IT314 Software Engineering Team 7

SLDC Document

Version 1.0

05 February, 2013

Winter 2012-13
DA-IICT, Gandhinagar

Document Revision History:

Version	Author(s)	Description	Reviewer(s)	Date
1.0	Akash, Manjeet, Sumit	SDLC Document	Nitish, Prashant	05 February 2013

Table of Contents

1. Software Development life cycle

It is a series of identifiable phases that software undergoes during its lifetime. The phases primarily include:

- Life cycle phases:
  - Feasibility
  - Requirements specifications
  - Design
  - Coding & Unit testing
  - Integration
  - Testing
  - Maintenance

2. Software Development Life Cycle (SDLC) models

An SDLC model is a framework that describes the activities performed at each stage of a software development project both how and when.

For e.g. Classical Waterfall Model, Iterative waterfall model, Prototyping model, Spiral model, Rapid Application Development (RAD) model etc.

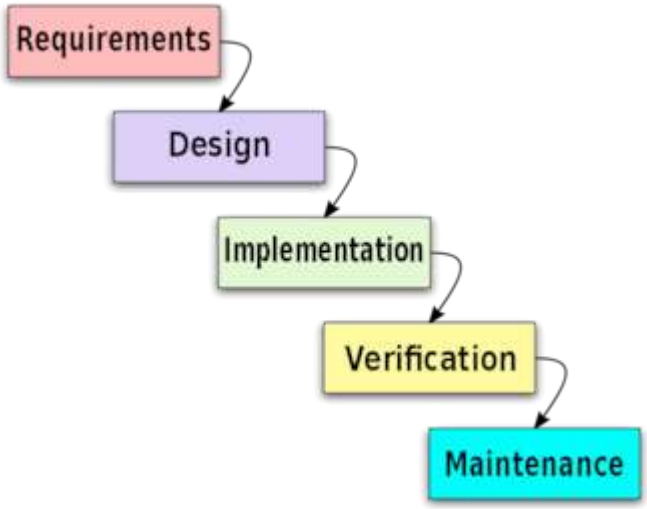
Q. Why do we use SDLC Models?

We need an SDLC because defining life cycle model encourages development of software in *systematic* and *disciplined* manner.

2.1 The Classical Waterfall Model

- Introduction

  Also called as 'linear sequential' model, this model is applied mainly when the Requirements of a Problem are reasonably well understood. This model suggests a systematic, sequential approach to software development that begins with Customers Specifications Requirement and progresses through planning, modelling, construction and deployment. Software Development team can't jump to any phase unless they have completely gone through each and every part of the previous phases.



- Merits

  a. It brings discipline to the Software development process, which was lacking in earlier Software development processes.
  b. Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
  c. Project management, both at internal level and client's level, is easy because of visible outputs after each phase. Deadlines can be set for the completion of each phase and evaluation can be done from time to time, to check if project is going as per milestones.

The Waterfall model is the oldest paradigm of Software Engineering. However as the variety of projects increased, this model was unable to cater to all the needs of the projects. The Demerits section considers what went wrong.

- Demerits

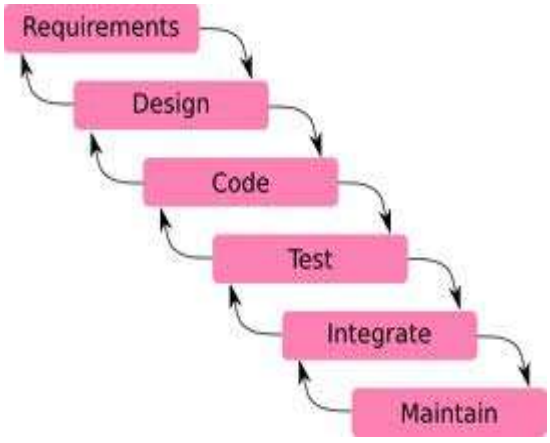
a. Real Projects rarely followed the sequential flow that the model proposes. As a result, changes caused confusion as the project progressed.
b. It is often difficult for the customer to state all the requirements explicitly mostly. The waterfall model requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of the project.
c. The customer must have patience. This model often produces the working version of the programs at the very end, i.e. late in the project time span. A major blunder if undetected, until the working program is reviewed, can be disastrous for the teams' reputation. This can easily happen because our experience with the activities of each phases and knowing explicitly the work which is to be done may lead us to overlook some not so obvious but serious issue.
d. This model leads to "blocking states" in which some project team members must wait for other members of the team to complete dependent tasks. In most of the cases, the time spent waiting exceeds the time spent on productive work.

