Candidate Process Models for Component Based Software Development

K. Kaur and H. Singh
Department of Computer Science and Engineering,
Guru Nanak Dev University, Amritsar, 143005, India

Abstract: Component Based Software development approach promises high quality, low budget software with shorter time to market. Reuse of the already existing software components (irrespective of their development or implementation platform) is the focal point of this approach. Component Based Software is different from traditional programs. Software development process models used for writing the traditional programs cannot be used for assembling the application using reusable software components. This study points out the differences in these two development paradigms and the inappropriateness of the traditional process models for developing component based software. A process model for this new approach should incorporate the activities to address the important aspects (such as component selection, customization and composition) required for building high quality component based software. The study investigates the component based software development process models in view of the prescribed international standards. It also presents a review of the current industrial practices as far as the software development process models are concerned.

Key words: Software development, component based software development, software process, component based software development process

INTRODUCTION

Component Based Software Development (CBSD) refers to the development of software systems making considerable use of already existing software components. Instead of building software systems from scratch, they are assembled from reusable software components. This approach facilitates the development of software within time and budget constraints. It also results in quality and productivity gains (Szyperski et al., 2000). These reusable software components may be built in-house or acquired from third party vendors for a price (called Commercial Off The Shelf components-COTS) or free of cost (called Open Source Software Components).

The CBSD 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 software development. There is a significant difference in CBSD and traditional software development approach. The traditional software development processes do not address challenges and risks of using reusable components. Best components selected in the beginning of life cycle may suffer from incompatibility problems towards the end. Technical standardization is necessary and a method suited to
CBSD has to be followed. This study gives an account of the various component based development process models available in the research literature. Towards the end, a review of the existing scenario, regarding the use of software process models in the software industry, is also presented.

SOFTWARE DEVELOPMENT PROCESS

Software development process focuses on the activities related to software production, for example design, coding and testing. It consists of a sequence of steps that are always performed in a particular order. The process helps to achieve consistency in a project, which involves a large number of people with different skills. Here we briefly discuss various process models proposed in the research literature (Schach, 2008).

Waterfall model gives a fixed ordering to the basic set of process activities such as requirement analysis, design, implementation and testing. Simplicity of the model is its strength. However, heavy documentation and the constraint of pre-decided fixed set of requirements are some of the weaknesses. Prototyping model helps to understand uncertain requirements. It also reduces the risks involved in developing novel systems. Rapid Application Development (RAD) is a variation of prototyping model. RAD relies heavily on rapid application tools to quickly deliver the product. Iterative enhancement (evolutionary) model combines benefits of both the waterfall and prototyping models. Initially basic functionality of the system is implemented. Quality of the system is improved in next iterations. Agile process model puts less emphasis on analysis and design. Implementation starts much earlier in the life cycle of the software product. This process model demands fixed time rather than fixed features. A new version of software is delivered after every 3 weeks. Extreme Programming is a new software development approach, based on iterative enhancement model. It is a test driven approach. To begin with, programmers design test cases for a new task. Task is implemented and it is ensured that all the test cases are satisfied by the task. Spiral model includes the best features of waterfall, prototype and iterative enhancement models and introduces a new concept - risk assessment and resolution. This new feature makes the spiral model more realistic than others process models. All these process models are suitable for development from scratch. Now days, majority of the software development projects involve some degree of reuse of existing artifacts such as design or code modules. Rational Unified Process (RUP) considers reuse of existing classes in one of its initial phases. But it focuses only on object oriented development. So we need to look for process models which support the reuse of already existing reusable software components irrespective of their development or deployment platform.

