By Padmal Vitharana

Risks and Challenges of Component-Based Software Development

Component developers, application assemblers, and customers must all know CBSD advantages and disadvantages before developing components and component-based applications.

As more and more facets of the business enterprise become ingrained in computer-based application systems, pressure increases on software developers to construct quality systems quickly. Systems are no longer built from scratch using archaic software development life-cycle methodologies. Following the success of the structured design and OO paradigms, Component-Based Software Development (CBSD) has emerged as the next revolution in software development.

A component is generally defined as a piece of executable software with a published interface [5]. Following the manufacturing principle of part fabrication and assembly, CBSD proponents argue that components provide significant quality and productivity gains to the software industry similar to those realized in the computer hardware industry [3]. Practitioners identify the following key CBSD advantages in future software development efforts:

Reduced lead time. Building complete business applications from an existing pool of components;

Leveraged costs developing individual components. Reusing them in multiple applications;

Enhanced quality. Components are reused and tested in many different applications; and

Maintenance of component-based applications. Easy replacement of obsolete components with new enhanced ones.

CBSD also involves significant challenges to constructing software [1, 7]. Although there's plenty of anecdotal evidence of the success of component-based systems, the literature does not clearly identify the risks

for the various CBSD stakeholders. Here, I identify key CBSD risks and challenges encountered by each category of stakeholder. My aim is to provide insight for software project managers concerning the new paradigm before they embark on critical component and CBSD projects.

Stakeholders

CBSD encompasses three primary types of stakeholders (see Figure 1) [2]:

Component developers. Included are freelance developers, IS departments, and entire organizations specializing in component fabrication;

Application assemblers. They locate suitable components and assemble them in integrated application systems that satisfy customer requirements; and

Customers. They employ component-based application systems to perform business tasks. I assume the interests of the people paying for the system and the people using the system coincide; hence I do not attempt to distinguish between them here.

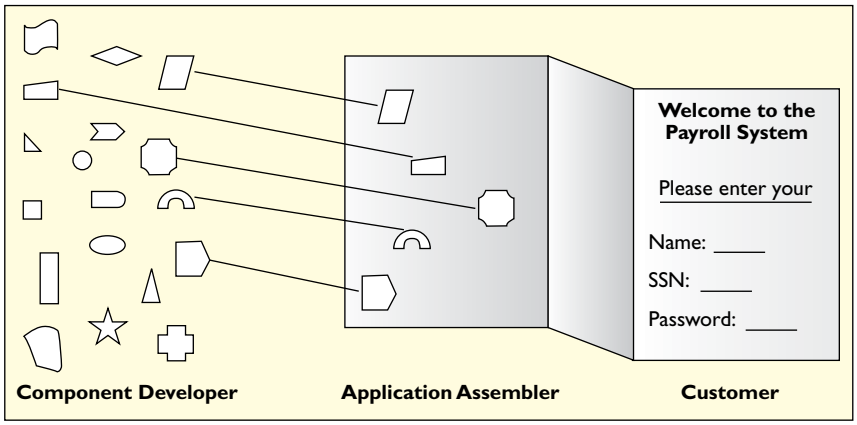


Figure 1. Stakeholders of the CBSD paradigm.

A particular firm might function in two or even all three stakeholder capacities; for example, an IS department could develop components and assemble applications to be used by the firm's users.

Developers. A developer encounters certain risks and challenges in developing components, managing component-development projects, and subsequently marketing the components. Developers generally fabricate components under two scenarios: survey the overall software industry and build components for the mass market; or develop components for a specific client or assembler. In pursuing a mass-market strategy, developers must identify business areas or domains that would generate enough yield to justify component development. But deriving a set of components encompassing the entire domain knowledge and its related business processes is a challenge [3]. As domains are rarely constant over time, developers must adapt to changes in the domain once the components are fabricated. After investing considerable resources, a developer risks having its component repositories become obsolete due to poor planning or unfavorable industry trends. Moreover, if a developer carries legacy assets, leveraging them by adding component "wrappers" around them might require considerable additional effort.

Developers must also determine the optimal way to fragment the domain into a cohesive set of components. Because these components contain the domain knowledge, the number of components, their granularity, or size, and their dependencies play a vital role in developing component-based applications [1, 8]. Developers face additional challenges in verifying that the intended domain "knowledge" and "business processes" in fact map to the components being developed. In light of the CBSD emphasis on interfaces, deriving well-defined interfaces specifying how components work, along with their inputs, outputs, and exception-handling procedures, also poses a considerable challenge. Conventional methods provide little guidance in tackling these issues.

