IASET: Journal of Computer Science and Engineering (IASET: JCSE)
ISSN(P): Applied; ISSN(E): Applied
Vol. 1, Issue 2, Jul - Dec 2016; 19-24

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

A CRITICAL ANALYSIS OF COMPONENT-BASED SOFTWARE ENGINEERING

SIKIRU OLANREWAJU SUBAIRU & JOHN ALHASSAN

Department of Cyber Security Science, Federal University of Technology, MInna, Nigeria

ABSTRACT

Component-based software engineering is a solution phenomenon that came up as a result of high cost of developing new software; couple with high risk failure and delay time of delivery of finished software. This takes into consideration the right type of software architecture and then selecting the various adequate components from components repository, integrates them base on the new software requirements and a new software is born.

This paper critically looks into object orientation versus domain engineering, various sources of components such as re-usable codes, free-ware and free software (libre), open source software, usable code libraries on the internet, software portability and best practices for component base software development.

KEYWORDS: Component-Based Software Engineering

INTRODUCTION

Component-based software engineering surfaces as a result of failure of object-oriented development to support reuse adequately and effectively. Many years analysis has demonstrated that object-oriented development is incapable to adapt to ever changing requirement of today application software. This is because it lacks software architecture capable to adapt to changes in current requirement, though it easily develop a model that shows vividly the domain problem.

Another shortcoming of object-oriented development is that there is no separation when it comes to computational and compositional aspect and this separation is highly needed to adapt to ever changing requirement; this formed the core of the component-base software development which make it flexible by distinguishing stable components from specification of their composition. Combining these components lead to evolving a new solution software that is rapid in development, quick availability to the market and easily adaptable to changes in requirement. This is rather another advantage by investing in changes of those areas of the component-based solution than embarking on another release.

What the software engineer needs to do is to establish the requirement of the new software, come up with the architectural design and examine each requirement to see which part of it is amenable to composition rather than embarking on new construction. In order to establish these facts, the following questions are important for each of the requirement:-

- Availability of commercial off the shelf (COTS) components capable of implementing the requirement
- Availability of internally reusable components to implement the requirement.
- Availability of component interface that is compatible with the architecture of the system to be built.

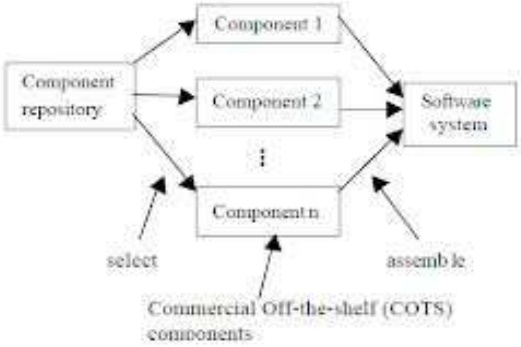

Figure 1: Component-Based Software Development

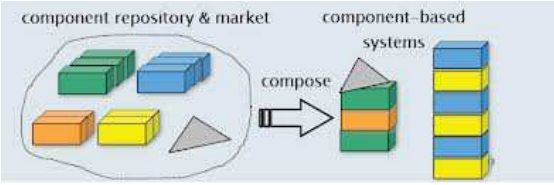

Figure 2: Simple Model of Component Repository

Sometimes after an analytical examination of some requirements, some may be found to non-changeable or adaptable or even non removable. A new component needs to be engineered using conventional or object- oriented software engineering process. This component must meet the requirement of the components. Series of activities such as component qualification, component adaptation, component composition and component update will have to take place as the case may be.

Domain Engineering

This comprises of three basics activities such as analysis, construction and dissemination. According to Paul Clements; domain engineering is 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 advantages of those components.

Domain Analysis Process

This process is applied to any paradigm in software engineering as it applied to both conventional as well as object-oriented software development. Applying this process may follow these steps:-

- Detailed the domain to be investigated
- Classify the items extracted from the domain
- Take a sample from the application
- Critically examine each application in the sample
- Build an analysis model for the object.

Reusable Codes

This approach has long been sought for, until 1968 when Douglas Mcllroy of bell laboratories proposed that software industry can be based upon reusable component. By so doing, a code that was used in one program at one time

can be reused in another program at later time; thus saving time and energy by reducing redundancy.

This practice has to be standardising through a software procedure line otherwise known as domain engineering. Great caution has to be taken so as to avoid codes duplication which is usually cause by cut and paste programming.

Communication interface such as call must be defined for the newly written codes to interact or used the already existing codes.

Type of Reuse

Base on certain factors and motivation, reuse can be one of the following:

- **Opportunistic:-** This occurs when software engineer realises at the beginning of a project that there exists a component that can be reuse
- **Planned: -** This when a software engineers design components to be reuse in future projects.
- **Internal Reuse:** - This is when software engineers use their own components. This decision might be business based so to control components critical to the project.
- **External Reuse:** - Software engineers may choose to use third-party license components. License third-party components cost less than to develop internally component, but it has demerits in terms of time to search for it, learn it and integrate the component.

Freeware and Free Software (LIBRE)

Freeware was coined by Andrew Fluegelman and it has often being used in 1980s and 1990s for program released only as executables with the source code not available. Freeware do not have a definite definition or it is loosely defined but according to the free software foundation, it must be distinguishable from free software or libre software.

