International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:3, 2007

Towards a Suitable and Systematic Approach for Component Based Software Development

Kuljit Kaur, Parminder Kaur, Jaspreet Bedi, and Hardeep Singh

***Abstract*—**Software crisis refers to the situation in which the developers are not able to complete the projects within time and budget constraints and moreover these overscheduled and over budget projects are of low quality as well. Several methodologies have been adopted form time to time to overcome this situation and now in the focus is component based software engineering. In this approach, emphasis is on reuse of already existing software artifacts. But the results can not be achieved just by preaching the principles; they need to be practiced as well. This paper highlights some of the very basic elements of this approach, which has to be in place to get the desired goals of high quality, low cost with shorter time-to-market software products.

***Keywords*—**Component Model, Software Components, Software Repository, Process Models.

I. Introduction

COMPONENT based software development (CBSD) refers to the development of software systems making considerable use of software components. CBSD can help the software industry realize productivity and quality gains similar to those achieved in hardware and manufacturing industry. Instead of building software systems from scratch, they are assembled from already developed components. This approach facilitates the development of software within time and budget constraints. It also results in quality and productivity gains[1].

Component-based development has a lot of promises. But it is not a silver bullet. For achieving all the gains, CBSD approach needs to be followed religiously. We cannot get along with the scheme of things used in the traditional way of development of software. There is significant difference in component based software development and traditional software development approach. Technical standardization is necessary, and a method suited to CBD has to be followed.

The effective use of COTS components demands a new way of doing business: new skills, knowledge, and abilities; changed roles and responsibilities; and different processes– and these changes are not happening [2]. Developing a component based software system is a complex activity and for achieving a cost-effective, time bound and high quality solution using this approach, it is required that at least the following elements [3] are in place:

1. Software Components repository
2. A Component Model.
3. Component Based Development Process.

In this paper, an elaborative study regarding these elements is carried out in the following sections.

Manuscript received March 30, 2007.

Kuljit Kaur is Lecturer in department of Computer Science and Engineering, Guru Nanak Dev University, Amritsar, Punjab, India (phone: 011-0183-5089481; e-mail: kuljitchahal@yahoo.com).

Parminder Kaur is a Lecturer in the department of Computer Science and Engineering, Guru Nanak Dev Universsity, Amritsar, Punjab, India.

Jaspreet Bedi is a lecturer in department of computer science and engineering, Guru Nanak Dev niversity, Amritsar, Punjab, India.

Hardepp Singh is Professor and Head, depart of Computer Science and Engineering, Guru Nanak Dev University, Amritsar, Punjab, India.

II. The Software Components Repository

In contrast to the traditional approach of developing software products, component based software development approach is based on integration of already existing software components. So to start with, there should be an available set of software components that can be reused. This Software components repository can be housed by in-house built software components or Software components that can be acquired from third party vendors (known as Commercial off the Shelf Components-COTS). Before developing software application for a problem domain, it is necessary to identify the software components that can be used in multiple applications in that domain. So domain engineering is a key part of the component based development process. This repository of components (also called software reuse library) has to be maintained so that desired components can be retrieved when required. So proper component classification, browsing and, retrieval techniques need to be incorporated in the repository management process. Some of the existing software reuse libraries can be taken as a reference point such as Comprehensive Approach to Reusable Defense Software (CARDS), Defense Software Repository system (DSRS), Asset Source for Software Engineering Technology (ASSET)[13].

III. Component Models

Standards play a big role in formalizing the development process for component based applications. In the absence of standards, even when the required software components are in hand, they can not be used because the chosen parts do not fit together [4]. There must be a backplane in which the components can exist and communicate. There must also be a component model that can support the assembly and

interaction of software components. A component model is the backbone of a component based system which provides basic infrastructure for component based composition, communication, deployment and evolution [3}. The cornerstone of any CBD methodology is its underlying component model which defines what components are, how they can be constructed, how they can be assembled. At present several component models exist such as EJB, CORBA, .NET, Koala, SOFA, Kobra, Architecture Description languages, UML 2.0 [5]. Conformance to a component model can help in the following ways:

A. Independent Extensions

Legacy software lacks flexibility whereas components can be extended and a component model prescribes exactly how extensions are made. Extended components can be even deployed in a running application. The component models and frameworks ensure that that extensions do not have unexpected interactions. Thus extensions may be independently developed and deployed.

