# Chapter 2

- ***Software Process*:** Already Explained in Chapter 1.
- ***SDLC:***-

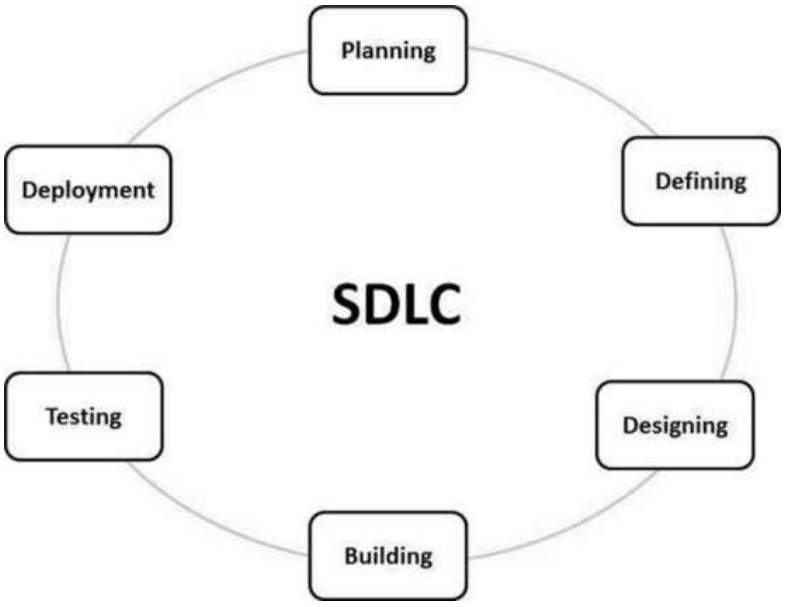
SDLC is the acronym of Software Development Life Cycle.

It is also called as Software Development Process.

SDLC is a framework defining tasks performed at each step in the software development process.

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC.




A typical Software Development Life Cycle consists of the following stages −

## Stage 1: **Planning and Requirement Analysis**

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

## Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an **SRS (Software Requirement Specification)** document which consists of all the product requirements to be designed and developed during the project life cycle.

## Stage 3: Designing the Product Architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

## Stage 4: Building or Developing the Product

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

## Stage 5: Testing the Product

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

### Stage 6: Deployment in the Market and Maintenance

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

Following are the most important and popular SDLC models followed in the industry &miuns;

- Waterfall Model
- Iterative Model
- Spiral Model
- V-Model
- Big Bang Model

## ➢ Process Models:

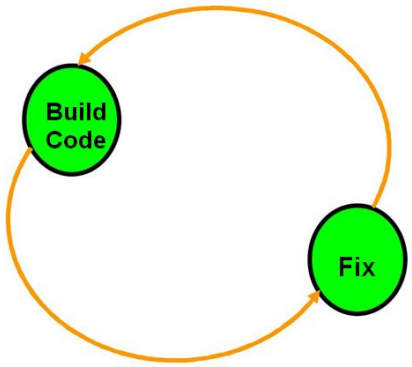
The goal of Software Engineering is to provide models and processes that lead to the production of well-documented maintainable software in a manner that is predictable.

"The period of time that starts when a software product is conceived and ends when the product is no longer available for use. The software life cycle typically includes a requirement phase, design phase, implementation phase, test phase, installation and check out phase, operation and maintenance phase, and sometimes retirement phase".

## Build & Fix Model

1. Product is constructed without specifications or any attempt at design
2. Adhoc approach and not well defined
3. Simple two phase model
4. Suitable for small programming exercises of 100 or 200 lines
5. Unsatisfactory for software for any reasonable size
6. Code soon becomes unfixable & unenhanceable

