NIMS Suite

Version 1

Software Development Life Cycle Model Assessment

Team-16

Revision History

Date	Description	Author	Comments
30/1/2012	Version 1	Megha Tak	N/A

Software development cycles are a series of identifiable phases that a project goes through during its lifetime.

The basic phases are:

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

We have studied the various models and here are the details:

1. <u>Classical Waterfall Model</u>

<u>Explanation:</u>

According to this model after completion of one phase we can never go back to that phase again. We have to move in forward direction. If there is any error then it is usually detected in later stages of testing, which is when very difficult to manage and correct. So this model is definitely not suitable for our project. Because it is natural to have human errors in each phase of development and it will be very costly to manage all the incorrect data at the end of the project. And generally customers are not aware of all the functional requirements that they will need at the starting of the project. It is obvious that they will have suggestions and changes during the development phases of the project. These changes will be prohibitively costly to implement in this model.

<u>Merits:</u>

- Most easy to understand
- Very easy to implement but very difficult in real time problems

<u>Demerits:</u>

- Not good for product visualization.

- Requires expertise of team and clear picture of the project
- The model requires that all the requirements are stated explicitly.
- Changes can cause confusion as the project proceeds.
- The model has difficulty in accommodating the natural uncertainty that exists in many projects.

2. Iterative Waterfall Model

Explanation:

In this model if there is any error detected then we can revert back to the appropriate phase for error correction unlike that in classical model. Generally, it is most widely used model for software development. But the main point of emphasis is that it is easier to detect and correct the error in the same phase itself rather than reverting back from some other phase. So it is easy to correct any error. Also in this model if there are any changes in requirements during development process then it can be implemented easily.

Merits:

- It is fast process
- It has the potential to revert back to phases and correct then, in case of problems.

Demerits:

- It is a sequential model.

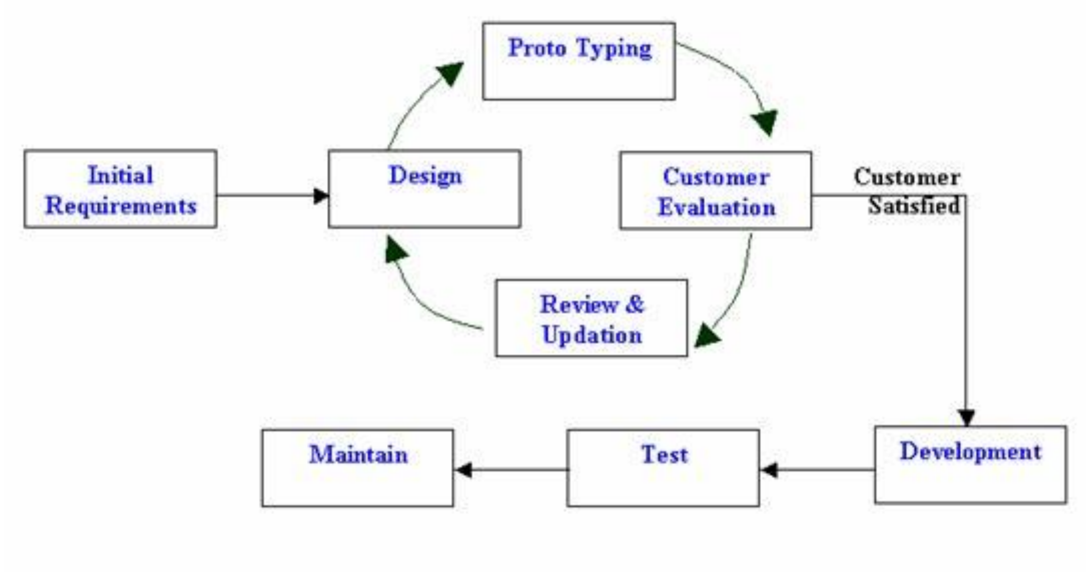
3. Prototyping Model

Explanation:

This model includes building prototype before developing the actual software. Prototypes are developed involving shortcuts like Limited Functional Capabilities, low reliability and inefficient performance.

