A Process Model for Software Architecture

A. Rama Mohan Reddy
Associate Professor

Dr. P Govindarajulu
Professor

Dr. M M Naidu
Professor

Department of Computer Science and Engineering
Sri Venkateswara University College of Engineering / Arts and Science
Sri Venkateswara University
TIRUPATI – 517 502. Andhra Pradesh, INDIA

ABSTRACT

Software development life cycle (SDLC) is a process model adopted and followed during the development of software.. Software Engineering encompasses software engineering process models, project planning, management, and Software Development Life Cycle activities. In this paper, we are proposing a software process model for architecture-based software development from the conventional models by taking spiral process model. This process model is coined as Software Architecture Development Life Cycle (SADLC).

Key words:

Software Architecture, Software Development Life Cycle, Components, connectors, configurations, Spiral model.

1. Introduction

Software systems come and go, through a series of phases or activities that starts from the Inception, Initial Development, Productive Operation, Upkeep, and Retirement. The process provides interaction between stakeholders and serves as the medium for communication, with each new round of the iteration eliciting more useful knowledge from the stakeholders. Building computer software is an iterative learning process, and the outcome, called Software A software process defines the approach that is taken as software is engineered [PAU93]. This paper examines a number of methods for software modelling how software systems are developed. It begins with related works and definitions of traditional software life cycle process models. These models that are in use that form as the basis for organizing a process model for software architecture

2. Related work

Many models explicitly used for the earliest projects for developing large software systems in the 1950's and 1960's [Hosier 1961, Royce 1970]. Since the 1960's many descriptions of the classic software development life cycle have appeared [Hosier 1961], [Royce 1970], [Boehm 1976], [Distaso 1980], [Scacchi 1984], and [Somerville 1999]. Royce [1970] began the formulation of the software life cycle using the familiar waterfall model, shown in **figure 1.**

2.1 The software life cycle model

A descriptive model describes the history of how a particular software system was developed [Curtis, Krasner, Iscoe, 1988]. Prescriptive models are used as guidelines or frameworks to organize and structure how software development activities should be performed, and in what order.

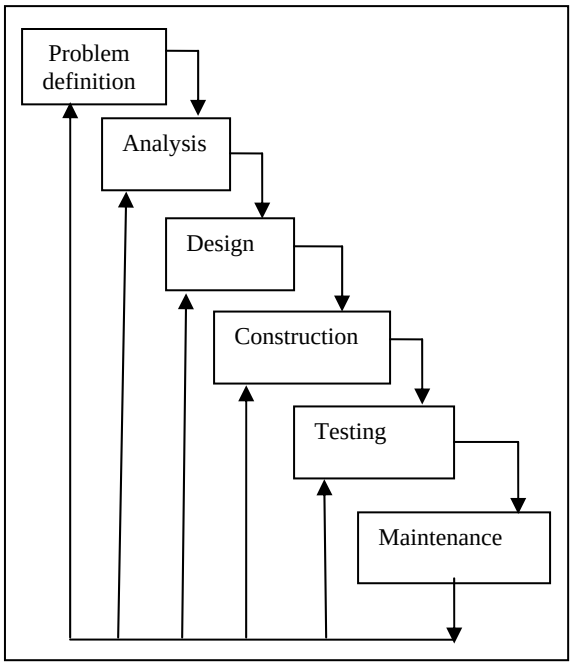

Figure 1. Conventional Software development life cycle (SDLC)

Prescriptive models are also used to package the development tasks and techniques for using a given set of software engineering tools or environment during a development project.

Descriptive life cycle models, characterize how particular software systems are actually developed in

Manuscript received April 5, 2007
Manuscript revised April 25, 2007

specific settings. These two characterizations suggest that there are varieties of purposes for articulating software life cycle models. These characterizations serve as guidelines to organize artifacts to be delivered the customer, tool and methods, and resource allocation and consumption [Boehm 1981].

