International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 03 Issue: 05 | May-2016 www.irjet.net p-ISSN: 2395-0072

Component Based Software Engineering

Sharanjit Singh[1], Amardeep Singh[2], Samson[3], Sandeep Singh[4]

[1]*Professor, Department of Computer Science, GNDU Regional Campus Gurdaspur, Punjab, India*
[2]*Professor, Department of Computer Science, GNDU Regional Campus Gurdaspur, Punjab, India*
[3]*Student (M.TECH), Department of Computer Science, GNDU Regional Campus Gurdaspur, Punjab, India*
[4]*Professor, Department of Computer Science, GNDU Regional Campus Gurdaspur, Punjab, India*

Abstract—*Component Based Software Engineering is an approach in which software system is developed through assembly of components. Development of parts as reusable entities is desired. In this paper we have discussed about the component based software life cycle. CBSE technology risks are discussed and model of RAM process which is used to control the risks is described. Embedded system implementation using component based approach is also discussed.*

Keywords —COTS, Components, ASL, Interfaces, RAM.

I. INTRODUCTION

Modern software systems become more and more large-scale, complex and uneasily controlled, resulting in high development cost, low productivity, unmanageable software quality and high risk to move to new technology. Therefore, there is a growing demand of searching for a new, efficient and cost-effective software development paradigm. One of the most promising solutions today is the component-based software engineering approach. This approach is based on the idea that software systems can be developed by selecting appropriate off-the-shelf components and then assembling them with a well-defined software architecture. Over the last few years, we have seen the focus of software development shift from the generation of individual programs to the assembly of large number of components into systems or families of systems. The two central themes that emerged from this shift are components, the basic system building blocks, and architectures, the descriptions of how components are assembled into systems. There is the use of commercial-off-the- shelf (COTS) products as system components. COTS means to refer to things that one can buy, ready-made off some manufacturer's virtual store shelf e.g. through a catalogue. It carries with it a sense of getting, at a reasonable cost, something that already does the job. It replaces the difficulties of developing one's own unique.

These commercial off-the shelf(COTS) components can be developed by different developers using different languages and different platforms. This can be shown in Figure 1, where COTS components can be checked out from a component repository, and assembled into a target software system[5].

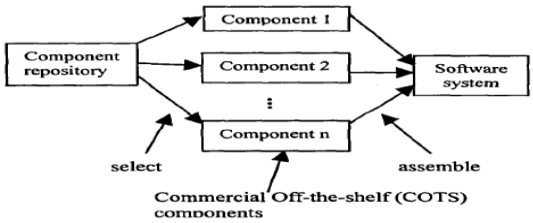

Fig. 1. Component based software development

Component-based software engineering becomes a new approach in software development. The idea behind the component-based approach is designing the desired system in terms of components. The advantage of the component-based approach is providing the reusability of these components. Moreover, if the component is the logical unit of work, maintenance, testing and updating systems using these components will be easy and fast. With component-based software engineering (CBSE), it may be argued that software development risk is reduced, as one is reusing existing tried-and-tested software rather than software developed from scratch.

II. PROBLEM

The development of software systems from existing components continues to hold the attention of the software engineering community. Problem is to select those components so to reduce cost and development time, while increasing the quality of systems. Component Based Software Engineering or CBSE represents a new development paradigm: assembling software systems from components. Reuse of components should be increased.

III. RELATED WORK

Jim Q. Ning, 1997, has discussed the technology infrastructure necessary to support CBSE. Component-based Architecture Specification Language called ASL is also described in this paper. Interfaces and specification of components is also discussed. Ivica Crnkovic, 2003, has discussed established methodologies and tool support covering the entire component and system lifecycle including technological, organisational, marketing, legal,

and other aspects. Advantages of cbse is also discussed in this paper. Xia Cai, Michael R. Lyu, Kam-Fai Wong Roy KO 2000, discussed the current component-based software technologies, describe their advantages and disadvantages, and discuss the features they inherit. Researchers also address QA issues for component-based software. As a major contribution, they propose a QA model for component-based software development, which covers component requirement analysis, component development, component certification, component customization, and system architecture design, integration, testing, and maintenance. W. Lam, A.J. Vickers 1997, discusses the risks associated with the adoption of CBSE technology. A model of the risk analysis and management (RAM) process for CBSE technology is proposed as a means of controlling risks. Five CBSE technology risk areas are identified – domain inadequacies, shortfalls in reuse components, shortfalls in the architecture, deficiencies in the CBSE infrastructure and educational issues - and examined. A number of risk management techniques are proposed. Mohammed A. Abdallah 2008, discusses the use of CBSE in embedded systems. This paper considers the basic overview of component-based model and general issues about embedded systems. The core issue is providing an example showing that how useful to implement an embedded system using the component-based software engineering.

