International Journal of Business, Human and Social Sciences
ISSN: 2517-9411
Vol:9, No:2, 2015

Some Pertinent Issues and Considerations on CBSE

Anil Kumar Tripathi, Ratneshwer

***Abstract*—**All the software engineering researches and best industry practices aim at providing software products with high degree of quality and functionality at low cost and less time. These requirements are addressed by the Component Based Software Engineering (CBSE) as well. CBSE, which deals with the software construction by components' assembly, is a revolutionary extension of Software Engineering. CBSE must define and describe processes to assure timely completion of high quality software systems that are composed of a variety of pre built software components. Though these features provide distinct and visible benefits in software design and programming, they also raise some challenging problems. The aim of this work is to summarize the pertinent issues and considerations in CBSE to make an understanding in forms of concepts and observations that may lead to development of newer ways of dealing with the problems and challenges in CBSE.

***Keywords*—**Software Component, Component Based Software Engineering, Software Process, Testing, Maintenance.

I. Introduction

THE basic idea of CBSE lies in processes, methods, frameworks and tools that support the way of software development by using components. A conventional Software Engineering process deals with 'Creation of a new system' where as a CBSE process deals with 'Composition from existing components'. This basic change in nature of development indicates that the conventional software processes, architectures, frameworks etc need to be redefined in order to make them applicable to CBSE. The goals of CBSE are: to provide an environment for development of software systems by assembly of components, and to provide an environment for development of reusable components and facilitating the maintenance, management and up-gradation of systems by customizing and replacing the components. The benefits of CBSE include reduced time-to-market and cost, better quality, better complexity management and easy maintenance.

Though these features provide distinct and visible benefits in software design and programming, they also raise some challenging problems. The challenges are: lack of precise component specification, component models, Component oriented software life cycle, composition predictability, dependency analysis, CBSE specific metrics, certification and tool support etc. It is required that CBSE should keep striving for providing ways to improve quality and productivity by making use of experiences that the industry undergoes from time to time. The role of an academic research is that of analysis for the purpose of theorization of the understandings in forms of concepts and observations that may lead to development of newer ways of dealing with the problems and challenges in CBSE.

Researchers and practitioners have been considering various aspects of CBSE and related issues. It is required to consider these efforts for the purpose of identification of issues, challenges and problems in the field so as to be able to ascertain some important considerations that need urgent, and possibly immediate attention. This paper attempts to present, in a concise manner, the research efforts related to the topic of discussion. A careful reading of literature helps in identifying research efforts that have gone into classification of concepts and understandings regarding CBSE.

The rest of the paper is organized as follows. In Section II, component based software engineering and difference between objects and components are briefly mentioned. In Section III, some specific issues of CBSE are summarized. In Section IV, some current challenges of CBSE are mentioned. In Section V, a comparison of CBSE and Service oriented system is given. In Section VI, limitations of CBSE as perceived by practitioner and in newer context are given. Finally we conclude the paper in Section VII.

II. Component Based Software

In a general term, a component is considered as a part of a system that performs certain functionality for that system. In the similar analogy, a software component can be defined as an independent executable unit that performs certain functionality when get plugged into an application software system.

One of the earliest definitions of software component is given by [1]: "A reusable software component is a logically cohesive, loosely coupled module that denotes a single abstraction". Later, Szyperski presented his well-known definition of a software component at the 1996 European Conference on Object Oriented Programming [2]: "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party." This definition is well accepted in the CBSE community because it highlights the major properties of software components that are not addressed in traditional software modules, such as context independence, composition, deployment and contracted interfaces [3]. In 2000, a broader, more general notion of software components was given by [4]: 'An independently deliverable piece of functionality providing access to its services through interfaces'. Another definition of a software

Anil Kumar Tripathi is serving as professor in Department of Computer Science and Engineering at Indian Institute of Technology (BHU)-Varanasi, India (e-mail: aktripathi.cse@iitbhu.ac.in).

Ratneshwer is serving as an Assistant Professor in Department of Computer Science (MMV) at Banaras Hindu University, India (e-mail: ratnesh@bhu.ac.in).

component is given in UML [5]: "A component represents a modular, deployable and replaceable part of a system that encapsulates implementation and exposes a set of interfaces". Recently, Councill and Heineman gave a definition to emphasize the importance of a consistent component model and its composition standard in building components and CBS [6]: "A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a component standard."

A. Difference between Components and Objects [7]-[9]

A common view is that a component is closely related to an object and that Component Based Development is therefore an extension of Object-Oriented Development. However many factors, such as granularity, concepts of composition and deployment, and even the development processes, clearly distinguish components from objects.

