SOFTWARE DEVELOPMENT LIFECYCLE MODEL (SDLC) INCORPORATED WITH RELEASE MANAGEMENT

Ms. Gajalakshmi P.

**1Assisant Professor, Department of Computer Science Auxilium College (Autonomous), Vellore, TamilNadu, India*

---***---

Abstract - ***Software Development Life Cycle or System development Life Cycle or simply SDLC (system and software is interchanged frequently in-accordance to application scenario) is a step by step highly structured technique employed for development of any software. SDLC allows project leaders to configure and supervise the whole development process of any software. There are various SDLC models widely accepted and employed for developing software. SDLC models give a theoretical guide line regarding development of the software. Employing proper SDLC allows the managers to regulate whole development strategy of the software. Each SDLC has its advantages and disadvantages making it suitable for use under specific condition and constraints for specified type of software only. Developers employ SDLC models for analyzing, coding, testing and deployment of software system. Software developed by employing the suitable SDLC models is better performers in the market when compared with their competitors. SDLC Models helps in regulating the software-system development time and helps in effective cost scheduling. A model which guarantees the development and delivery (release) teams engaged in some project have strong co-ordination and collaboration leading to enhanced productivity, efficiency, effectiveness and longer market life is developed. This can be achieved by incorporating concept of Release Management with basic SDLC phases.***

Key Words: *Software Development Life Cycle system, software system, application life cycle management.*

I. INTRODUCTION

The System Development Life Cycle framework provides a sequence of activities for system designers and developers to follow. The ideas about the software development life cycle (SDLC) have been around for a long time and many variations exist, such as the waterfall, spiral, prototype and rapid application development model (RAD). These variations have many versions varying from those which are just guiding principles, to rigid systems of development complete with processes, paperwork and people roles. It consists of a set of steps or phases in which each phase of the SDLC uses the results of the previous one. A Systems Development Life Cycle (SDLC) adheres to important phases that are essential for developers, such as planning, analysis, design and implementation. A number of system development life cycle (SDLC) models have been created: waterfall, spiral, prototype and rapid application development model (RAD).

Various software development life cycle models are suitable for specific project related conditions which include organization, requirements stability, risks, budget and duration of project. One life cycle model theoretical may suite particular conditions and at the same time other model may also looks fitting into the requirements but one should consider trade-off while deciding which model to choose. A software life cycle model is either a descriptive or prescriptive characterization of how software is or should be developed. A descriptive model describes the history of how a particular software system was developed. Descriptive models may be used as the basis for understanding and improving software development processes A prescriptive model prescribes how a new software system should be developed. Prescriptive models are used as guidelines or frameworks to organize and structure how software development activities should be performed and in what order.

To present an abstract definition of software development life cycle model (SDLC) incorporated with release management. There are various SDLC models widely accepted and employed for developing software. SDLC models give a theoretical guide line regarding development of the software. Employing proper SDLC allows the managers to regulate whole development strategy of the software. Each SDLC has its advantages and disadvantages making it suitable for use under specific condition and constraints for specified type of software only. We need to understand which SDLC would generate most successful result when employed for software development. There are various SDLC models such as Waterfall, Spiral, Prototype, Incremental and RAD model etc.

