Software Development Lifecycle Models

Software development is a continuous process. It involves various phases which are to be completed for the complete development of the software. By using a software development lifecycle model in our project we make sure that our project works systematically and provides us the final product in a timely and cost effective manner.

The various phases which are included in the software development cycle are:

- Feasibility
- Requirements Specification
- Design
- Coding and Unit Testing
- Integration
- Testing
- Maintenance

Some of the lifecycle models we have assessed include:

- Classical Waterfall model
- Iterative Waterfall model
- Prototyping model
- Coding and Unit Testing
- Rapid Application Development model
- Evolutionary model
  - Incremental model
  - Spiral model
  - Concurrent Development model

Let us look at each of these models in detail and discuss the merits and demerits of each model.

Classical Waterfall model

This model is called the classical waterfall model because it diagrammatically represents a group of cascading waterfalls. It is a model in which each one of the lifecycle phases work in a sequential manner. This type of model requires user requirements to be understood very clearly and explicitly.

Merits:

- Model is simple and easy to use.
- Model is effective when the problem is well understood.
- Rigidity of the model makes adhering to deadlines easier.
- Model works better for smaller projects.

De-Merits:

- In this model reverting back to previous phases is costly and not feasible.
- Huge amount of risk.
- It requires all requirements to be stated explicitly.
- Difficult to accommodate uncertainty.
- A major mistake is as good as an irreversible mistake.

Iterative Waterfall Model

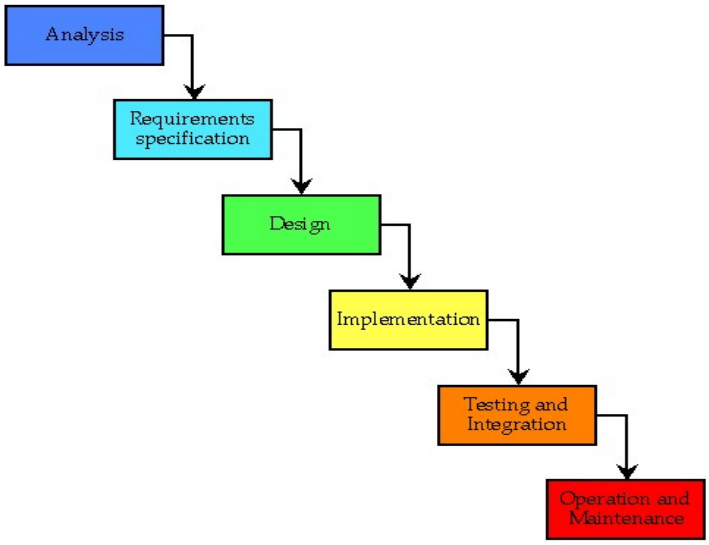

Figure 1: Classical waterfall model - Diagrammatic representation

An iterative waterfall model works on the principle that the errors should be detected and rectified as soon as possible. In this model one can go the previous phase where the error can be detected to ensure that all the previous phases uptil now are error free.

Merits:

- If the errors are detected at an early stage, this model becomes extremely systematic and easy to implement.
- Results are obtained periodically.
- Risk are identified and resolved during iteration.
- Easy to measure progress.
- Supports changing requirements.
- Easy to measure progress.

Demerits:

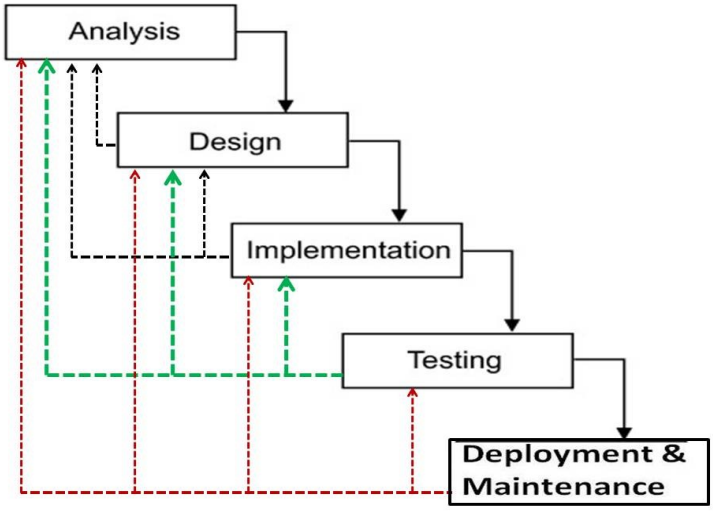

Figure 2: Iterative Waterfall Diagram

- If errors are not detected early, the whole development model fails.
- More management is needed.

