CHAPTER 5

X MODEL - A NEW COMPONENT-BASED MODEL

5.1 Introduction .................................................................................................................. 81
5.2 Component-Based Lifecycle Process Models ............................................................... 81
5.3 X Model for Component-Based Software Development .............................................. 82
5.4 Working of X Model .................................................................................................... 85
5.5 Development of Reusable Components .................................................................... 87
5.6 Development with Reuse by Building Software from Reusable Components .......... 92
5.7 Component-Based Software Development with X Model ........................................ 97
5.8 Summary ...................................................................................................................... 101
5.1 INTRODUCTION
In software engineering, development lifecycle processes of any software include all activities of software during its entire life. Historically, traditional model of software development have focused primarily on the functionality of a software system. There are two main groups of models - sequential and evolutionary. The sequential model defines a sequence of activities in which one activity follow after a completion of the previous one. Evolutionary model allows performance of several activities in parallel without requirements on a stringent completion of one activity to be able to start with another one. Well known examples of sequential models are waterfall model and V model, and of evolutionary models are iterative and incremental development, and spiral model. This study proposes a new X model for Component-Based Development (CBD) which comes under the evolutionary model. X model is a complex multifaceted activity that involves several phases - software analysis and specifications, design, coding and archiving, component testing, component wrapping, domain analysis, domain engineering, frameworking, assembling, archiving and design of software components. Finally, X model appears to cover the likely phases of large software development and enforces software reusability along its phases with two processes – development for reuse and development with reuse.

5.2 COMPONENT-BASED LIFECYCLE PROCESS MODELS
Many of the models, process and principles of software engineering used in other types of system will be used in the same or a similar way in Component-Based Software Engineering (CBSE). There is however one difference; CBSE specifically focuses on questions related to components and in that sense it distinguishes the process of “component development” from that of “system development with components”. Components raise the level of abstraction to that which can be easily used by a domain expert who is not necessarily an expert programmer. A typical development effort using components would be importing the components of interest and customizing each one it without explicit coding and finally wiring together the components to form an application. The
components in software development help in increasing productivity gained by reuse of design and implementation, help in increase reliability by using well tested code, help in lower the maintenance costs because of a smaller code base, help in minimizing the effects of change since Black Box Programming (BBP) tends to rely on interfaces as compared to explicit programming and components also provide a well encapsulated mechanism to package, distribute and reuse software. Components are built to be used and reused in many applications. Software development with components is focused on the identification of reusable entities and relations between them, beginning from the system requirements and from the availability of components already existing. In reality the processes are already separate as many components are developed by third parties, independently of software development. Even components being developed internally in an organization which uses these very same components, are often treated as separate entities developed separately. V model is widely used in many organizations typically large organization building complex long-life products. The Y model, (Luiz, 2005) has been proposed as a viable alternative to address software reusability during Component-Based System (CBS) production. The creation of software is characterized by change and instability, hence the Y model considers overlapping and iteration where appropriate. Although the main phases may overlap each other and iteration is allowed, the planned phases are - domain engineering, frame working, assembly, archiving, system analysis, design, implementation, testing, deployment and maintenance. To avoid iteration and overlapping this study propose a new X component-based model.

5.3 X MODEL FOR COMPONENT-BASED SOFTWARE DEVELOPMENT

This study propose a new X model in which the processes start in usual way by requirement engineering and requirement specification as shown in Figure 5.1. In a non component-based approach the process would continue with the unit design, implementation and test. Instead of performing these activities that often are time and efforts consuming, simply select appropriate components and integrate them
in the system. However, two problems appear here which break this simplicity - It is not obvious that there is any component to select, and the selected component only partially fits to our overall design. The main characteristic of this software life cycle model is reusability in which software is developed by building reusable components for software development, and software development from reusable and testable components. In software development, this research uses two main approaches, develop software component for reuse and software development with or without modification in reusable component. Evolution and the production of potentially reusable components are meant to be useful in future software projects. Reusability not only involves reusing existing components in a new software system but also producing components meant for reuse. When a software system has been developed, the software engineer may realize that some components can be generalized for potential reuse in the future. Reusability implies the use of composition techniques during software development; this is achieved by initially selecting reusable components and assembling them, or by adapting the software to a point where it is possible to pick out components from a reusable component repository and testable component repository. This study introduces two main phases first is building reusable components for software development and second is building software from reusable and testable components (Tomar and Gill, 2006) of X model which help in developing a component-based software.