Due to the relative recency of CBSD compared to life-cycle methodologies, formal approaches and tools for component development are still emerging [2]. Although successful methods (such as the Unified Approach from Rational Software, www.rational.com) have been reported, developers face a daunting task in choosing suitable methodologies and tools for constructing components. As CBSD tools require enormous initial investment, their selection involves considerable thought. Moreover, adoption of appropriate middleware (such as CORBA and DCOM) needs to be aligned with the requirements of either the client or the mass market.

In managing component-development projects, a developer must monitor each component from inception to delivery. Because multiple versions of a component may be developed (following bug fixes and enhancements), versioning protocols take on added significance, as components used in multiple applications need to be coordinated. Developers must also

assess how often to release components and how to inform clients, or assemblers, of new versions. The emerging literature on open-source software might facilitate some aspects of CBSD project management, including version control.

Developing quality components requires a comprehensive testing program [9]. Almost like an individual application, though on a smaller scale, each component must undergo verification and validation testing throughout its development process. However, unlike traditional applications, individual components can be used by many assemblers for myriad uses in multiple applications [8]. This complicates the testing process, as the developer is forced to unit-test each component without knowing exactly how it will be used or who will ultimately use it.

As an emerging paradigm, CBSD requires a new suite of metrics. Conventional measures of critical indicators (such as reliability and productivity) may not necessarily work well in the new paradigm. CBSD also demands new personnel at both the technical and the managerial levels [10]. Designing components requires unique skills necessitating top management apportion significant resources to retrain current personnel and/or hire new personnel.

Before embarking on component-development projects, a developer must conduct cost-benefit analyses to determine whether to accept a client, or assembler, and contract or construct components for the mass market [10]. The cost of component construction encompasses domain analysis, as well as identifying, developing, and testing components. The key benefit of component development is the sale of the components; in the case of the mass market it's to multiple clients for multiple uses; in the case of specific clients, development costs must be estimated, so the amount to charge the client is assessed. Though component-specific cost models are not widely available, traditional cost models, especially those derived from the OO paradigm might be adopted. In the mass-market scenario, developers face additional uncertainties as to future market demand, price, and competition—all difficult to ascertain during the early stages of component development. Moreover, the possible interplay among them produces additional complexity in assessing the costs and benefits of developing components for the mass market.

In the part-fabrication-assembly paradigm, the quality of the parts directly influences the quality of the assembled whole. Hence, a developer must assume responsibility for the quality of the components it constructs. As more components become available from more and more vendors, certifying the components and possibly the developers, too, becomes crucial for establishing a sense of trust in the component market [7]. Today, online component seller www.flashline.com offers to certify components and makes relevant documents available to potential customers hoping to boost their confidence. As the component market expands to include third-party sellers and online auctioneers, assuring component authenticity is vital to warding off unscrupulous parties from offering counterfeit products. Moreover, appropriate controls must be in place to prevent imitators from reverse engineering and subsequently reengineering the developer's component code. Developers must also address security issues to alleviate client concerns about possible hacker-prone, corrupt, or virus-infected components.

AFTER INVESTING CONSIDERABLE RESOURCES, developers risk having their component repositories become obsolete due to poor planning or unfavorable industry trends.

In tailoring their components for the mass market, developers often rely on other parties to sell them. Online marketers, including www.componentsource.com and www.flashline.com, offer search mechanisms for retrieving components. However, only limited research is available on effective component classification, search, and retrieval. An effective classification and coding scheme and corresponding retrieval system are essential for enabling component seekers to locate needed components [2]. Nevertheless, due to the availability of a multitude of disparate component repositories, developers frequently grapple with the choice of a suitable online marketer to promote their products.

Even though third parties market components, developers still face other marketing issues. As components involve numerous payment methods, including outright purchase, pay-per-use, and time-expired use, possibly limiting usage to a year, each developer must determine which pricing option optimizes its profits. Developers and third-party marketers must

devise suitable licensing contracts that document "appropriate" usage and developer liability in case components do not work according to the stated specification. In the case of specific clients, the developer and the client, or assembler, should formally agree who owns the components and other project artifacts. In the mass-market scenario, component developers encounter some of the same intellectual property issues as developers of conventional applications [4].

Assemblers. Assembler risks and challenges primarily concern the assembly of components in applications, the management of component-based application assembly projects, and the uncertainties of the component market.

Because the assemblers' component search is likely to be iterative, involving both broad and narrow searches, its ability to find needed components depends on the classification and retrieval mechanisms provided by online marketers. Like developers, assemblers work with an array of disparate component repositories to identify and locate needed components. Even when components are found, they might not perform the specified functionality or fail to interoperate with one another, thus requiring some fine-tuning by the assembler.

