Abstract:

Component Based Software Engineering (CBSE) approach is based on the idea to develop software systems by selecting appropriate components and then to assemble them with a well-defined software architecture.

In this evolving era of Information Technology, there is lot of pressure on software research community to reduce development cost as well as development time. Though there are number of life cycle models available for traditional software development process. And now days where CBSE is using rapidly so there is a great need of a life cycle model which can define different phases for Component Based Software Engineering.

In this paper, we are suggesting a life cycle model which incorporate testing or verification as a continuous process is being proposed. We examine the various software life cycle models. We are proposing a software life cycle model for component based software development from the conventional models known as Elite Life Cycle Model (ELCM).

Keywords: Component Based Software, Life Cycle, Model, ELCM

1. Introduction

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 phases of the software cycle.[7] It also depicts the order in which these activities are to be commenced. A life cycle model maps the different activities performed on a software product from its inception to retirement. To develop a software product in a systematic and disciplined manner, the development team must identify a suitable life cycle model for the specific project and then follow it. The basic activities are included in all life cycle models, though the activities may be carried out in different orders in different life cycle models. The general basic model is shown below:

1.1 Requirement

Requirement analysis is the first phase in most of software life cycle models. It is process after a feasibility study has been performed. In the requirement analysis phase we decide user desire in terms of functionality, performance, behavior and interfacing. In this phase user of the system or customer and developer are closely interact to decide all these things.

1.2 Design

In the design phase we translate requirements into a representation of the software.

1.3 Implementation

In the implementation phase design is translated into a machine readable form.

1.4 Verification

Once code are prepare, program verification begins

1.5 Maintenance

Once a software system passes all the verification and validation, it is delivered to the user and enters in the maintenance cycle. Any modification made
to the system after initial delivery is part to this phase.

Figure (1) - Basic Model of Software Development.

2. Component Based Software Life Cycle Process Model

Various process models have been designed by a number of researchers so far for component based software development. Most common among them are studied and described briefly.

2.1 Rapid Application Development Model

The Rapid Application Deployment (RAD) model is a high-speed adaptation of the linear sequential model. The RAD (Rapid Application Development) Model achieves rapid development using a component based construction approach. It follows a sequence of evolutionary system integrations or prototypes that are reviewed by users. The development of each integrated delivery is restricted in time, usually 2 – 3 months. [4] The RAD model is an adaptation of the Waterfall Model which works well when a system can be modularized or be component based.

The RAD model relies on several teams developing each module or component in parallel thus reducing the overall life cycle of development.

This model is proposed when requirements and solutions can be modularized as independent system or software components, each of which can be developed by different teams. Then after it smaller components are developed, they are integrated to produce a large software system. [5] The RAD approach encompasses the following phases.

1. Business modeling: Information flow among business functions, the source of information, etc. Information flow among business functions is modeled in such a way that answers the following questions.
   - What information derives the business process?
   - What information is generated?
   - Who generates it?
   - Where does the information go?
   - Who processes it?

2. Data Modeling: Information flow is defined above is refined into a set of data objects that are needed to support the business. Attributes of each object are identified along with the relationships between the objects.

3. Process Modeling: The data objects defined in the data modeling phase are transformed to get processing descriptions for adding, modifying, deleting, or retrieving a data object.

4. Application generation: RAD assumes the use of fourth generation techniques and automated tools. This involves reuse of existing program components (when possible) or creation of reusable components.

5. Testing and turnover: since the RAD process emphasizes reuse, many program components have already been tested. This reduces overall testing time. Only new components need to be tested.

Figure (2) - The Rapid Application Development (RAD) Model
2.2 The X Model

In this X Model, the processes are started by requirement engineering and requirement specification. The main characteristic of this software life cycle model is reusability in which software is developed by building reusable components for software development and software development 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.[2]

![Figure (3) - The X Model](image)

The X model appears to be the best for large software development and has main focus on reusability but it does not depict risk analysis, feedback in various phases which is one of the reasons for the very success of Boehm’s Spiral Model. Moreover the X-shape represents overlapping and increases complexity.

2.3 The Y Model

The Y Software Life Cycle Model (Luiz, 2005) 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.[3]

![Figure (4) - The Y Model](image)

The reusability within this life cycle is smoother and more effective than within the traditional models because it integrates at its core, the concern for reuse and the mechanisms to achieve it. There is an upper layer that shows the component templates and a lower layer that consists of run-time objects that depicts the behavior of a particular software system. The development of a component should therefore be with generality and reuse in mind placing perhaps less emphasis on satisfying the specific needs of an application that is being developed.

2.4 The Knot Model

Knot Model emphasis 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 [6]

1) When a new component is not available in the repository, then it develops a new component for reuse and places it in the pool
2) When a component is partially available, then modifies it for reuse and places it in the pool.
3) When a component is available in the repository then reuses it during the new proposed system. An utmost care should be taken that each component is
created for reuse, so the component is not based on particular application’s specification but must carry general specifications.

