Chapter 4

Emergence of Component Based Software Engineering

4.1 Introduction

In early days, software engineering approach was ad hoc. Around 1970s, introduction of structured programming" gave a formal shift in software engineering from the ad hoc to a systematic approach. Then around 1980s, introduction of object oriented programming with some advancement explores new areas in software engineering. In recent dates, with the introduction of Component Based Software Development (CBSD), the industry is moving in a new direction. The basic insight is that most software systems are not new. Rather, they are variants of systems that have already been built. This insight can be leveraged to improve the quality and productivity of the software production process (73). These days software systems are more complex as compared to those of early. These complex, high quality software systems are built efficiently using component based approach in a shorter time. Component based systems are easier to assemble and therefore less costly to build than developing such systems from scratch. The importance of component based development lies in its efficiency. In addition, CBSE encourages the use of predictable architectural patterns and standard software infrastructure, thereby leading to a higher result. In the remaining part of this thesis the term "component"and "software components"will be used interchangeably.

Based on the publication in the "*International Journal of Advanced Research in Computer Science and Software Engineering(IJARCSSE)*", ISSN: 2277 128X, 2(3), 311–315, 2012.

4.2 The journey of the Component Based Software Development (CBSD) Era

In 1968, Douglas McIlroy's (74) first share the idea of Component Based Software Development (CBSD) at the NATO conference on software engineering in Garmisch, Germany in his paper titled "Mass Produced Software Components". He discussed the idea that, software may be componentized i.e. built from pre-developed software components. His subsequent inclusion of pipes and filters into the Unix operating system was the first implementation of an infrastructure for this idea. Later, Brad Cox set out to create an infrastructure and market for these components by inventing the Objective-C programming language. IBM led the path with their System Object Model (SOM) in the early 1990s. Some claim that Microsoft paved the way for actual deployment of component software with OLE and COM. As of 2010 many successful software component models do exist.

4.3 Understanding Software Components

In early days, the principles of software engineering have been focused in developing software system from the very scratch development for individual software. It means that, for all the functionalities to be supported by a system have been designed and coded individually for the proposed systems under development. Brown (75) posits that it would be infeasible for developers and organizations to consider constructing each new information system from scratch. Instead, information systems would need to be developed with reused practices, software components and products that have been tested and proven to be effective and efficient in order to remain in business and gain competitive advantage (75, 76, 77). Upon long observation it was found that- there are certain functionalities those are common in many systems. Hence, if these common functionalities can be developed independently - may be reused in different systems without redevelopment from scratch. Later, these can be integrated to any system as part whenever it is suitable! At the same time, it will reduce the development effort of such systems as there is no need to develop the same common parts again and again for different systems. This originates the concepts of Software Components. Although, a software components is a small parts of a system, but often a large system as a whole may be seen as a software component as well. It is also possible that, the system consists of components is a component itself. In all cases, the components are required to be reusable components after all.

Historically, "component"in software is a rough synonym for "module"or "unit"or "routine". A generally accepted view of a software component is that, it is *a software unit with provided services and required services from others too* (Figure 4.1). The provided services are operations performed by the component whereas, the required services are the services needed by the component to provide target services. The one or more interface of a component consists of the specifications of its provided and required services (78).

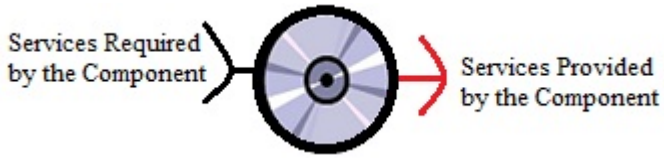

Figure 4.1: Software Components (An Abstract View)

As component is simply a data capsule, information hiding becomes the core construction principle underlying components. Clemens Szyperski (79) suggests shifting the focus away from code source. He defines a software component as executable, with a black-box interface that allows it to be deployed by those who did not develop it. It is important for a software component to be easily combined and composed with other software components. This is because a software component will only achieve its usefulness when it is used in collaboration with other software components (80).
According to Herzum and Sims (81), the term "component"is used in many different ways by practitioners in the industry. However, some rather broad and general yet useful definitions are as follows:
"*An independently deliverable piece of functionality providing access to its services through interfaces.*"–Brown (82)

"*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 third parties.*"– Clemens Szyperski (83).

