A Systematic Research Survey on Component Based Quality Assurance

Like any engineering activity, software engineering paradigm also has main objective to produce high quality software. To achieve these goals, component developers as well as integrators must apply effective method to ensure the quality. This chapter reviews several journal articles and conference papers on component based software quality attributes and quality assurance to evaluate the progress and direct future research on this software engineering problem. This survey does not describe all these quality assurance models for practitioners in detail. Our aim is to classify studies with respect to metrics and methods that have been used in these papers.

Component Based Software Engineering primarily focus on assembling pre-built components to build a software system, with a potential benefit of delivering quality systems by using quality components[Brown1998]. It is different from the traditional software engineering process which software is developed from scratch. Component based software paradigm is integration centric. The quality of a component based system using high quality components does not provide guarantee
a high quality system, but depends on the quality of its components, a framework and integration process used. Hence, techniques and methods for quality assurance and assessment of a component-based system would be different from traditional software engineering methodology. It is essential to quantify factors that contribute to the overall quality, for instances, the tradeoff between cost and quality of a component, analytical techniques and formal methods, quality attribute definitions and measurements.

Studies show that component-based systems that are being built recently are exceeding 70% of the total developed software systems. Therefore, a model that ensures quality characteristics of such systems becomes a necessity. Among the most critical processes in component-based software are the evaluation and selection of the components. Software quality is an important practical issue which deals with the question of how efficiently, elegantly and correctly a software program accomplishes certain task. More specifically, software quality is thought in terms of conformance to requirements and absence of faults. To ensure quality, planning in this discipline plays significant role. There are various teams to ensure quality like software quality assurance, software quality control and software quality engineering.

3.1 Scope and Goal

This study presents a literature survey of component-based system quality assurance and assessment, the areas survey include formalism, cost estimation, and assessment and measurement techniques for the following quality attributes: reliability, security, maintainability and testability.
3.2 **A Preamble**

Through the review of survey literature on quality estimation in the early design phase of component based software life cycle, various techniques have also been discussed in detail. Quality attributes of a component are not only determined by the component itself, but are influenced by the usage model, the deployment environment, its internal structure, and the services used by the component [Goulao 2002]. In most cases, it is unclear how the information about all these factors is obtained and integrated. This is due to a lack of distinction concerning the roles and responsibilities during the development process.

Software integrators are still asking how they can measure the quality of their component based software before it is used despite the substantial research effort spent attempting to find an answer to this question over the last twenty years. There are many papers advocating statistical models and metrics which purport to answer the quality question. Defects, like quality, can be defined in many different ways but are more commonly defined as deviations from specifications or expectations which might lead to failures in operation.

Generally, efforts have tended to concentrate on the following four problem perspectives:

1) Estimating the number of faults in the system.

2) Estimating the reliability of the component as well as system.

3) Impact of design and testing framework.
4) Security characteristic of component.

Several component-based quality assurance approaches have been proposed [Gao 2003]. Most of these approaches focus on the analysis part and only contain very brief descriptions on how they are going to be integrated into the development process.

For example, the component-based reliability, performance and safety prediction approaches consider a pure bottom-up development where already existing components are assembled. This is a strong restriction, since combined top-down (starting from requirements) and bottom-up (starting from existing components) approaches are more realistic.

Since we want to evaluate quality attributes at an early development stage, we need additional information about the internal component structure, the usage model, and the deployment environment. All these information cannot be given by system architects themselves. Therefore, support of domain experts, component developers, and system developers is required.

A component developer cannot know how the component will be used and so cannot certify it for an arbitrary use; but if the component buyer must certify each component before using it, component-based development loses much of its appeal.

This dilemma is resolved if the component developer does the certification and provides the results in such a way that the component buyer can utilize that information, without having to repeat the certification.
3.3 Quality Assurance attributes

There are numbers of quality assurance attributes, which directly or indirectly effect component based system quality i.e, reliability, security, maintainability, architecture, environment, interface etc [Sedigh-Ali 2001].

Software reliability is an important metric in determining overall system stability and quality, through error prevention, fault detection and removal.