A prototype (an early approximation of a final system or product) is built, tested, and then reworked as necessary until an acceptable prototype is finally achieved from which the complete system or product can now be developed using simple waterfall model. Certain specifications are:

- Requirements gathering phase includes prototyping
- Prototype code may be thrown away
- Time/effort spent on prototype is worth



Proto Type Model

Merits:

- Provides a good mechanism for understanding the customer requirement -Illustrates data formats, messages, reports, interactive dialogues
- Especially useful for GUI (Graphic User Interface) development
- Facilitates critical examination of technical issues associated with software development.
- Reduces wasted development effort

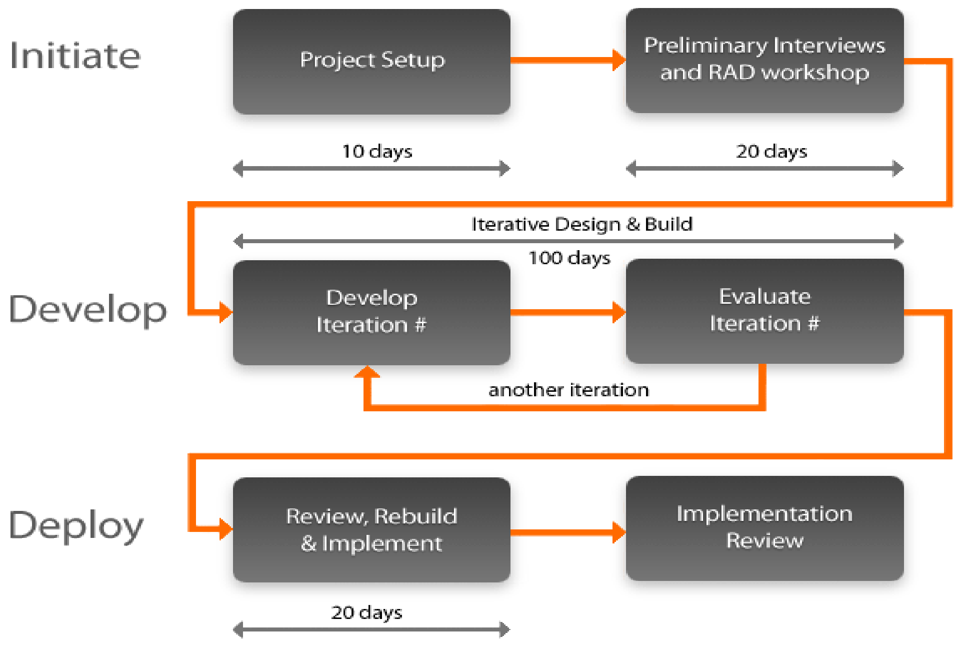
Demerits:

- Usually prototype without considering the overall software quality or long-term performance.
- Prototype development is very different from actual product development. Prototype is confused with the actual product
- Customer demands that the prototype be quickly converted into actual product.
- Compromises on the choices of OS, programming language is often made during prototyping.

4. Rapid Application Development (RAD) Model

Explanation:

As the name suggests, it is a technique that aims at fast development of software, with less planning requirements. This is generally done when we have strong teams with dedicated people committed to work for the product development. It gives fast and cost effective results, and planning is interleaved in coding phases mostly.



Merits:

- It allows making software in very short span of time.
- It uses available resources or try to make reusable codes and software
- Focuses more on testing and reliability
- It is easy to make changes in the software at later stages due to the way codes are written.

