International Journal of Computer Science and Engineering (IJCSE)
ISSN 2278-9960
Vol. 2, Issue 2, May 2013, 123-134

International Academy of Science, Engineering and Technology
Connecting Researchers; Nurturing Innovations

A TECHNIQUE FOR COMPONENT BASED SOFTWARE DEVELOPMENT

NAVNEET KAUR & ASHIMA SINGH

Computer Science and Engineering Department, Thapar University, Patiala, Punjab, India

ABSTRACT

The software development paradigm is moving from traditional software development , which is based on building software systems from scratch, to Component-Based Software Development (CBSD), which is based on using the reusable components as building blocks for software development. The focus of this new approach is on development of new systems by selecting and assembling a set of existing components within an appropriate software architecture. This approach provides various advantages like reduced effort ,time and cost during software development , improved software system maintainability and flexibility by allowing new components to replace old ones and enhanced system quality by allowing components to be developed by those who are specialized in the application area and systems to be built by software engineers who are specialized in component-based software development. Although CBSD has proven its usefulness in the field of software engineering but it still consists of a number of issues or factors that affect many aspects of CBSD like integration and testing effort and affect the produced software cost, maintainability,etc .In this paper some issues and the techniques to resolve those issues have been discussed . Thus a technique has been proposed for Component Based Software Development Organizations(CBSDO) to be used during the development of the software system by using CBSD approach.

KEYWORDS: Black Box Component, CBSD, CBSDO, Component, Reusability Software Architecture

INTRODUCTION

Over a few decades, software systems are becoming comparatively larger and more complex than before. Use of the traditional software development approach to this new situation may result in problems like failure to meet deadline, budget and quality requirements. It was realized that by developing software products each time from the scratch, it would never be possible to overcome these problems. Thus the concept of software re-usability was developed which provides the idea of Component Based Software Development (CBSD). Now CBSD has become a very popular and effective approach for software development. Component based development has improved quality, productivity and effective management for complex software. The differences between traditional software development and CBSD approaches have been shown below in table 1.

Table 1: Differences between CBSD and Traditional Software Development

Component Based Software Development	Traditional Software Development
Building system from pre-existing components .	Each time building system from scratch.
Appropriate component selection and evaluation are special life cycle phases .	In the life cycle, there is no special phase like that.
Much effort is required in the selection of components, testing and verification phases.	Much effort is required for system development.
Reusability is the main theme.	Reusability usually not Considered.

Although CBSD has proven its usefulness in the field of software engineering but it still consists of a number of issues or factors that affect many aspects of CBSD like integration and testing effort and affect the produced software cost, maintainability and quality etc. In this paper a case study of SMDM(Student Marks Details Management) system has been

discussed to describe some issues affecting CBSD approach and the way to resolve those issues . Thus a technique has been proposed for Component Based Software Development Organizations(CBSDO) to be used during the development of the Component based software to provide a maintainable, good quality and low cost software .

COMPONENT BASED SOFTWARE DEVELOPMENT LIFE CYCLE

CBSD Life Cycle includes all the activities and work products required to develop a component based software system.

Main Phases of Component Based Software Development Life Cycle

- **Component Selection:** Selects potential components for reuse.
- **Component Qualification:** Qualifies the components to be sure that they properly fit the architecture for the system. Alternatively, create a proprietary component to be used in the system.
- **Component Adaption:** Adapts components if modifications must be made to properly integrate them.
- **Component Testing:** Test components after adaption**.**
- **Components Assembling:** Integrates the components to form subsystems and the application as a whole.
- **System Evolution:** Replace earlier versions with later versions of components.