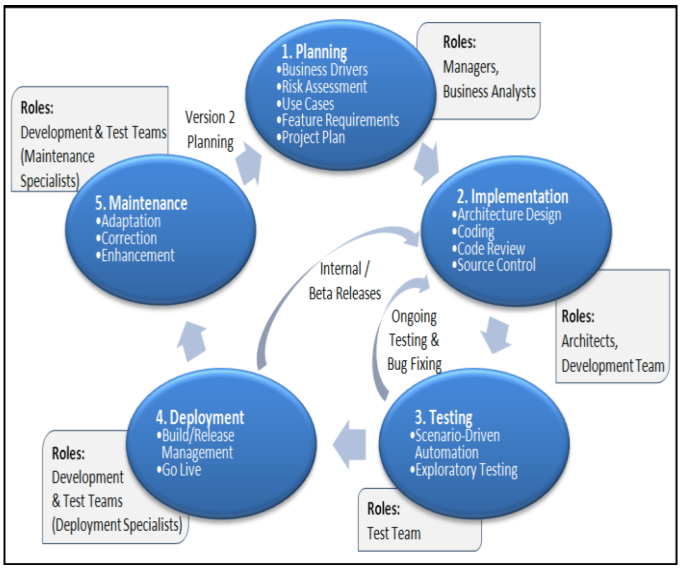


Fig Steps of the Software Development Life Cycle

II. RELATED WORK

WHAT IS SDLC

Software Development Life Cycle is a process to develop software. This process is divided into some phases such as of the areas there are many other papers that describe relevant work.

A. Requirement Analysis:

Requirement analysis is the initial phase of the Software Development Life Cycle. The goal of this phase is to understand the client's requirements and to document them properly. The emphasis in requirement analysis is an identifying what is needed from the system. It is most crucial phase in Software Development Life Cycle. The output of requirement analysis is Software Requirement Specification (SRS) [4],[5].

B. Design:

It is the first step to move from the problem domain towards the solution domain. It is the most creative phase in Software Development Life Cycle. The goal of this phase is to transform the requirement specification into structure [1]. The output of this phase is Software Design Document (SDD).

C. Coding:

In this phase Software Design Document (SDD) is converted into code by using some programming language. It is the logical phase of the Software Development Life Cycle. The output of this phase is program code.

D. Testing:

This is most important and powerful phase. Effective testing will contribute to the delivery of high quality software products, more satisfied users, lower maintenance costs, and more accurate and reliable results [1],[5],[6].

Waterfall Model

The waterfall model is a sequential software development process, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Requirement definition, System and software design, Implementation and unit testing, Integration and system testing, Operation and maintenance. Small to medium database software projects are generally broken down into five stages.

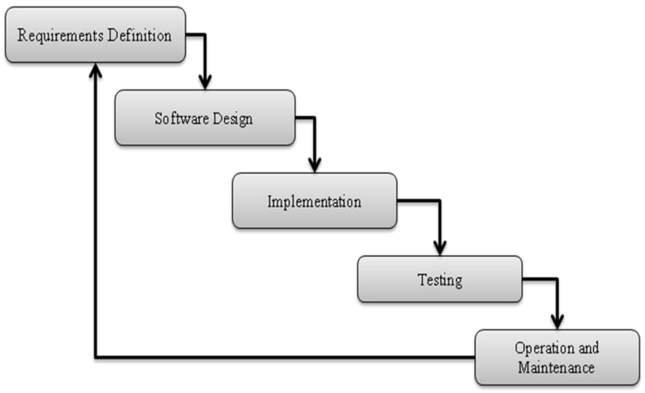


Fig : Water Fall Model

Stages of the Waterfall Model

Requirement Analysis and Definition

All possible requirements of the system to be developed are captured in this phase. Requirements are a set of functions and constraints that the end user (who will be using the system) expects from the system. The requirements are gathered from the end user at the start of the software development phase. These requirements are analyzed for their validity, and the possibility of incorporating the requirements in the system to be developed is also studied. Finally, a requirement specification document is created which serves the purpose of guideline for the next phase of the model.

Advantages

- Simple to understand and use.
- Easy to arrange tasks.
- Process and results are well documented.
- Each phase has specific deliverable and a review.
- Works well for projects where requirements are well understood.

- Works well when quality is more important than cost/schedule.

Disadvantages

- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- No working software is produced until late in the life cycle.
- Risk and uncertainty is high with this process model.
- Adjusting scope during the life cycle can end a project

2. Prototype Model

In this model prototype is built as per the client requirements. Instead of freezing the requirement before a design or coding can proceed. The purpose of a prototype is to allow users of the software to evaluate proposals for the design of the eventual product by actually trying them out, rather than having to interpret and evaluate the design based on descriptions. Prototyping has several benefits: The software designer and developer can obtain feedback from the users early in the project. The client and the developer can compare if the software made matches the software specification, according to which the software program is built. It also allows the software engineer some insight into the accuracy of initial project estimates and whether the deadlines and milestones proposed can be successfully met. A prototype model is not a standalone, complete development methodology, but rather an approach to handle selected part of a larger, more traditional development methodology. It attempts to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process. User is involved throughout the development process, which increases the likelihood of user acceptance of the final implementation. Small-scale mock-ups of the system are developed following an iterative modification process until the prototype evolves to meet the user‟s requirement. While most prototypes are developed with the expectation that they will be discarded, it is possible in some cases to evolve from prototype to working system. A basic understanding of the fundamental business problem is necessary to avoid solving the wrong problem [3],[8],[9].

