Conclusion

Use of simulation in Component Based Software Engineering is one step forward in achieving objectives of obtaining reliable software components from component repositories in lesser time by putting lesser efforts and within optimum cost. In past one decade various component based technologies have been developed by different corporate houses and these have achieved commendable success also. Through software reuse existing solutions can be applied to new problems. This way duplication of efforts, required in developing that solution, time and money can be saved. Various aspects of software components and components based software need to be experimented upon before components can be integrated together to give shape to a component based software. These aspects of software components and component based software can be measured or simulated on the basis of probability distributions of the behaviour of different aspects in real life environment.

Due to the ever increasing costs and risks associated with actual experimentations, simulation techniques have been applied in various field of human life. Software Engineering in general and Component Based Software Engineering in particular is a sophisticated engineering discipline where simulation has not been used to the extent it has been used in other disciplines. But just like other fields of life here also application of simulation has great potential. In the presented research work potential of simulation in Component Based Software Engineering has been explored and several simulators have been designed and developed and their results studied in order to study the behaviour of component based software.

Chapter 3 of this research elaborates upon the emphasis on simulation experiments as tools for research in component based software engineering. In this chapter following areas of component
based software engineering process where simulation as a tool can be applied for getting useful results were identified.

- Component Development Process and Release.
- Component Requirement Management.
- Component Selection.
- Reliability evaluation of component based systems.
- Component Based Project Management
- Component Integration.
- Risk Management of Component Based Systems.
- Component and Integration Testing.
- Rapid Application Development

Reliability of any large system is a function of the reliabilities of its system components. Component based software is also not an exception. A model that that computes the reliability of a component based system as a function of its components was proposed in Chapter 4. This model is based upon the Markov chain process and transition probabilities. A Component Control Flow Graph (CCFG) represents the component based system with each node of the graph representing one component and that corresponds to a state of Markov process. Execution control from one component to another takes place as per the Markov process. By simulator implementation a sensitivity analysis on the system was performed where individual component reliabilities were taken as inputs and system reliability was generated as output. It was concluded that every component has some effect on the overall application reliability although effect may vary from component to component. Simulation results showed that overall application reliability increases when reliabilities of individual components are increased. The results of this simulator can be helpful in choosing a component from some existing package or application for fitting it in a new application. Simulator also shows the relationship between numbers of components in a component based
system and overall system reliability. This simulator can be an asset to the project team in deciding how many components can be there in an application keeping in view how much reliable software is desired. As any component based system may be composed of many heterogeneous components, while integrating or composing these components it is important to insure that they are tested for proper working. But this is also true that due to obvious reasons it is very difficult to test each and every component. There are some components that need to be tested more rigorously and thoroughly as compared to other components. Identifying such components is very important. After identifying such components, more efforts and resources can be put in testing such components. In Chapter 5 a simulator was developed that can be helpful in identifying these type of critical components. Basis for identifying the most critical components in the system is a graph called Component Execution Graph (CEG). CEG is a network representation of a Component Based System. Each node of the CEG represents a Component in the system and an edge from node i to node j represents the transfer of execution control from component i to component j. Each execution starts at first component of the CEG and finishes at the last component of the CEG. For achieving a meaningful result at least one path must be executed. In between it may take any courses of execution, depending upon the result desired. All nodes of the CEG are not covered during each execution. Nodes that are executed most number of times are identified as most critical components. This simulator can be very handy in situations where project team need to identify the most error prone components and execution links for testing and that need more testing time and efforts and to decide how to distribute the human as well as financial resources in testing the components and their interactions. This simulation can further be extended to include
features that can be helpful in deciding which component and execution link need to be tested up to what level. Identifying best suitable component from a number of components against the required criteria is uphill task. Chapter 6 discusses a method for doing so. The component storage pool becomes more efficient in terms of the request processing time and quality delivery of software component when there are more than one candidate components found for given requirement. The system proposed in the chapter works on the ranking algorithms that updates the property tables of the components. Depending upon the usability feedback from the system where the component are used, the system increments or decrements the numeric values of the properties of components depending upon its usability experience with that component. The proposed system categories the components for their suitability to frameworks, arrange components which can be used in different environments, components which can be used with defined cross interface notations of multiple frameworks. The proposed system makes the component reusability system more robust, precise and efficient. Well-defined notations for the ranking of components make the system capable of learning after the component reusability. The proposed system makes it possible to provide the best possible component for the raised requirement. The source of request is used to identify the target framework and deliver the best possible component for that environment. System checks if the component is available for the same framework, as required, if yes, then it again searches for the best available components on the basis of previous usability experience.

This chapter also explains about the experimental details and its results carried out on a simulated environment in order to achieve the precision in application development after reusing the components.
The model proposed in the chapter, named as CAS (Component Access System), ensures that the reusable component is made available to the users in less time with more appropriateness towards the requirements. Therefore the proposed system is capable of improving the overall performance of Component Based system by adding the suitable attributes and interfaces.

As Component Based Development lays emphasis on composing software from pre-existing commercially off the shelf (COTS) components, component repositories are searched for the existing components according to requirement specifications and then components are integrated in the system. Though all the components are important for the success of a Component Based Software, some of them may be more important than others. While distributing the cost, efforts, time and other resources, starting from component requirement specification to component integration, there is always a need to differentiate between more and somewhat less important components and distribute the resources accordingly. In Chapter 7, a simulator was developed that identifies the critical components in a component based system so that resources can be distributed optimally while integrating the components in the system. Proposed simulator can be used to plan the distribution of available resources in a better way. This will help to overcome the problems of cost and time overrun while integrating and deploying components in a Component Based System.

Component Based Development is a market driven technology where Components are not developed for a specific application, rather they are developed in such a way so that they can be reused in many different kinds of applications across different platforms. Different organizations in the common marketplace offer different components for the same functionality. Competition in the market is very high and time to market is very short. It is not possible that the entire
requirements be implemented in one go in a component. To survive in
the market, companies release the components in versions with very
first version implementing only bare minimum requirements. When a
particular version is released in the market and used by the users,
some defects are noticed that can always be taken care of in next
versions. In Chapter 8, a simulator was developed to study the
operational characteristics of the requirement implementation process
and defect removal process. In this simulator, incoming requirements
and defects are assumed to form a common queue. Requirements and
defects are separated and assigned to different teams for their
respective processing. System was modelled as a queuing system with
two parallel queues to be served by two different servers.
Requirements and defects initially form a common queue and then
they are categorized as requirements or defects depending upon their
characteristics. Requirements and defects are then handled by
different teams. Busy and idle times of both the teams can be studied
and depending upon that team size can be decided. Simulator can
also be used to study other operational characteristics like the time a
requirement or defect has to spend waiting before it is implemented/
removed and maximum length of the queues formed by requirements
and defects at each team.
Simulators developed in the present thesis work can be an asset to
the people working in Component Based Software Development while
selecting and composing/ integrating software components from
various component libraries’ for the development of Component Based
Software in lesser time, with lesser efforts and using lesser resources.
The simulators presented here can be used in development houses to
quickly compose and launch their software products in market in
order to compete with other players in market in this age where time
to market is very short.