CHAPTER 1
INTRODUCTION AND STATEMENT OF THE PROBLEM

1.1 Introduction
Software Engineering is an art and science. It is the process of constructing acceptable artefacts with scientific verifications and validations within the boundaries of time and budget. The term “Software Engineering” came into existence in 1960s when NATO Science Committee organized conferences to counter with the problem of “Software Crisis”. Krueger [1] described software crisis as “the problem of building large, reliable software systems in a reliable, cost-effective way”. Earlier the focus of the industry, research community and the academia was concentrated on the development of capable and competent hardware. The result of this effort was the availability of powerful and cheaper machines. Now, there was the requirement of large, functionally efficient software to fully utilize the ability of available hardware machines and other resources. Therefore the focus of community shifted from small, number crunching programs to the problem of developing software that can address the real world problems. The outcome of those conferences was the evolution of the term ‘Software Engineering’.

Software is a set of executable programs, related documentation, operational manuals, and data structures. Engineering is the step by step evolution of constructs in an organized, well defined and disciplined manner. Eminent researchers and practitioners have defined Software Engineering in prominent ways. Some leading definitions are:

“Software Engineering is the establishment and use of sound engineering principles in order to obtain economically developed software that is reliable and works efficiently on real machines” by Fritz Bauer, 1969 [2].

“Software Engineering is the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures, and associated documentation” by Barry Boehm, 1981 [3].

“Software Engineering is the systematic approach to the development, operation, maintenance, and retirement of software”, IEEE, 1987 [4].
“Software Engineering is a discipline whose aim is the production of quality software, software that is delivered on time, within budget, and that satisfies its requirements” by Stephen Schach, 1990 [5].

“The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software” by IEEE, 1993 [6].

The conclusion of all these definitions is that, the ultimate goal of Software Engineering is to develop quality software within the reasonable time-limits and at affordable costs. Software Engineering is an engineering domain that efforts to implement standard ways of developing and maintaining software, through standard methods using standard tools and techniques. Definitely the focus of these standards is to develop quality software at reasonable cost.

1.2 Component-Based Software Engineering (CBSE)

Component-Based Software Engineering (CBSE) is an elite form of Software Engineering that offers the feature of reusability. Reuse of software artefacts and the process of reusability make CBSE a specialized paradigm of software development. CBSE stands on the philosophy of “the buy, don’t build”. The basic idea of CBSE is to reuse pre-constructed and available software constructs rather than developing them from the beginning as shown in Figure 1.1. Gaedke [7] defined Component-Based Software Development (CBSD) as “Component-Based Software Development (CBSD) aims at assembling large software systems from previously developed components (which in turn can be constructed from other components)”. CBSD methodology promotes the development of software systems by picking suitable and apposite pre-existing, pre-built and reusable (off-the-shelf) software work-products called ‘components’ and integrate those components with a pre-defined architectural design. CBSD emphasizes on the thought of assembling heterogeneous, context independent and pre-designed components to develop software applications [8]. Component-Based Software applications are constructed by the assembling of reusable, pre-existing as well as new components which are integrated through error-free interfaces [9][10][11][12][13]. The objectives of Component-Based Software Engineering are to develop extremely large and complex software systems by integrating ‘Commercially Off-the-shelf Components (COTS)’, third-party contractual components and newly developed components to minimize the development time, effort and the cost [14][15]. Component-Based Software Engineering offers an improved and enhanced reuse of software components with additional
properties including flexibility, extendibility and better service quality to discharge the needs of the end users [16][17][18][19].

Figure 1.1 Framework of Component-Based Software Engineering.

In CBSE the purpose is to develop a component (including classes, functions, methods or operations) only once and re-use them in various applications rather than being re-constructed every time. Reusing pre-developed and pre-tested components make the development life-cycle shorter, help to increase the reliability of the overall application, and reduce the time-to-market.

The CBSE development paradigm is used to develop generalized as well as specific components. The development processes of CBSE applications deploy four parallel processes, which are involved in the creation of more than one application concurrently, as described in Figure 1.2. Processes including new component development, selection of pre-existing components from the repository and integration of components take place concurrently in the generation of application in the Component-Based Development. With each process there must be a feedback method to
address the problems and errors like, component selection problems, problems in developing new components, interaction and integration errors among the components, and their side effects. To manage all these parallel activities there must be a control procedure or manage procedure which will assess not only the development process but it will manage the analysis of requirements, selection of components, integration of components and the most importantly the quality of components submitted in the repository for future reuse.