IV. DETAIL DESCRIPTION

Component based software engineering is a approach which mainly depends on building systems from the existing components and, providing support for the development of systems as assemblies of components.

A. Basic Principles of the Component-based Software Engineering

1) *Reusability*: It means that the same component can be used in many systems. The desire to reuse a component leads to some technical constraints such as: good documentation should be available to be able to reuse a component as well as a well-organized reuse process and the similar architecture of the components should be provided to ensure the consistency and the efficiency of the system.

2) *Substitutability*: The overall system should work in spite of which component is used. There are some limitations in this area such as: the runtime replacement of the components.

3) *Extensibility*: Extensibility can take one of two shapes either extending components that are part of a system or increase the functionality of individual components. But there are technical challenges such as: design-time.

B. Characteristics of the Component-Based Software Engineering

Component-based software development is a new way for more flexibility of software generation, composition and integration. The main characteristics of the components are -

- components general do something useful,
- a small related set of functions or services,
- real OO programs are component based,
- classes are not components,
- components are composable,
- frameworks often define component families.

C. Architecture Specification Language.

ASL stands for Architecture Specification Language. Its key features are interfaces, components, bindings, and configurations.

1) Component Interfaces: Component is a functional unit of system and its functionality is defined by its interfaces[1]. A component has one or more provided and required interfaces. Provided interfaces specify those services or capabilities that a component offers to other components. Conversely, the required interfaces specify those services that a component must receive from other components in order to carry out its own responsibilities. A component with required interfaces does not have to be "hard-wired" to specific server components and can therefore be manufactured and distributed for reuse independently. ASL supports features such as operations, attributes, exceptions, multiple inheritance and name spaces. Interface also has additional semantic-oriented features such as pre- and post conditions, invariants, and states. States are used to specify the legal sequences of operation invocations on the interface, i.e., they specify the protocol of the interface.

2) Components and Bindings: A component can be either primitive or composite. A composite component contains other components as its sub-components, which are themselves atomic or composite. Components are connected together to form composite components. A connection is realized by binding a required interface of one component with a provided interface of another component.

- *Configurations*–Configuration specifications provide the information necessary to integrate heterogeneous component instances into an executable distributed system. Configurations address the issue that component implementations have dependencies on their development such as programming languages, object model, etc. and on execution environments such as platforms, middleware, etc. These dependencies are called the component

configuration properties. Default values are assumed for essential properties that are not specified. Multiple configurations may be specified for a single component and used in system assembly.

D) Component-Based System Development Lifecycle

CBSE covers both component development and system development with components. Components are built to be used and reused in many applications. **A** component must be well specified, easy to understand, sufficiently general, easy to adapt, easy to deliver and deploy and easy to replace. The component interface must be as simple as possible and strictly separated both physically and logically from its implementation. Development with components is focused on the identification of reusable entities and relations between them, starting from the system requirements. The early design process includes two essential steps Firstly, specification of a system architecture in terms of functional components and their interaction, this giving a logical view of the systems and secondly, specification of a system architecture consisting of physical components. The different steps in the component-based system development process are:

1) *To find components which may be used in the system.* Components are listed for further investigation. To perform this procedure successfully, a vast number of possible candidates must be available as well as tools for finding them. This is an issue not only of technology, but also of business.
2) *Select the components which meet the requirements of the system.* Often the requirements cannot be fulfilled completely, and a trade-off analysis is needed to adjust the system architecture and to reformulate the requirements to make it possible to use the existing components.
3) *Alternatively, create a proprietary component to be used in the system.* This procedure is less attractive as it requires more efforts and lead-time. However, the components that include core functionality of the product are likely to be developed internally as they should provide the competitive advantage of the product.
4) *Adapt the selected components so that they suit the existing component model or requirement specification.* Some components would be possible to directly integrated in to the system, some would be modified through parameterization process, some would need wrapping code for adaptation, etc.
5) *Compose and deploy the components using a framework for components.* Typically component models would provide that functionality.
6) *Replace earlier with later versions of components.* This corresponds with system maintenance. Bugs may have been eliminated or new functionality added.

