# COMPONENT BASED SOFTWARE ENGINEERING: A CUSTOMIZED APPROACH FOR SYSTEM DEVELOPMENT

**Deeksha Gupta**

[1]*Department of Computer Science and Applications, MCM DAV College for Women, Chandigarh*

**ABSTRACT**

*CBSE-Component Based Software Engineering technology embodies the "the buy, don't build' philosophy". CBSE is aiming at realizing long-awaited software reuse by changing both software architecture and software process. Because of the extensive uses of components, the CBSE process is quite different from that of the traditional waterfall approach. This paper covers the basic concept behind CBSE approach and system and software architecture required for CBSE. This paper also covers the advantages and limitations of CBSE.*

***Keywords: Architecture, Component Based Development, Contract, Interface, Pattern***

## I. INTRODUCTION

In modern era easy installation, reusability, security, portability and integration have become the most vital features of software. In order to meet all these required features the resulting software are becoming very large in size with high complexity. Usually in software development the main focus is given to release deadlines, budget, complexity and compatibility of software with external software, which in turns overlook the evolutionary requirements of the system. This change in focus results in adverse effect on software development process which may include cost overrun, time overrun, compromise with quality, expensive software maintenance. [1].

The required features like modern large-scale software development, quality, flexibility, reliability of software can be obtained by making change in software development approach. If, for the development of a software system or application system, all required software components are to be made entirely from scratch then it is difficult to deal with the complexity , timely completion and budget issues of the software product. So the solution of this challenge is Reusability. That is why nowadays software development approach is modified to component based development approach (CBD) In this approach software system applications are developed from reusable components which are readymade, pretested and prepared for collaboration with external system. This new approach of software development provides many advantages that include better complexity management, improved time management, improved cost management, better productivity, better quality and high degree of flexibility. [2].

The concept of building software from existing components is not new. But, with new technologies it becomes very easy and effective to build software systems and applications by assembling these prebuilt components. The main objective of software development can be achieved through CBD only if this approach is implemented

in a systematic and effective manner. The basic software development life cycle should be modified so as to incorporate component based software development approach so that good quality applications could be developed in short span and within budget. [2]. The outcome approach is known as *Component-based Software Engineering* (CBSE)

The main focus in CBSE is to build reusable components, to develop software systems and applications using these reusable components and modifying the components for maintaining and improvising the system performance [3]. A systematic approach is required throughout the process of development of component for system and systems from reusable components. CBSE comes with many concepts like components, interface, contract, pattern etc. The heart of CBSE approach is Component which is discussed in the next section.

## II. COMPONENT

A Component is the most important concept of component based software engineering. While making a high quality software component the main focus is to make it reusable so that it can be arranged in any system with a small or no change. These components can be deployed only in those applications which have provision to do so. For such applications there are two ways to obtain the reusable components either built them within the same organization or get them from third party [4].Third party components will save development cost and time though the developer of the application still needs to integrate the components and test the integrated system

According to Clemens Szyperski in context of Component based software engineering- A software component is combination of conventional interfaces and unambiguous framework implementation. It can be acquired from third party and can be integrated with the system independently [5].

The component is an independent and 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 component interactions take place through that interface. Developing a system software using large numbers of these reusable components requires a methodical approach and effort but it results in high productivity and better quality application. [5]

### 2.1 Component characteristics

**Standardized:** Component that is used in a CBSE process has to conform to some standardized component model. This model may define component interfaces, component meta-data, documentation, composition and deployment [6].

**Autonomous:** A component should be Autonomous – the implementation of a component is kept separate from its interface so it can be added to a system with a little or no modification. If it is not specified explicitly then no other external element is required to deploy any component in an application. As a reusable component is independent in terms of definition as well as implementation so if new component is added in an application there is no need to recompile the component [7].

**Compostable:** For a component to be compostable, all external interactions must take place through publicly defined interfaces. In addition, it must provide external access to information about itself such as its methods and attributes.

**Deployable:** To be deployable, a component has to be self-contained and must be able to operate as a stand-alone entity on some component platform. That is, the integration of a component into an application should be

independent of the component development lifecycle and that there should be no need to recompile or re-link the application while updating with a new component [8].

**Documentation:** A good quality component must be reliably documented about its characteristics, features and environments. Documentation of component should specify each and every thing about its working. Proper documentation describes each feature of the component and assists the user in realizing these features.

Component design follows the object oriented approach as the data and functions are kept together in a component. Component interface and component implementation are two main aspects of a component. During the development of component first components requirement and specifications are given which specify the functional and non functional requirements of component in clear, concise and unambiguous manner. After this component is design by defining its interface and behavior. Once component is fully developed, it is tested and then it is implemented and documented. Next section contains the interface details.