"*A component denotes a self contained entity (a.k.a. black box) that exports functionality to its environment and may also import functionality from its environment using well defined and open interfaces. In this context, an interface defines the syntax and semantics of the functionality it comprises (i.e., it defines a contract between the environment and the component). Components may support their integration into the*

surrounding environment by providing mechanisms such as introspection or configuration functionality."– Michael Stal (84).

In essence, more precisely a component may be defined as:
"An independently deployable and compositional software element conforming to software architecture"

4.4 Using Software Components: Component Based System Development

The usage of a component in a software system includes using it to replace an out of date component to upgrade the system or a failed component to repair the system, adding it to the system to extend the system services, or composing it into the system while the system itself is still being built. Some researchers insist on a component being reusable during dynamic reconfiguration (85). Component Based System Development (CBSD) advocates developing software systems by selecting reliable, reusable and robust software components and assembling (shown in Figure 4.2) them within appropriate software architectures.

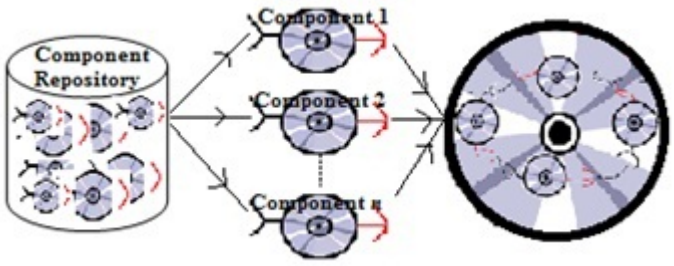

Figure 4.2: Component Based System Development

With the help of middleware technologies- a set of specifications or rules in the form of functions, which when incorporated into the code allows the software to be integrated with software developed using other platforms/languages (86). Example of such middleware technologies are COM, DCOM, CORBA, JavaBeans, EJB etc. Component Based Software Engineering (CBSE) is a process that aims to design and develop software systems using existing, reusable and adaptable software components as opposed to programming them. Hence, *CBSE shifts the emphasis from programming to composing software systems.*

4.5 Objectives of Component Based Software Development

The goal of component based development is to build and maintain software systems by using existing software components (77, 87, 88, 89, 90, 91). As said by Dr. Randall W. Jensen (92), "High customer demand, reduced software development budgets, and a competitive software market drive the need for reusable software". When correctly applied and implemented, developing software systems using Commercial Off The Shelf (COTS) software components promises benefits like increase productivity, shorten time-to-market, improve software quality, reduce maintenance cost, allow for inter-application interoperability, decreased level of risks, leverage technical skills and knowledge, and improve system functionality (93, 94, 95). As, software crisis may be loosely defined as the set problems associated with the software development process (62) i.e. quality, development cost and time-to-market of software, we may conclude that the indirect objective of CBSD are to alleviate software crisis. In addition, the particular objectives of software components are to:

1. **Develop Replaceable Components:** A component must have useful *replaceable property* i.e. easy to assemble and easy to disassemble.

2. **Increase Reusability:** Develop once and reuse several instances of the same over the period.

3. **Facilitating System Change Management and System Maintenance:** The plug and play feature of a component allows easy component composition and inclusion in the information systems.

4. **Enhancing Development Flexibility:** Components are an independent software element that can be designed and developed independently enhancing the development flexibility.

5. **Reduced System Development Time:** Reusing pre-developed existing components instead of new fresh development will reduce total development time.

6. **Reduced System Development Cost:** Reduced development time will result in significant reduction in total development cost.

7. **Improve Software Quality:** Ideally, a component is pre-tested for errors and quality parameters. Hence, using such pre-tested, high quality software components improves the quality of complete software systems.

8. **Reducing Project Risk:** From management perspective, if an asset's costs can be optimized through a large number of uses, it would then be possible for the management to expend more effort and allocate more budgets to improve the quality of software components. This in turn reduces the level of risk faced by the development effort and will undeniably improve the likelihood of success (96).

9. **Improve interoperability:** When systems are developed using reused components, they are expected to be more interoperable as they rely on common mechanisms to implement most of their functions (97).

10. **Increase System Learning for User:** Dialogs and interfaces used by these systems would be similar and would improve the learning curve of users who utilize several different systems built using the same components (96, 97).

11. **Ease and Efficient System Debugging:** As the components are previously tested, if any error occurs, must be during the integration. Hence, the domain and range for debugging is minimized and localized. This facilitates the debugging process quite easy and efficient.