Object Oriented Programming focuses on the relationships between classes that are combined into one large binary executable, while Component Oriented Programming focuses on interchangeable code modules that work independently and do not require being familiar with their inner workings to use them. The fundamental difference between the two methodologies is the way in which they view the final application. In the traditional object oriented world, even though one may factor the business logic into many fine-grained classes, once those classes are compiled, the result is monolithic binary code. All the classes share the same physical deployment unit (typically an EXE), process, address space, security privileges, and so on. If multiple developers work on the same code base, they have to share source files. In such an application, a change made to one class can trigger a massive re-linking of the entire application and necessitate retesting and redeployment of all the other classes. On the other hand, a component –oriented application comprises a collection of interacting binary application modules- that is, its components and the calls that bind them. It you need to modify a component implementation, changes are contained in that component only. Components can even be updated while a client application is running, as long as the components are not currently being used. Improvements, enhancements and fixes made to a component may immediately be available to all applications that use that component, whether on the same machine or across a network. Unlike classes, the implementation of a component is generally completely hidden and sometimes only available in binary form. Internally, a component may be implemented by a single class, by multiple classes, or even by traditional procedures in a non-object oriented programming languages. Unlike classes, component names may not be used as type names. Instead, the concept of type (interface) and the concept of implementation are completely separated. Finally, the most important distinction is that software components conform to the standards defined by a component model.

III. Specific Issues in CBSE

Software engineering, over the last few decades, has been promoting the development of software systems with reusable software pieces. Teaching the principle of *separation of concerns*, [10] has shown that pieces of a program could be developed independently. In 1968, [11] proposed that components could be largely applied to different machines & different users and should be available in families arranged according to precision, robustness, generality and performance. According to him, components could be used to maintain the software industry mass production. Parnas has introduced the concept of information hiding [12] for decomposing a system into parts that hide implementation details behind interfaces. In such a system, any module can be replaced by a different one satisfying the same interface. This paradigm has now been anointed with the name Component Based Software Development (CBSD). CBSD is changing the way large software systems are developed. CBSD embodies the *buy; do not build* philosophy espoused by [13].

Considerable efforts have been spent, both by academia and industry, to advance the state-of-the-art of component technology. Some of these efforts are summarized below.

A. Software Processes for CBSE

A CBS system is different from a conventional software system in multiple ways. The traditional discipline of Software Engineering needs newer methodologies to support Component Based Development. Pree [14] has suggested a way to manage complexity and rapid change adaption through reuse of already developed components. This implies that the development process must change focus – from programming-intensive activities to reuse, integration, standards, management of complex and flexible structures, finding proper solutions, tradeoff analysis and marketing survey. These require changes in development procedures, tools, but also developers' skills and organizational changes. Brown & Wallnau [15] have presented a reference model for the assembly of component-based systems that can be used as the basis for defining Q systematic approach to the development of such systems. Morisio, Seaman, Parra and Basili [16] have summarized the results of a study on fifteen projects that used a COTS-based approach. The process they followed is evaluated to identify essential differences in comparison to traditional software development. The main differences, and the activities for which projects require more guidance, are requirements definition and COTS selection, high level design, integration and testing. In [17], a comparison of CBSE processes and Conventional software processes has been demonstrated.

B. Development of Components from Legacy Systems

Many legacy systems have suffered from lack of standardization and openness, difficulty of change, and absence of distributed architecture. Especially, according as legacy system has been deteriorating from an architectural point of view over the years, we must continually maintain these legacy systems at high cost for applying new

technologies and extending their business requirements. For the purposes of transforming a legacy system into component system, we need systematic methodologies and concrete guidelines [18]. Some significant works have been observed related to component oriented re-engineering that include extraction of software components from a legacy software system, component based forward engineering etc. Keller and others [19] have given an overview of an environment for pattern based reverse engineering of design components. Skarmstad and others [20] have formulated a framework within which extracted computing units could be gradually migrated as independent COTS. Favre and others [21] have pointed out some issues in reengineering the architecture of evolving CBS. They have shown the use of a Meta model in understanding and reasoning about components, and how this Meta model constitutes a good basis for building a reverse engineering environment. Lundberg, Jonas & Lowe [22] have given a presentation of dominance analysis, and how it can be used to identify software components in object oriented legacy systems. Kontogiannis and Zou [23] have proposed a framework that allows for the identification of reusable business logic entities in large legacy systems in the form of major legacy components, the migration of these procedural components to an object oriented design, the specification of interfaces of these identified components, the automatic generation of CORBA wrappers to enable remote access, and finally, the seamless interoperation with Web services via HTTP based on the SOAP messaging mechanism. In [24] an approach has been described for identifying reusable components from legacy systems.

C. Representation of Component Based Software

We have a general idea of what a component is, but because, in the software context, what we know as components have so many varied forms, functions and characteristics, (as source code modules, parts of an architecture, parts of a design, binary deployable executables, etc.), there is a correspondingly large number of definitions of a component [25]. Ning [26] has proposed a process model and supporting technologies to describe explicit representation of software components and component based architectures. Collins-Cope and Matthews [27] have proposed a reference architecture for component based systems consisting of five layers. The purpose is to show how this model helps us to understand the overall structure of a system, how layering helps to clarify our thoughts, and how it encourages the separation of concerns such as the technical vs. the problem domain, policy vs. mechanism, and the buy-or-build decision. Bosch and Stafford [28] have introduced the notion of software architecture as a key success factor for component based software development. Some architectural styles found in the CBSE literature are: pipe and filter architecture, blackboard architecture, object-oriented architecture, shared repository, layered abstract machine, domain specific architecture etc. Conradi and others [29] have modeled the COTS-based software process using a software process modeling language (E3) and discussed some of their limitations, and given suggestions for improvement of these. Smeda and others [30] have presented a short summery of an approach to model a component-based system, in which connectors are defined explicitly and raised to the level of components. Hatebur and others [31] have proposed a method for Component-based software and system development, where the interoperability between the different components is given special consideration.