To solve actual problems in an industry setting, a software engineer or a team of engineers must incorporate a development strategy that encompasses the process, Methods, and tools layers and the generic phases [Roger R.S Pressman, 2003]. This strategy is often referred to *as a process model or software engineering paradigm.*

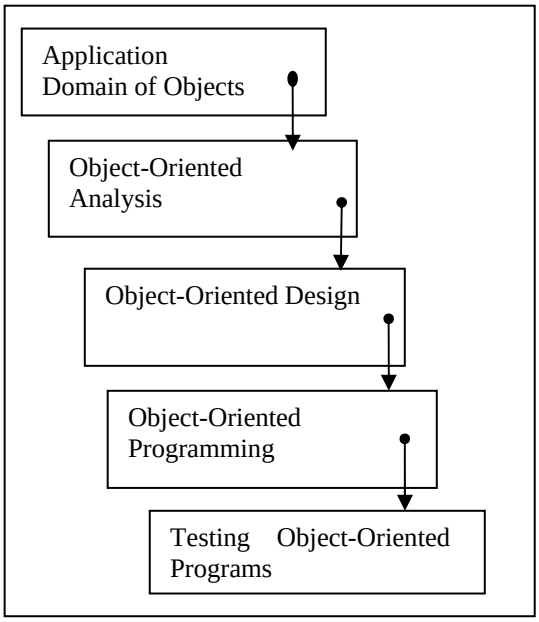

Figure 2 Software Development Life Cycles for Object-Oriented Software Development

All software development methods can be characterized as a problem-solving loop in which four distinct stages are encountered. The current state of affairs, problem definition identifies the specific problem to be solved, technical development solves the problem through the application of some technology, and solution integration delivers the results, documents, programs, data, new business function, new product, to those who requested the solution in the first place. Software process models often represent a networked sequence of activities, objects, transformations, and events that embody strategies for accomplishing software evolution. Software process networks can be viewed as representing multiple interconnected task chains [Kling 1982, Garg 1989]. Task chains can be employed to characterize either prescriptive or descriptive action sequences. Prescriptive task chains are idealized plans of what actions should be accomplished, and in what order. For example, as shown in **figure 2**, a task chain for the activity of object-oriented software designs.

Clearly, this sequence of actions could entail multiple iterations and non-procedural primitive action invocations in the course of incrementally progressing toward an object-oriented software design. The progressive steps of software evolution are often described as phases, such as requirements specification, preliminary design, and implementation.

2. 2 The linear sequential model

The linear sequential model, sometimes called the Classic life or the waterfall model, proposed by winsten Royce [Roy 70]. The linear sequential model suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing, and support as shown in **figure 3.**

.

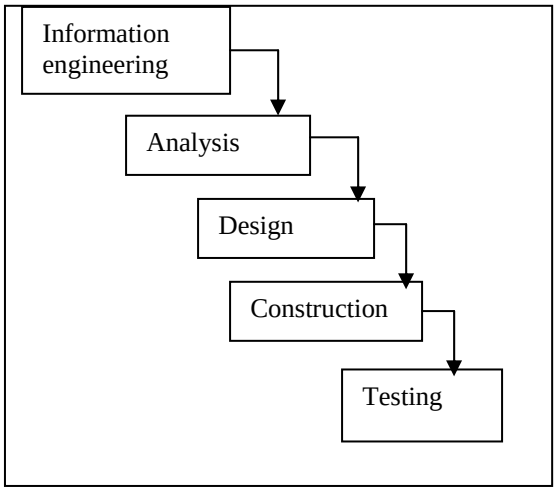

Figure 3. Software Development Life Cycle for Conventional software development (The Linear Sequential Model).

2.3 The Prototyping Model

A customer defines a set of general objectives for software but does not identify detailed input, processing, or output requirements. In these, and many other situations, a prototyping paradigm may offer the best approach

2. 4 The RAD Model

Rapid application development (RAD) is an incremental software development process model that emphasizes an extremely short development life cycle.

It is a component-based construction, but high speed linear sequential.

2. 5 Evolutionary Software Process Model

Software evolves over a period of time [GIL 88]. Evolutionary models are iterative. Iterative way of developing software is one of the modern software development processes. Evolutionary models enable software engineers to develop increasingly more complete versions of the software.

2.6 Incremental Models

It combines elements of the linear sequential model applied respectively with the iterative philosophy of prototyping. First increment is often a core product. It is iterative in nature.

2.7 The Spiral Model

The spiral model, proposed by Boehm [BOE 88], is an evolutionary software process model that couples the iterative a nature of prototyping with controlled and systematic aspects of the linear sequential model as shown in **figure 4**. A spiral model is divided into a number of framework activities, also called tasks or regions. Customer communication is for effective communication between developer and customer. Planning is to define resources, timelines. Risk analysis is for assessing both technical and management risks. Engineering, Construction and release are to build one or more representations of the applications. Customer Evaluation is for obtaining customer feedback on evaluation of the software representations created during the engineering stage and implemented during the installation stage.