Component Based Software Development Process

Boehm and Abts (1999) regard both the waterfall model and evolutionary development unsuitable for component based development because:

- In the sequential waterfall model, requirements are identified at an earlier stage and the software components are chosen at a later stage. This increases the likelihood of the software components not offering the required features
- Evolutionary development assumes that additional features can be added if required. However, software components acquired from third party vendors (COTS) cannot be upgraded for one particular development team. The absence of source code bars the development team from adjusting these software components to their needs
During component based software development, reusable artifacts have to be explicitly considered and the development process has to be tailored to appropriately reflect the reuse-based approach. Reuse-related activities such as component selection and composition have to be modeled as separate process steps rather than being performed under the hood of conventional development steps. CBSD clearly signals a paradigm shift. Some of the characteristics of a component based software process model which make it different from a traditional process model are presented by Kaur et al. (2007).

Sommerville (2005) identifies the intermediate stages in a reuse-oriented process:

- **Component Analysis:** After the requirement specification, component repository is searched to identify the matching components
- **Requirement Modification:** In case matching components are not available, requirements may have to be modified. If not possible, a re-search is done to find the alternate solutions
- **System Design with Reuse:** New system is designed keeping in mind the reusable components
- **Development and Integration:** If appropriate components are not available, they are developed from scratch. New and old components are integrated to build the final solution

**COMPONENT BASED SOFTWARE DEVELOPMENT PROCESS MODELS THE STATE OF ART**

Component based software development is a twin process activity. The two processes are used at different stages of the life cycle of an application. First process is the component development process. It caters to the development of reusable software components, which are to be used in building an application using the second category process. Second process is the component based software development process and it is related to the development of a software application using already existing reusable software components. Both the processes are independent of each other with respect to time-frame and organizational homes. Individual components may be developed at different time and at different places. They may be later composed in to a software system at a different time and at a different place. The first process i.e., the software component development process resembles a traditional software development process. In addition to that it requires a horizontal analysis of the relationships and constraints that the component may have with the external environment so as to produce a product which can be used in multiple applications. Here the focus is to find an adequate trade-off between different factors, such as generality and complexity, in order to produce a component that is effectively reusable. A more general term to refer to this process in CBSD terminology is domain analysis. The output of domain analysis is not a single component but a set of components that can be used, in combination, to support functionalities of several applications in a specific domain.

This study discusses the process models only for the second type of CBSD activity i.e., software development using reusable software components. Here we discuss the component based development process models proposed in the research literature. The focus is on pure component oriented development. Evolutionary models such as RUP with object oriented development approach are not discussed here.
Somerville's Model

CBSD process model suggested by Sommerville follows sequential approach and is shown in Fig. 1 (Sommerville, 2005). This process model has 6 phases:

- **Outline System Requirements**: The user requirements are outlined briefly (rather than developed in detail, as specific requirements limit the number of components that might be used)
- **Identify Components**: A complete outlined set of requirements are used to identify as many components as possible for reuse
- **Negotiate Requirements**: Requirements are refined and modified so that they can comply with components
- **Architectural Design**: Architectural design is developed
- **Requirements can be negotiated to incorporate architecture compatible components. After system architecture is designed, steps 2 and 3 may be repeated**
- **Integration**: Finally the chosen components are integrated to get the software system

Kotanya et al. (2003) Model

Kotanya et al. (2003) has suggested a component based development process, which takes into account activities related to component use as well as component management. The development process phases are given as below:

- **Define System Requirements**: The phase includes activities such as requirements elicitation, requirements scoping, requirements modeling and specification
- **Design System Architecture**: This is the process of partitioning system requirements into logical components or subsystems
- **Compose System**: This is the process of replacing abstract design components with concrete off-the-shelf components. The composition process must devise mechanisms for integrating the different components without compromising the system quality (i.e., ensuring that the components conform to the adopted component model) (Fig. 2)
- **Verification**: The verification process corresponds to the whole development process. Focus of verification changes in each phase. At the requirements stage, verification is used to establish the availability of software components and viability of a component-based solution. At the design stage verification is concerned with ensuring that the design matches the system context (i.e., in terms of non-functional requirements,

![Diagram of Sommerville's Model](image-url)

**Fig. 1: The CBSD process model proposed by Sommerville (2005)**
architectural concerns and business concerns). This may require detailed blackbox testing of the software components and architectural analysis of the design. At the composition stage verification translates to design validation, through component assembly testing and system testing.

