A Review on Various Software Development Life Cycle (SDLC) Models

Prof. Seema Suresh Kute, Prof. Surabhi Deependra Thorat
Assistant Professor
MCA Department, MIT(E)
Aurangabad(M.S), India
Seemak0518@yahoo.in, Thorat.surabhi@gmail.com

Abstract
To satisfy the need of automate the different activities and satisfy the need of some standard and structural procedure or methodology which to be introduced in the industry so that the transition from manual to automated system became easy. This leads to the concept of System Development Lifecycle (SDLC) Models came into existence that emphasized on the need to follow some structured approach towards building new or improved system. There are five development models namely Linear Sequential Model, Iterative Model, Prototype Model, Spiral Model, V-shaped Model. All these models have cons as well as prons. The main objective of this research is to represent the different models of software development and make a comparative study of them to show the features and defects of each model.

Keywords:
Software Development Model (SDLC), Linear Sequential Model, Incremental Model, Prototype Model, Spiral Model, V-Shaped Model.

1. Introduction
Everybody knows how the computer is important in our day to day life. Now, computer uses in every field like in industry, hospital, medicine, education, agriculture even though in military area. Computer is used for developing the countries. The tasks which are very difficult and time consuming for human being that task can be done by the computer within some microseconds. Many companies make the software programs for the purpose of giving facilities to the offices.

During early software development some problems are occurred. So, for avoiding these problems or obstacles follow some structured flow of the software engineering and develop the software programs. Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.

2. System Development Life Cycle (SDLC) [1]
The Systems Development Life Cycle (SDLC), or Software Development Life Cycle in systems engineering, information systems and software engineering, is the process of creating or altering systems, and the models and methodologies that people use to develop these systems. The concept generally refers to computer or information systems. In software engineering the SDLC concept underpins many kinds of software development methodologies. These methodologies form the framework for planning and controlling the creation of an information system the software development process.

- Understand the Project.
- Do the project planning.
- Understand the requirement.
- Do the designing according to that coding the planned solution.
- Test the actual program.

For large software systems, each activity can be very complex and procedures are needed to perform it efficiently and correctly. So, the activities may be so large that it cannot be handled in single step and must be broken into smaller modules. For example, design of a large software system is always broken into multiple, distinct design phases, starting from a very high level design specifying only the components in the system to a detailed design where the logic of the components is specified. The basic activities or phases to be performed for developing software system are:

- Determination of System’s Requirements
- Design of system
- Development (coding) of software
- System Testing
### 2.1 Linear Sequential Model

The Linear Sequential Model is the classical model of software engineering. This model is one of the oldest models and is widely used in government projects and in many major companies. As this model emphasizes planning in early stages, it ensures design flaws before they develop. In addition, its intensive document and planning make it work well for projects in which quality control is a major concern.

The pure linear sequential lifecycle consists of several non-overlapping stages. The model begins with establishing system requirements and software requirements and continues with architectural design, detailed design, coding, testing, and maintenance. The waterfall model serves as a baseline for many other lifecycle models. The following list details the steps for using the waterfall starting coding. There is no overlap between stages. In real-world development, however, one can discover issues during the design or coding stages that point out errors or gaps in the requirements.

The linear sequential method does not prohibit returning to an earlier phase, for example, returning from the design phase to the requirements phase. However, this involves costly rework. Each completed phase requires formal review and extensive documentation development. Thus, oversights made in the requirements phase are expensive to correct later.

Because the actual development comes late in the process, one does not see results for a long time. This delay can be disconcerting to management and customers. Many people also think that the amount of documentation is excessive and inflexible.

Although the linear sequential model has its weaknesses, it is instructive because it emphasizes important stages of project development. Even if one does not apply this model, he must consider each of these stages and its relationship to his own project.

#### 2.1.1 Linear Sequential Model Major Potential

- This model is straightforward and easily understood for use.
- This model is easily manageable.
- One phase of a model is completed at one time.
- By using this model small qualitative projects are done.

#### 2.1.2 Linear Sequential Model Problem Area

- If the problem occurs in starting phase of a model then it is very hard to do the changes in that phase.
- Deliverable product is not produced within short time.
- High amounts of risk and uncertainty.
- This model is not used for complicated and object oriented projects.
- This model is not suitable for big quality projects.
- Not suitable for the projects where requirements are change frequently.

#### 2.1.3 When to use the Linear Sequential Model

- Requirements are very well known, clear and fixed.
- Product definition is stable.
- Technology is understood.
- There are no ambiguous requirements.
- Ample resources with required expertise are available freely.
- The project is short.

