Unit-1

Software Development Process

❖ SOFTWARE

(IEEE Definition of software): Software is a "Collection of computer programs, procedures, rules, associated documents and concerned data with the operation of data processing system."

It also includes representations of pictorial, video, and audio information.

Software is divided into two broad categories:

System software: It is responsible for controlling and integrating the

hardware components of a system.

Hence, the software and users can work with them. It also controls the peripherals of computer like monitors, printers, storage devices etc.

Example: Operating system

Application software: It is used to accomplish some specific tasks. It should be collection of small programs. It is a program or group of programs generally designed for end users.

Example: Microsoft word, Excel, Railway reservation system etc.

Computer hardware is built from peripherals, device or components, while computer software is logical rather than physical.

Software has following distinct characteristics.

▪ Software characteristics:

Different software characteristics decide whether the software is good or bad. These attributes reflect the quality of a software product. And these are depended on the application of the software also.

For example, a banking system must be secure while a telephone switching system must be reliable.

Following are the characteristics of good software: (Qualities of good software).

Understandability: Software should be easy to understand, even to novice users. It should be efficient to use.

Cost: Software should be cost effective as per its usage.

Maintainability: Software should be easily maintainable and modifiable in future.

Modularity: Software should have modular approach so it can be handled easily for testing.

Functionality: Software should be functionally capable to meet user's requirements.

Reliability: It should have the capability to provide failure-free service.

Portability: Software should have the capability to be adapted for different environments.

Correctness: Software should be correct as per its requirements.

Documentation: Software should be properly documented so that we can re-refer it in future.

Reusability: It should be reusable, or its code or logic should be reusable in future.

Interoperability: Software should be able to communicate with various devices using standard bus structure and protocol.

Verifiability: Software should be verifiable with its properties and functionalities with its planning and analysis done in previous phase.

Along with these characteristics, software has some special characteristics which are explained below.

- **Software doesn't wear out:**

This can be well understood with the help of figure given below.

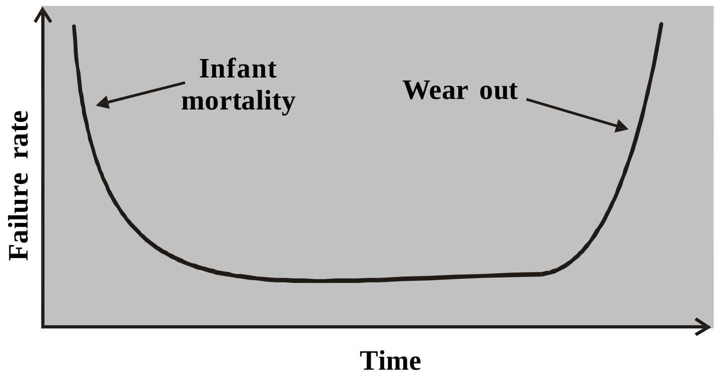


Figure: hardware failure curve

In above figure, the relationship between time and failure called "bath-tub curve". It indicates that hardware has relatively high failure rates early in its life, defects are corrected and the failure rate drops to a steady-state level for some period of time. As time passes, however, the failure rate rises again as hardware components suffer from the effects of dust, vibration, abuse, temperature extremes, and many other environmental factors. So simply, we can say hardware begins to wear out.

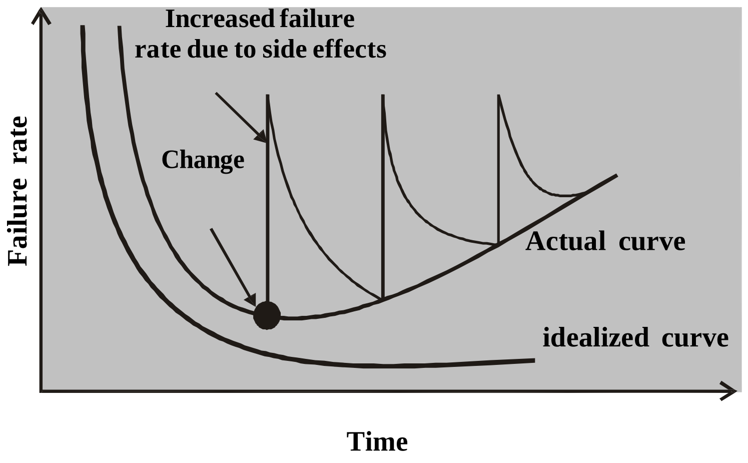


Figure: Software failure curve

Above figure shows the software failure rate.