The component-based development (CBD) model incorporates many of the characteristics of the spiral model. It is evolutionary in nature, demanding an iterative approach to the creation of software. It composes applications from pre-packaged software components.
The engineering activity begins with the identification of candidate components by examining the data to be manipulated by application. Components created for the previous software engineering projects are stored in the library or repository. Once components are identified, the repository is searched to determine if these components already exists. If they exist, they are extracted from the repository and reused. If the component does not exist, it is engineered. The first iteration of the application to be built is then composed, using components extracted from the repository and any new components built to meet the unique needs of the application. Process flow then returns to the spiral and will ultimately re-enter the component assembly iteration during subsequent passes through the engineering activity.

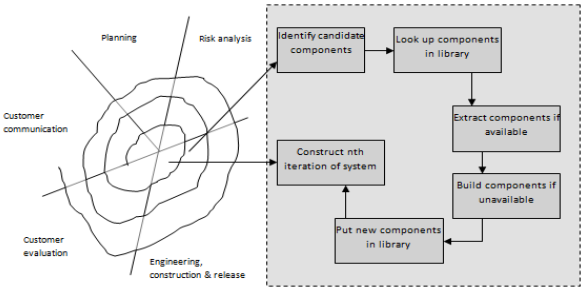

Fig. 2: Component-Based Development

The development cycle for a component-based system is different from those of the traditional systems, such as the waterfall, iterative, spiral and prototype models [2]. Obviously, the development with components differs from traditional development (Larsson, 2000).

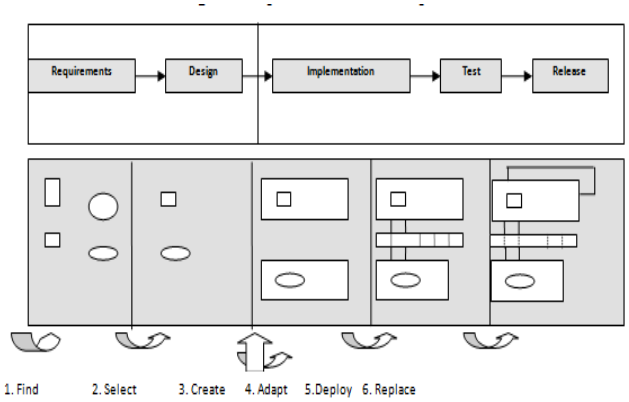

Fig. 3: Component Development Cycle versus Waterfall Model (Larsson, 2000).

Determining requirements and designing in the waterfall process correspond to the finding and selection of components in component-based system. Implementation, test and release phases of waterfall model correspond to create, adapt, deploy and replace process of component-based system[2].

E) Component infrastructure technologies

The infrastructure of components (sometimes called a component model) acts as the "plumbing" that' allows communication among components. Among the component infrastructure technologies that have been developed, some have become somewhat standardized: OMGs CORBA, Microsoft's Component Object Model (COM) and Distributed COM (DCOM).

1) Common Object Request Broker Architecture(CORBA)- CORBA is an open standard for application interoperability that is defined and supported by the Object Management Group (OMG), an organization of over 400 software vendor and object technology user companies. Simply stated, CORBA manages details of component interoperability, and allows applications to communicate with one another despite of different locations and designers. The interface is the only way that applications or components communicate with each other. The most important part of a CORBA system is the Object Request Broker (ORB). The ORB is the middleware that establishes the client-server relationships between components. Using an ORB, a client can invoke a method on a server object, whose location is completely transparent. The ORB is responsible for intercepting a call and finding an object that can implement the request, pass its parameters, invoke its method, and return the results. The client does not need to know where the object is located, its programming language, its operating system, or any other system aspects that are not related to the interface. In this way, the ORB provides interoperability among applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems. CORBA is widely used in Object-Oriented distributed systems including component-based software systems because it offers a consistent distributed programming and run-time environment over common programming languages, operating systems, and distributed networks.[5]