2.2 Iterative Waterfall Model

- Introduction

This model differs from classical waterfall model primarily on the basis of 'Prior Knowledge'. The rigor with which verification and validation at the end of each phase is done is a bit more lenient than classical waterfall model. At any point in time we easily revert back to the previous phases if we find error. It is most widely used model for software development.

- Merits



a. It dramatically reduces rework.
b. Works well for smaller projects where requirements are very well understood/stable
c. It has the potential to revert back to phases and correct then, in case an error is detected.

- Demerits

a. This process can become time consuming and costly.
b. Little opportunity for customer to preview the system.

2.3 Prototype Model

- Introduction

In this model, 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.

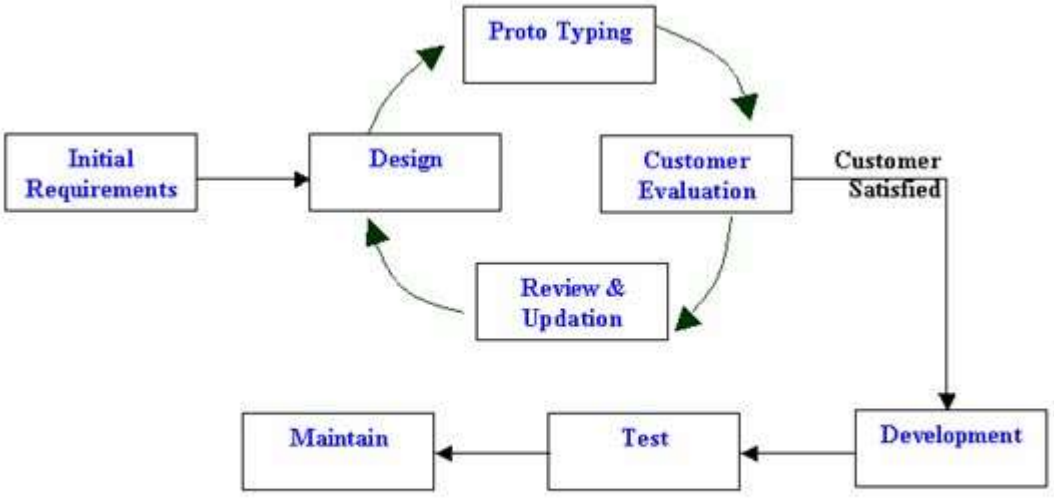
Prototype paradigm begins with requirements gathering. Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory.

A quick design occurs which leads to the construction of prototype. The prototype is evaluated by the customer/user and used to refine the requirements for the software to be developed.

Iteration occurs as the prototype is tuned to satisfy the user requirements, while at the same time enabling developer to better understand what needs to be done.

The prototype can be tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done. This model is very helpful in bringing the client and the developer onto the same track. Usually this model is followed when the client is not computer savvy.

It is more commonly used as a technique that can be implemented within the context of any of the software development process models.



Proto Type Model

- Merits

  a. Users get a sense of the final system in the early phase.
  b. Prototype helps the client visualizing the final product.
  c. As mentioned earlier, this model is very helpful in bringing the client and the developer onto the same track.
  d. It emphasizes clarity in requirements. It is a way to confirm that the image formed by the client on the given set of requirements is same as the image formed by the Developer about the product.
  e. It is especially useful for GUI (Graphic User Interface) development.