Jacobson, Griss and Jonsson [20] defined four similar concurrent processes used in CBSE and these processes are associated to develop various applications. These processes are creation, management, support and reuse. Creation process involves in developing new applications by reusing the existing software components. The support process provides help and maintenance activities in the development of new applications. This process is used to provide existing components from the repository. The reuse process collects the requirements and analyse these to select the components from the repository. This reuse process is responsible for the actual development of components through the property of reusability.

![Diagram](image.png)

Figure 1.2 Concurrent Processes of Component-Based Software Development
1.2.1 Evolution of Component Based Software Engineering

The idea of components was introduced by Douglas McIlroy in the NATO conference on Software Engineering in 1968 in his defining speech titled “Mass-Produced Software Components” [21]. In eighties and nineties the term component gained identity as a building block and architectural unit. But after the period of about 40 years, in year 2000 Component-Based Software Engineering found its place and grown as a standard paradigm of Software Engineering. Today Component-Based Software Engineering is one of the most promising and capable paradigm for the development of large, complex and quality software systems.

Since its orientation, the evolution of CBSE can be divided into 4 phases: the preparatory phase, the progression phase, the defined phase, and the expansion phase.

**Phase 1: The Preparatory Phase**

In the initial phase of Component-Based Software development researchers contribute to collect the information about CBSE. Worldwide conferences and workshops were organized to initiate and define approaches, challenges and implications in the context of CBSE. The first workshop on CBSE was held in Tokyo in 1998. The major achievements of the initial phase are:

a. CBSE became a popular paradigm of developing software,

b. Component is assumed and defined as a pre-existing and reusable entity,

c. Standard literatures in the context of CBSE came into existence like Heinemann and Councill’s “Component-Based Software Engineering: Putting the Pieces Together” [13].

**Phase 2: The Defined Phase**

In the second phase, practitioners focused on more specific and specialized branches of CBSE. Terms related to CBSE were specifically defined and classified. Worldwide conferences and workshops were organized and new research fields and research topics like “predictable assembly”, “component trust”, “component specifications”, “automated CBSE” and similar came into existence. The milestones of the defined phase are:

a. Specific and concrete terms in the context of CBSE were defined, like component architecture, component specification, component adaptation, component acquisition, etc.

b. These defined topics created new research areas and research groups.

c. Number of academic and research publications increased and organizations like ACM started their publications in the area of CBSE.
Phase 3: The Progression Phase

Third phase commences new and inventive areas of CBSE. In this period of time not only the research groups but software industry also tributes his efforts to cope the benefits of CBSE. Major research areas of CBSE during this phase are Component-Based System Modelling and Design, testing framework of components, Component-Concepts for Embedded-Structures, estimation of resources at dynamic time in multi-task CBS environments, design and analysis of Component-Based Embedded Software, component structure for critical dynamically embedded contexts, hierarchical framework for Component-Based Real-time Systems. This phase is regarded by the collaboration of Component-Based paradigm in Software Engineering of other areas. The recognized progress of this phase is that the new and collaborative areas have been identified and defined, like, multi-task, real-time systems, embedded systems, critical systems, performance attributes in collaboration with CBSE.

Phase 4: The Expansion Phase

It is assumed that this phase is still continuing. This is the time in which parallel and balancing approaches recognized in Software Engineering and software development. Areas like “Service-oriented Development”, “Aspect-oriented Programming”, or “Model-driven Engineering” stood parallel with CBSE. The major areas of research in this phase are Component-Based web services, service-oriented architectures, Development, customization and deployment of CBSE systems, components supporting Grid computing, Global measurement, prediction and monitoring of distributed and service components, Integrated tools, techniques and methods for constructing Component-Based service systems, Components for networked dynamic and global information systems linking sensor.

1.2.2 Characteristics of Component-Based Software Engineering

Heineman and Councill [13] explained three fundamental properties of CBSE: first the development of systems from existing software entities, second the capability to reuse these pre-existing entities as well as the newly developed software entities in other applications and third comfortable preservation, maintenance and fabrication of these entities for future implementations. Component-Based Software Engineering possesses a number of properties over the other Software Engineering paradigms. Some primary characteristics of CBSE are listed below.