Software Prototype Model

A system prototype refers to a working model of the system. A prototype gives an idea about the functionality of the system but it does not contain the exact logic of the system. Software prototyping is done to provide the user with an idea of what the system is and how it will work. It is done to ensure that the user is satisfied with the system's functionality before the actual implementation of the system.

Below is a diagram representing the different phases of this model.

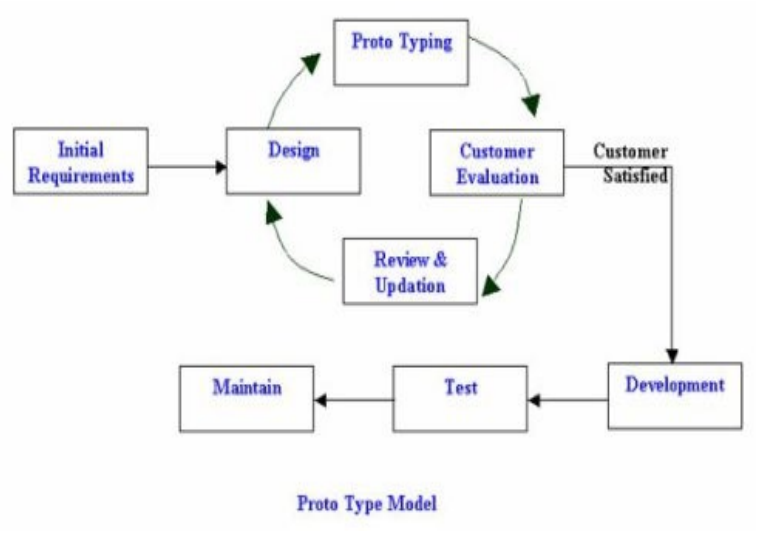

Figure 3: Software Prototype Model

Let us look at the merits and demerits of using such a model.

Merits:

- This models ensures better involvement of users in the development process which is always a good thing.
- This model reduces cost and time as defects in the features and functionality can be detected and rectified at an early stage.
- Since a working model of the system is provided to the user, he/she gets a better idea of the system being built and he can state his requirements more clearly.
- Missing functionality or misinterpreted functionality can be detected easily.

Demerits:

- If the prototype development process is not coordinated, this can consume a lot more time than any other model.
- As there is too much dependency on the prototype, there is a risk of requirements analysis becoming inefficient.

- If there are a large number of prototypes, the user may get confused.
- Developing a prototype is dependent on the skills and understanding of the development team. Hence this model becomes inefficient for totally new development environments and new areas of development.
- Developers may use the existing prototypes to make the actual system even when it is not technically feasible.

Spiral Model

The spiral model combines the idea of iterative development with the systematic, controlled aspects of the waterfall model. Spiral model is a combination of iterative development process model and sequential linear development model i.e. waterfall model with very high emphasis on risk analysis. It allows for incremental releases of the product, or incremental refinement through each iteration around the spiral.

Each loop has four sections or quadrants :

1. To determine the objectives, alternatives and constraints: We try to understand the product objectives, alternatives in design and constraints imposed because of cost, technology, schedule, etc.
2. Risk analysis and evaluation of alternatives: Here we try to find which other approaches can be implemented in order to fulfill the identified constraints. Operational and technical issues are addressed here. Risk mitigation is in focus in this phase. And evaluation of all these factors determines future action.
3. Execution of that phase of development: In this phase we develop the planned product. Testing is also done. In order to do development, waterfall or incremental approach can be implemented.
4. Planning the next phase: Here we review the progress and judge it considering all parameters. Issues which need to be resolved are identified in this phase and necessary steps are taken.

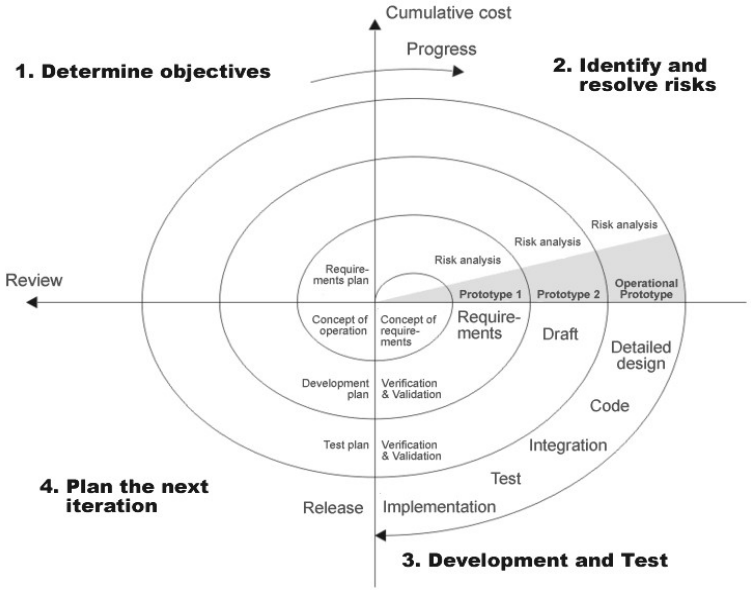

Figure 4: Spiral Model

NOTE : *Radius of the spiral (at any point) indicates the cost incurred on the project (so far). Angular dimension represents progress.*