The spiral model of software development and evolution represents a risk-driven approach to software process analysis and structuring (Boehm 1987, Boehm et al, 1998). This approach, developed by Barry Boehm, incorporates elements of specification-driven, prototype-driven process methods, together with the classic software life cycle. It does so by representing iterative development cycles as an expanding spiral, with inner cycles denoting early system analysis and prototyping, and outer cycles denoting the classic software life cycle. The radial dimension denotes cumulative development costs, and the angular dimension denotes progress made in accomplishing each development spiral as shown in Figure **4.** Risk analysis, which seeks to identify situations that might cause a development effort to fail or go over budget/schedule, occurs during each spiral cycle.

In each cycle, it represents roughly the same amount of angular displacement, while the displaced sweep volume denotes increasing levels of effort required for risk analysis. In this model, System development therefore spirals out only so far as needed according to the risk that must be managed. Finally, efforts are now in progress to integrate computer-based support for stakeholder negotiations and capture of trade-off rationales into an operationa ı of the WinWin Spiral Model [Boehm et al, 1998 are considering the parts of this model to propos ware Architecture Development Life Cycle. (SADLC).

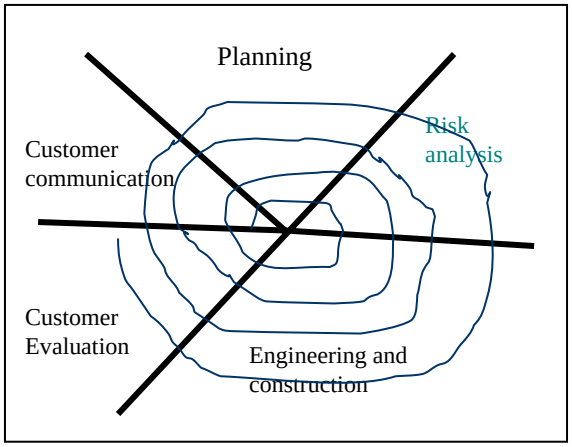

Figure 4 The spiral process model

2.8 The WinWin Spiral Model

The Customer wins by getting the system or product that satisfy the majority of the customer's needs and the developer wins by working to realistic and achievable budgets and deadlines. Boehm's WINWIN spiral model [BOE 98] defines a set of negotiation activities at the beginning of each pass around the spiral.

2.9 The Concurrent Development Model

The concurrent development model is also called concurrent engineering, Davis and Sitaram [DAV 94]. The concurrent process model defines a series of events that will trigger transitions from state to state for each of the software engineering activities. A system and component activities occur simultaneously and can be modelled using the state-oriented approach described previously. Each activity on the network exists simultaneously with other activities.

2.10 Component-Based Development

Object-Oriented technologies provide the technical framework for a component-based process model for Software Engineering.

3. SOFTWARE ARCHITECTURE AND ARCHITECTURAL ISSUES

Object-Oriented Technologies provide the technical framework for a component-based process model for software engineering. The Object-Oriented paradigm emphasizes the creation of components that encapsulate both data and the algorithms used to manipulate the data. *The software architecture process model incorporates* many of the characteristic of the spiral model and Object-Oriented process model. It is evolutionary in nature, demanding an iterative approach to the creation of software. The engineering activity begins with the identification of candidate components from the business logic. Software architecture shift focus of developers from the line of the code to coarse-grained architectural elements and their overall interconnection structure as shown in **figure 5.**

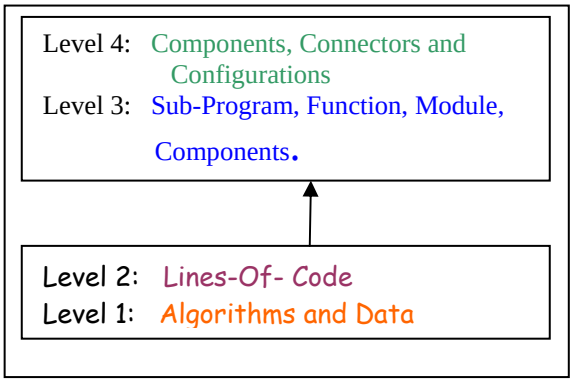

Figure 5. Architecture is shown as it is above the algorithms and Lines- of-Code

Architecture description languages (ADLs) have been proposed as Modeling notations to support architecture-based development. We have considered UML for Modeling. The components and connectors identified in the analysis and design are used for architecture analysis and architecture design. The first iteration of the application to compose and to build new components to meet the unique needs of the application. The Process flow then returns to the spiral and will ultimately re-enter the architectural issues loop during subsequent iteration through the engineering activity. The software architecture-based development model leads to software reuse, and reusability provides software engineers with a number of measurable benefits.