FACTORS AFFECTING COMPONENT BASED SOFTWARE DEVELOPMENT

Parameter Incompatibility

During component based development when a component is integrated with other components the exchange of data may occur between them. But some times problems may occur in the exchange of data between components. Because in the case of component based development the components may be from different vendors and most of the times the source code is not provided with the black box components. So it may be difficult for the component consumer to predict the functionality of black box components. So it may be impossible or very difficult to modify the black box component (if customization interfaces are provided). Thus it is difficult to use black box component during component based development. For example when the value returned by one component's function is passed to the another component's function as an argument to perform its function but their data types are different then there will be parameter incompatibility problem. In this case an error may arise and it may affect the other component's functionality connected to the affected component. This may also results in wrong output and in some cases the system may hang. This problem can be reduced by selecting the independent components or the components having low coupling with other components ,during the component based development.

Interface Complexity

Controlling and minimizing software complexity is one of the most important objective of each software development paradigm because it affects many other aspects like software reusability, testability and maintainability etc. Interface complexity is an important factor to be considered while selecting a component during CBSD. The interface complexity can be defined by considering its interactions (interfaces) with other components. So the selected component should have low number. of incoming and outgoing interactions with other components, in another words the component should have low coupling with other components. Less interface complexity helps in reducing the integration and maintenance efforts. Thus the components having less complex interfaces can be easily integrated with other components

to provide the required functionality and are more reusable . Thus by using the independent components or the components having less number of incoming and outgoing interactions i.e having less interface complexity ,the integration and maintenance effort can be reduced during CBSD. It will also make it easy to understand the use of component for the component integrator.

Testing

Testing is an important part of every software development process. To provide a good quality software product testing must be done thoroughly. But in case of CBSD , now a days black box components are being generated to be reused and in most of the cases the source code is not provided with the component by component vendors. So it is very difficult to trust the functionality of black box component by the component consumer. So the component consumer needs to test the component appropriately. But it is very difficult for component consumer to generate appropriate test cases because in most of cases source code is not available. So it results in difficulty for testing component behaviour and it is even more difficult to test the whole behaviour of a software system build by integrating the black box components having high coupling. Because it will be more difficult to generate the test cases , track errors and fix them. But the use of independent components or the components with low coupling will result in less number of and simple test cases and it will be easy to find the error.

Less Ease of Modification and Replacement

Maintenance is also an important part of every software development life cycle. Because it helps in keeping the system up to date and reduces the chances of software failure .Thus sometimes for the purpose of software system maintenance a need may arise for replacing a component with another component or making some modifications in the existing component. But this may arise some problems .This may lead to the requirement of making modification in the components that are interacting with the modified or replaced component. This will consume more effort, time and cost. Thus the component that is less replacable and modifiable should be avoided during the component based software development. This problem can also be solved by using independent components or by using components that have less coupling with the other components.

SMDM SYSTEM: A CASE STUDY

In this paper, a case study of SMDM (Student Marks Details Management) system has been considered to illustrate the impact of some factors affecting CBSD process, software product developed by using CBSD process and the techniques to resolve those factors. This system stores student marks details for different departments in their respective files. In this system first of all ,the user have to log in by providing correct password. Then a list of different departments will be displayed from which the user can select any department for which the user wants to store the student marks details in a file or check the student marks details for that department. Whole system can be build from the components. Like SMDM system has been divided in the form of components as shown in Figure 1. In this case study the components in C++ language has been considered.

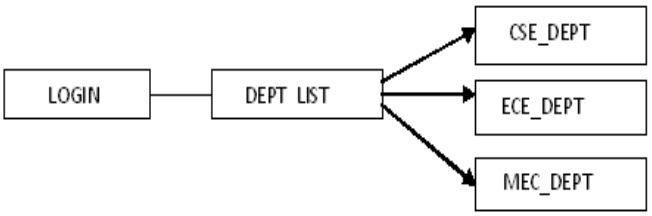

Figure 1: Representation of SMDM System in Form of Components

LOGIN: This component is responsible for authenticating the user while the user provide the password to log in the system.

DEPT_LIST: This component is responsible for displaying the list of different departments for which user want to keep the record of student marks details.

CSE_DEPT: This component is responsible for keeping the record of student marks details for CSE department.