5.3.1 Building Reusable Components for Software Development
A component is built to be reused and reusability implies generality and flexibility, and these requirements may significantly change the component characteristics. The development process of building components can follow an arbitrary development process model as shown in Figure 5.2a. The generality requirements imply often more functionality and require more design and development efforts and more qualified developers. The component development will require more efforts in testing and specification of the components. The components should be tested in isolation, but also in different configurations. Finally the documentation
and the delivery will require more efforts since the extended documentation is very important for increasing understanding of the component.

![Figure 5.1 X Model - A New Component-Based Model for Component-Based Software Development](image)

**5.3.2 Building Software from Reusable and Testable Components**

The main idea of the component-based approach is building systems from pre-existing components as shown in Figure 5.2b. This assumption has several consequences for the system lifecycle. First, the development processes of CBS are separated from development processes of the components; the components should already been developed and possibly used in other products when the system development process starts. Second, a new separate process will appear - finding and evaluating the components. Third, the activities in the processes will be different from the activities in non component-based approach; for the system development the emphasis will be on finding the proper components and verifying them, and for the component development, design for reuse will be the main concern.
5.4 WORKING OF X MODEL

X model has two main phases during Component-Based Software Development (CBSD) and two components (repositories).

5.4.1 Phases of X Model for Component-Based Software Development

First phase introduces the building reusable components for software development and second introduces the building software from reusable and testable components.

5.4.2 Components of X Model

There are two main components of X Model - Testable Component Repository (TCR) and Reusable Component Repository (RCR).

5.4.2.1 Testable Component Repository

TCR is a testable component repository used to store the testable component which is created at the end of each phase of development for reuse and development with reuse (development after modification and development without modification) after the component testing.
5.4.2.2 Reusable Component Repository

RCR is a reusable component repository and is created at the completion of CBS. Maintenance of software system is last phase that does not only allow the software to evolve but also the reusable library in RCR concerning the existing systems expands during the maintenance and after the completion of development phases.

This study is shows here the working of X model with the help of diagram. According to the Figure 5.6, after completing the software analysis and specifications, and design, developer check the RCR for reusable component.

- When components are not available according to software analysis, specifications and design from RCR, then start development of component for reuse from scratch as shown in Figure 5.3.
- If components specifications match 70 to 80% according to software analysis, specifications and design, then start development with reusable components after modification as shown in Figure 5.4.

![Figure 5.3 Development for Reuse in V shape with Green Boxes](image-url)
• When reusable components are available according to software analysis, specifications and design from RCR, then start development with reusable components as shown in Figure 5.5.

5.5 DEVELOPMENT OF REUSABLE COMPONENTS

According to the Figure 5.6 after completing the software analysis and specifications, and design this study check the RCR for reusable component. When components is not available according to software analysis, specifications and design from RCR, then start development of component for reuse from scratch as shown in Figure 5.6.

![Diagram showing the development process with reusable components](image)

**Figure 5.4 Development with Reusable Component after Modification with Blue Boxes**

5.5.1 Software Analysis and Specifications

The software analysis is the activity that feeds information to the specification activity. During analysis a complete and consistent set of specifications emerge for the software development through answer to questions, questionnaires, information from documentation. The software analysis phase emphasizes
identification of high-level components in a real-world application and decomposition of the software system. The software analysis phase demands the systems analyst to study the application and its constraints, understand the essential features of the system, understand the requirements expected to be satisfied by the software system and create an abstract model of the application in which these requirements are met. The main product of the software analysis phase is a graphical or textual description of an abstract model of the application which helps in figure out its subsystems and major components. This phase is not concerned with the details of the components when compare functional decomposition. This abstract model of the application comprising high-level abstractions of software components is better understood for software specification. The software specifications are instead to establish what services are required from system, for system operation and development (Pressmen, 2001). Software specification is the activity of translating the information gathered during the analysis activity into a document that defines a set of requirement which help in design of components.