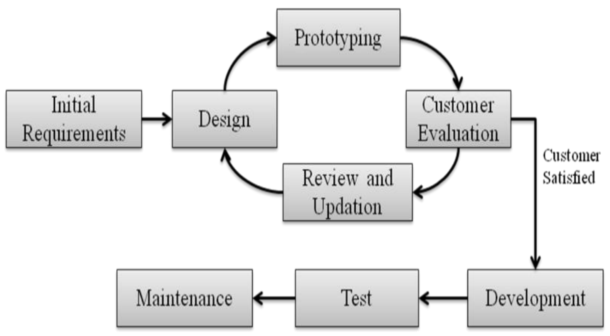


Fig : Prototype Moedel

Advantages

- Users are actively involved in the development
- When prototype Model is shown to the user, he gets a proper clarity about his requirements. And feel the functionality of the software, so can suggest the changes and modifications.
- It reduces risk of failure, as potential risks can be identified early and steps can be taken to remove that risk.
- The customer does not need to wait long for working software.

Disadvantages

- Wastage of Time and money to build prototype, if client not satisfied.
- Too many changes can disturb the rhythm of the developer team.
- Long term procedure.
- It follows the "Quick and dirty" approach- the prototype is through away after showing to the client.

III. NEW PROPOSED SDLC MODEL

The New SDLC model is designed in such a way that it allows client and developer to interact freely with each other in order to understand and implement requirements in a better way to produce a high quality software within budget and schedule. As the Software Development process began with the client's need, so the proposed model tries to discover most of the requirements of the client. It helps in developing an efficient software product that satisfies client. In the sphere of computer based system products, client satisfaction is dependent on how system development process evolves to build operational product systems that satisfy the perceived and actual client's need and associated system requirements. Ultimately, client satisfaction depends upon the depth of „through-life‟ understanding about the client needs and associated user requirements for a future system, and the ability to communicate those requirements to the system developer. In addition, client satisfaction and confidence depends upon the level of system assurance offered throughout the system development lifecycle. Requirements understanding problems inevitably lead to poor client-developer relationship, unnecessary re-work, and overrun cost and time. The client satisfaction is totally depended on client needs for this reason SDLC focus on the initial phases.