3.3.1 Component Based Software Reliability

Software reliability is a quantitative measure of software quality. It is defined as a probability of failure free execution given a specific environment and a fixed time interval.

Software reliability can be defined as the probability that the system will perform its intended functionality under specified design limits. Software reliability techniques are aimed at reducing or eliminating failures in software systems [Musa 1987].

The aim of software reliability engineering is to model the failure behavior of component based software systems to estimate the project reliability. Reliability can be defined as “the ability of a system or component to perform its required functions under stated conditions for a specific period of time”. The reliability measurement of component based system comprises two main activities. First, the reliability measurement of individual component and secondly, the reliability measurement of the whole system based on the component context model [Stafford2002].

---

*Putting the Pieces Together In Reference Of Component Based Software Engineering*  54
There are two activities related to software reliability analysis: estimation and prediction. Estimation is retrospective and it is performed to determine achieved reliability from a point in the past to the present time, whereas the prediction activity parameterizes reliability models used for estimation and utilizes the available data to predict future reliability [Musa1992].

Dolbec proposed model provides software system reliability estimates from the reliability of software components and the usage ratio of each component [Dolbec 1995]. Dolbec model assumes that software components have high reliability, the component failures are independent, and the execution path failures are independent. Its proposes that given the total system execution time ($T_s$) and total component execution time ($t_k$), the usage ratio of the components can be determined as:

$$\phi_k = t_k / T_s$$  \hspace{1cm} (3.1)

And software system reliability is presented as

$$R_s = 1 - Q_s \approx 1 - \sum_{k=1}^{m} \phi_k D_k$$  \hspace{1cm} (3.2)

where $R_s$ is software reliability, $Q_s$ software unreliability, $m$ is number of components, $D$ represents probability of failure of component and $F$ represents the usage ratio of component. The calculation of component execution time, dependence between software components and quantifying the impact of component dependence on the overall software reliability and impact analysis are still potential research areas. Component Based Reliability Estimation (CBRE) technique using reliability of the components is proposed, that is based on test information and test cases.
The approach does not consider component interface faults, although they are important factors in reliability analysis of component-based systems.

Shukla proposes a conceptual framework for the reliability assessment of software components that incorporates test case execution and output evaluation [Shukla2004].

Cortellessa proposed an analytic model which includes the error propagation phenomenon in reliability measurement [Cortellessa2007].

Hamlet et al. address how component developers can design and test their components to produce measurements that are later used by system designers to calculate composite system reliability — without implementation and test of the system being designed. The theory describes how to make component measurements that are independent of operational profiles, and how to incorporate the overall system-level operational profile into the system reliability calculations [Hamlet 2001].

Gran describes how a Bayesian Belief Network for a software safety standard, can be merged with a BBN on the reliability estimation of software systems [Gran2001]. Xiaoguang et al. presents a general model — component probability transition diagram, which is compatible with different kinds of components and enables reliability tracing through component-based software processes.

Xiaoguang proposes a general reliability model that is based on the assumption that all components reused in component-based software are third-party ones and glue logic is entirely reliable [Xiaoguang2003]. The software reliability is calculated as
Where \( R_{pi} \) is reliability of path \( P_i \), \( F_{pi} \) is occurrence frequency of path \( P_i \).

Zhu propose a method to calculate the reliability of component based software using data sheet of COMPONENT, partitioning every component’s input into sub-domains. The system reliability is calculated based on the relation of components using Markov chain [Zhu2003]. Zhu et al. propose that suppose \( D \) is an input domain of every component, \( f_i \) is the failure rate of every sub-domain \( D_i \), \( h_i (\sum h_i = 1) \) is the probability that each sub-domain \( D_i \) will occur, the reliability can be calculated as

\[
R = \frac{\sum_{i=1}^{n} (R_{pi} \times F_{pi})}{\sum_{i=1}^{n} F_{pi}}  \tag{3.3}
\]

The unanswered question from this approach is how to partition sub-domain for every component Probabilistic models are used as a base to calculate the reliability of the component based systems. Yacoub et al. presents one of such techniques named Scenario-Based Reliability Analysis (SBRA). Using scenario, a probabilistic model named “Component Dependency Graph (CDG)” is constructed. CDG are directed graphs that represent components, component reliabilities, link and interface reliabilities, transitions and transition probabilities [Yacoub1999]. This technique addresses the issue of calculating reliability of a component based system with the assumption that estimates of the components are available. Its comprises
three steps (a) estimation of the parameters used in the reliability model, construction of the component dependency graph and application of the algorithm for reliability analysis.

