REVIEW ON COMPONENT BASED SYSTEM
Manpreet Kaur1, Sunil Khullar2
1Research Scholar, 2Assistant Professor,
1,2Department of Computer Science & Engineering, Rayat Institute of Engineering & Information Technology,
Railmajra, SBS Nagar (Punjab), India
1manpreetsainimani@gmail.com, 2Sunilkhullar222@yahoo.co.in

ABSTRACT: CBSD Technique is based on the idea to develop software systems by selecting appropriate off-the-shelf components and then to join them with a well-defined software structure. Because the new software development paradigm is more different from the traditional approach, quality assurance (QA) for CBSD is a new topic in the software engineering community. We survey current component-based software technologies, describe their benefits and limitation, and discuss the features they adopt. We also address QA issues for component-based software. As a major contribution, we propose a Quality Assurance model for CBSD which covers component requirement analysis, component development, component certification, component customization, and system architecture design, integration, testing, and maintenance. We are implementing this by using MATLAB.

1. INTRODUCTION
1.1 Software Engineering:
A software engineer is a licensed professional engineer who is expert in the application of engineering discipline to the creation of software. A software engineer is often confused with developer, but the two are vastly different disciplines. While a programmer creates the program that make a program run, a software engineer develop the designs the programmer implements. By U.S. law no person may use the title "engineer" (of any type) unless the person holds a professional engineering license from a state licensing board and are in good standing. A software engineer is also held accountable to a specific code of ethics. Engineering to the development, design, maintenance, testing, and evaluation of the software and systems that make computers or anything containing software work. Software engineering is a systematic and disciplined approach to developing software. It applies both computer science and engineering principles and practices to the creation, operation, and maintenance of software systems. All the process which happen with the software.

Software engineers apply the principles of At the University of Waterloo, Software Engineering is an independent, interdisciplinary program supported by both the Faculty of Mathematics and Engineering. Graduates of this program will earn a Bachelor of Software Engineering (BSE) degree. Typical formal definitions of software engineering are:
• "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software [in whole process of developing]."
• "an engineering discipline that is concerned with all aspects of production of software" [2]
• "The establishment and use of sound engineering principles in order to financially obtain software that is reliable and works efficiently on real machines".
The term has been used less formally:
• As the informal contemporary term for the broad range of activities that were formerly called computer programming and systems analysis;[4]
• As the broad term for all aspects of the practice of computer programming, as opposed to the theory of computer programming, which is called computer science.
• As the term embodying the advocacy of a specific approach to computer programming, one that urges that it be treated as an engineering rather than an art, and advocates the codification of recommended practices.

1.2 Components Based Software Engineering:
Component-based software engineering (CBSE) is a branch of software engineering that emphasizes the separation of concerns in respect of the wide-ranging functionality available throughout a given software system. It is a reuse-based approach to defining, implementing and composing loosely coupled independent components into systems. This
practice aims to bring about an equally wide-ranging degree of benefits in both the short-term long-term for the software itself and for organizations that sponsor such software. Software engineering practitioners regard components as part of the starting platform for service-orientation. Components play this role, in service-oriented architectures (SOA), whereby a component is converted by the web service into a service and subsequently inherits further characteristics beyond that of an ordinary component.

1.3 Components Based Software Engineering Process

• When reusing components, it is essential to make tradeoffs between ideal requirements and the services actually provided by available ideal requirements and the services actually provided by available components.
• This involves: Developing outline requirements; Searching for components then modifying requirements according Search for components then modifying requirements according to available functionality.
• Searching again to find if there are better components that meet the revised requirements. CBSE is a reuse-based approach to defining and implementing loosely based approach to defining and implementing loosely coupled components into systems.
• A component is a software unit whose functionality and dependency.
• During the CBSE process, the processes of requirements engineering and system design are interleaved. A component is a software unit whose functionality and dependencies are completely defined by its interfaces.

• When choosing compositions, you have to consider required and system functional requirements and system evolution, functionality, non-functional requirements

