Available online at http://www.journalcra.com

International Journal of Current Research
Vol. 7, Issue, 09, pp.20019-20026, September, 2015

INTERNATIONAL JOURNAL OF CURRENT RESEARCH

ISSN: 0975-833X

RESEARCH ARTICLE

COMPONENT-BASED SOFTWARE ENGINEERING: A PRAGMATIC APPROACH TO SOFTWARE DEVELOPMENT IN 21ST CENTURY

***Ugah, J.O. and Ugwu, G. E.**

Department of Computer Science, Ebonyi State University, Abakaliki-Nigeria

ARTICLE INFO

Article History:
Received 15th June, 2015
Received in revised form
25th July, 2015
Accepted 23rd August, 2015
Published online 16th September, 2015

Key words:
Software Development,
Components-Based,
Reuse,
Pragmatic,
21st Century,
Adaptability

ABSTRACT

Component-based software engineering approach to software development employs the techniques of selecting reliable and reusable software components and assembling them within appropriate software architecture to form a robust software system. This approach is unique because it brings about designs that make a clear separation between the stable parts of the system from the specification of their composition. The rising need of software in 21st century is indeed of a great concern. In view of the increase in demand for software, software engineers need a proactive and pragmatic approach that will handle the situation on ground. Although object oriented approach helped in giving birth to software that reflect the object of the problem domain, the approach does not necessarily produce software architectures that can easily adapt to the changing requirements of users in this 21st century. In component-based software engineering, the software team will first establish requirements for the system to be built, the architectural design is made and detailed design omitted instead the team examines requirements to determine what subset is directly amenable to composition rather than construction. Component-based software engineering would help software developers to build high-quality, reliable and easily maintainable software that meets user's requirements. It will also bring about significant reduction in the development cost and time since developers can quickly assemble such systems from a set of reusable software components rather than build everything from the scratch. Furthermore, it will lead to designs that will be easily adapted to meet the changing needs of users.

***Citation*:** Ugah, J.O. and Ugwu, G. E.2015. "Component-based software engineering: a pragmatic approach to software development in 21st century", *International Journal of Current Research*, 7, (9), 20019-20026.

INTRODUCTION

We are living in the computer age when most devices are computer-based and hence need computer software to function. Today's society is information driven society and scientist are trying their best to come up with both hardware devices and software systems that would make life easier and help members communicate effectively without much difficulties. Major sectors in local, national and international economy such as telecommunication, electricity generation and distribution, logistic systems, modern cars and airplanes are actually being driven by software. From all indications, software is becoming more and more an integrated part of the 21st century society. Software allows new organizational forms, such as business process reengineering and virtual organizations which allow organizations to operate at efficiency levels that are an order of magnitude better than earlier models (Uwaje, 2012).

****Corresponding author: Ugah, J.O.***
Department of Computer Science, Ebonyi State University, Abakaliki-Nigeria.

Furthermore, most of the devices which aims at increasing effectiveness and efficiency in diverse sectors of the economy would not be available without software. A typical example is the current situation in the communication sector. There is at present an explosion of mobile devices in both developed and the developing countries. Other sectors are also facing similar situation. In view of this development, most organizations are realizing the importance of having unequivocal software architecture for their systems. In trying to come up with such explicit software architecture, software engineers have been faced with the challenge of developing multiple, inter-related systems from the scratch while pressured by financial and time constraints. This challenge has made it very essential to establish a well-defined and proactive approach in order to build high-quality, reliable, and easily maintainable component and/or systems that satisfies user requirements and is delivered on record time (Osuagwu, 2005). The question here is- what is the best approach to adopt in solving this teething problem. It is our strong belief that component-based software engineering would proofer solution to this problem.

The last decade has shown that object-oriented technology is not enough to cope with the rapidly changing requirements of present-day applications (Mohammad, 2001). The truth is that object oriented methodology when used helps in developing rich models that reflect the objects of the problem domain. But, these models do not necessarily yield software architectures that can be easily adapted to changing requirement of users especially in our dynamic present day society. According to (Rob, 2007), one particular loop hole of object-oriented methods is that it does not lead to designs that make a clear separation between computational and compositional aspects but this separation is common in component-based systems.