Wohlin presents a method proposal for estimation of software reliability before the implementation phase. The method is based upon that a formal description technique is used and that it is possible to develop a tool performing dynamic analysis, i.e. locating semantic faults in the design. The analysis is performed with both applying a usage profile as input as well as doing a full analysis, i.e. locate all faults that the tool can find. The tool must provide failure data in terms of time since the last failure was detected. The mapping of the dynamic failures to the failures encountered during statistical usage testing and operation is discussed. The method can be applied either on the software specification or as a step in the development process by applying it on the design descriptions. The proposed method will allow for software reliability estimations that can be used both as a quality indicator, but also for planning and controlling resources, development times etc. at an early stage in the development of software systems [Wohlin1992].

Singh et al., proposed a reliability prediction algorithm to analyze component reliability of the system before it is built, taking into account component reliability estimates and their anticipated usage [Singh2001]. The analysis framework is fully integrated with UML models. It incorporate information about the expect system usage patterns and failure probabilities of the individual components. It can be used in the early phase of system design, as soon as the properly annotated Use Case
Diagrams and Sequence Diagram become available. They provide result as a Histogram including Calculation of System Failure Probability and 95% Confidence Interval. Existing approaches for component based software reliability modeling largely rely on the availability of implementation-level artifacts.

Wang et. al., provide a moving average non-homogeneous Poisson process (MA NHPP) reliability model which includes the benefits of both time domain, and structure based approaches. This method overcomes the deficiency of existing NHPP techniques that fall short of addressing repair, and internal system structures simultaneously. Our solution adopts a MA approach to cover both methods, and is expected to improve reliability prediction. This paradigm allows software components to vary in nature, and can account for system structures due to its ability to integrate individual component reliabilities on an execution path. Component-level modeling supports sensitivity analysis to guide future upgrades, and updates. Moreover, the integration capability is a benefit for incremental software development, meaning only the affected portion needs to be re-evaluated instead of the entire package, facilitating software evolution to a higher extent than with other methods [Wang2007].

Zhong propose a Petri Net based approach to predict the reliability of web service composition. The first step of this approach involves the transformation of web service composition specification into Stochastic Petri Nets (SPN) model. From the SPN model, they derive the reliability and performance measure of web service composition [Zhong2006].

---

*Putting the Pieces Together In Reference Of Component Based Software Engineering* 59
Govseva et. al., propose a methodology for uncertainty analysis of architecture-based software reliability models suitable for large complex component based applications and applicable throughout the software life cycle. First, it describes different approaches to build the architecture based software reliability model and to estimate parameters. Then, they perform uncertainty analysis using the method of moments and Monte Carlo simulation which enable us to study how the uncertainty of parameters propagates in the reliability estimate [Govseva2003].

Cheung et. al. address early prediction of component reliability because of many uncertainties associated with components under development. These challenges in developing a software component reliability prediction framework [Cheung2008].

McGregor et. al. reports on an investigation of the feasibility of using design constructs as a means of treating several methods as a single unit, exploring how to provide useful information about reliability to acquirers of components. Rather than provide a single value for the entire component, we provide reliability information about each role that the component is intended to support. The acquirer can then compute the effective reliability they would experience given their intended use of the component. The intention is to provide accurate information about reliability in support of component commerce and prediction of assembly reliability [McGregor 2001].

Wang et.al., introduces a moving average reliability growth model to describe the evolution of component-based software. In their propose model, the reliability of a system is a function of the reliabilities of its constituent components. The moving
average provides a trend indicator to depict reliability growth movement within the evolution of a series of component enhancements. The moving average can reduce the effects of bias or measurement error of certain components by rendering a smoothed trend of system reliability growth [Wang2007].

