CHAPTER 3

LITERATURE REVIEW

3.1 INTRODUCTION

The literature review assesses the past and current status of research work in the area of component based software development. There is relatively little information published on the component based software development and its quality improvement. The work done by the earlier researchers in this area are classified as follows:

i) Component Based Software Development

ii) Software Quality and Reliability

iii) Component Reliability

iv) Genetic Algorithm

v) Component Selection

vi) Component Assembly

vii) Software Quality Function Deployment

The literature of above said topics are surveyed and explained in the following sections.

3.2 COMPONENT BASED SOFTWARE DEVELOPMENT

Component based software development is a process that aims to design and construct software systems using reusable software components. It
focuses on reusing and adapting existing components as opposed to just coding in a particular style. CBSD encourages the composing of software systems as opposed to programming them (Brown and Wallnau 1998).

Componentizing software had been suggested by McLlory (1969) as a way of tackling the software crisis, yet only in the last decade or so has the idea of component-based software development taken off. Nowadays there is an increasing market place for COTS components, embodying a buy, don't build approach to software development (Brooks 1987). The promise of CBSD is not only reduction in development costs but also component systems are flexible and easy to maintain due to the intended plug-and-play nature of components.

CBSD has many advantages. These include more effective management of complexity, reduced time to market, increased productivity, improved quality, a greater degree of consistency and a wider range of usability (Brown 2000).

Heineman and Council (2001) suggested that the major goals of CBSD are the provision of support for the development of systems as assemblies of components, the development of components as reusable entities, and the maintenance and upgrading of systems by customizing and replacing their components.

The reuse approach to software development has been used for many years. However, the recent emergence of new technologies has significantly increased the possibilities of building systems and applications in various ways from reusable components. Both customers and suppliers have had great expectations from CBSD, but their expectations have not always been satisfied. Experience has shown that component-based
development requires a systematic approach to and focus on the component aspects of software development (Crnkovic and Larsson 2000).

3.3 COMPONENT SELECTION

Various issues and problems associated with the selection of suitable reusable components have been addressed by the reuse community. In many organizations the selection process is typically not defined in a proper way. Each project finds its own approach to it, often under shorter schedule and budget pressure. This leads to deterioration of the quality and reliability. So the selection of the right reusable component is often a non-trivial task and requires careful consideration of multiple criteria and careful balancing among application requirements, technical characteristics and financial issues.

COTS systems need to be customized with respect to individual requirements and as a consequence, the proper selection of COTS components is critical for subsequent phases of the software development life cycle. Inefficient decisions not only affect the correctness, quality and dependability of a COTS based application, but they may also cause substantial cost increases with respect to the development process and for future maintenance activities (Ruhe 2002, Ruhe 2003).

Kontio et al (1995) proposed the Off-The-Shelf Option (OTSO) method which takes into account of requirements specification, organizational infrastructure, application architecture, project objectives and availability of libraries. System development using Off-The-Shelf is carried out in six phases, namely: COTS alternatives, searching, screening, evaluation, analysis, deployment and assessment. During the evaluation phase, this method compares COTS alternatives according to their functional and non functional
requirements. The Analytic Hierarchy Process is used to consolidate the evaluation results for decision making.

Ncube (1999) described a method to identify the software component which is called as Procurement-Oriented Requirements Engineering (PORE). This PORE method guides a software development team to acquire customer requirements and select COTS components which satisfy these requirements. It supports an iterative process of requirements acquisition and COTS software selection.

Alves (2003) introduced COTS based Requirements Engineering (CRE) which is an iterative COTS selection method that selects COTS by rejection. It considers domain coverage, time restriction, cost rating and vendor guarantees in its evaluation process. The key feature of CRE is the definition and analysis of non-functional requirements.

COTS-Aware Requirements Engineering (CARE) process uses soft-goals for the purposes of trading-off requirements for using COTS. It uses elimination method by analyzing the level of tradeoff required to use a particular COTS. The method assumes that COTS candidates already exist as the system requirements are under development. In addition, the method requires an enormous amount of effort and coordination between stakeholders during the analysis stage (Chung and Cooper 2003).

In the Comparative Evaluation Process (CEP) COTS selection method addresses both the functional and quality aspects of COTS. Moreover the method includes organization policy and credibility of sources of information in the selection process. CEP COTS evaluation model uses weighted average as well as a Multi-Criteria Decision Making technique to estimate the performance of the products (Cavanaugh and Polen 2002).
Another work presented by Kunda and Brooks (1999) described the STACE (Social-Technical Approach to COTS evaluation) Framework; it is an approach that emphasizes social and organizational issues to COTS selection process. The main limitation of this approach is the lack of a well-defined process of requirements acquisition/modeling. Moreover, the STACE does not provide a systematic analysis of the evaluated products using a decision-making technique.

A commonly used alternative is the analytic hierarchy process which includes a method for determining weights and component scores against attributes (Maiden and Ncube 1998, Kunda 2003). These scores are based on pair-wise comparisons and thus use the same units even when combining qualitative and quantitative data.