- Demerits

  a. Customer wrongly assumes the prototype to be the working version of the software, but in reality, in rush to get it work, the team doesn't considers overall software quality and maintainability.
  b. The developer compromises implementation, in order to get the prototype working quickly.
  c. This model may become risky if much time is devoted only in redesigning the prototypes.

2.4 Rapid Application Development (RAD)

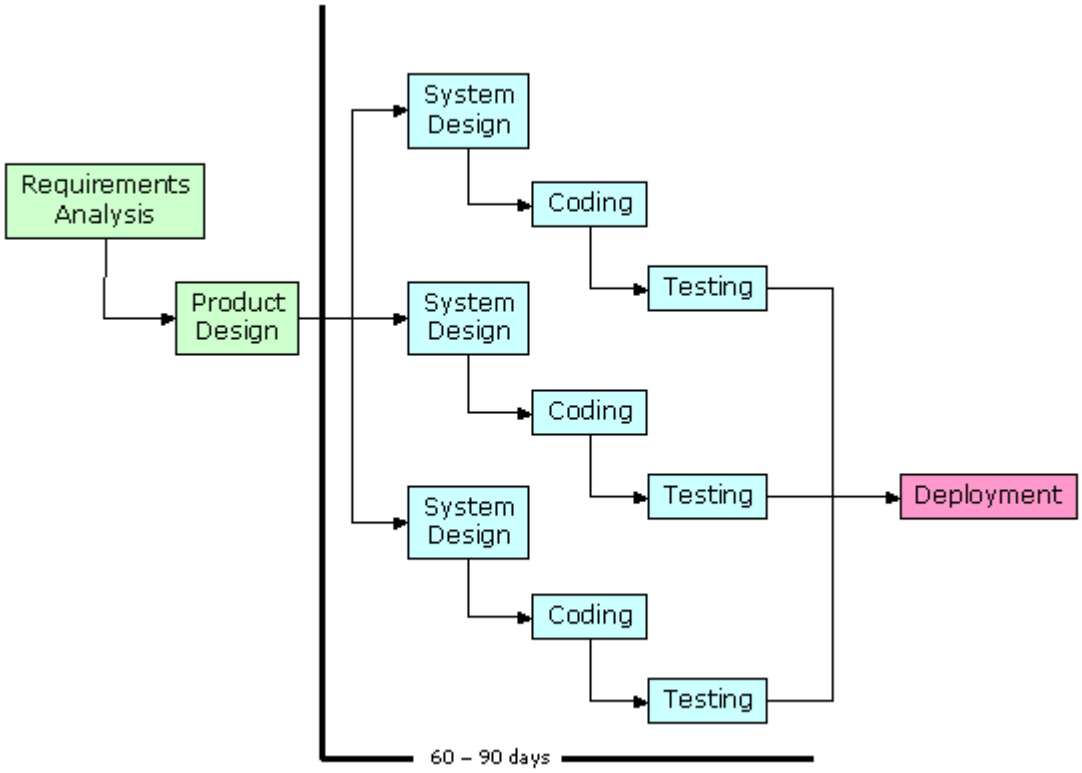
- Introduction

  This is a high speed adaption of the waterfall model in which **rapid development** is achieved by using a component based construction approach. Initial phase aims at understanding the business problem and information characteristics that the software must accommodate (much like any other model).

  Planning phase: It is essential and involves scheduling of multiple software teams working in parallel on different system functions.

  Design phase: It encompasses representations that serve as basis for RAD's development phase.

  Development phase: This phase emphasizes the use of pre-existing software components and application of automated code generation.



- Merits

  a. RAD emphasizes reuse many of the program components have already been tested, which minimizes the testing and development time.
  b. It is a software process model that emphasizes a short development cycle, on well understood requirements and constrained project scope.

