Component-Based Software Development: Life Cycles and Design Science-Based Recommendations

Jason H. Sharp
jsharp@tarleton.edu
Computer Information Systems, Tarleton State University
Stephenville, TX 76401, USA

Sherry D. Ryan
Sherry.Ryan@unt.edu
Information Technology & Decision Sciences, University of North Texas
Denton, TX 76203, USA

Abstract

Component-Based Software Development (CBSD) continues to be a relevant area of research in the systems development and software engineering fields. Its focus is on the integration of pre-fabricated software components to build systems characterized by increased portability and flexibility. While the basic premise of CBSD is to build systems by assembling pre-existing software components, existing CBSD development lifecycles have not adequately separated component development from the assembly of the system. In order to fully leverage the potential of CBSD this conceptual separation is necessary. Thus, we propose the CBSD Dual Life Cycle model, which divides the development process into two distinct parts: (1) component development and (2) system development. We describe each phase of these two life cycles and use design science principles, whose objective is to develop business-relevant technology-based solutions, to help formulate recommendations for enhancing quality.

Keywords: component-based software development, life cycle model, design science, systems development

1. INTRODUCTION

The study of Component-Based Software Development (CBSD) continues to be a valid alternative approach in the areas of systems development and software engineering. One assertion is that "the traditional built-from-scratch software ideology is behind us, and the trend is in the CBSD involving component fabrication and component assembly" (Vitharana & Jain, 2000, p. 302). Although some feel that the next phase of CBSD, commonly referred to as service-oriented computing (SOC) is gaining wide acceptance, especially in the distributed systems development paradigm (Fujii & Suda, 2006; Yang, 2003). Elfatatry (2007) asserts that services complement and extend, rather than replace components altogether. The objectives of CBSD can be summarized as follows: development of software from pre-fabricated parts, reuse of those parts across applications and contexts, and easily maintainable and customizable parts to develop new functions and features. The end result is that complex application systems can be developed rapidly by assembling components with well-defined interfaces.

CBSD represents a departure from traditional systems development. Because organizations may not be involved in the actual design of the components themselves, but rather select appropriate components from outside vendors or reuse existing components that were either previously purchased or developed in-house at an earlier time, component development and system development phases should be decoupled. However, existing CBSD development lifecycles have not adequately separated component development from the assembly of the system (Capretz, 2005; Crnkovic, Stig, & Michel, 2005; Fahmi & Choi, 2008; Sommerville, 2004; Vitharana, Zahedi, & Jain, 2003). In order to fully leverage the potential of CBSD this conceptual separation is necessary. Thus, we propose the CBSD Dual Life Cycle model, which divides the development process into two distinct parts: (1) component development and (2) system development as shown in Figure 1 (see Appendix).

Component development involves domain selection and analysis, the design of the component architecture, and component fabrication. Because the basic premise of the model contends that component development is distinct from system development, the model indicates that after the individual components are completed, they are "funneled in" or selected during the component selection phase of system development. Conceptually, "n" number of components may be selected.

System development, on the other hand, is conducted within the organization and follows its own separate life cycle phases. First it involves the determination of requirements; next the design of systems architecture and the subsystems; third, the construction of appropriate schemes for cataloging and retrieval and selection of components created during component development; and finally an implementation phase that involves component assembly.

We use design science principles, whose objective is to "develop technology-based solutions to important and relevant business problems" (Hevner, March, & Park, 2004, p. 83), as a lens to help formulate recommendations for enhancing quality in each life cycle phase. Prior to discussing each phase of the CBSD Dual Life Cycle model, we describe tenets of design science.

2. TENETS OF DESIGN SCIENCE

The design science paradigm "seeks to create innovations that define the ideas, practices, technical capabilities, and products which the analysis, design, implementation, and use of information systems can be effectively and efficiently accomplished," (Hevner et al., 2004, p. 76). Therefore, design science is an appropriate rubric by which to evaluate software development and identify recommendations.

Simon (1996) identified five elements of design theory:

(1) *The Evaluation of Designs* - selecting and applying appropriate theories of evaluation, such as utility theory, as well as suitable computation methods for either optimal or satisfactory alternatives.

(2) *The Formal Logic of Design* – using both imperative and declarative logic.

(3) *The Search for Alternatives* - including search methods such as means-ends analysis which is an example of a problem solving technique that exploits factoralization.

(4) *The Theory of Design Structure and Organization* – decomposing complex systems hierarchical systems into successively smaller functional systems so that the design of each subcomponent can be carried out with some degree of independence.