D. Composability in Component Based Software

While Composability is a much desired quality for software artifacts, there is no consensus whatsoever on what Composability really is, or how it can be achieved [32]. Seco [33] has discussed the design and implementation of a composition language for the .NET platform. The language is based on a simple core language that includes specific abstractions for composition. Sitaraman [34] has discussed a variety of issues in compositional performance specification and reasoning, including the impact of abstraction, precision, and parameterization. Wuyts and Ducasse [35] have claimed that one of the important problems that should be addressed by component languages is the composition of components. CBSE is also important for another basic reason. It enables compositional or modular reasoning, and therefore, it facilitates production of high quality systems. The defining characteristic of composability is the ability to combine and recombine components. There are both syntactic and semantic forms of composability; they deal respectively with technical aspects of enabling components to work together and with whether their combined computation is meaningful [36]. A need arises for specialized environments that explicitly support the composition of an application out of deployable components. Such composition environments are starting to emerge, but an overall understanding of their nature and functionality is currently lacking [37]. Reussner and others [38] have demonstrated parametric performance contracts and their applications and also show that these contracts are compositional. One needs a better understanding of the requirements involved in successful composition, and in addition defines the situations where composition fails. With this aim, [39] have (a) introduced a general model of composing systems from multiple concerns, (b) introduced a number of requirements for design-level Composability and (c) defined a category of Composability problems that are inherent for given composition models, which they term as composition anomalies. To integrate the previously existing components into a new system, a lot of problems may occur. These problems frequently occur when composing two or more components in a CBSD process.

E. Reusability of Software Components

Designing with reuse of existing components has many advantages. The software development time can be reduced and reliability of the product can be enhanced. Reuse of software component is justifiable if the cost of reuse is less than the cost of developing new components. Hence, one empirical attribute for reusability is the effect or cost required

to reuse a certain software component [40]. Caldiera and Basili [41] propose the three reusability factor: cost of reuse, usefulness of reusable components and quality of reusable components. Broadly, there are two types of component based reuse: with no change to an existing component, and with change. Kim, in his paper, mentioned the following factors that make the reusability complex: functional differences, programming language differences, operating environment differences, industry standard differences, data format differences and data structure differences.

IV. Challenges in Component Based Software

Software industry is now moving away from giant, monolithic and difficult-to -maintain code based software development. Component Based Development has established itself as the overriding software development methodology. Some of the research challenges are summarized below.

A. Testing of Component Based Software

CBS systems raise new problems for the testing community. Although the technology for constructing component-based software is relatively advanced, there is a lack of sufficient theoretical basis for testing component-based software. Rosenblum [42] has initiated the development of such a theory. The main result is a formal definition of the concept 'C-adequate-for-M' for adequate unit testing of a component and the concept 'C-adequate on-M' for adequate integration testing of a component-based system. Martins and others [43] have given an approach to improve component testability by integrating testing resources into it, and hence obtaining a self-testable component. Weide [44] has given an approach of Modular Regression Testing. Weyuker [45] has outlined some of the potential problems to be expected when a project team decides to use a software component originally written for a different component infrastructure with different usages patterns. Mariani and Pezze [46] have described a verification technique to check the completeness and compatibility of the services provided by the components to reveal possible conflicts. They have proposed a technique to automatically identify behavioral differences between different versions of the system, to deduce possible problems from inconsistent behaviors. Denaro and others [47] have proposed to derive application-specific test cases from architecture designs so that the performance of a distributed application can be tested based on the middleware software at early stages of a development process. Behavioral differences among components may cause subtle failures difficult to reveal and remove.

B. Maintenance of Component Based Software

Building a commercial system from software components changes neither the importance of, nor the expense associated with, maintenance, evolution and management. That we are using a system built from components rather than source code does change the nature of the maintenance activities [48]. Clapp and Taub [49] have considered the issues and risks in using COTS software over the life cycle and how to control them. They described changes in the software maintenance process that are needed to manage a COTS-based system. Voas [50] has given an overview of the maintenance challenges raised by component based development. He has presented a consumer-oriented methodology for predicting what impact on system quality a particular Commercial-Off-The-Shelf (COTS) software component will have. Vigder and Dean [51] have described how organizations can use software architecture, software instrumentation, and component-based configuration management to support the ongoing system management activities. Dig & Johnson [52] have explored what software maintenance technologies will be needed in order to successfully maintain component based systems. Vigder and Dean [51] have identified the major activities of a system maintainer, described the properties that can be designed into a system to facilitate these activities, and outlined a checklist of items that can be verified during a design or code review, or during the evaluation of a COTS component in order to guarantee these properties are built into the system.