The unified software development process is representative of a number of architecture-based development models that have been proposed in the industry. Using the Unified Modeling Language (UML), the unified process defines the components that will be used to build the system and the interfaces that will connect the components. Using a combination of iterative and incremental development, the unified process defines the function of the system by applying a scenario-based approach. It then couples function with an architectural framework that identifies the form the software will take. **Figure 6** shows the overview of the transition from the Lines-Of-Code to architectural elements.

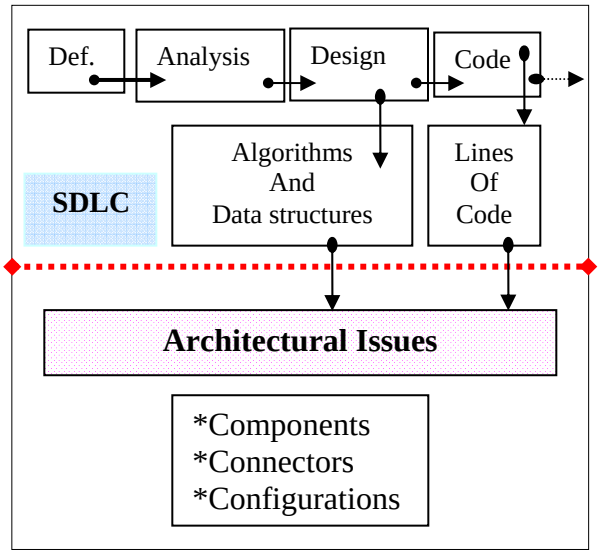

Figure 6 Overview of Transitions from algorithms, Ds & LOC to Architectural Elements

4. MODERN SOFTWARE EVELOPMENT

4.1 Transition design methods to emphasize component-based development

Moving from a line-of-code mentality to a component-based mentality is necessary to reduce the amount of human-generated source code and custom development.

Software architecture is the central design problem of a complex software system as shown in **figure 7**. Software architecture has several additional dimensions of complexity. There are many heuristics and fuzzy guidelines, but the fundamental measures of goodness are highly situation-dependent.. The requirements model addresses the behaviour of the system as seen by its end users, analysts, and testers. This view is modeled statically using use case and class diagrams and dynamically using sequence, collaboration, state chart, and activity diagrams. The design model addresses the architecture of the system

and the design of the components within the architecture, including the functional structure, concurrency structure, implementation structure, and execution structure of the solution space, as seen by its developers. Static descriptions are provided with structural diagrams (like, class, object, component, deployment diagrams). Dynamic descriptions are provided with any of the UML, Behavioural diagrams (collaboration, sequence, state chart, activity diagrams).

Figure 8, shows the procedure for architecture analysis and design. The input is from the business architecture or from software development life cycle. We propose here SADLC; it has the every thing about Software Architecture Analysis, Architecture design, Evaluation of design. It mainly concentrates on Architectural Issues

A detailed view of transforming or generating Architecture elements from conventional SDLC is shown in **figure 9**. The transitions are shown gradually from SDLC to Software Architecture

5. A TECHNICAL PERSPECTIVE OF THE ARCHITECTURE:

Although software architecture has been discussed at length over the past decade, convergence on definitions, terminology, and principles has been lacking. Software architecture encompasses the structure the software systems, their behaviour and the patterns that guide these elements, their collaborations and their composition. An Architecture framework is defined in terms of views that are abstractions of the UML models in the design set. Most real-world systems require four views: design, process, component, and deployment.