- Demerits

a. Requires a system that can be modularized. If a system cannot be properly modularized, building the components for the RAD will be problematic.
b. If high performance is an issue, and performance is to be achieved through tuning the interfaces to system components, the RAD approach may not work.
c. RAD may not be appropriate when technical risks are high (e.g. when a new application makes heavy use of new technology.)
d. Modules developed using the previously developed components in RAD inherit the reliability of the component. So if the component is unreliable, the produced module will also be unreliable.

2.5 Evolutionary Model

- Introduction

This model is also known as SUCCESSIVE VERSIONS model. It involves breaking a system down into several modules (or functionalities) such that these modules can be delivered in an incremental fashion. Business and product requirements often change as development proceeds, making a straight-line path to an end product unrealistic. Tight market deadlines make completion of a 'comprehensive' software product impossible, but a limited version must be introduced to meet competitive or business pressure; a set of core products or system requirements is well understood, but the details of product or system extensions have yet to be identified. These issues are addressed by Evolutionary models sometimes called as 'successive versions' model.

Evolutionary models are thus iterative in nature.

Operationally it involves-

1) Breaking the system down into several modules (or functionality) such that,
2) These modules can be delivered in incremental fashion.

Developer initially develops the core module and then refines it by incrementally adding new functionalities. Important thing to note is that **each successive version** of the product **is a working version**.

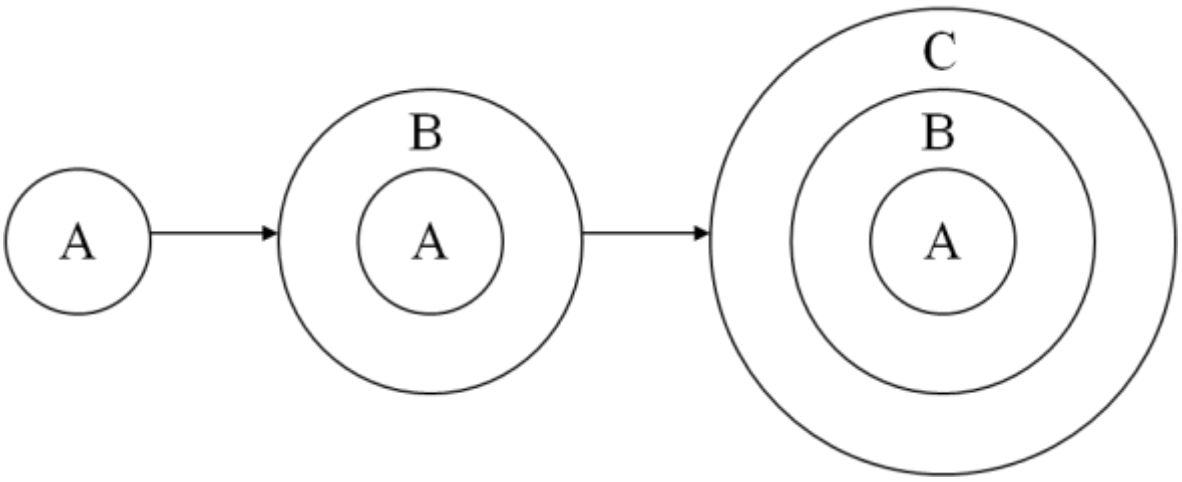


Figure 5a: Evolutionary Model (in terms of Functionality)