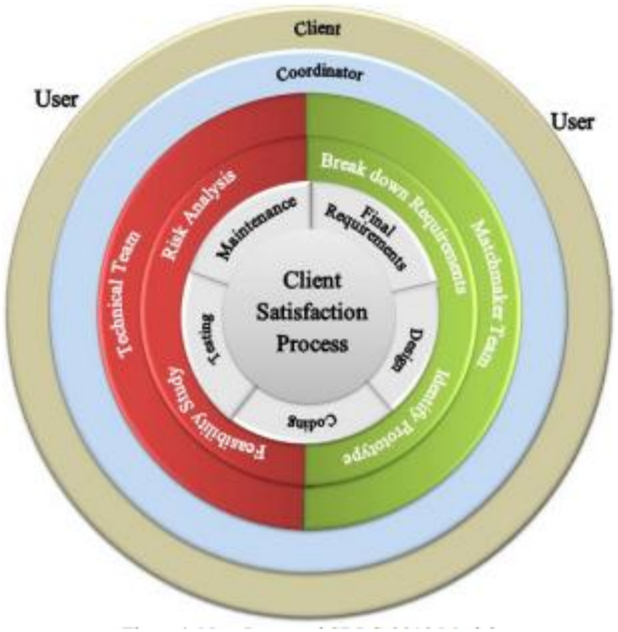


Fig: New Proposed SDLC Model

A. Coordinator

Coordinator have a general knowledge of every aspect of software development process, software applications, various applicable operating systems or platforms as well as various business functions to be performed. He coordinates with all the phases of the software development process. Coordinator deals with the client for gathering the requirements and passes these requirements to the matchmaker team and any query of client is also solved by the coordinator. After finalizing requirements coordinator estimates the cost, time and effort required to develop the software product. Then he passes the final requirements to the technical team. If client wants any change in the final requirements during the process, then coordinator firstly checks whether it can be implemented or not and what are impacts of change on the whole process in terms of cost, schedule and effort. If change is possible and its impact is little or very less then change will be accommodated

B. Matchmaker Team

Matchmaker team is an expert team and its team members are updated with new technologies and new software products. This team interacts with coordinator and technical team during its or king. Matchmaker team studies the requirements Received from the coordinator which in turn get these requirements from the client. This team identifies and gets the existing software whose requirements match with the current proposed Software's requirements. And accordingly breakdown the requirements into two parts implemented requirements and non-Implemented requirements. Implemented requirements are those requirements which are already implemented in some existing software.

C. Technical Team

It is a technically expert team. The member of this team is full of skills and interacts with coordinator and matchmaker team. Technical team works on non-implemented requirements. This team studies the feasibility of requirements to check whether these are technically possible or not. This team also identifies and resolves the various risk associated with the implementation of non-implemented requirements. After feasibility study and risk analysis the technical team verify the final requirements and pass these to the next phases, i.e. designing, coding, testing, each of these phase also followed by validation process.

IV. SYSTEM IMPLEMETNATION

4.1 Developing Software

Server-Application-Support Engineer

Release manager responsible for analyzing the troubleshoot problems with an application not typically at a code level but more at initial analysis level. He is even responsible for analyzing release policy and release plan. They help in identifying the problems that might occur during the later stages of the development of the software. There might be many issues that are likely to occur and will lead to reduce functionality of the software being developed. If these futuristic problems are identified and a help desk is stood against them then the efficiency and effectiveness of the software will increase manifolds. Thus server application support engineers' responsibility increases in early stages of SDLC as most of the requirements both functional and technical are identified and most likely problems that might associate with them are identified. Following are the two main functions being performed by the server application support engineers,

Analyze Release Policy: Release policies are high-level statements of how releases are to be managed, organized, and performed in your environment. Policies include management goals, objectives, beliefs, and responsibilities. Use these topics to learn more about release policies and to learn how to view, create, and edit the policies.

- **Release policy purpose**: Release policies are typically written at an overall strategy level. The management directives contained in a policy determine the way in which activities and tasks within a given release work-flow are performed.
- **View release policy:** Use this procedure to view release policies that have been written and posted to the database. These policies provide guidelines for carrying out a release work-flow.
- **Create/Edit release policy**: Use this topic to specify a link to a release policy that you have created or edited. After you create the link, the policy name and description are displayed in the Release Policies table. The policy document can then be opened and reviewed by any user in your environment who is involved with IBM Tivoli Release Process Manager

Analyze Release Plan: A release planning meeting is used to create a release plan which lays out the overall project. The release plan is then used to create iteration plan for each iteration release. It is important for technical people to make the technical decisions and business people to make the business decisions. Release planning has a set of rules that allows everyone involved with the project to make their own decisions. The rules define a method to negotiate a schedule everyone can commit to.

EVALUATION RESULT:

- Develop the process iteratively by knowing all requirements in advance. Several software development processes exist that deal with providing solution on how to minimize cost in terms of development phases.

- Manage requirements and always keep in mind the requirements set by users.

- A use component that breaks down an advanced project is not only suggested but in fact unavoidable.

- Model Visually: Use diagrams to represent all major components, users, and their interaction to make this task more feasible.

- Verify quality: Always make testing a major part of the project at any point of time. Testing becomes heavier as the project progresses but should be a constant factor in any software product creation.