Krishnamurthy propose Component Based Reliabilities Estimation (CBRE), for the estimation of reliability of a software system using reliabilities of its components. CBRE involves computing path reliability estimates based on the sequence of components executed for each test input. Path reliability estimates are averaged over all test runs to obtain an estimate of the system reliability [Krishnamurthy1997].

Hu presents a new approach to evaluate the reliability of the component-based software system in this open distributed environment by analyzing the reliabilities of the components in different application domains, the reliabilities of the connections to these components and the architecture style of their composition [Hu2007].

Lo et. al., propose a new approach to analyzing the reliability of the system, based on the reliabilities of the individual components and the architecture of the system. They also present the sensitivity analysis on the reliability of component-based software in order to determine which of the components affects the reliability of the system most [Lo2005].

Gokhale proposed genetic algorithm based approach to select components that maximize the system reliability while meeting the cost constraint. In a realistic situation, where the system comprises hundreds of components, each with several reliability/cost pairs, or even a continuous cost/reliability relation, exhaustive enumeration is impractical. A genetic
algorithm can however be used to maximize the system reliability under cost constraints [Gokhale1999].

Musa provides an overview of a step-by-step approach for developing the operational profile used in software reliability engineering [Musa1992].

Everett describes an approach to analyzing software reliability using component analysis. It walks through a 6-step procedure for performing software component reliability analysis. The analysis can begin prior to testing the software and can help in selecting testing strategies. It uses the Extended Execution Time (EET) reliability growth model at the software component level [Everett1999].

Gokhale et. al., demonstrate the flexibility offered by discrete-event simulation to analyze such complex systems through two case studies, one of a terminating application, and the other of a real-time application with feedback control. It simulates the failure behavior of the terminating application with instantaneous as well as explicit repair. They also present the effect of having fault-tolerant configurations for some of the components on the failure behavior of the application. In the second case of the real time application, we initially simulate the failure behavior of a single version taking into account its reliability growth [Gokhale1998].

The other important attribute of quality assurance is security in component based software development paradigm.

3.3.2 Component Based Systems Security:

Security is a very important concern for component based software system and atomic components. With the use of Component Based Software Systems (CBSS) in
many critical applications security has now become a hot topic for research community. The security in component based systems has wide scope. Security can be defined as a capability of the component to protect information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems cannot be denied to access them.

Measuring the security of a component based software systems is a difficult problem. The present work argues that the security properties of a composite system can be viewed either from the end-user's point of view, or from the software integrator’s point of view. In component based software engineering security cannot be defined as a binary logic. It means systems are neither secure nor in secure. Security is often described in relative context. A system is more, or less secure than something else. A component may be proved secure in one application in a particular operating environment, but the same component may not be considered secure at all in a completely different application. One of the measure problem in component based software systems, security is actually expressed and measured in different ways and therefore, it is hard to compare and trace it.

Khan proposes a security characterization structure of software components and their composition. The structure provides a preliminary modeling of security properties of stand-alone software components and some of their compositional primitives. It was particularly interested in security properties related to user data protection of software components. The proposed compositional specification makes an attempt to model the resulting effect between security attributes of two contracting components. The
compositional specification structure can capture the results of combined security specifications of two participating components in a contract. Its security specification syntax is based on four compositional elements: identities of contracting components, actions to be performed in a compositional relationship, security attributes supported by components, and resources to be used by other components [Khan2003].

Khan presents a framework for constructing compositional security contracts (CsC) based on the security property exposed by the atomic component. The framework uses interface structure of components in order to determine the CsC of software components. An active interface provides the component a basis for reasoning and assessing a component's suitability to meet certain security requirements of a particular application. Based on the security information available from the component interface, an active interface can reason whether the candidate component meets the security requirements for an envisaged system wide application [Khan 2001].

Any security mismatches or discrepancies between components can be identified by the participating components before an actual composition takes place. Khan describes Component security concerns are twofold: how to build secure components and secure composite systems from components, and how to disclose components' security properties to others. They present a security characterization framework. Its approach concerns the security functions of software components by exposing their required and ensured security properties. Through a compositional security contract between participating components, system integrators can reason about the security
effect of one component on another. A CSC is based on the degree of conformity between the required security properties of one component and the ensured security properties of another [Khan2002].