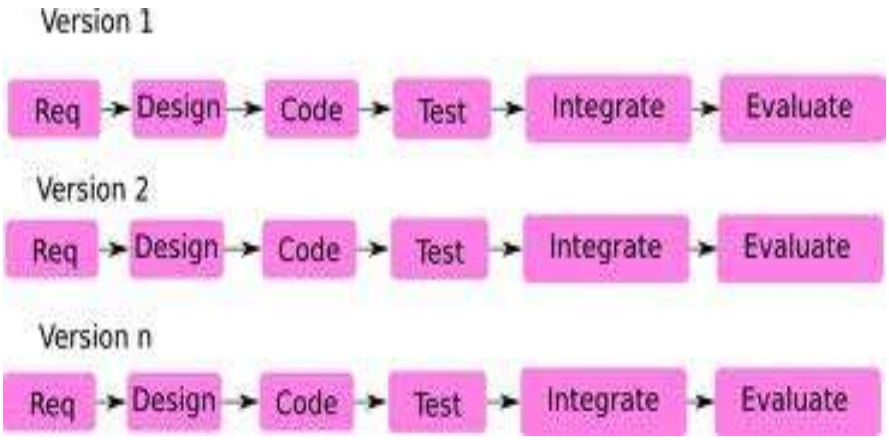


Figure 5b: Evolutionary Model (phase wise), Note: no time scale shown, Different versions can go in parallel or sequentially in time

- Merits:

  a. Users get an opportunity to experiment /use the partial system much before the fully developed version is released.
  b. Helps in eliciting requirements.
  c. Core module gets tested very thoroughly. (since it gets tested at the time of each release)

d. Entire resource requirements need not be committed to the project at the same time.

- Demerits

a. It is difficult to break down a system into functional units that can be implemented in an incremental fashion.

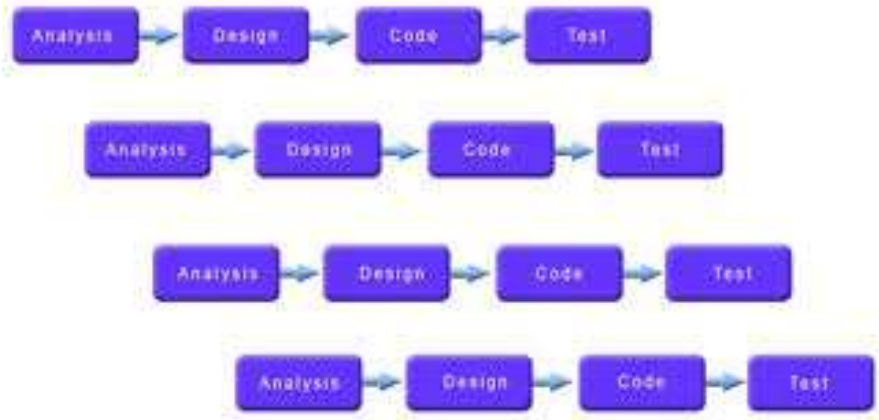
2.6 Incremental Model

- Introduction

Incremental model combines waterfall model applied in an iterative fashion. It applies linear sequences of waterfall model in a 'staggered' fashion as the 'calendar' time progresses. Each linear sequence produces a deliverable termed 'increment' of the software.

The process is repeated following the delivery of each increment until the complete product is delivered.

The first increment is the *core* product which addresses the basic requirements. That is, the supplementary features (some known, others unknown) remain undelivered in *core* product. The core product is used by the customer (undergoes detailed evaluation), as a result of use 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 delivery of additional features and functionality.



- Merits

  a. This model is useful when staffing is unavailable for a complete implementation by the business deadline of the project

  b. Early increments can be implemented by fewer people, and depending on the acceptability of the core product, more resources can be added to implement subsequent stages.

  c. 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

- Demerits

  a. Agreement on the core product is not easy.

  b. The process can be time consuming.

  c. Depends on the client agreeing for it. The client may not like the idea of getting piecewise solutions to his problem. The client may lose 'faith' in the increment output delivered.

  d. Clear and complete definition of the whole system is needed before it can be broken down and built incrementally.

  e. Total cost is higher than waterfall.

2.7 The Concurrent Development Model

- Introduction

  The Concurrent Development model sometimes called concurrent engineering, can be represented as a series of framework activities, software engineering actions and tasks and their associates "**STATES**".

