International Journal of Computer Applications (0975 – 8887)
Volume 63– No.21, February 2013

Elicit – A New Component based Software Development Model

Lata Nautiyal
Assistant Professor
Graphic Era University
Dehradun - India

Neena Gupta
Assistant Professor
Kanya Gurukul Campus,
Dehradun- India

ABSTRACT

This is an era of embryonic software development where there is loads of pressure not only on developers but also on software development organizations in order to stumble on unswerving, fast and lucrative methods in software production. Therefore, to fulfill the need, Component-Based Development (CBD) has been broadly used in software development as it enhances reusability, flexibility, and reduces the cost as well as time.

In this paper, we are proposing a life cycle model commonly encountered in component based development methodologies. This model is divided into five stages or phases. Each phase describes its sub segments and activities necessary to develop Component Based software in a short span of time, effectively and efficiently. This model emphasizes not only on the development architecture but also focuses on validation of the integration of components, to accommodate client needs and requirements. The proposed model can be used for developing a process for producing Component-Based systems.

General Terms

Software Engineering

Keywords

Commercial Off the Shelf (COTS), Software Life Cycle Model, Software Reusability, Component-Based Development (CBD)

1. INTRODUCTION

Now - a – days, Component-Based Development (CBD) is the most brisk growing trend in IT industry. A component is an encapsulated unit of functionality with a well-defined interface that allows it to connect to other components, and be independently deployed. Moreover, the assembling components are the ones which define Component-based applications.

The foremost benefits associated with component-based technologies include: reduced system development cost and time, enhanced quality, and reduced system evolution and maintenance cost. Over the past decade, standard component-based specifications have been developed in lieu of which the importance of CBD has grown rapidly in the embedded system industry.

It has been claimed that the component-based software development endorses reusability, improves software quality and increases software engineers' productivity. A component is a self-contained piece of software that provides clear functionality, has open interfaces and offers plug-and-play services.

2. Literature Review

The brainstorm gained its real momentum after COM+ [1] from Microsoft, Enterprise JavaBeans [2] from SUN, and IBM Component Broker [3] and CORBA [4] have made their way among mainstream software technologies [5]. Additionally, incremental delivery of software features or platforms that comprise a software product line is expected to be at the forefront of software development in the next few years, therefore component-based software engineering has broad implications for how software engineers acquire, build and maintain software systems [6].

Thus, we should see dramatic changes in designers' primary roles and required skills for software development in the near future.

A Software Life Cycle Model is an expressive and pictorial representation of all different stages or phases of the software process. Software development life cycle (SDLC) model describes the segments of the software cycle. [7]

The Twin Peaks model [8] also suggests for a concurrent, iterative development of requirements and architecture during software development. It presents a partial and simplified way to develop the software.

In X Model, the processes are started by requirement engineering and requirement specification. The chief characteristic of this software life cycle model is reusability in which software is developed by building reusable components for software development and from reusable and testable components. In software development, it uses two main approaches, develop software component for reuse and software development with or without modification in reusable component. [9]

The Y Software Life Cycle Model describes software reusability during CBSD. The Y Shape of the model considers iteration and overlapping. Although the main phases may overlap each other and iteration is allowed, the planned phases are: domain engineering, frame working, assembly, archiving, system analysis, design, implementation, testing, deployment and maintenance. [10]

Knot Model emphasizes on reusability considering risk analysis and feedback in each and every phase. This model may be best suited for medium or larger complex system's development. It is based on three states of the component [11]:

The Elite Life Cycle Model (ELCM) is an emerging lifecycle for the development of new product using component based technology. This model describes a general process of Software development with the help of in built components. [12].

3. The Elicit – Proposed Model

The Elicit (See figure 1) CBSE life cycle model has been proposed as a feasible substitute to address software reusability throughout component-based software fabrication. The formation of software is characterized by revolutionize and unsteadiness, hence the diagrammatic representation of the Elicit model considers overlapping and iteration where apposite. Although the main phases may superimpose each other and iteration is allowed yet the planned phases are: domain engineering, frame working, assembly, archiving, system analysis, design, implementation, testing, deployment and maintenance.

3.1.1 Phases of ELICIT Model

3.1.1.1 Assess Prerequisite

This phase analyze requirements in a broad manner. This stage is completed when general lines are agreed upon between analysts and users (See figure 2). It consists of three important elements.

a) Client Interface:

b) Endeavor Depiction:

c) Domain Analysis:

a) Client Interface:

The exceedingly first stage of the project commencement is the client interface which includes the following.

