Comparative Study of Object Oriented Design and Component Based Software Engineering

Aalisha Sheth[1] Biyanta Shah[2] Aayushi Shah[3]
[1,2,3]Institute of Technology, Nirma University

Abstract— Object Oriented Approach and Component Based Software Engineering are two of the popular software paradigms. In this paper we have shown how these two paradigms differ from each other by conceptual comparison. This paper will cover the property comparisons of these two concepts and will also conclude a better concept in a particular situation. It will also cover the future work possible in this field.

Key words: Object Oriented, Component based components, objects and classes, design, software development paradigms

I. INTRODUCTION

Object oriented approach is the process of preparing a system to interact with objects so as to solve a particular problem. It is one of the methods for software designing. Objects contain data, which represent the functionalities of data or point to a particular function of the software. Object oriented analysis analyzes the problems being faced by system and accordingly designs the objects and classes. Classes are the basis from which objects are created. Objects may be distributed and they may be executed sequentially or in parallel [1]. Objects are also easy to maintain and they are reusable for various components. There are four main pillars of object oriented programming which are Inheritance, Polymorphism, Encapsulation and Data Abstraction. These concepts provide the security and privacy that is needed for data, increases the power of programming language by creating user defined data types [2].

Component based software engineering is a branch of software engineering that focuses on combining different components from various applications to form an entirely new application. It is typically based on reuse approach to define and collaborate separate components into one. Component Based Software Engineering (CBSE) was developed due to the failure of object-oriented approach. Object oriented approach had too specific classes while CBSE has abstract classes and thus are considered to be lone service providers [3]. There are four main functionalities in CBSE like distinct components which are defined by their particular interfaces, specific component standards which help in component integration, middleware for interoperability and a development module that can be used for reuse [3]. CBSE also has independence feature because the components do not interfere with each other and also their implementations are well hidden. This gives some amount of security to the developer. The platforms on which the components work are shared and thus reduce the costs [3].

In this paper we provide a top down approach of the Object-oriented systems and component-based systems conceptual comparison and their architectural and qualitative analysis, which would clarify the boundaries between OOA and CBSE. We also identify the various criteria that affect the core qualities of these paradigms.

The flow of this paper goes as follows. Following the Introduction, in section II we have the conceptual differences between these two software development paradigms. In section III we cover the quality analysis of the paradigms. In section IV we have the quantitative and qualitative aspects, which include their subsections as well. Section V contains the future work that is possible in this topic. The paper ends with the conclusion, acknowledgments and references.

II. DIFFERENCES IN CONCEPTS

In Component Based Software Engineering (CBSE) the components which are to be included in the new application are to be integrated into the software and the components need to communicate with the interfaces in order to make the application work [4]. In Object Oriented Approach (OOA) according to the need and requirement of the software to be developed, to solve certain problems (or find solutions) the various classes and objects are programmed and integrated.

In CBSE only the functioning of the various components, their maintenance and the quality of service of the components integrated in one application are to be handled by the component provider. The management of services, executing the service and deploying, versioning of the service all are the functions of the service provider. While in OOA all the classes and objects belong to the same service provider and thus all the functionalities to be handled, the services to be provided, maintenance and other functions are all provided by the service provider. Also the services of OOA are remotely executed and have to handle multiple connections while in CBSE though the components can be used at different places but they would not be related to each other. Hence OOA has a higher stake in ownership than CBSE.

Heterogeneity is another important property of software. In CBSE as we know the different components that make up the applications are picked up from the distinct sources and they have to be integrated to work with each other. Thus heterogeneity is a well-known factor in CBSE. While in OOA the classes and objects are all programmed into one application so these do not need to be compatible. Thus if the classes and objects are taken from different applications heterogeneity could be a problem for integration. Thus heterogeneity property does not create a problem in CBSE but in most cases of OOA it does create a problem. However in CBSE there are a large number of component models to choose from but only one model can be chosen from because it is very difficult to combine components of all the different models [4].

Coupling is the next property we will discuss. Coupling is the amount of dependency on another program module [6]. Coupling can be low or high. Low coupling is better because it depends less on its other modules and hence less dependability means less complications when some changes are made. In Object-oriented all the objects and classes are general and external hence coupling is

reduced. In CBSE design of an application is done according to requirements hence there will be dependencies, thus coupling is high here.

Contrary to coupling is another property known as cohesion. In cohesion related functionalities are put into a single unit. Cohesion should be high because high relativity means better interoperability. In CBSE the components are not at all related to each other. They are taken or downloaded from different applications. Thus cohesion is low. While in Object-oriented approach all the objects are related to one another because the solution to the software problem is to encapsulate all the information together [6]. Hence the components in OOD are related to each other and cohesion is high.

The properties that we saw above are related to the dynamic nature of the two concepts. As we noticed CBSE is more for general-purpose applications.