  Any of the activities can be in one of the following "States":

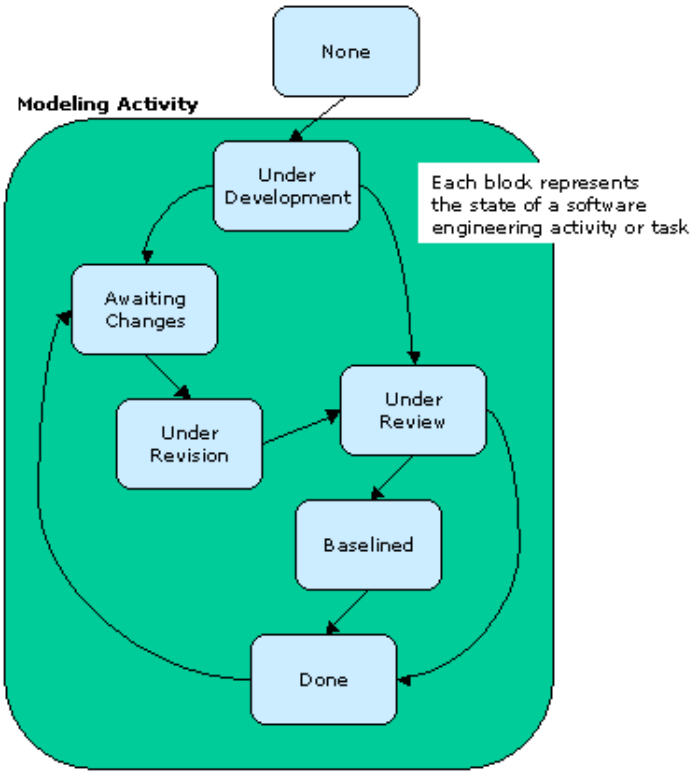
  1. None
  2. Under Development
  3. Awaiting Changes
  4. Under Revision

5. Under Review
6. Baselined
7. Done

For ex: The activity modelling may be in any one of the states noted at any given time. Similarly other activities or tasks (communication or construction)can be represented in an analogous manner. All activities exist concurrently but reside in different states. For ex: Early in a project the communication activity has completed its first iteration and exists in "**awaiting changes"** state. The modelling activity which existed in **none** while initial communication was completed now makes a transition into the **Under Development** state. If however, the customer indicates that changes in requirement must be made, the modelling 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, actions or tasks. For ex: During early stages of "*Design*", an inconsistency in the analysis model is uncovered. This generates the event **analysis model correction** which will trigger the analysis action from the **done** state into the **awaiting changes** state.

Rather than confining Software engineering activities, actions or tasks to a sequence of events, it defines a network of activities, actions and tasks. Each activity .action or task on the network exists simultaneously with activities, actions or tasks. Events generated at one point in the process network trigger transitions among the **States**.



- Merits
  a. The concurrent model is applicable to all types of Software Development and provides an accurate picture of the current state of the project.
  b. It's flexible – the number incremental releases can be determined by the project team.
  c. Immediate feedback from testing
  d. New features can be added late in the project
  e. No surprises during formal validation because testing has been continuous.

- Demerits
  a. The SRS must be continually updated to reflect changes.
  b. It requires discipline to avoid adding too many new features too late in the project.

2.8 The Spiral Model

- Introduction:

  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.