## III. INTERFACES

An interface is a collection of operations that are used to specify a service of a component [9]. It states signatures and protocols of a *collection of operations*. Interface focuses upon the behavior, not the structure of a given service that is why it offers no implementation for any of the operation of the component. But in case the operation is not performed successfully then interface also provide the facility of exception handling [10].So the interface element of a component helps to identify the functional characteristics and external behavior of the component so that component can be used as a independent module of the software system.. Interfaces are of two kinds.

1) Service Interface
2) Client interface

***Service Interface*** defines the services that are provided by the component to other components. While ***Client Interface*** defines the services that specify what services must be made available for the component to execute as specified.

To enforce the idea that a component must interact with other software, we might also want to include a property that the service interface *or* the client interface might be empty, but not both [12].

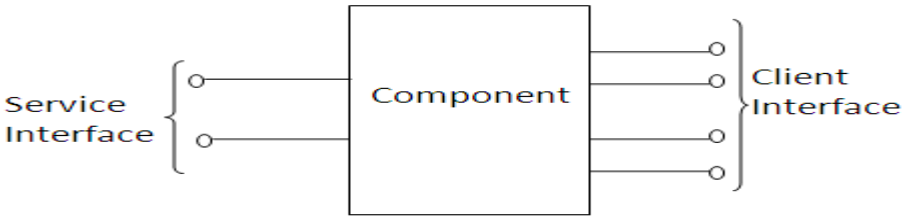


**Fig. 1: Component Interfaces**

Depending upon the nature of the application, Different type of interfaces is required for a component. On the basis of various application environments we can have different type of interfaces, as described below:

- For a communication concurrent application interface will include the communication protocol information but for sequential applications no such information is required.
- Interface for real time systems will represent the real time control while the interface for non real time system will not require such constrains.
- For web based business application the routing information, address and location information will be included by interface.
- If component is to be used completely independently then interface should be simple enough to specify any constraint.

## IV. CONTRACT

Contracts are used to give more precise specifications about the characteristics of a component. While designing the interface the contracts are attached with it to legalize the communication between client and component developer. Contracts increase the confidence of client into the component provided. Contract is needed because the client and the module are co-dependent [13]

In a contract the functional and non functional behavior of the interface is specified. Functional behavior specifies any external requirement for the implementation of the operations and the expected output of the operation. The nonfunctional behavior in the contract states about the various features like quality, portability, productivity, fault tolerance etc. of an interface.

Contracts are of more importance in CBSE because of the premium placed on component substitutability. Two interpretations [11] of contract are possible-

**Component-contract –** This kind of contract specifies a pattern of interaction on that component. The contract specifies the services provided by a component and the obligations of clients and the environment needed by a component to provide these services.

**Interaction-contract** – This kind of contract specifies a pattern of interaction among different roles and the reciprocal obligations of components that fill these roles.

## V. PATTERN

Patterns are the group of techniques, strategies and actions used to find the solution of frequently occurring problem in application development. Patterns are formally used with object oriented approach. A pattern is a template that can be employed to handle a same problem in different situations. Pattern specifies the association between system behavior and mechanism. In other words a design pattern is as independent and reusable as a component. [14].

**Patterns can be classified into three major categories:**

***Architecture Based Patterns-*** These kinds of patterns represent the entire configuration of the application software [15]

***Design Based Patterns*** – These Patterns represent the association between the subsystem and components of an application. They process organization and structure of the components with the subsystems of the software system [11]

***Idioms*** Patterns - These patterns depends upon the tools selected like programming language, database etc. These are also called low-level patterns.

Out of these three patterns while developing a component based software design patterns are frequently used. Design patterns help to understand the behaviour of subsystems and components, which in terms help to develop the reusable component.

## VI. COMPONENT BASED SOFTWARE DEVELOPMENT LIFE CYCLE

Software developed using Component based software engineering are supposed to have a kind of life cycle model specified in Fig. 2.

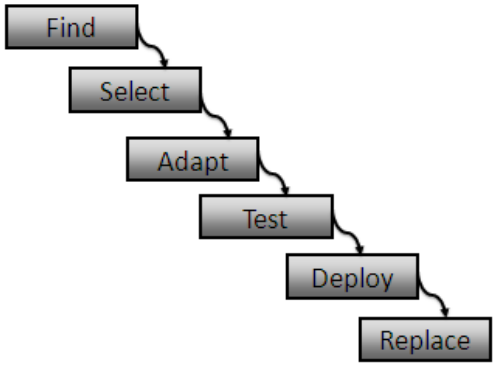


**Fig2: Component based software Development Life Cycle**

The component-based system development process included following different steps:

**Selection of appropriate component:** For the success of a component based software development it is very important to select the component that caters the particular need of the application. A verity of tools and techniques are available for this purpose. Selection of component depends upon various factors like technology used, business need, budget etc.