Baker et al (2006) present two algorithms for component selection and prioritization for the next release problem where components will determine special, additional features or functions of the next release in the evolution of the software.

In general COTS component selection needs to be more flexible on requirements as an exact match may not be possible and so relaxation of criteria is required (Cechich et al 2003).

Given a set of attributes, the ideal specification (requirements) and the values presented by a particular component, the selection process needs to combine the data to create a ranking or recommendation. Much of the literature uses the Weighted Score Method (WSM) to aggregate a value by summing attribute weights multiplied by their respective values (Williams 1992, Solberg and Dahl 2001). Both qualitative and quantitative factors are easily handled in this method and different scale can be used for different factors. Even though it is a straight forward process, an optimized approach is
needed to find the right component along with reducing time and effort through automation.

The above literature review shows the importance of software component selection and evaluation in CBSD process in the development stage. These concepts are considered in the development of selection process in this research which is explained in chapter 4.

3.4 COMPONENT ASSEMBLY

The integration of COTS software into highly complex software systems introduces many important issues. The primary issue is the conflict between cost and complexity in software integration. The consumers who want to use these systems want reliable software system within minimum budget and schedule is also the important issue in component based software. In the literatures, several methods have been proposed for dealing with COTS component assembly. In all of them a key point is the ability to assemble the components with minimum cost and schedule.

Many authors have discussed about the difference between software component assembly and hardware component assembly. Component based software development can help the software industry to realize the quality and productivity improvement similar to those achieved in electronic and mechanical component assemblies (Brown 2000, Szyperski 2003).

Many of the problems encountered when integrating COTS components cannot be determined before integration begins. Thus estimating development schedules and resource requirements is extremely difficult (Vidger et al 1996).
Voas (1998) described that the component based software development approach can potentially be used to reduce software development costs, increase flexibility, assemble systems rapidly and reduce the spiral maintenance burden associated with the support and upgrade of large systems.

Vitharana et al (2003) suggested that the weightage must be given to the cost effectiveness with the ease of assembly model in software industries when the business strategy is low cost and component based development. He also pointed that the design models with quantified technical features support the component developer’s business strategy. Software architecture based approach in which the architectural analysis and code synthesis are combined together in order to efficiently and correctly assemble a system out of a set of already implemented components.

Fenton and Neil (1999) suggested that engineers can benefit from Bayesian Belief Networks (BBN) without any need for specialized education in statistics and probability. BBNs can be applied to model candidate software system architectures. Different COTS software selection and integrations can be modeled as different instances of BBNs and changes to the architecture are modeled as simple changes to BBN. Using COTS component software system architecture, it can be modeled and diagnose the problems in this architecture and use of these emergent properties to test the architectures compliance to measurable system requirements through scenario based approach. But this work doesn’t focus elaborately about software assembly.

Inverardi and Tivoli (2003) have proposed an approach to the integration problem in the CBSD setting is to compose system by assuming a well defined architecture style in such a way that it is possible to detect the integration anomalies.
Nunn and Dengo (2002) have described a component architecture which is defined by an XML schema that specifies the composition of components into a persistent, suitable and composite application.

Bucchiarone et al (2006) have proposed a software architecture based approach in which architectural analysis and code synthesis are combined together in order to assemble a system out of a set of already implemented components. More importantly for integration process, it should be possible not only to provide an assembly guide, but also to predict or calculate the qualities of an assembly from those of its constituent components.

According to the past literatures most of the assembly procedures are using the software architecture for integration. But there is no step by step procedure to know about the order of assembly of components which makes perfect assembly in COTS system. A proper methodology is needed to make high quality software for present software industries.

3.5 SOFTWARE QUALITY AND RELIABILITY

Generally, software quality is conformance to requirements because if the software contains too many functional defects, the basic requirement of providing the desired function is not met. To increase the software quality and preventing software errors, the focus must be on comprehensive requirements and software testing. Software testing is the basic requirement for software reliability engineering (Whittaker 2000). Software reliability engineering is a keystone to total quality management. It provides a user oriented metric that is correlated with customer satisfaction. It cannot determine reliability of software based systems without its associated software reliability measure (Musa 2005). Software reliability engineering is based on a solid body of theory (Musa et al 1987) that includes operational profiles, random process
software reliability models, statistical estimation, and sequential sampling theory. Software personnel have practiced software reliability engineering extensively over a period dating back to 1973 (Musa and Iannino 1991). But still it is under research because of new technologies emerging in software based systems.

Musa et al (1987) presented exponential distribution for reliability estimation model for assessing the reliability of individual components based on its field failure intensity.

Osterweil et al (1996) concluded in their paper as “The quality of software products is clearly unacceptably poor. A large and challenging program of long-term research is needed to support developing the technologies needed”.

Actually, the literature is jammed to overflowing software reliability models, so the practitioner should be aware of selecting models for a particular application, which may be quite suited to the situation. The two most important criteria for selecting a software reliability model and estimation method for use in tracking reliability growth are model simplicity and maturity. The model which is used to predict the reliability should be in simple understandable format by the software engineers.