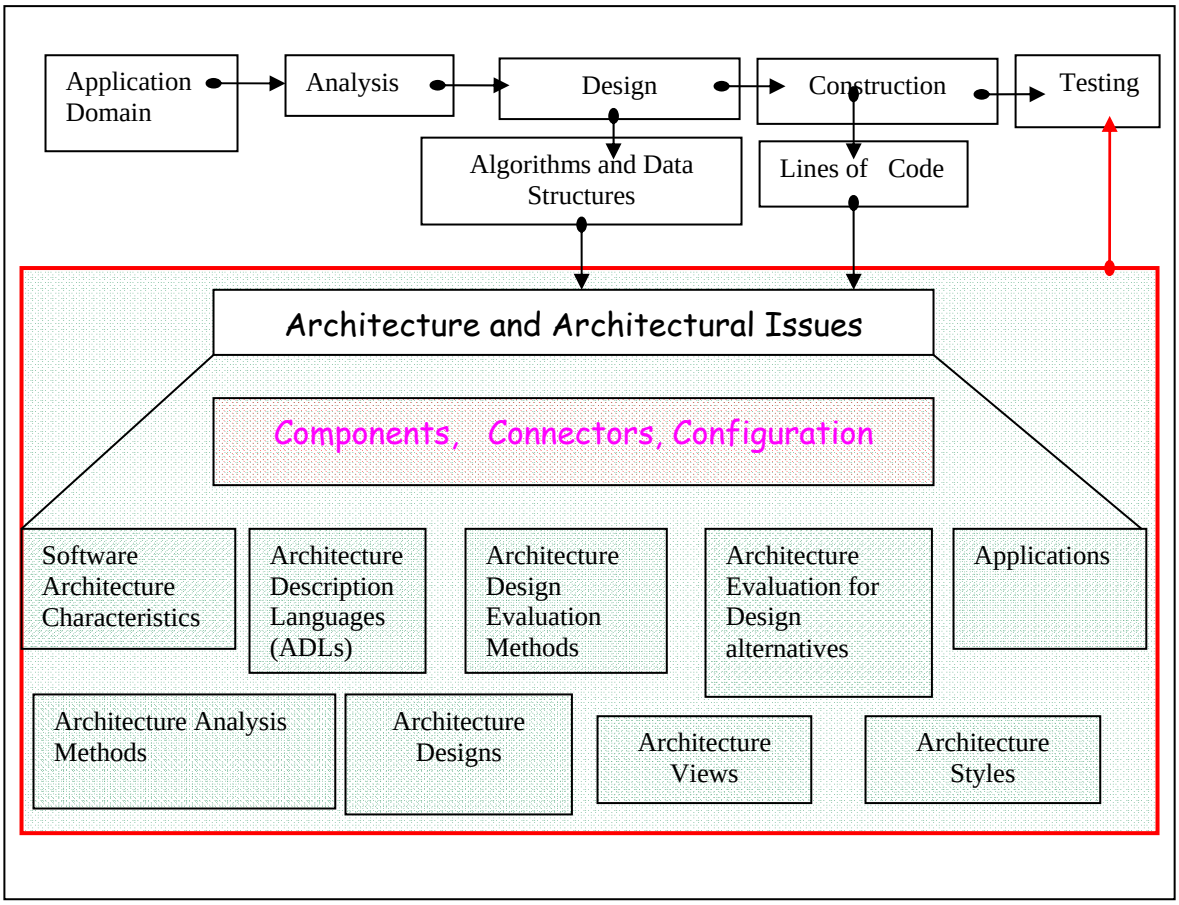

Figure 7 Software Development Life Cycle for Architecture-Based Software Development and its related issues

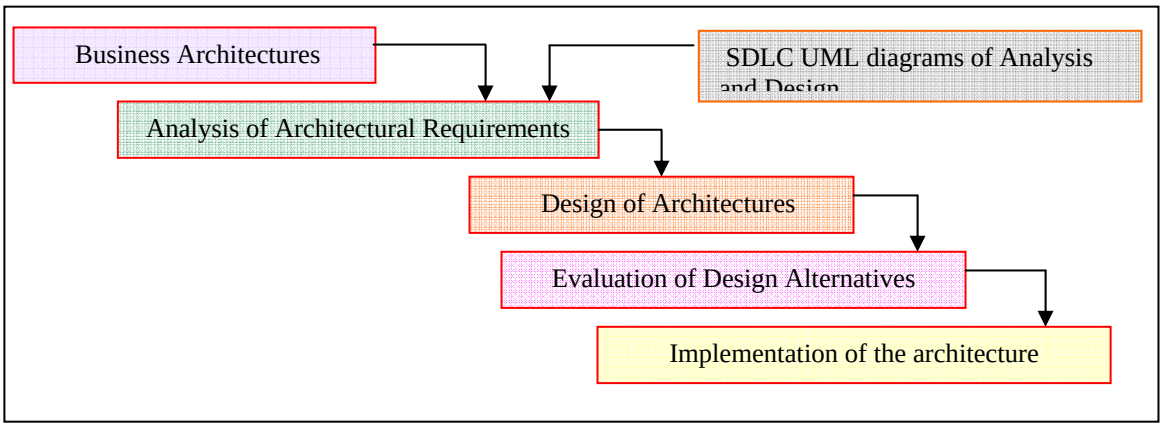

Figure 8 Software Architecture Development Life Cycles (**SADLC**)

The purposes of these views are as follows and shown in **figure 10**. Design view, Process, Component view and Deployment view. The design view is probably necessary in every system: the other three views can be added to deal with complexity of the system at hand. For example, any distributed system would need a process view ad a deployment view Most large systems, as well as systems that comprise a mixture of custom and commercial components would also require a separate component view. The **figure 10** summarizes the artifacts of the design set, including the architecture views and architecture descriptions are defined as collections of UML diagrams.