Jeong develops a certification process for testing software components for security properties. The anticipated results from this paper are a process, set of core white-box and black-box testing technologies to certify the security of software components and a framework for constructing compositional Component Security Assurance (CSA) based on the security property exposed by the atomic components [Jeong2005].

Huang et. al., propose a black-box testing framework for automated Web application security assessment [Huang2005].

Hangkon Kim[2004] investigate the fundamental issues related to building and composing secure components. The approach outlined to develops a certification process for testing software components for security properties.

Khan explores how to characterize security properties of software components, and how to reason about their suitability for a trustworthy compositional contract. Khan uses logic programming as a tool to represent security properties of atomic components and reason about their compositional matching with other components [Khan 2003].

Khan proposes an approach of defining systems level security properties of component-based composite systems. It argues that the security properties of a composite system can be viewed either from the end-users' point of view, or from the
software integrators’ point of view. End users look more for the ultimate security goals achieved in the composite system, whereas software integrators are more interested in the compositional security properties of the system in terms of the required and ensured properties. Software integrators need to know how a composite system could be assembled further as a coarse-grained component with other applications. It is equally important for the end user of the system to know the actual security objectives achieved at the systems-level [Khan 2005].

Schmidt et. al. defines key requirements for an architecture-based approach to trustworthy components. They also provide a brief overview architecture definition language RADL with a focus on compositionality and extra-functional properties. In RADL are oriented towards modern middleware technologies such as .NET and EJB and to software-engineering methods such as UML. RADL dynamic models are centered around contracts, state machines and Petri nets. These are associated to contact points and connectors for defining connection constraints in architectural specification. They define configuration and behavioral contracts when they are associated to components and architectural assemblies of components. RADL contracts permit static compatibility checks and automatic gate adaptation for true black-box reuse. Dynamic monitoring of deployed components complements this with execution-based mechanism enabling prediction of extra-functional properties during architectural design [Schmidt 2002].

Schmidt et al., attempt for explain the requirement of trusted component suitable for formal analysis and partial automatic synthesis [Schmidt 2000].
Ren et. al., argues for a more comprehensive treatment based on software connectors. Connectors provide a suitable vehicle to model, capture, and enforce security. Its approach models security principal, privilege, trust, and context of architectural constituents. Extending our existing architecture description language and support tools, our approach can facilitate describing the security characteristics of an architecture generating enabling infrastructure, and monitoring run-time conformance [Ren 2005].

Meyer describe Reusable components equipped with strict guarantees of quality can help reestablish software development on a stronger footing, by taking advantage of the scaling effect of reuse to justify the extra effort of ensuring impeccable quality [Meyer2003].

Zhong developed a security method that mitigates the risk of using COTS components. It's based on the sandbox model, originally developed for fault tolerance. Rather than eliminating actual failures— which would require accessing COTS component source code— our method provides a restricted environment to confine application behavior, which confines the damage caused if an application accidentally or maliciously misbehaves. This method requires B-level security features not found on most conventional OSs. B-level trusted Oss are defined in the Trusted Computer System Evaluation Criteria3 (TCSEC), developed by the US Department of Defense [Zhong1998].

Han consider one particular quality-of-service attribute, i.e., security, and outline an approach to (1) specifying the security characteristics of software components and (2)
analyzing the security properties of component-based systems in terms of their
close contact characteristics and system architectures. The approach is partially based
on the Common Criteria for Information Technology Security Evaluation (TSO/IEC
International Standard 15408). They also introduce ensuring the integrity of software
components as part of the infrastructural support for component based software
engineering [Han2000].

Masood et. al., describe the use of components can introduce various security and
reliability risks in the system. They propose a methodology for efficient management
of all the system security concerns involved in the design of component based
systems. This methodology is based on formally representing the system security
specifications and component capabilities. It’s identified the metrics for correlating
both and suggests extensions to a previously proposed software development process,
for selection of suitable components and integration mechanisms. The proposed
solution ensures due treatment of all the security concerns for the complete system in
the acquisition efforts [Masood2005].

