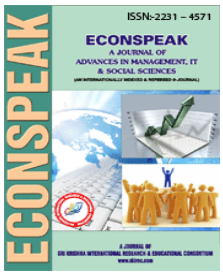
The Journal of Sri Krishna Research & Educational Consortium

ECONSPEAK:
A JOURNAL OF ADVANCES IN MANAGEMENT, IT & SOCIAL SCIENCES

Internationally Indexed & Listed Referred e-Journal



COMPONENT BASED SOFTWARE DEVELOPMENT- THE APPROACH TO COMPOSE A SOFTWARE

SANDEEP KUMAR*

*Galaxy Global Group of Institutions,
Dinarpur, Ambala,Haryana,India.

ABSTRACT

Here is a new approach of software development, called Component Based Software development, where we do not build the software, rather we compose a software from existing components. Using CBSD, we can reuse the existing components that are developed independent of any application. It results in faster development, lesser cost and more usability. Idea is to 'Buy, not Built' new components. So approach and methodology are different from conventional approaches.

INTRODUCTION

Till today, it may be argued whether Software Engineering is actually an engineering field or not. At least it is not pure engineering in its present form. In all traditional engineering fields, a final product is given shape by attaching independent components that may be purchased off the shelf. Take the example of a car manufacturing process. After preparing the design, various components, that are rather independently designed and developed, are attached into an assembly line to give the vehicle its final shape. Even if one component breaks down or stops working, a new component is purchased and the old one is replaced. But this type of development is far away as far as software engineering is concerned. Component Based Development is a step towards this direction. The primary role of component-based software engineering is to address the development of systems as an assembly of parts (components), the development of parts as reusable entities, and the maintenance and upgrading of systems by customizing and replacing such parts. Component-based development requires a systematic approach to and focus on the component aspects of software development [1]. Traditional software engineering disciplines must be adjusted to the new approach, and new procedures must be developed. Component-based Software Engineering (CBSE) has become recognized as such a new sub-discipline of Software Engineering. CBSE is emerging as new paradigm and a coherent engineering practice in software industry [7]

COMPONENT SPECIFICATION

Before we can understand what Component Based Development is all about, we must agree on what a Component is and what a Component is not. There are many definitions of Component in the literature [3] [4], but one given by Szyperski [2] is worth mentioning here:

A software component is a unit of composition with contractually specified interface and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parts.

A Component's implementation is different from its interface. Every component has a 'provides' and a 'require' interface. Through 'Provide' interface it provides its services to other components and through 'Requires' interface it takes the services from other components. Integration of a component into an application is independent of its development life cycle, and whole application should not be recompiled when a new component is attached with it.

WHY CBSD

Main goal of component-based development is to build and maintain software systems by using existing software components, e.g. [5, 9, 2,]. They must interact with each other in system architecture [6, 9, 8,].Question arises Why should we adopt CBSE? The answer lies in the definition itself. Moreover CBSE aims to improve the Quality (Flexibility, Maintainability, Reliability) of software systems and productivity & Time-to-market of software development through enabling the easy assembly of software from existing building blocks. Other reasons can be seen by comparing the conventional design and development with Component Based Software Development. Table 1 below gives these comparisons:

TABLE 1: COMPARISONS OF CONVENTIONAL APPROACH TO CBSD

Cartelistic	**Conventional Approach**	**CBSD**
Architecture- How the system is setup	Monolithic	Modular
Components- the pieces of the system	Implementation and White Box (Programmer is given access to the source code)	Interface and Black Box (Components can not be adapted or changed)
Process- How the system is put together	Big Bang & Waterfall(Analysis to design to implementation to testing)	Evolution and concurrent engineering(Component development to component integration)