Guidelines are followed for performing a simulation study for software development life cycles. It is composed of ten processes, ten phases, and thirteen reliability evaluation stages. Its purpose is to assess the credibility of every stage after simulation and match it with the initial requirements and specifications. The model provides one of the most documented descriptions for simulating life-cycles in the software engineering field.

Software engineering process simulation model (SEPS) is used in proposed system for the dynamic simulation of software development life cycles. It is based on using feedback principles of system dynamics to simulate communications and interactions among the different SDLC phases and activities from a dynamic systems perspective.

Basically, Software engineering process simulation model is a planning tool meant to improve the decision-making of managers in controlling the projects outcome in terms of cost, time, and functionalities. Discrete open source event simulation model is used for simulating the programming and the testing stages of a software development process using Math Lab.

Assumptions and Specifications

Prior to simulating the Waterfall model, a number of assumptions and specifications must be clearly made. Basically, projects arrive randomly at a software firm with inter-arrival time from a Triangular distribution with a lower limit of 30 days, an upper limit of 40 days, and a mode of 35 days. The probability density function is then given as:

$$f(x|a,b,c) = \begin{cases} 0 & \text{for } \ x < 30 \\ \frac{2(x-a)}{(b-a)(c-a)} & \text{for } \ 30 \leq x < 35 \\ \frac{2}{b-a} & \text{for } \ x = 35 \\ \frac{2(b-x)}{(b-a)(b-c)} & \text{for } \ 35 < x \leq 40 \\ 0 & \text{for } \ 40 < x \end{cases}$$

Projects can be divided into three groups based on their complexity and scale: 70% of the projects are small-scale projects, 25% are medium-scale projects, and 5% are large-scale projects.

The business analysis phase requires a Uniform distribution with a lower limit of 3 days and an upper limit of 5 days.

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } 3 \leq x \leq 5 \\ 0 & \text{for } x < 3 \text{ or } x > 5 \end{cases}$$

The design phase requires a Uniform distribution with a lower limit of 5 days and an upper limit of 10 days.

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } 5 \leq x \leq 10 \\ 0 & \text{for } x < 5 \text{ or } x > 10 \end{cases}$$

The implementation phase requires a Uniform distribution with a lower limit of 15 days and an upper limit of 20 days.

The testing phase requires a Uniform distribution with a lower limit of 5 days and an upper limit of 10 days.

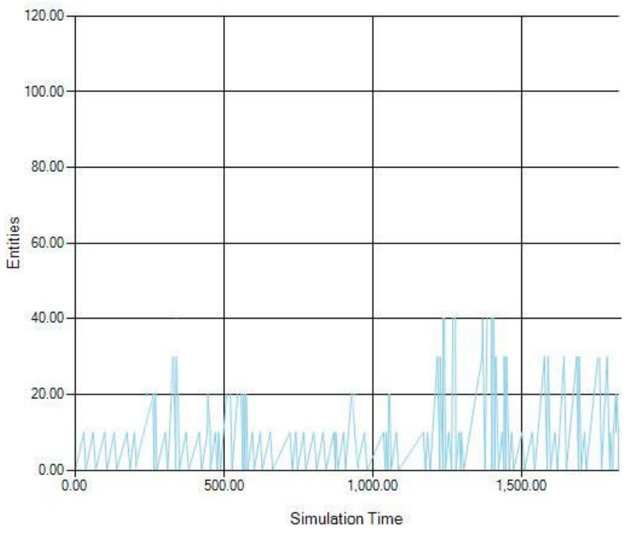
$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } 1 \leq x \leq 3 \\ 0 & \text{for } x < 1 \text{ or } x > 3 \end{cases}$$

Essentially, the existing model consists of a set of resource, queue, task, probability branch, capture and counter modeling elements. The resources are the basic employees and workers assigned to work on the phases of the Waterfall model. Each resource has a FIFO queue which accumulates and stores processing events to be processed later.