Similarly the components ECE_DEPT and MEC_DEPT are responsible for keeping the record of student marks details for ECE and MEC departments respectively.

Description of the Working of Different Components of SMDM System

LOGIN Component

This component receives the password from the user and compare it with the password stored in a file named 'password.txt'. If match is found then it returns 1 otherwise 0.

The interface of LOGIN component has been shown below:

```
class LOGIN
  { public:
    virtual int check_password()=0;
  };
```

DEPT_LIST Component

If the password is correct then the component named DEPT_LIST is enabled. This component displays the list of various departments for which the user want to keep the record of student marks details . From the displayed list the user can select any department for which he/she want to record the student marks details or check student marks record.

The interface of the DEPT_LIST component has been shown below :

```
class DEPT_LIST
 {    public:
      virtual int  select_dept()=0;
 };
```

The implementation detail for DEPT_LIST component has been shown below:

```
class choice: public DEPT_LIST
 {  public:
    int select_dept()
    {  int a;
       clrscr();
       cout<<"\nEnter 1 For CSE Students Marks Regarding Operations";
       cout<<"\nEnter 2 For ECE Students Marks Regarding Operations";
```

```
        cout<<"\nEnter 3 For MEC Student Marks Regarding Operations";
        // Like store marks details or check marks details.
        cin>>a;
        if(a==1||a==2||a==3)
           return a;
        else
        {    cout<<"Enter valid option";
            return 0;
        }
         clrscr();
     }
};
```

This component is coupled with three components as shown in the Figure 1. It passes the return value to the connected components CSE_DEPT, ECE_DEPT and MEC_DEPT to conform the selected department. .

CSE_DEPT Component

This component performs two functions, one for recording the student marks details in a file named "cse_record.txt" and second for checking the student marks details for CSE department.

The interface of CSE_DEPT component has been shown below:

```
class CSE_DEPT
  { public:
    virtual void cse_marks( char)=0;
    virtual void check_CSEmarks( char)=0; };
```

The implementation for CSE_DEPT component has been described below

```
#include<fstream.h>
class cmarks : public CSE_DEPT
 { public:
   void cse_marks( char d)
   { clrscr();
      if d=='c';
       {     // implementation of  cse_marks(char d) function.
          // Enter the roll no between the range 1-50.
```

```
        // Enter the marks for sub1 in integer form between the range 0-100.
       // Enter the marks for sub2 in integer form between the range 0-100.
      // Enter the marks for sub3 in integer form between the range 0-100.
     // Enter the marks for sub4 in integer form between the range 0-100.
    // Enter the marks for sub5 in integer form between the range 0-100
   // Store the roll no. and marks details for different subjects for that roll number in 'cse_record.txt' file.
  }
}
void check_CSEmarks( char d)
{  clrscr();
  if d=='c';
   { // Display the student marks details by reading file "cse_record.txt";
   }
 }
};
```

It has been assumed that ECE_DEPT and MEC_DEPT have similar kind of interface description and implementation.

POINTS TO DISCUSS

- In this case study of SMDM system , LOGIN component acts like an independent component. Because it does not pass any information to the another component named DEPT_LIST . The value returned by LOGIN component is used only in glue code to connect this component to the DEPT_LIST component. The value returned by LOGIN component will be checked in the glue code upon the basis of which the DEPT_LIST component will become enabled or remain disabled. Use of this kind of component make it easy to replace this component or modify this component without arising the need for modifying the another components that are connected to it like DEPT_LIST component. There will be need to change glue code only. Thus this component can be easily integrated with another components and if any error occur it can be easily tracked. This can be easily replaced or changed with the changes in glue code only. Thus it will help in reducing the integration effort and increasing the maintainability of software system.
- When DEPT_LIST component executes it displays a list of departments from which user can select any department. When the department is selected to perform student marks details management regarding operations then 1,2,3 integers are return for CSE, ECE and MEC departments respectively. When this component is integrated with CSE_DEPT, ECE_DEPT, MEC_DEPT components , this may cause incompatibility problems. Because, the value return by this component is passed to the CSE_DEPT, ECE_DEPT,MEC_DEPT components to conform the selected department. Then there will be parameter incompatibility problem because DEPT_LIST component's function returns integer value for selected department but the CSE_DEPT component use char value

'C' to conform the selected department. Similarly ECE_DEPT and MEC_DEPT components use char values 'E' and 'M' to conform the selected department. Thus it may create problem during component integration.

