International Research Journal in Global Engineering and Sciences. (IRJGES)
ISSN : 2456-172X | Vol. 1, No. 4, January, 2017 | Pages 19-26

Software Engineering – A Conceptual Study on Software Development Life Cycle

Geetanjali Rave
Software Engineer, Accenture Services Pvt. Ltd, Bangalore, Karnataka

Abstract: With the growing market trends, success or failure abides a direct correlation towards impetus to change in the environment caused by the various surrounding bodies. A change to the existing operation is vital as its imperative for the practitioners to revisit their IT strategies and to seek business solutions that accommodate requirements pertaining to complexity, scalability, service and delivery, all at once. Impetus to change, Requirement gathering, Feasibility study, System Analysis, Strategy, Enactment, Verification, Validation, Post Validation and Disposition is flow the SDLC is outlined. Imbibing the requirement for the spur to change the existing model a step-by-step, spiral, V, Iterative and Incremental Method or Evolutionary prototyping method is used. Each method has its own pros and cons. Ever increasing need to shift to a framework where Business, Analytics, Technology and Operations world break the silos and become intertwined in a manner that allows business solutions to be synthesized. SDLC would thus no longer be a series of isolated phases, but encompass the entire mechanism of transforming a business problem to an executable solution.
Keywords – Software Development Cycle; Development Models

1.0 Introduction

Software Development Life Cycle is a methodical approach and precise, practiced for the improvement of a steadfast high quality software system. This paper deals with few of those SDLC models, namely; Waterfall model, Iterative model, Spiral model and Win-Win spiral model. Each development model has certain advantages and disadvantages. Software Maintainability must be given adequate focus during software development process to diminish the interruption. Instilling the maintainability features in the software application during its development can minimize the maintainability efforts during its real time use. This paper presents a new, Software Development Life Cycle model (SDLC) introducing maintainability development tasks or activities to be followed during the SDLC. Therefore, the main objective of this paper is to represent different models of software development and make a comparison between them to show the features and defects of each model.

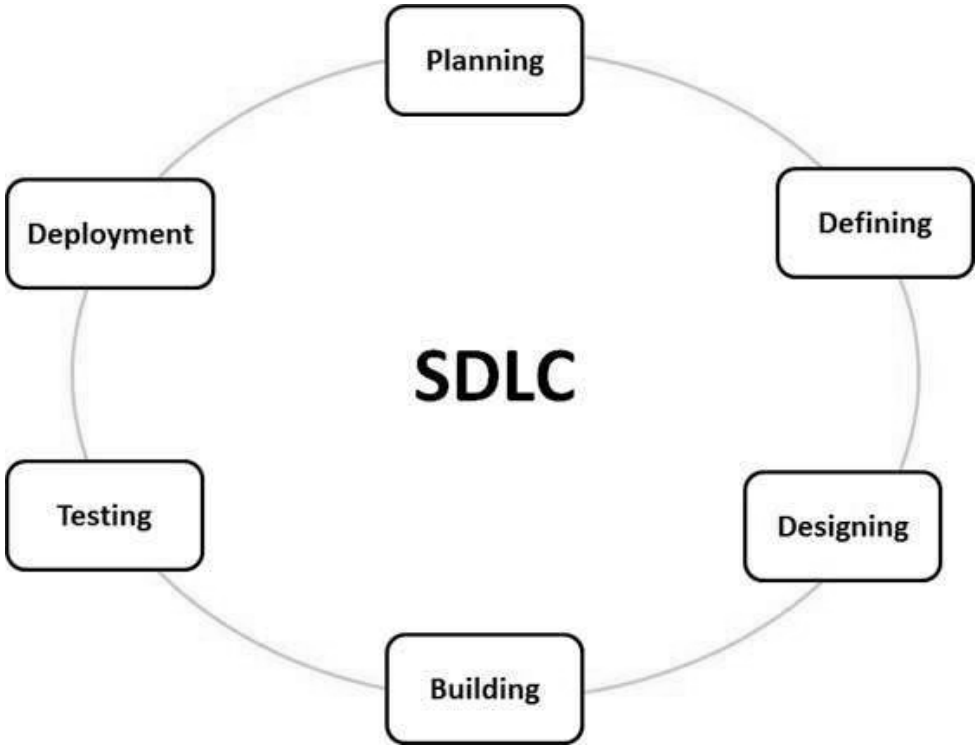

Image 1 – Software Development Life Cycle

2.0 Outline & Conceptual Summary

2.1 Software Process Models

SDLC model is an intangible representation of a procedure. It presents a description of a process from some particular perspective as:

1. Specification.
2. Design.
3. Validation.
4. Evolution.

General Software Process Models are:

1. Waterfall model.
2. Iterative model.
3. Spiral model.
4. Win-Win spiral model.

Many amendments of these models have been developed in the recent past e.g. Ceremonial development is the platform for waterfall-like process use but the specification is prescribed that is refined through several stages to an implementable design.

2.2 Waterfall model

The waterfall model is the orthodox model of software engineering. This model is one of the oldest models which is extensively used in many major companies. Because this model stresses on planning in premature stage. It ensures design blemishes before they could be developed. In addition, its concentrated document and planning makes it work well for projects in which quality control is a key. This model is used only when the requirements are flawless and permanent. Definition of the product is perfect in this model before the development would commence. The requirement is not vague. Generous resources with required expertise are available freely. Waterfall model helps the project to be short, precise and crisp.

The Waterfall Model is the oldest and most well-known SDLC model. The distinctive feature of the Waterfall model is its sequential step-by-step process from requirements analysis to maintenance. The major weakness of the Waterfall Model is that after project requirements are gathered in the first phase, there is no formal way to make changes to the project as requirements change or more information becomes available to the project team. Because requirements almost always change during long development cycles, often the product that is implemented at the end of the process is obsolete as it goes into production. The Waterfall Model is a poor choice for software development projects where requirements are not well-known or understood by the development team. It might not a good model for complex projects or projects that take more than a few months to complete.

Think about doing a home improvement project (such as new hardwood floors) for the first time and only being allowed to go the hardware store one time. The risk the project will fail is high. What are good candidate software development projects for the Waterfall Model? Systems that have well-defined and understood requirements are a good fit for the Waterfall Model. For instance, a military system aiming an artillery shell is a system with a single, simple requirement; put the shell on the target. This also assumes that the developers have worked on similar systems in the past and are experts in the application domain (artillery fire control systems). To follow the home improvement example, after a visit to a home to get specifications, an experience flooring contractor could install new hardwood floors with only one trip to the hardware store. In this case, the risk of project failure is low.

A smaller amount of customer's action is involved during the development of the product. The end product completion only can be moved to the user / customer. The product developed with any failure occurrence then the cost of fixing such issues are very high, because we need to update from the apex everywhere from document till the logic.

