Separation of Concerns "Component Based Software Engineering"

Jasmeet Kaur

Abstract: **Component-based software engineering (CBSE)(also known as component-based development (CBD)) is a branch of software engineering that emphasizes the separation of concerns in respect of the wide-ranging functionality available throughout a given software system. Software community faces a major challenge that is raised by fast growing demand for rapid and cost-effective development and maintenance of large scale and complex software systems. To overcome the challenge, the new trend is to adopt separation of concerns based software engineering. The key difference between CBSE and traditional software engineering is that CBSE views a software system as a set of off-the-shelf components integrated within appropriate software architecture. CBSE promotes large-scale reuse, as it focuses on building software on building software systems by assembling off-the-shell components rather than implementing the entire system from scratch. CBSE also emphasis on selection and creation of software architecture that allow systems to achieve their quality requirements.**

1. INTRODUCTION

For rapid assembly of flexible software systems Component-based Software Engineering (CBSE) has emerged as a new technology. CBSE combines elements of software architecture, modular software design, software verification, configuration and deployment.

Component-based software engineering (CBSE) is an approach to software development that relies on software reuse. It emerged from the failure of object-oriented development to support effective reuse. Components are more abstract than object classes and can be considered to be stand-alone service providers. In CBD, software systems are built as an assembly of components already developed and prepared for integration. The many advantages of the CBD approach include effective management of complexity, reduced time to market, increased productivity, a greater degree of consistency, and a wider range of usability. Any significant effort to implement software reuse must include a measurement program which allows the project management to assess how well the project is doing with respect to developing and using reusable software. The following key points which a reuse measurement program should address.

l. The percentage of a system that is made up of reusable components

2. The changes that were made to a reusable component in order to reuse it

3. The effort that was required to: locate, understand, adapt, and integrate a reusable component

Some of the requirements pertaining to an effective reuse technique are:

1. If reuse helps on a specific development stage, but has a negative overall effect, then it should not be applied. For instance, if by reusing a particular software architecture the development group is drawn towards a concept that they are not familiar with (for example the use of OO techniques), then the effects on the overall development may be extremely harmful.

2. It is a huge draw back if one has to spend a lot of effort in order to be able to reuse a specific artifact. "It must be easier to reuse artifacts then to develop them from scratch." Otherwise reuse is just not worth it, one would be better off by simply building custom code.

3. If locating an appropriate reusable artifact takes longer than building another one with the same quality level, then one should just build it from scratch.

Components are in general considered as black boxes with little or no information easily accessible. The information needed from the software components relates to their quality attributes. If the components' attributes are not known, the attributes of the system of which they are part of is even more difficult to reason about. Quality attributes are often referred to as properties, non-functional or extra-functional attributes because they relate to the quality of the component and not explicitly to the component functionality. The quality attributes describe the characteristics of a component. The properties of a component are the concrete accessible values that represent the characteristics. A component can express its quality attributes in terms of its properties. Not all quality attributes can be expressed as properties at the component level some are better described at the application level.

It is important to reason about the quality of the product and the impact of functionality added due to a change in the system. Thus steps must be taken to ensure that one

---

1. Lecturer, G.H Raisoni College of Engineering and Management,Pune
kaur.jrb@gmail.com,
jasmeet.kaur@raisoni.net

can easily predict the quality attributes of a component based system.

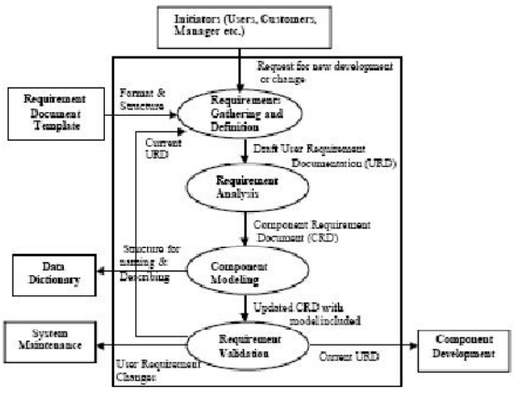
2. METHODOLOGY

By adapting separation of concerns, Component-based software systems are developed selecting various components and assembling them together rather than programming an overall system from scratch, thus the life cycle of component-based software systems is different from that of the traditional software systems. The life cycle of component-based software systems can be summarized as follows

- Requirements analysis
- Software architecture selection, construction, analysis, and evaluation
- Component identification and customization
- System integration
- System testing
- Software maintenance.

2.1 Requirements analysis

Component requirement analysis is the process of discovering, understanding, documenting, validating and managing the requirements for component.



Fig, 1.1 Requirements analysis

2.2 Software Architecture Selection

The objective of this phase is to select the architecture of the component according to the user requirements .In this we will construct the component and that component can be Component off the shell (COTS) component.

2.3 Identification

Identification of the component can be done by selecting the right components in accordance to the requirement for both functionality and reliability

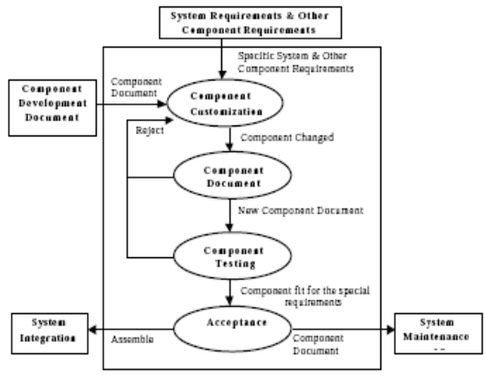


Fig.1.2 Identification of the component

2.4 Component Customization

It is the process that involves
1) Modifying the component for specific requirement
2) Doing necessary changes to run the component on special platform
3) Upgrading the specific component to get a better performance and higher quality.