**Modification of architecture:** If the required component is not compatible with the existing architecture of the system then to make the deployment of component possible modification in architecture of the system is done.

**Adapt the selected components so that they suit the existing component model or requirement specification**. Some components would be possible to directly integrate into the system, some would be modified through parameterization process, some would need wrapping code for adaptation [16].

**Testing the resultant system.** Resultant system after the integration of selected components is tested against an optimal test suite to see if it is working as per requirements.

**Deployment of components in the system**. Components framework is used for the successful deployment of the components.

**Replacement of components.** This step comes under maintenance phase of System. Regular updation of installed components is required to meet the changing needs of the system. This modification may be required to remove any bug from component or to add new functions and features in components.

Other important issues in component based development of application include configuration control management, cost management, schedule management, marketing, business issues, legal issues etc. [17]

## VII. COMPONENT-BASED SOFTWARE SYSTEM ARCHITECTURE

To ensure that a component-based software system can run properly and effectively, the system architecture is the most important factor. The layered approach is used to build the architecture of component-based software systems [2] [3].

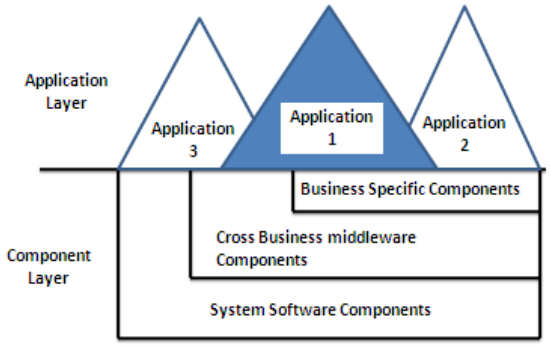


**Fig3: CBSE architecture**

The architecture is divided into two layers. The first layer of the architecture includes the various applications developed using component based approach. This layer is also called application layer. The second layer of the architecture is component layer. This layer includes the various components that are being deployed in various applications. This layer is further divided into three sub layers.

First sub layer includes only component used for explicit applications and specific business. The second sub layer is cross business middleware components. It includes the components that are more general and can be used for multiple applications for example common interfaces and software. The last sub layer of components includes low level components that are required to support the existing system software and hardware.[13]

Current component technologies have been used to implement different software systems, such as object oriented distributed component software [16] and Web-based enterprise application [13]. Many famous software enterprise like Microsoft, sun microsystems, IBM etc. are using component based approch for developing software [10]. An outstanding example is the IBM SanFrancisco project [4]. It provides a reusable distributed object infrastructure and an abundant set of application components to application developers to developing components for further distribution. This infrastructure is compatible with their base languages, tools and techniques. But this infrastructure must satisfy the required constraints of communication and services to provide coordination between various components [2].

## VIII. RISKS /ISSUES INVOLVED IN CBSE

Various risks are also involved in component based approach as compared to traditional software development.

- **Development time of components**. It is very time consuming to make component that can serve various applications.
- **Development effort of components**. To make a reusable component required three to five time more effort as compared to make application specific component..
- **Difficult Requirements specifications.** To make component it is very much important to specify the functional and non-functional requirements of under process component in a clear, concise and complete form**.** Keeping in mind the reusability feature of component it becomes very difficult task. [18]
- **Expensive**. Components are made to be general in nature. But due to this characteristic the functional complexity of component increases very much and these are also expensive in terms of prerequisite resources required with components.
- **High maintenance costs**. The maintenance cost of components is very high as they have to be deployed with different infrastructure and different environments. Although the maintenance cost of overall system reduces with the use of components.
- **Change is not always welcomed:** As the lifecycle and processes of application software is different from that of component, so making some change in application software may affect the functioning of any component of it. This change may be change of operating system, any system software and even change in other components. Reliability and sensitivity to changes. Sometimes change may cause failure of entire application.
- **Difficult to check component reliability:** Sometimes third party components are used for application development and these components are distributed to the client in binary format. So reliability becomes the serious issue. Although there is no standard definition for reliability of components but in general component should do only the specified task and there should not be any masked feature of component that is not known to the developer of the application.
- **Lack of well established certification of components:** The reliability feature of components can be added by certifying the components. But component certification is not a well established procedure especially in terms of software. As under certification the components are tested only against their specified functional and non functional requirement. No checking is done to explore the concealed features of the component[4][19]

## IX. CONCLUSION

CBSE is a reuse-based approach to define and implement loosely coupled components into systems. CBSE is concerned with certified components, predicting the properties of systems and rapid assembly. In a component the most important element is its interface which is used to specify the function of the component and how to use it.. CBSE is a newly developed discipline which deserves several effective and perceived benefits, nevertheless its adoption requires, as any new practice, same care by software development organizations. The adoption concerns by practitioners turn easily into a variety of research issues still to be tackled and properly addressed. In particular to make effective the adoption of CBSE, its differences towards other software development paradigms need to be made more clear and evident outside the market.