#### 2.1.4 Projects where Linear Sequential Method is suitable for SDLC

- In development of database-related software, e.g. commercial projects.
- In development of E-commerce website or portal.
- In Development of network protocol software.

### 2.2 Incremental Model

The incremental model combines elements of the linear sequential model (applied repetitively) with the iterative philosophy of prototyping. The Incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces a deliverable “increment” of the software. For example, word-processing software developed using the incremental Paradigm might deliver basic file management, editing, and
document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment. It should be noted that the process flow for any increment can incorporate the prototyping paradigm. When an incremental model is used, the first increment is often a core product. That is, basic requirements are addressed, but many supplementary features (some known, others unknown) remain undelivered. The core product is used by the customer (or undergoes detailed review). As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality. This process is repeated following the delivery of each increment, until the complete product is produced.

The incremental process model, like prototyping and other evolutionary approaches, is iterative in nature. But unlike prototyping, the incremental model focuses on the delivery of an operational product with each increment. Early increments are stripped down versions of the final product, but they do provide capability that serves the user and also provide a platform for evaluation by the user. Incremental development is particularly useful when staffing is unavailable for a complete implementation by the business deadline that has been established for the project. Early increments can be implemented with fewer people. If the core product is well received, then additional staff (if required) can be added to implement the next increment. In addition, increments can be planned to manage technical risks. For example, a major system might require the availability of new hardware that is under development and whose delivery date is uncertain. It might be possible to plan early increments in a way that avoids the use of this hardware, thereby enabling partial functionality to be delivered to end-users without inordinate delay.

2.2.1 Major Potential of Incremental Model
Develop active software product within early stages of software life cycle.
- This model is more adaptive – less costly to do the changes in requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can reply to every built.
- Delivery cost of the product is less.
- Handling the risk is very easy because the risks are handled during the iteration.

2.2.2 Problem Area of Incremental Model
- High quality planning and design is needed.
- Clear interpretation of the whole system is necessary before it can be broken down and develops incrementally.
- Total cost is higher than linear sequential model.

2.2.3 When to use the Incremental Model
- This model can be used when the requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some details can evolve with time.
- There is a need to get a product to the market early.
- A new technology is being used.
- Resources with needed skill set are not available.
- There are some high risk features and goals.

2.3 Prototype Model
The basic idea here is that instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements. This prototype is developed based on the currently known requirements. By using this prototype, the client can get an “actual feel” of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system. Prototyping is an attractive idea for complicated and large systems for which there is no
manual process or existing system to help determining the requirements. The prototype is usually not complete systems and many of the details are not built in the prototype. The goal is to provide a system with overall functionality.

2.3.1 Major Potential of Prototype Model
- Energetic users are involved in the development.
- In this model the working system is provided so that the users get a clear idea better of the system being developed.
- Delusion can be perceived much earlier.
- Rapid user feedback is available leading to better solutions.
- Problems related to functionality can be identified easily.

2.3.2 Problem Area of Prototype Model
- Forefront is implemented and then repairing way of building systems.
- Practically, when we use this model if the scope of the system may increase beyond the expectation then the complexity of the system is also increases.
- Incomplete application may cause application not to be used as the full system was designed.
- Incomplete or deficient problem analysis.

2.3.3 When to use Prototype Model
- Prototype model should be used when the desired system needs to have a lot of interaction with the end users.
- Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.

Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system. They are excellent for designing good human computer interface systems.

2.4 Spiral Model [8]
The spiral model, originally proposed by Boehm, is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model. It provides the potential for rapid development of incremental versions of the software. Using the spiral model, software is developed in a series of incremental releases. During early iterations, the incremental release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.

A spiral model is divided into a number of framework activities, also called task regions. Typically, there are between three and six task regions.

The spiral model that contains six task regions:
- Customer communication—tasks required to establish effective communication between developer and customer.
- Planning—tasks required to define resources, timelines, and other project related information.
- Risk analysis—tasks required to assess both technical and management risks.
- Engineering—tasks required to build one or more representations of the application.
- Construction and release—tasks required to construct, test, install, and provide user support (e.g. documentation and training) [8].

2.4.1 Major Potential of Spiral model
- Avoidance of risk is intensifying due to high amount of risk analysis.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Software is produced early in the software lifecycle.
- Additional Functionality can be added at a later date.
- Software is produced early in the software lifecycle.