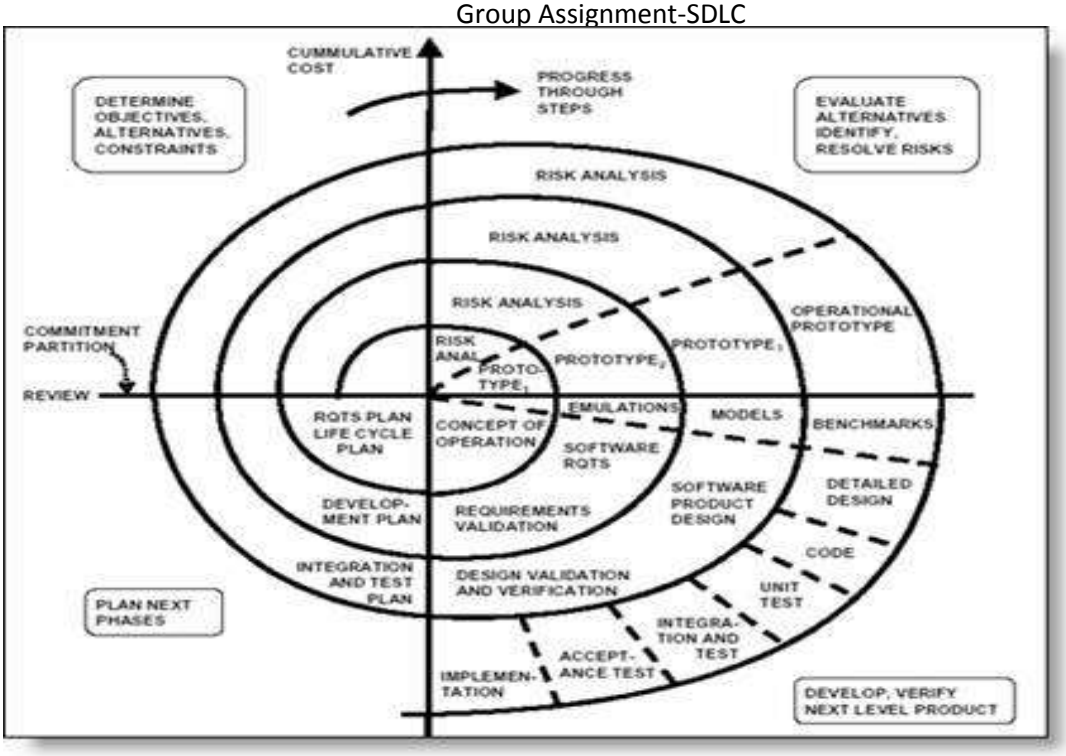


Figure 8: The Spiral Model

- Merits:

  a. Introduces risk management- High amount of risk analysis.

  b. Good for large and mission-critical projects.

  c. Software is produced early in the software life cycle

  d. Prototyping controls costs- Estimates (i.e. budget, schedule, etc.) get more realistic as work progresses, because important issues are discovered earlier.

  e. It is more able to cope with the (nearly inevitable) changes that software development generally entails.

  f. Early and frequent feedback from users- Software engineers can get their hands in and start working on a project earlier.

  g. Release builds for beta testing.

- Demerits

  a. Can be a costly model to use.
  b. Risk analysis requires highly specific expertise.

c. Project's success is highly dependent on the risk analysis phase.
d. Doesn't work well for smaller projects.
e. Time spent for evaluating risks too large for small or low-risk projects.
f. Lack of risk management experience.

3.0 Conclusion: Our Preference

Each SDLC model has its own certain basic qualitative and quantitative resource requirements that are appropriate for different projects.

After a thorough analysis and taking under consideration the merits and demerits of various SDLC models, we infer that for our project, which aims at redesigning the 'Entelechy' website, we shall go with 'The Evolutionary Model'. This model helps us divide our work into distinct modules and add them successively to our product, which is analogous to our project in the way that our project consists of different modules, in the form of plug-ins and themes to be build on Wordpress platform.

Another factor is that our requirements are very clear and stable so the challenge of managing the project in simultaneous phases becomes easier.

The reviews and appraisals obtained from the various stakeholders via survey analysis and personal interviews with the concerned authorities led us to properly and clearly identify our requirements. This will help us in assembling the System Requirement Specifications (SRS) document with intensive rigor, which will clearly make a demarcation line between the requirement phase and the design phase.

Our project emphasizes planning in early stages to ensure design flaws before development. In addition, its intensive document and planning make it work well for projects in which quality control is a major concern. So, opting for evolutionary model reduces the risk involved in design and implementation phases.

Sequential and Systematic execution is desired by the team: Not all members of our team are at the same level of skills related to development. So, an evolutionary model will help our non-experienced members to learn quickly since it will give them a chance to work hands-on on the platform with the experienced members.