Institute of Information Technology, University of Dhaka

SE 406 Software Requirements Specification and Analysis

Report on Component-based development (CBD)

Group 1

Members:
Abhijit Paul (BSSE 1201)
Sazzad Hossain (BSSE 1207)
Rifah Tasfia Faria (BSSE 1213)
Sampad Sikder (BSSE 1219)
Nazmus Sakib (BSSE 1225)
Fahim Mahmud (BSSE 1231)

What is CBD [1]?

Component-based development (CBD) is a procedure that accentuates the design and development of computer-based systems with the help of reusable software components. With CBD, the focus shifts from software programming to software system composing.

What is Component?

Component is a distributable unit of software. These components can be designed as either conventional software modules or object-oriented classes or packages of classes. They provide targeted functionality with well-defined interfaces that enable the component to be integrated into the software that is to be built.

Properties of a component:

A component must be-

1. Reusable
2. Accessible
3. Upgradable/Should remain up-to-date with the latest technologies and patches.
4. Well-defined properties/possible documentation of features.
5. Ease of integration

History[2]:

The idea that software should be componentized - built from prefabricated components - first became prominent with Douglas McIlroy's address at the NATO conference on software engineering in Garmisch, Germany, 1968, titled Mass Produced Software Components. The conference set out to counter the so-called **software crisis.** McIlroy's subsequent inclusion of pipes and filters into the Unix operating system was the first implementation of an infrastructure for this idea.
Brad Cox of Stepstone largely defined the modern concept of a software component. He called them Software ICs and set out to create an infrastructure and market for these components by inventing the Objective-C programming language. (He summarizes this view in his book Object-Oriented Programming - An Evolutionary Approach 1986.)

Development Phases:

The existence of reusable components does not guarantee that these components can be integrated easily or effectively into the architecture chosen for a new application. It is for this reason that a sequence of component-based development actions is applied when a component is proposed for use.
The component-based development model incorporates the following steps (implemented using an evolutionary approach):

1. **Component Qualification:**

Component qualification ensures that a candidate component will perform the function required, will properly "fit" into the architectural style specified for the system, and will exhibit the quality characteristics (e.g., performance, reliability, usability) that are required for the application.

Among the many factors considered during component qualification are:

- Application programming interface (API).
- Development and integration tools required by the component.
- Run-time requirements, including resource usage (e.g., memory or storage), timing or speed, and network protocol.
- Service requirements, including operating system interfaces and support from other components.
- Security features, including access controls and authentication protocol.
- Exception handling.

2. Component Adaptation.

In an ideal setting, we will get components that can be easily integrated into an application architecture. In reality, however, even after a component has been qualified for use within an application architecture, conflicts may occur in one or more of the areas just noted.

To avoid these conflicts, an adaptation technique called **component wrapping** is sometimes used. It can be of three types.

1. White-box wrapping : When a software team has full access to the internal design and code for a component (often not the case unless open-source COTS components are used), white-box wrapping is applied. Like its counterpart in software testing ,white-box wrapping examines the internal processing details of the component and makes code-level modifications to remove any conflict.
2. Gray-box wrapping is applied when the component library provides a component extension language or API that enables conflicts to be removed or masked.
3. Black-box wrapping requires the introduction of pre- and postprocessing at the component interface to remove or mask conflicts. You must determine whether the effort required to adequately wrap a component is justified or whether a custom component (designed to eliminate the conflicts encountered) should be engineered instead. (e.g. pytesseract)

3. Component Composition:

The component composition task assembles qualified, adapted, and engineered components to populate the architecture established for an application. To accomplish this, an infrastructure must be established to bind the components into an operational system. The infrastructure (usually a library of specialized components) provides a model for the coordination of components and specific services that enable components to coordinate with one another and perform common tasks. Because the potential impact of reuse and CBSE on the software industry is enormous, a number of major companies and industry consortia have proposed standards for component software.

OMG/CORBA: The Object Management Group has published a common object request broker architecture (OMG/CORBA). An object request broker (ORB) provides a variety of services that enable

reusable components (objects) to communicate with other components, regardless of their location within a system.

Microsoft COM and .NET: Microsoft has developed a component object model (COM) that provides a specification for using components produced by various vendors within a single application running under the Windows operating system. From the point of view of the application, "the focus is not on how [COM objects are] implemented, only on the fact that the object has an interface that it registers with the system, and that it uses the component system to communicate with other COM objects". The Microsoft .NET framework encompasses COM and provides a reusable class library that covers a wide array of application domains.

Sun JavaBeans Components: The JavaBeans component system is a portable, platform-independent CBSE infrastructure developed using the Java programming language. The JavaBeans component system encompasses a set of tools, called the Bean Development Kit (BDK), that allows developers to

(1) analyze how existing Beans (components) work,
(2) customize their behavior and appearance,
(3) establish mechanisms for coordination and communication,
(4) develop custom Beans for use in a specific application, and
(5) test and evaluate Bean behavior.

None of these standards dominate the industry. Although many developers have standardized on one, it is likely that large software organizations may choose to use a standard based on the application categories and platforms that are chosen.

Types of Component Composition:

While integrating the components, one of the following composition procedures can be followed:

1. Sequential Composition: Components are integrated sequentially according to the requirements.
2. Hierarchical Composition: A component can call another component to form a hierarchical relationship.
3. Additive Composition: Multiple components are added together such that they form a new component.

Industry Practices:

Commercial off-the-shelf (COTS) software components, developed by vendors who offer them as products.

- Already in use to a certain extent (e.g. codecanyon.com)
- Enormous potential so many frameworks are already in use
- Follows industry standards.
- COTS development is considered a **future** we are eventually going towards.

Advantages:

1. Reusability

2. Testability
3. Efficiency
4. Consistency
5. Maintainability
6. Faster Development & More Profit

Disadvantages:

1. Difficult to determine the component qualification of commercial off-the-shelf (COTS) or third-party component
2. Component trustworthiness: How can we trust a component if there is no source code available?
3. Component certification: Who will certify the components for its quality?
4. Disadvantages of Spiral model
5. Architecture Design Dilemma

Challenges:

But a number of questions arise.

1. Is it possible to construct complex systems by assembling them from a catalog of reusable software components?
2. Can this be accomplished in a cost- and time-effective manner?
3. Can appropriate incentives be established to encourage software engineers to reuse rather than reinvent?
4. Is management willing to incur the added expense associated with creating reusable software components?
5. Can the library of components necessary to accomplish reuse be created in a way that makes it accessible to those who need it?
6. Can components that do exist be found by those who need them?

Increasingly, the answer to each of these questions is “yes.”

References:

1: https://www.techopedia.com/definition/31002/component-based-development-cbd
2: https://en.wikipedia.org/wiki/Pipeline_(Unix)