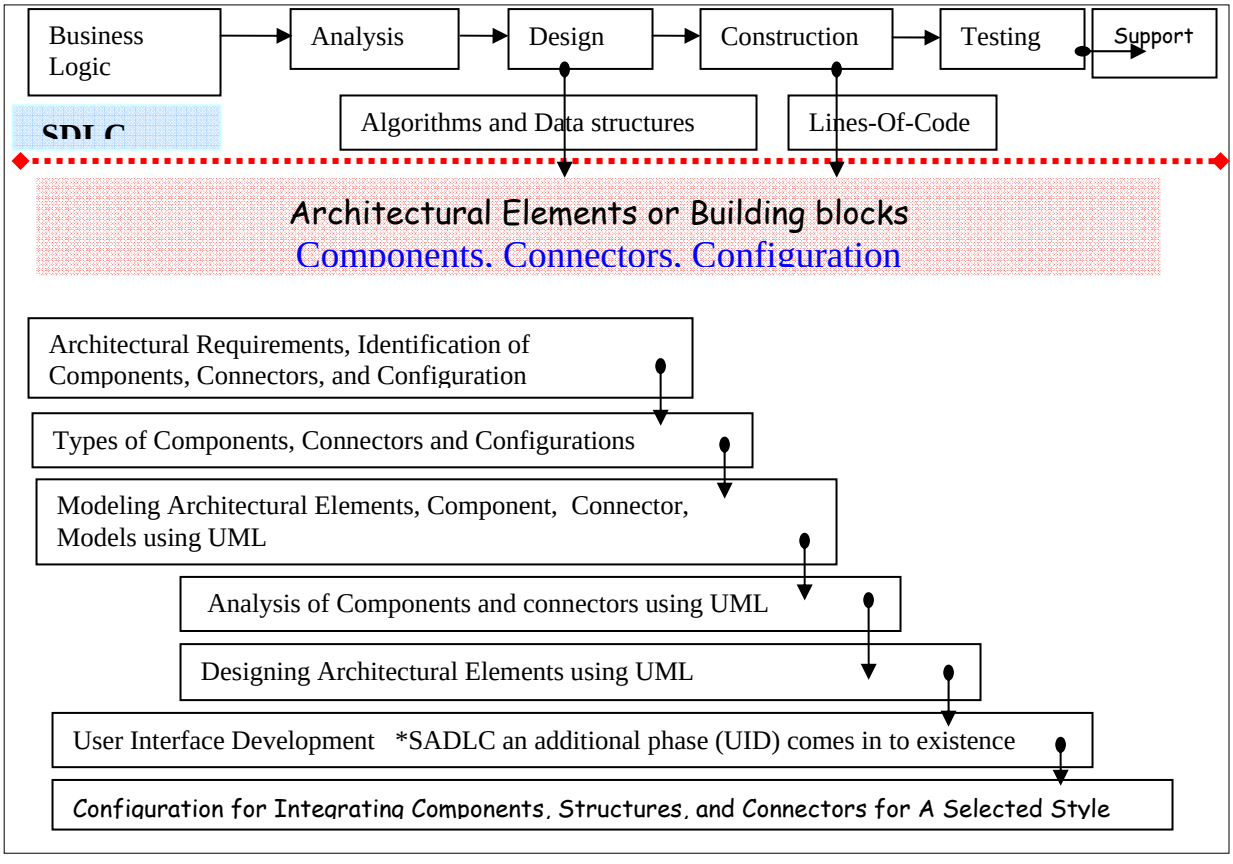

Figure 9 Software Development Life Cycles for Architecture-Based Software Development