7. No room for structured design
8. Maintenance is practically not possible

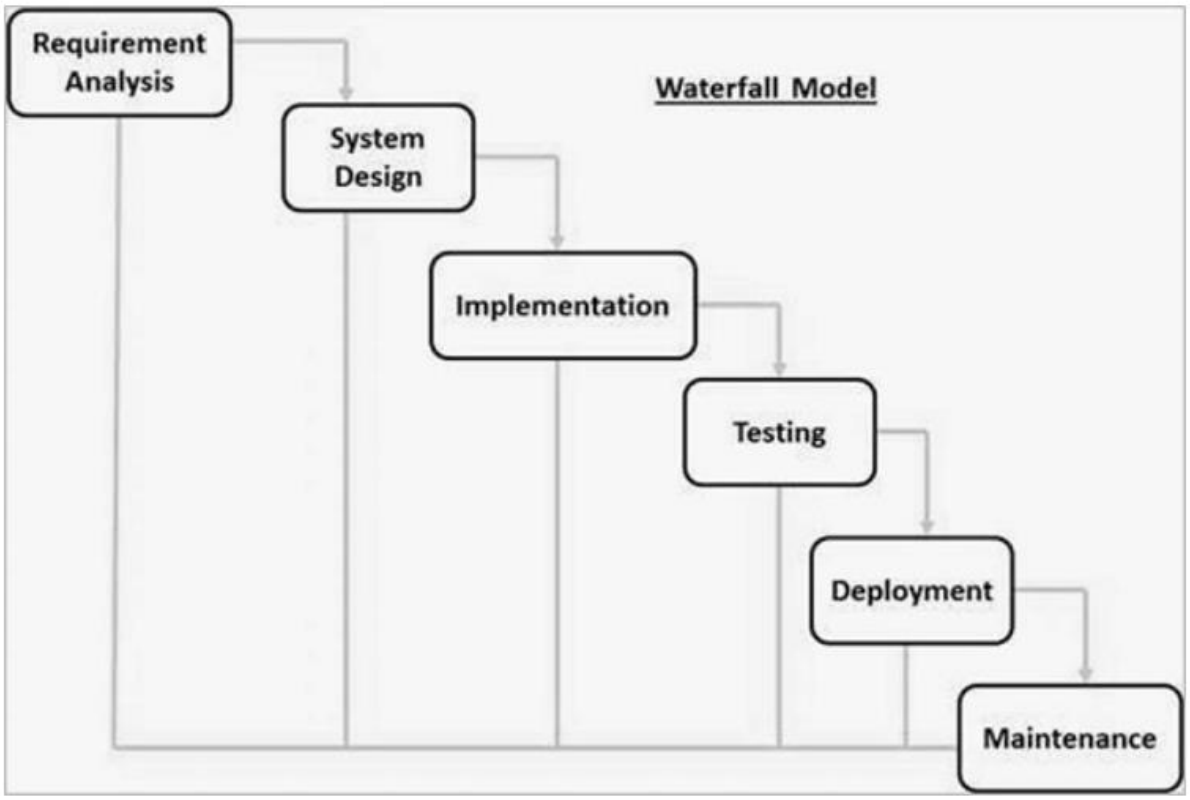



## ➢ Waterfall Model

This model is easy to understand and reinforces the notion of "define before design" and "design before code".

The model expects complete & accurate requirements early in the process, which is unrealistic. Sometimes it is also called classic Lifecycle Model

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.




The sequential phases in Waterfall model are −

- **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- **System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- **Implementation** − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
- **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of system** − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

**Problems of waterfall model**

i. It is difficult to define all requirements at the beginning of a project

ii. This model is not suitable for accommodating any change

iii. A working version of the system is not seen until late in the project's life

iv. It does not scale up well to large projects.

v. Real projects are rarely sequential.

**Pros**

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

**Cons**

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.

- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

## ➢ Evolutionary Process Models

- These models are basically iterative.
- Once the requirements are analysed, they pass through a series of iterations till the complete software is developed.
- The evolutionary model mainly support the programmer to develop the complete version of a software.
- After each release, based on the review given by the reviewer, further iterations are performed.

The main two evolutionary models are

1. Increment model
2. Spiral model

### 1. Increment Process Model

They are effective in the situations where requirements are defined precisely and there is no confusion about the functionality of the final product.

After every cycle a useable product is given to the customer.

Popular particularly when we have to quickly deliver a limited functionality system.

