Software Engineering

Component Based Software Engineering

D. Sarvani
Department of Information Technology
Saradha Gangadharan College
Velrampet, Pondicherry

R. Saradha
Department of Information Technology
Saradha Gangadharan College
Velrampet, Pondicherry

***Abstract*— Component-based software engineering (CBSE) (also known as component-based development (CBD)) 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. This approach promises to alleviate the software crisis at great extents. The objective of this paper is to gain attention towards this new component based software development paradigm and to highlight the benefits of the approach for making it a successful software development approach to the concerned community and industry**

I. INTRODUCTION

The idea that software should be componentized - built from prefabricated *components* - first became prominent with Douglas McIlroy's address at the NATO conference on software engineering in Garmisch, Germany, 1968, titled *Mass Produced Software Components*. The conference set out to counter the so-called software crisis. McIlroy's subsequent inclusion of pipes and filters into the UNIX operating system was the first implementation of an infrastructure for this idea.

The software components are used in two different contexts and two kinds: (i) using components as parts to build a single executable, or (ii) each executable is treated as a component in a distributed environment, where components collaborate with each other using internet or intranet communication protocols for IPC (Inter Process Communications). The above belongs to former kind, while the below belongs to later kind.

A component is a non-trivial, nearly independent and replaceable part of a system that fulfils a clear function in the context of a well-defined architecture. It offers the following features:

- Independent and replaceable part of the system
- Performs a particular function
- Works on a well-defined architecture
- Communicates with the help of interfaces

Thus, component based software engineering is a process of integrating the various software components to form an application to satisfy a functionality. This approach is different and easy going from the other traditional approaches. The components can be developed in different languages and on different platforms which are further integrated to form a particular software system. The Component based approach can be elaborated using the following diagram:

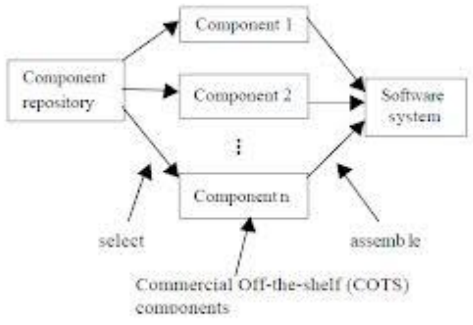

Figure 1. Component-based software development

Component based systems are easier to assemble and therefore less costly to build than developing such systems from scratch. The importance of component based development lies in its efficiency.

II. COMPONENT BASED PROCESS MODEL

The creation of software is characterized by change and instability and therefore any diagrammatic representation of the component-based process model should consider overlapping and iteration between its phases. A consensus may be drawn on the phases pertinent to a software life cycle. Although the main phases may overlap each other and iteration is also possible, the planned phases are: system analysis, domain analysis, design (static and dynamic) and implementation. Maintenance is an important operational phase, in which bugs are corrected and extra requirements met.

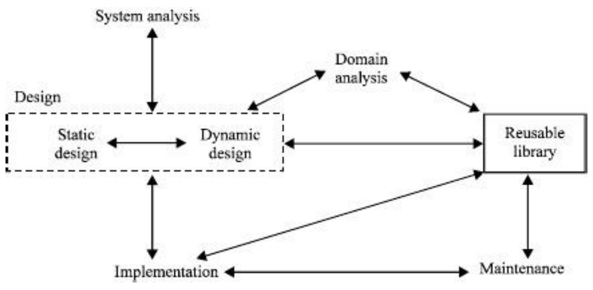

Figure 2. CBSE process model

The component-based software life cycle (CSLC) is the life cycle process for a software component with an emphasis on business rules, business process modelling, design, construction, continuous testing, deployment, evolution, and subsequent reuse and maintenance. In general, analysis and design phases for component-based process models take more time than traditional ones take.
Reusability within this life cycle is smoother and more effective than within the traditional models because it integrates at its core the concern for reuse and its mechanisms**.**

Systems analysis is "the process of studying a procedure or business in order to identify its goals and purposes and create systems and procedures that will achieve them in an efficient way".
The system analysis phase demands the system analyst to:

- Study the application and its constraints
- Understand the requirements expected to be satisfied by the software system
- Create an abstract model of the application in which these requirements are met

The services delivered by a software system at this stage helps to figure out its subsystems and major components. The result of this phase is an abstract model of the application.
A. Domain Analysis:

Domain analysis, or product line analysis, is the process of analysing related software systems in a domain to find their common and variable parts. It is a model of wider business context for the system. The term was coined in the early 1980s by James Neighbors. It is a key method for realizing systematic software reuse. During this phase, the abstract model of the application comprising high-level abstractions of software components may be refined and new components can be defined. Therefore, the boundary between system analysis and domain analysis may at times seems fuzzy because identifying key abstractions in the application domain may be viewed as part of system analysis or domain analysis. Nevertheless, at this level, domain analysis is also concerned with the identification and organization of potentially reusable components.
B. Design:

In the design phase the architecture is established**.** This phase starts with the requirement document delivered by the requirement phase and maps the requirements into architecture. The architecture defines the components, their interfaces and behaviors. The design may include the usage of existing components. The construction of the design model involves identifying relevant components and producing both the static design and the dynamic design. The static design captures the generic and essential features of a system and can be expanded to other systems within the same application domain. In contrast, the dynamic design captures behavioural aspects of a certain system and is therefore more difficult to generalize to other systems. The components should undergo further refinements until they become generic and robust enough to be placed in a reusable library.

C: Implementation:

Implementation refers to the final process of moving the solution from development status to production status. The implementation phase is the translation of a design model into a programming language. The design model comprises static concepts and dynamic behaviour represented by the output of the design phase. In this phase the major tasks involve the implementation of the identified components, along with the cooperation among them.

III. OBJECTIVES OF COMPONENT BASED MODEL

The particular objectives of software components are to:
a. To have a useful ‘replaceable property' i.e. easy to assemble and easy to disassemble.
b. Increase Reusability: Develop once and reuse several instances of the same over the period.
c. Facilitating System Change Management and System Maintenance: The plug and play feature of a component allows easy component composition and inclusion in the information systems.
d. Enhancing Development Flexibility: Components are an independent software element that can be designed and developed independently enhancing the development flexibility.
e. Reduced System Development Time: Reusing pre developed existing components instead of new fresh development will reduce total development time.
f. Reduced System Development Cost: Reduced development time will result in significant reduction in total development cost.
g. Improve Software Quality: Ideally, a component is pre-tested for errors and quality parameters. Hence, using such pre-tested, high quality software components improves the quality of complete software systems.
h. Reducing Project Risk: From management perspective, if an asset's costs can be optimized through a large number of uses, it would then be possible for the management to expend more effort and allocate more budgets to improve the quality of software components. This in turn reduces the level of risk faced by the development effort and will undeniably improve the likelihood of success
i. Improve interoperability: When systems are developed using reused components, they are expected to be more interoperable as they rely on common mechanisms to implement most of their functions
j. Increase System Learning for User: Dialogs and interfaces used by these systems would be similar and would improve the learning curve of users who utilize several different systems built using the same components.

IV. ENGINEERING OF COMPONENT BASED SYSTEMS

CBSE is similar to conventional or object-oriented software engineering. The process begins when a software team establishes requirements for the system to be built using conventional requirements elicitation techniques. An architectural design is established, but rather than moving immediately into more detailed design tasks, the team examines requirements to determine what subset is directly amenable to composition, rather than construction. Now let us focus on few basic definition regarding software components.

- Component—it is a nontrivial, nearly independent and replaceable part of a system that fulfils a clear function in the context of a well-defined architecture.
- Run-time software component—it is a dynamic bindable package of one or more programs managed as a unit and accessed through documented interfaces that can be discovered in run time. Software component—it is a unit of composition with contractually specified and explicit context dependencies only.
- Business component—It is the software implementation of an "autonomous" business concept or business process. In addition to these descriptions, software components can also be characterized based on their use in the CBSE process. In addition to COTS components, the CBSE process yields
- Qualified components—assessed by software engineers to ensure that not only functionality, but performance, reliability, usability, and other quality factors conform to the requirements of the system or product to be built.
- Adapted components—adapted to modify (also called mask or wrap) unwanted or undesirable characteristics.
- Assembled components—integrated into an architectural style and interconnected with an appropriate infrastructure that allows the components to be coordinated and managed effectively.
- Updated components—replacing existing software as new versions of components become available

V. CBSE FRAMEWORK ACTIVITIES

If the requirement(s) cannot be changed or deleted, conventional or object-oriented software engineering methods are applied to develop those new components that must be engineered to meet the requirement(s). But for those requirements a different set of software engineering activities commences:

a. Component Qualification: System requirements and architecture define the components that will be required. Reusable components are normally identified by the characteristics of their interfaces. That is, "the services that are provided and the means by which consumers access these services". But the interface does not provide a complete picture of the degree to which the component will fit the architecture and requirements. The software engineer must use a process of discovery and analysis to qualify each component's fit.

b. Component Adaptation: we noted that software architecture represents design patterns that are composed of components (units of functionality), connections, and coordination. In essence the architecture defines the design rules for all components, identifying modes of connection and coordination. In some cases, existing reusable components may be mismatched to the architecture's design rules. These components must be adapted to meet the needs of the architecture or discarded and replaced by other, more suitable components.

c. Component Composition: Architectural style again plays a key role in the way in which software components are integrated to form a working system. By identifying connection and coordination mechanisms (e.g., run-time properties of the design), the architecture dictates the composition of the end product.

d. Component Update: When systems are implemented with COTS components, update is complicated by the imposition of a third party (i.e., the organization that developed the reusable component may be outside the immediate control of the software engineering organization).

