Available online at www.sciencedirect.com
ScienceDirect
Advances in Engineering Software 39 (2008) 88–94
ADVANCES IN ENGINEERING SOFTWARE
www.elsevier.com/locate/advengsoft

A reusable software component-based development process model

M.R.J. Qureshi, S.A. Hussain *

Department of Computer Science, COMSATS Institute of Information Technology, Lahore, Pakistan

Received 11 July 2006; received in revised form 15 January 2007; accepted 15 January 2007
Available online 12 March 2007

Abstract

The concept of component-based development (CBD) is widely used in software (SW) development. CBD facilitates reuse of the existing components with the new ones. The well known architectures of CBD are ActiveX, common object request broker architecture (CORBA), remote method invocation (RMI) and simple object access protocol (SOAP). The objective of this paper is to support practice of CBD by comparing it with traditional software development. This paper also evaluates object oriented (OO) process model and proposes a novel process model for CBD. The importance of repository has also been discussed.

Keywords: CBD; CORBA; ActiveX; RMI; SOAP; Repository

1. Introduction

Reuse of SW components concept has been taken from manufacturing industry and civil engineering field. Manufacturing of vehicles from parts and construction of buildings from bricks are the examples. Spare parts of a product should be available in markets to make it successful. The best example in this case is:

- The manufacturers of Honda, Toyota and Suzuki cars would have not been so successful if these companies have not provided spare parts of their cars?

Software companies have used the same concept to develop software in parts. These companies have provided parts with their softwares to market themselves successful. Software parts are shipped with the libraries available with SW.

These SW parts are called components. Different people have defined component in different ways. A binary code that can be reused is called a component. The component concept is similar to the object concept of Object Oriented (OO) Programming. A component is an independent part of the system having complete functionalities. A component is designed to serve a particular purpose, i.e., for example command button and text box of VB. Component is like a pattern that forces developers to use the predefined procedures. It meets the specifications to plug it into new SW components.

Microsoft Corporation and *Sun Microsystems* are two major software-providing organizations. Their tools are widely used in SW markets applications development. These two companies and other vendors are also providing components with their tools to make them successful.

Section 2 describes related work in CBD. Section 3 describes comparison of process models. Section 4 is covering the motivation for CBD process model. Section 5 proposes a process model for CBD. Section 6 describes the role and importance of repository in CBD. Section 7 describes how CBD is better than the traditional customized software development.

2. Related work

According to Lycett and Giaglis [1], it is extremely difficult to evaluate information systems in terms of reuse.

* Corresponding author. Tel.: +92 300 4791679; (M) 03334492203.
E-mail addresses: rjamil@ciitlahore.edu.pk (M.R.J. Qureshi), asadhussain@ciitlahore.edu.pk (S.A. Hussain).

0965-9978/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.advengsoft.2007.01.021

They are of the view that all development approaches have a danger and risk to reuse the existing components. According to them, the main reuse risks can be avoided by following ways:

- Integrate business driven evaluation at an early stage during selection of components. It will reduce effort of evaluation and selection.
- Evaluation and selection should be an on going process.

The authors discussed Discount Cash Flows (DCF), Net Present Value (NPV), Return on Investment (ROI), Internal Rate of Return (IRR), SESAME and cost benefit analysis sheet (CBA) but without providing any facts and figures. They did not provide comprehensive evaluation criteria to evaluate and integrate the previously developed components in order to be reused. They proposed a content, context and process (CCP) analysis in order to evaluate a component to reuse, which is not practical in terms of implementation. CCP analysis is not practical because it is very subjective for selection and evaluation of a component for reuse. Constructive Cost Model II proposed by Boehm et al. [2] is far better to calculate cost of new systems to be developed by new and reusable components.

According to Merijn de Jonge [3], one of the reuse requirements is to develop independent components and those must be integratable. He reported another requirement that best reuse practice and processes need to be achieved and followed. He proposed techniques to integrate reusable independent components within one system and across systems. He proposed a source tree composition technique to integrate:

- core modules of components;
- package development that permits development of more than one component concurrently according to the scope of different software systems.

Package based software development is a popular research area. The author concentrates on source tree composition technique for software configuration management of more than one system. This is a problem in terms of management of reusable components in repository. The author did not discuss crosscutting development of components and their integration in development of multiple systems.