III. Quality Analysis

The properties that we will discuss now belong to the core of these concepts. These properties decide the usability of the software development paradigm. These properties are the foundation of discussion of other properties like flexibility, safety, security, privacy etc. [4].
There are three main properties:

A. Reusability

Reusability means to reuse the components of the model again and again without any modification in the components.

B. Composability

Composability is to safely integrate all the architectural elements to create a new system [4].

C. Dynamicity

Dynamicity means to develop applications, which adjust to the change in the developing environment or the user requirements. Dynamicity is to adjust to these changes automatically and autonomously.

For these two development paradigms the comparison goes as follows:

Reusability is using the same components again and again. In CBSE the same components cannot be used again without some minor modification. While in object-oriented approach the same objects can be used in some other application program without any modification. Thus Reusability is higher in OOA.

Composability is to integrate the architectural elements. The main crux of CBSE is to integrate various architectural elements to form a new application. In OOA every architectural element may not be compatible and may fail to integrate. Thus Composability is stronger in CBSE.

Dynamicity is automatically adapting to the changes in the environment of the developing platform or requirements of the user. CBSE paradigm develops the components taken from different applications in such a way that they dynamically adjust themselves to the changes in the environment or changes in the user requirements. In OOA the objects need to undergo a change if their platform is changed. Thus dynamicity is stronger in CBSE.

We have depicted these properties as discussed above comparing these two paradigms in a graphical approach as shown below in figure 1.

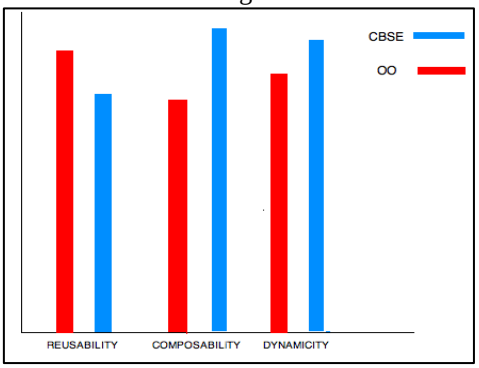

Fig. 1 Comparison of core qualities

IV. Quantitative and Qualitative Aspects

A. Quantitative Aspects

The quantitative aspects define the concepts and structure of the two paradigms. It gives the important theoretical view of each paradigm. The most crucial categories are Product and Process.

A Product is an entity, which is due to some action or due to a process [4]. A process is a set of specific actions that are performed to get a specific product or modify one.
The product category is divided into two parts:

1) Basic architectural elements
- These form the basic architectural structure of the paradigm.

2) Composite architectural elements
- These are the modified version of the already existing architectural elements. This points out the reused elements and their relationships.
- There are two more abstraction levels namely the design time and the run time in this category.

In CBSE the basic architectural elements at design time would be component and connector types while at run time they would have some instances of connectors and components [4]. In Object-Oriented approach the basic element in the design time would be the class while at run time it would be the object.

In CBSE the composite architectural elements at design time would be composite components and connectors while at run time would be instances of composite components and connectors. In Object-Oriented approach the composite elements in design time will be the composite class and at run time will be composite object.

We discussed the basics of product so now we turn towards the process. Process category targets the reusability in these two paradigms. The process category is further divided into three categories, which are based on abstraction and description levels

1) Inside Description Level
- All the processes in the same description level between two abstraction levels are congregated.

2) Between Description Levels
- Processes that target products from different description levels than the one from which it was produced are congregated. It has design time and run time levels [4].

3) Between Abstraction Levels

- The processes targeting product from design time and then producing a product from run time are congregated.

For Inside Description Level during design time CBSE is associated with horizontal composition, refinement and selective inheritance and object- oriented system is associated with inheritance processes [4]. While during run time where the communication processes are the main aspects, CBSE uses functionality call and object-oriented system is associated with different method call processes.

For the between description levels at design time in CBSE, takes help of vertical composition while object oriented system takes help of composition process to develop a composite architectural element. During run time it is the same case as inside description level where for CBSE which takes help of functionality call and delegations and Object oriented system takes help of method call processes [4].

Abstraction levels need to link types and instances for example in object-oriented class and object while in CBSE components and connectors. Both CBSE and object-oriented have instantiation processes to rely on for linking types and instances [4].

This subsection was all about the structure of the paradigms and how they function. Now we move towards the software quality criteria.

B. Qualitative Aspects

The software quality criteria that we will discuss now are based on several applications domain systems. The main aim of this section is to see which of these criteria impact the core qualities and classify these two paradigms; CBSE and Object-oriented approach according to that.

There are six main features, which reflect the impact on the core qualities discussed above, namely reusability, Composability and dynamicity. These are:

1) Loose Coupling

- It measures the amount of dependencies between separate entities.

2) Expression Ability

- Expression Ability is based on the amount of concepts and processes that are provided by the paradigm to modify its functions.

3) Abstraction of communication

- For security purposes we need to abstract the communication layer which executes the applications so that data is not manipulated in any way. Thus this property defines the ability of a paradigm to provide this kind of secure abstraction at the communication layer.