This kind of problem may be solved by writing appropriate glue code for connecting the components. In which the value return by one component will be converted in required form before passing as an argument to another component's function. But this task may require the detailed working description of connected components which may not be available. This may seem easy here but in the case of complex component based systems this may require lot of effort and cost.

This problem can also be solved by using the independent components or by using the components that have less coupling with another components. In that case either the component will perform all of its operations by itself or it will have minimum incoming and outgoing interactions with the another components. So it will reduce the effort, cost and time to integrate the components and it will also help in reducing the testing effort and increasing the maintainability of the resulting system.

- The interface complexity also affects CBSD process. The interface complexity can be defined by considering its interactions (interfaces) with other components. So the interface having less number of services with limited number of incoming and outgoing interactions should be selected during CBSD. If a component has many incoming and outgoing interaction then it will depend upon another components to provide its functionality. It will require more effort, time and cost to integrate the component with another components. So it may reduce the ease of replacement and modification of a component. By using the independent component or the component with less coupling this problem can be solved, because these kind of components have less interface complexity. It will also help component integrator to easily understand the services and use of services of the component. There are many metrics which can help in measuring the interface complexity of a component.
- As discussed in the section III, testing is an important part of CBSD life cycle. But in case of component based system in which components are heavily coupled with the another components it is difficult to track the behavior of a component and more number of and complicated test cases may have to be used for testing. But use of the independent component or the component having low coupling will help in reducing the number and complexity of the test cases.

  For example to test the functionality of CSE_DEPT component, which is coupled with DEPT_LIST component, the following Table 2 has been generated. Table 2 represents the various combinations of inputs and expected outputs to test the functionality of CSE_DEPT component. Thus in case of CSE_DEPT component 31 different test cases will have to be considered to test its functionality. But use of the component that are independent or have less incoming and outgoing interactions may reduce the number of test cases and it will also cause in generation of simple test cases. Suppose there exists an another component named CSE_DEPT1 that is an independent component. The interface of CSE_DEPT1 component has been shown below:

```
class CSE_DEPT1
{  public:
   virtual void cse_marks( )=0;
   virtual void check_CSEmarks( )=0;
  };
```

Use of this independent component instead of CSE_DEPT component in system will reduce the number of test case from 31 to 25. different as shown below in the form of Table 3. That are less and simple than in the cases of Table 1 as shown above.

Table 2: Different Combinations of Inputs and Expected Outputs for Test Cases for CSE_DEPT Component

Seq No.	Function Name	Input	Expected Output
1	cse_m arks(char)	‘ C’	Display the page for recoding student marks details for CSE department in “cse_record.txt” file
2	cse_marks(char)	Any other character	Does not respond
3	cse_marks(char)	Any variable of other data type	Does not respond
4	cse_ma rks(char)	1<=Roll_no<=50	Store the roll_no in file.
5	cse_marks(char)	Roll_no<1	Enter correct information
6	cse_marks(char)	Roll_no>50	Enter correct information
7	cse_marks(char)	Value other than integer data type for Roll_no	Enter correct information
8	cse_marks(char)	0<=sub1<=100	Store marks details for sub1 in file.
9	cse_marks(char)	sub1<0	Enter correct information
10	cse_marks(char)	sub1>100	Enter correct information
11	cse_marks(char)	Enter value other than integer or float for sub1	Enter correct information
12	check_CSEmarks(char d)	‘ C’	Student Marks details for CSE department can be viewed.
13	check_CSEmarks(char d)	Any other character	Does not respond
14	check_CSEmarks(char d)	Any variable of other data type	Does not respond
15	check_CSEmarks(char d)		Stored information is represented in required for m.