Kelkar et. al., describe Components based software integration can introduce
security vulnerabilities due to mismatches between security constraints coupled with
inadequate knowledge of interaction requirements. Though a component can be
validated against its stand-alone functional and security requirements, two aspects of
the validation for its integration are missing. First, no straightforward process exists
to guide the developer in identifying integration-induced security risks. Second,
interaction properties contributing security risks are not part of COTS product evaluation [Kelkar2006].

Puntigam clarifies the trust in components just to some degree. Component users apply a number of methods to improve protection against malicious components. He briefly analyzes some of these concepts and their relationships to our trust in components. It turns out that weak protection methods can be beneficial for components we partially trust especially if potential holes are clearly visible. Visible holes build a basis for extending responsibility from the component user to the component supplier [Puntigam2003].

Gousios et. al., examines the security threats those systems must confront and the solutions proposed by the major existing component architectures. A comparative evaluation of both security features and implementation issues is carried out to determine each architecture's strong points and drawbacks [Gousios2005].

For validation and verification of quality assurance in component based software systems we need some mechanism that combined known as testing.

3.3.3 Component Based Testability:

Quality assurance, including testing, is conducted in development and use of a component can be considered according to the two distinct perspectives. These perspectives are the component developers and integrators. The component provider corresponds to the role of the developer of a component and the component user to that of a client of the component provider, thus to that of the developer of a system using the component. The "component trust problem" refers to the adequate
guarantees and documentation about a component's behavior which can be transferred from the component developer to its potential users. The capability to test a component when deployed in the target application environment can help to establish the compliance of a candidate component to the customer's expectations and certainly contributes to "increase trust".

The activity of a program is normal when it behaves according to its functional specification. When the program presents deviations from its functional specification, its activity is said to be abnormal or exceptional, since it is expected that these deviations occur only rarely. The use of component based development in the development of critical systems highlights the importance of considering the exceptional activity and the overall system quality through validation techniques. Testing activities, most of which can be automated, are distributed amongst all development phases in order to improve the reliability of the produced system. There are two main activities: testability improvement by embedding built-in testing capabilities into the component under test, and test cases generation. All the testing artifacts can be reused every time when the component is tested: during its development or every time it is reused.

The move toward component-based software development offers many promises for improved productivity and quality, but it also highlights the need for effective methods of testing reusable software parts. Indeed, components that are independently developed or commercially purchased underscore the need to detect errors well before system integration time. The term verification refers to the process
of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase, while validation is the process of evaluating software at the end of the software development process to ensure compliance with software requirements [IEEE1990]. Verification is intended to ensure that the product is built correctly, while validation assures that the correct product is built. Maintainability [IEEE1991] is “The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment”. Effective maintenance involves detailed observations of the behavior of a system and is driven by software complexity.

Addy describes a framework that extends for performing verification and validation in reuse-based software engineering. It includes descriptions of the types of activities to be performed during each of the life-cycle phases, and provides motivation for the activities [Addy1998].

Wang describe types of software maintenance, classification, and scopes of software components in component-based software maintenance are analyzed. A new kind of built-in test (BIT) components for maintainable software is developed. Based on this, a test/maintenance mode of maintainable software is proposed [Wang1998]

Harrold briefly explain the state of the art in software testing, outlines some future directions in software testing, and gives some pointers to software testing resources [Harrold2000].
Edwards propose a general strategy for automated black-box testing of software components that includes: automatic generation of component test drivers, automatic generation of black-box test data, and automatic or semi-automatic generation of component wrappers that serve as test oracles [Edwards2001].

Beydeda find out a limited exchange of information between the developer and integrator of a component. A limited exchange and thereby a lack of information can have various consequences, among them the requirement to test a component prior to its integration into a software system. A lack of information cannot only make test prior to integration necessary, it can also complicate this tasks [Beydeda 2001].

Carney et al. argues that expanded purview of configuration management, justification and documentation of modification, and maintenance responsibility of vendor are the key principles of component based system development in order to have maintenance of component based system become possible. Understanding component behaviour during the evolution of components and applications is important as maintenance is often necessitated because of changes in components, changes of requirements and a host of other reasons[Carney2000].