3.6 COMPONENT RELIABILITY

CBSD is touted as the approach to improve application qualities, decrease time to market, and improving maintainability and reliability. As a result, many techniques have emerged to estimate and analyze the reliability of component. Few Models are available for assessing the reliability of individual components also.
Dolbec and Shepard (1995) provided a model which estimates the reliability of CBS system based upon component usage ratio. The component severity analysis along with this model helps to focus testing effort in areas where best potential reliability improvements can be achieved. This leads to savings of time and money.

Krishnamurthy and Mathur (1997) assessed the reliability of component based software system. Their approach is based on the test information and test cases. For each test case the execution path is identified, the path reliability is calculated using the reliability of the components assuming a series connection. The reliability of the application is the average estimate of the reliability of each test path. This approach does not consider component interface faults.

Hamlet et al (2001) presented a theory which describes how a component developer can design and test their components to produce measurements that are later used by system designer to assess the composite system reliability.

May (2002) derived a new model that describes how test based software reliability estimation depends on the component structure of the code. The models show how component test results can be re-used and how this re-use can provide failure probability estimation for software systems built from COTS components.

Yacoub et al (2004) proposed a new technique called scenario based reliability estimation which builds on scenarios used in the analysis phase of component based system development. In this approach, component dependency are derived to establish a probabilistic model on which the reliability estimation technique is developed. This algorithm used to identify critical components, its interfaces and links to estimate the system reliability.
Shukla et al. (2005) proposed a theoretical framework for assessing the reliability of component-based software systems that incorporates test case execution and output evaluation. The framework requires the user of the component to define an operational profile for the component and appropriate reliability parameters.

Comparing the factors of simplicity, generality, and applicability, the software reliability model proposed by Dolbec and Shepard (1995) is considered as a favorable model which focuses on the heavily used component in a component-based software system to improve reliability.

3.7 SOFTWARE QUALITY FUNCTION DEPLOYMENT

Software process improvement has received attention from more and more software development organizations nowadays. It is a common practice for an organization, especially a large organization, to select a software engineering standard and model for its process improvement. Quality Function Deployment is a technique used in product and process design to identify the priorities for customer needs and expectations and to transfer these priorities into the technical design of the product. The benefits of such an approach include such deliverables as increased coverage of explicit and implicit requirements; improved acceptance of product; establishment of priority product characteristics to support design, development, and testing; and increased efficiency of product development.

Lamia (1995) from Carnegie Mellon University gave an overview of integrating QFD with object-oriented software design methodologies. This article puts forth the proposition that QFD methods for collecting information about the users of a system, the functions performed by the system, and the quality demands the users expect of the system can be very effective in the initial stages of object-oriented analysis. A number of QFD-based tabular
forms are proposed in this article which parallel the graphical diagrams used in Object Oriented Analysis (OOA). He concludes with some suggestions for opportunities to integrate QFD and OO automated support tools.

Haag et al (1996) presented the adaptation and use of quality function deployment for software development. They concluded that it is significant that all of the organizations that utilize Software Quality Function Deployment (SQFD) also use quality policies based on Total Quality Management (TQM). They identified that the use of SQFD as a requirements-gathering tool in Software Development Life Cycle (SDLC) and that they expected SQFD use will increase in future.

Richardson (1997) presented the importance of improving software process to small software development companies. He has also presented the difficulties faced by small companies when implementing such improvements. The author presented a model in his paper, based on Quality Function Deployment, which can be used as tool to aid the implementation of a software process improvement action plan, and discussed the use of this model in a small software development company in Ireland.

Liu et al (2005) applied QFD in software process management and improvement based on CMM. They have suggested a framework for software process improvement using QFD. In that, the considered four phases are requirement integration, CMM goal prioritization, Key practice prioritization and Action plan prioritization. The advantage of this framework is that the requirements from various perspectives are incorporated into the final set of actions.

So, quality function deployment is a process used to determine product development characteristics that combine technical requirements with customer preferences. Using an integrated matrix known as the "House of
Quality", QFD considers the different influences bearing on the design to promote concurrent engineering, resulting in increased product acceptance. The basic QFD methodology can also be utilized with common software quality considerations to create a hybrid software requirements elicitation model. According to the survey of literature, The House of Quality can be applied to software design, and the resulting software requirements are diverse in their scope and coverage. The result is that product acceptance extends beyond basic functionality to serve as an indicator of reliability, usability and other customer preferences along with design considerations.

3.8 CONCLUSION

Based on the detailed literature review carried out for component based software engineering, it can be inferred that although some papers touched the selection, assembly and quality improvement of software components, no one integrates in the following areas.

i) Study of behavior of software components in component based software system.

ii) Identification of proper methodology for the selection of components.

iii) Improved component assembly process by considering the interaction between the components.

iv) Analysis of the reliability of component based software system.

v) Quality improvement based on user requirement specifications.

Hence, it has been decided to carryout the research focusing all of these aspects in component based software system based on the literature review.