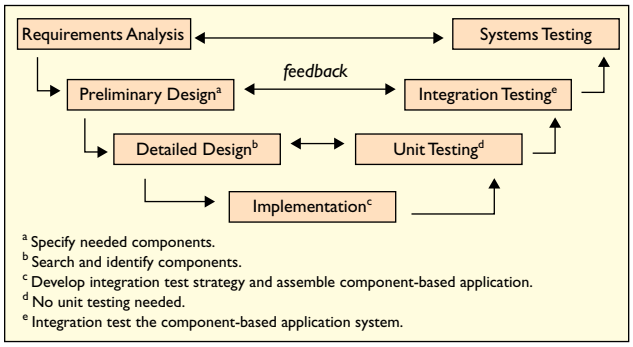


Figure 2. Component-based application assembly life cycle.

More often, assemblers can't find at least some of the components they need on the open market [1]. They are thus compelled to either have a developer construct custom components or request their customers adjust or revise their requirements to correspond with the components that are actually available. In such a scenario, the assembler, in concert with the customer, conducts a thorough cost-benefit analysis to determine whether to custom-build components, incurring greater cost, though specified requirements are identically matched, or to adjust requirements in order to acquire components at the low mass-market price. Other crucial challenges in assembling component-based applications that match user requirements include:

Matching system requirement specifications. They are typically documented in formal languages (such as VDM and Z) with components in the repository typically coded in some proprietary classification scheme;

Demarcating the requirements document into smaller subsets. Based on the availability of components in the open market, demarcating the requirements into smaller subsets is a tedious process involving many iterations; and

Confirming the overall selected component set. Does it satisfy all the requirements of the planned application system?

Although developers are expected to profess that components are unit-tested and bug-proof, whenever an assembler employs a component from a less-reputable source, it has difficulty gauging the quality and reliability of component-based applications. Assemblers might thus be forced to conduct additional unit testing. Nevertheless, component-based applications must undergo thorough integration testing. As components selected for particular applications are likely to come from multiple sources, assemblers must devise comprehensive test suites to ensure these components work in unison [9]. Because assemblers might use components in ways not envisioned by their fabricators, integration testing thus plays a crucial role in application success.

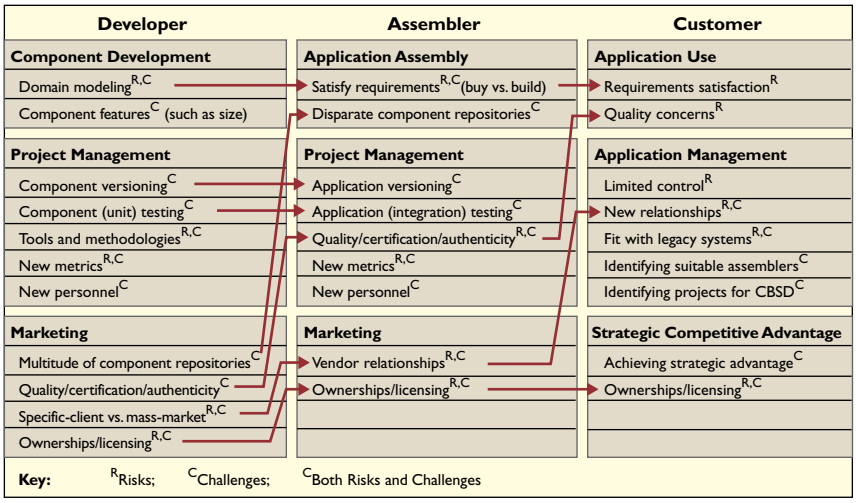
Even though the application assembly life cycle resembles the traditional systems life cycle, corresponding processes involve markedly different tasks (see Figure 2). For example, preliminary design entails component specification, while detailed design consists of component search and identification. For each application developed, assemblers need to track all the components used, along with their version information. As assemblers themselves often release multiple versions of an application, it's crucial that they track component versions used in applications.

Component-based application assembly requires personnel with a different set of skills from those employed in the traditional life-cycle methodologies. Analysts need additional expertise in matching solicited user requirements with components available in the repository before assembling them into applications. Assemblers also need newer sets of metrics to assess the effectiveness of the component assembly process and its outcomes. Advanced tool support is also needed because, as practiced today, component assembly demands considerable effort; reliable visual tools allowing easy assembly of components on assem-

blers' desktops in minutes have not fully emerged.

Assemblers must also deal with the lack of visibility into the component-development process. An assembled application will exhibit the same shortcomings as those of the components used to build it. Some customers might demand assemblers provide details of the components used to build their applications. If industrywide standards for certifying components do not emerge, assemblers might be inclined to work with only a few select developers, thereby negating the benefits of open-market component trading.

