Component based Software Development- New Era with new Innovation in Software Development

Lata Nautiyal
Assistant Professor
Graphic Era University,
Dehradun, India

Umesh Kumar Tiwari
Assistant Professor
Graphic Era University,
Dehradun, India

Sushil Chandra Dimri
&
Shshidhar G. Koolagudi
Professor
Graphic Era University,
Dehradun, India

ABSTRACT

Traditional software development technology could not catch up with the speed of the many existing and proposed techniques for software development, it seems clear that component-based software development (CBSD) will be at the vanguard of new approaches to the production of software systems and holds the guarantee of substantially enhancing the software production and maintenance process. But the fundamental problem with CBSD is the Selection and Customization of components to meet the requirements of the proposed software. In this paper we are proposing a Selection and Customization Framework for CBSD. In CBSE, selection and composition of components require their interface without showing their idiosyncrasies. This methodology is very similar to the concepts of Object Oriented methods, but the Object Oriented approaches focus on inheritance rather than reusability. We have categorized and prioritized components according to their participation in the software development.

General Terms

Development, Object Oriented Methods, Reuse

Keywords

CBSD, Selection, Customization, Framework, Traditional Software Engineering, Component Model

1. INTRODUCTION

As we know that today software plays a vital role. In this age of information and technology it's not only a product, but it creates markets for the new products. It is a product as well as a vehicle to deliver those products. Today it's not only branch of computer science but is the generator of new and innovative fields. Now, amendment is a clasp of software tribe. As shown in Fig 1, we are witnessing a historical encounter of change of environment and change of software technology.

Widespread use of the personal computers and the Internet make computers commodity supplies and created new market and users. This requires substantial change to software industry. New users, most of them are consumers; require to considerably reducing the price and/or cost of software in order to match the ever-decreasing hardware price. Demands to new applications such as electronic commerce and groupware are high. However, traditional methodologies have not achieved such drastic gain of the productivity and quality yet. We are requested to fundamentally re-think the way of software development.

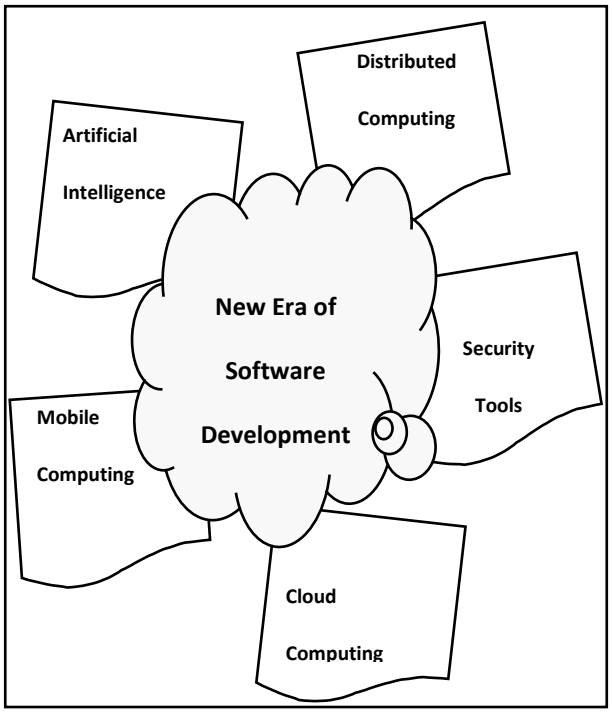

Fig 1: New Era of Software Development.

Since early 1990's, Component-Based Software Engineering (CBSE) have emerged [1] [2].

At the early time, CBSE emphasized on the End-User Computing such as composing applications on the PCs. However, the use of *COTS (Commercial Off-The-Shelf Components)* software promoted the CBSE in the development business applications [2].

Furthermore, quick evolution of the Internet technology such as Web and Java-based technologies even open up new possibilities of CBSE such as network distribution of

components, and the reuse and interoperation of components over the Internet. Now, a few set of technologies have been widely deployed and are still evolving. They include ActiveX/DCOM from Microsoft CORBA [6] from OMG and JavaBeans SUN Microsystems [8].

2. COMPONENT BASED SOFTWARE ENGINEERING

2.1 What is Software Component?

A component is any part of in which something is already made.