- Software is not highly affected by environmental effects. The "idealized curve" shows software failure.
- In the early stage, due to lot many errors, software could have high failure. But it becomes reliable as time passes instead of wearing out. Once software is made it has a longer life span that we can see in idealized curve.
- In actual curve (above figure), we can see that software may have increased failure rate as it may become old as and when the new development environment changes. Spike in the curve is due to chance of maintenance and side effects.

- Software may be retired due to new requirements, new expectations, new technology new technologies etc. Hence, software doesn't wear out, but it may get worse.

▪ **Software is engineered, not manufactured like hardware:**

Once a product is manufactured, it is not easy to modify or change it. While in case of software we can easily change or modify it for later use. Even making multiple copies of software is a very easy task rather it is much more tuff in case of hardware.

In hardware, costing is due to assembly of raw material and other processing expenses while in software development, no assembly needed like hardware. Hence, software is not manufactured as it is developed or it is engineered.

▪ **Reusability of components:**

- If we assemble hardware, we will have every part and component from different vendors and then we may produce a finished product.
- In case of software, every project is a new project and we have to start from scratch and design every unit of software product. Huge number of efforts required to develop a software system. So, building a standard code or design is very useful for making many new projects. These codes are reusable.
- We can reuse software codes, modules or any logical components in any other related software projects. (For example, we prepared a payroll system for any organization. We can reuse some of the logical components while we are developing the related types of payroll systems for any other company or organizations.)
- Generally, GUI (graphical user interface) software is built using reusable components.

▪ **Software is flexible for custom built:**

- A software program can be developed to do anything. Any kind of change needed in software can be done easily.
- A software program or product can be built on user requirements basis or custom built. Even the developed software product can also be changed as per the user demands.
- We can say software is very much flexible for custom built rather than hardware.
- Hence, now a days industries are moving toward component-based assembly, software continues to be custom built.

❖ SOFTWARE ENGINEERING

- **Definition:**

Software Engineering discipline began since 5 decade and provides solution to software crisis.

Software crisis is the difficulty of writing useful and efficient computer program in the required time.

- ***Software Engineering is an engineering discipline that delivers high quality software at agreed cost & in planned schedule.***
- Three main aspects of Software Engineering are:
  - Provide quality product
  - Expected cost
  - Complete work on agreed schedule

(IEEE Definition) Software Engineering is the application of a systematic, disciplined and quantifiable approach to the development, operation and maintenance of software.

- **Software Engineering - A layered approach:**

Software engineering can be viewed as a layered technology.

It contains process, methods and tools that enable software product to be built in a timely manner.

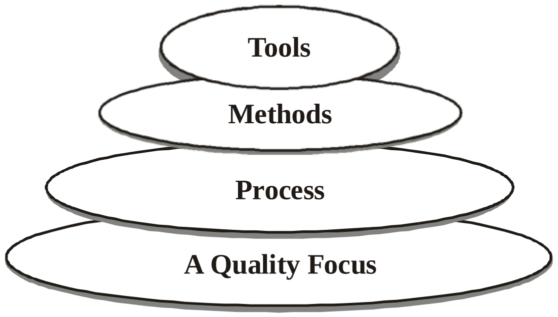


Figure: Software Engineering Layers

As we can see in the above figure, this layered approach contains mainly four layers.

1. **A quality focus layer:**
   - Software engineering mainly focuses on quality product.
   - It checks whether the output meets with its requirement specifications or not.
   - Every organization should maintain its total quality management (TQM).
   - This layer supports software engineering.
2. **Process layer:**
   - Software process is a set of activities together if ordered and performed properly, the desired result would be produced.
   - Main objective of this layer is to develop software in time.
   - This layer is the heart of software engineering.
   - It holds all the technology layers together like GLUE.
   - It is also working as foundation layer.
   - It defines the framework activities.
3. **Method layer:**
   - It provides technical knowledge for developing software. It describes 'how-to' build software product.
   - It creates software engineering environment to software product using CASE tools. (CASE tools combines software, hardware and software engineering database).
   - This layer includes requirements analysis, design, program construction, testing, and support.
4. **Tools layer:**
   - It provides support to below layers using automated or semi-automated tools.
   - Due to this layer, process is executed in proper manner.