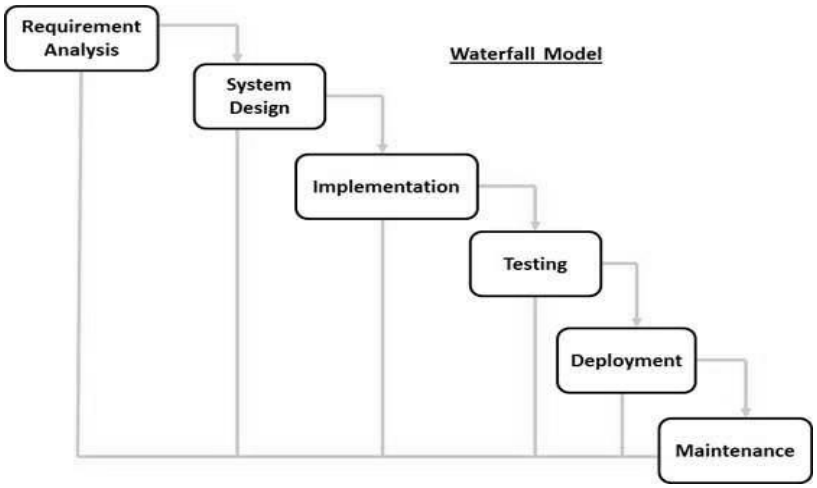

Image 2 – Water fall model diagram

2.3 Iterative model

The glitches with the Waterfall Model formed a demand for a new method of developing systems that could provide quicker results and require less straightforward information with mainly flexibility. In this model the project is divided into smaller modules, which allows the development team to validate results earlier, which helps to obtain valuable feedback from users/ customers. Widely, iteration consists of a small module -Waterfall process with the feedback from one phase providing dynamic information for the design of the succeeding phase. In a variation of this model, the software products, which are produced at the end of each step, can go into production immediately as incremental releases. Hence, Iterative model provides a results often for users to give feedback which in turn is an input for further development.

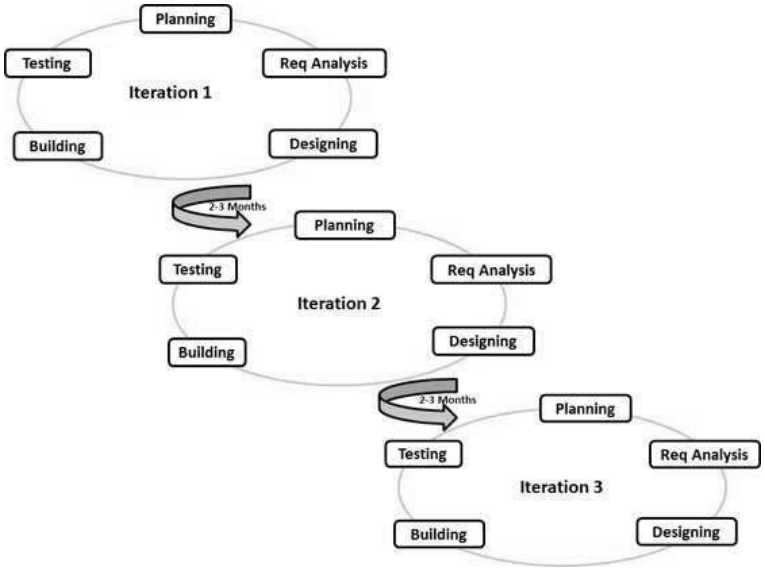

Image 3 – Iterative model diagram

2.4. Spiral Model

Analyzing the Risk needs distinction therefore, a model similar to Incremental model was developed - 'Spiral model'. The 4 stages that the model contains are Planning, Risk Analysis, Engineering and Evaluation. This model is similar to a spider web. The 4 stages consist of sub stages based on the baseline spiral. A software project repeatedly passes through these phases in iterations. In the planning phase the need for the requirement are assembled and the risk is weighed. Each succeeding spiral builds on the baseline spiral. Requirements are gathered during the planning phase. A process is undertaken to identify risk and alternate solutions which defines the purpose of the model. A pattern is produced at the end of the risk analysis phase. Software is produced in the engineering phase, along with testing at the end of the phase. The evaluation phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral. In the spiral model, the angular component represents progress, and the radius of the spiral represents cost. This model is suited for large and mission-critical projects. Software is produced early in the software life cycle. On the contrary this model's Risk analysis requires highly specific expertise because project's success is highly dependent on the risk analysis phase. This doesn't work well for smaller projects.