B. Components for Third Party Composition

Component models prescribe the necessary standards to ensure tat independently developed components may be deployed in a common environment and any kind of unanticipated interactions will not be experienced.

C. Reduced Time to Market

Use of components that confirm to prescribed standards also reduces the time it takes to design, develop and deploy systems .Time is reduced because key architectural decisions have already been made.

D. Improved Reliability

Component models can be designed to support those quality attributes that are most important in application areas. Component models specify design rules that are uniformly enforced over all components deployed in a component model. This uniformity means that various global properties such as scalability, security and so forth can be predicted for the system as a whole.

The key contribution of component models is the enforcement of architectural principles. By forcing component instances to perform certain tasks, the component model can enforce principles. The use of component models can be an appropriate way of component development. If components are developed independently, it is highly unlikely that they will be able to cooperate usefully.

IV. Component Based Software Development Process

The development cycle of a component based system is different from those of the traditional models such as waterfall, iterative, prototyping and incremental models [6]. CBSD clearly signal a paradigm shift. Building software using pre-existing components is different from typical custom development in the sense that the components are not designed to meet a specific - project requirements. COTS components are built to satisfy the needs of a market segment. Therefore, an understanding of the components' functionality and its evolution over time must be used to modify the user-requirements. Major points of difference of the traditional development and component based development approach are listed below:

A. User Centered

Traditional approaches tend to be developer centered. Although users may initiate requests for applications to be developed, developers control the development process. In the beginning of the development process, users participate to provide information and then later to verify that the end product meets their requirements. But users are not involved in the design process itself. The CBSD paradigm, on the other hand, is more users centered. With the availability of components supporting the general functions (such as User Interface, data storage and retrieval) in the commercial component market, a user does not have to depend upon a development team to develop and maintain a new system. The users may combine their application domain knowledge with fairly limited technical knowledge to produce useful systems.

B. Reuse- Based

Component based software development paradigm focuses on reuse of already existing software components using a systematic reuse approach. The reuse process is interwoven into the thread of development process. Pre-built components are stored in the component repository. Requirements of the system are identified and then negotiated depending on the availability of the reusable software components in the market. The design is also based on existing components. Of course this means that there may be requirements compromises. The design may be less efficient than a special purpose design. However the lower costs of development, more rapid system delivery and increased system reliability should compensate for this [7].

C. Different Processes

Component based software engineering addresses challenges and problems similar to those encountered elsewhere in software engineering. Many of the methods, tools and principles of software engineering used in other types of system will be used in the same or a similar way in CBSE. But however, CBSE focuses on system development from two different perspectives. It distinguishes the process of component development from that of system development using components. There is a difference in requirements and business ideas in these two cases and different approaches are necessary [8].

The two approaches are:

1) Developing reusable components – the producer perspective
2) Developing with reusable components – the consumer perspective

This paper focuses on the consumer perspective only, assuming that "high quality" reusable components exist either in house or can be procured from outside.

D. Design Optimization v/s Design Selection

All software design problems exhibit optimization and selection decisions. Systems developed primarily from custom-built software are dominated by optimization decisions, while systems designed from pre-existing software components are dominated by selection decisions.

Optimization decisions arise when there are too many design options to itemize. In custom-made systems, the designer is free to partition functionality into components of arbitrary scope, to define component interfaces in arbitrary ways, and to select arbitrary mechanisms to support interaction of components. This freedom means there is an infinite number of design options, hence optimization.

Selection decisions arise when there is a bounded and usually small set of a priori design options. In this situation, the fundamental design problem is to select the option that best satisfies specific design qualities. In systems built from pre-existing software components, the designer is not free to define the scope of components, their interfaces, and their interaction mechanisms, as these decisions have already been made by the component developer. This greatly restricted design freedom leads to the primacy of selection decisions.