- **Need of software engineering:**

The reasons why software engineering is needed to develop software products are given below:

- To help developers to obtain high quality software product.
- To develop the product in appropriate manner using life cycle models.
- To acquire skills to develop large programs.

- To acquire skills to be a better programmer.
- To provide a software product in a timely manner.
- To provide a quality software product.
- To provide a software product at an agreed cost.
- To develop ability to solve complex programming problems.

Also learn techniques of specification, design, user interface development, testing, project management, etc.

❖ SOFTWARE DEVELOPMENT

- Software development is the process of developing software through successive phases in an ordered way.
- This process includes not only the actual writing of code but also the preparation of requirements and objectives, the design of what is to be coded, and confirmation that what is developed has met objectives or not.
- In other words, Software development is the computer programming, documenting, testing, and bug fixing involved in creating and maintaining applications and frameworks involved in a software release life cycle and resulting in a software product.
- Software can be developed for a variety of purposes.
- Three most common being for software development:
    - To meet specific needs of specific clients. (i.e., banking software)
    - To meet the need of some set of potential users. (i.e., Facebook or WhatsApp etc.)
    - To develop for personal use. (i.e., software that is built for individual use)

Software development process is a set of steps that a software program goes through when developed.

General phases of software development are:

Requirements → Analysis → Design → Implementation → Testing / Verification → Documentation → Maintenance

- First in the software development process, the requirements phase outlines the goals of what the program will be capable of doing.
- Next, the design phase covers how the program is going to be created, who will be doing what etc.
- The implementation phase is where the programmers and other designers start work on the program.
- Testing and verification step can begin to help verify the program has no errors. During the testing phase, problems found are fixed, until the program meets the company's quality controls.
- After the program's development, the documentation phase begins to describe how to use the program or product.
- Finally, maintaining (updating) the program must continue for several years after the initial release.

Importance of software development:

- Software is important to make the hardware working.
- Software is important to build up security where it needs to be done, such as, in banks, money transaction etc.
- Software is important to make a task easier, such as, distribution of products, products information etc.
- Software is important to use the power of computer or working efficiency to perform those tasks which cannot be done or controlled by human.

- Shortly, it can be said that software is important to make things easier, faster, more reliable and safer.

Generic view of software engineering

- The work associated with software engineering can be categorized into three generic phases $\rightarrow$Definition phase, Development phase and Support phase.

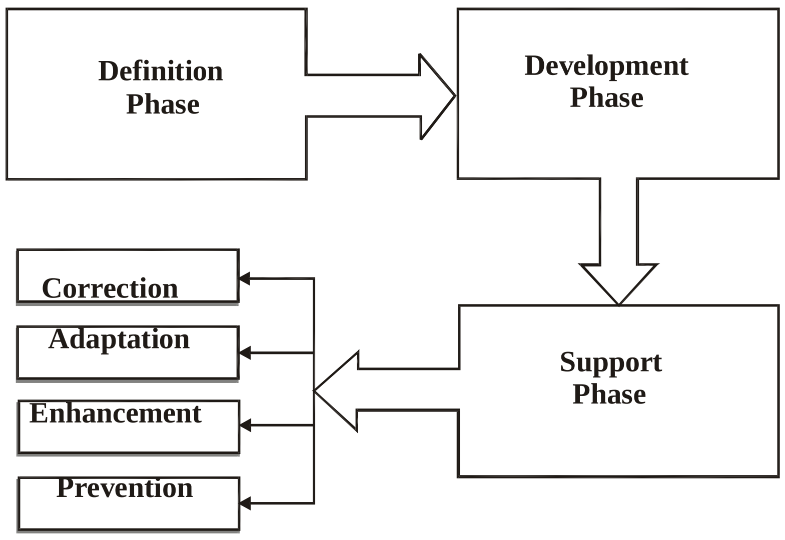


Figure: Software generic view

1. **The Definition Phase:**

- It focuses on *"what part"*. That is, during definition, the software engineer attempts to identify what information is to be processed, what function and performance are desired, what system behaviour can be expected, what interfaces are to be established, what design constraints exist, and what validation criteria are required to define a successful system.
- The key requirements of the system and the software are identified.
- Three main activities performed during this phase: system or information engineering, software project planning and requirement analysis.