Moreover, relying on external developers for components places assemblers at significant risk due to the limited control they have over the type of components available, the release schedule for subsequent versions, and the necessary assurance that new versions are compatible with the old ones [7]. Being responsive to customers requires assemblers to fix errors promptly and enhance application systems as needed. Having to depend on a developer offering components in the mass market (and with whom the assembler might not have a long-term relationship) presents clear risks.



Key CBSD stakeholder risks and challenges and their cascading effects.

Customers. Customers face both risks and challenges in using component-based applications to meet their enterprise requirements, as well as in managing their component-based and legacy application systems and in achieving and sustaining strategic competitive advantage over their rivals.

One key risk involves whether a system is actually capable of satisfying customer requirements. Due to the market's lack of suitable components, assemblers might deliver systems that do not completely satisfy customer requirements. Customers face additional risks, as application quality (or lack of quality) based on component quality hinders their ability to carry out key business activities.

Each customer faces risks in managing its repertoire of component-based applications. For example, when software does not function as intended, developers and assemblers might blame each other as to what went wrong. As a result, customers have limited control over when and if bugs are fixed and enhancements are made [7]. Moreover, as customers often have a set of legacy applications critical to running their operations, they risk new component-based applications not interoperating with their existing legacy software.

Customers also face tough decisions in identifying, evaluating, and using assemblers to build new applications. For customers, the ability to influence assemblers in their selection of developers and components would be advantageous. With the ability to influence the component market, customers could persuade developers to incorporate their business practices into the components being built.

Finally, moving from traditional to component-based application development, customers must carefully assess which projects are more suitable for CBSD. A low-risk strategy might be to first pursue component-based solutions for routine and nonstrategic software projects.

Employing component-based enterprise solutions, customers face considerable challenges trying to achieve strategic advantage through their component-based systems. If a customer's information systems use prefabricated commodity components available on the open market, wouldn't its ability to achieve and sustain competitive advantage from its information systems be hindered? Achieving competitive advantage might depend on its ability to mix-and-match components available in the open market with those that are custom-built, either in-house or on a contract basis, as in conventional outsourcing arrangements.

As assemblers increasingly depend on components developed by developers and customers depend on applications assembled by assemblers, these risks and challenges propagate from one stakeholder to another. The table here outlines the key risks and challenges facing each stakeholder, as well as the cascading effect of risks/challenges among all stakeholders; for instance, component versioning by developers affects

assemblers' application versioning tasks. Consequently, risks faced by one stakeholder are transferred to the next, ultimately constraining each customer's ability to leverage component technology in developing its application systems.

Conclusion

Since 1997 when Bill Gates said, "It has been a long time in coming, but the industrial revolution of software is finally upon us," the number of components and component-based applications has skyrocketed. While CBSD helps overcome inadequacies in traditional development, it also poses risks to the profitability and even long-term survival of each of its stakeholders. From uncertainties in leveraging existing legacy code to the inability to find needed components, they confront challenges in constructing component solutions that address their evolving enterprise requirements. Hence, before embarking on component-based development projects, each stakeholder must assess its risks and devise sound strategies to address them. C

References

1. Brereton, P. and Budgen, D. Component-based systems: A classification of issues. *Comput. 33,* 11 (Nov. 2000), 54–62.
2. Brown, A. *Large-scale Component-based Development.* Prentice Hall, Upper Saddle River, NJ, 2000.
3. Brown, A. From component infrastructure to component-based development. In *Proceedings of the International Workshop on Component-Based Software Engineering* (Kyoto, Japan, 1998).
4. Chavez, A., Tornabene, C., and Wiederhold, G. Software component licensing: A primer. *IEEE Software* (Sept./Oct. 1998), 47–53.
5. Hopkins, J. Component primer. *Commun. ACM 43,* 10 (Oct. 2000), 27–30.
6. Levitt, J. One-stop software component shop. *InformationWeek* (Oct. 28, 2000), 146.
7. Stafford, J. and Wallnau, K. Is third-party certification necessary? In *Proceedings of the International Workshop on Component-Based Software Engineering* (Toronto, Canada, 2001).
8. Szyperski, C. *Component Software: Beyond Object-Oriented Programming*. ACM Press, New York, 1998.
9. Weyuker, E. Testing component-based software: A cautionary tale. *IEEE Software* (Sept./Oct. 1998), 54–59.
10. Woodhouse, C. Principles of adopting component-based software engineering. In *Proceedings of the International Workshop on Component-Based Software Engineering* (Los Angeles, 1999).

Padmal Vitharana (padmal@syr.edu) is an assistant professor of information systems in the School of Management at Syracuse University.

Michael Sparling of Castek Corp. contributed to this article.