ISSN:2249-5789

Study of Component Based Software Engineering

Ishita Verma
House No.4, Village Dayalpur Karawal Nagar Road
Delhi-110094, India
ish.v.16@gmail.com

Abstract

Component based software engineering is an approach of software development that emphasizes the design and construction of computer based systems using reusable software components. A set of pre-built, standardized software components are made available to fit a specific architectural style for some application domain. Component Based Software Engineering encompasses two parallel engineering activities: domain engineering and component based development. This paper studies about Components, Component based development, its advantages, disadvantages and comparison of Component Based Software engineering with Conventional Software Engineering.

General Terms

Component Based Software Engineering

Keywords

Component, reuse, Component based development, Component based software engineering.

1. Introduction

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A Software Component can be deployed independently and is subject to composition by a third party. Component Based Software Engineering (CBSE) is an approach that relies on the reuse of Software components. A set of pre-built, standardized software components are made available to fit a specific architecture for some application domain. The application is then assembled using these components, rather than the discrete parts of a conventional programming language. Component based systems are easier to assemble and therefore less costly to build than systems constructed from discrete components. A Component is an independent executable entity. It does not have to be compiled before it is used with other components. The services offered by a component are made available through an interface and all the interactions take place through that interface. Advantages of Component based development are increased quality, accelerated development and reduced risk.

2. Component Interfaces

The interfaces of a Component, as in Fig 1, are as follows:

Provides Interface: defines the services that are provided by the component to other components.

Requires Interface: defines the services that must be made available to the component to execute as specified.

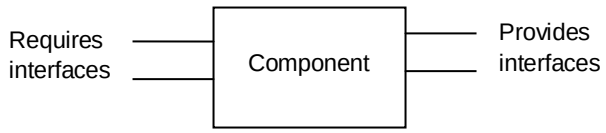

Fig 1: Component and its interfaces

The principles which govern Component Based Software Engineering design is as follows:

1) Components are independent so do not interfere with each other.
2) Component implementation is hidden.
3) Communication is through well-defined interfaces.
4) Component platforms are shared and reduce development costs.

3. Component Based Software Systems

Component Based Software Engineering (CBSE) is an approach to software development that relies on reuse. The process begins when a software team establishes requirements for the system to be built. An architectural design is prepared but rather than moving immediately into more detailed design tasks, the team examines requirements to determine what subset is directly amenable to composition rather than construction. That is, the team asks the following questions for each system requirement:

1.) Are commercial off-the–shelf (COTS) components available to implement the requirement?
2.) Are internally developed reusable components available to implement the requirement?
3.) Are the interfaces for available components compatible within the architecture of the system to be built?

The team attempts to modify or remove those system requirements that cannot be implemented with COTS or in house components. If the requirement(s) cannot be changed or deleted, software engineering methods are applied to build those new components to meet requirement(s). For those requirements that can be addressed with available components the following activities take place:

1) Component qualification
2) Component adaptation
3) Component composition
4) System evolution and component upgrade

3.1 Component Qualification

Qualification is the process of determining the suitability of a component for use within the intended final system. 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 that are required for the application. The interface description provides useful information about the operation and use of a software component, but it does not provide all the information needed to determine if a proposed component can be reused effectively in a new application.

3.2 Component Adaptation

Even after a component has qualified for use within application architecture, there may be some gaps between the component features and the requirements as a result of which some conflicts may occur. The aim of Adaptation is to ensure that the conflicts among components are minimized.

An Adaptation technique called as Component Wrapping is often used. When a software team has full access to the internal design and code for a component then code level modifications can be done to remove a conflict. This is called as White box wrapping. Gray box wrapping is used when the component library provides an API that enables conflicts to be removed. Black box wrapping introduces Pre and Post-processing at the component interface to remove conflicts.

3.3 Component Composition

The component assembly task assembles qualified, adapted and engineered components to populate the architecture established or an application. To accomplish this, an infrastructure must be established to bind the components into an operational system.

3.4 System Evolution and Component Upgrade

Because components are the units of change, system evolution is based around the replacing of outdated components by new ones. The components are treated as plug replaceable unite. But in practice this replacement may not be trivial as there can be a mismatch between the old and the new component thus triggering another stage of component adaptation.

4. Component Based Software Engineering Process

Component Based software Engineering, as in Fig 2, encompasses two parallel engineering activities: Domain Engineering and Component Based development. Also there is the process of Component Engineering which is used for constructing new software components for those requirements for which no suitable pre-built software components exist.

4.1 Domain Engineering

Domain Engineering explores an application domain with the intent of finding functional, behavioral and data components that are candidates for reuse. These components are placed in Reuse libraries. The intent of Domain Engineering is to identify, construct, catalog and disseminate a set of software components that have applicability to existing and future software in a particular application domain. Domain Engineering includes three major activities: Analysis, Construction and Dissemination.