The extent of software reuse depends upon the reuse strategy followed [4]. They proposed a set of six dimensions to support the reuse practices after conducting a survey during which data was gathered from 71 software development groups. These dimensions are planning and improvement, formalized process, management support, project similarity, object technology and common architecture. On the basis of dimensions, they found out five reuse strategies that are practiced in software development groups. The five strategies are ad hoc reuse with high reuse potential, uncoordinated reuse attempt with low reuse potential, uncoordinated reuse attempt with high reuse potential, systematic reuse with low management support and systematic reuse with high management support. The main objective of their research is to classify reuse strategies so that development groups can get benefit and achieve success to complete software projects. The authors support the last strategy, i.e., systematic reuse with high management support, but in this scheme there is a need of more detailed analysis by gathering data from various software houses working in different geographical locations, before reaching a conclusion.

Selection of reusable components is important to improve productivity of component-based software [5]. They proposed a component repository for facilitating enterprise java beans (EJB) component reuse (CRECOR) to store and manage the reusable components. They mentioned that working on the reusable components with their repository software could have many benefits. For example:

- Specification viewing.
- Adapting.
- Testing.
- Deploying.

The repository proposed by the authors does not have a version control and change control functions to manage different versions of reusable components. Version and change control are important functions of a repository to manage and update different versions of a component.

According to Haddad [6], the SW organizations have to invest huge sums of money to start successful reuse methodology and it's a barrier for them. In his opinion, the core of reuse is source code. According to an estimate mentioned by the author, "domain specific components represent up to 65% of the application size. One approach to effective reuse practices focuses on domain specific components". He proposed an integrated approach for component-based development to support domain specific components. An integrated approach is a collection of reusable components in a development environment.

The author also discusses the concept of interface to describe a wrapper interface mechanism. The wrapper interface mechanism will be used to manage and control the interface between or among the integrated collection of reusable components. The objective of his research is to develop benchmarks for software organizations so that they begin reuse practices by emphasizing mainly on programming effort and not on management and operational perspectives. The main focus of author's research is development of domain specific reusable components but not on the construction of reusable components of different domains of concern. This problem can be handled through software engineering for adaptive and self-managing systems.

The research by Arndt and Dibbern [7] reveals that, there are two traditional approaches to construct software

systems, i.e., customization and use of standard components libraries. They proposed a composition of customized and component libraries domains approaches to achieve benefits of both. There are many opposing factors to resist practice of this new domain. The authors to explain the change process to support composition approach have adopted a logical solution. They also introduce a concept of mindful innovation to show that how modular development can avail the benefits that domain change provides.

The advantages of component-based software development (CBSD) domain are also discussed that are already described in many papers. It is a theoretical concept. However it has been written to support the practice of CBSD domain. CBSD acceptance process is not considered by the authors.

3. Comparison of process models

Waterfall process model is the traditional and classical model used for software development. Communication, Planning, Modeling, Construction and Deployment are the main phases of Waterfall process model. This model is iterative in terms of system development life cycle (SDLC) phases. It is only applicable when comprehensive requirements are available regarding the re-engineering of existing SW. It is not feasible for commercial projects because of iteration of phases. This model requires more time and cost to develop and implement the system. Another obstacle, is that the working version is available during the implementation phase. It can result in total disaster if customers' requirements are vague. It is not practically possible to collect complete requirements initially at the start of the project that is the fundamental requirement of this model [8].

Prototype process model is proposed to integrate with other methodologies such as waterfall to make them more effective. Listen to Customer, Develop a Demo/Rebuild and Evaluate are the main phases of Prototype process model. It is cyclic in nature. Iteration continues until the project is complete. Fast development results in dramatic time and cost saving. No proper documentation is made for current and future projects. Fast development often results in a poor quality SW. Prototype process model is never used as a standalone model to develop systems. It relies heavily on end user in terms of functionalities and interfaces. This can result in poor efficiency. End users some times may not want to make SW project successful due to the threat of job loss. They may eliminate some key features or functions of the SW [9].