In our context this 'something' means system. In software engineering, this would allow a software system to have as components assembly language instructions, sub-routines, procedures, tasks, modules, objects, classes, software packages, processes, sub-systems, etc. The widely accepted goal of component-based development is to build and maintain software systems by using existing software components, e.g. [3], [4], [6], and [7]. They must interact with each other in system architecture. Interaction between these independent components, to achieve their goal is based on some properties:

1. Specified services,
2. Fully explicit context dependencies,
3. Interactive interfaces,
4. Independent deployment,
5. Communication protocols.

Component Characteristics

- Standardized – Follows a standard component model
- Independent – Usable without Adaptors
- Compos able – External interactions use public interface
- Deployable – Stand-alone entity
- Documented – Full Documentation

2.2 CBSE Vs. Traditional Engineering

Traditional Software Reuse and CBSE

Although object-oriented technologies have promoted software reuse, there is a big gap between the whole systems and classes. To fill the gap, many interesting ideas have emerged in object-oriented software reuse for last several years. They include software architecture [4], design patterns [5], and frameworks [9].

CBSE has been broadly used in software development, as it enhances reusability and flexibility, and reduces the costs and risks involved in systems development [10], [11], [12].

CBSE takes different approaches from the conventional software reuse in the following manner.

i.) On the Demand:

Component should be available and we must be able to scale the system with other components and/or frameworks so that component can be composed at any time without compilation.

ii.) Interface-Implementation Abstraction:

Idiosyncrasies of implementation details must not be shown during providing interface. Implementation and Interface must out of reach from one another so that interface can be composed without knowing their implementation.

iii.) Predefined Frame work:

Components are designed on a pre-defined frame work so that they can interoperate with other components and/or frameworks.

iv.) Standardization:

Component interface should be standardized so that they can be manufactured by multiple vendors and widely reused across the corporations. This will increase competition and availability of components in the market.

v.) Distribution through Market:

Components can be acquired and improved though competition market, and provide incentives to the vendors.

The CBSE generally embodies the following fundamental software development principles:

- CBSE is a reuse-based approach to define and implement loosely coupled components into systems.
- During the CBSE process, the processes of requirements engineering and system design are interleaved.
- Truly applying CBSE is not easy. One has to think about potential risks and carefully to plan ahead

3. COMPONENT BASED SOFTWARE DEVELOPMENT

The nature of CBSE suggests that the model of component-based software development should be different from the traditional development model. Table 1 summarizes major characteristics of traditional software development and component-based software development, which are briefly discussed in the following sections.

Table 1. Comparison of Traditional VS CBSE

Characteristics	Traditional Software Engineering	Component Based Software Engineering (CBSE)
Architecture	Huge	Modular
Components	Implementation & White-Box	Interface & Black-Box
Process	Big-bang & Waterfall	Evolutional &Concurrent
Methodology	Build from Scratch	Composition from COTs and Others
Organization	Big Industries	Specialized: Component Vendor, Broker, & Integrator
Emphasis is on	Process identification, Domain Specific people	Domain Specific Component identification
Deliverables	Sequential, Incremental or Evolutionary	Working core
Foundation basis	Inheritance of Properties and Abstraction	Reusability and Abstraction
Risk Factor	People, Resource availability	Component availability, Composition Framework

3.1 Architecture

CBSE emphasizes modular architecture so that we can partially develop a system and incrementally enhance the functions by adding and/or replacing components. To make such design possible, we need a sound foundation of software systems, that is, software architecture. Most component-based systems assume underlying software architecture such as MFC (Microsoft Foundation Class) and CORBA. They are provided in the form of frameworks. Frameworks are workable reference to the underlying software architecture.

3.2 Process

CBSE makes software development and delivery is evolutional. Since some parts of a system can be acquired from the component vendors and/or be outsourced to other organizations, some parts of software process can be done concurrently.

Architecture of Software Process

To make software reuse happen, software process should be reuse-oriented so that designers can reuse artifacts at different levels of abstraction along with software process. Fig. 2 illustrates an example of CBSE process.

CBSE process consists of two processes; component development and component integration. Since these two processes can be done by different organizations, these two processes can be concurrent.