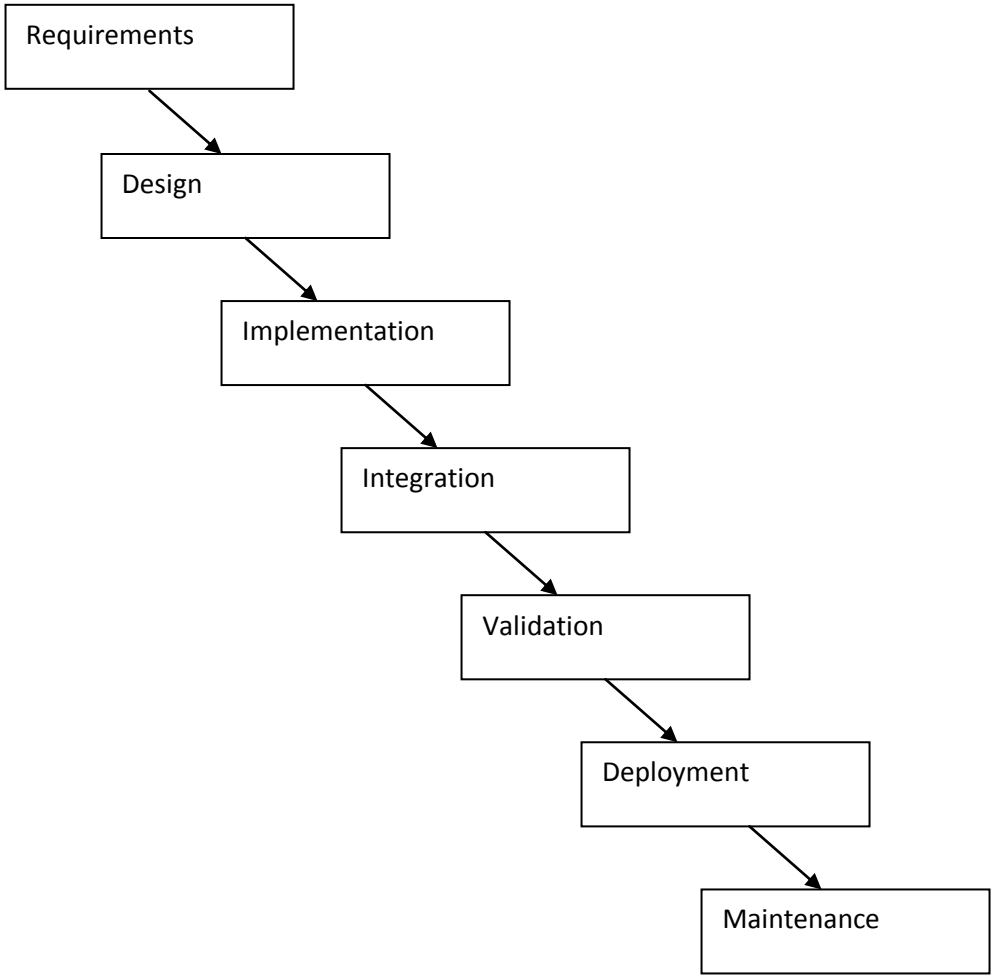
Methodology-How the system changes through time	Built from scratch	Composition
Organization- Market for buying and selling components	None	Specialized- Component vendors and integrators

Basically conventional approach to software development can be compared with CBSD using five parameters. In the context of software architecture conventional approach adopts monolithic development whereas CBSD adopts a Modular approach. The development process followed in conventional approach is generally waterfall model which goes from analysis to testing one phase at a time, whereas in case of CBSD a more robust evolutionary and concurrent engineering model is adopted there is more emphasis on component development and component integration. Another level where conventional methodology differs from CBSD is the methodology. The conventional software are developed from scratch whereas components are composed together to make a new product in case of CBSD. Another advantage is that in case of CBSD there are specialized vendors and integrators for buying and selling the components.

