CHAPTER 6

MODIFIED DEVELOPMENT PROCESS OF CBSE

6.1 Introduction ...............................................................105
6.2 Component-Based Software Engineering and Component Properties ..........105
6.3 Modified Development Process of Component-Based Software Engineering ...107
6.4 Problem Solving Using Modified Development Process of CBSE ...............109
6.5 Benefits of Reusable Components in Programming ...............................110
6.6 Summary .....................................................................112
6.1 INTRODUCTION
As the rapid increase of software programs in the size and complexity, it is very important to reduce high software cost and complexity while increasing reliability and modifiability. With the advances of programming, more distributed systems are built to meet diverse application needs. So to build the software according to the new programming approaches this study proposes a modify development process of Component-Based Software Engineering (CBSE). CBSE emphasizes on building system by reusing high quality configurable software components. This modified process helps to develop high quality software components. As more third-party software components are available in the commercial market, good software development process is required to use the component engineering approach to develop component-based approaches for distributed applications. Lastly, on the basis of X model and modified development process of CBSE, this chapter also presents component reuse into two different approaches, namely, composition-based approach and generation-based approach. This chapter also introduces various abstraction levels for reusable components to increase the reusability with the benefits of reusable components in programming.

6.2 COMPONENT-BASED SOFTWARE ENGINEERING AND COMPONENTS PROPERTIES
Component-Based Software Engineering (CBSE) is a discipline leading software engineering into a new generation. CBSE offers a promising way to promote software reuse, decrease the cost and time of the development process, via proper information encapsulation and separation of concerns at various levels. The basic unit in CBSE generally has a large granularity. Therefore, CBSE enhances the modularity of a software system at a much higher level. Software components often take the form of objects or collections of objects from Object-Oriented Programming (OOP), in some binary or textual form, adhering to some Interface Description Language (IDL) so that the component may exist autonomously from other components in a computer. For large and complex application, some components need to be developed separately specifically tailored to the need of
the application. The most commonly reused software product is source code, which is the final and most important product of software development. In addition to code, any intermediate life cycle products can be reused, which means that software developers can pursue the reuse of requirement documents, system specifications, modular designs, test plans, test cases, and documentation in various stages of software development. A component needs to use another component in order to function; it adopts a used interface which specifies the services that it needs. Today, modern reusable components encapsulate both data structures and the algorithms that are applied to the data structures. It builds on prior theories of software objects, software architectures, software frameworks and software design patterns, and the extensive theory of OOP and design. Programmer generally considered the following properties in the development of software component.

6.2.1 Segregation of Design
Principle of segregation of design decisions in a program is information hiding in computer science, thus protecting other parts of the program from extensive modification if the design decision is changed. The protection involves providing a stable interface which protects the remainder of the program from the implementation (http://en.wikipedia.org). A component should totally hide its implementation from the party who uses it.

6.2.2 Explicit Interface
A component must have a well-defined interface such that component users know how to compose it. With regards to system-wide coordination, components communicate with each other via interfaces. When a component offers services to the rest of the system, it adopts a provided interface which specifies the services that can be utilised by other components.

6.2.3 Context Independency
A component should be independent of other components, which guarantees that change to any other components won't break the component. It is an independent,
compositional and deployable unit. A component that has implicit dependencies on the context in which it is used is harder to reuse. A system in which implicit dependencies exist between components is brittle. Making a change to one component can break others which have an implicit dependency on it, but the cause of the failure will not be immediately obvious. (http://c2.com/cgi/wiki).

6.2.4 Substitutable
Another important property of components is that they are substitutable, so that a component could be replaced by another at design time or run-time, if the requirements of the initial component are met by the successor component. Consequently, components can be replaced with either an updated version or an alternative for example, without breaking the system in which the component operates.

6.3 MODIFIED DEVELOPMENT PROCESS OF COMPONENT-BASED SOFTWARE ENGINEERING
Component-Based Software Engineering (CBSE) is a process that aims to design and construct software systems using reusable software components. The goal of CBSE is to build a software system by assembling small components in the same manner as hardware composition. Modified development process of CBSE change the reusability approach and divide the reusability into two different approaches. It can be generation-based and composition-based. The generation-based approach reuses the process of previous software development efforts, often embodied in computer tools that automate a part of the development life cycle (Henderson et al., 1990). The composition-based approach reuses existing software products in a new system to avoid repetitive work. The modified development process of CBSE is quite different from the traditional software engineering approach (Gill, 2002; Jalote, 1997; Sommerville, 2009; Pressman, 2004; Mall, 2003). It can be generalized as having the following eight steps in the following Figure 6.1. Domain engineering, design, coding and archiving, component testing, assembly and system testing are already discussed in the
previous chapter. Now this chapter present in detail only two phases (domain analysis and specifications, and component and system architecture).

- **Domain Analysis and Specifications**
  The goal of the domain analysis and specification is to find or create those classes that are broadly applicable, so that they may be reused. The software domain analysis is identification, analysis and specification of common requirements from a specific application domain, typically for reuse on multiple projects within that application domain (Crnkovic et al., 2003). Domain analysis help in defining the domain to investigate and categorize the item extracted from the domain.
• Component and System Architecture

There are two central parts in CBSE - components and architecture. Components are the basic building blocks for a software development in CBSE. A component is an entity which fulfills one or more requirements. It has clearly defined and documented interfaces that interacting with other components according to the system architecture. It may have explicit context dependencies and this has already been discussed in section 6.2.3. Architecture describes how components are assembled into an application system. Architecture of the component and system are helpful in assembling of components in software development.