Merits:

- Changing requirements can be accommodated later.
- Software is produced early in the software life cycle.
- Development can be divided into smaller parts and more risky parts can be developed earlier which helps better risk management.

De-Merits:

- Project's success is highly dependent on the risk analysis phase.
- Process is complex as large number of intermediate stages requires excessive documentation.
- Spiral may go indefinitely.
- Not suitable for small or low risk projects and could be expensive for small projects.

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
- Significant changes are expected (research and exploration)

Concurrent Development Model It is suitable for all types of software but is generally used for client server applications. It provides a precise state of the current state of the project. Rather than confining software engineering activities to a sequence of events, it defines a network of activities. Each activity on the network exists simultaneously with other activities. It focuses on concurrent development activities such as prototyping, analysis modeling, requirements specifications and design. The model can be represented systematically as a series of major technical activities, tasks and their associated states. Events generated at one point in the process network trigger transitions among the states.

Merits:

- Applicable to all types of softwares.
- Gives precise state of the project at a given time.
- Defines a series of events that will trigger transition from state to state for each of the software engineering activities and action or task.

De-Merits:

- The SRS document must be continuously updated to reflect the changes.
- It is inconvenient to avoid adding too many new features at later stages of software development.

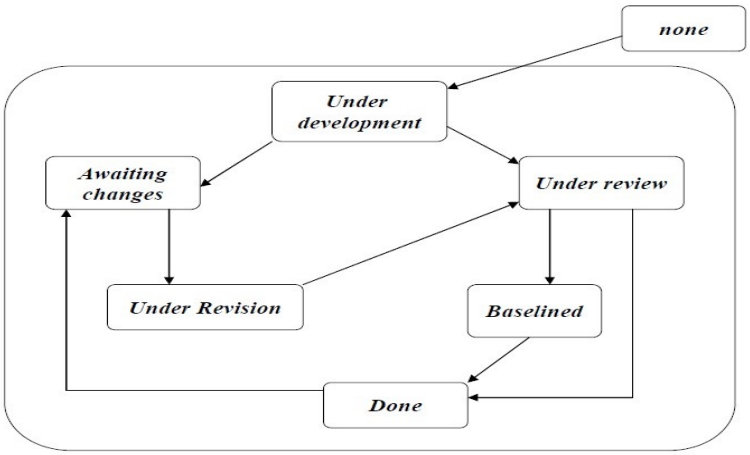

Figure 5: Concurrent Development Model

Incremental Model

Incremental build model is a method of software development where product is designed, implemented and tested incrementally until the product is finished.The product is defined as finished when it satisfies all the requirements. In this model the product is decomposed into number of component, each of which is designed and build separately. Each component is delivered to client when it is complete. The incremental model applies the waterfall model incremental.

Let us look at the merits and demerits of using such a model.
Merits:

- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- Customer can respond to features and review the product for any needful changes.

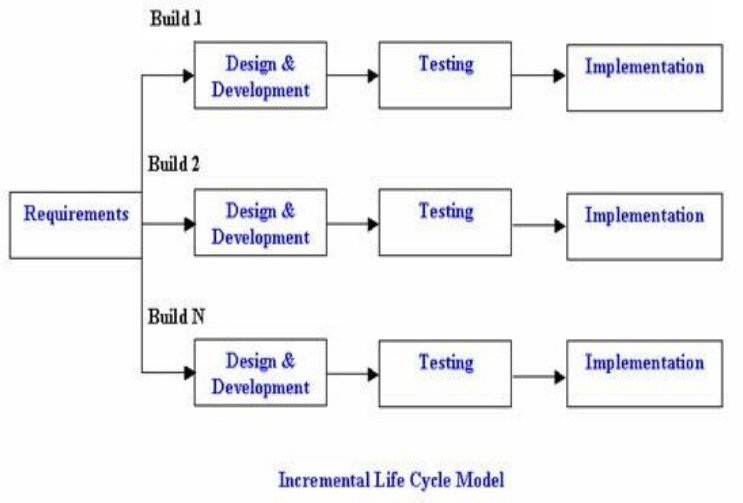

Figure 6: Incremental Model

De-Merits:

- Resulting Cost may exceed the cost of the organization.
- This model require good planning and design.
- In this, we require complete and clear definition of whole system before, It can be broken down and build incrementally.