CBSD LIFECYCLE

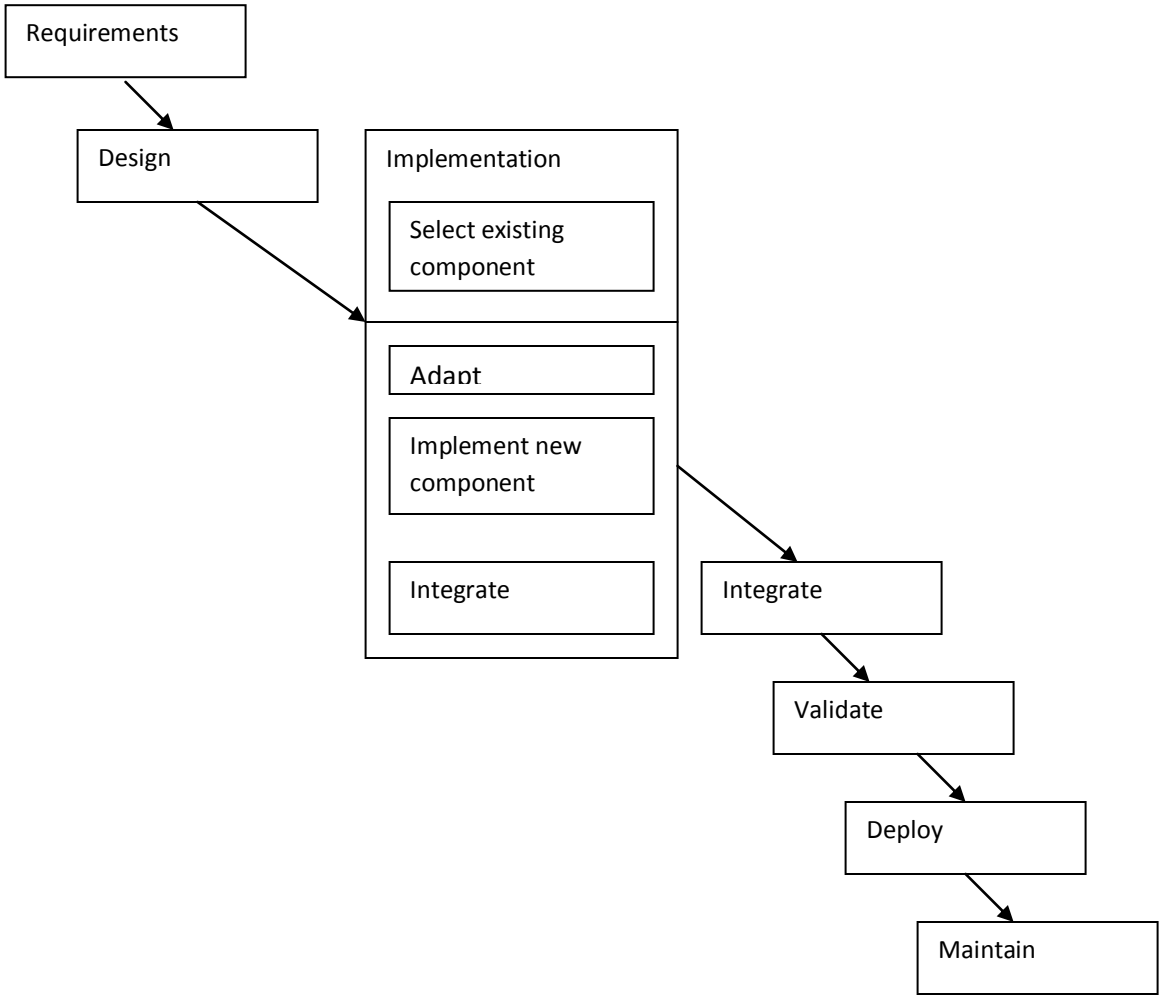
In Component based software development, systems are built from already existing components. So development process of a system is different from that of a component. Quite possible that that component have already been developed and used in some other application before system development starts. A new phase, may be called Component Assessment can be part of the main process, which involves finding and evaluating components for the present application. Waterfall model of software development can be used as a basis for illustrating the component based software process development model. Figure 2 shows the main phases of the waterfall development model.

