CBSD has the potential to drastically alter how information systems are developed. We hope that our synthesis of the literature and propositions for future research serve as a building block for additional research in this important area.

References