E. Role of Architecture

Traditionally software architecture is focused on in the early design phase. But at the time of execution, the application reduces to a monolithic piece of software. Architecture of the software application is concealed. Whereas in component based software development approach, application is assembled from components, which remain recognizable even at the time of application execution. So software architecture remains in the picture (an important factor) during the execution phase as well.

F. Early Integration

From a component based perspective the process of system design involves the selection of components, together with an analysis of which components can be acquired from external sources and which ones must be developed from scratch. In contrast to other kinds of systems in where system integration is often the tail end of an implementation effort, in component based systems determining how to integrate components is often the primary task performed by designers.

V. Process Models for Component Based Development

Component Based development can be architecture driven or component driven, that depends upon whether the software architecture is decided first and then components are selected or components are selected first and then architecture is decided to suit this selection. The former approach is more appropriate for component based development as it will result in high levels of reuse and it also conforms to the international standards laid down for reuse based processes.

1) The process model for CBSD as proposed by [10] supports the former approach. In component Based software engineering, Component Based development occurs in parallel with domain engineering. Domain Engineering performs the work required to establish a set of software components that can be reused by the software engineer. Once the architecture has been established, it must be populated by components that are available in the component repository. If required components are not available in the repository, they can be acquired from third party vendors.

2) Sommerville [7] proposes a reuse driven process for application development as shown in Fig. 1. Rather than design then search for reusable components, the reusable components are first searched. In this way the design is based on the components that are available. The system requirements may have to be modified in accordance with the behavior of the available components. This may end in requirement compromises also.

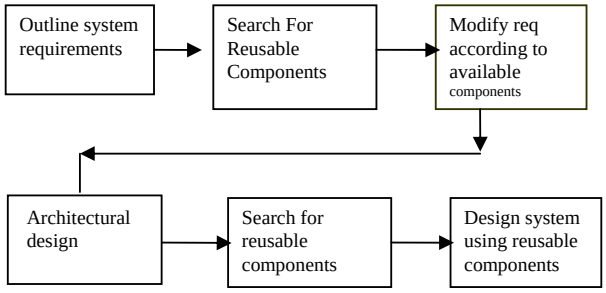

Fig. 1 Software process using reusable artifacts

3) The V-model for component based development approach – In the V-model [6] adopted for component based software development [11], throughout the life cycle of the software product (spanning development and maintenance phases), a component pool is available from which components can be selected in the initial phases and to which components can be added in the later phases (newly developed components for an application). The process starts with requirement engineering, where the requirement engineers try to find, from the component pool, the components that can fulfill the requirements. If such components are not available, then the requirements are possibly negotiated and modified in order to use the available software components.

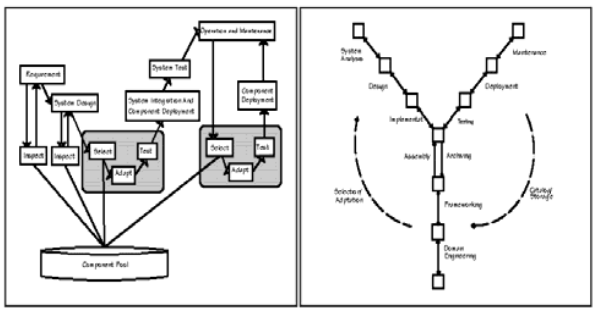

Fig. 2 (a) The adopted V-model [11] and (b) The Y-Model [12]

4) The Y-model –
The Y-model [12] also emphasizes the use of existing component archives. To this component archive, components can be added after the domain engineering exercise, and any new set of components developed for an application as some new requirement arises for which no suitable component is available in the archive. In the initial phases, components can be selected and adapted from this repository.

5) The EPIC approach – The Evolutionary Process for Integrating COTS [2] is a modified form of Rational Unified Process (RUP) [6]. To accommodate the continuous change induced by the COTS marketplace, EPIC uses a risk-based spiral development process. EPIC users manage the gathering of information from the marketplace and the stakeholders and refine that information through analysis and negotiation into a coherent, emerging solution that is embodied in a series of executable representations through the life of the project. Stakeholders actively participate in EPIC as key players in day-to-day negotiations that also continue through the life of the solution [6]. EPIC evolved from a U.S. Air Force need to meet the challenges of building, fielding, and supporting COTS-based business solutions.