4.2 Component Based Development

Component based development elicits requirements from the customer, selects an appropriate architectural design to meet the objectives of the system to be built and then

(1) Selects potential components for reuse
(2) Qualifies the components to be sure that they properly fit the architecture of the system,
(3) Adapts components, if modifications must be made to properly integrate them and,
(4) Integrates the components to form subsystems and the application as a whole.

In addition, custom components are engineered to address those aspects of the system that cannot be implemented using existing components. This activity takes place in parallel with Domain engineering. Using analysis and architectural design methods, the software team refines an architectural style that is appropriate for the analysis model created for the application to be built. Once the architecture has been established, it must be populated by components that are available from reuse libraries and/or are engineered to meet custom needs. When Reusable components are available for potential integration into the architecture, they must be qualified and adapted. When new components are required, they must be engineered. The resultant components are then integrated (composed) into the architecture and tested thoroughly.

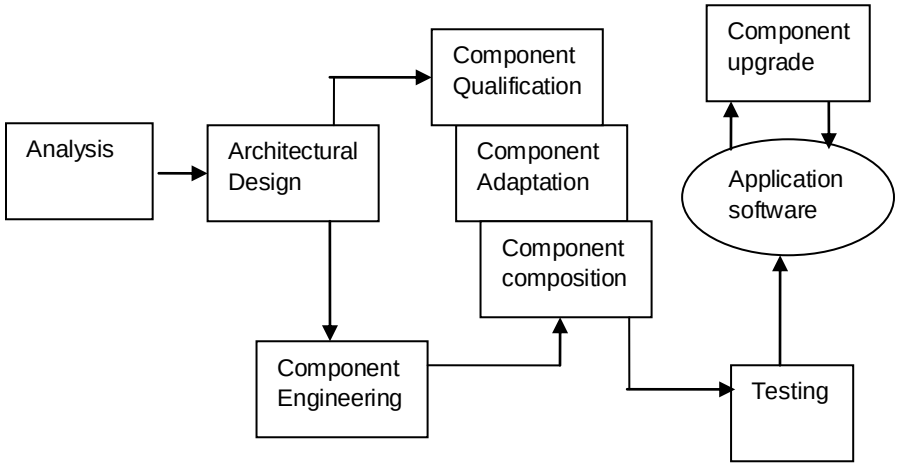

Fig. 2: Component Based Software Engineering process model

5. Comparison of Component Based Software Engineering with Traditional Software Engineering

Component based software engineering is advantageous than Traditional software engineering (see Table 1)

Table 1: Comparison of CBSE with Traditional Software engineering

Attribute	Component Based Software Engineering	Traditional Software Engineering
Cost	It involves reuse of software components in building software thus reducing the cost.	There is no reuse of components and hence no reduced costs.
Development Time	Because of the use of pre-built software components the development time is reduced.	No pre-built components are used and hence no reduction in the development time.
Quality	The software components which are used for building the software, exhibit quality characteristics like performance, reliability and usability thus increasing quality.	No reusable components are used here and hence no increased quality is achieved.
Applicability	This approach of software development is applicable only to software with pre-built software components.	There is no such restriction with traditional software engineering.

6. Comparison of Component Based Software Engineering with Object Oriented Software Engineering

Object oriented software engineering is another approach to software development in which the real world things are modeled as objects which have certain attributes and methods which affect those attributes. A collection of objects of similar type is called a class. An object may seem to be similar to a component but object is much more specific (see Table 2) and hence its use is limited.

Table 2: Comparison of CBSE with Object oriented software engineering

Component Based Software Engineering	Object Oriented software Engineering
Components are not too specific and provide some degree of abstraction and hence can be reused in software development.	Classes and Objects are too specific and hence are difficult to be reused in software development.
Object oriented concepts like Inheritance, Encapsulation and polymorphism cannot be used in software development.	Object oriented concepts of Inheritance, polymorphism and Encapsulation are used in software development.

7. Disadvantages of Component Based Software Development

The disadvantages of Component Based Software Development are as follows:

1) Finding suitable components which fit the architectural design of the software to be developed may be sometimes difficult because gaps exist between the component features and the software requirements.
2) Component Based development has not been widely adopted in domains of embedded systems because of inability of this technology to cope with the important concerns of embedded systems like resource constraints, real time or dependability requirements.

3) Component Based Software Engineering is young; therefore long term maintainability is still unknown.

8. Conclusions

Component Based Software development can increase the productivity and quality of the software. It also reduces the development cost as components are designed to be independent. The development cost is also reduced. But because of the inability of this technology in dealing with real time requirements it is not used widely in the field of embedded systems.

9. Acknowledgments

The help rendered by friends and colleagues in finalizing this paper is duly acknowledged.

10. References

[1] Pressman Roger, "Software Engineering: A Practitioner's Approach", McGraw Hill, Sixth Edition, 2012.
[2] Ian Sommerville, "Software Engineering", 8TH Edition, 2012.
[3] Ivica Crnkovic, "Component based Software Engineering for embedded systems", in IEEE proceedings of International Conference on Software Engineering, 2005.
[4] http://www.cs.man.ac.uk