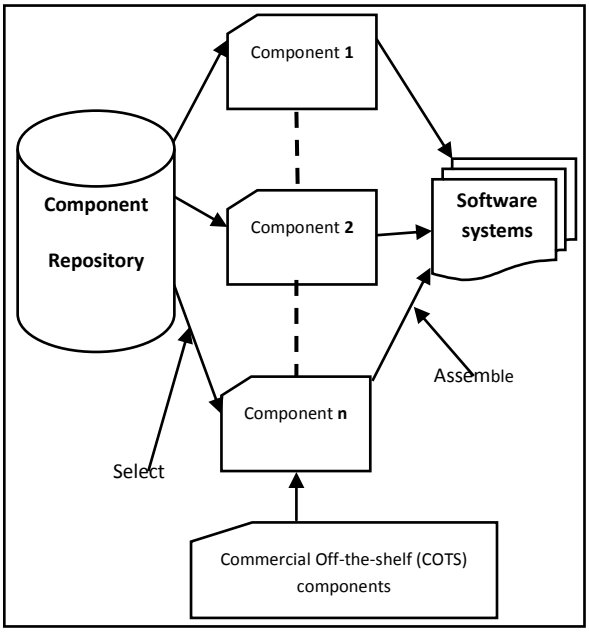

Fig 2: Traditional CBSE approach

Unlike traditional process, CBSE process needs a new process for component acquisition.

4. PROPOSED METHODOLOGY

Fig. 3 illustrates an overview of development methodologies of CBSE. As illustrated, methodologies need to deal with component selection from repository, component development, integration, regression and component composition.

A component-based software system can be obtained as a result of the composition of some components with defined interfaces [13], [14].

In CBSE, selection and composition of components require their interface without showing their idiosyncrasies. This methodology is very similar to the concepts of Object Oriented methods, but the Object Oriented approaches focus on inheritance rather than reusability.

Composition of components needs a defined and common behavior (based on their application domain) of multiple components. That is, CBSE methodologies emphasize on Interface-Implementation Abstraction and Application-

Domain- oriented design such as Catalysis [4] and connection-oriented programming [1].

Component Identification and Design: Based on the Application domain and Requirements we must identify suitable and application specific components. The component identification consists of: Component Selection, Component Prioritization, and Component Customization.

Component Selection: As per the application domain and gathered requirements, Components must be selected from the Component warehouse repository. This selection may include the Exact Components availability, Extensible/Modifiable Components Availability, Non-Available Components.

1. ***Exact Components Availability:***

These are the components which fulfill our up to the mark requirements. They may be used as it is, without any modification. It may be possible that more than one component from more than providers are available. According to our application domain and proper market research we can select these components.

2. ***Extensible/Modifiable Components Availability:***

These components are same as the class inheritance in Object Oriented Software's; where we do not have to start from the scratch instead we can acquire an available component and make modifications or extension as per our requirement.

3. ***Non-Available Components:***

It may be possible that we have to develop some components from the scratch. Those components which are not available in the repository will come in this category.

Component Prioritization: Only component selection is not the sole criteria to identify the components. After component identification we can prioritize the components according to the applications requirement and user needs. Component prioritization may help when we want to deliver our software in increments. We can prioritize our requirements and then to deliver most basic functional software, we prioritize our components. Component prioritization may include: Components Fulfilling Basic Requirements, Most Available Components, and Frame Work Components.

1. ***Components Fulfilling Basic Requirements:***

We can prioritize components according to our requirements. We can select those components which will help in delivery of first increment of the software. These components are selected after the prioritization of requirements first. The most basic requirements are delivered first, and then next increment requirements are delivered and so on. According to these requirement priorities we can prioritize our components.

2. **Most Available Components:**

Second prioritization criteria may be the availability of the components in our repository. We can deliver our software based on the easily available components, to quick delivery of the software. It will help in faster integration and rapid development of the software. After Integrating and delivering available components we can go for modifiable components and then development of new components.

3. ***Frame Work Based Components:***

Next criteria may be based on the prioritization of components which are fulfilling our Frame Work structure of the software. It may include either selection of available component or development of new components.