2.4.2 Problem Area of Spiral Model
- This model is costly to use.
- For doing the Risk analysis requires highly specific expertise.
- Success of projects is highly dependent on the risk analysis phase.
- This model doesn’t work well for smaller projects.

2.4.3 When to use Spiral Model
- When costs and risk evaluation is important.
- For medium to high-risk projects.
- Long-term project commitment unwise because of potential changes to economic priorities.
- Users are unsure of their needs.
• Requirements are complex.
• New product line.
• Significant changes are expected (research and exploration).

2.5 V-Shape Model [4]
V-model means Verification and Validation model. Just like the waterfall model, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. Testing of the product is planned in parallel with a corresponding phase of development.

![V-Shape Model Diagram]

**Fig.05 V-Shape Model**

2.5.1 Major Potential of V-Shape Model
• This model is very simple and easy to use.
• Testing activities like planning, test designing happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
• Defects that are found at early stage that can be track easily and solve.
• This model is keep away from the downward flow of the defects.
• This model is used for small projects where requirements are easily understood.

2.5.2 Problem Area of V-Shape Model
• Very rigid and least flexible.
• Software is developed during the implementation phase, so no early prototypes of the software are produced.
• If any changes happen in midway, then the test documents along with requirement documents has to be updated.
• The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.
• The V-Shaped model should be chosen when ample technical resources are available with needed technical expertise. High confidence of customer is required for choosing the V-Shaped model approach. Since, no prototypes are produced, there is a very high risk involved in meeting customer expectation.

3. Conclusion
After studying of all models through the various factors, it has been found that the linear sequential model is used in database related projects in which all the requirements are known to the project team members before the project starts. Incremental model is used for new technology and where needed resources are not available. Prototype model used to develop online systems for transaction processing, web interfaces i.e. where very high amount of interaction with end users. Since significantly lead to the creation of working model in lower capital cost. Spiral model is used for high-risk projects, Long-term projects and where significant changes are expected at lowest cost. The V-shaped model is used for small size projects where requirements are clearly defined and fixed and when all type of technical resources are available. One model overcomes the disadvantages of other model.

4. References:
ISSN : 0976-8491(Online).

Table 1. Comparative Chart for Various SDLC Models

<table>
<thead>
<tr>
<th>Features</th>
<th>Linear Sequential Model</th>
<th>Incremental Model</th>
<th>Prototyping Model</th>
<th>Spiral Model</th>
<th>V-Shape Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement Specification</td>
<td>Beginning</td>
<td>Beginning</td>
<td>Frequently Changed</td>
<td>Beginning</td>
<td>Beginning</td>
</tr>
<tr>
<td>Understanding Requirements</td>
<td>Well Understood</td>
<td>Not Well Understood</td>
<td>Not Well Understood</td>
<td>Well Understood</td>
<td>Well Understood</td>
</tr>
<tr>
<td>Cost</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
<td>Expensive</td>
<td>Expensive</td>
</tr>
<tr>
<td>Availability Of Reusable Component</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Complexity Of System</td>
<td>Simple</td>
<td>Simple</td>
<td>Complex</td>
<td>Complex</td>
<td>Complex</td>
</tr>
<tr>
<td>Risk Analysis</td>
<td>At Beginning</td>
<td>No Risk Analysis</td>
<td>No Risk Analysis</td>
<td>Yes</td>
<td>High Risk</td>
</tr>
<tr>
<td>User Involvement In All Phases Of SDL C</td>
<td>Only At Beginning</td>
<td>Intermediate</td>
<td>High</td>
<td>High</td>
<td>Only At Beginning</td>
</tr>
<tr>
<td>Guarantee Of Success</td>
<td>Less</td>
<td>High</td>
<td>Good</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Overlapping Phases</td>
<td>No Overlapping</td>
<td>No Overlapping</td>
<td>Yes</td>
<td>Yes</td>
<td>No Overlapping</td>
</tr>
<tr>
<td>Implementation Time</td>
<td>Long</td>
<td>Less</td>
<td>Less</td>
<td>Depends On Project</td>
<td>Long</td>
</tr>
<tr>
<td>Flexibility</td>
<td>Rigid</td>
<td>Less Flexible</td>
<td>Highly Flexible</td>
<td>Flexible</td>
<td>Rigid</td>
</tr>
<tr>
<td>Changes Incorporated</td>
<td>Difficult</td>
<td>Easy</td>
<td>Easy</td>
<td>Easy</td>
<td>Easy</td>
</tr>
<tr>
<td>Expertise Required</td>
<td>High</td>
<td>High</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Cost Control</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Resource Control</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>