2) Component Object Model (COM) and Distributed COM (DCOM) Introduced in 1993, Component Object Model (COM) is a general architecture for component software. It provides platform-dependent, based on Windows and Windows NT, and language-independent component based applications. COM defines how components and their clients interact. This interaction is defined such that the client and the component can connect without the need of any intermediate system component. Specially, COM provides a binary standard that components and their clients must follow to ensure dynamic interoperability. This enables on-line software update and cross-language software reuse As an extension of the Component Object Model (COM), Distributed COM (DCOM), is a protocol that enables software components to communicate directly over a network in a reliable, secure, and efficient manner. DCOM is designed for use across multiple network transports, including Internet protocols such as HTTP. When a client and its component reside on different machines, DCOM simply replaces the local interprocess communication with a network protocol. Neither the client nor the component is aware the changes of the physical connections.

F) Risks in CBSE technology

Software development is an inherently risky process. For the customer, there is no absolute guarantee that procured software will: a) be delivered on-time and within budget, b) fulfill all its requirements and c) perform without fault. In the following, the risks in each risk area are described in greater detail[4]

1) Domain Inadequacies - The reuse research community **is** currently showing great interest in the notion of domain-specific reuse.

- *Domain not properly scoped* -The boundaries **of** the domain are not explicitly and clearly defined. Its consequence is that not all the components are identified.
- *Domain lacks stability*- Applications in the domain differ significantly ; there is only a slight similarity between applications.
- *Weak domain history* – The organization has little experience of developing applications in the domain.
- *Insufficient economic justification*- Few applications are developed in the domain to cover the up-front costs of developing reusable components.

2) Shortfalls in the components - There are broadly two kinds of components: those which are developed 'in-house' by the organization to serve a specific purpose (often as a result of domain analysis), and more general purpose components, such as databases and OLE (Object linking and embedding) applications that are typically developed by others.

- *Inefficient implementation*- The components leads to an inefficient implementation in code. This may be because in an attempt to make a component generic, the component need to process additional parameters or include large conditional CASE statements.

- *Not trusted*- Application developers do not trust the components. This may be through "bad press" via the organizations grapevine or just the result of a "not developed here" syndrome.

3) Shortfalls in the Architecture- Garlan highlighted the problem of 'architectural mismatch' when attempting to integrate large components which themselves are stand-alone applications in their own right. Clearly, components developed in isolation from each another are unlikely to work together without an agreed management and communication protocol.

- *Not open enough* – It is difficult to update or extend the architecture with extra functionality without affecting the existing components.
- *Inflexible* – The architecture is difficult to adapt a for particular circumstances for use on a variety on different processor configurations.

4) Deficiencies in the CBSE Infra-structure- Successful technology transfer relies on having an infrastructure within the organization for refining new technology for full production use. Deficiencies in the infrastructure are likely to impede the technology transfer process.

- *Lack of defined procedures*- Application developers do not have adequate guidance in the access, selection, customization or general use of components.
- *CBSE technology not properly prepared*- CBSE components, methods, tools , training and documentation have not reached an acceptable level of quality for use on real projects.
- *Conflict with existing methods*- CBSE technology does not integrate with existing methods or tools used within the organization.

5) Educational Issues- Educational issues are often at the centre of cultural issues - one tends to encourages culture shifts through re-education.

- *Lack of managerial commitment* – management fails to see the benefits of CBSE technology to the business and is unwilling or enable to support the development of CBSE technology.
- *Technology not understood*- The technology and its potential benefit is not understood by its key individuals within the organization.

G) Risks Analysis And Management In CBSE

Gilb's risk principle states: "If you don't activity attack the risks, they will actively attack you". Risk Analysis and Management (RAM) is a structured process for controlling risks. During risk analysis, one is concerned with a) the identification of risks b) the estimation of risks (often in terms of a measure of severity), and c) the evaluation of risks. Risk management involves making decisions about the risks after they have been analyzed – one is concerned here with the selection and implementation of risk management techniques, and the monitoring of their effectiveness during the project.