Demerits:

- Needs larger teams and individual commitment, with proper skills and coordination for projects
- Cannot be applied to new, innovative works where scope is not well defined and modularization is not simple
- Not to be used for projects which need a lot of field work and planning

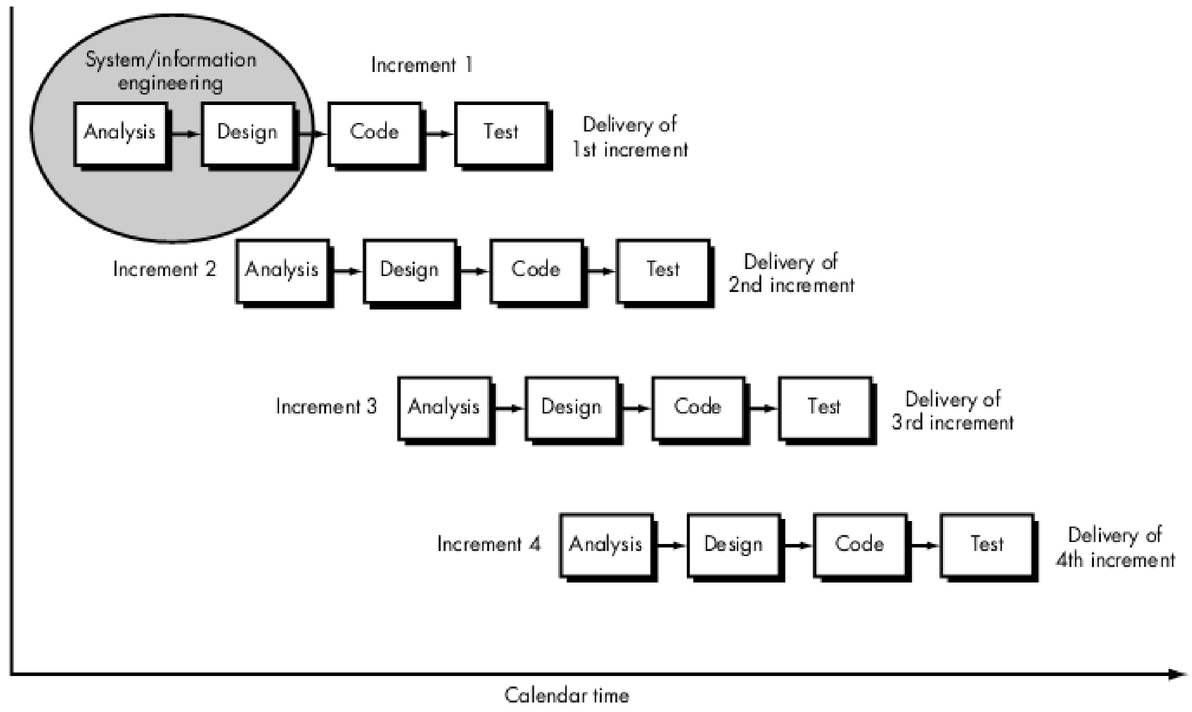
5. Evolutionary Models

There are various kinds of evolutionary models, which are discussed and explained below:

5.1 Incremental Model

Explanation:

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.



Merits:

- Natural way. Like we do. drafts 1, 2 etc and then the final doc
- Even major software comes in a similar manner.

- We can put the first core increment for testing while we are still developing the other features. this we'll get more user reviews.(because we'll have more time)

- Advised by Roger Pressman: This is good when meeting deadlines is difficult

Demerits:

- We'll have to go through the stages again and again.
- Coding can be difficult because finding the right place to fit in the new code would be cumbersome.
- The same activity might take more time and in that case we could be in a tight spot meeting deadlines