FIGURE 2: WATERFALL MODEL



In case of Component based systems, the first objective should be to look for the existing components wherever in the market they are available. If they are not available then they should be developed in house. So it should be considered in the requirement and design phase itself whether existing components are available or not. Waterfall model can be modified with implementation phase incorporating the component selection. The modified model is shown in the following diagram, figure 3.

FIGURE 3: WATERFALL MODEL FOR COMPONENT BASED DEVELOPMENT



The difference can be seen at the implementation level. If we can have a component that has already been built and used in some application, that is selected and integrated with the current application or else a component may have to be adapted to the current application. Third alternative is to design and develop a totally new component and then integrate it into the current application.

CBSD: PROS AND CORNS

CBSD has its potential advantages and disadvantages .Advantages include.

- **REDUCED DEVELOPMENT TIME:** It takes a lot less time to buy a component than it does to design it, code it, test it, debug it, and document.

- **INCREASED RELIABILITY OF SYSTEMS:** An off-the-shelf component may have been used in many other systems earlier, and should therefore have had more bugs found out of it.

- **INCREASED FLEXIBILITY:** Positioning a system to accommodate off-the-shelf components means that the system has been built to be immune from the details of the implementation of those components.

This was one side of the coin. It has its own negative side also.

- If the primary supplier goes out of business or stops making the component, other suppliers may not step in to fill the gap

- The vendor may stops supporting the current version of the component, and the new versions may be incompatible with the old

- If the system demands high reliability or high availability, how can the consumer be sure that the component will allow the satisfaction of those requirements?

CONCLUSION

CBSE is the new paradigm of industry. It is changing the way large software systems are developed. CBSE emphasizes on "buy; don't build" philosophy. As early subroutines liberated the programmer from thinking about details, CBSD shifts the emphasis from programming software to composing software systems. Focus is now on integration rather than implementation. At its foundation is the assumption that there is sufficient commonality in many large software systems to justify developing reusable components to exploit and satisfy that commonality. The day may not be far away when software engineering will become true engineering discipline, just like any other engineering field.

REFERENCES

[1] Crnkovic, I. and Larsson, M. A Case Study: Demands on Component-based Development, Proceedings 22nd International Conference on Software Engineering, ACM Press, 2000.

[2] Szyperski C., Component Software –Beyond Object-Oriented Programming. Addison-Wesley, 1998.

[3] Szyiperski C. and Pfister C., Workshop on Component-Oriented Programming,

Summary. In Muhlhauser M. (ed.) Special Issues in Object-Oriented Programming – ECOOP96 Workshop Reader, Springer 1997

[4] ICSE 1999, Workshop on Component-Based Software Engineering (CBSE 2),

http://www.sei.cmu.edu/cbs/icse99/cbsewkshp.html

[5] Schneider J.G. and Nierstrasz O., Components, scripts and glue. In L. Barroca, J. Hall, and P. Hall, editors, Software Architectures Advances and Applications, pages 13 – 25. Springer, 1999.

[6] Shaw M. and Garlan D., Software Architectures: Perspectives of an Emerging Discipline. Prentice Hall, 1996.

[7] Kamiya, T., Kusumoto S., Inoue K., Mohri Y., Empirical Evaluation of Reuse Sensitiveness of Complexity Metrics, pages 297-305, Information and Software Technology Journal, 1999,.

[8] Hoek A., Rakic M., Roshandel R., and Medvidovic N., Taming architecture evolution. In Proceedings of the 6th European Software Engineering Conference (ESEC) and the 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9), 2001.

[9] Medvidovic N. and Taylor R.N., A classification and comparison framework for software architecture description languages. IEEE Transactions on Software Engineering, 26(1):70–93, 2000.

[10] Blevins D., Overview of the Enterprise JavaBeans Component Model, in G. T. Heineman and W. T. Councill (editors), Component-Based Software Engineering: Putting the Pieces Together, Addison Wesley, 2001.

[11] Garlan D., Monroe R.T. and Wile D., Acme: Architectural description of component-based systems. In G.T. Leavens and M. Sitaraman, editors, Foundations of Component-Based Systems, pages 47–68. Cambridge University Press, 2000.