1) *RAM Process:* The RAM process can be described as follows[4]:

- *Identify risk areas:* From experience, the CBSE technology developer identifies and records the main areas of risk in the use of CBSE technology, so that other CBSE technology developers and CBSE technology users are aware of the pitfalls.
- *Estimate risk severity:* The CBSE technology developer assesses the severity of a risk, so that risks of a high severity can be given special priority.
- *Define possible risk management techniques:* Risk areas are related to possible risk management techniques for dealing with specific risks in the risk area.
- *Select appropriate risk management techniques:* The CBSE technology user recognizes the main risk areas in his/her own project, and selects the most appropriate risk management techniques for handling the risks.
- *lmplement the risk management techniques:* The risk management techniques are built into the project plan or 'put into action' during the project.
- *Monitor effectiveness:* The CBSE technology user keeps a vigil over the risks in the project, and monitors how well the implemented risk management techniques are controlling the risks.
- *Add or update risk knowledge:* The organization learns from its experience by identifying new risk areas and /or developing new risk management techniques.

2) Advantages

The component-based approach has some advantages over the traditional programming.

- Advantages from the business point of view in terms of shorter time-to-market, lower development and maintenance costs.
- Advantages from technical and engineering point of view which can be increased understandability of complex systems and increased the reusability, interoperability, flexibility, adaptability, dependability.
- They have advantages from strategic point of view of a society such as increasing software market, generation of new companies.

3) Disadvantages

- Time and effort required for development of' components- Among the factors which . can

discourage the development of reusable components is the increased time and effort required, the building of a reusable unit requires three to five times the effort required to develop a unit for one specific purpose.

- Unclear and ambiguous requirements. In general, requirements management is an important part of the development process, its main objective being to define consistent and complete component requirements. Reusable components are, by definition, to be used in different applications, some of which may yet be unknown and the requirements of which cannot be predicted.
- Conflict between usability and reusability. To be widely reusable, a component must be sufficiently general, scalable and adaptable and therefore more complex (and thus more complicated to use), and more demanding of computing resources (and thus more expensive to use).
- Component maintenance costs. While application maintenance costs can decrease, component maintenance costs can be very high since the component must respond to the different requirements of different applications running in different environments, with different reliability requirements and perhaps requiring a different level of maintenance support.
- Reliability and sensitivity to changes. As components and applications have separate lifecycles and different kinds of requirements, there is some risk that a Component will not completely satisfy the application requirements or that it may include concealed characteristics not known to application developers.[2]

H. Case study

In this section, a research system is introduced with implementing it via the component-based approach. A general overview on this system is introduced to know the purpose of this system and where it can be useful. This system depends on the system - on-a-chip design technology. Systems-on-a-chip design technology provide the computational power to eliminate the need of a computer and interfaces by providing integrated design of processors, on chip memory, and peripheral controllers. State of the art advances in integrated circuit technology, especially in the programmable logic devices, initiated the era of Systems-on-Programmable-Chips. Field Programmable Gate Arrays (FPGA) are modern, complex programmable logic devices (chips) that are manufactured with a high density of configurable blocks.[3]

1) *The Purpose of the System:* This system is considered as a data acquisition system. It can capture analog signals from an environment such as industrial or medical environments without using a PC. Then these analog signals are converted into digital form and then go to some processing such as filtering or amplification. In addition to the capability of plotting these original signals in a Liquid Crystal Module (LCM) for monitoring or testing. This system is designed using the latest advances in FPGA chip design. It eliminates the need of a personal computer and the cumbersome and expensive data acquisition systems currently in use in the field. As shown in this figure, the FPGA board is able to capture the signals and then all signal analysis and processing are take place inside the FPGA. Finally, the original signals can be plotted into this LCM.

2) *Implementation of the System using Traditional Programming* Figure 4 shows how the desired system is implemented via the traditional programming. The language that is used to implement this system is the Verilog. The IDE that used is the Altera Quartus II. The whole code is written in the single module.