1. Communication with the consumer
2. Identification of client's needs and requirements
3. Precedence and Importance of client's requirements
4. Review of client's requirement list
5. Fixing client's requirements.

b) Endeavor Depiction:

After fixing some or all requirements of the client, we must determine the overall planning and description of the project i.e. problem domain. This stage includes the basic and detailed descriptions of the project; at least a skeleton of the problem domain must be established.

c) Domain Analysis:

Domain Analysis is a process of analyzing an application domain in order to ascertain areas of cohesion and ways to describe it using a uniform vocabulary. Thus, domain engineering is an activity that should be carried out at the commencement of software specification if reuse is to be considered. As domain engineering can yield an initial set of vocabulary reflecting the main conceptual entities within an application domain, essential properties of that domain are captured and initial candidates for reusable components emerge.

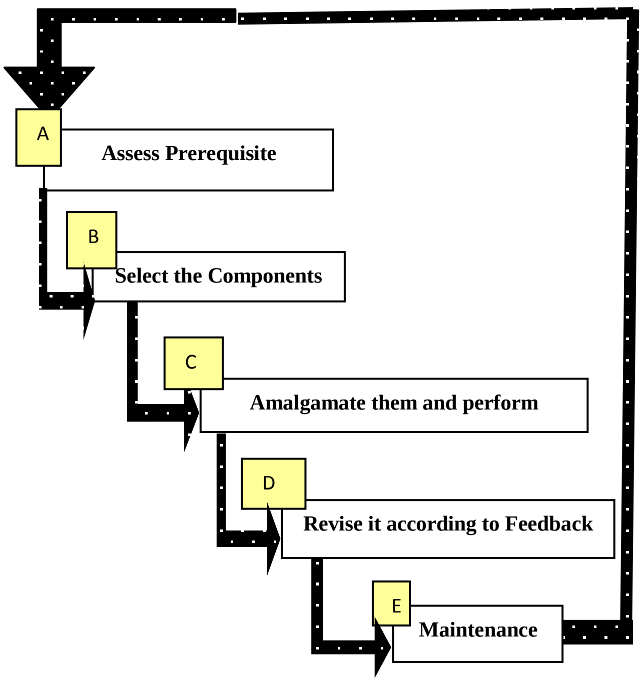

Fig 1. Elicit Model

User needs, software requirements, provided functionality, objectives and constraints of the system are very much of interest during the system analysis and domain engineering phases. Thus, it is important to understand the real-world application and an abstract model of that appliance to be depicted. Therefore, the boundary between system analysis and domain engineering may be at times seems fuzzy because identifying key abstractions in the application domain may be viewed as part of system analysis or domain engineering. Nevertheless, at this level, domain engineering is also concerned with the identification of potentially reusable components.

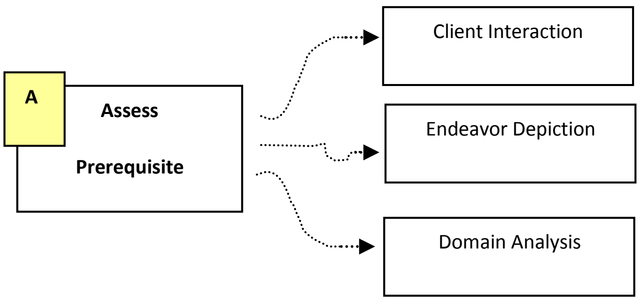

Fig 2. Constituents of the Access Prerequisite Phase

3.1.1.2 Select the Components

It focuses on selecting an assortment of reusable components or frameworks from specific application domains. (See figure 3) There are differences in the mechanisms used to accomplish reusability when different kinds of reusable components are involved. The most basic software components are often reused by composition, which can be seen as a process of building a piece of software from elementary self-contained components.

A framework could be viewed as a generic structure that provides a skeleton for producing software in a certain application domain. Frame working attempts to identify components and establish inter relationships perceived important within the application domain. Such identification of components may arise from the well known functionality common to that application domain, usually in the form of semantic relationships between components.