- **Management**: This phase looks after activities such as managing assets, system configurations, managing change and assuring quality. Asset management includes acquisition of components, their usage and evolution. Components (especially COTS) in a component based system evolve at different times with different objectives giving multiple versions of the same component in the same application. So a framework for tracking and controlling the versions of black box software components and custom software installed at all locations for the system is required. Management phase is also responsible for market research. New software as well as hardware products are introduced in the market every day. A framework guides for determining the timescale and kind of changes to be made to the commercial components that are already in a component based system.

**The Y-Model**

In this model, basic activities of a software development process such as System Analysis, Design, Implementation, Testing, Deployment and Maintenance are available as such. Some additional activities are added to support the component based software development (Fig. 3). New phases introduced in this process are as follows:

- **Domain Engineering**: Domain engineering is concerned with identification of potentially reusable components for a particular domain. For example in the banking domain, the components such as CreateAccount, CheckBalance, DepositAmount and WithdrawAmount can be reused across multiple banking applications. Domain engineering is a process of analyzing an application domain in order to identify areas of commonality and ways to describe it using a uniform vocabulary.
Fig. 3: The Y-model (Caprtz, 2005)

- **Frameworking:** A framework is a skeleton or a template used for producing software in an application domain. Frameworks capture the semantic relationships between the components of a particular domain. The main purpose of frameworking phase here is to reuse the software components already developed and then classify them further to form new frameworks.

- **Assembling:** In this phase, the software application is composed from the existing reusable software components.

- **Archiving:** The components developed for a particular software application are archived for future use in other related applications. This involves activities such as component cataloguing and storage.

**The V-Model**

The V-model for traditional software development is adapted by (Crnkovic et al. (2005) for building a system from pre-existing components, the main theme of component-based approach. The activities in different phases of the development process are shown in Fig. 4 and described below:

- **Requirements Analysis and Specifications:** Requirements are analyzed and specified keeping in mind the available components. If possible, requirements can be negotiated as well in order to make use of the available components.

- **System and Software Design:** In this phase again, component pool is available. Components selected in the previous phase may have to be rejected if they do not fit into the overall design of the software. Alternatives can be selected from the component pool.

- **Implementation and Unit Testing:** An ideal case to build an application is the direct integration or connection of component interfaces. But in practice, glue-code/component wrappers need to be written to bridge component interface mismatches. Sometimes new functions have to be written to fill the gaps in system requirements and component capabilities. These modules are then tested separately.
Fig. 4: V-model adapted for component based software development

- **System Integration**: The integration process includes integration of standard infrastructure components such as database management and communication software components that build a component framework and the application components
- **System Verification and Validation**: Standard tests and verification techniques are used here. Error identification is more difficult in case of black box components, which are procured from various vendors
- **Operation Support and Maintenance**: In the operational system, new components may have to be added or existing components have to be modified/removed in order to support the changing requirements. The support and maintenance process includes deployment of new or modified components in the system, change in glue-code, or replacement of troubling components

**Qureshi and Hussain (2008) Model**

This model proposed recently is largely influenced by object oriented software development life cycle. Main phases of this process model are given below:

- **Communication**: Every new project starts from communication with the customer
- **Planning**: Project feasibility is evaluated in the planning phase
- **Analysis and Component Selection**: If feasibility report is OK then analysis of the detailed requirements of the project starts. Alongside requirement analysis, suitable components are selected
- **Engineering and Testing**: In the engineering and testing phase, selected components are adapted, if required. Development of new components also takes place in the same phase, in case existing components could not fulfill all the requirements of the software application. Integration and system testing of the newly developed and of the reused components are performed
- **Evaluation**: Developed system is submitted to the customer for feedback
Fig. 5: Component based development process Qureshi and Hussain (2008)

Existence of a component repository is essential in this life cycle model. Component repository supports the application developer during the analysis and selection phase (Fig. 5). But there is no explicit mention of how and when components are added to the repository.