Component-based systems achieve flexibility by clearly separating the stable parts of the system from the specification of their composition. It is very clear that building new software by combining existing and viable software components would improve and support rapid software development. Rapid software development will in turn lead to the overall growth in software industry to a level that would sustain the rising need of the 21st century. It is therefore imperatives that we adopt component-based development approach in software development in our days since the benefits it offers are enormous. In the section below, we shall take a brief tour of other software development methodologies to help bring out the need for this campaign.

Review of software Development Methodologies

Software undergoes through certain phases to evolve. According to (Alan and Barbara, 2003) the basic software phases are planning, analysis, design and implementation. However, other authors may further break these five to more phases. There are also many software design and development methodologies that have been in use since the evolution of software engineering. Some of these methodologies include the waterfall, rapid application development (RAD), object oriented Methodology, component-based software engineering etc. Let us now discuss some of these methodologies.

Waterfall Approach

The waterfall methodology is about the oldest. Waterfall methodology adopts an approach that is much similar to construction and manufacturing workflows. It follows a sequential design process that begins from conception, analysis, design, construction, testing, implementation, and maintenance (Nilesh, 2005). The basic concept of the waterfall methodology is that you move from one step to another sequentially. It therefore presupposes that you would move from one phase only when it's proceeding phase has been completed. The different phases in waterfall approach include requirements, design, implementation, verification and maintenance. In the requirement phase, the analyst analyzes user's requirement and performs further investigation of requirements that enables him to come up with developers' version of requirements. This phase is usually very critical and important since most users are usually unable to explain what they need very clearly. The design phase here focuses on both architectural and detailed design. The structures of the envisaged system are formed at this stage. The programs that are needed and how the individual programs will work, how the interface design will look like, what data is required and other aspect of the software are also defined. Developers translate the design made into codes at the implementation phase. Any high level language of choice is employed for the implementation. The verification phase is when each module and subsystem is tested. The modules and the subsystems are then integrated and the whole system tested to verify weather the system works as intended.

In the maintenance phase the system undergoes different degrees of amendment either addition of more features or removal and in some ceases restructuring to suite the changing needs of users. These five phases are followed chronologically in the waterfall approach. The waterfall approach appears simple and is easily explainable to the users. Its linear sequential model also makes it orderly and interesting to management as well. There is also a strong believe that other processes depend strongly on the waterfall approach and that it is a prerequisite to the study of other approaches (Robertson and Bednar, 2006). As good as the waterfall approach is, it has a lot of weaknesses that makes it unfit and unreliable for the development of 21st century software. According to (Nilesh, 2005), some of the weakness of waterfall approach includes among others the following:

- The waterfall approach does not make allowance for the change of defined requirements as the project progresses The implication of this is that software developed using this approach stand the risk of not fully meeting up with the requirement of the user.
- Since it does not make allowance for change of defined requirement, if the user has made an error in their requirement the whole processes has to be restarted. This will cause a lot of waste and delay in software development.
- Another weakness of the waterfall approach is that it is time consuming. Due to the fact that going back to any phase already left is difficult, developers' takes time to ensure that all necessary issues are settled and documented before advancing to the next phase.
- Using the waterfall methodology, the whole software product is only tested at the end. The implication of this is that critical errors in code (bugs) written early would be discovered late and at the point when it would be almost impossible to correct. Such bugs usually affect the overall performance of the product.

Considering these weaknesses discussed and many others that we did not highlight here, it is clear that it would be a difficult task depending on the waterfall approach in developing software for the 21st century. It will delay the development process and may not also produce software that can be easily adapted to the changing needs of users.

Rapid Application Development (RAD): The RAD approach to software development is also iterative in nature. The three RAD categories include the phased which involves series of versions, prototyping (system prototyping) and throw-away prototyping (design prototyping) (Alan and Barbara, 2003).

The basic objective of RAD is to provide a platform for fast development and delivery of high quality systems at a relatively low investment cost. According to (Nabil and Govardhan, 2010), RAD produces high quality systems quickly through the use of iterative approach. Prototyping which is one of the categories of RAD brings about active user involvements and computerized development tools.