C. Metrics for Component Based Software

Creating quality products requires insight, control and management throughout the CBSE life cycle. Metrics provide that data and, when properly used, greatly enhance the control over the component development process and quality of CBS. Verner and Tate proposed a method for estimating early in the software life-cycle the LOC of a system [53]. Relying on COTS components increases the systems vulnerability to risks arising from third-party development, such as vendor longevity and intellectual-property procurement. One way of alleviating such concerns is by using software metrics to guide quality and risk management in a Component-Based System (CBS), accurately quantifying various factors contributing to the overall quality, and identifying and eliminating sources of risk [54]. The limitations of traditional models demonstrate the need for a new approach to effort estimation in CBSD [55]. This method, described as a component-based method (CBM), sizes individual components or modules first and then adds the component sizes to obtain an estimated system size. The partition in components depends on the environment (type of software) and, therefore, is not fixed. Poulin [56] has outlined some of the basic metrics for CBSE projects. He has grouped them into areas to address issues in project management, quality, reuse and technology.

D. Component Based Software for Real Time Systems

Villela and others [57] have described a framework for distributed real-time embedded systems, which has been developed to help developers in task of modeling and implementation of embedded real-time applications. Although attractive, CBD has not been widely adopted in domains of embedded systems. The main reason is inability of these technologies to cope with the important concerns of embedded systems, such as resource constraints, real-time or dependability requirements [58]. Existing component models provide no support for real-time services and some real-time

extensions of component models lack consideration for reusability of components in providing real-time services. Wang and others [59] have developed a real-time component-based system that maintains the reusability of components. Key challenges in distributed real-time embedded (DRE) system developments include safe composition of system components and mapping the functional specifications onto the target platform. Model-based verification techniques provide a way for the design-time analysis of DRE systems enabling rapid evaluation of design alternatives with respect to given performance measures before committing to a specific platform [60]. Sinha and Hanumantharyab [61] have illustrated how the concepts of category theory can be utilized to develop component-based fault-tolerant software systems that encompass software components capable of tolerating particular types of faults. Ducasse and others [62] have described a data-centric component model for embedded devices that (i) minimizes the number of concurrent tasks needed to implement the system, (ii) allows one to verify whether components meet their deadlines by applying rate monotonic analysis, and (iii) can generate and verify schedules using constraint logic programming.

E. Dependency Analysis of Component Based Software

Component-based development has become an important area in the software engineering field. In [63], a definition of inter-procedural dependence analysis and its implementation in a prototype tool that supports software maintenance have been presented. In spite of this, there has been little effort to understand and to manage the different forms of dependencies that can occur in systems built from components [64]. Managing dependencies among components is one of the most crucial problems one has to solve before a system can be dynamically reconfigured at runtime [65]. Chen has described a dependence management for dynamic reconfiguration of distributed systems. The dependence management analyzes not only the static dependencies among components, but also the dynamic dependencies that take place at runtime, in order to support an efficient consistent reconfiguration of distributed systems [66]. As the size and complexity of software systems growing, the identification and proper management of interconnection dependencies among various pieces of a system have become responsible for an increasingly important part of the development effort. Guo made an attempt to address this problem by using category theory and given a framework of the dependencies modeling [67]. In using components, the most difficult issues are ensuring that hidden dependencies won't cause failures and that non-functional properties (such as real-time performance) are being met.

V. Component Based Development vs. Service Based Development

Although components and services have common concepts of reusability in their origin but they have differences at their architectural, internal description and abstraction levels. Components are known by their classes and interfaces whereas services are known by their service contracts. Objects have tight coupling but components and services have loose coupling. The major differences are in their connections and the way they provide services to third party. Service oriented computing provides a way to create a new architecture that reflects components trend towards autonomy and heterogeneity [68]. Software components support black box and white box encapsulation both but software services support only black box encapsulation. In CBSE, there is a limited composition support for components of different models but software services are implementing in diverse technologies, on different platforms.

Both, components and services are self content entities having some specific functionality. Component is meant for particular language but services facilitate components to go across language boundary. In CBSE, components are the building blocks that can be deployed independently and are subject to composition by third party. In service based development, software services are building blocks that can be reused and offer particular functionality. They are generally implemented as coarse-grained discoverable software entities [69]. Although both CBSE and service oriented system engineering are interface based, the separation between service description and service implementation is more explicit than the separation between component specification and component specification and implementation [70]. Services operate in distributed environment and focus on document centric communication. In contrast, component based development does not take that much stand on how the components interact with one-another, this depends on the technology that the components are based on. For integrating existing components, SOA has a concept of service composition which is facilitated by service bus. CBD uses function calls and application framework such as object request brokers, message passing middleware and application servers [71].

VI. Limitations on CBSE

In this section, we have mentioned some limitations of the CBSE observed from literature.