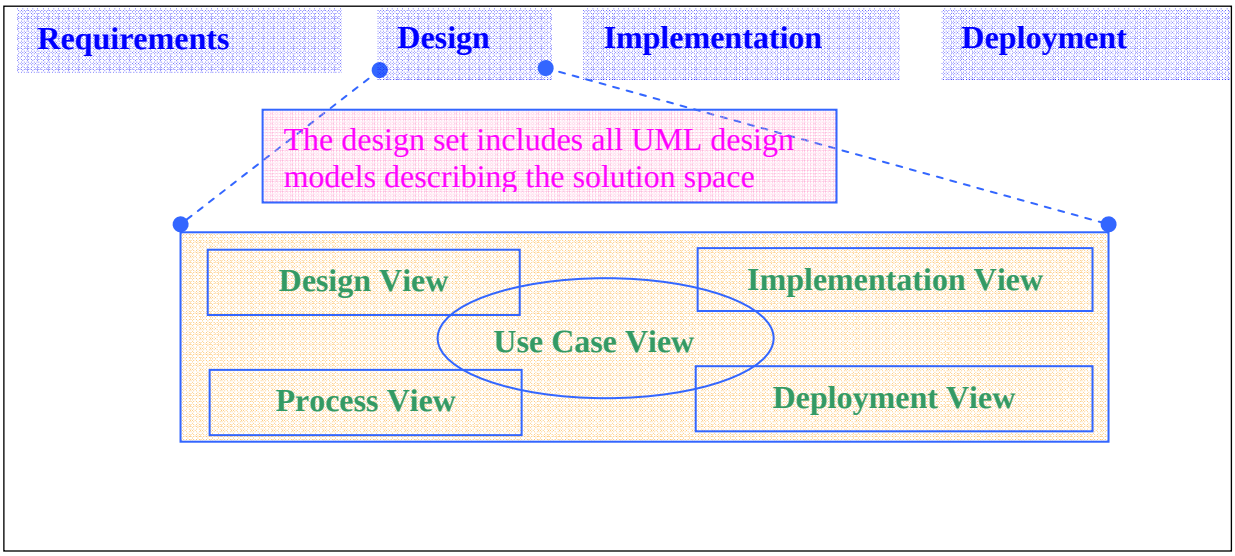

Figure 10 Generation of software architectures from the design of SDLC.

6. The proposed work

The process model adapted for object-oriented systems is a component assembly model [RSP 2003]. This model in turn uses the spiral model. In spiral model one of the regions is engineering and construction, from which the component assembly model takes a separate path and enters Object-Oriented software development area, where it searches for objects. If objects are found, they may be considered. Otherwise, using the concepts and principles of Object Oriented Analysis and Object Oriented Design, it constructs the required Objects and comes back and joins the engineering and construction region of spiral model. Next it goes for another spiral. This is way, it iterates till to achieve the required system.

In the proposed work, the **figure 11** shows the complete over view of the Software Architecture Development Life Cycle (SADLC). In SADLC we named some parts as spiral model area and architectural issues area. We have considered the conventional spiral model with out any deviation. The architectural issues, is the area, which encloses the principles and concepts of software architectures and every thing that are necessary for architecture analysis, architecture design, architecture evaluation for particular quality attribute, architectural analysis and design methods, architectural styles, views, and for description of architectures (ADLs) that are shown in figures (7), (8), (9), and (10). . In this regard, similar to the object-oriented assembly process model, in the SADLC, the control moves from spiral model to the architectural issues area with design (SDLC) information and resolve all architectural issues.

We are trying to understand and show the architectural elements are directly taken from business architectures and design of conventional Software Development Life Cycle (SDLC). The proofs and validity are being proposed in our extensions work.