Computerized development tools such as computer aided engineering (CASE) tools; Database management tools (DBMS), graphical user interface builders and host of others are employed. The RAD approach to system development also gives room for documentation necessary to facilitates future development and maintenance (Selecting a Development Approach, 2008). Although the RAD approach to system development helps in fast system development and faster delivery when compared to the waterfall approach, it posses some inherent weakness that does not suggest that 21st century software developers should completely rely on it to meet up with the rising need of robust software in different sphere of life in our information driven society. Some of the weaknesses include:

Developers and users must be committed to the series and quick activities in abridged time structure (Otto *et al*., 2000). This is not always easy to meet up by both parties. The RAD approach requires a system that could be divided into modules (Contributor, 2006).

- Using this approach may bring about pushing difficult problems to the future since prototypes could just demonstrate some functions of the system in order to win management support.
- It would also take a great effort and planning to have well-defined interfaces since some modules will be completed much earlier than others.

In view of these weaknesses and others, we believe that although RAD is a very good approach to software development, a more proactive and pragmatic approach such as the component-based should be used.

Object Oriented Methodology: Object oriented approach describes the system through a set of business processes it performs as well as the object classes that these processes deal with (Benneth and McRobb, 2002). This methodology employs set of diagrams or models to represent various views and functionality of a system. Tools that are usually used to draw these set of diagrams or create this models is the unified modeling language (UML). Generally object oriented approach takes the following format (i) identify the information system's purpose (ii) identify the information system's actors and features (ii) identify the Use Cases and create a Use Case Diagram (iv) identify Objects and their Classes and create a Class Diagram (v) create interaction/scenario Diagrams (vi) create detail Logic for operations (vii) Repeat these processes as required to refine the plan as the case may be. The states of the system at each phase of the development may include inception, elaboration, construction and transition. Inception deals with the initial work required to set up and agree on terms of the project (Stevens and Pooley, 2000). Elaboration deals with putting the basic architecture of the envisaged system in place (Bell and and Thayer, 1996). Construction deals with building the system while transition deals with processes involved in transferring the system to the clients and users. According to (Booch, 1994), object oriented analysis and design emerged as a complement to object-oriented programming in the 1980s. Object-oriented programming adopts problem solving strategy which approaches every problem from an object (e.g person, place, thing or concept) perspective rather than the functional perspective of traditional approach. This is what the object oriented approach rides on to come up with robust and dynamic systems. Object oriented is therefore timely especially with increased emphasis on graphical user interface (GUI) and software that runs on distributed and heterogeneous, client-server (multi-tier) computer hardware platforms across the Internet.

The truth is that the older software engineering methodologies have limitations in representing a model of the information system that can readily be implemented using an object-oriented programming language primarily due to their focus on functions or data, not object (Lewis, 1995). Object oriented approach promises many benefits such as reduction of development time, and resources required to maintain existing systems, increase code reuse, and provide a competitive advantage to organization that use it (Roger, 2001). Although, object oriented provides the advantages listed above over the traditional approach, Object oriented approach had not led to extensive reuse as originally suggested. Single object oriented system is too detailed and specific and often had to be bound with an application at compile time (Brown, 1997). This makes marketing object as components difficult. Again, another constraints that limits object oriented is its inability to bring about designs that makes a clear separation between the stable parts of the system (the components) from the specification of their composition.

Component-based Software Engineering (CBSE)

Component-based Software engineering (CBSE) is an approach that emphasizes the design and construction of computer-based systems using reusable software component (Baoming, 2000). In today's society, software takes on a dual role. Software is both a product and a vehicle for delivering other products. As a product it delivers the computing potentials embodied by computer hardware and network of computers that are accessible by local hardware. It is an information transformer producing, managing, acquiring, modifying, and transmitting information in our information driven society.