A. As Perceived by Practitioners

In large companies the development can be geographically distributed throughout many different offices. The larger distance between the offices, the harder it is to keep closer contact with other developers. This makes it harder to coordinate the development. It is especially hard to coordinate changes in a component's interface [72]. The one problem is lack of belief. A software developer uses components that she has not developed. So, developers sometimes are not feeling confident during development. This problem becomes more significant due to lack of proper documentation. The components are characterizing by their document. If exceptions, assumptions associated with the components are not properly documented, then developer could misunderstood about the uses of components. Different software developers are different views about documentation. Some of them require the complete description of interfaces; some of them

want to know about exceptions etc. So, proper documentation is required. Components are generally known as reusable entities as 'developed once and reuse many times'. In order to make a component reusable, one has to incorporate all general requirements require for a specific application. The component has to be designed so that it can be used in different application which causes additional cost associated with the component. During maintenance of system, replacing component with their newer version may cause some functional/behavioral mismatches. One limitation is to select right components for specific application. If a software company is having internal component repository (component developed within the company) then searching is easy. But if one has search component from some external repository then searching process will be difficult because we have to use their searching criteria. It may be difficult to measure how well a component meets a certain criteria and lead to difficulty in selecting a component. This is especially true if there is no prior knowledge about the components or prior extensive experience of creating measurable test cases [73]. Evaluating non-functional qualities of a component is hard because of incomplete understanding of them and inadequate tools and techniques for determining if they meet the expectations.

B. As Perceived in Context of Multi-core Computing

During the last few years there have been a number of emerging trends in CBSE. One important issue is multicore computing in CBSE environment. The number of cores on a device is still fairly modest, and individual software components are developed for a single computational cluster by component developer and then assembled into a multicore system. Development tools for this methodology improve steadily as virtualization of hardware through middleware is derived by efforts such as SCA (Software Communication Architecture). Auto generation of glue code between components is the norm [74]. The issues of multicore computing includes interrupt handling, writing code for especially for multi core processors, time management etc. The one approach may be to partition the systems into components and assign components to certain cores to realize the effect of multicore computing in CBSE environment.

Component-based programs, as a kind of user level programs, make full use of the multi-core architecture with user level threading model. OS and compiler make their changes to support component-based as well as multi-threaded. This kind of changes may be partial according to different model and little changes should be made [75]. The basic multi-core programming assumes each component has a dedicated scheduler for its tasks and further, a system-level scheduler is responsible for scheduling the components on the processor. The available processor capacity is divided between components using bandwidth server techniques. A separate server is allocated to each component and each server is seen by the system-level scheduler as a task with a unique priority. At the next level the tasks of the component are scheduled by the dedicated component scheduler according to a component specific algorithm [76].

The issues of multi-core computing in CBSE are as follows:

- Multi-core specific development debugging tools
- Multi-core component standards
- Process models for multi-core development
- Programming models for multi-core development.

VII. Conclusion

CBSE has always been considered as a useful and promising technology. The work attempted to consider the methodologies and concepts of conventional software engineering so as to be able to bring out pertinent observations regarding these methodologies and concepts in context of CBSE. Various researches and key practices regarding CBSE have inferred that the conventional development process must change focus and have demonstrated that CBSE requires established methodologies and tool support covering the entire component and system lifecycle including technological, organizational, marketing, legal, and other aspects. CBS systems raise new problems for the testing and maintenance community. Although the technology for constructing a CBS is relatively advanced, there is a lack of sufficient theoretical basis for testing and maintenance of a CBS. CBSE specific metrics are required to enhance the control over the component development and management process and quality improvement.

References

[1] G. O. Young, "Synthetic structure of industrial plastics (Book style with paper title and editor)," in *Plastics*, 2nd ed. vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 15–64.
[2] Booch, G. (1993). Software Components with Ada: Structures, Tools, and Subsystems, 3rd Edition. Reading, MA: Addition-Wesley.
[3] Szyperski, C. (1999). Component Software- Beyond Object Oriented Programming, Reading, MA: Addition-Weslay.
[4] Gao, J., Z., Jacob Tsav, H., S., & Wu, Y. (2003). Testing and Quality Assurance for Component Based Software, MA: Artech House, INC.
[5] Brown, A. J. (2000). Large-scale Component Based Development. Englewood Cliffs, NJ: Prientice Hall.
[6] OMG. (2000). OMG Unified Modeling Language Specification, version 1.4, Object Management Group, 2000.
[7] Councill, W. T. &Heineman, G. T. (Eds.). (2001). Component Based Software Engineering: Putting the Pieces Togather, Reading, MA: Addison Wesley, 2001.
[8] Lowy, J.(2005). Programming .Net Components. O'Reilly Media, Inc. Sebastopol,CA.
[9] Crnkovic, I. & Larsson, M. (2002). Building Reliable Component Based Systems. MA: Artech House Publishers.
[10] Weinreich, R. &Sametinger, J. (2001). Component models and component services: concepts and principles. In G. T. Heineman and W. T. Councill, (Eds.), Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley Longman Publishing Co., Boston, MA, 33-48.
[11] Dijkstra, E. (1968),The Structure of the 'T.H.E.' Multiprogramming System. (Electronic Version).Communications of the Association of Computing Machinery, Vol. 11, No. 5, pp. 453-457.
[12] McIlroy, M. D. (1968). Mass Produced Software Components. In: NATO Software Engineering Conference Report, Garmisch, Germany, October, pp. 79-85.
[13] Parnas, D. (1972). On the Criteria for Decomposing Systems into Modules. (Electronic Version).Communications of the Association of the Computing Machinery, Vol. 15, No. 12, pp. 1053-1058.
[14] Brooks, F. (1987). No Silver Bullet: Essence and Accidents of Software Engineering. IEEE Computer, Vol. 20, No. 4, 1987, pp. 10-19.
[15] Pree, W. (1997). Component-Based Software Development- A New Paradigm in Software Engineering?In Proceedings of the Fourth Asia-