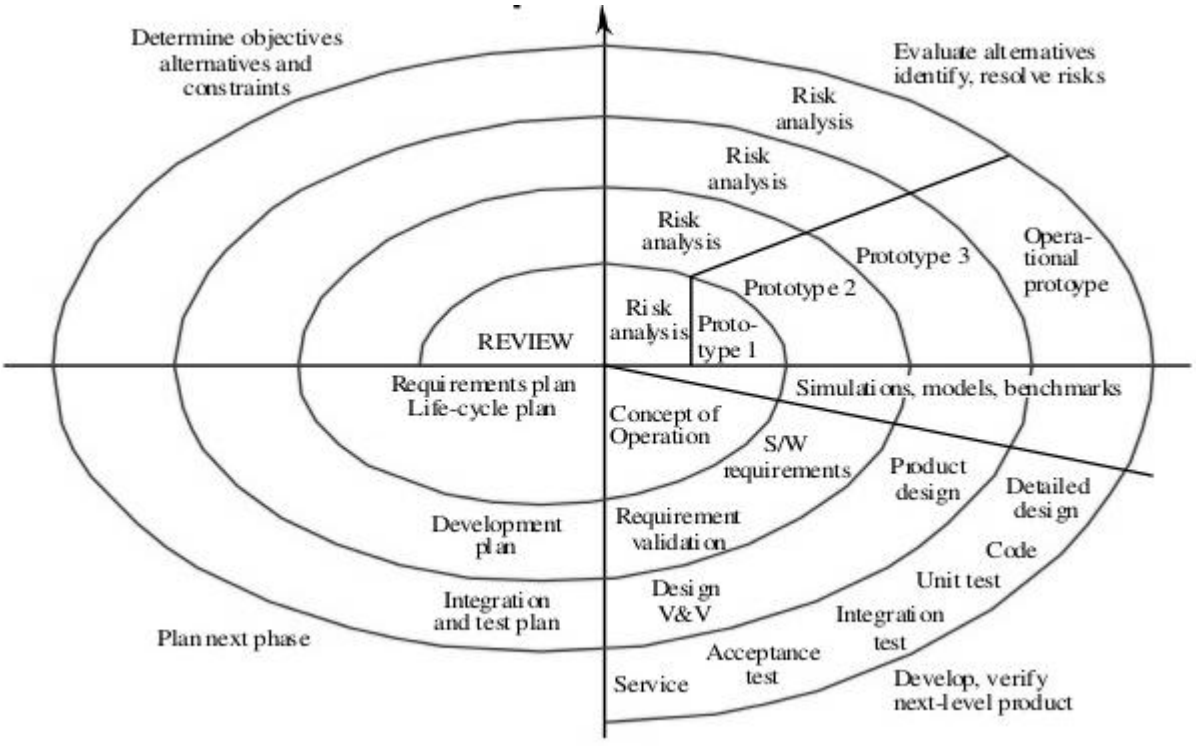

Image 4 – Spiral model diagram

2.5 Win-Win spiral model

The main difficulty in applying the spiral model has been the privation of explicit process guidance in determining these objectives, constraints, and alternatives. The Win-Win Spiral Model [Boehm 94] uses the theory W (win-win) approach to congregate on a system's next-level objectives, constraints, and alternatives. This Theory W approach involves identifying the system's stakeholders and their win conditions, and using negotiation processes to determine a mutually satisfactory set of objectives, constraints, and alternatives for the stakeholders. In particular, as illustrated in the figure, the nine-step Theory W process translates into the following spiral model Extensions:

1. Determine Objectives: Identify the system life-cycle stakeholders and their win conditions and establish initial system boundaries and external interfaces.
2. Determine Constraints: Determine the conditions under which the system would produce win-lose or lose-lose outcomes for some stakeholders.
3. Identify and Evaluate Alternatives: Solicit suggestions from stakeholders, evaluate them with respect to stakeholders' win conditions, synthesize and negotiate candidate win-win alternatives, analyses, assess, resolve win-lose or lose-lose risks, record commitments and areas to be left flexible in the project's design record and life cycle plans.
4. Cycle through the Spiral: Elaborate the win conditions evaluate and screen alternatives, resolve risks, accumulate appropriate commitments, and develop and execute downstream plans.