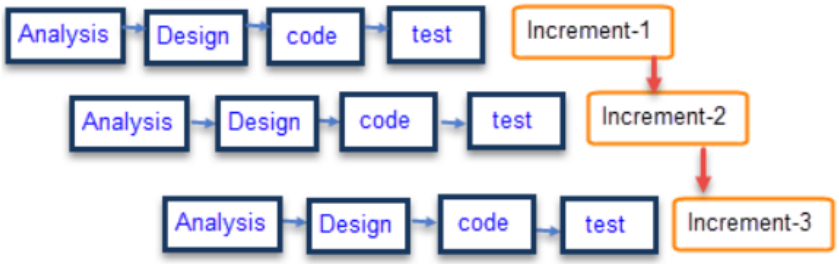
This model has the same phases as the waterfall model, but with fewer restrictions. Generally the phases occur in the same order as in the waterfall model, but they may be

conducted in several cycles. Useable product is released at the end of the each cycle, with each release providing additional functionality.

Customers and developers specify as many requirements as possible and prepare a SRS document.

Developers and customers then prioritize these requirements

Developers implement the specified requirements in one or more cycles of design, implementation and test based on the defined priorities.

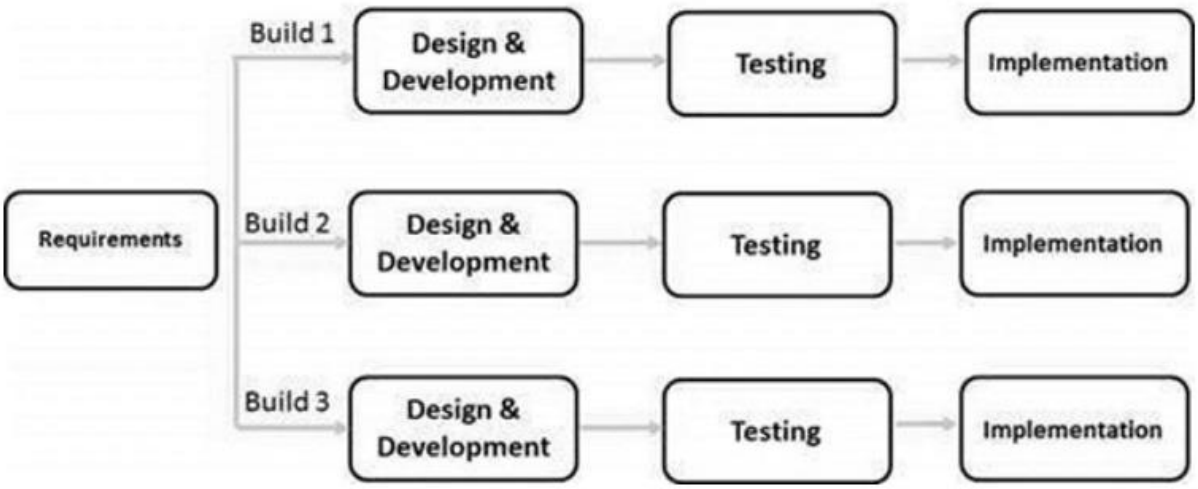



Incremental Model

Incremental model is of two types :-

1.**Iterative Enhancement Model**:-

In the Iterative model, iterative process starts with a simple implementation of a small set of the software requirements and iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed.