Pacific Software Engineering and International Computer Science Conference, APSEC.IEEE Computer Society, Washington, DC.
[16] Brown, A. W. and Wallnan, K. C. (1996). Engineering of Component-based Systems.In Proceedings of the 2nd IEEE international Conference on Engineering of Complex Computer Systems (ICECCS '96) (October 21 – 25).ICECCS.IEEE Computer Society, Washington, DC, 414.
[17] Morisio, M., Seaman, C. B., Parra, A. T. &Basili, V. R. (2000). Issues in COTS-Based Software Development. Presented at COTS Workshop - Continuing Collaborations for Successful COTS Development, Limerick, Ireland.
[18] Tripathi, A. K., Ratneshwer& Gupta, M. (2007). Some Observation on software processes for CBSE. Software Process: Improvement and Practice. Volume 13, Issue 5, September 2008, pages 411-419.
[19] Kim,H. K., Chung,Y. K. (2006), "Transforming a Legacy System into Components", Springer-Verlag Berlin Heidelberg, 2006.
[20] Keller, R. K., Schauer, R., Robitaille, S., &Pagé, P. (1999). Pattern-based Reverse-Engineering of design components.In Proceedings of the 21st international Conference on Software Engineering (Los Angeles, California, United States, May 16 – 22).International Conference on Software Engineering. IEEE Computer Society Press, Los Alamitos, CA, 226-235.
[21] Skramstad, T., Khan, K., Rashid, M., (1999). Constructing Commercial Off-the- Shelf from Legacy system: A Conceptual Framework, Australasia Conference on Information Systems (ACIS), Wellington, 1-3 December 1999, pp. 798-805.
[22] Favre, J. M., Cervantes, H., Sanlaville, R, Duclos, F. &Estublier, J. (2001). Issues in Reengineering the Architecture of Evolving Component-Based Software. SWARM forum (Software Architecture Recovery and Modeling) at the Working Conference on Reverse Engineering (WCRE'2001) Stuttgart, Germany, October 2001.
[23] Lundberg, Jonas &Löwe, W. (2003). Architecture Recovery By Semi-Automatic Component Identifcation. In: Software Composition (SC'03) - a workshop afiliated with the Europian Joint Conference on Theory and Practice of Software (ETAPS'03), Poland, April 2003.
[24] Kontogiannis, K. &Zou, Y. (2004). Reengineering Legacy Systems Towards Web Environments. In: K. M. Khan and Y. Zheng (eds.): Managing Corporate Information Systems Evolution and Maintenance, Idea Group Publishing, Hershey. pp. 138–146.
[25] CGI. (2004). White Paper on "Component Mining: An approach for Identifying Reusable Components from Legacy Systems", By CGI Groups. http://whitepapers.techrepublic.com.com/abstract.aspx?docid=122639. Retrieved August 2007.
[26] Lüders, F., Lau, K. K. and Ho, S. M. (2000), "Specification of Software Components", In IvicaCrnkovic and Magnus Larsson (Editors),Building Reliable Component-Based Software Systems, ISBN 1-58053-327-2, Artech House Books, 2000.
[27] Ning, J. Q. (1996). A Component-Based Software Development Model. In Proceedings of the 20th Conference on Computer Software and Applications (August 19 – 23). COMPSAC.IEEE Computer Society, Washington, DC, 389.
[28] Collins-Cope, M. & Matthews, H. (2000). A Reference Architecture for Component Based Development. in Proceedings of the 6th international Conference on Object Oriented information Systems (December 18 - 20, 2000). Springer-Verlag, London, 225-237.
[29] Bosch, J. & Stafford, J. A. (2002). Architecting Component Based Systems. In I. Crnkovic.and M. Larsson (Eds.), Building Reliable Component Based Systems (p. 41-54). MA: Artech House, publishers.
[30] Conradi, R., Jaccheri, L. &Torchiano, M. (2003). Using software process modeling to analyze the COTS based development process. In proceedings of Prosim'03 Workshop, May 2-3, Portland State University, Maryland.
[31] Smeda, A., Oussalah, M. &Khammaci, T. (2004). Improving Component-Based Software Architecture by Separating Computations from Interactions.First International Workshop on Coordination and Adaptation Techniques for Software Entities, WCAT'04, Oslo, Norway.
[32] Hatebur, D., Heisel, M. &Souquieres, J. (2006). A Method for Component-Based Software and System Development. In Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (August 29 - September 01, 2006). EUROMICRO. IEEE Computer Society, Washington, DC, pp. 72-80.
[33] Lucas, C &Steyaert, P. &Mens, K. (1996). Research Topics in Composability.In Proceedings of the CIOO Workshop at ECOOP, Linz, Austria, July 8-12, 1996.