**Top Down v/s Bottom Up approach of Component Based Software Development**

An important question to ask is as to what type of role software architecture plays in building an application-active or passive. Should the software architecture be specified first and then components selected accordingly that fit into the specified architecture? Or the components, which match the application requirements, should be chosen first and then architecture of the application decided. A brief overview of the definitions of software architecture and component architecture give an idea of level of interconnection required for smooth composition of software components:

- **Software Architecture:** The software architecture defines the static organization of software into subsystems interconnected through interfaces and defines at a significant level how nodes executing those software subsystems interact with each other.

- **Component Architecture:** Component system architecture consists of a set of platform decisions; a set of component frameworks and an interoperation design for the component frameworks (Crnkovic and Larsson, 2002)

Ideally every component should conform to a component model for easy interoperability with other components. NET, JAVA and CORBA are some of the popular component models. It is possible to use components implemented in different component technologies, but in practice it is very difficult to achieve interoperability between different component models. A component model requires a particular architectural framework and the application is supposed to use this framework. This has a direct impact on architectural decisions. For example, if the component model requires a client-server architecture style, it is obvious that the application will use that style and not another one for example pipe-filter. This will put limitations on the system design.

So for a successful and cost-effective component integration, either components have to follow the interaction mechanisms imposed by the software architecture of the application, or the software architecture of the application should adapt to the interaction mechanism or platform restrictions embedded in the software components.

We discuss below both the strategies of software development (Top/Down and Bottom/Up).
Architecture Driven (Top/Down) Software Development

This approach is similar to the traditional way of software development. First a high level design and specifications of needed software components are created. Once the software architecture has been defined, the components which are to constitute the system must be developed or selected. Components typically need to be adapted before integrating them into the system. This top-down approach ensures the fulfillment of requirements to some extent. However, this approach does not encourage the reuse of preexisting components, since there is a high degree of probability that only a few or no pre-existing components exactly fit into the system (Brownsworth et al., 1998).

Component Driven (Bottom/Up) Software Development

The architecture driven component based development process follows the traditional approach i.e., first analyze the requirements and then identify the components that match the requirements. But such a process is essentially a failure for component based software. Because it ignores the existence of reusable components and tries to find a component that matches the newly identified requirements and the chosen architecture. What if the required components do not exist?

In component driven software development, the system requirements and architectural design are already influenced by available software components. The system design is based on available components and compared to architecture driven development, results in a higher degree of reuse. Sametinger (1997) opines that development based on components should preferably be done in a bottom up manner rather than top to down.

The Y model supports both the strategies of software development-top/down as well as bottom/up. In this software process model, choice in top-down or bottom-up software development strategy depends upon the knowledge of the software engineer about the application domain. A novice developer may use bottom-up approach, but an experienced one may prefer to work on high level abstractions. This knowledge naturally determines the prevailing strategy to software development.

Availability of reusable software components also influences the choice of software development strategy. Different development process is required depending on the degree to which the components are pre-existing (Crnkovic and Larsson, 2002).

A strictly top-down or bottom-up strategy to software production is not quite appropriate. A mix of bottom-up and top-down approaches begins with the system requirements and the analysis of possible candidate components. Component specification and selection have impact on final requirements and the system architecture. There is a critical relationship between component selection, requirement specification and architecture definition. Pragmatically, three essential elements (requirements, architecture and component selection) must be worked in parallel with constant trade-offs among them (Brownsworth et al., 1998). But this juggling may result in cyclic requirement-component dependency problem experienced by Tran et al. (1998). The problem can be described as follows:

- Creation or modification of a system requirement forces reexamination of COTS products that have been previously selected
- Selection of a new COTS product forces reexamination of all associated system requirements

They have also outlined some risk mitigation approaches to minimize the impact of this problem.
The Development Process and Component Activities

Component-based software development approach consists of activities, which do not exist in the traditional software development approach. The development processes described in section 3 are compared on the basis of component-related activities supported for component-based development. As shown in Table 1, the parameters for comparison are the specific activities needed for software development using component-based approach. These parameters are listed in column 1. Other columns of the table show the information about the process models in this order: Sommerville’s model (2005), Kotonya et al.’s model (Kotonya et al., 2003), V-model (Crnkovic et al., 2005), Y-model (Capertz, 2005) and Qureshi et al.’s Model (Qureshi and Hussain, 2008).