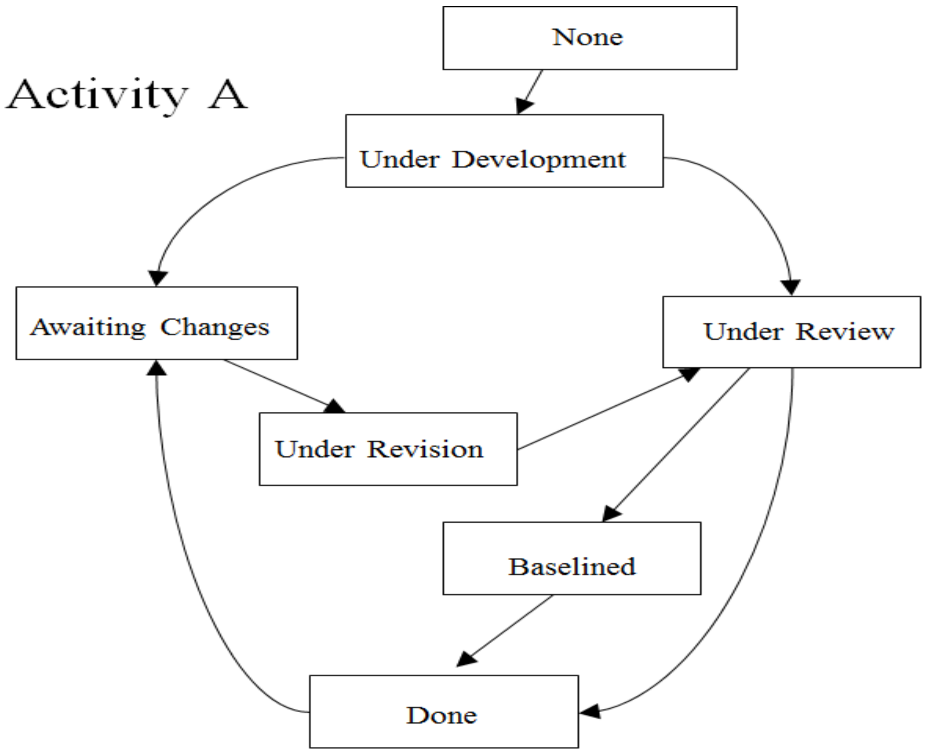
5.2 Concurrent Development Model

Explanation:

The concurrent development model, sometimes called concurrent engineering. The concurrent process model can be represented schematically as a series of major technical activities, tasks, and their associated states. For example, the engineering activity defined for the spiral model is accomplished by invoking the following tasks: prototyping and/or analysis modeling, requirements specification and design.

The activity-analysis-may is in any one of the states noted at any given time. Similarly, other activities (e.g. design or customer communication) can be represented in an analogous manner. All activities exist concurrently but reside in different states. For example, early in a project the customer communication activity has completed its first iteration and exists in the awaiting changes state. The analysis activity (which existed in the "none" state while initial customer communication was completed) now makes a transition into the under development state. If, however, the customer indicates that changes in requirements must be made, the analysis activity moves from the under development state into the awaiting changes state. The concurrent process model defines a series of events that will trigger transitions from state to state for each of the software engineering activities.

Any activity of the project can be in any of the following states at a time:



<u>Merits</u>

- It's flexible – the number incremental releases can be determined by the project team
- Immediate feedback from testing
- New features can be added late in the project
- No surprises during formal validation because testing has been continuous.
- Usually used for a client and server type application (like our project is).
- Product can be turned over fairly quickly.

<u>Demerits</u>

- The SRS must be continually updated to reflect changes.
- It requires discipline to avoid adding too many new features too late in the project.
- Possible miscommunication between different developing parties.