[34] Seco, J. C., (2000). Type Safe Composition in .NET.In the proceedings of The First Microsoft Research – Summer Research Workshop, Cambridge, England.
[35] Sitaraman, M. (2001). Compositional Performance Reasoning. In proceedings of 4th ICSE Workshop on Component Based Software Engineering Component certification and system prediction, may 14-15, Toronto, Canada.
[36] Wuyts, R. &Ducasse, S. (2002). Composition languages for black-box components. http://www.iam.unibe.ch/scg/Archive/Papers/Wuyt01c.pdf, February 2002.
[37] Petty, M. D. (2002). Semantic Composability and XMSF", XMSF Technical Challenges Workshop 2002, Monterey CA.
[38] Lüer, C. &Hoek, A. V. (2002). Composition Environments for Deployable Software Components.Technical Report.UCI-ICS-02-18. Department of Information and Computer Science, University of California, Irvine, August 2002.
[39] Reussner, R. H., Firus, V. & Becker, S. (2004). Parametric Performance Contracts for Software Components and their Compositionality.In Proceedings of the 9th International Workshop on Component-Oriented Programming (WCOP 04), Oslo, Norway, June 2004.
[40] Bergmans, L., Tekinerdogan, B., Glandrup, M. &Aksit, M. (2000). On Composing Separated Concerns- Composability and Composition Anomolies. ACM OOPSLA Workshop on Advanced Separation of Concerns, Minneapolis, Minnesota, USA, 15-19 Oct, 2000.
[41] Gill N.S. (2003). "Reusability Issues in Component based Development", ACM SIGSOFT SEN, Volume 28, Number 4, 2003, ACM Press, New York, NY, USA, Pages 4-7.
[42] Caldiera,G. and Basili, V. R. (1991). "Identifying and qualifying reusable software components," Computer, vol. 24, no. 2, pp. 61-70, Feb. 1991.
[43] Rosenblum, D. S. (1997). Adequate testing of component-based software. Technical Report. Department of Computer Science.University of California, Irvine. Technical Report UCI-ICS97 -34.
[44] Martins, E., Toyota, C. M. &Yanagawa, R. L. (2001). Constructing Self-Testable Software Components. In Proceedings of the 2001 international Conference on Dependable Systems and Networks (Formerly: Ftcs) (July 01 - 04, 2001). DSN. IEEE Computer Society, Washington, DC, 151-160.
[45] Weide, B. (2001). Modular regression testing": Connections to component-based software. In proceedings of 4th ICSE Workshop on Component-based Software Engineering: Component Certification and System Prediction. May 14-15, Toranto, Canada, p. 47-51.
[46] Weyuker, E. J. (2001). The Trouble with Testing Components.In G. T. Heineinnan and W. T. Councill. Component Based Software Engineering- Putting the Pieces Together (pp: 519-522). MA:Addison Wesley Professional.
[47] Mariani, L. &Pezze, M. (2003). Behavior Capture and Test for Controlling the Quality of Component-Based Integrated systems. In: Proceedings of the ESEC/FSE Workshop on Tool integration in system development, Helsinki.
[48] Denaro, G., Polini, A. &Emmerich, W. (2005). Performance Testing of Distributed Component Architectures. In: Building Quality into COTS Components - Testing and Debugging. Springer, pp. 294-314.
[49] Vigder, M. (2001). The Evolution, Maintenance and Management of Component Based Systems. In G. T. Heineman& W. T.Councill (Eds.), Component Based Software Engineering: Putting the Pieces Togather (pp. xvii). Addison-Wesley Longman Publishing Co., Boston, MA.
[50] Clapp, J. A. &Taub, A. E. (1998). A Management Guide to Software Maintenance in COTS-Based Systems. MITRE Corporation. http://www.mitre.org/resources/centers/sepo/sustainment/manage_guide_cots_base.html.
[51] Voas, J. et al. (1996). Gluing Together Software Components: How Good Is Your Glue? Proc. Pacific Northwest Software Quality Conf., Pacific Northwest Software Quality Conf. Inc., Portland, Ore., 1996, pp. 338-349.
[52] Vigder, M. R. & Dean, J. C. (1998(a)). Managing long-lived COTS based systems. In Software Engineering Standards Workshop, Monterey, California.
[53] Dig, D. & Johnson, R. (2006). Automated Upgrading of Component-based Applications. In Companion To the 21st ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (Portland, Oregon, USA, October 22 - 26, 2006). OOPSLA '06. ACM, New York, NY, 675-676.
[54] Verner, J. & Tate, G. (1992). A Software Size Model.IEEE Transaction of Software Engineerung. 18, 4 (Apr. 1992), 265-278.