It could be observed in Table 1 that all models support all the activities except 2, namely domain analysis and Component Evolution. But for systematic reuse these two activities are very important. In order to actually gain positive economic benefits, availability of a set of suitable reusable components is essential. To this end, domain analysis provides a pool of reusable components that can address the requirements of various applications in that particular domain. Domain analysis helps in preparing for application development using already available reusable components, whereas component evolution helps in application maintenance. Only Kotonya’s model looks into this issue related to new versions of software components available in the market. Significance of this issue can be judged from the fact that vendors upgrade their components every 10 months on average and component releases go unsupported after an average of 3 releases (Yang et al., 2005). Market analysis helps to study the component changes required as a result of continual component updates. So the application maintainers have to be in look out for new versions of the components to maintain the application successfully.

WHAT THE STANDARDS SAY?

IEEE 1517 was published in 1999 as a supplement to IEEE/EIA 12207 (IEEE, 1999). It is intended as a specification of the minimum requirements that a software life cycle model must meet to enable and support proper reuse. IEEE 1517 organizes the required life-cycle processes in four categories: cross project, primary (software development process), secondary (such as configuration management process, quality assurance process and project management process) and Organizational. The last 3 processes are adapted from ISO 12207 and the cross project category is added to accommodate software reuse. Domain engineering is the only process categorized as a cross project process, where domain is defined by IEEE 1517 as a problem space. Domain engineering covers the life cycle of a product and provides the assets that can be used by the primary process. So the process models that follow the bottom-up approach of software development conform
to the standards specified for component based software development process. V-model and Y-model both conform to the standards defined for a reuse based development process model.

THE STATE OF PRACTICE

Here, we present the state of practice. Initially we discuss some of the studies/experiences reported in the research literature. In the end of the section, we present findings of an investigation that we carried out to know the present practices followed in the software industry.

Li et al. (2009) did an empirical study on the adoption of the component based development process by the software industry boasting of reuse based development. The study analyzed 133 software development projects from 127 companies. It is reported that there is a discrepancy between academic theory and industrial practices regarding the use of components. Software companies have not adopted any reused based process but adapted their traditional processes to incorporate component integration related activities. In around 75% of the studied projects, development process was chosen in advance without reuse in mind. Formal methods for component selection, evaluation etc., proposed by researchers were not followed. Reasons given are that formal methods are more time consuming; only a few components are available; developers use (select) only those components which they themselves have already experienced with; company have a long-term partnership with a specific component provider and lastly components do not constitute important parts of the system. Mainly waterfall, waterfall with prototyping and incremental mixed with prototyping processes are used. Prototyping is used to negotiate a match between identified requirements and available components.

In a pre study of this study (Li et al., 2006) focused on Norwegian software companies, 16 reuse based projects were studied. It was found that in all 12 successful projects out of the 16, developers were making use of components that they were already familiar with and components were selected in the initial phases. The remaining unsuccessful projects were using top/down approach of software development i.e. components were selected in the later phases.

Another experience is reported by Morisio et al. (2000). The Flight Dynamics Division (FDD) at NASA/Goddard Space Flight Center, is responsible for the development of large amounts of ground support software for NASA satellites. In the 1980s and 90s, the FDD achieved effective software reuse levels as high as 90%. This trend has gradually evolved into COTS based software development. A study carried out by Software engineering Institute, Carnegie Melon University, revealed that the process followed for the projects at NASA is not Component based. Requirements are finalized first and then suitable components are searched for. No formal component selection method is being used.