Figure 5.5 Development with Reusable Component without Modification with Parrot Green Boxes
5.5.2 Design

Software design is a description to the structure of the software and component to be implemented. In this phase the major tasks involve the implementation of components, in order to fulfill the required software functionality. Implementing a component requires defining the data structures and corresponding algorithms to provide the overall software services. The data which is part of the system, the interface between system components and sometimes, the algorithm used (Gill, 2002). According to the Figure 5.6 after completing the software analysis and specifications and design this study checks the RCR for reusable component.
When components are not available according to software analysis, specifications and design from RCR, then start development of component for reuse from scratch as shown in Figure 5.6. Design and development of components from scratch require expert software engineers. The number of key issues discussed in this study to design for reuse is standard data, standard interface protocols and program template which help in making new components that conform to this have a higher portability of subsequent reuse. When designers face an application, they should not ask “how do I work out a solution to this problem”? (Luiz, 2005). Instead, they should ask, “where are the components that I can directly or indirectly reuse to solve this problem”? At this point, they should be able to examine a reusable library (TCR and RCR) and to select components that closely match the entities necessary to build the software. The designer looks for components trying out a variety of schemes in order to discover the most natural and reasonable way to refine the software solution. The design process may involve developing several model of the system at different levels of abstraction. As design is compared, errors and omission in earlier stages are discovered. There has been a tendency to present software design in such a manner that it looks easy to do. Nevertheless, in the design of large and complex software, identification of key components is likely to take some time. Repetitions are not unusual, since a good design usually takes several iterations. The number of iterations also depends on the designer’s insight, experience and knowledge about the application domain. The design process should stop when the key generic abstractions and the software behavior are detailed enough to be translated into a programming language. Hence, the design stage generates the templates for the coding and archiving. When researchers and practitioners are try to reuse executable components they are inevitably constrained by detailed design decision that has been made by implementation of these components. These range from the particular algorithm that has been used to implement the components to the objects and types in the component interfaces. If this design decision conflict with your particular requirements than reusing the component is either impossible.
5.5.3 Coding and Archiving

The design must be translated into a machine readable by using coding. Coding completely depend on a good design because if design performed in a detailed manner, code generation can be accomplished mechanistically by translation of a design model into a programming language. Software reusability is an attribute that refers to the expected reuse potential of a software component. The software development community is gradually drifting toward the promise of widespread software reuse, in which any new software system can be derived virtually from the existing code (Gill, 2003). Reusability not only involves reusing existing components in a new software system but also producing components meant for reuse. When a software system has been developed, the software engineer may realize that some components can be generalized for potential reuse in the future. An important consideration in the quest of reusability is how to make a potentially reusable component available to future projects. Archiving (Luíz, 2005) should reflect the activities involving cataloging and storage of components. The component must be understandable, well written and well documented.

5.5.4 Component Testing

Testing of CBS is best viewed as two distinct activities - the testing of the component as a unit which have completed at the end of each phases - development for reuse and development with reuse (development after modification and development without modification). In developing a large software system, testing usually involves several stages and testable component stored in TCR for CBSD. First, each component is tested on its own, isolated from the other components in the system. Such testing, known as component test or unit test verifies that the component functions properly with the types of input expected based on the component’s design. Once software components are created, it is time to test them. During the testing phase it is not the first time when faults occur, they can be carried through from the software analysis and design phases. But testing is focused in finding faults and there are many ways to make testing efforts more efficient and effective. There are several techniques that
can be used throughout this process such as white-box, black-box, code inspection, walkthroughs, formal proof and so on for components testing.

5.6 DEVELOPMENT WITH REUSE BY BUILDING SOFTWARE FROM REUSABLE COMPONENTS

Check components according to software analysis, specifications and design from RCR. If components specifications match 70 to 80% according to software analysis, specifications and design then start development with reusable components after modification as shown in Figure 5.7.

5.6.1 Development of Software after Modification in Reusable Component

Development of software after modification in reusable component is completed with the help of following phases, namely, select the component from repository, evaluate and adaptability, component wrapping and archiving, and component testing.

5.6.1.1 Select the Components from Repository

If components specification are very near to client requirements then select the components according to software analysis, specifications and design from RCR and TCR and then start evaluation and adaptability of selected component.

5.6.1.2 Evaluate and Adaptability

Component adaptation and evaluation is a challenge in CBSD because a component is written to satisfy the requirements of particular CBS (Gill, 2003). But when researchers and practitioners want to use it in another application, it is not adaptable and evaluate directly in almost all the cases because the requirement, standards, environment, specification and code of the component does not match with the requirements, standards, environment, and specification of another application. So after a little change component can develop to meet the requirement of software. Software architecture represents design patterns that are composed of components, connections and coordination. In essence the architecture defines the design rules for all components, identifying mode of connection and coordination. In some cases, existing reusable component may be
adapted to meet the needs of architectures design rules. These components must be adopted to meet the needs of architecture or discarded and replaced by other, more suitable components. In reality ever after a component has been qualified for use within application architecture, it may exhibit conflict in one or more of the area just noted. To mitigate against these conflicts an adoption technique called component wrapping is often used.