As shown in Table 2 there are 4 test cases for sub1. Similarly there will be 4 test cases for each of sub2, sub3, sub4, sub5 i.e . Thus there will be total 31 test cases.

Table 3: Different Combinations of Inputs and Expected Outputs for Test Cases for CSE_DEPT1 Component

Seq No.	Function Name	Input	Expected Output
1	cse_ma rks( )	1<=Roll_no<=50	Store roll_no in file.
2	cse_marks( )	Roll_no<1	Enter correct information
3	cse_marks( )	Roll_no>50	Enter correct information
4	cse_marks( )	Any another data type value for Roll_no variable	Enter correct information
5	cse_marks( )	0<=sub1<=100	Store marks details for sub1 in file.
6	cse_marks( )	sub1<0	Enter correct information
7	cse_marks( )	sub1>100	Enter correct information
8	cse_marks( )	Any another data type value for sub1 variable	Enter correct information
9	check_CSEmarks( )		Stored information is represented in required form

As shown in Table 3 there are 4 test cases for sub1. Similarly there will be 4 test cases for each of sub2, sub3, sub4, sub5 i.e . Thus there will be total 25 test cases which are less than the test cases for CSE_DEPT component.

Thus if the independent component or the component having low coupling are used in building a software system then less number of and simple test cases will be requires to test the each component functionality and it will also be easy to test the functionality of the system as whole. It will make it easy to track the errors. This results in good quality product.

- As it has been discussed in section III, less ease of modification and replacement of component in the system reduces the maintainability of the system. Because it may result in increasing the effort and cost for replacing or modifying a component. As shown in Figure 1, suppose the DEPT_LIST component functionality has been increased to pass the department number and department name to conform the selected department. But the CSE_DEPT, ECE_DEPT and MEC_DEPT components can receive only department name. Then it will affect the functionality of these components that are coupled with component DEPT_LIST. In this case there will be need to make modifications in the CSE_DEPT, ECE_DEPT and MEC_DEPT components which may not be possible or very difficult(if customization interfaces have been provided) in case of black box components. Thus this factor will affect whole software system. Because it may be necessary to replace or modify a component for maintenance purpose.

This problem can also be solved by using the independent components or the component having low coupling with another component. Because in this case if the component is replaced or modified, it will not affect the functionality of other components that are connected to it or it will affect as less as possible. So the component can be replaced or modified with low cost and effort. Thus it will help in increasing the maintainability of the software system

For example suppose similar to component CSE_DEPT1, there exists the components ECE_DEPT1 and MEC_DEPT1 which have similar interface description as in the case of CSE_DEPT1 component and are independent component.

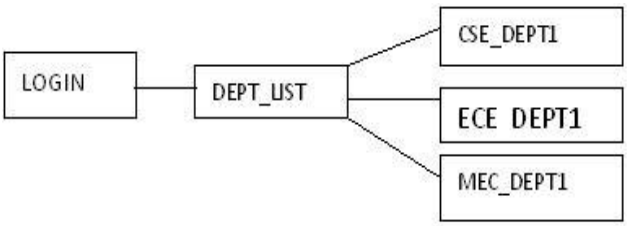

Figure 2: Representation of SMDM System Using Independent Components

Suppose CSE_DEPT1, ECE_DEPT1 and MEC_DEPT components have been used instead of ECE_DEPT and MEC_DEPT components respectively to develop the SMDM system as . Then it will help in solving the above problem. Because as shown in Figure 2 now no return value is passed from the DEPT_LIST component to CSE_DEPT1, ECE_DEPT1 and MEC_DEPT1 components. The returned value will be used in glue code to connect the CSE_DEPT1,ECE_DEPT1 and MEC_DEPT1 components to DEPT_LIST component and to conform the selected department. Thus by using CSE_DEPT1,ECE_DEPT1,MEC_DEPT1 components for developing SMDM system , the system can be easily updated.