Lastly we discuss the findings of a small study we carried out to know the reuse based development processes followed in the software industry. We randomly selected over 30 medium to large size software companies, both of national and international level. In order to collect the data, regarding the software development processes followed by the software companies, we studied the information published on their web sites. On the website, following types of hyperlinks pointed to the desired information:

- Brochure
- Services>new product development
- How do we work?
A software company offers many services ranging from new product development to product support or 3rd party integration. But we restricted our search to new product development only. However, we could not find the process related information from the website of every company, we visited. Only some of the companies furnish this information explicitly.

Table 2 summarizes the data collected from the web sites of software companies. It is not surprising to find that waterfall model is still followed in the software industry. Waterfall model with prototyping for requirement analysis, spiral and classical V-model are used for traditional software development. To meet the shorter time-to-market constraints, RAD and agile methodologies are also preferred. Many software companies claim to follow reuse based approach. But it is not mentioned as to which reuse based development process, they use or have adapted to. We could not get any references to the process models discussed in this study except one. Only one out of the 30 companies studied, mentions V-model as the development process followed for reuse based development.

Service Oriented Architecture or web services are the next catchy words in the software industry after software components. Number of companies using SOA/web services is the maximum in this study.

DISCUSSION

For a reuse based solution, traditional development processes are not suitable. Several activities, such as component search, evaluation, selection, adaptation, which are important for a reuse based solution do not find any mention in the key process areas of the traditional processes. Research community has put in efforts to look for tools and techniques to realize reuse based solutions. But these solutions are not as such adopted by the practitioners. As discussed, a number of component based software development process models have been proposed. Many of them also comply with the prescribed standards of a reuse based process. Component selection is another important activity in a reuse based approach. Many component selection methods have been suggested in the research literature (Land et al., 2008). But studies of the industrial practices reveal that a component selection method is seldom used. The reason behind this academic-industry gap can be attributed to less number of empirical studies on the proposed methods. Perhaps application developers are not convinced with researchers’ proposals published in conference proceedings and journals. Practitioners find the proposed processes as too lengthy and heavyweight to deliver software in shorter time periods. In their opinion, more effort and time is required to develop software using these formal processes.

While looking at information on the websites of software companies, it could be realized that Service Oriented Architecture (SOA) is the buzzword now a days. Service-Oriented Software Engineering (SOSE) is a software engineering paradigm that aims to support the
development of rapid, low-cost and easy composition of distributed applications even in heterogeneous environments (Papazoglou et al., 2007). It utilizes services as fundamental elements for developing applications and solutions. SOA, a key element in SOSE, enables a collection of services to communicate with each other. An emerging technology leveraging use of SOA is Software as a Service (SaaS) in which the software users do not purchase a software package but pay for its use through subscription (Olsen, 2006). As Brown et al. (2003) discuss, both SOA and CBSE are complementary to each other. Rather than using these two technologies in isolation, they should be integrated together to develop software systems which are more predictable, dependable, dynamic and flexible (Brown et al., 2003; Zhu, 2005). Collet et al. (2007) propose that CBSD and SOA should be integrated to enable some form of component based engineering of services.

CONCLUSION

Software process lays the foundation for a good quality product. A reuse based approach to software development requires a software process different from the traditional process. The reason behind this is that traditional processes cannot accommodate the risks that are associated with the use of reusable components. Several reuse based process models have been proposed in the literature and some processes conform to the laid down international standards as well. But as far as the practice goes, a few projects, implementing reusable components, actually use the proposed reuse based development processes. Most of the projects are still using traditional processes by just adding to them some additional activities related to component identification and component selection etc. To start with Component quality is not given any consideration. Under the prevalent circumstances, it is further required to design a series of experiments, in which relative costs and benefits of choosing a component based process can be weighed against the choice of a traditional software development process. Taking up such studies will influence the developers decision regarding choice of the suitable development process.

REFERENCES


Sommerville, I., 2005. Software Engineering. 7th Edn., Addison-Wesley, USA.