Bertolino propose the CDT framework for Component Deployment Testing. CDT provides the customer with both a technique to early specify a deployment test suite and an environment for running and reusing the specified tests on any component implementation. The framework can also be used to deliver the component developer's test suite and to later re-execute it. The central feature of CDT
is the complete decoupling between the specification of the tests and the component implementation [Bertolino2003].

Brito, et. al., presents a systematic way to deal with exception handling, from the requirement specification phase to the implementation and testing phases, in component-based software development. Testing activities are performed since the early stages of development, promoting an increase in the quality of the produced system.

Shukla et. al., describe tool that supports the statistical testing of software components. The tool supports a wide range of operational profiles and test oracles for test case generation and output evaluation. The tool also generates appropriate values for different types of input parameters of operations. STSC automatically generates a test driver from an operational profile. This test driver invokes a test oracle that is implemented as a behaviour-checking version of the implementation [Shukla2005].

Voas provides an overview of the maintenance challenges raised by CBSD by identifying reasons including frozen functionality, incompatible upgrades, unreliable components and complex middleware. Number of software metrics measuring maintainability has been proposed by means of theoretical and empirical studies [Voas1998]. However, component based system presents a unique maintenance challenges

Mao et. al., present an improved regression testing method based on built in test design for component-based systems. It needs the mutual collaboration between
the component developers and users. Component developers are responsible for analyzing the affected methods and constructing the corresponding testing-interfaces in the new component version, and then component users can conveniently pick out the subset of test cases for regression testing with these testing-interfaces [Mao 2007].

Mariani et.al., try to solve both the problem of quickly identifying components that are syntactically compatible with the interface specifications, but badly integrate in target systems, and the problem of automatically generating regression test suites. They proposed technique to automatically generate compatibility and prioritized test suites is based on behavioral models that represent component interactions, and are automatically generated while executing the original test suites on previous versions of target systems [Mariani2007].

Speck et. al., introduces an approach to validating component compositions and showing how such a process can be supported by tools. They introduce a way to compare the interface specification of components automatically against the code [Speck 2005].

Memon describes techniques that can be employed by developers to build testable COTS components. A taxonomy of these techniques based on the phases of software testing and the role of the component developer/user is presented [Memon2004].

Fraga presents a component model for building distributed applications with fault-tolerance requirements. The AFT-CCM model selects the configuration of replicated services during execution time based on QoS requirements specified by the
user. The configuration is managed using a set of components that deal with the non-functional aspects of the application [Fraga2003].

Brito et. al., presents a systematic way to deal with exception handling, from the requirement specification phase to the implementation and testing phases, in component-based software development. They also propose methodology for the Definition of Exception Behavior, MDCE, in the architectural design, implementation, and testing phases [Brito2005].

Grunske et. al., propose a component based dependability analysis technique that annotates components with failure mode assumptions. The probabilities and dependencies of these failure modes are specified by Component Fault Trees (CFT’s) [Grunske2005].

Gergic proposes an approach to manage versioning in the context of CBSD with focus on identification and description of versioned entities which occur during the component software assembly and configuration phase. Version model supports evolution of versioned entities employing the concept of user defined attributes taxonomies and entity relations. The proposed version model also supports variants. It also reduces the versioning issues by moving the scope of version identification from the individual versioned entities to distribution packages [Gergic2003].

Kansomkeat et. al., propose byte code analysis technique. It increases component testability without requiring access to the source. A component’s byte code is analyzed to gather control and data flow information, which is then used to obtain definition and use information of method and class variables. Then, the
definition and use information is used to increase component testability during component integration testing [Kansomkeat2005].

Unlike the traditional software systems, maintenance or testing cannot be done by viewing or changing the source codes of the component, but are restricted to reconfiguring and reintegrating components. The first decision that needs to be made here is to determine the appropriate level of activities to apply the maintainability process of component based system. These activities range from component replacement to troubleshooting and configuration management. The key maintenance activities for the component based system include component reconfiguration, configuration management, system tailoring, system monitoring, troubleshooting and repair. The maintenance of the system requires that the system posses a set of properties that target maintenance activities. The desired properties are to have consistent architectural style, ease of tailoring, control of component interface, use of open standards, minimize coupling and encapsulation of component collaborations [Mahmood2005].