As a vehicle for delivering other products, software acts as the basis for the control of the computer (operating system) the communication of information and the creation and control of other programs (Dayan, 1999). In view of vital roles software is playing in our day, the importance of software has grown. There is therefore a compelling need of technologies that will make development of high quality software easier, faster, and less expensive. There is a great need of software development approaches that would yield software architecture that can easily adapt to the changing requirement of users in our day.

Component-based software engineering gives room for flexibility and helps to yield software architectures that adapts to the changing requirement of users by clearly separating the stable parts of the system (i.e. component) from the specification of their composition.

Materials and Methods for CBSE

Component-based software engineering uses software integrated circuits which are pieces of reusable and reliable software which can "fit" into the architectural style specified for a given system and will exhibit the quality characteristics that are required for the application. The steps that are used begin with the software engineering team establishing requirements for the system to be built using conventional requirement elicitation techniques. An architectural design is then established. The software team searches for qualified reusable components which are in turn adopted if it matches the design architecture. The adopted components are then integrated together following the design made to come up with the desired system. Our simple illustration of the processes involved in CBSE is shown in Figure 1 below.

It has however been established that component-based software engineering encompasses two parallel engineering activities namely domain engineering and component based development. Based on this premise, a process model that supports CBSE was established by (Debayan, 2001) as shown in shown in Figure 2 below.

These components are placed in reuse libraries. Domain engineering is all about finding commonalities among systems to identify components that can be applied to many systems, and to identify program families that are positioned to take fullest advantage of those components (Micheal, 2001). On the other hand component-based development involves sequence of component based development activities applied when a component is proposed for use.

Component-based development elicits requirements from the customer, selects an appropriate architectural style to meet the objectives of the system to be built and then selects potential components for reuse. It goes further to qualify the components to be sure that they properly fit the architecture for the subsystem and adapts components if modification must be made to properly integrate them. Component-based development now integrates the components to form subsystems and the application as a whole. In addition, custom components are engineered to address those aspects of the system that cannot be implemented using existing components. These processes actually requires developing wrappers that glue these reusable components within a given appropriate software architecture. For a software developer to make use of the component-based approach to achieve a tangible result, he must do an extensive analysis into the system to be built and all its parts. He should seek to have an in-depth understanding of the different parts of the system otherwise the components in the resulting system will not be independently producible and deployable.