Resource	Average Utilization
Business Analysts	4.8
Designers	10.2
Programmers	20
Testers	6.5
Maintenance	1.9

Table: Resources with their Average Utilization in Waterfall Model

Essentially, the existing model consists of a set of resource, queue, task, probability branch, capture and counter modeling elements. The resources are the basic employees and workers assigned to work on the phases of the Waterfall model. Each resource has a FIFO queue which accumulates and stores processing events to be processed later



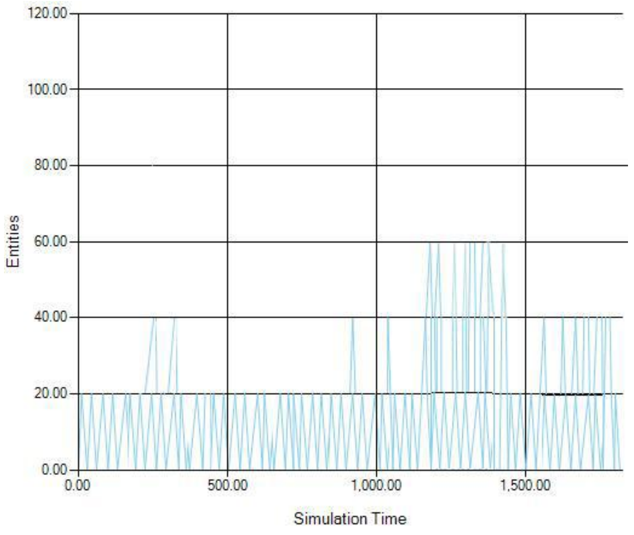


Fig: Utilization of the Designer Resource in Waterfall Model

The simulation model was executed 5 times, for 1500 milliseconds (2.5 minutes) with 50 incoming projects using the Simphony.NET environment. Delineates the obtained statistics including the number of projects received and delivered, in addition to the time based on release management. Delineates the average utilization of every resource after the completion of the simulation with release management. Furthermore, a graphical representation for resource utilization is plotted in Fig A-12 for the programmer resource while, is for the designer resource

Resource	Average Utilization
Business Analysts	5.2

Designers	11.6
Programmers	21.02
Testers	7.4
Maintenance	2.09

Table 2: Simulated Resources with their Average Utilization incorporating Release Management

The model starts with a new entity element which sets the number of incoming projects and a counter that counts the number of projects being received, and ends with another counter that counts the number of projects being delivered.

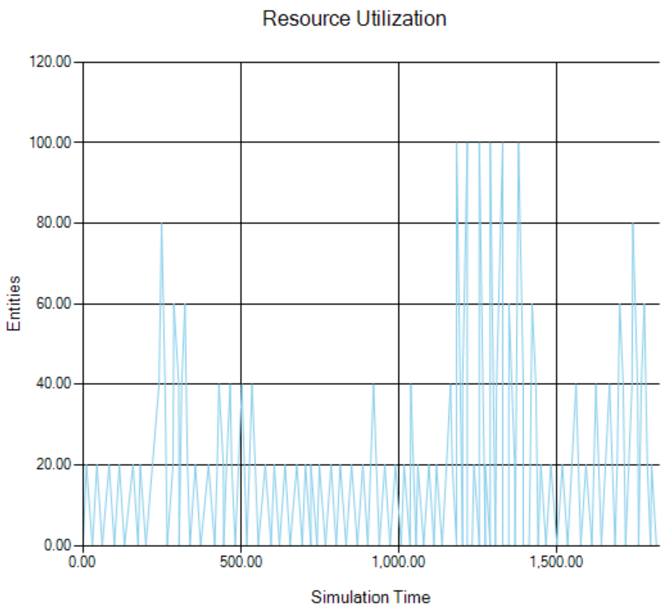
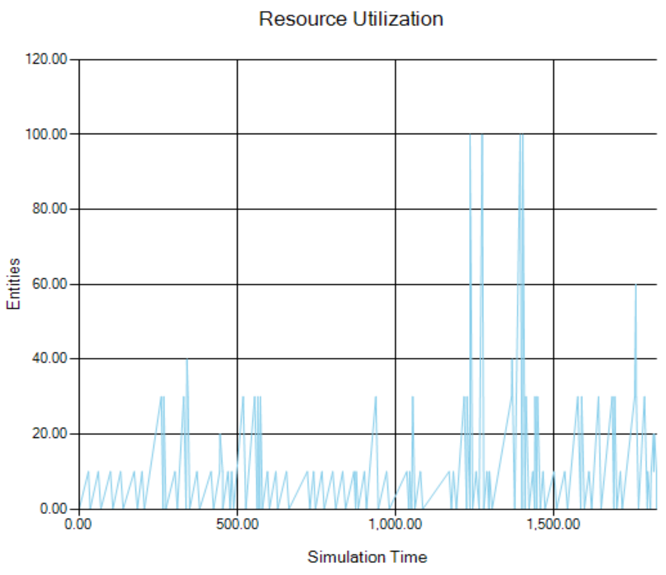