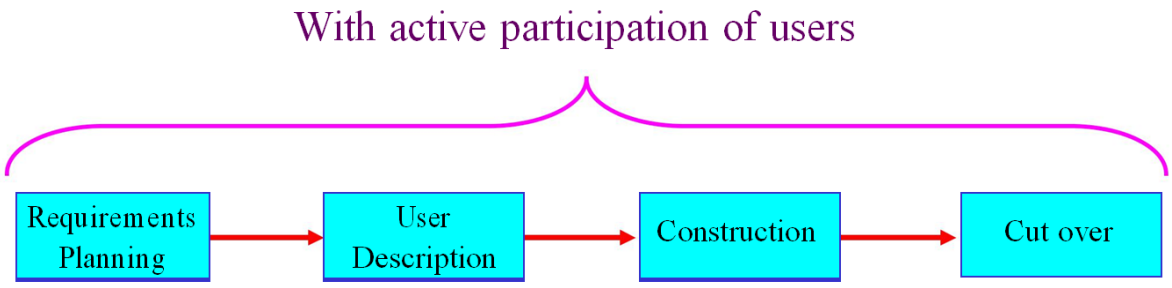



## 2.**Rapid application Development Model**

Build a rapid prototype

Give it to user for evaluation & obtain feedback

Prototype is refined




1. Requirement Planning:-Plan the requirements.
2. User description:- joint team of developers and users constituted to prepare, understand and review the requirements.
3. Construction phase:- This phase combines the detailed design, coding and testing phase of waterfall model. Here,we release the product to customer.
4. Cutover phase:- This phase incorporates acceptance testing by the users, installation of the system and user testing.

**Pros of Incremental Model**

- 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.
- In this model customer can respond to each built.
- Lowers initial delivery cost.

**Cons of Incremental Model**

- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.

## 2. Spiral Model

Spiral model is one of the most important Software Development Life Cycle models, which provides support for Risk Handling. In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a Phase of the software development process. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using spiral model.

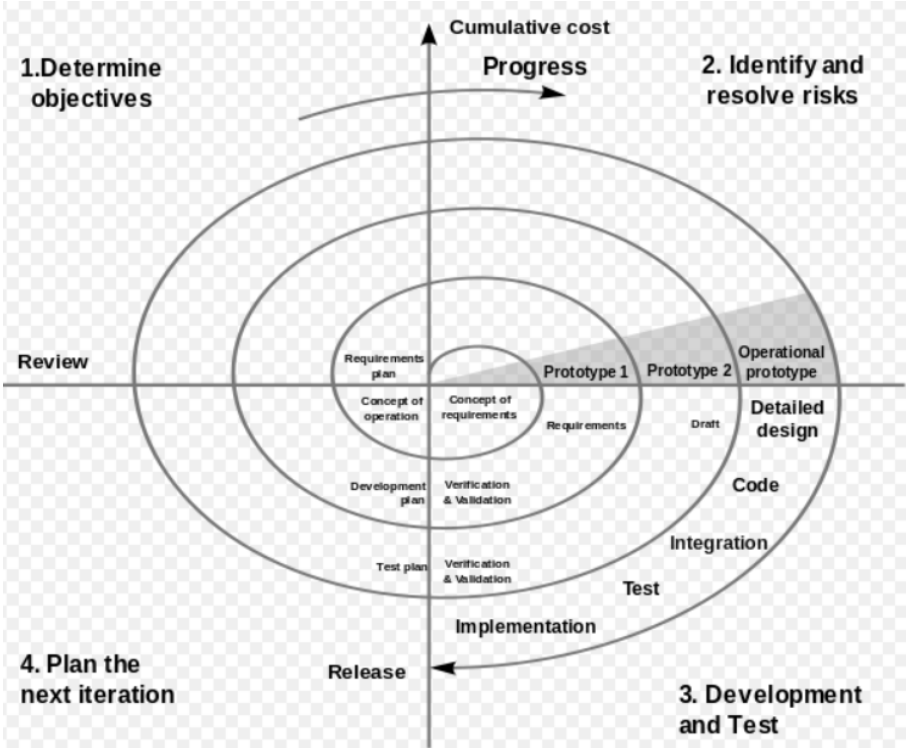
The Radius of the spiral at any point represents the expenses(cost) of the project so far, and the angular dimension represents the progress made so far in the current phase.
Each phase of Spiral Model is divided into four quadrants as shown in the above figure. The functions of these four quadrants are discussed below-

1.**Objectives determination and identify alternative solutions**: Requirements are gathered from the customers and the objectives are identified, elaborated and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.
2.**Identify and resolve Risks**: During the second quadrant all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution is identified and the risks

are resolved using the best possible strategy. At the end of this quadrant, Prototype is built for the best possible solution.

3.**Develop next version of the Product**: During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.

4.**Review and plan for the next Phase**: In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.




## Pros:-

- Changing requirements can be accommodated.
- Allows extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

## Cons:-

- Management is more complex.

- End of the project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go on indefinitely.
- Large number of intermediate stages requires excessive documentation.