Free software, unlike freeware has no restriction to its usage. The user can modify it, reengineered, run it, adapt it to their needs with or without changes. This phenomenon makes it a source for components during component-based software engineering. This definition for free software was defined by Richard Stallman and its definition still holds today. The followings are the definitions:

- **Freedom 0:-** The freedom to run the program for any purpose.
- **Freedom 1:** The freedom to study how the program works, and change it to make it do what you wish.
- **Freedom 2:-** The freedom to redistribute copies, so you can help your neighbour.
- **Freedom 3:-** The freedom to improve the program, and release your improvements (and modified versions in general) to the public, so that the whole community benefits.

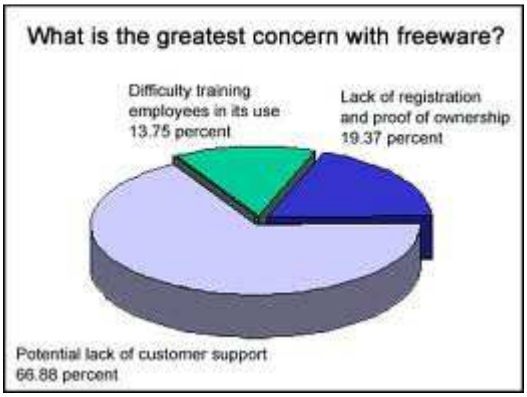

Figure 3: Chart Showing Freeware Peculiar Problem

Open Source Software

Open source software has its code being available and accessible to the general public and the author has granted license to any user to study its code, change it to suit one need and distribute it either freely or any other way. This is usually developed publicly by collaboration by volunteer or software enthusiast developer.

The license granted is usually for free distribution and for further development and application. An example of open source license is the GNU General Pubic License (GPL).

Figure 4: Showing the Icon for Open Source and other Open Source Softwares

Open source distribution makes source codes available and accessible while their licenses allow the author to define how it is going to be accessed.

Software Portability

This concept in software development greatly reduces cost, since it allow the usability of the same software in different environment platform.

Abstraction is a basic requirement for portability and it occurs between the application logic and system interface. Software portability involves the following strategies:

- Transferring, installed program files to another computer of basically the same architecture.
- Reinstalling a program from distribution files on another computer of similar architecture.
- Building executable program for different platforms from source code; otherwise known as porting.

The following need to be considered for software portability:-

- Similarity of system
- Different operating system, but similar processor
- Different processor

Reusable Code Libraries on Internet

A typical example of using an already code (reuse) is by using software library technique. Information among different well known format, how to access the external files, storage, manipulation and so on could be store in the software library. Developers do not need to re-invent codes when he or she can make use of so many codes in the library to complete its project.

Library implementation have its benefit but its demerit is in the area of inability to get details of codes which may ultimately affects desired output or performance, and the time and cost of findings, learning and configuration.

Best Practices for Component-Based Software Engineering

Commencing a project, software engineer needs to establish the requirement of the new software, come up with the architectural design and examine each requirement to see which part of it is amenable to composition rather than embarking on new construction.

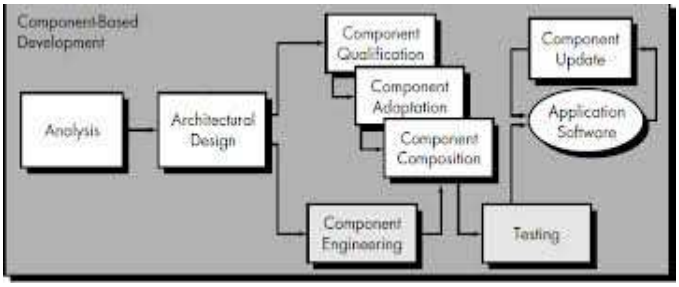

Figure 5: An Overview of Best Practices of Component-Based Software Deveopment Process

Then, the software engineer has to take into consideration the following:

- **Component Qualification:** - System requirement and architecture define the component that will require. The process of discovery and analysis must be used to qualify each component suitability.
- **Component Adaptation:** - These components must be adapted to meet the need of the architecture or discarded and replaced by other fitted one.
- **Component Composition:** - This is another area software engineer has to put into right perspective as it dictates the composition of the end product.
- **Component Update:** - Basically, when systems are implemented with commercial-off the shelf (COTS) component, update is complicated as the owner of those component may be out of organisational control, so a kind of agreement may be reached.

REFERENCES

1. Champ man, M; Van der Merwe, Alta (2008) Saicsit 2008, South Africa
2. Colombo, F. (2011), "It's not just reuse" - http://sharednow.blogspot.com/2011/05/its-not-just-reuse.html accessed on 13th June, 2013.
3. Wallace, Bruce (May 19, 2010). "A hole for every component, and every component in its hole"
4. Raphael Gfeller (December 9, 2008). "Upgrading of component-based application"
5. McConnell, Steve; Rapid Development: Taming Wild Software Schedules, (1996), ISBN 978-1-55615-900
6. Dinu Madau 2008 "An architecture for designing reusable embedded systems software"
7. Verts, William T. (2008-01-13). "Open source software"
8. James D Mooney, Developing portable software, West virginal University WV26506 USA
9. Roger S. Pressman PhD, Software Engineering, A Practitioner Approach, 5th edition
10. Arvinder Kaur, Kulvinder Singh Mann, Component base software engineering, *International Journal of Computer Volume 2 – No.1, May 2010*