![Diagram of X Model - A New Component-Based Model](image-url)

**Figure 5.7 X Model - A New Component-Based Model (Development with Reuse and Development after Modification)**
5.6.1.3. **Component Wrapping and Archiving**

Component wrapping is a process in which a software developer has the power to change the reusable component (*Pressmen, 2004*). When a software team has full access to the internal design and code for a component, White Box Wrapping (WBR) is applied. WBR examines the internal processing details of component and make code level modification to remove the conflict. Gray Box Wrapping (GBW) is applied when component library provides a component extension language or Application Programming Interfaces (API) that enable conflicts to be removed or marked. Black Box Wrapping (BBW) requires the introduction of pre and post processing at the component interface to remove or mask conflict. Component archiving has already been discussed in section 5.5.3.

5.6.1.4 **Component Testing**

This issue has already been discussed in section 5.5.4 for component testing.

5.6.2 **Development of Software without Modification in Reusable Component**

Development of software without modification in reusable component is completed with the help of following phases, namely, domain analysis and specification, domain engineering, frameworking, and component testing.

5.6.2.1 **Domain Analysis and Specifications**

When reusable components are available according to requirements from RCR and TCR, then software developer start development with reusable components as shown in Figure 5.8. The goal of the domain analysis is to find or create those classes that are broadly applicable, so that they may be reused. The 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 be investigate and categorize the item extracted from the domain.

5.6.2.2 **Domain Engineering**

Domain engineering is used 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 is a process of analyzing an application domain in order to identify areas of commonality and ways to describe it using a uniform vocabulary (Luiiz, 2005). Thus, domain engineering is an activity that should be carried out at the beginning of software specification if reuse is to be considered and the main aim of the domain engineering is to establish mechanism that enable software engineers to share these components to reuse them during work on new and existing system. User needs, software requirements, provided functionality, objectives and constraints of the system are very much of interest during the system analysis and domain engineering phases. Thus, it is important to understand the real-world application and an abstract model of that application should be depicted. Therefore, the boundary between system analysis and domain engineering may be seems fuzzy because identifying key abstractions in the application domain may be viewed as part of system analysis or domain engineering. Nevertheless, at this level, domain engineering is also concerned with the identification of potentially reusable components as shown in Figure 5.8.

5.6.2.3 Frameworking

In frameworking, the software engineer might have a sketchy idea about components for reuse. On the other hand, as frameworks comprise sets of components that express a design for a family of related applications. A framework could be viewed as a generic structure that provides a skeleton for producing software in a certain application domain. Frameworking attempts to identify components and establish interrelationships perceived important within the application domain (Luiz, 2005). Such identification of components may arise from the well-known functionality common to that application domain, usually in the form of semantic relationships between components. Consider, for example, the application domain of airline reservation systems; typical entities of these systems are - seats, flights, crews and passengers; and interrelationships can be - reserve a seat, assign a crew to a flight, schedule a flight and so on. So, there are important relationships among these entities, which can be organized into a
framework according to their semantic meaning in that application domain. Building software by using frameworking is faster and easier than starting from scratch, although frameworks will not be as generally useful outside their application domain because they contain domain-dependent components. Within the proposed life cycle model, the main result of the frameworking phase should be the reuse of software components already developed and the classification of components to form new frameworks.

5.6.2.4 Component Testing

This issue has already been discussed in section 5.5.4 for component testing.
CHAPTER-5  X Model - A New Component-Based Model

5.7 COMPONENT-BASED SOFTWARE DEVELOPMENT WITH X MODEL

After completing the two phases - first is building reusable components for software development and second is building software from reusable and testable components through X model which develop CBS with the help of reusability. This study start the main development of software by using testable and reusable component from the TCR and RCR with the help of different phases like assembly of component, system testing, implementation and deployment and maintenance. At the end of component development, there may be many new reusable components developed that need to be verified, catalogued, classified and then stored into RCR. Figure 5.9 shows here development for reuse in green boxes and development with reuse (development after modification in blue boxes and without modification in parrot green boxes). The components are then selected, adapted and reused through composition, generalization and specialization mechanisms. Figure 5.10 and 5.11 shows through flow of lines how all components from TCR and RCR comes together to develop CBS.