Rapid application development (RAD) process model is one of models that are suitable for real life small-scale commercial projects. Requirements and planning, user design, construction and cutover are the main phases of RAD model. It results in quicker development and cost saving. The core benefit of RAD model is more complete user requirements. This is because of the using prototype approach during the user design and construction phases. RAD model focuses on the main features of the SW from the user's point of view. It covers all phases of the waterfall model but at a rapid rate. All the drawbacks of the prototype approach are incorporated into the RAD model. RAD model is not suitable for safety critical projects where life risks are involved.

Evolutionary or incremental process model supports a SW engineer to develop more complete comprehensive SW. Communication, planning, analysis, engineering, testing and customer evaluation are the main phases of Evolutionary process model. It is suitable for modular, incremental, web development and less number of developers. Software development organization can hire more staff when core functions are accepted.

Rational unified process (RUP) model incorporates almost all characteristics of evolutionary model but the focus is on pure object oriented development. Its main phases are inception, elaboration, construction and transition. Construction phase supports reusability of existing classes. The main limitations of RUP model are high time and cost and emphasis on documentation rather than on development [10]. According to Highsmith and Cockburn [11], all the agile process models emphasis on the need of quality design. Extreme programming (XP) is the most widely used model among all agile models, such as dynamic system development methodology (DSDM), scrum, crystal and feature-driven development (FDD). XP model main phases are planning, design, coding and testing. Agile models focus on delivering the first increment in couple of weeks and complete SW in couple of months. The main limitations and benefits of all models are fast development and cost saving. Fast development leads to poor quality SW and incorporates all disadvantages of RAD process model. A comparison of waterfall, prototype, RAD, evolutionary, OO, RUP, agile and proposed CBD process models is shown in Table 1.

4. Motivation for CBD process model

Bailey and Basili [12], have proposed software-cycle model for reuse and re-engineering. The model suggests five stages to reuse a component:

- Analysis of existing programs to sort components to be reused.
- Re-engineer to eliminate domain specific troubles.
- Save reusable components in the repository.
- Construct independent status components with a reusing approach and store in a repository.
- Reuse components to develop new programs.

The model proposed by authors is not a complete process model for CBD but core focus of their paper is on reuse activities.

As mentioned by Jain et al. [13], poorly gathered SW requirements could fail a SW project. Authors reported

Table 1
Comparison of waterfall, prototype, rapid application development (RAD), evolutionary, object oriented (OO), rational unified process (RUP), agile and proposed CBD process model

	Reuse of classes	Reuse of components	Architectural support for interoperability	Upgradability	Complexity	Time required	Reliability	Quality
Waterfall [10,21]	Not achievable	Not possible	No intercommunication	Not possible	High complexity	Larger duration	Low reliability	Poor quality
Prototype [10,21]	Not possible	No artifact	No provision	No feature	Highly complex	Never used as standalone model	Poor reliability	No quality support
RAD [10,21]	Not attainable	Not likely	No support	Not attainable	High complexity	Fast development	Weak reliability	Low quality
Evolutionary [10,21]	Not attainable	Not achievable	No intercommunication	Possible	High complexity	More time require	Reduced reliability	Sufficient quality
OO [10,21]	Achievable	No feature	Allowed	Adequate	Average complexity	Long period require	Adequate reliability	Adequate quality
RUP [10,21]	Attainable	Not included	High provision	Proportionate	Average complexity	Massive amount of time	Medium reliability	Satisfactory quality
Agile [11]	Feature provided	No attribute	Permitted	Allowed	Average complexity	Rapid development	Poor reliability	Weak quality
Proposed CBD [19,20]	High support	Core functionality	High support	High provision	Low complexity	Faster than RAD and agile	High reliability	High quality

that it is because of natural drawback in requirements determination methods. They proposed an approach to construct SW by categorizing the components in a knowledge base. Existing components are recognized, chosen and integrated in a newly developed SW by using the knowledge base. Different classification schemes to reuse artifacts have been discussed as well. These are enumerated, keyword, faceted and hypertext. The objective of this paper is to ease requirements gathering using knowledge base but the paper is lacks in suggesting a comprehensive process model for CBD.

Eduardo Santana de Almeida et al. [14], have presented an incremental method for distributed CBD. It is based on two phases. The first phase is composed of requirements of the problem domain and construct employable components in object oriented (OO) language. These employable components are stored in a repository. SW Engineers look up MVCASE tool to select the necessary components to develop the SW system in the second phase.