❖ **Reusability:** Reusability is the focal property behind the Component-Based Software Engineering. Krueger defined software reusability as, “the process of creating software systems from existing software rather than building them from scratch” [1]. Software reuse is the
process of integrating predefined specifications, design architectures, tested code, or test plans with the proposed software [22, 23, 24]. CBSE relies on the re-using these artefacts rather than re-developing them. Components are developed in a way that they can be heterogeneously used and then reused in various environments and also various environments can use and reuse heterogeneous components.

- **Composability**: One of the fundamental characteristic of CBSE is that the components are the reusable, composable software entities, that is, applications are composed of different individual components. These individual reusable components are designed in a way that they can be reused in composition with other components in various applications with minimum or no fabrication. Components are composed of components, that is, components itself made up of components, which further made up of other components, and so on [25]. A component can be a part of one or more components.

- **Shorter Development Cycle**: Component-Based Development paradigm follows the concept of divide, solve and conquer approach. In this paradigm complex and bulky applications are divided into smaller, more manageable units or modules. Then rather to start coding of a complete module from the first line, we search and assemble existing elements that satisfy the requirements of the module under consideration. This increases the development speed of software. In addition several modules can be implemented concurrently, regardless of location, or context. Thus the development time is saved and development cycle becomes shorter.

- **Maintainability**: The effortlessness by means of which we can add new functionalities to the application, modify, update or remove old features from the software is referred to as the maintenance. Since CBSE based software are made up of reusable and replaceable components, we can add, update or remove or replace components according to the requirements of the software. Maintaining composable and independent components is much easier than maintaining monolithic software.

- **Better Quality and Reliability**: Component-Based technology assures for superior quality as the CBSE integrates and couples pre-tested and qualified components. These components are specifically tested at least at the unit level. During their integration with other pre-tested components, the developer performs integration as well as system tests [26]. This regressive form of testing makes Component-Based applications more robust and better quality product. In a broader scale the effort, cost and time of testing is reduced to the noticeable extent.
Components are independently developed and are deployed in various contexts at the same time with minimal or no fabrications, they are integrated according to the pre-defined architecture; hence it is assumed that there are no unwanted interactions among the components which make these components unreliable. All the interactions paths are pre-defined so the reliability and predictability of components increased.

- **Flexibility and Extendibility:** Software developers have choice to customize, assemble and integrate the components from a set of available components according to their requirement. Replaceable and compasable components are easy to add, update, modify or remove from the application without modifying other components. Error navigation and fixing are relatively easy as it is limited to component level only.

### 1.3 Components

The literary meanings of a component defined by different dictionaries are:

- "Contributing to the composition of a whole", *The Oxford English dictionary*.
- "A component is any part of which something is made", *Oxford Dictionary*.
- "A constituent part; an ingredient", *The Webster's Dictionary*.
- "A part which combines with other parts to form something bigger", *The Cambridge dictionary*.

The core element of Component-Based Software Engineering is the component itself. Components are the basic unit of reusability and the foundation blocks of building Component-Based Software. They are the focal point and can be assumed as the prime operative part of Component-Based Development paradigm. In literature components have been defined by various researchers in variety of ways. Some leading definitions are-

**Definition 1: Szyperski**, defined component as: “A software component is a unit of composition with contractually specified interfaces and context dependencies only. A software component can be deployed separately and is subject to composition by third parties” [27].

**Definition 2: Booch**, defined component as: “A component is a logically cohesive, loosely coupled module” [28].
Definition 3: OMG, defined component as: “A component is a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces” [29].

Definition 4: Brown, defined component as: “A software component is an independently deliverable piece of functionality providing access to its services through interfaces” [30].

Definition 5: Heinmann and Councill, defined component as: “A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard” [13].

Definition 6: Szyperski, defined component as: “Software components are binary units of independent production, acquisition and deployment that interact to form a functioning system” [31].

Definition 7: D'Souza and Wills, defined component as: “A coherent package of software artifacts that can be independently developed and delivered as a unit and that can be composed, unchanged, with other components to build something larger” [32].

Definition 8: Michael, defined component as: “A component is a language neutral, independently implemented package of software services, delivered in an encapsulated and replaceable container, accessed via one or more published interfaces” [33].

Definition 9: Philippe, defined component as: “A software component is a nontrivial, nearly independent, and replaceable part of a system that fulfils a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces” [34].

Definition 10: Hopkins, defined component as: “A software component is a physical packaging of executable software with a well-defined and published interface” [35].