Arsanjani et al. addresses the issue of software maintenance of component based systems by identifying encapsulating and externalizing the variations around design decisions. The approach is based on the notation of Enterprise Component (EC). EC is defined as an architecture patterns that provide a uniform mechanism for management of component boundaries between systems. The process includes the identification of requirements for the manners of a system and component, formalizing and abstracting them into a domain-specific language’s grammar. This
approach enables a highly re-configurable architectural style to help build and maintain reusable components that are responsive and resilient to changing requirements[Arsanjani2002].

Ardimento et al. reports the results of empirical study aimed at understanding how characterization of components effect the maintenance effort of the component based systems. They have made the assessment that (i) functionality of each component should be as concentrated as possible over a single aspect of the application domain, (ii) the training time offered by the component’s producer usually indicates the complexity of understanding it and if a component is difficult to understand, then it is also difficult to maintain; and (iii) a deep knowledge of the component is necessary for the organization before its adoption, therefore, a trial usage of components is advised before the final decision about their adoption. An automated decision support framework for software development and maintenance, present in , analyses different component assembly choices and assist the user in making cost effective decisions. It is builds a formal model of the software and uses automated planning techniques to produce alternatives choices to maintain the software under consideration while respecting the user’s effort and performance objectives. However, this framework is still in the prototype phase and full scale evaluation of the framework will lead to more accurate model [Ardimento2004].

Testability is “the degree to which a system or a component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met”. In the CBSD, the validation and verification phases play a
vital role. These component based systems are usually complex and testing the final product is time consuming and highly costly. Thus it is necessary to have testability analysis of the component based system, to indicate the difficulty level of component based system testing.

3.4 Conclusion

This part of study probes the literature survey of component based software in reference of quality assurance and assessment. It is accepted that this assessment can not be done only on the basis of qualities of component involved but on the overall qualities of the framework and integration process. The various aspects have been surveyed including formalism, cost estimation, measurement techniques, reliability, maintainability, testability. The findings of respective areas can be concluded in the following manners.

3.4.1 Formalism:

The formalism is a kind of an application for modeling and validation of component based software development activities as well as specification of non-functional properties. Different techniques have been developed for component based verification and reasoning, component based specification languages analysis of individual components and component composition etc. Formalism is also one of these techniques. It should be remembered that through formalism we don not get any information of system quality attributes. The recent researches also apply formalism to component trustworthiness, component certification, composition predictability, component configuration etc.
3.4.2 Measurement:

Measurement is concerned with the quantification of the quality aspects of component based software systems. In traditional quality measurement process, the focus was on identifying the key attributes of component based software systems quality and its difference from the traditional one.

Today, it is remarkable that we can determine the required quality with different requirement of different users. We are able to have useful quality data that make confident to the users for evaluation and assessment of quality attributes.

3.4.3 Cost Estimation:

Cost estimation is also an important factor that attracts the attention of software professional. Various research have been done for cost predication with reference to deal the requirement and complex maintenance processes.

3.4.4 Reliability:

The reliability of a component based systems depends not only the reliability estimation of individual components but also on their interconnection between the components of a system. To consider the failure of component interfaces is the main issue for the predication of reliability.

3.4.5 Security:

The evaluation of security properties in the early design phase is, compared to the other quality attributes, and there is still much work ahead to create encapsulated evaluation models for security and to quantitatively analyze security properties. One
major problem is that security deals with unknown threats. Another obvious problem that needs to be solved is the composability of security attributes.

3.4.6 Maintainability:

The activities of component replacement to troubleshooting and configuration management are associated with component based maintainability. We have the techniques for maintainability measurement based on re-configuring and re-integrating situations.

3.4.6 Testability:

Testability of a component based system is defined in terms of controllability, traceability and configuration management. Controllability and observability have been applied to measure the testability of individual component. The dependant relationship between program inputs and outputs has also been analyzed through this. The recent approaches have been extended to analyze the testability of whole system.