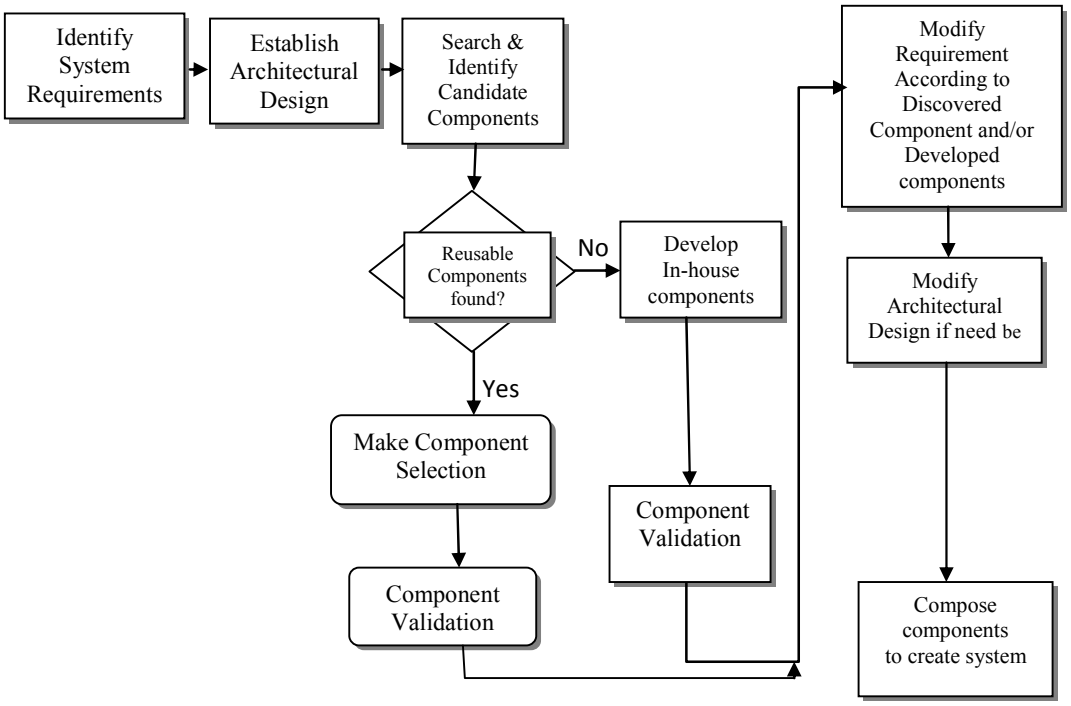

Figure 1. Processes model for CBSE

In domain engineering, the software engineering team explores an application domain with the specific intent of finding functional, behavioral, and data components that are candidates for reuse.

In view of this, we wish to now explain briefly how proper analysis could be carried out in such a manner that it would lead to selecting appropriate software components.

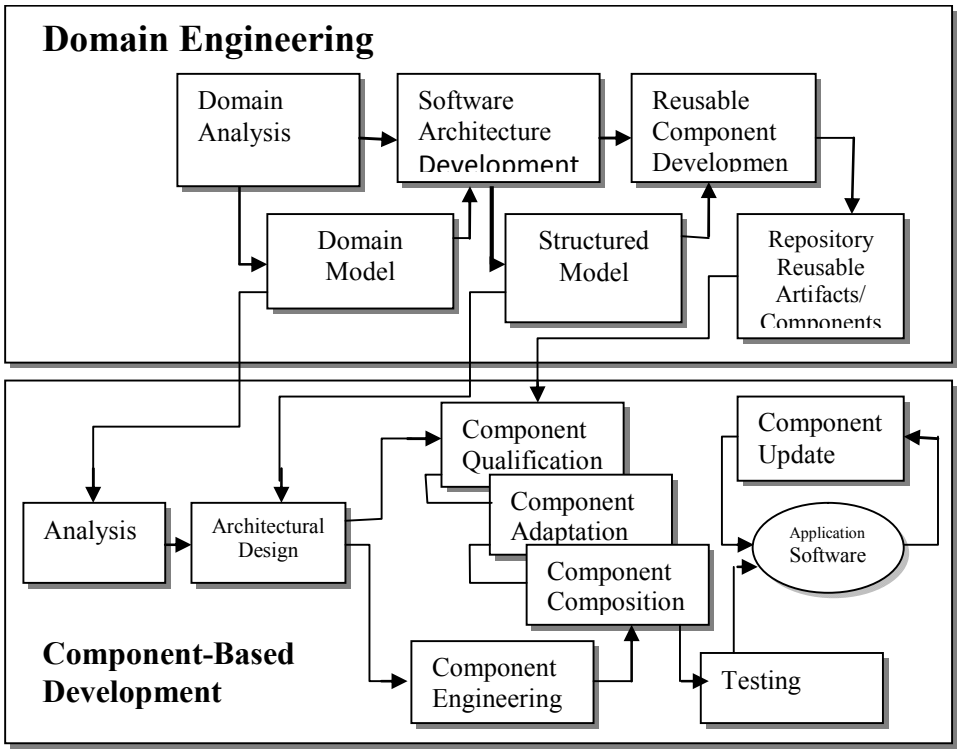

Figure 2. A Process Model that Supports CBSE. Adapted from (Debayan, 2001)

Systems Analysis and Design for CBSE

According to (Jeffrey, 2000), systems analysis is a problem-solving technique that decomposes a system into its component pieces for the purpose of studying how well those components parts work and interact to accomplish their purpose. Systems analysis is a phase that primarily focuses on the business problem independent of any technology that could be used to implement a solution to that problem. Usually, systems analysis is made up of the preliminary investigation, problem analysis and requirement analysis. You will know that you understand the process of systems analysis when you can:

- Define systems analysis and relate the term to the preliminary investigation, problem analysis, requirement analysis and decision analysis of the system development methodology.
- Describe a number of systems analysis approaches for solving business system problems.
- Describe the preliminary investigation, problem analysis, requirement analysis and decision analysis phases in terms of your information system building blocks.
- Describe systems analysis in terms of purpose, users inputs, outputs, techniques and steps

The ultimate goal of systems analysis is to produce a system proposal to improve the member services information system. That proposal will trigger the design construction, and implementation of proposed system. System analysis is driven by the business concerns of system owners and system users; hence, it addresses the data, processes, and interface building blocks from system owners and system user's perspective. Since there are always many approaches to problem solving, there are also many approaches to systems analysis.