The authors do not propose a clear-cut process model and a use of a specific CASE tool is the requirement of this model.

Luiz Fernando Capretz et al. [15], have anticipated a software life cycle that supports CBD using OO construction. The main phases of model are Domain Engineering, System Analysis, Design and Implementation. The major problem of this model is the selection of reusable components during the design phase. The selection of reusable components should be during the analysis phase. Therefore analyst can estimate the cost, schedule and effort required to develop and integrate the components.

Hutchinson et al. [16], have discussed a four-stage component-based development process mode. It is very complex in nature to implement. The core objective of this paper is to integrate off-the-shelf components with the newly developed components rather than in house development. Repository has not been used here.

In Ning [17] CBD process model main phases are Component Analysis, Architectural Design, Component Brokerage, Component Production and Component Integration. The author modified the Waterfall process model with the integration of above-mentioned phases for the CBD process model. The Waterfall model is not suitable for commercial applications because of the verification of phases repetition. It is time and cost consuming process model suitable only for research projects. A comparison of component-based development process models is shown in Table 2.

5. Process model of CBD

The object oriented process model is the only process model that indicates the reuse of existing SW parts. That model can be modified to implement reuse of component-based development. The engineering, construction and testing phase reflect the reuse of existing classes. The main phases of CBD process model are shown, in Fig. 1.

Table 2
Comparison of component-based development process models

Other CBD process models	Drawbacks
The software-cycle model for re-engineering and reuse [12]	The model proposed by authors is not a complete process model for CBD but core focus is on reuse activities
An assessment model for requirements identification in component-based software development [13]	The objective of this model is to ease requirements gathering using knowledge base. The model lacks in suggesting a comprehensive process model for CBD
Distributed component-based software development: An incremental approach [14]	A clear-cut process model is not proposed and use of a specific CASE tool is the requirement of this process model
Component-based software development [15]	The major problem of this model is the selection of reusable components during design phase. The selection of reusable components should be during the analysis phase so that an analyst can estimate the cost, schedule and effort required to develop and integrate components
A service model for component-based development [16]	The core objective of this paper is to integrate off-the-shelf components with the newly developed components rather than in-house development. Repository has not been used here
A component-based software development model [17]	CBD process model main phases are component analysis architectural design, brokerage, production and integration. The authors modified the waterfall process model with the integration of above-mentioned phases for the CBD process model. The waterfall model is not suitable for commercial applications because of the repetition of phases. It is a time consuming and costly model and is suitable only for research projects

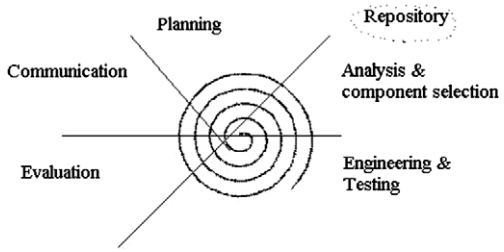

Fig. 1. CBD process model.

- Communication.
- Planning.
- Analysis and component selection.
- Engineering and testing.
- Evaluation.

The customer is *communicated* at the start of the project to gather the basic requirements. Initial use cases are developed at this stage. Project specification or proposal document is prepared during the *planning* phase. Project specification or proposal document is composed of feasibility and risk assessments that are made to prepare a cost benefits analysis (CBA) sheet. CBA sheet helps to estimate that whether SW project is feasible for the customer or not. *Analysis* phase is only started if the customer approves the proposal. Analysis phase of the OO process model can be modified according to the CBD process model. The new name of analysis phase can be '*analysis and component selection*' phase. This is the phase where an analyst gathers detailed requirements and tries to identify and select those components that can be reused from the components repository. The relationships among the components are identified. The properties and behaviors of the components are identified as well. Core objective of this phase is to reuse maximum components, rather than reinventing the wheel. It will also improve productivity and efficiency of software engineers. *Engineering and testing* phase of the OO model matches the requirements of CBD process model. The selected components are modified according to the requirements of new system to be developed and tested. The new components are designed, developed and tested on unit basis.