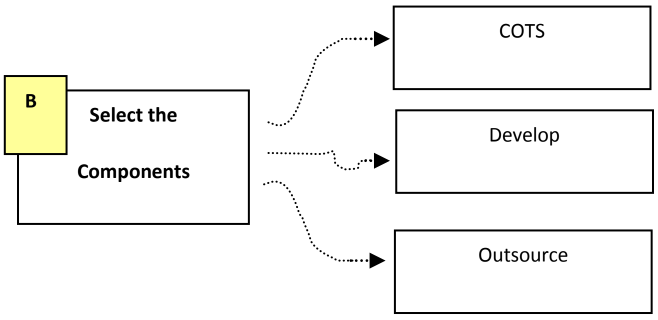

Fig 3. Constituents of the Select the Components Phase

Consider, for example, the application domain of airline reservation systems; typical entities of these systems are: seats, flights, crews and passengers; and interrelationships can be: reserve a seat, assign a crew to a flight, schedule a flight and so on.

So, there are important relationships among these entities, which can be organized into a framework according to their semantic meaning in that application domain. There are 3 main categorories of traditional components.

1. Outsourced:
2. Develop:
3. COTS (Commercially of the Shelf Components):

a) COTS (Commercially of the Shelf Components):

These are the components which are accessible in the Components repository, pre-developed and pre-tested. We can use these components without any modification or with some condition based alterations, according to their availability and client's requirement suitability.

b) Develop:

It is not always possible to ensure the availability of these COTS components. It may be possible that no component fits according to the problem domain or because of problems distinctiveness. In such cases we have to develop some components from scratch. It may also possible that development of new component is more feasible than using or modifying COTS components.

c) Outsourced:

At times it is not feasible to reuse COTS components or the development of new components since some risk factor may be involved, or we need some portion of a component for very short span of time. In such cases some third party components may be outsourced.

3.1.1.3 Amalgamate them and perform testing

Reusability not only involves reusing existing components in a new software system but also producing components meant for reuse. When a software system has been developed, the software engineer may realize that some components can be generalized for impending reuse. In this phase, it includes integration of New, Modified and out sourced components. This phase also includes testing at regression level to find the maximum numbers of errors. Basically, it focuses on selecting a collection of reusable components or frameworks from specific application domains (See figure 4). There are differences in the mechanisms used to achieve reusability when different kinds of reusable components are involved. The most basic software components are often reused by composition, which can be seen as a process of building a piece of software from elementary self-contained components; although reusability is naturally accomplished by reusing classes through inheritance during object-oriented development, in such case, it takes place by specialization and generalization of commonalities among classes.

After integrating these components we must have validate them according to the user needs or according to their architecture. It may be doable that individual components perform their intended task efficiently but their integration introduces some sort of undesired results. To validate these we must have to perform Integration Regression testing, which includes:

a) Integration of components based on System Architecture.

b) Perform Integration testing.

c) Perform Regression testing.

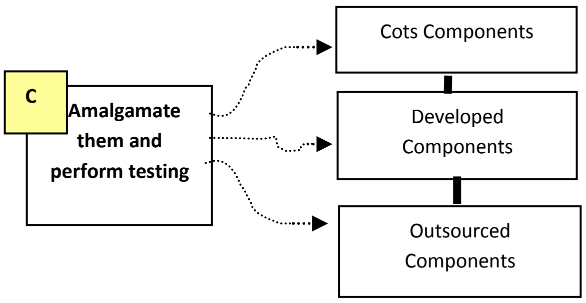

Fig 4. Constituents of the Integration Phase

Make necessary changes accordingly and perform another Regression testing with extended test cases.

Prioritize Minimize test cases and perform second level Regression testing.

Validate this integrated software with the user requirements and perform Regression testing accordingly.

3.1.1.4 Revise it according to feedback

The developer will now revise the system and return to the first step for deeper and more elaborate analysis.

The above cycle is repeated until the system reaches the final acceptance stage.

This is virtually an end of system development. Now the system is ready to be presented to the customer. Nevertheless, deployment involves more than putting the system into place, it is the time when users should be helped to understand and feel comfortable with the software. If deployment is not successful, users will not make the most of the system and may be unhappy with its performance. In either case, users will not be as productive or effective as they could be and the care taken to build a high-quality system is put in jeopardy.

The two key issues to successful transfer from the developer to the user are documentation and training, which should be integrated with the software. As the system is developed, software engineers should plan and come up with aids that help users learn about the system, such as on-line help. Accompanying the system is documentation and manuals to which users refer for problem solving, trouble shooting or further information. The quality and type of documentation can be critical, not only to training, but also to the success of the system. Training for users and operators is based predominantly on major system functionality; there is no need to be aware of the system's internal operation. Therefore, system deployment should be considered with more care and professionalism than it has been usually dealt with.