2. **The development phase:**

- It focuses on *"how part"* of the development. During development a software engineer attempts to define how data are to be structured, how function is to be implemented within a software architecture, how interfaces are to be characterized, how the design will be translated into a programming language, and how testing will be performed.
- Main activities are performed under this phase are: software design, code generation and software testing.

3. **The support phase:**

- The support phase focuses on "change" associated with error correction.
- Four types of change are encountered during the support phase:
  - $\rightarrow$ **Correction:** corrective maintenance changes the software to correct defects.
  - $\rightarrow$ **Adaption:** Adaptive maintenance results in modification to the software to accommodate changes to its external environment.
  - $\rightarrow$ **Enhancement/Perfection:** Perfective maintenance extends the software beyond its original functional requirements.

→ **Prevention:** Preventive maintenance to enable the software to serve the needs of its end users.

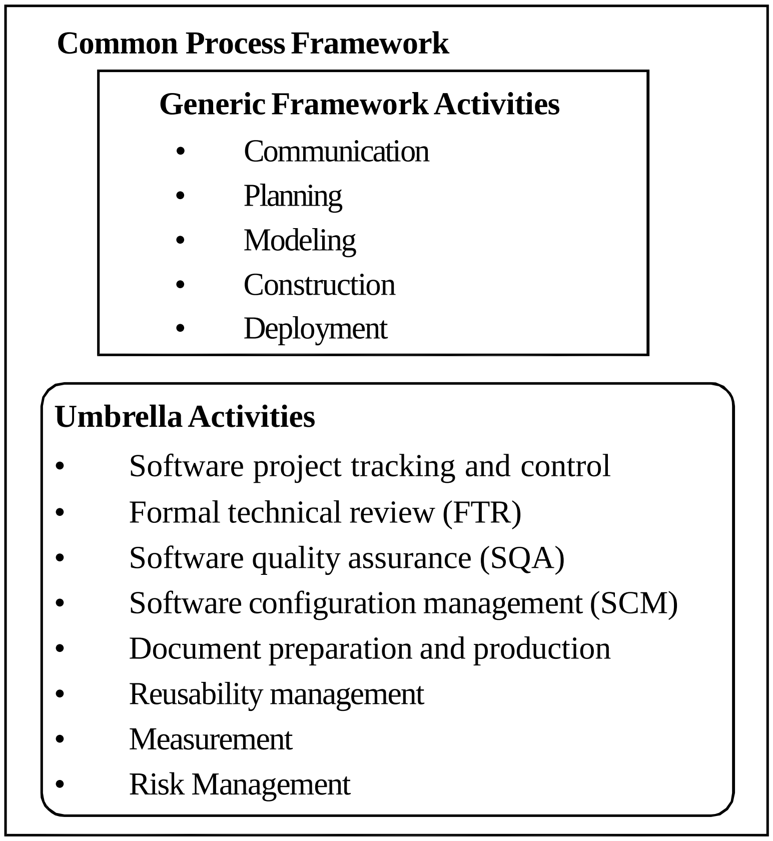
The definition phase focuses on **'what'**
The development phase focuses on **'how'**
The support phase focuses on **'change'**

❖ GENERIC (NOT SPECIFIC TO A GROUP) FRAMEWORK ACTIVITIES

A software framework provides a standard way to build deploy software product. And a software process is the set of activities and associated results that produce a software product.

Any standard software process model would primarily consist of two types of activities: **A set of framework activities,** which are always applicable to all the projects and **A set of umbrella activities** which are non SDLC activities that are applicable throughout the process.

It provides common process framework for all projects.



Common Process Framework

Generic Framework Activities

- Communication
- Planning
- Modeling
- Construction
- Deployment

Umbrella Activities

- Software project tracking and control
- Formal technical review (FTR)
- Software quality assurance (SQA)
- Software configuration management (SCM)
- Document preparation and production
- Reusability management
- Measurement
- Risk Management

Figure: Generic framework and Umbrella activities

The Adaptable Process Model (APM) defines the following set of framework activities.

Communication

The software development starts with the communication between customer and developer.

Planning

It consists of complete estimation, scheduling for project development and tracking.

Modeling

Modeling consists of complete requirement analysis and the design of the project like algorithm, flowchart etc.

Construction

- Construction consists of code generation and the testing part.
- Coding part implements the design details using an appropriate programming language.
- Testing is to check whether the flow of coding is correct or not.
- Testing also check that the program provides desired output.