Integration and system tests of the newly developed and of the reused components are performed. Interface Definition Language (IDL) is coded to integrate components if programmer is using CORBA or RMI architecture. *Evaluation* phase of OO process model fits in the requirements of the CBD process model. Customer is requested to evaluate and verify SW that whether it meets his/her requirements or not.

6. Roles and importance of repository in CBD

As mentioned in the Section 5, selection of reusable components is important to improve the productivity of component-based software. The repository is used to store and manage the reusable components. The main benefits achieved, while working on the reusable components having a repository, are as follows:

- Classification.
- Searching.
- Modification.
- Testing.
- Implementation.
- Version control.
- Change control.
- Up to date and consistent documentation.

There could be one or more than one repository to select and retrieve components [18]. The main benefit of more than one repository is that repositories can be developed and categorized on the basis of specific component domains.

7. How CBD is better than traditional SW development

The major advantages of CBD are:

1. Reusability.
2. Interoperability.
3. Upgradability.
4. Less complexity.
5. Time effective.
6. Cost effective.
7. Efficient.
8. Reliability.
9. Improved quality.

Reusability is an important advantage of developing SW applications using CBD. Developing a mail transfer web application system in ASP can be an example application using the CBD. Collaboration Data Objects for Windows NT Server (CDONTS) component is provided by Microsoft Corporation to develop e-mail systems. This component can be used in similar applications. The built-in components within a particular tool also reflect the benefits of reusability approach such as Microsoft VB 6.0 text box and command button objects. The programmer needs intensive coding to develop a text box with its complete functionality using C language. This text box with its complete functionality in VB 6.0 is a component and can be used frequently without writing a single line of code. That makes VB 6.0 the most widely used Rapid Application Development (RAD) tool. Reusability thus helps us to concentrate on adding more complex functionalities in our applications rather than focusing on developing basic components.

CBD architectures allow components to intercommunicate with each other and this makes them interoperable. Interoperability facilitates programmers to integrate one application with other applications. The information systems used by ATM machines of different banks are integrated with each other. One link system is the benefit of interoperability provided by CBD.

Upgradability of web applications is easy if some new component has been introduced during the life of an application. Programmers just have to replace the previous component with the new component without a change in the front-end code of the application. The client–server architecture is used for development of distributed applications. There are three types of client–server architectures. Single tier architecture integrates presentation, application logic and data layers into single SW application such as MS Excel and Access SW. Two-tier architecture integrates presentation and an application logic layer into front-end SW application and data layer is handled by back-end SW application. This can be explained by taking an example of a business application that is coded by using VB or ASP to handle the presentation and application logic layers where as data layer is handled by MS Access or SQL server database. The three tier architecture is also called multiple or *n*-tier architecture. The presentation, application logic and data layers are handled by three SW applications. The front-end tool such as VB or ASP manages the presentation layer; the COM or COM + component manages business logic layer while MS Access or SQL server database manages data layer.

Presently programmers prefer to use three-tier architecture to develop business applications. The business logics are continuously changing due to current business environment. Therefore, it is very easy for programmer to replace the COM component without changing front-end VB or ASP code and back-end MS Access or SQL server database.

An advantage of CBD is that programmers do not need to understand how a component is working. They should be concerned only with the integration of that component with the application. This is similar to a car driver who does not need to know how a car engine operates. The programmer can concentrate more on main features of system rather than wasting time on designing the basic components of the application such as text box and command button. CBD saves programmers from complex programming. It helps to reuse components frequently in the similar applications that result in time and cost savings. It also results in efficient design process.

The reusable components are already thoroughly tested and maintained when these are used for the first time. Therefore the reusable components are reliable in terms of usage as compared to the newly developed components. Quality of the application developed from reusable components also significantly improved. The reusable components are well tested and maintained. Newly developed components need a lot of testing and maintenance. There would still be a that some of the bugs are left unreported. Quality of SW is effected because of such problems.

7.1. Examples

As indicated by Lim [19], reuse projects have achieved gross savings from $4.1million to $5.6 million and return on investment (savings/cost) from 216% to 410%. He has also reported that the defect rate for reuse code is 0.9 defects per kilo line of code (KLOC) where as it is 4.1 defects per KLOC for new code. He made a comparison between two applications, one was developed with reuse strategy and other was developed without it. The improvement in the development time was 51% for the application that was developed from the reusable components.