The four phases are, as in RUP, Inception, Elaboration, Construction, and Transition.

1. The goal of the Inception Phase is to achieve concurrence among affected stakeholders on the life-cycle objectives for the project. The Inception Phase establishes feasibility through the business case that shows that one or more candidate solutions exist.
2. The goal of the Elaboration Phase is to achieve sufficient stability of the architecture and requirements; to select and acquire components; and to mitigate risks so that a single, high-fidelity solution can be identified with predictable cost and schedule.
3. The goal of the Construction Phase is to achieve a production-quality release ready for its user community. The selected solution is prepared for fielding.
4. The goal of the Transition Phase is to transition the solution to its users. The selected solution is fielded to the user community and supported.

The four EPIC phases are repeated for each solution. Thus across the life of a large or complex project, many solutions–often overlapping–are created and retired in response to new technology, new components, and new operational needs.

In general following activities are to be performed in the life cycle of component based software [13]. The way each activity is performed depends upon a number of factors such like availability of resources, commitment of the developer's team as well as the clients.

Find – The process of finding components defines how to document and create repositories of components. Finding components is an activity in the domain engineering phase. Domain engineers mine families of similar products to document core components.

Select – specific components from the component repository are selected for use.
Adapt – Adaptation is the process of customizing selected components to satisfy user requirements in the new context in which the component is used.
Create – in component based software development, it might happen that selected components do not fit into the application requirements even after adaptation. In such situations the component integrator has to develop and create new components for this specific application.
Compose – Composition is an assembly and integration process. The effort of integration depends on the nature of the component to be integrated.
Replace – The replacement process is related to product maintenance. Component systems evolve over time to fix errors in components and add new functionalities. The old version of the component is swapped out and a new version is swapped in, this is often referred as component upgrade.

Current trends in component based design point toward an increased focus on this type of development approach because it enforces a structured approach to component based development that is expected to deliver same kind of benefits obtained as a result of use of structured programming techniques in computer programs.

VI. Conclusion

No Doubt in the future, the demand for custom applications will still be there. However, given an increase in domain or application specific component models, component driven architectural design will be preferred. It is crucial to follow a proper reuse based process and to use the building blocks (software components) that confirm to standards (component models), in order to come out of the software crisis.

References

[1] Szyperski, C, Component Software: Beyond Object-Oriented Programming, Addison Wesley, 1999.
[2] Cecilia Albert and Lisa Brownsword, Evolutionary Process for Integrating COTS-Based Systems (EPIC): An overview, Technical Report CMU/SEI-2002-TR-009 ESC-TR-2002-009, July, 2002.
[3] Jerry Zeyu Gao, Jacob Tsao, Ye Wu, Testing and Quality Assurance for Component Based Software, Artech House Publishers, 2003.
[4] David Garlan et al, "Architecural Mismatch: Why Reuse is so Hard", IEEE software, 1995.
[5] Kung –Kiu and Zheng Wang, "A survey of Software Component Models", School of computer Science, University of Manchester, April, 2005, available at http://www.cs.man.ac.uk/preprints/index.htm.
[6] Ian graham, Object Oriented Methods, - Principles and practice, 3rd Edition, Addison Wesley, Object Technology Series.
[7] Ian Sommervilee, Software Engineering, 7th Edition, Pearson Education.
[8] Ivica Crnkovic. and M. Larson, 2002, Building Reliable Component Based Software Systems, Artech House, Boston,
[9] IEEE, IEEE standard for information technology – Software life cycle Processes – Reuse processes (IEEE 1517-1999), Piscataway, NJ, 1999.
[10] R.S.Pressman, Software Engineering – A Practioners Approach, Fourth Edition, McGraw Hill International Series.
[11] Ivica Crnkovic, component Based Development Process and Component Life Cycle, 27th International Conference on Information Technology Interfaces, IEEE, Caretat, Croatia, 2005.
[12] Luiz Fernado Caprtz, Y: A New software development life cycle Model, Journal of Computer Science 1(1):76-82, © Science Publications, 2005.
[13] *Hafedh Mili et al*, Reuse Based Software Engineering, Techniques, organization and Controls, John Wiley and Sons, 2002.