According to (Jeffrey, 2000), the basic processes of analysis is divided into three steps

- Identifying the current ("as-is") system.
- Identifying improvements and
- Developing requirements for the proposed ("to-be") system

When no current system exists, understanding the state of things is done in a brief manner. This may also be because the existing system and processes are irrelevant to the future system or if the project team is using development methodology in which the "as-is" system is not emphasized. The analyst needs strong critical thinking skills to be able to move users "from here to there". He must have critical thinking ability to recognize strengths and weaknesses and then recast an idea in an improved form and also understand issues so as to develop new business processes. These skills are needed to thoroughly examine the results of requirements gathering to identify business requirements and to translate those requirements into a concept for a new system. Doing such comprehensive systems analysis would help greatly in coming with good software architecture and corresponding reusable software component to use in building the software of choice.

DISCUSSION

The traditional structured waterfall approach to software development is good. It is the foundation for most of the newer software development methodologies. The waterfall approach is easy to understand and easy to use. It provides structure to inexperienced analyst or staff (Robertson and Bednar, 2006). It helps to set requirements and works well when quality is more important than cost or schedule.

However, this approach has many weaknesses which make it difficult to rely on it for meeting up with the software demand of 21st century. Some of this weaknesses includes the fact that thee waterfall approach does not make allowance for the change of defined requirements as the project progresses. This may make the software not to fully meet the requirement of users. It is also time consuming and is only very suitable when the system requirement is not vague but specific and is not subject to change. More recent methodologies like the Rapid Application Development (RAD) address some of these weaknesses. The RAD approach reduces cycle time and brings about improved productivity with fewer people and less cost. RAD minimizes the risk of not achieving customer satisfaction and business needs.

The RAD approach on its own is hard to use with legacy system (Robertson and Bednar, 2006). It is best fit for systems that can be modularized and best when functionality is delivered in increment. It does not make room for reuse. Object oriented methodology was later introduced to help represent data, issues things and people with objects. Object oriented was to make room for reuse. With time it became clear that objected oriented could not make room for extensive reuse, as originally suggested (Selecting a Development Approach, 2008). It was discovered that a single object-oriented were too detailed and specific and often had to be bound with an application at compile time. This made marketing object as reusable components difficult (Baoming, 2000). Component-based software engineering (CBSE) introduced. CBSE brought about flexibility and leads to design that make clear separation between the stable parts of the system from the specification of their composition. Component-based software engineering uses software integrated circuit which are pieces of reusable and reliable software which can be plugged or unplugged to form a complete and reliable software system just like the hardware integrated circuit are assembled together to form a hardware system.

The components to be used must undergo some standardization processes so as to check out for some characteristics that will qualify them for usage. Standardization here means that a component used in CBSE process has to conform to some standardized component model. This model may define component interface, component metadata, documentation, composition and deployment. According to (Bellur, 2009), Some of these characteristic include:

- **Independent**: For any component to qualify for usage in CBSE, it should be possible to compose and deploy such components without having to use other specific components. Every component that could be used must be able to interact with other components through publicly defined interfaces.
- **Deployable**: A component has to be self-contained and must be able to operate as a stand alone entity on some component platform that implements the component model.
- **Documented**: Usable component have to be fully documented so that potential users of the component can decide whether or not they meet their needs.

The structure and semantics of all component interfaces should be specified as well.