5.3 Spiral Model

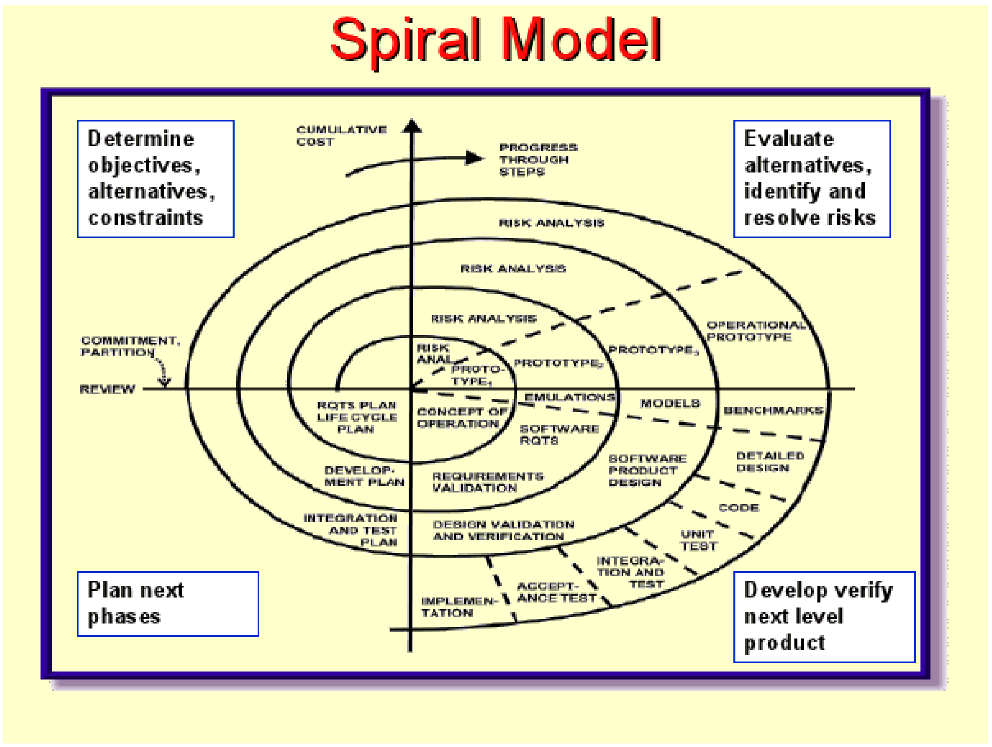
Explanation:

Spiral model is an evolutionary version of incremental prototyping. Each iteration of the prototype represented as a cycle in the spiral. The Spiral software development model is risk-oriented.

Four Quadrants of Spiral Model:

1. Identify objectives of the product and identify alternative solutions.
2. Evaluate alternative solutions. Identify potential risks. Resolve risks by building prototype
3. Develop next level of product. Verify this product
4. Evaluation of the product by customer.

The radial dimension represents the cumulative cost incurred in accomplishing the steps done so far.
The angular dimension represents the progress made in completing each cycle of the spiral.



Each cycle in the spiral begins with the identification of objectives for that cycle and the different alternatives are possible for achieving the objectives and the imposed constraints.

Merits:

- Introduces risk management- High amount of risk analysis.
- Good for large and mission-critical projects.
- Software is produced early in the software life cycle
- Prototyping controls costs- Estimates (i.e. budget, schedule, etc.) get more realistic as work progresses, because important issues are discovered earlier.
- It is more able to cope with the (nearly inevitable) changes that software development generally entails.
- Early and frequent feedback from users- Software engineers can get their hands in and start working on a project earlier.
- Release builds for beta testing

Demerits:

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.
- Time spent for evaluating risks too large for small or low-risk projects.
- Lack of risk management experience

CONCLUSION:

We worked and discussed the various development models and as per our project functioning and currently going work, modularization ahead and final completion we decided to adopt the “Incremental Model”.

We thought of 3 models in the end- Spiral, Incremental and Concurrent.

Rejection reason for spiral model:

The basic stress of a spiral model is on Risk management and analysis, which is not a major activity of our work. We aim to develop an application and website, for which risk management is not a major issue by now. So we rejected this model which requires a lot of effort on testing and risk analysis.

Rejection reason for concurrent model:

It was not clear how to integrate various parallel activities in the concurrent model, and it would get tough to integrate the 3 parallel works of Android app development, server side implementation and website development in the end. So we decided to reject this model for this reason.

Reason for taking up Incremental Model:

This is a combination of iterative and prototyping model, which allows us to do parallel execution of work with the flexibility to go back in phases in case of any problems. Also it helps to develop projects in a version form, whereby we have some version completely made by the deadline in case of problem in adding many better functionalities.