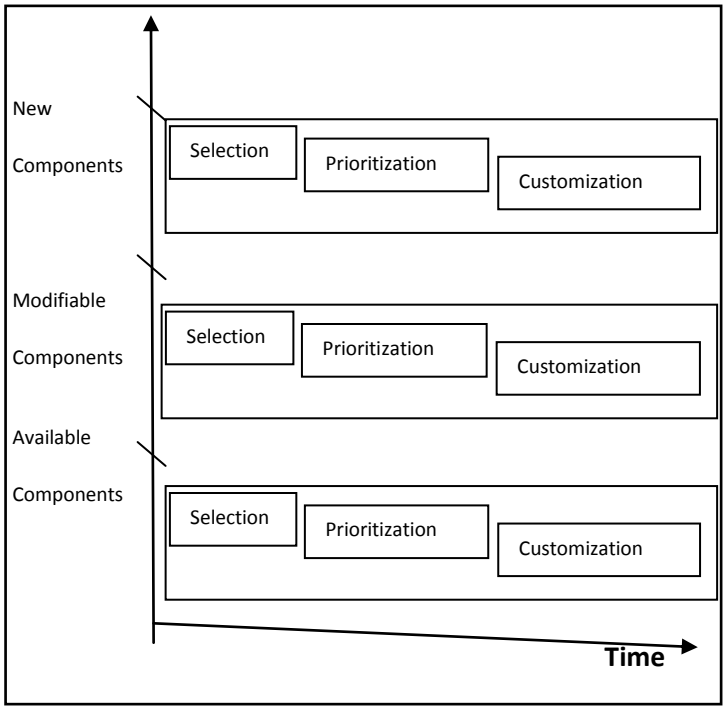

Fig 2: Proposed CBSE Methodology

Component Customization: Component customization is the process of addressing the issues related to the selection and deployment of optimum components. We cannot integrate too much components, since they will ultimately increase the integration complexity of the software. Component selection must be based on some criteria, standards and norms. Component Customization may consist of:

1. Identifying components providing Same Functions,

2. Merging components providing Related Features,

3. Removal of Ambiguous Featured Components,

4. Removal of Extraneous Interactions among components.

5. CONCLUSION

With CBSE, we can change the way of software development. Software development should be with modular process, modular architecture and specialized association so that we can accumulate our technology and expertise.

CBSE can be a fundamental technology for software development so that it requires re-thinking o f a variety of aspects of software development. Besides technical issues, non technical issues such as commerce of components and management issues are also important.

To make our dreams come true, new technology and professional will be desirable.

ACKNOWLEDGMENT

We would wish to thank Dr. R.C. Joshi, Honorable Chancellor Gaphic Era University, for his valuable support and guidance. Also a great Thanks for our President Honorable Professor Kamal Ghanshala, for providing such a research oriented platform for us.

REFERENCES

[1] C. Szyperski, Component Software, Addison-Wesley, 1998.

[2] M. Aoyama, *Componentware: Building Applications with Software Components*, J. of IPSJ, Vol. 37, No. 1, Jan. 1996, pp. 71-79 (In Japanese).

[3] Lata Nautiyal, Umesh Kumar Tiwari, Sushil Chandra Dimri, Shivani Bahuguna, *Elite: A New Component-Based Software Development Model*, International Journal of Computer Technology & Applications, Vol 3, Issue 1, Jan 2012, pp 119-124

[4] D. F. D'Souza and A. C. Wills, *Objects, Components and Frameworks with UML: The Catalysis Approach*, Addison Wesley, 1998

[5] M. E. Fayad and D. C. Schmidt (ed.), *Object-Oriented Application Frameworks*, CACM, Vol. 40, No. 10, Oct. 1997.

[6]. Ning, *A Component-Based Software Development Model*, Proc. COMPSAC '96, Aug. 1996, pp. 389-394.

[7] A. Thomas, Enterprise JavaBeans: *Server Component Model for Java*, White Paper, Dec. 1997, http://www.javasoft.com/products/ejb/

[8] I. Sommerville. *Software Engineering (6th Edition*). Addison-Wesley, 2001.

[9] M. Vidger. *The Evolution, Maintenance and Management of Component Based Systems*. Boston: Addison-Wesley, 2001. [12]

[10] Vitharana, P., Zahedi, F.M., Jain, H, "Design Retrieval and Assembly in Component-Based Software Development", *Communications of the ACM*, 46 (11), pp. 97-102.

[11] Basili, V.R., Boehm, B. (2001), "COTS-Based Systems Top 10 List", *IEEE Computer*, 34 (5), pp.91-93.

[12] Jianguo Chen et.al, "Complexity Metrics for Component-based Software Systems", *International Journal of Digital Content Technology and its Applications*, 5 (3), pp. 235-244, 2011.

[13] Shepperd M, "A critique of cyclomatic complexity as software metric", Software Engineering Journal, March 1988.

[14] Capretz, L.F, "Y: A New Component-Based Software Life Cycle Model", Journal of Computer Science, 1 (1), pp. 76-82.