Because these components can be easily modified or replaced with another components without affecting any other components like DEPT_LIST .Similarly if the component DEPT_LIST is replaced or modified it will not affect the functionality of other components . Because now the component DEPT_LIST will have no outgoing interaction with other components, There will be need to modify the glue code . Thus it will be easy to made modification or updations in the system.

As discussed in this case study, a component's coupling with other components affects integration and testing effort, cost, maintainability and other quality attributes. Thus some of the various factors that affect CBSD process and the resulting product can be resolved by using the independent component or the component having less coupling.

PROPOSED WORK

The following steps can be followed by any Component Based Software Development Organization during the component based development in order to resolve the factor affecting integration and testing effort, cost, maintainability and other quality attributes of the resulting system. Thus it will help in reducing the integration and testing effort, cost and providing the more maintainable and good quality software system.

- **Component Selection:** Selects potential components for reuse.
- **Component Qualification:** Qualifies the components to be sure that they properly fit the architecture for the system. Alternatively, create a proprietary component to be used in the system
- **Test Coupling:** Test the coupling of components**.**
- **Selection Based on Coupling:** Select independent components or the components with low coupling which fulfills other necessary criteria too.
- **Component Adaption:** Adapts components if modifications must be made to properly integrate them.
- **Component Testing:** Test Components after adaption**.**
- **Components Assembling:** Integrates the components to form subsystems and the application as a whole.
- **System Evolution:** Replace earlier with later versions of components.

CONCLUSIONS

Although the CBSD is increasingly being adopted for software development, because of its advantages like reduction in development effort, time and cost , increase in quality with many others. But it still consists of a number of issues or factors that affect many aspects of CBSD like integration and testing effort and affect the produced software cost, maintainability, quality etc Many of these factors or issues can be resolved by using the independent components or the components with low coupling. Thus a technique has been proposed for the component based software development organizations to be followed during the component based software development. Thus it will help in reducing the integration effort, testing effort, cost and providing the more maintainable and good quality software system.

REFERENCES

1. Navneet Kaur, Ashima Singh, “Generating More Reusable Components while Development: A Technique, International Journal of Innovative Technology and Exploring Engineering , Volume-2, Issue-3, February 2013.
2. W. B. Frakes and K. C. Kang, "Software reuse research: status and future," IEEE Transactions on Software Engineering, vol. 31, pp. 529-536, 2005.
3. M. Morisio , "Success and Failure Factors in Software Reuse," IEEE Transactions on Software Engineering, vol. 28, pp. 340-357, 2002.
4. Dan Laurenţiu Jişa , “Component Based Development Methods – comparison ,” International Conference on Computer Systems and Technologies – CompSysTech,2004.
5. B. Weide , "Reusable software components," Advances in computers, vol. 33, pp. 1-65, 1991

6. Ivica Crnkovic, Stig Larsson and Judith Stafford, "Component-Based Software Engineering:Building systems from Components ," 9th IEEE Conference and Workshops on Engineering of Computer-Based Systems,May 2002.
7. Gui Gui and Paul. D. Scott , "Measuring Software Component Reusability by Coupling and Cohesion Metrics", JOURNAL OF COMPUTERS, VOL. 4, NO. 9, SEPTEMBER 2009.
8. Nasib S. Gill , "Reusability Issues in Component-Based Development", Department of Computer Science & Applications,M.D. University, Rohtak , Haryana
9. Dogru, A.H. and Tanik, M.M. , "A process model for component-oriented software engineering ," IEEE Software, Vol. 20(2), 34-41,2003.
10. SONG Cui-Ye, DU Cheng-Lie , "Formal Interface-Component Based SoftwareAnalysis and Design", School of Computer Science and Technology, Northwestern Polytechnical University,China,2010.
11. Usha Kumari and Shuchita Upadhyaya, "An Interface Complexity Measure for Component-based Software Systems", International Journal of Computer Applications , Volume 36– No.1, December 2011.