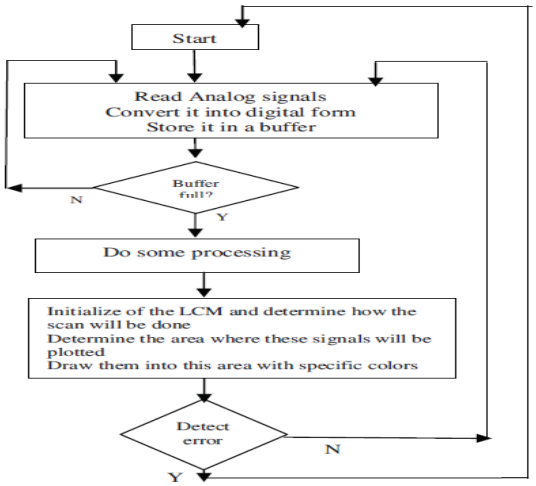

Fig.4. Implementation of the System using Traditional Programming

3) *Component-based implementation:* Without component based implementation, there are many functions listed in the same block and all these blocks were written in the same module. So, this single module is the only responsible of the total task of the system. If any error is occurs, the designer cannot know easily where the location of the error because the whole module is not working although there is one error in a specific part in this module which is faulty. So, if we can collect the functions that do a single kind task in single module, it will be easier to detect the location of the error knowing the nature of it. These collected functions can be done in a component. So, a single component is responsible for a single task and then it is necessary to integrate these components together via interfaces to build the

total system. This idea can be supported in Altera Quartus II using Verilog language. This language deals with modules so, each component is considered as module do a specified task. It is needed to design a top level module that will integrate or woven these components together to achieve the total system requirements in efficient way. Figure 3 illustrates how the traditional programming can be handled in more efficient way if it deals with components. Each block now is corresponding to a single component which in turn is mapped to a single module.

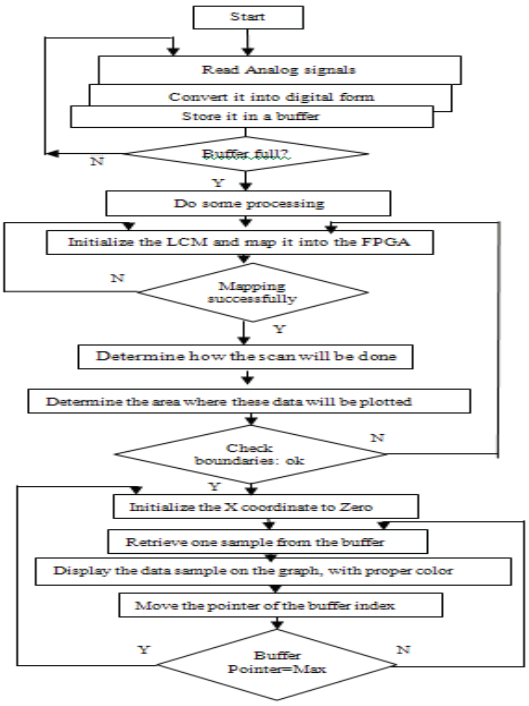

Fig.5. Component-based implementation

V. CONCLUSION

Component based software engineering is a approach of developing software systems by selecting off-shelf components and then assemble the components with well defined software architecture. This approach is used to deploy the reusability in the software engineering and to assure the quality for software development. Model of RAM process is a structured approach which is used for controlling risks involved in the system.

VI. FUTURE SCOPE

CBSE can be widely used by non-programmers for building their applications for which tools can be developed by component assembly. Automatic component update over the Internet, will be a standard means of application improvement. Quality Assurance model can be applied to real world projects so that it can actually guide the practices of component based software development. Standardization of domain-specific components on the interface level will make it possible to build applications and system from components purchased from different vendors.

REFERENCES

[1] Jim Q. Ning, "Component-Based Software Engineering (CBSE)", IEEE 1997.

[2] Ivica Crnkovic, "Component-based Software Engineering - New Challenges in Software Development" , 2nd Int. Conf. information Technology lnterfaces ITI/ 2003, June 16-1 9, 2003, Cavtat, Croatia.

[3] Mohammed A. Abdallah" Implementing the Component-based Software Engineering in Embedded Systems" IEEE 2008

[4] W. Lam, A.J. Vickers "Managing the Risks of Component-Based Software Engineering" ,IEEE 1997.

[5] Xia Cai, Michael R. Lyu, Kam-Fai Wong ,Roy KO" Component-Based Software Engineering:Technologies, Development Frameworks, and Quality Assurance Schemes", IEEE 2000.