4) Explicit architecture

- The user needs to know about the architecture of the paradigm before using its applications. Ability of a software paradigm to give a clear-cut view about the architecture of the paradigm is known as explicit architecture.

5) Evolutionary ability

- Over a point of time it might be necessary for architectural elements to be able to evolve. The ease with which they can evolve to provide powerful concepts is known as evolutionary ability.

6) Ownership

- Some of the services such as maintenance, execution, and management are with the provider of the composite architectural element. It sometimes allocates the responsibilities of these services to the customer. These responsibilities are known as ownership, which show the level of trust and liberty given by the provider to the customer.

Loose coupling for an object based system is rare because they are basically tightly coupled while component based systems are comparatively loosely based. Hence loose coupling is considered to be a challenge for component-based systems.

Providing functionalities is the gist of Expression ability. In CBSE there are not much such functionality. The functions performed in CBSE are mostly based on object - oriented systems. Though lately CBSE has evolved with some advanced concepts such as inheritance and polymorphism [4] but they have still not got the level that object-oriented system attained. As we know in object-oriented programming we have numerous functionalities like polymorphism, abstraction, inheritance, encapsulation, granularity etc. Thus Object-Oriented systems have higher expression ability than Component based systems.

Abstraction of communication deals with the abstraction of communication layer while executing the applications. In CBSE there is no provision for abstraction functionality. Also in CBSE the communication takes place inside the connectors and thus global behavior is split. The workflow is also not explicit and thus harder to manipulate. While in Object-oriented systems have fine granularity due to classes. These properties accentuate the drawback of abstraction of the communication layer.

Giving an overview of the architecture of the paradigm is Explicit Architecture. In CBSE this quality is enhanced up to some extent but it still needs to be developed more [4]. Object oriented systems certainly lack this over view because it is only one system working within itself. So the architecture is not specified properly.

Evolutionary Ability is related to explicit architecture. The architecture is represented as nodes and edges in a graph. Thus the nodes or edges are targeted for evolution. As we know, CBSE can study the evolution from the graph because it supports explicit architecture. While Object-oriented is devoid of explicit architecture and thus there is no focus on its graph.

For Ownership the component based systems disintegrate the responsibility at deployment level. The service provider gives liberty to the customer and the customer is responsible for execution and management of the application. While in Object-Oriented systems there is absolutely no kind of ownership. All the processes are transparent between the provider and the customer and the customer is free to do what he likes.

We have depicted these properties as discussed above comparing these two paradigms in a graphical approach as shown below in figure 2.

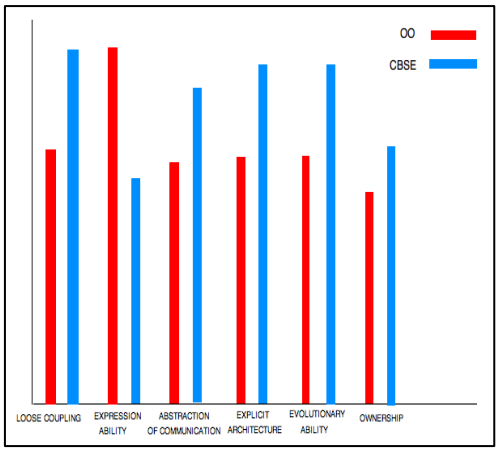

Fig. 2: Comparison of features

V. FUTURE WORK

As in every software paradigm or applications these paradigms also have problems. The most serious issue is the trustworthy property of the component. The component coming from a remote source cannot be trusted. Also the certification of various components is a troublesome question. These problems are being worked upon to make component-based systems efficient enough.

Also the hierarchy in properties shown in the two graphs is only on the basis of theoretical information and they certainly cannot be used for technical approach of ranking these paradigms. For a fully technical approach other features not included in this paper will have to be compared.

VI. CONCLUSION

This paper defines the process, product and quality of the two software paradigms and compares them so as to provide a better understanding of CBSE and Object-oriented Systems.

From this paper we have seen that CBSE is a better paradigm when we have to use different components from different applications and work on interoperability so as to make them work in a single application. But OOA is more useful in cases of coupling and cohesion and also provides more security than CBSE systems. Hence according to the requirements of the user, he/she can select any one paradigm most suited for the particular application.

ACKNOWLEDGMENT

We would like to thank our professor, Prof Dhaval S. Jha who mentored us during our paper and guided us with his experience and useful comments.

REFERENCES

[1] http://www.codeproject.com/Articles/567768/Object-Oriented-Design-Principles.
[2] http://www.slideshare.net/sudarshan/object-oriented-design
[3] Sommerville, "Software Engineering": Pearson, Eighth Edition.
[4] Anthony Hock-Koon, Mourad Oussalah, "Product-Process- Quality Framework" in 2011, 37th EUROMICRO conference on Software Engineering and Advanced Applications.
[5] sst.umt. edu.pk /courses/cs540/CouplingandCohesion-student.ppt
[6] https://in.answers.yahoo.com