(5) *Representation of Design Problems* – formulating recommendations as a function of the design representation that makes some aspects evident aspects and others obscured.

We use these principles to help formulate recommendations for each life cycle phase as discussed below.

3. PHASES OF COMPONENT DEVELOPMENT

Domain Selection and Analysis

Domain selection and analysis involves the identification of problems that need to be addressed. Component manufacturers must select the domains for which they wish to develop commercially available components. An issue for component manufacturers is the potential available market for their components. This market is indispensable for the growth of CBSD since it will facilitate competition which ultimately leads to better products at lower prices.

It appears that there are an insufficient number of components available on the market to meet the demand, especially in the functional areas of businesses. Vendors sell many utility components such as components to resize windows, upload and transfer files, or compress files. General application components for email, spreadsheets, or calendaring also exist. Less prevalent are components that address requirements in businesses' functional areas, though finance and accounting appear to have more components than those in other areas.

From a design science perspective, the selection of domains component manufacturers choose for development can be approached with optimization techniques such as means-ends analysis. According to Simon, the problem is to "find an admissible set of values of command variables, compatible with the constraints that maximize the utility functions for the given value of environmental parameters" (Simon, 1996, p. 116). In this case, the utility function might be to optimize the potential sales of a component based upon a number of constraints and parameters (for example, mandatory software functions included in a given domain, the importance or weight of optional functions, etc.). Thus, we recommend:

Employ optimization techniques to determine the command variables and constraints to allow for creation of generic, domain-specific components with maximum utility for functional domains.

Design of the Component Architecture

Component-based systems are a type of hierarchical system that can be decomposed into component-based subassemblies, which can further be decomposed until an elementary level is reached. Each component has an "inner" environment, which includes its data structure, algorithms and controls, and an "outer" environment, or the setting in which it operates. In other words, a component consists of a software element and a well-defined interface. The interface, the junction at which the inner and outer environments meet, is concerned with ensuring that the required functionality is delivered to meet the desired design goals, but can succeed only if the inner environment appropriately adapts to the outer environment. The separation of the component's physical packaging and the interface represents a unique characteristic of the CBSD approach.

Important design factors include appropriate encapsulation, appropriate granularity, specificity, reusability, functional completeness, reliability, variability, adaptation, and clean interfaces. The later is especially important in terms of component adaptability. Often, whether a component achieves a particular goal or adapts appropriately to its environment is dependent upon only a few characteristics in the outer environment and not on the details of the environment. Including additional, but unnecessary constraints only serves to limit the number of outer environmental conditions within which the component can operate. Thus we recommend:

Design interfaces without extraneous constraints else the components will be restricted in terms of the environments in which they may be placed and the markets to which they may be available.

Component Fabrication

This phase entails building and testing the component via a selected programming language. In CBSD, the internal processes of a component are known only to the component developers. The external functionality is made known through a published description of the component. One of the key distinguishing factors in component-based development is the separation of the interface and its implementation (Hopkins, 2000). According to Jain et al. (2003, p. 49), interfaces "describe how a client program should interact with the component while hiding implementation details". Considering that an implementation of a component may have multiple interfaces, a task of the component developer is to ensure compatibility between the implementation and the interfaces.

Quality and performance are crucial factors that should be evaluated during component development. Metrics should be developed to anticipate the performance of component-based systems before they are actually built. The performance, quality, and behavior of a component are partially dependent on the external environment with which it must interact. Because CBSD differs from traditional approaches in that various combinations of independently built components are combined, it is crucial that models be created to assist in evaluating the way the distinct components will interact with each other. While the "inner environment" of a component may be well understood, due to the large number of possible permutations of external environments in which a commercially available component may be placed, techniques for discovering the implications of component attributes must be made. Essentially, the component must be tested without knowing how it will be used or the environment to which it will eventually be embedded.

One technique that sheds light on predicting behavior is simulation. Even if correct premises about a component's internal architecture are made, it may be very difficult to discover what they imply when placed in a variety of external environments. Therefore, simulation techniques are needed to work out the implications of the interactions of vast number of variables. Therefore we recommend:

Use simulations techniques for evaluating a component's architecture in a variety of external environments to identify potential quality and performance problems.

4. PHASES OF SYSTEM DEVELOPMENT

Figure 1 (see Appendix) shows the component and system development phases as separate life cycles. In CBSD, the phases of Component Development, discussed above, will often be conducted by a component manufacturer that commercially sells software components. The phases of System Development, discussed below, will typically be conducted by an organization that requires the overall system functionality.