## X. FUTURE ASPECT

CBSE is an emerging technology and there are many aspects of CBSE that can be worked upon in future. There are many aspects of CBSE like development and updation of component based applications, system development models of components, component requirement specifications and verification, testing and validation of component based system, prediction model for component composition etc so that CBSE can fulfill its mean and purpose in a complete sense.

## REFERENCES

[1] CBSE Network, "Component based software engineering workshop", Budapest April 3-4

[2] APSEC2000, "Software Engineering Conference", Proceedings, Seventh-Asia-Pacific, 2000.

[3] Sajan Mathew, "Software Engineering", Edition 2nd S.Chand.

[4] George T. Heineman and William T. Councill, "Component-Based Software Engineering Putting the Pieces Together", addison-Wesley, Boston, MA ,880, June 2001.

[5] Sitaraman, M., Atkinson, S., Kulczycki, G., Weide, B. W., Long, T. J., Bucci, P., Heym, W., Pike, S., and Hollingsworth, J. E., "Reasoning About Software Component Behavior", Proceedings Sixth International Conference on Software Reuse, Springer Verlag LNCS 1844, 266-283, 2000.

[6] J. Poulin, J Caruso and D Hancock, "The Business Case for Software Reuse", IBM Systems Journal, 32(40),567-594, 1993.

[7] Dean, J. and M. Vigder, "System Implementation using Commercial- Off-The-Shelf (COTS) Software",1997. URL: http://seg.iit.nrc.ca/papers/NRC40173.pdf.

[8] M. Sitaraman and B. W. Weide, "Special Feature Component-Based Software Using RESOLVE", ACM SIGSOFT Software Engineering Notes 19, No. 4, 21-67, October 1994.

[9] Sitaraman, M., Weide, B. W., Long, T.J., Ogden, W. F., "A Data Abstraction Alternative to Data Structure/Algorithm Modularization", Volume on Generic Programming, LNCS 1766 608, 102-113 Springer- Verlag, 2000.

[10] Murali Sitaraman, Timothy J. Long ,E. James Harner. Bruce W. Weide, "A Formal Approach to Component-Based Software Engineering Education and Evaluation", In ICSE 2001: Proceedings 23rd International Conference on Software Engineering}, pp. 601-609, 2001.

[11] Edwards, S., Shakir, G., Sitaraman, M., Weide, B. W., and Hollingsworth, J., "A Framework for Detecting Interface Violations in Component-Based Software", Proceedings of the Fifth International Conference on Software Reuse, IEEE Computer Society Press, Victoria, Canada, pp. 46-55, June 1998.

[12] Brown A. Large-Scale Component-Based Development. Prentice Hall, 2000

[13] N. Medvidovic, R. Taylor, and E. Whitehead, "Formal Modeling of Software Architectures at Multiple Levels of Abstraction", In Proceedings of the California Software Symposium 1996, Los Angeles, CA, pp. 28-40., April 1996.

[14] G. Pour, M.Griss,J.Favaro, "Making the Transition to Component" 2001

[15] Xia Cai, Michael R. Lyu, Kam-Fai Wong Roy Ko ,"Component-Based Software Engineering Technologies Development Frameworks and Quality Assurance Schemes", The Chinese University of Hong Kong Hong Kong Productivity Council.

[16] Heineman G. and Councill W. Component-based Software Engineering, Putting the Pieces Together. Addison Wesley, 2001

[17] A.W.Brown, K.C. Wallnau, "The Current State of CBSE", IEEE Software ,Volume:15 5, pp. 37-46.,Sept.-Oct. 1998.

[18] Eun Sook Cho et al., "Component Metrics to Measure Component Quality ", Proceedings of the eighths Asia-Pacific Software Engineering Conference, 1530-1362,2001. [14] G.Pour, M. Griss, J. Favaro, "Making the Transition to Component" 2001

[19] G. Pour, "Component-Based Software Development Approach: New Opportunities and Challenges", Proceedings Technology of Object- Oriented Languages, TOOLS 26.,pp. 375-383,1998.