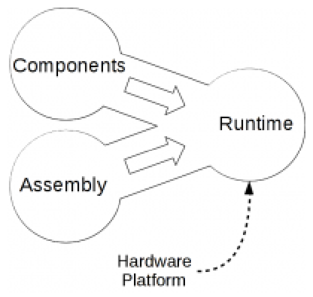

FIGURE 3. CBSE FRAMEWORK

VI. COMPONENT SOFTWARE PROPOSED BY INDUSTRIES

Due to reuse of CBSE on the software industry, a number of major companies and industry consortia3 have proposed standards for component software:

A. OMG/CORBA:

The **Common Object Request Broker Architecture** (**CORBA**) is a standard defined by the Object Management Group (OMG) designed to facilitate the communication of systems that are deployed on diverse platforms. CORBA enables collaboration between systems on different operating systems, programming languages, and computing hardware. CORBA has many of the same design goals as object-oriented programming: encapsulation and reuse. CORBA uses an object-oriented model although the systems that utilize CORBA do not have to be object-oriented. CORBA is an example of the distributed object paradigm. CORBA uses an interface definition language (IDL) to specify the interfaces that object present to the outer world. CORBA then specifies a *mapping* from IDL to a specific implementation language like C++ or Java.

B. MICROSOFT COM:

Microsoft COM (Component Object Model) technology in the Microsoft Windows-family of Operating Systems enables software components to communicate. COM is used by developers to create re-usable software components, link components together to build applications, and take advantage of Windows services. COM objects can be created with a variety of programming languages. Object-oriented languages, such as C++, provide programming mechanisms that simplify the implementation of COM objects. The family of COM technologies includes COM+, Distributed COM (DCOM) and ActiveX® Controls.

C. SUN JAVA BEANS Components:

The Java Bean component system is a portable, platform independent CBSE infrastructure developed using the Java programming language. The JavaBean system extends the Java applet4 to accommodate the more sophisticated software components required for component-based development.

VII. CONCLUSION

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 parties. Developing software using Commercial off the Shelf reusable component is known as Component Based Software Development (CBSD). Informally, application of Software Engineering principles and practices in CBSD is known as Component Based Software Engineering (CBSE). We may conclude that, component based development is the future development process to cater the present software crisis. Industries must follow this development practices, built and enlarge their component library for future reuse. At the same time industries must train their developers to encourage and practice the component based software development process and need to set up appropriate facility centres for supporting CBSD process

REFERENCES

[1] William B. Frakes and Kyo Kang, Software Reuse Research: Status and Future, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 31, NO. 7, JULY 2005

[2] M. D. McIlroy, Mass Produced Software Components , NATO Software Engineering Conference Report, Garmisch, Germany, October, 1968, pp. 79-85.

[3] Brown, A. W., 'Preface: Foundations for component-based software engineering', in Component-based Software Engineering - Selected Papers from the Software Engineering Institute, A. W. Brown (ed), 1996, pp. vii - x.

[4] V. Lakshmi Narasimhan, P. T. Parthasarathy, and M. Das, “Evaluation of a Suite of Metrics for Component Based Software Engineering (CBSE)”, Issues in Informing Science and Information Technology Volume 6, 2009

[5] Antonia Bertolino and Raffaela Mirandola, “Towards Component-Based Software Performance Engineering”.

[6] Boertin, N., Steen, M., Jonkers., H, “Evaluation of Component-Based Development Methods”. In EMMSAD’2001, Sixth CAiSE/IFIP8.1, 2001..

[7] Alejandra Cechich and Mario Piattini-Velthuis, “Component-Based Software Enginnering”, proceedings of The European Journal for the Informatics Professional UPGRADE Vol. IV , No 4 , August 2003, pp.15-19.

[8] Xia Cai, Michael R.Lyu, Kam-Fai Wong and Roy Ko, “ComponentBased Software Engineering: Technologies, Development Frameworks, and Quality Assurance schemes”, IEEE Computer Society, 1530-1362/00, 2000, pp.372-379

[9] Fu Lingyun, “An Approach for Component-Based Software Development”, International Forum on Information Technology and Application.2010,pp.22-25