Requirements Analysis

One of the commonly held reasons that many systems fail is the fact that requirements are not correctly identified in traditional approaches. An argument has been made that in comparison with these traditional approaches, CBSD can improve the requirements determination process because CBSD emphasizes more of an active role of the user in requirements determination and throughout the development life cycle. Initial requirements are gathered from users in the initial requirements gathering phase, but are further refined and elaborated as components are selected. Thus, we stress the cyclical nature of CBSD life cycles. When users and developers work together to identify components that satisfy given requirements, they further hone and prioritize requirements given the availability, or lack thereof, of components with functionality to meet those requirements. In addition, because commercially available components may have features that incorporate industry best practices within the domain, users and system developers may discover requirements they might not have otherwise identified.

We suggest that not only the development approach, but also its combination with the method of external problem representation used, will produce differing outcomes in requirements quality and completeness. A goal of this phase is to make the solution as clear as possible. One of the tenets of design science is that the way in which a problem is represented can make a significant difference, obscuring some aspects and illuminating others. Three-dimensional representations of problem spaces yield different insights than two dimensional maps or textual lists. In traditional development settings, research has been done investigating the effects of alternative problem representations and has shown that there are indeed variations (Umanath & Vessey, 1994). Therefore we recommend:

Use a variety of external problem representations to enhance the quality and completeness of requirements specified.

Design of the System and Subsystems Architectures

Design of the system and subsystems architectures involves the selection of the appropriate component model which allows the components to communicate with one another. Existing component models include Microsoft's Distributed Component Object Model (DCOM), Object Management Group's Common Object Request Broker Architecture (CORBA) and Enterprise Java Beans (EJB) from Sun Microsystems. More recent component models include Microsoft's ActiveX, .NET, and Sun's J2EE. Though it is true that a fundamental tenet of components is that they represent self-contained, selffunctioning units, the components are still required to "talk" to each other.

Although use of CBSD is growing, some believe that a comprehensive component model is still insufficient (Dahanayake, Sol, & Stojanovic, 2003). The fact that there are a number of proprietary models places limitations on the use of CBSD across platforms. For pure CBSD to exist, consideration of components beyond the implementation and deployment phases must be taken into account. Ultimately, components must become the central focus of the complete development process. A factor hindering this shift in focus is the fact that many of the component models are 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., p. 18). Only when components become the central focus will infrastructures be fully utilized for the development of complex systems. Thus, we recommend:

Select robust component models that have their central focus on components.

Component Selection, Cataloging, and Retrieval

Three important characteristics to take into consideration when selecting components are: (1) utility - whether the component is relevant to the problem domain; (2) capacity - the sophistication of the functionality of the component especially in terms of its reuse; and (3) versatility – whether it can be easily integrated into the desired system (Waguespack & Schiano, 2004). It has also been suggested that managerial goals should be mapped with technical features to ensure that components adequately meet system requirements. Managerial goals may include cost effectiveness, ease of assembly, customization, reusability, and maintainability. Ultimately, these goals assist in determining the most effective component design to delivery the greatest benefit.

With the growing number of components, locating and retrieving the appropriate software components in order to meet the system requirements becomes a complex task (Waguespack & Schiano, 2004). Approaches such as attribute value, hypertext, facet-based, semantic-based, information entropy, and keyword search strategies have been devised to address this problem. In spite of research emphasizing storage, retrieval, and reuse, the ability to store and retrieve components is still somewhat inadequate.

As the number of available software components grows, the design science concept of "satisficing" rather than optimizing may be more appropriate. The selection would be based on whether the component in question satisfies all the carefully specified design criteria. The time for the search would be therefore dependent on the standards for acceptability and little on the total number of components available for search whereas optimization techniques are also dependent on the total size of the universe being searched. We therefore we recommend:

Create effective search algorithms for components by specifying standards of acceptability and "satisficing."

Component Assembly

Component assembly consists of integrating components to build an application system. This entails the ability to demarcate the requirements into smaller subsets and to confirm the overall selection of the component set.

Thorough integration testing should be conducted and metrics should be developed to measure the effectiveness of the assembled components. Because of the iterative nature of the system development process, there is a constant cycle of select/assemble/test. Through this iterative process, subassemblies are constructed from components in a hierarchical fashion. Leveraging stable subassemblies allows a designer to further use the power of hierarchical structures when building component-based systems. It allows the designer to potentially reduce the current problem to a previously solved problem, then identify what steps must be taken to reach the new solution. Thus we recommend:

Use stable subassemblies to increase flexibility in the creation of component-based systems.