As stated by Henry and Faller [20], the improvement in quality of SW is 35% because of reusable components.

8. Conclusion and future work

This paper supports practice of CBD in contrast to traditional software development. A process model has also been proposed for the Component Based SW Engineering (CBSE). Role and importance of repository in CBD has also been discussed. From these discussions it is concluded

that CBD technology is more cost effective, time saving and productive for SW development. A working prototype will be developed to support CBD process model.

References

[1] Mark Lycett, George M. Giaglis. Component based information systems: Towards a framework for evaluation. In: Proc of 33rd annual international conference on system sciences, Hawaii, 4–7 January 2001.

[2] Donald J. Reifer, Barry W. Boehm, Sunita Chulani, The rosetta stone: Making COCOMO 81 Files Work With COCOMO II. Available from: http://sunset.usc.edu/publications/TECHRPTS/1998/usccse98-516/usccse98-516.pdf, 1998.

[3] Merijn de Jonge, Package-based software development. In: Proc of 29th EUROMICRO conference new waves in system architecture, 2003.

[4] Rothenberger Marcus A, Dooley Keven J, Kulkarni Uday R, Nada Nader. Strategies for software reuse: A principal component analysis of reuse practices. IEEE Trans Software Eng 2003;29(9):825–37.

[5] Jihyun Lee, Jinsam Kim, Gyu-Sang Shin. Facilitating reuse of software components using repository technology. In: Proc of 10th Asia–Pacific software engineering conf, 2003.

[6] Hisham M. Haddad, Integrated collections: Approach to software component reuse. In: Proc of 3rd international conference on information technology: New generations, 2006.

[7] Jens-Magnus Arndt, Jens Dibbern. The Tension between Integration and Fragmentation in a Component Based Software Development Ecosystem. In: Proc of 39th Hawaii international conference on system sciences, 2006.

[8] Laplante Phillip A, Neill Colin J. The demise of the waterfall model is imminent and other urban myths. Queue ACM Press 2004;1(10).

[9] James Purtilo, Aaron Larson, Jeff Clark, A methodology for prototyping-in-the-large. In: Proc of 13th international conference on software engineering, 1991.

[10] Andreas Schmietendorf, Evgeni Dimitrov, Reiner R. Dumke. Process models for the software development and performance engineering tasks. In: Proc of 3rd international workshop on software and performance WOSP '02, 2002.

[11] Highsmith J, Cockburn A. Agile software development: The business of innovation. IEEE Comput 2001;34(9):120–2.

[12] Bailey John W, Basili Victor R. The software-cycle model for reengineering and reuse. ACM Press 1991:267–328.

[13] Jain Hemant, Padmal Vitharana, Fatemah Mariam Zahedi. An assessment model for requirements identification in component-based software development. The DATA BASE for Adv Inform Syst 2003;34(4).

[14] Eduardo Santana de Almeida, Alexandre Alvaro, Daniel Lucredio, Antonio Francisco do Prado, Luis Carlos Trevelin, Distributed component-based software development: An incremental approach. In: Proc of 28th annual international computer software and applied conference, 2004.

[15] Luiz Fernando Capetz, Miriam AM Capretz, Dahai Li, Component-based software development. In: Proc of the 27th annual conference of the IEEE industrial electronics society, 2001.

[16] John Hutchinson,Gerald Kotonya, Ian Sommerville, Stephen Hall, A service model for component-based development. In: Proc of 30th EUROMICRO Conf, 2004.

[17] Ning Jim Q. A component-based software development model. IEEE Software 1996:389–94.

[18] Ying Pan, Lei Wang, Lu Zhang, Bing Xie, Fuqing Yang, Relevancy based semantic interoperation of reuse repositories. In: Proc of SigSoft' 04/FSE-12, 2004.

[19] Lim WC. Effects of reuse on quality, productivity and economics. IEEE Software 1999;5:23–30.

[20] Henry E, Faller B. Large scale industrial reuse to reduce cost and cycle time. IEEE Software 1995:47–53.

[21] Center for Technology in Government University at Albany/SUNY. A Survey of System Development Process Models, CTG.MFA-003, 1998.