2.5 System Integration

It is the process of assembling components selected into a whole system under the designed system architecture. The objective of system integration is the final system Composed of several components.

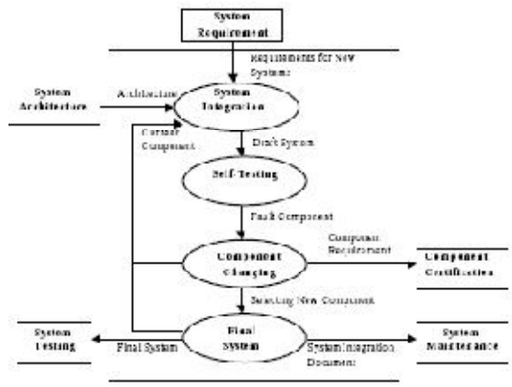


Fig 1.3 System Integration of the component

2.6 System Testing

System testing is the process of evaluating a system to
1) Confirm that the system satisfies the specified requirements
2) Identify and correct the defects in system in system implementation
The objective of system testing is the final system integrated by components selected in accordance to the system requirements

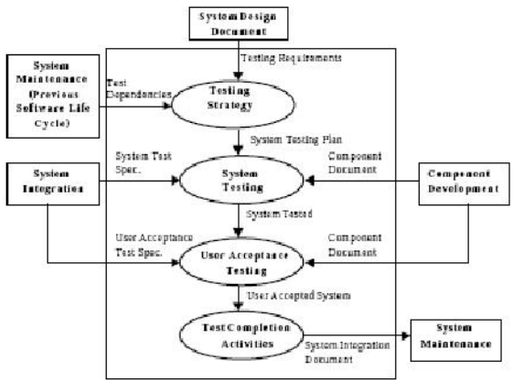


Fig 1.4 System testing of the components

2.7 System Maintenance

It is the process of providing service and maintenance activities needed to use the software effectively after it has been delivered

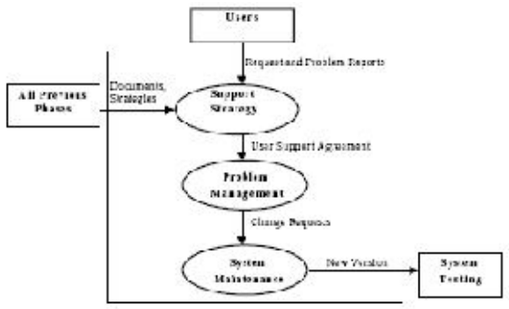


Fig 1.5 showing the system maintenance

3. CONCLUSIONS

CBSE is a reuse-based approach to define and implement loosely coupled components into systems. A component is a software unit whose functionality and dependencies are completely defined by its interfaces. A component model defines a set of standards that component providers and composers should follow. Components are implementation of interfaces (syntactically and semantically). From a component, the designer can extract the consistent provided interfaces and required interfaces to provide publications of the component.

4. RESULTS

CBSE can bring fundamental changes in the way software is developed. Through Component Based Development, the process of programming components gets separated from that of composing components to applications. The hope is that components can be plugged together as easily as Lego bricks without the need for writing source code. Thus, even domain experts with rather few technical skills can assemble applications. Furthermore, CBD is expected to have a strong impact on the quality of software development:
Due to the simplicity, the software development speeds up. There is no need to develop everything from scratch because one can buy third-party components. The shorter development time results in reduced costs. Components enhance the reliability of the software, as they are improved, tested and debugged over years. Aside from this, the composer can choose and combine the best components of different vendors.
The extensibility and resolvability of software systems is improved, because components can flexibly be substituted by another component that satisfies the requirements

5. REFERENCES

[1] Component-Based Software Engineering-the Need to Link Methods and their Theories .He Jifeng, Xiaoshan Li and Zhiming Liu
[2]Component-Based Software Engineering:
Technologies, Development Frameworks, and Quality Assurance Schemes Xia Cai, Michael R. Lyu, Kam-Fai Wong The Chinese University of Hong Kong Hong Kong Productivity Council {xcai@cse, lyu@cse, kfwong@se}.cuhk.edu.hk
[3] Component-Based Software Engineering, In Pervasive Computing Environments,David Garlan School of Computer Science
Carnegie Mellon University, 5000 Forbes Avenue,Pittsburgh, PA 15213 garlan@cs.cmu.edu
[4]Introduction to component based software engineering Ivica Crnkovic Mälardalen University, Department of Computer Engineering,Sweden,ivica.crnkovic@mdh.se
[5] Software architecture classification for estimating the cost of COTS integration.Daniil Yakimovich University of Maryland,Computer Science Department,University of Maryland College Park, MD 20742, USA,
[6] Rainer Niekamp. Software Component Architecture Gestión de Congresos - CIMNE/Institute for Scientific Computing, TU Braunschweig. p. 4. Retrieved 2011-07-29. "The modern concept of a software component largely defined by Brad Cox of Stepstone,