5.7.1 Assembly

Component integrated into an architecture style and interconnected with an appropriate infrastructure that allows the component to be coordinated and managed effectively as shown in Figure 5.11 from RCR and TCR. It focuses on selecting a collection of reusable components or frameworks from specific application domains. There are differences in the mechanisms used to achieve reusability when different kinds of reusable components are involved. The most basic software components are often reused by composition, which can be seen as a process of building a piece of software from elementary self-contained components; although reusability is naturally accomplished by reusing classes through inheritance during object-oriented development. This phase is usually akin to sifting through a junkyard of books rather than visiting a library. Testable component assembled together to build a CBS for particular application domain by using all testable components according to domain engineering, software analysis and specification of user.
5.7.2 System Testing

System testing as shown in Figure 5.1 is the process of verifying that the system components work together as described in the design specification. After a collection of components has been unit-tested, the next step is ensuring that the interfaces among the components are well defined. Component-Based Software Testing (CBST) is an important activity of CBSD cycle and is based on two main perspectives (Gill et al., 2007). First one is component providers that refer to the testing of components by their developers on the basis of the source code and the second one is the component user that refers to the testing of components by their users, without access to their source code (Mary, 2000). Even if source code is available, components should be only black-box tested because of the missing knowledge about the internal structure of the component and the danger of losing oneself in too much detail. The component provider perspective addresses testing
issues that are of interest to the providers of the software components. The component providers view the components independently of the context in which the components are used, and effectively test all configurations of the components in a context-independent manner. The component user perspective, in contrast, addresses testing issues that concern the user of software components. The component users view the components as context-dependent units because the component users' application provides the context in which the components are used. The user is thus concerned only with those configurations or aspects of the behavior of the components that are relevant to the component users' applications.

Testing issues in the software testing (Hans, 2002) include redundant testing during integration of components, availability of source code, heterogeneity of language, platforms and architectures, monitoring and control mechanism in distributed software testing, deadlocks and race conditions. Testing requirements and test process documentation are the two main factors that help in improving the CBST.

Figure 5.10 X Model - A New Component-Based Model for CBSD in which Components from all Directions Assembled for CBSD
5.7.3 Implementation and Deployment

The implementation phase is characterized by the implanting the component together to build a complete software at user site. The CBS must be easily configurable or adaptable at user site. Some components picked out during the implementation phase should undergo further refinements, e.g. treatment of exceptional conditions and verification, until they become generic and robust enough to be placed in a reusable library. This surely adds an overhead to software construction, which is more than compensated for by the long term savings when such components are reused in future projects. Implementation and deployment is almost the end of system development, now the system is ready to be presented to the customer. Nevertheless, deployment involves more than putting the system into place, it is the time when users should be helped to understand and feel comfortable with the software. If deployment is not successful, users may be unhappy with its performance. In either case, users will
CHAPTER-5

not be as productive or effective as they could be and the care taken to build a
high-quality system. The two key issues to successful transfer from the developer
to the user are documentation and training, which should be integrated with the
software.

5.7.4 Maintenance

Software is normally subject to continuing changes after it is built, when it is
operational. Thus the efforts turn now to the challenge of maintaining a
continually evolving system. During software maintenance, as shown in Figure
5.1 changes are introduced to a software system. Such changes are not meant only
for correcting errors occurred in the operational software; these changes may be
also for improving, updating the system to anticipate future errors or adapting the
system in response to a modification in the environment. Therefore, during the
maintenance phase, maintenance of software system does not only allow the
software to evolve but also the reusable library concerning the existing systems
expands during the maintenance of a legacy system. When the system is
implemented and deployed at user site than the maintenance phase start. Many
software engineers wrongly assume that once a system is delivered their problems
are over. A system life does not end with deployment. During the maintenance
phase, software components may be accessed from, as well as new ones may be
added to a reusable library of the concerned application domain. For instance, a
change to adapt the software to a new environment may specialize already
existing components, so that characteristics of the new environment are taken into
consideration, hence expanding the spectrum of environments the reusable
components are able to operate (Luiz, 2005). After changes are introduced to the
system, an updated release of the software is generated.

5.8 SUMMARY

This chapter proposed a new X component-based model for CBSD. X model for
CBSD attempts to rationalize the development of a software system with several
phases - software analysis and specifications, design, coding and archiving,
component testing, component wrapping, domain analysis, domain engineering, frameworking, assembling, system testing, implementation and deployment, and maintenance of software components. X model helps in developing CBS with the help of two CBSD approaches, namely, development for reuse and development with reuse. Finally, X model appears to cover the likely phases of large software development and enforces software reusability along its phases. Moreover, it takes into account previous knowledge that software engineers may have about the application domain, which has an influence on the prevailing approach to be followed during the software development with this model.