Why Should Software Engineers Move to CBSE

Based on our discussion so far, it is obvious that there are number of good reasons why software developers should consider adopting component-based approach in their work. The first issue here is that software system in our days are becoming larger and more complex and in turn customers are demanding more dependable software that is developed more quickly and that is less expensive. CBSE is trying to fulfill the customers demand by bringing about better management of complexity, quicker development time, and improved quality. CBSE also give room for extensive reuse of software components. Users today also expect software to be easy to maintain in order to decrease maintenance cost and operating expenses. CBSE will help to effectively handle those problems.

Some Specific Benefits of CBSE

According to (Pole, 1999), component-based software development offers among others the under listed advantages over the conventional software development methodologies:

- CBSE brings about significant reduction in the development cost and time. This is made possible by assembling existing set of reusable software component to build a new system instead of building from the scratch.
- CBSE offers more reliable software. This is basically because reusable components have been tested and therefore their quality can be assured.
- CBSE brings about improving the maintainability of enterprise software systems by allowing new, high-quality components to replace old ones.
- CBSE makes it easier to manage software development. Component partitioning enables parallel development thereby allowing several organizations to be involved in development of larger and more complex software.
- CBSE brings about enhancing the quality of enterprise software systems. Usually domain experts develop components, and then software engineers who specialize in component software engineering assemble those components into enterprise software systems.

Disadvantages of CBSE

There is no gain saying that CBSE offers a lot of advantages to both software developers and software users. Some of these advantages were highlighted above.
According to (Yang, 1997), CBSE has among other the following challenges confronting its usage.

- Component trustworthiness: This is the challenge of users trusting component with no available source code.
- Emergent property prediction: The challenge of being able to predict the emergent properties of component composition.
- Time and effort it takes to develop reusable components.
- Reusable component can actually be very expensive
- Conflict between usability and reusability of components.

Conclusion

This paper discussed the imperatives of adopting component-based software engineering (CBSE) as a pragmatic approach to software development in 21st century in other to meet up with the demands of software in our day. In recent time, software size and complexity has grown to an advanced stage. Traditional software development methodologies for building software from scratch seem to have become more and more inefficient in terms of productivity, cost and delivery time. Software is becoming more and more an integrated part of the society and different kinds of devices which increase effectiveness and efficiency in diverse human endeavors now depend on software to operate. In order to meet up with the requirements of quality, deal with complexity, bring about flexibility, introduce the concept of reuse and also deliver software on record time, new software engineering paradigm like CBSE should be adopted.

Object oriented based development brought about reduction of development time and resources required to maintain existing systems, increased code reuse and provided a competitive advantage to organizations using it. However, it seems to have not actually led to extensive reuse which is needed dearly at this time. Component based development builds on the object oriented but gives more abstract view of the software systems than the object oriented. Components are totally encapsulated, they strictly prevents access to their internals while in object oriented knowledge about objects' inner working is necessary. Components exist at different sizes varying from single objects inside a library to whole applications. Component technologies therefore allow the representation of the architecture of software systems as reusable entities. As software engineers, we have seen the enormous challenges facing software developers and users. We have noticed that there is a great need to find a means of solving the problem of flexibility, usability, robustness. We therefore submit that component-based software engineering if adopted by software engineers would help increase efficiency and effectiveness in software development. It would also help meet the always changing requirements of software users in this century.

REFERENCES

Alan, D. and Barbara, W. 2003. Systems Analysis and Design. Second Edition. Hermitage Publishing services. Printed and bound by Von Hoffiman pres, Inc

Baoming, S. 2000. A model for Component-based Software. A Thesis Submitted to the Faculty of Computer Science, Dalhousie University-DalTech, Halifax, Nova Scotia in partial fulfillment of the Requirement for the degree of master of computer Science

Bell, T. and Thayer 1996

Bellur. U. 2009. The Role of Components and Standards in Software Reuse. Available at: http://www.Objs.com/ workshops/ws9801/papers.pdf/

Benneth, S., McRobb, S. 2002. Objected Oriented Systems Analysis and Design using UML, 2nd edition, London: McGraw-Hill, 2002

Booch, G. 1994. Object Oriented Analysis and Design with applications, 2nd edition. The Benjamin/Cummins Publishing Company, redwood City.CA.