5. CONCLUSION

CBSD echoes the objective of design science which is to create technology-based solutions for significant business problems. The

CBSD Dual Life Cycle model divides the development process into (1) component development, often conducted by commercial component manufacturers and (2) system development, typically conducted by organizations that will use resulting software systems. Our design science-based recommendations for enhancing quality for both categories of development are summarized in Table 1 and Table 2 (see Appendix).

Overall, CBSD has the potential to significantly alter how information systems are developed. Design science can greatly inform these processes as has been shown by the creation of the proposed CBSD Dual Life Cycle model and associated recommendations.

6. REFERENCES

Capretz, Luiz Fernando (2005) "A New Component-Based Software Life Cycle Model." *Journal of Computer Science* (1:1), pp. 76-82.

Crnkovic, Ivica, Stig Larsson and Michel Chaudron (2005) "Component-Based Development Process and Component Lifecycle." *Journal of Computing and Information Technology* (13:4), pp. 321-327.

Dahanayake, Ajantha, Henk Sol and ZoranStojanovic (2003) "Methodology Evaluation Framework for Component-based System Development." *Journal of Database Management* (14:1), pp. 1-26.

Elfatatry, Ahmed (2007) "Dealing With Change: Components Versus Services." *Communications of the ACM* (50:8), pp. 35-39.

Fahmi, Syed Ahsan and Ho-Jin Choi (2008) "Life Cycles for Component-Based Software Development." Proceedings of IEEE 8th International Conference on Computer and Information Technology Workshops*,* July 8-11, pp. 637-642.

Fujii, Keita and Tatsuya Suda (2006) "Semantics-Based Dynamic Web Service Composition." *International Journal of Cooperative Information Systems* (15:3), pp. 293-324.

Hevner, Alan R., Salvatore T. March and Jinsoo Park (2004) "Design science in Information Systems Research." *MIS Quarterly* (28:1), pp 75-105.

Hopkins, Jon (2000) "Component Primer." *Communication of the ACM* (43:10), pp. 27-30.

Jain, Hemant, Padmal Vitharana, P. and Fatemah "Mariam" Zahedi (2003) "An Assessment model for Requirements Identification in Component-based Software Development." *Database for Advances in Information Systems* (34:4), pp. 48-63.

Sommerville, Ian (2004) Software Engineering, 7th Edition. Addison Wesley.

Simon, H.A. (1996) The Sciences of the Artificial. The MIT Press.

Umanath, Narayan S. and Iris Vessey (1994) "Multiattribute Data Presentation and Human Judgment: A Cognitive Fit Perspective." *Decision Sciences* (25:5-6), pp. 795-824.

Vitharana, Padmal and Hemant Jain (2000) "Research Issues in Testing Business Components." *Information & Management* (37:6), pp. 297-309.

Vitharana, Padmal, Fatemah "Mariam" Zahedi and Hemant Jain (2003) "Design, Retrieval, and Assembly in Component-based Software Development." *Communications of the ACM* (46:11), pp. 97-102.

Waguespack, Leslie and William T. Schiano "Component-based IS architecture." *Information Systems Management* (21:3), pp. 53-60.

Yang, Jian (2003) "Web Service Componentization." *Communications of the ACM* (46:10), pp. 35-40.