Definition 11: Meyer, defined component as: “A component is a software element (modular unit) satisfying the conditions: a) it can be used by other software elements, b) it possesses an
Analysing these prominent and diverse definitions, one can deduce that all these definitions contain some universal points. In the context of this work and in the light of these universal directives we can identify a general definition of components. We can define a software component as:

- A component is an identifiable, functionally reusable unit (element/part/piece/package) of larger software system.
- A component is developed independently.
- A component is delivered and deployed in the context free environment.
- A component interacts with other components through well-defined interfaces to access and to provide services.
- Software component can be replaceable, modifiable piece of code which can be integrated with other replaceable, modifiable components.

Component is an identifiable and functionally reusable unit refers to the fact that component can be reused at various levels with different degrees of reusability. Bennatan [37] classifies three classes of components:

- **Off-the-shelf components:** In this category, components are pre-developed, pre-tested and can be reused without any modification. Off-the-shelf components are either made available by the third party or selected from the repository. Components residing in the repository are constructed by the developers of the organization as part of the earlier developed software. Off-the-shelf components are like Black-Box components as the developers can use or reuse them without concerning the idiosyncrasies and internal details of that component. That is, the interface of the component should be self-descriptive about the functionality of the component. These components can be deployed in a context free environment without making any change or modification. Off-the-components are also called as Commercially-Off –the-Shelf (COTS) components.

- **Adaptable components:** In this category, components are assembled with existing software artefacts including applications requirements, architectural design, available code, or pre built
test cases with little or large modifications. We can divide adaptable components into two broad classes: *fully-qualified components* and *partially-qualified components*. This division is based on the degree of modifiability of the component. The fully-qualified components need no alteration or a minimum amount of alteration. These components can be reused at their maximum extent with minimum modification. The requirement of modification depends upon the requirement of the current application. The partially-qualified components need alteration to the greater extent. These components can be reused in the application under consideration with major modifications [38].

- **New components:** New components are those components that are engineered by the organizations developers from the first line of code to the end considering the requirement specification of the specific software application. New components are developed when neither off-the-shelf nor adaptable components satisfy the requirements of the software system. In Component-Based Software Engineering, components are developed in a way that they can be reused in various applications. Newly developed components are submitted in the repository for future use.

Components can be developed independently, delivered and deployed in the context of free environment. Components interact with each other to gain and to offer their services. In Component-Based Development, components interact and communicate to contribute their services to the software system [39]. When components communicate with each other they share information including data, logic, and other worthy substances [40], [41].

To gain as well as to provide valuable information, components require three fundamental properties: *Interfaces, Services, and Deployment techniques*.

- **Interface:** To provide and to access information and services, components require some sort of intermediary among them, that is, an interface. An interface is a medium which defines “how a component can interact with other components”. It states the communicating structure, limits and parameters and also describes the form and structures of the result in return. In CBSE, developers use interfaces to assemble and to make interaction among the dependent and independent components. Interfaces define the mode of coupling between two or more components. The signature (data type, count) of corresponding parameters and the nature of
communication among the components decide the type and way of coupling. Components use interfaces to pass the sequence of control from one component to others, to gain and provide files and data, and to transmit instructive information to each other. When components are integrated through various coupling techniques to contribute their services and functionalities to the whole system, interface is a medium which provides the way of interaction among these components [42].

 **Services:** To survive in a system, every component must have to provide some desired and defined service (set of services) or functionality (set of functionalities). In Component-Based Software systems, the service specifies a component’s role in the system. When developers select a component from the repository, one of the basic criterion of the selection is its service which it will provide to other components and to the system. In addition, when components are stored in the repository, the basic aim is to reuse their services in future applications. Component’s service consist not only the functional behaviour of the component but also the functional procedures that would be assembled with other pre-existing components [43]. In the context of CBSE, it is clear that –

- Services must have some specific and defined purpose,
- Services should exhibit robust behaviour,
- Services should be reliable,
- Services should exhibit efficiency in terms of performance and adaptability.

 **Deployment Techniques:** In Component-Based Development, components are deployable units. Therefore, it is necessary to have a deployment technique or the way of deployment with every component. Deployment techniques are the methods through which a component is customized for the purpose of its implementation. If component’s source code is available then customization is done through its executable code otherwise through its interface [44]. Deployment techniques follow the syntax and semantics of the language in which component was developed. In the context of black-box and context free components, generally internal details are not available with the component. In such cases interfaces play a major role in the deployment. To make deployment effective, interfaces must be robust and reliable.