Brown, D. 1997. Object –oriented Analysis: objects in plain English, New York: John Wiley, 1997

Contributor, M. 2006. Rapid Application Developmet Methodology Note. Retrieved online from http://articles. techrepublic.com.com/5100-10878_11-6118423.html

Dayan, H. *et al*. 1999. Legacy Software Systems Issues. Progress and challenges. Technical Report TR-74.165-k, April. Available at http;//www.cas.ibm.com/Toronto/ publications/TR-74/Legacy.html.

Debayan, B. 2001. Component-Based Development: *Application in software Engineering* by Indian Statistical Institute.

Jeffrey, L. *et al*. 2000. Systems Analysis and Design Methods 5th edition. Published by McGraw-Hill Higher Education (A division of the McGraw Hill Companies) 1221 Avenue, New York, NY, 10020. Also available www.mhhe.com/whitten

Lewis, T. *et al*. 1995. Object Oriented Application Framework, Manning Publications Co. Retrieved Online.

Micheal, R. *et al*. 2001. Component-based Software Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes. The Chinese university of Hong Kong Productivity Council.

Mohammad, S. 2001. An Analysis of component-based development-maximizing reuse of existing software. A thesis in partial fulfillments of the requirement for the award of masters of Science (MSc) degree in information technology. Retrieved online from: http://www.peterindia.net/SoftComponentsLinks.html

Nabil, M. and Govardhan, A. 2010. *Comparison between five Models of Software engineering*. International Journal of Computer Science (IJCSI), Vol. &, Issue 5, September 2010. ISSN: 1694-0814: Online at www.IJCSI.org.

Nilesh, P. 2005. The Waterfall Model Explained. Retrieved online from http/www.buzzle.com

Osuagwu, O. 2005. Software Engineering. A pragmatic and Technical Perspective. Second Edition. Oliverson Industrial Publishing House (OIPH). A division of Hi-Technolohy Concepts (WA) Ltd, 9/14 Mbonu Ojike Street, Owerri, Imo State Nigeria.

Otto, V., Olesen, R. and Korsaa, M. 2000. Software Development. A practical view. Delta, Department of Software engineering. Retrieved online at: www.delta.dk/iterativ

Pole. T. 1999. An Empirical Study of Representation Methods for- Reusable Software component, IEEE transactions on Software Engineering. *International Workshop on Component Based Software Engineering, Los Angeles, USA*.

Rob, M. A. 2007. Dilemma between the structured and Object-oriented approaches to Systems Analysis and design. School of Business, University of Houston-clear Lake, 2700 Bay Area Boulevard Houston, Texas. Online at http://www.msmc.la.edu/include/learning_resources/

Robertson, D. and Bednar, J. 2006. Development Methodologies: Waterfall Model. Available at : from :http:// www.inf.ed.ac.uk/ teaching/ courses/ seoc2/ 2004_ 2005/slides/methodologies_4up.pdf

Robertson, D. and Bednar, J.A. 2006. Development Methodologies. Waterfall Model. Retrieved from http://www.inf.ed.ac.uk/teaching/courses/seoc2/2004_2005 /slides/methodologies_4up.pdf

Roger, P. 2001. Software Engineering; A practitioner's Approach, fifth Edition

Selecting a Development Approach. Prepared by Department of Health and Human Services Centre for Medicare and Medicated services, February 17, 2005 .Revalidated March 27, 2008.

Software Requirements: Are they really a problem? *Proceedings of the 2nd international conference on software engineering*. IEEE computer society.

Stevens, P. and Pooley, R. 2000. Software Engineering with Objects and Components. Harlow: Addison-Wesley, 2000.

Uwaje, C. 2012. The Future of Software Development in Nigeria. *A communiqué on Institute of Software Practitioners of Nigeria- National Software Conference and competition*-Tinapa, Calabar, Cross-River start-Federal Republic of Nigeria.

Yang, H. *et al*. 1997. A Design Framework for system *Re-engineering, in Proceedings of Asia Pacific and International Computer Conference pp 230-250. Available at:* http://www.inf.ed.ac.uk/internationalcconf/comp/SE

****