Appendix

Figure 1. CBSD Dual Life Cycle Model

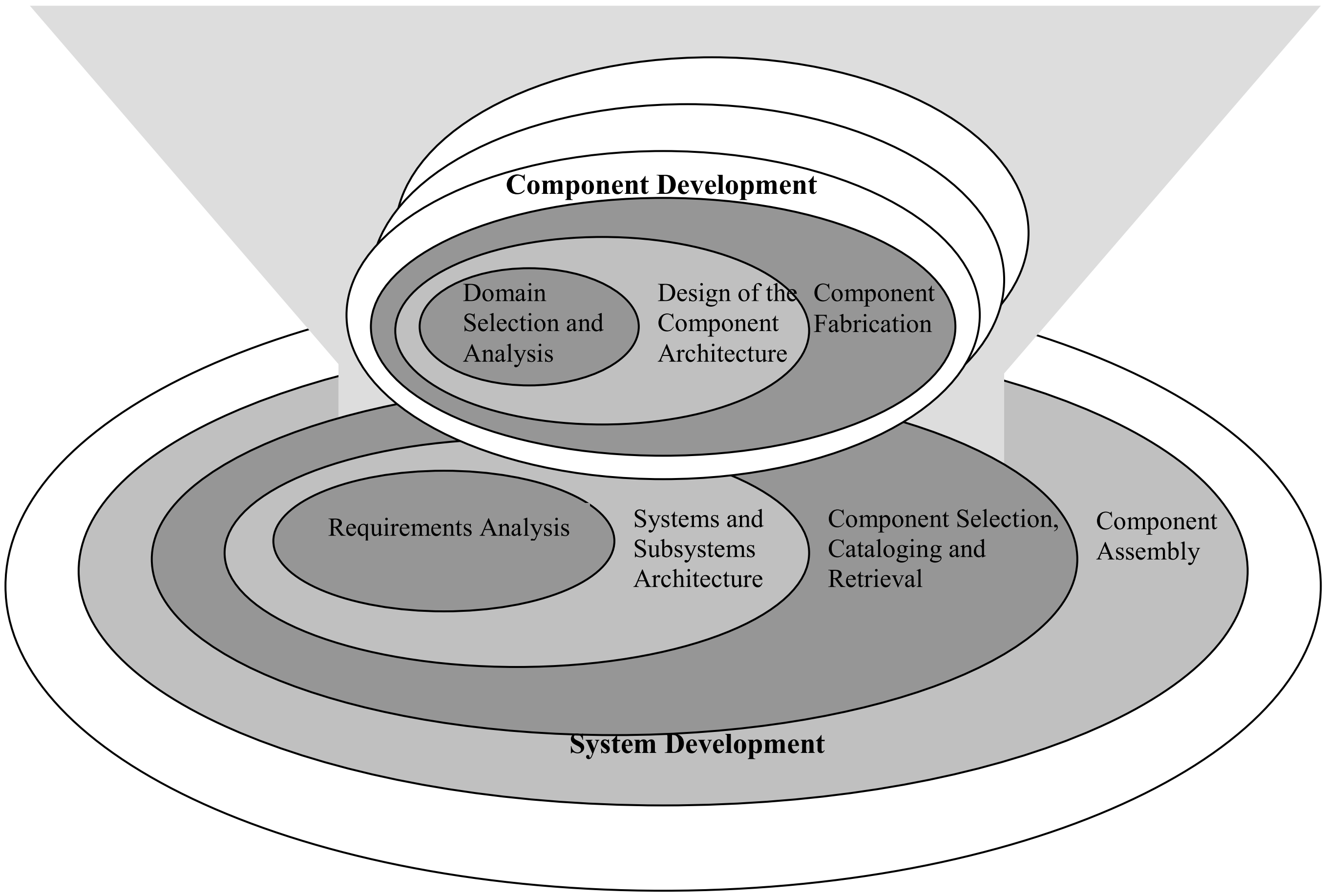
Component Development
Domain Selection and Analysis
Design of the Component Architecture
Component Fabrication
Requirements Analysis
Systems and Subsystems Architecture
Component Selection, Cataloging and Retrieval
Component Assembly
System Development

Table 1. Design Science Based Recommendations for Enhancing Quality in Component Development

Phase of Component Development	Design Science Element	Recommendation
Domain Selection and Analysis	The Search for Alternatives	Employ optimization techniques to determine the command variables and constraints to allow for creation of generic, domain-specific components with maximum utility for functional domains.
Design of Component Architecture	Theory of Structure and Design Organization	Design interfaces without extraneous constraints else the components will be restricted in terms of the environments in which they may be placed and the markets to which they may be available.
Component Fabrication	The Formal Logic of Design	Use simulations techniques for evaluating a component's architecture in a variety of external environments to identify potential quality and performance problems.

Table 2. Design Science Based Recommendations for Enhancing Quality in System Development

Phase of System Development	Design Science Element	Recommendation
Requirements Analysis	Representation of Design Problems	Use a variety of external problem representations to enhance the quality and completeness of requirements specified.
Systems and Subsystems Architecture	Theory of Structure and Design Organizations	Select robust component models that have their central focus on components.
Component Cataloging and Retrieval	The Search for Alternatives	Create effective search algorithms for components by specifying standards of acceptability and "satisficing."
Component Assembly	Theory of Structure and Design Organizations	Use stable subassemblies to increase flexibility in the creation of component-based systems.