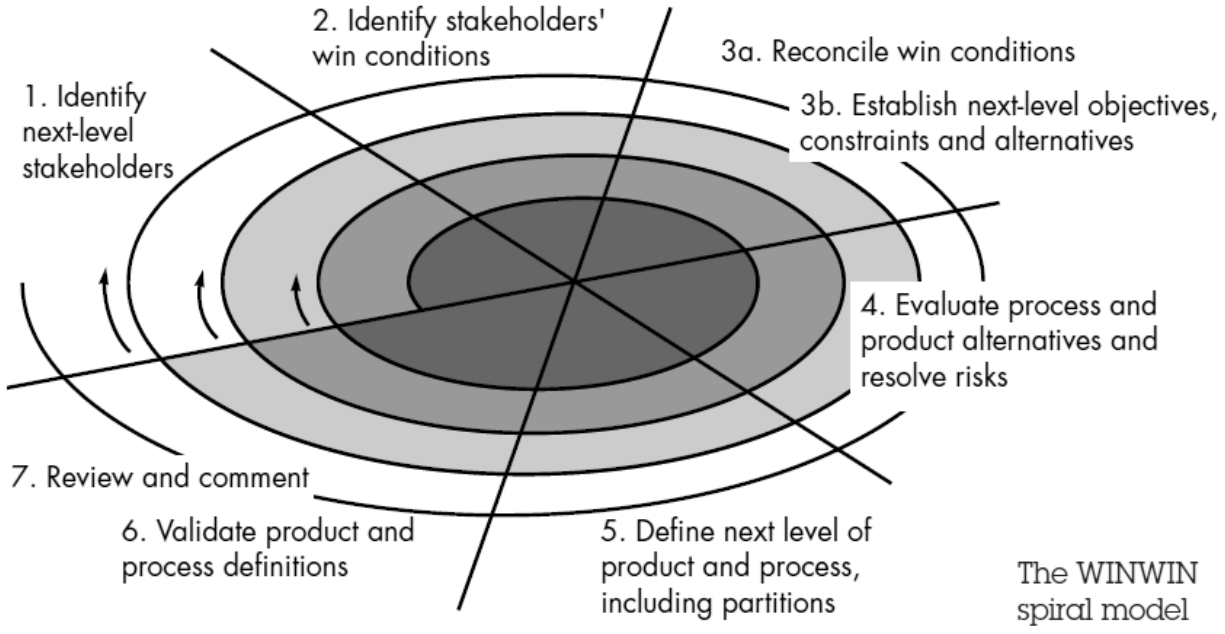

Image 5 – Win-Win Spiral model diagram

3.0 Objective

3.1 Model Selection

This paper did mention many models and their advantages and disadvantages. The question that now arises is, "Which model should I choose?" Note that we should choose the right type of the Model to implement based on the scope of the software project. This depends on a number of factors, some of which are given below.

- The Scope of the Project
- The Project Budget
- The organizational environment
- Available Resources

4.0 Conceptual Summary

Software Development Life Cycle (SDLC) is the process of developing information systems through analysis, planning, design, implementation, integration maintenance and testing of software applications. SDLC is also known as information systems development or application development. The development of quality software involves the usage of a structured approach towards the design and development of the end product. In a nutshell, the success of the SDLC process for building a successful software system rests on the following:

- Scope Restriction
- Progressive Enhancement
- Pre-defined Structure
- Incremental Planning at each of the stages

4.1 Societal Development

If each of these steps can be followed in its entirety, most of the risks that evolve in the software development life cycle can be mitigated. This article has provided a comprehensive explanation of the important and widely used Software Development Life Cycle (SDLC) phases and its models. It has also provided the advantages and disadvantages of each of these models.

5.0 Suggestions

To enhance the skills on improvising the knowledge of SDLC - suggesting a model to simulate advantages that are found in different models to software process management. Making a comparison between the suggested model and the previous software processes management

models. Applying the suggested model to many projects to ensure of its suitability and documentation to explain its mechanical work

6.0. Conclusion

After completing this research, it is concluded that there are various existing models for developing systems for different scopes of projects and requirements. Waterfall model and spiral model are used universally & frequently in developing systems. Each model has advantages and disadvantages for the development of systems, so each model tries to eliminate the disadvantages of the previous model

References

[1] Software Engineering (9th Edition) 9th Edition by Ian Sommerville
[2] Software Engineering: A Practitioner's Approach, 7th International edition 7th Edition by Roger Pressman
[3]Software Engineering Best Practices: Lessons from Successful Projects in the Top Companies by Capers Jones