Fig: Utilization of the Programmer Resource after Release Management

- Divide the Waterfall model into independent phases.
- Understand the concept and the requirements that lie behind every phase.
- Define the resources, tasks, entities, and the work flow of every phase.
- Simulate each phase apart and record results.
- Integrate the whole phases together, simulate the system, and record results.



CONCLUSION

SDLC system/software development life cycle is a step by step systematic approach from planning to testing and deployment of the software. There are some basic phases that are strictly followed in the specified order as analysis, designing, coding, testing and implementation. Different SDLC models are employed for developing variety of software's depending upon the type and usability of the software. Release Management is an entirely new concept that has come into scenario in recent days. Conceptually release management itself has some major phases that ensure smooth and timely delivery of the software-system to the client. The proposed SDLC model merges the basic phases of software development and release management in such a manner that a new SDLC model is evolved making the fullest use of release management thus increasing the effectiveness and software life in the market. The proposed model is basically a four tier architecture comprising of Client side, End user, Developer side and the Release manager as individual (units) levels. It maps various release managers onto specific SDLC phases thus providing complete co-ordination of various release managers along with analyzers, designers, coders, testers over specified phases of SDLC. Release managers are provided with the centralized control over SDLC phases leading to regular release cycles. Clear guidelines have been established to address, which Release Manager has to do what and when in the various stages in SDLC.

The above model is practically successful as all the phases and the roles and responsibilities of each person involved are clearly established. Their interaction and inter-relation with each other is well defined and thus this model successfully addresses all development phases that are integrated with the concept of release management.

REFERENCES:

1. Ian Sommerville, Software Engineering, Addison Wesley, 9th ed., 2010.
2. Richard H. Thayer, and Barry W. Boehm, *"software engineering project management"*, Computer Society Press of the IEEE, pp.130, 1986.
3. Craig Larman and Victor Basili, *"Iterative and Incremental Development: A Brief History"*, IEEE Computer, 2003.
4. N. Munassar and A. Govardhan, *"A Comparison Between Five Models Of Software Engineering"*, IJCSI International Journal of Computer Science Issues, vol. 7, no. 5, 2010.
5. P.Humphreys,*"Extending Ourselves: Computational Science, Empiricism, and Scientific Method"*, Oxford University Press, 2004.
6. Royce, W., *"Managing the Development of Large Software Systems"*, Proceedings of IEEE WESCON 26, pp.1-9, 1970.
7. IEEE-STD-610, *"A Compilation of IEEE Standard Computer Glossaries"*, IEEE Standard Computer Dictionary, 1991.
8. Andrew Stellman, Jennifer Greene, *"Applied Software Project Management"*, O'Reilly Media, 2005.
9. Jim Ledin, *"Simulation Planning"* PE, Ledin Engineering, 2000.
10. Software Methodologies Advantages & disadvantages of various SDLC models.mht
11. No.1, January 2010, *"Evolving A New sModel (SDLC Model-2010) For Software Development Life Cycle (SDLC)"* PK.Ragunath, S.Velmourougan, P. Davachelvan, S.Kayalvizhi, R.Ravimohan.
12. Hoek, A. van der, Wolf, A. L. (2003) *"Software release management for component-based software. Software—Practice & Experience".* Vol. 33, Issue 1, pp. 77–98. John Wiley & Sons, Inc. New York, NY, USA.