2. RELATED WORK

Sang Hun Ohv et al [1] “Software Quality Manager: a knowledge-based management tool of software metrics” The Software Quality Manager (SQM) goes for the development of a product measurements assessment environment in which the entire estimation methodology is performed by means of a designing point of view. All the information, data and learning needed for designers amid the methodology, for example, assessment models and choice supporting learning is formally spoken to and figured out how to improve consistency and to decrease misconception. Simultaneous estimation and examination undertakings happening amid the methodology are facilitated in SQM, keeping in mind the end goal to make the results critical. In the wake of characterizing the product measurements quickly, we display destinations and the three stage structure of SQM in point of interest. SQM helps subjective/quantitative estimation procedures and is furnished with learning bases for programming designing and application areas. It likewise uses space information of uses as programming parts. At last, we introduce our continuous endeavors to make the strategy pragmatic, including development of two information bases, incorporation of assessment procedures, bound together alignment units for programming measurements, subjective conduct era systems et cetera. Li-Xin JIANG et al [2] “Research on Size Estimation Model for Software system Test based on testing steps and Its Application” In this paper, a function testing size estimation model based on testing steps is proposed. The model applies to black box testing, and can be used in dependent test as a third party; this model is conducted overall analysis on COCOMO model. Its basic steps include: design test cases, sum total test steps, define the parameters in the model based on the pattern of model, define size factor, and calculate test Size. The model is applied in some test projects. The applications indicate that this model can be used to estimate the size, and bring positive effect to software test. Zheng Yan et al [3] “Autonomic Trust Management for a Component-Based Software System" Trust plays an important role in software systems, especially component-based systems in which components or their environments vary. This paper introduces an autonomic trust management solution for a component-based software system. We propose an adaptive trust control model to specify, evaluate, establish, and ensure the trust relationships among system entities. This model concerns the quality attributes of the entity and a number of trust control modes supported by the system. In particular, its parameters can be adaptively adjusted based on runtime trust assessment in order to reflect real system context and situation. Based on this model, we further
develop a number of algorithms that can be adopted by a trust management framework for autonomic management of trust during component execution. We verify the algorithms’ feasibility through simulation and demonstrate the effectiveness and benefits of our solution. We also discuss the issues for successful deployment of our solution in a component software platform. Georgiana Macariu et al [4] “Enabling Parallelism and Resource Sharing in Multi-core Component-based Systems” Complex real-time embedded systems require guarantees regarding the assurance of their timing requirements. Such guarantees can be derived using advanced design and analysis methods. Many design solutions address the complexity of these systems using component-based techniques. In this paper we focus on resource sharing in component-based systems with several components executing on a multi-core processor. We consider that the tasks of each component can be scheduled on any core with the possibility of two tasks belonging to the same component executing in parallel. We propose the Parallel Hierarchical Resource Policy, a novel resource sharing policy for multi-core component-based systems. We also develop a detailed response-time based schedulability analysis for the individual components and for the composed system, assuming that intra and inter-component resource sharing takes place. Sajjad Mahmood et al [5] “A Degree Centrality-Based Approach to Prioritize Interactions of Component-Based Systems” Component Based System (CBS) development focuses on integrating software components, often developed by different parties, to build an application. Component integration plays a critical role in overall CBS development. As the number of interactions increases in a CBS, there is a need for a better management of the component integration process. In a large and complex CBS, a system integrator is interested in analyzing different interactions to identify which interactions are more important than others. In this paper, we present an interaction prioritization process which uses the concept of 'degree centrality' to analyze component interactions and subsequently prioritize the use case scenarios of the system based on the centrality of individual interactions. We also present an application of our technique to a hotel reservation system. Singh Sandhu, P.et al [6] “A Neuro-Fuzzy Based Software Reusability Evaluation System with Optimized Rule Selection” There are measurements for distinguishing the nature of reusable parts yet the capacity that makes utilization of these measurements to discover reusability of programming segments is still not clear. We discriminantly dissected the CK measurements, attempted to uproot the inconsistencies and conceived neuro-fluffy system that gets enter in manifestation of tuned WMC, DIT, NOC, CBO, LCOM estimations of a product segment and yield can be gotten as far as reusability. This paper additionally indicates how a little number of fluffy principles can be chosen for planning beginning fluffy principle base for neuro-fluffy frameworks. It comprises of two stages: era of hopeful administers by IDS choice tree calculation and tenet pruning by assessment of tenets with help of two principle assessment criteria. The created reusability assessment framework has delivered high exactness results. Thus, the created framework can be utilized for recognizable proof and extraction of OO based reusable segments from legacy frameworks and assessment of created or creating reusable parts.

3. APPROACHES USED COCOMO MODEL

The Constructive Cost Model (COCOMO) is an algorithmic software cost estimation model developed by Barry W. Boehm. The model uses a basic regression formula with parameters that are derived from historical project data and current as well as future project characteristics. First published in Boehm's 1981 book Software Engineering Economics as a model for estimating effort, cost, and schedule for software projects. It drew on a study of 63 projects at TRW Aerospace where Boehm was Director of Software Research and Technology. The study examined projects ranging in size from 2,000 to 100,000 lines of code, and programming languages ranging from assembly to PL/mc. These projects were based on the waterfall model of software development which was the prevalent software development process. References to this model typically call it COCOMO 81. In 1995 COCOMO II was developed and finally published in 2000 in the book Software Cost Estimation with COCOMO II. COCOMO II is the successor of COCOMO 81 and is better suited for estimating modern software development projects. It provides more support for modern software development processes and an updated project database. The need for the new model came as software development technology moved from mainframe and overnight batch processing to desktop development, code reusability, and the use of off-the-shelf software components. This article refers to COCOMO 81.

CBS (COMPONENT BASED SYSTEM)

CBS is a branch of software engineering that emphasizes the separation of concerns in respect of the wide-ranging functionality available throughout a given software system. It is a reuse-based approach to defining, implementing and composing loosely coupled independent components into systems. This practice aims to bring about an equally wide-ranging degree of benefits in both the short-term and the long-term for the software itself and for organizations that sponsor such software. Software engineering practitioners regard components as part of the starting platform for service-orientation. Components play this role, for example, in web services, and more recently, in service-oriented architectures (SOA), whereby a component is converted by the web service into a service and subsequently inherits further characteristics beyond that of an ordinary component.
4. CONCLUSION AND FUTURE SCOPE

Component-based software engineering (CBSE) is a branch of software engineering that emphasizes the separation of concerns in respect of the wide-ranging functionality available throughout a given system. In this paper we enhance the interaction prioritization technique to rank all use cases of a CBS. We also use the interaction prioritization concept to develop a CBS integration testing framework. After the implementation of these techniques anyone can conclude that our system gives us better results.

REFERENCES