6.4 PROBLEM SOLVING USING MODIFIED DEVELOPMENT PROCESS OF CBSE

Problem solving is an important determinant of the range of problem and solutions that will be considered, as well as an important source of problem-solving difficulty. Reusability is an important characteristic of a high quality software component that helps in solving problem with the help of reusable component by using different abstraction levels viz. architectural, modular design and framework level are discuss below. A software component should be designed and implemented according to the different abstraction level so that it can be reused in many different programs. Reusable software components provide programmers with higher level abstraction concepts that are both close to application domains and easy to implement. Components increase programmers' ability to frame the problem into representation that is easier to solve. Without the support of reusable components, programmers have to frame each concept in the problem domain based on their knowledge of the programming language. With the support of software components, however, the difficulty of problem framing is reduced because certain concepts can be directly mapped to the components.

6.4.1 Architecture Level

Research on software architecture is currently aiming to define different software architecture styles for different families of software systems (Perry and Wolf, 1992). A software architecture style describes the formal arrangement of
CHAPTER-6  Modified Development Process of CBSE

architectural elements, and can be reused by software developers to construct their new software systems once the style is well defined (Shaw and Garlan, 1996). The domain-independent multifaceted architecture is an architecture style for domain-oriented design environments, which has been reused in and refined through the development.

6.4.2 Modular Design Level
Reusable knowledge on modular design can be codified in design patterns (Alexander et. al, 1977) and frameworks. A design pattern is the description of a solution to recurring problems. It specifies a problem to be solved, and the context in which the solution works (Gamma et al., 1994). Design patterns provide a common vocabulary for software developers to discuss their designs and can be passed from one developer to another developer for reuse.

6.4.3 Framework Level
The concept of framework comes from OOP languages (Fischer et al., 1995). A framework describes the interaction pattern among a set of collaborative classes or objects, and can be represented as a set of abstract classes that interact with each other in a particular way (Johnson, 1997). Programmers can reuse frameworks directly in their development after providing implementations for those abstract classes. Framework reuse is a mixture of knowledge reuse and code reuse.

6.5 BENEFITS OF REUSABLE COMPONENTS IN PROGRAMMING
This chapter presents the benefits of reusable component in programming which is developed by the modified development process of CBSE. Programming knowledge at the level of code is represented as program plans that can also be reused by programmers if a suitable representation form is defined (Rich and Waters, 1988). In the 1960s, scientific subroutine libraries were built those were reusable. These subroutine libraries had a limited domain of application but reusable component had unlimited domain of application and also know that a library of reusable components is the single largest contributor not only to
programming productivity but also to the quality application software. Components in programming and software development are pieces of the application puzzle. When a programmer wants to do more than just "the basics", he may choose to use pre-built third party components to add features to his software application that he either does not have the time and/or the expertise to code himself. Components encapsulate functionality into a reusable form that can easily be included with application. The component needs to be designed with good interface, fully documented, thoroughly tested, and logically designed with the use of any programming language. Software reuse approach is commonly thought to lead to fewer product problems, and development time, greater productivity and easier maintenance. Reusable software components have benefits in programming during the development of software systems. This study introduce few benefits that are the immediate benefits that a programmer can attain during the implementation of a programming task and few benefits may not be immediately enjoyed by the programmer who reuses the components, but they extend to the whole life cycle of the software system and to later programming activities of the programmer. Few of the benefits are -- reduced development time and cost; improved quality, easily maintainable, improved evolvability and lower defect density.

6.5.1 Reduced Development Time and Cost
Reduced development time by reusing existing software components, fewer programs are written, and thus less time is spent in programming. Furthermore, because reusable components are usually carefully tested already, less time is needed in debugging and testing, which are the "hard and slow part" of programming (Brook, 1995).

6.5.2 Improved Quality
Improved quality because software components are often repeatedly reused, the defect fixes from each reuse accumulate, resulting in higher quality of the developed software systems.
6.5.3 Easily Maintainable
Reprocessable components contribute to easy maintenance not only because they have fewer defects, but also because they facilitate communication among software developers by providing a set of common vocabulary, especially for the indirect communication between system builders and system maintainers.

6.5.4 Improved Evolvability
To cope with constantly changing requirements and implementation platforms, software systems must be able to evolve. Reusing software components improves the evolvability of software systems because it can limit the needed change to components instead of identifying and changing all occurrences distributed all over the system (Yunven, 2001).

6.5.5 Lower Defect Density
Reused components had a lower defect density than non-reused ones, with defect density calculated by dividing the number of defects by the number of lines of code.

6.6 SUMMARY
This chapter deals with the modification of development process of CBSE. CBSE emphasizes on building system by reusing high quality configurable software components with higher reliability and better maintainability. The study proposes here a modified development process of CBSE with several phases - domain engineering, domain analysis and specifications, component and system architecture design, coding and archiving, component testing, and assembly. This modified process divides development of component into two different processes, namely, composition-based approach and generation-based approach. This study introduces the benefits of reusable components in programming which is built by keeping in mind the several abstraction levels, namely, architecture level, modular design level and framework level to increasing reusability with the help of modified development process of CBSE.