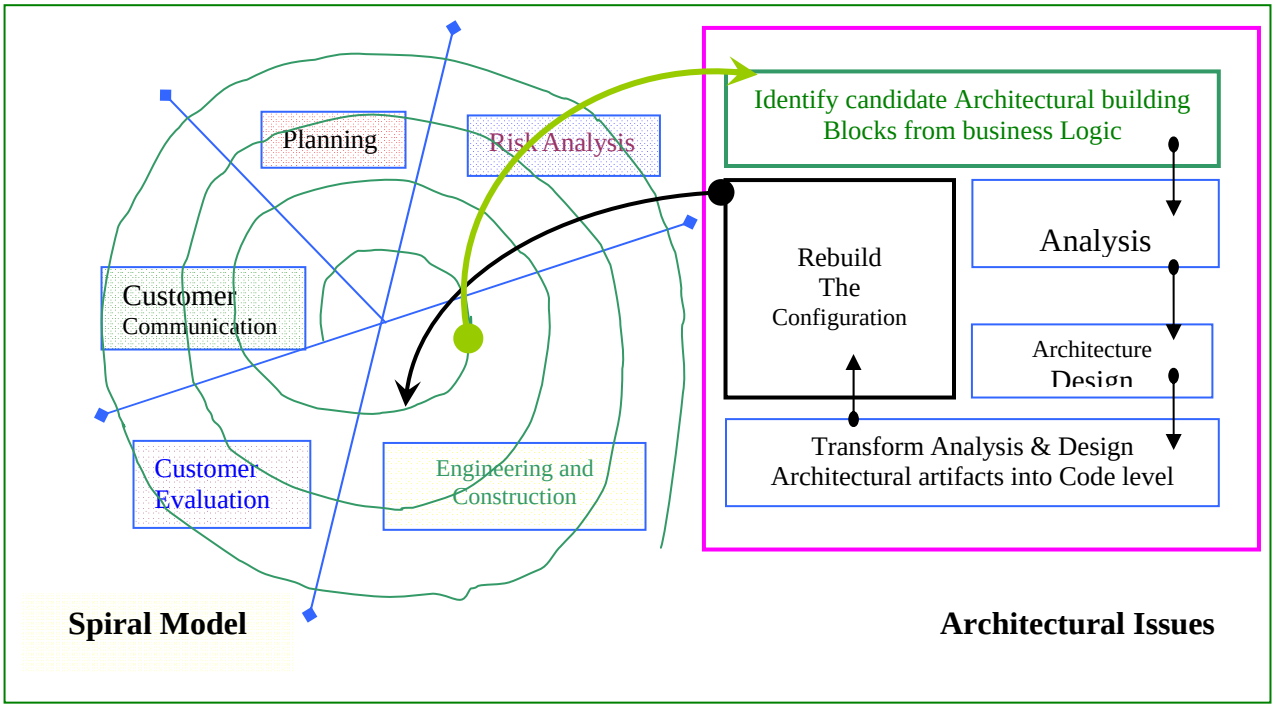

Figure 11 The proposed process models for software architecture development process model.

Conclusions

In software engineering, programming methodologies and software process models play important role. These two are interrelated and overlapped concepts and principles. Depending upon type, complexity and for particular non-functional and functional requirements various models are proposed and being adopted. Software Architecture is a branch of Software Engineering; it also requires a systematic and formal approach for implementing the concepts, principles while developing software architectures. In this context we are proposing a process model called Software Architecture Development Lifecycle (SADLC).

References

[BOE 88] Boehm, B., "A Spiral Model for Software Development and Enhancement," Computer, Vol. 21, no. 5, May 1988, pp. 61-72.

[BOE 98] Boehm, B., "Using the WINWIN Spiral Model: A Case Study," computer, vol. 31, no. 7, July 1998, pp, 33-44.

[Boehm 1976] Boehm, B., Software Engineering, IEEE Trans. Computer, C-25,12,1226-1241, 1976.

[Boehm 1981] Boehm, B. W., Software Engineering Economics, Prentice-Hall, Englewood Cliffs, N. J., 1981

[Boehm 1987] Boehm, B., A Spiral Model of Software Development and Enhancement, Computer, 20(9), 61- 72, 1987.

[Boehm et al, 1998] Boehm, B., A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy, Using the WinWin Spiral Model: A Case Study, Computer, 31(7), 33-44, 1998.

[Curtis, Krasner, Iscoe, 1988] Curtis, B., H. Krasner, and N. Iscoe, A Field Study of the Software Design Process for Large Systems, Communications ACM, 31, 11, 1268-1287, November, 1988

[DAV 94] Davis, A. and P. Sitaram, "A Concurrent Process Model for Software Development," Software Engineering Notes, ACM Press, vol. 19, n0. 2, April 1994, pp. 38-51

[Distaso 1980] Distaso, J., Software Management--A Survey of Practice in 1980, Proceedings IEEE, 68, 9, 1103-1119, 1980

[GIL 88] Gilb, T., Principles of Software Engineering Management, Addison-Wesley, 1988.

[Hosier 1961] Hosier, W. A., Pitfalls and Safeguards in Real-Time Digital Systems with Emphasis on Programming, IRE Trans. Engineering Management, EM-8, June, 1961

[Kling 1982, Garg 1989] Kling, R., and W. Scacchi, The Web of Computing: Computer Technology as Social Organization, Advances in Computers, 21, 1-90, Academic Press, New York, 1982.

Manuscript received April 5, 2007
Manuscript revised April 25, 2007

[PAU93] Paulk, M et al., “Capability Maturity Model for Software,” Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1993.
[ROY70] Royce, W.W., “Managing the Development of Large Software Systems: Concepts and Techniques, “Proc. WESCON, August 1970.
[RSP 2003] Roger R.S Pressman, Software Engineering 6th edition, 200

A Rama Mohan Reddy working as Professor of Computer Science and Engineering, Sri Venkateswara University., INDIA. He Completed his M.Tech Computer Science from NIT, Warangal. Currently he is pursuing Ph.D in software Architecture.