In addition, product flexibility is the new anthem of the software marketplace and software family fulfils the promise of tailor-made systems that are delivered quickly, at low costs, built specifically for the needs of particular customers and market segment. This requires constant improvement, upgrading and releases of new versions of a software system that is preferably compatible with old versions.

3.1.1.5 Maintenance

Many software engineers wrongly assume that once a system is delivered their problems are over. A system life does not end with deployment. Software is normally subject to continuing changes after it is built, when it is operational. Thus the efforts turn now to the challenge of maintaining a continually evolving system. During software maintenance, changes are introduced to a software system. Such changes are not meant only for correcting errors occurred in the operational software; these changes may be also for improving, updating the system to anticipate future errors or adapting the system in response to a modification in the environment. Many software engineers wrongly assume that once a system is delivered their problems are over. A system life does not end with deployment. Software is normally subject to continuing changes after it is built, when it is operational. Thus the efforts turn now to the challenge of maintaining a continually evolving system. During software maintenance, changes are introduced to a software system. Such changes are not meant only for correcting errors occurred in the operational software; these changes may be also for improving, updating the system to anticipate future errors or adapting the system in response to a modification in the environment. Therefore, during the maintenance phase, software components may be accessed from, as well as new ones may be added to a reusable library of the concerned application domain.. After changes are introduced to the system, an updated release of the software is generated. Maintenance of software system does not only allow the software to evolve but also the reusable library concerning the existing systems expands during the maintenance of a legacy system.

4. CONCLUSION

All these different software life cycle models have their own advantages and disadvantages. In this paper, we have discussed a number of activity areas that form a life cycle framework for component-based software development. Furthermore, we have also proposed a component based software engineering life cycle perspective on selection and development concerns.

The Elicit model supports "development with reuse" through component assembly. Initially, the software engineer identifies potentially reusable components from existing reusable libraries. The components are then selected, adapted and reused through various mechanisms. At the end of software development, there may be many new reusable components that need to be verified, catalogued, classified and then stored into reusable libraries. The proposed Elicit model covers the likely phases of large software development and enforces software reusability along its phases. This model emphasizes that if reusability is not feasible then outsourcing or new development can take place, that is domain analysis and problem architecture must be identified and defined.

5. Acknowledgement

From the core of my heart, I am really thankful to the Management of Graphic Era University for always being supportive. I would also like to express my gratitude to the management of Gurukul Kangri University for providing such a commendable research oriented platform to us.

6. References

[1] Microsoft, 2004.COM+, http://www.microsoft.com/com/tech/complus.asp.

[2] SUN, 2004. Enterprise Java Beans, http://www.java.sun.com/products/ejb/index.html.

[3] IBM, 2004. Component Broker, http://www.software.ibm.com/ad/cb.

[4] Object Management Group, 2004. The Common Object Request Broker Architecture, http://www.omg.org.

[5] Wallnau, K. C., S.A. Hissam and R.C. Seacord,2002. Building Systems from Commercial Components. Addison-Wesley.

[6] Clements, P. and L. Northrop, 2002. Software Product Lines. Addison-Wesley.

[7] S. Cohen, D. Dori, U. de Haan, "A Software System Development Life Cycle Model for Improved Stakeholders Communication and Collaboration", International Journal of Computers, Communications & Control,Vol. V (2010), No. 1, pp. 20-41

[8] Royce, W.W., 1987. "Managing the development of large software systems". Proceedings of 9th IEEE International Conference on Software Engineering, pp: 328-338.

[9] Gill N. S. and Tomar P., "X Model: A New Component-Based Model", MR International Journal of Engineering and Technology, 2008, Vol. 1, No. 1 & 2, pp. 1-9.

[10] Luiz Fernando Capretz, " Y: A new Component-Based Software Life Cycle Model ", Journals of Computer Science1 (1) : pp.76-82.

[11] Rajender Singh Chhillar, Parveen Kajla, "A New Knot Model for Component Based Software Development", International Journal of Computer Science Issues Year: 2011 Vol: 8 Issue: 3 Pp.: 480-484

[12] Lata Nautiyal, Umesh Kumar Tiwari, Sushil Chandra Dimri, Shivani Bahuguna, "Elite: A New Component-Based Software Development Model", International Journal of Computer Technology & Applications, Vol 3, Issue 1, Jan 2012, pp 119-124