Evolutionary Software Development Model

The ESD model divides the developmental cycle into smaller incremental waterfall models in which users are able to get access to the product at the end of each cycle. The users provide feedback on the product for the planning stage of the next cycle and the development team responds, often by changing the product,plans or processes.

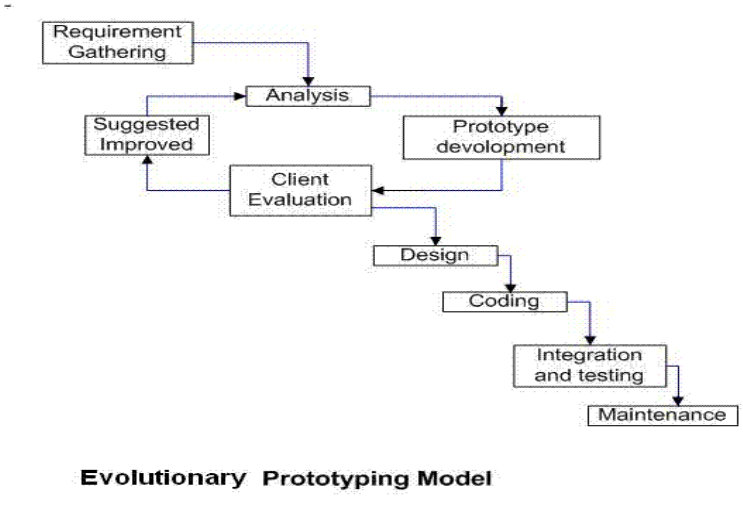

Figure 7: Evolutionary Software Development Model

Merits:

- There is a significant reduction in risk for software projects as by breaking the project into smaller pieces and increasing the visibility of management would ensure that these risks can be addressed and managed.
- ESD can reduce costs by providing a structured and a disciplined way for testing project
- It supports changing requirements and the Initial Operating time is less.
- Shorter cycles result in a more thorough and frequent improvement process.

De-Merits:

- Not suitable for smaller projects.
- Management complexity is more.
- End of project may not be determined which is a risk.
- Highly skilled resources are required for risk analysis.

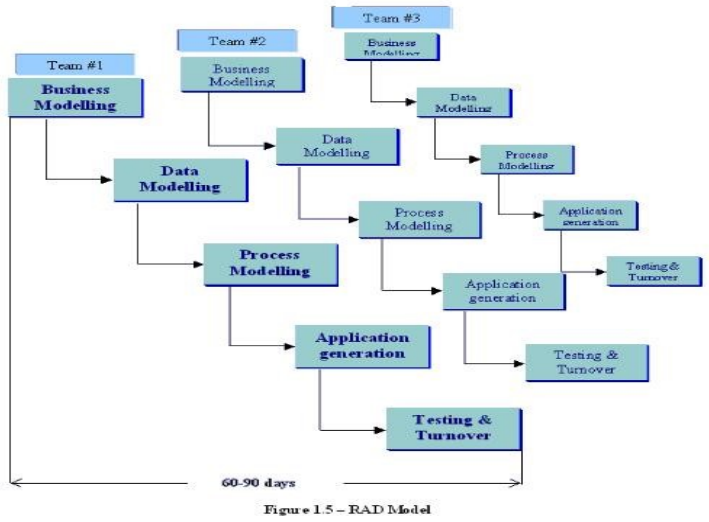

Figure 8: Rapid Application development

- Project's progress is highly dependent upon the risk analysis phase.

Rapid Application development (RAD)

Rapid application development RAD is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product. In RAD model the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery.

Since there is no detailed preplanning, it makes it easier to incorporate the changes within the development process. RAD projects follow iterative and incremental model and have small teams comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component or prototype.

Merits:

- Changing requirements can be accommodated.
- Progress can be measured.
- Iteration time can be short with use of powerful RAD tools.
- Productivity with fewer people in short time.
- Reduced development time.
- Increases reusability of components
- Quick initial reviews occur
- Encourages customer feedback
- Integration from very beginning solves a lot of integration issues.

De-Merits:

- Dependency on technically strong team members for identifying project requirements.
- Only system that can be modularized can be built using RAD.
- Requires highly skilled developers/designers.
- High dependency on modeling skills.
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.
- Management complexity is more.
- Suitable for systems that are component based and scalable.
- Requires user involvement throughout the life cycle.
- Suitable for project requiring shorter development times

Conclusion:
Decided Model