In this model risk is resolved in early stages of each phase. This results in the reduction of cost and time and makes the software more reliable and efficient. Moreover feedback at the end of each phase results in further improvement and revised form of component. It also reduces the cost and time by better management as it resolves the conflicts, if any, during that phase.

3. Elite Life Cycle Model (ELCM) - The Proposed Mode

The proposed Elite Life Cycle Model (ELCM) is an emerging lifecycle for the development of new product using component based technology. This model has been proposed as a viable alternative to address software reusability during component-based software production. The main characteristic of this model is the emphasis on reusability during software development, evolution and the production of potentially reusable components that are meant to be useful in future software projects. Reusability implies the use of composition techniques during software development; this is achieved by initially selecting reusable components and assembling them, or by adapting the software to a point where it is possible to pick out components from a reusable library. This model proposes following phases.

3.1 Requirement

The Requirement phase involves carrying out enough business/application/system modeling to define a meaningful build scope. A build delivers a well-defined set of business functionalities that end-users can use to do real work.

One of the most difficult tasks is identifying the real problem of the existing system. Without clear understanding of the problem in the system, any further work done will lead to wastage of effort at a later stage.

In a process-centric information system, for example, a build may represent the realization of one or more end-to-end business processes. The scope of a build is not a random selection, but rather a logical selection that satisfies specific development objectives. This phase defines the user requirements, or what the user expects out of the system. This phase also sets the project boundaries, which define what parts of the system can be changed by the project and what parts are to remain without any change. This also includes a rough idea of the resource requirements for the project as well as the estimated start and completion dates for each phase and the number of persons expected to be involved in each phase. This phase will cover:

(a) Pinpointing the problems and problem domains,  
(b) Prioritize the problem (domain),  
(c) Setting proper system goals according the problem priority,  
(d) Determining the boundaries of the project by taking into consideration the limitations of the available resources.

3.2 Elaboration of Scope

In this phase the emphasis is on determining the illustrations of build requirements in the form of technical, behavioral and structural specifications. Here the clearly specified and specifically defined requirements are taken in account. Scope and
limitations of the build are identified and addressed. These defined requirements may be priority basis as made by analyst or may be based on end user suggestions. It may include:
(a) *Technical Specifications* of requirements individually with resource requirements and resource limitations.
(b) *Behavioral Specifications*, which include the specific functionality and predetermined behavior of the requirements.
(c) *Structural Specifications*, which focuses on the implementation outlines. They specify the methods and techniques which may require deploying the requirement.

### 3.3 Using existing component

Once a build scope is established, we need to decide which of the required components can be used (e.g., already exist in the organization or can be bought off-the-shelf) and which ones need to be developed. Both these phases have their mini life cycles:

* Reusing an existing component may require some adaptation. For example, the component interface might not be exactly what is required or some of the method behaviors may need alteration. This is achieved through adaptation, which involves wrapping the component with a thin layer of code that implements the required changes.

### 3.4 Modify Existing Components (if required)

Developing a new module from the scratch is always avoided in component based development. It may highly possible that some existing components may require some minor or major modifications to accommodate with other components. We can modify an existing component according to the Scope Specification. After required modification it is necessarily advisable that validates and verifies these modified components to fulfill the Scope specification limitations.

### 3.5 Development of New Component

Building a new component should always begin with defining the component interface. This represents a permanent contract between the component and other components. Once the interface is defined and the intent of each method is established, the component can be designed and implemented.

### 3.6 Integration of Components

With all the components for a build in place, the components are then integrated and tested. The Integration process must be done keeping Scope Specification and Design Specification in mind. The integration of Elite model is based on *Clustering Approach*. We must start integration with bottom level and progresses toward the Clusters. These different clusters will ultimately form the software.

Integration will require the writing of intermediate code that establishes the interaction between the components.

### 3.7 Testing of Components

After that all the components are tested thoroughly by using

*White Box Technique* and *Black Box Technique*

### 3.8 Release

Typically, this phase will be including, including releases as well as bug-fix. Considerable effort is expended in developing user-oriented documentation, training users, supporting users in their initial product use.

The primary objectives of the Release phase include:
* Achieving user satisfaction completely

This phase can range from being very simple to extremely complex, depending on the type of product. For example, a new release of an existing desktop product may be very simple, whereas replacing a nation's air-traffic control will be very complex.

### 3.9 Customer Evaluation

The evaluation environment may be the same as the development environment (for earlier builds that are not mature), or a pseudo live environment (for later builds that are sufficiently mature). The outcome of the evaluation influences the direction of subsequent builds.
The evaluation phase involves the answers to these questions:
- Is the user satisfied?
Are the actual resources expenditures versus planned expenditures still acceptable or not.

Figure (6) - Elite Life Cycle Model (ELCM).

4. Conclusions

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.

In the component based software system, component selection and integration play important role. These two are interrelated concepts. In this context we are proposing a life cycle model for component based software engineering called Elite Life Cycle Development Model (ELCM). This paper has provided a component based software engineering life cycle perspective on election and development concerns.

5. References