To develop a complex and large system, it is divided into comparatively small manageable pieces. In Component-Based Systems, components represent these small pieces. Components are
replaceable, modifiable piece of code which can be integrated with other replaceable, modifiable components. To address the requirements of the system, it is essential to partition and modularize these requirements. Modularization must be done in a way that, each component should represent a specific requirement (or a set of requirements). Now, the primary question arises that what should be the level or degree of break-up. This trade-off is referred as the matter of ‘Componentization’.

Componentization: Componentization is the method of identifying the quantity of components in a specific application developed through Component-Based Development. Componentization addresses the issue of maintaining balance between the number of components and the complexity factors of the system. Essentially the level of componentization is equivalent to the level of requirement sufficiency. We consider as many components, as are adequate to solve the software application’s intension. Now, the question arises that ‘what will be the level of requirement sufficiency’. Figure 1.3 describes the concept of componentization, that is, Componentization Vs Integration Cost.

![Componentization Vs Integration Cost](image)

Figure 1.3 Componentization Vs Integration cost [42]

If we divide the problem into a large quantity of components providing small functionalities, it will increase the cost of integration, efforts of interaction. Not only the cost but the number of interactions, coding complexity, testing effort and number of duplicate test cases will also
increase. If an application is componentized in fewer amounts of components providing a bunch of functionalities in single components, it will cost in terms of testing cost and efforts as well as maintenance efforts. It is desirable to achieve a minimum cost region to balance the cost and effort with the number of components.

1.4 Statement of the Problem:
The main objective of this thesis is to develop an efficient model driven analysis of Component-Based Software Engineering. This has been divided into following sub problems-

- Analysing the reusability feature of components and developing reusability measures and metrics at component level as well as at system level, for utilizing and enhancing the usability of components in Component-Based Software Development.
- Analysing the interaction-integration complexities of components and developing measures and metrics to capture and reduce the interaction-integration behaviour of components in Component-Based Software.
- Analysing and developing the testing and test case generation techniques for Component-Based Software.
- Analysing and developing efficient and effective reliability metrics to enhance the quality of CBSE applications to gain benefits in the long run.

These above problems are taken as framework of the thesis which is shown in Figure 1.4.
This thesis focuses on these four aspects of CBSE with a model driven approach which consists of various scenarios and case studies in the forthcoming chapters. The aim of the thesis is to analyse and suggest measures and metrics to enhance the performance of the Component-Based Software.

1.5 Modeling Tools Used
This section discusses the tools and languages used to model the components, and Component-Based Software scenarios. Rational Rose is a Computer Aided Software Engineering tool. It incorporates two main constituents of software development: Component-Based Development as well as Controlled-Iterative Development. It facilitates users to create, view, manipulate and modify constituents in a Unified Modeling Language (UML). In this thesis Rational Rose is used in Creating and Displaying a Component Diagram, Component Diagram Toolbox, Assigning a Component to Another Package, Component Specifications, Package Specification, Deployment Diagrams and Specifications, Creating and Displaying a Deployment Diagram, Deployment Diagram Toolbox. Simulations are carried out with the help of ‘C’ language for various case studies.

1.6 Organization of Thesis
This thesis has been organized into 7 chapters. In chapter 1, the introduction of the thesis is given. Introduction includes defining the context of the thesis, definitions of components and CBSE, evolution of the CBSE paradigm, and statement of the problem of the thesis. The basic characteristics of CBSE, related problems and observations are also discussed in this chapter. Chapter 2 is devoted to review and analyse the background issues in the context of previous works done so far in the area of reusability, interaction and integration among components, testing issues and reliability considerations of Component-Based Software. Research gaps are also identified in this chapter. Chapter 3 explores the reusability features of components and the use of reusability in selection and verification of components. In this chapter metrics for reusability are developed at component level as well as at system level. These metrics are categorized according the degree of reusability as new, adaptable or off-the-shelf components. Chapter 4 defines some simple and effective interaction and integration metrics to assess the complexities for Component-Based Software. These interaction metrics are divided into two categories: black-box and white-box. For black-box components, Integration metrics are developed and for white-box components, Cyclomatic complexity metrics are constructed. Chapter 5 elaborates the testing and test case generation issues of Component-Based Software. Two techniques are defined in this chapter; one
for those components whose code is not accessible and the other for those components whose code is available. Chapter 6 illustrates the reliability of Component-Based applications using reusability feature of the components. To enhance the reliability of Component-Based Software, reusability metrics are used as the factor of reliability estimation. Finally, chapter 7 concludes the present work and gives a direction for the scope of future work.