[55] Paul, R. A. (1995). Metrics-guided Reuse. In Proceedings of the Seventh International Conference on Tools with Artificial Intelligence, p.120, November 05-08.
[56] Smith, R., Parrish, A. & Hale, J. (1998). Cost Estimation for Component Based Software Development", Proceedings of the 36th Annual ACM Southeast Conference, April 1-3, Marietta, Georgia.
[57] Poulin, J. S. (2001). Measurement and metrics for software components. In G. T. Heineman and W. T. Councill (Eds.), Component-Based Software Engineering: Putting the Pieces Together (p. 432-452). Addison-Wesley Longman Publishing Co., Boston, MA.
[58] Villela, C., Becker, L. B., and Pereira, C. E. (2001). Framework for Component-Based Development of Distributed Real-Time Systems.In Proceedings of the Sixth international Workshop on Object-Oriented Real-Time Dependable Systems (Words'01) (January 08 - 10, 2001).WORDS.IEEE Computer Society, Washington, DC, 85.
[59] Crnkovic, I. 2005. Component-based Software Engineering for Embedded Systems.In Proceedings of the 27th international Conference on Software Engineering (St. Louis, MO, USA, May 15 - 21, 2005).ICSE '05. ACM, New York, NY, 712-713.
[60] Wang, Y., King, G. &Wickburg, H. (1999). A Method for Built-in Tests in Component-based Software Maintenance.In Proceedings of the Third European Conference on Software Maintenance and Reengineering (March 03 – 05).CSMR.IEEE Computer Society, Washington, DC, 186.
[61] Madl, G. &Abdelwahed, S. 2005. Model-based analysis of distributed real-time embedded system composition.In Proceedings of the 5th ACM international Conference on Embedded Software (Jersey City, NJ, USA, September 18 - 22, 2005).EMSOFT '05. ACM, New York, NY, 371-374.
[62] Sinhaa, P. &Hanumantharyab, A. (2005). A novel approach for component-based fault-tolerant software development. Information and Software Technology, Volume 47, Issue 6, 15 April 2005, Pages 365-382.
[63] Wuyts, R., Ducasse, S. &Nierstrasz, O. (2005). A data-centric approach to composing embedded, real-time software components. (Electronic Version). Journal of System and Software. 74, 1 (Jan. 2005), 25-34.
[64] Loyall, J. P. &Mathisen, S. A. (1993). Using Dependence Analysis to Support the Software Maintenance Process. In Proceedings of the Conference on Software Maintenance D. N. Card, Ed. IEEE Computer Society, Washington, DC, 282-291.
[65] Vieira, M. and Richardson, D. (2002). Analyzing Dependencies in Large Component-Based Systems.In Proceedings of the 17th IEEE international Conference on Automated Software Engineering (September 23 - 27, 2002).Automated Software Engineering.IEEE Computer Society, Washington, DC, 241.
[66] Chen, J. 2007. Component Oriented Design Style. In Proceedings of the 31st Annual international Computer Software and Applications Conference - Vol. 2- (COMPSAC 2007) - Volume 02 (July 24 – 27).COMPSAC. IEEE Computer Society, Washington, DC, 651-657.
[67] Guo, J. (2002(a)). Using Category Theory to Model Software Component Dependencies.In Proceedings of the 9th IEEE international Conference on Engineering of Computer-Based Systems (April 08 - 11, 2002).IEEE Computer Society, Washington, DC, 185.
[68] Stankovic, j. A., Zhu, R., Poornalingam, R., Lu, C., Yu, Z., Humphrey, M. & Ellis, B. (2003). VEST: An Aspect-Based Composition Tool for Real-Time Systems. In Proceedings of the The 9th IEEE Real-Time and Embedded Technology and Applications Symposium, p.58, May 27-30,Washington, DC.
[69] Yan S, Wang J, Liu C, Liu L (2008), An approach to discover dependencies between service operations. Journal of Software, 3(9):36-43.
[70] Brown, A., Johnston, S.and Kelly, K. (2002). “Using Service Oriented Architecture and Component-Based Development to Build Web Service Applications”, A Rational Software White Paper, 2002. Breivold, H. and Larsson, M. &ldquo,(2007) Component-Based and Service-Oriented Software Engineering: Key Concepts and Principles,&rdquo, Proc. EUROMICRO Conf. Software Eng. and Advanced Applications, pp. 13-20, Aug. 2007.
[71] Iribarne, L.: Web Components: A comparison between Web services and software components. Colombian Journal of Computation, 5(1), 47–66 (2004).’
[72] Bucanac, C. (1995). Problems With Component Based Software Development- A Comparison between Theory and Practice. Master Thesis, Department of Software Engineering and Computer Science, University of Karlskrona, Sweden.
[73] Maiden, N. A. & Ncube, C. Acquiring COTS Software Selection Requirements, IEEE Software, March/April 1998.
[74] Gatherer, A., From 2008 to 2020:A history of developments in programmability, Article on Web. Access at: http://www.ti.com/lit/wp/spry118/spry118.pdf.
[75] Dai, H., Chen, T., Jia, Z. (2009), Component-Based Multi-Threading Support of the Multi-Core Mobile Software, WRI International Conference on Communications and Mobile Computing, 2009. CMC '09, Yunnan, 6-8 Jan. 2009, pp. 17-20.
[76] Macariu, G., Cretu, V. (2011),Enabling Parallelism and Resource Sharing in Multi-core Component-based Systems, 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 28-31 March 2011, Newport Beach, CA, pp. 269-277.