4.6 Impact of Component Based Software Development

The CBSD approach includes improvements in: quality, throughput, performance, reliability and interoperability; it also reduces development, documentation, maintenance and staff training time and cost (1, 94). In this approach, due to inherent functional independence software is assembled from components can be autonomously deployed and, the productivity and performance of the development team can be improved (81). The impact of software reuse in system development is highlighted below:

1. **Impact on Productivity:** Although percentage productivity improvement reports are notoriously difficult to interpret, it appears that 30–50% reuse can result in productivity improvements in the 25–40% range. According to Lim (93), Hewlett-Packard software projects reported productivity increases from 6% to 40%

with the incorporation of CBSD. Further, Pitney Bowes in the USA which has been reusing components since 1996 documented tremendous savings in labour as the company is now able to achieve 500 human-weeks of development progress in only 200 human-weeks by using existing components and by purchasing others from component markets (98).

2. **Impact on Quality:** In a study conducted at Hewlett Packard, Lim (93) reports that the defect rate for reused code is 0.9 defects per KLOC, while the rate for newly developed software is 4.1 defects per KLOC. Henry and Faller (97) reported that, for an application that was composed of 68% reused code, the defect rate was 2.0 defects per KLOC—a 51% improvement from the expected rate, had the application been developed without reuse. They further report a 35% improvement in quality by component reuse in system development. They again suggested that, the quality of information systems developed using this approach will also have fewer bugs and defects if compared with newly built-from-scratch systems.

3. **Impact on Time-to-Market:** The STG division reports that the same development effort using the reusable work product required only 21 calendar months compared to an estimated 36 calendar months had the reusable work product not been used, a reduction of 42 % (93).

4. **Impact on Cost:** Apart from productivity gains, component reuse allows organizations to reduce the critical path in the delivery of software systems, reducing the time-to-market and begin to accrue profits earlier. Study shows that there has been reduction in product cost up to 75–84% as a result of reuse (99).

4.7 Industrial Practices on Software Components

"The use of commercial off-the-shelf (COTS) products as elements of larger systems is becoming increasingly commonplace. Shrinking budgets, accelerating rates of COTS enhancement, and expanding system requirements are all driving this process. The shift from custom development to COTS based systems is occurring in both new development and maintenance activities. If done properly, this shift can help establish a sustainable modernization practice." – SEI COTS-Based Systems Initiative (100).

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. In recent years, component technologies have been well developed, such

as Enterprise Java Beans (EJB) of Sun (101), CORBA Component Model (CCM) of the OMG (102), and Component Object Model (COM) of Microsoft (103) and are discussed below :

- **SUN JavaBeans Components :** The JavaBean (101) component system is a portable, platform independent CBSE infrastructure developed using the Java programming language. The JavaBean system extends the Java applets to accommodate the more sophisticated software components required for component based development. The Bean Development Kit (BDK) encompasses a set of tools to facilitate the CBSD approach.

- **OMG/CORBA :** The Object Management Group has published common object request broker architecture (OMG/CORBA) (102). An object request broker (ORB) provides a variety of services that enable reusable components to communicate with other components, regardless of their location within a system. When components are built using the OMG/CORBA standard, integration of those components without modification in a system is assured if an Interface Definition Language (IDL) is created for every component.

- **Microsoft COM :** Microsoft has developed a Component Object Model (COM) (103) that provides a specification for using components produced by various vendors within a single application running under the Windows operating system. COM encompasses two elements: COM interfaces that are implemented as COM objects and a set of mechanisms for registering and passing messages between COM interfaces. From application point of view, "the focus is not on how 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 (104)."

4.8 Conclusion

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 third parties. Developing software using Commercial Off-The-Shelf reusable component is known as Component Based Software Development (CBSD). Informally, application of Software Engineering principles and practices in CBSD is known as Component Based Software Engineering (CBSE). CBSD

qualifies, adapts, and integrates software components for reuse in a new system. In addition, often engineers need to develop additional components that are based on the custom requirements of a new system that are unavailable from component library. CBSD offers inherent benefits in software quality, developer productivity, and overall system cost, but yet many roadblocks remain to be overcome before the CBSD is widely used throughout the industry. We may conclude that, component based development is the future development process to cater the present software crisis. Industries must follow this development practices, built and enlarge their component library for future reuse. At the same time industries must train their developers to encourage and practice the component based software development process and need to set up appropriate facility centers for supporting CBSD process. Despite lots of potentialities, there are still lots of issues that are to be explored for being the CBSD successful. Hence, researchers have to take the responsibilities on their shoulder to resolve these issues and make CBSD a successful software development approach.