Deployment

- Deployment step consists of delivering the product to the customer and take feedback from them.
- If the customer wants some corrections or demands for the additional capabilities, then the change is required for improvement in the quality of the software.

▪ **Umbrella activities:**

- The phases and related steps of the generic view of software engineering are complemented by a number of umbrella activities.
- Umbrella activities are performed throughout the process.
- These activities are independent of any framework activity.

Typical activities in this category include:

1. **Software project tracking and control**

   When project tracking and controlling done then software engineering tasks will enable to get the job done on time.

2. **Formal technical review (FTR)**

   This includes reviewing the techniques that has been used in the project.

3. **Software quality assurance (SQA)**

   This is very important to ensure the quality measurement of each part of software being developed.

4. **Software configuration management (SCM)**

   SCM is a set of activities designed to control changes made by identifying the work products that are likely to change, establishing relationships among them.

5. **Document preparation and production**

   All the project planning and other activities should be hardly copied and the production gets started here.

6. **Reusability management**

   This includes the backing up of each part of the software project they can be corrected or any kind of support can be given to them later to update or upgrade the software at user/time demand.

7. **Measurement (estimation)**

   This will include all the measurement or estimation of every aspects of the software project like: time estimation, cost estimation etc.

8. Risk management

- As we know that 'tomorrow's problem is today's risk'. Risk management is very important activity for any type of software development.
- It identifies potential problems and deal with them when they are easier to handle before they become critical.
- Risk management allows early identification of risks and provide management decisions to the solutions, and improve quality of the product.

❖ SOFTWARE DEVELOPMENT MODELS / LIFE CYCLE MODELS

- Every system has a life cycle. It begins when a problem is recognized, after then system is developed, grows until maturity and then maintenance needed due to change in the nature of the system. So it died and new system or replacement of it taken place. (This can be shown in below figure)
- Software process models describe the sequence of activities needed to develop a software product.

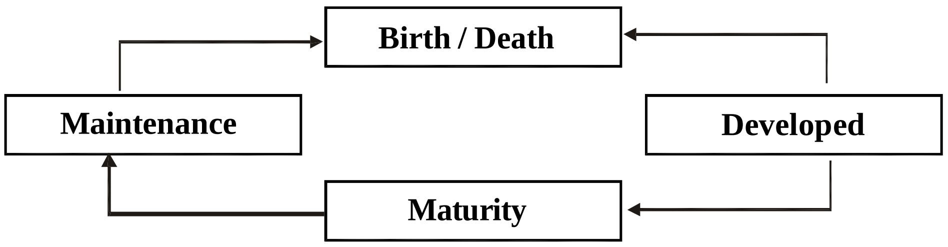


Figure: System Life Cycle

- The goal of the software development process is to produce high quality software product.
- As per IEEE Standards, software life cycle is: "the period of time that starts when software product is conceived and ends when the product is no longer available for use."
- A software life cycle model is also called a Software Development Life Cycle (SDLC). More suitable definition of SDLC is given below:

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality software, which meets exact customer's requirements and completion within time and cost estimation.

- Software life cycle is the series of identifiable stages that a software product undergoes during its lifetime.
- Software life cycle model (process model) is a descriptive and diagrammatic representation of the software life cycle.
- A life cycle model represents all the activities required to make a software product transit through its life cycle phases.
- General stages of software development life cycle are → feasibility study, requirement analysis and specification, design, coding, testing and maintenance.

Need of life cycle models:

- Process models provide generic guidelines for developing a suitable process for a project.
- It provides improvement and guarantee of quality product.
- Without using of a particular life cycle model, the development of a software product would not be in a systematic and disciplined manner.
- Provide monitoring the progress of the project-to-project managers.

- A software life cycle model defines entry and exit criteria for every phase.
- These criteria used to chart the progress of the project.
- A phase can begin only when the corresponding entry criteria are satisfied.
- If entry and exit criteria for various phases are satisfied, it's easy to monitor the progress of the project.
- The documentation of life cycle models enhances the understanding between developers and client.

Different Software Development Life Cycle (SDLC) models OR Software Development Models:

Many life cycle models have been proposed so far. Each of them has some advantages as well as some disadvantages. A few important and commonly used life cycle models are as follows:

(1) Classical waterfall model
(2) Iterative waterfall model
(3) Incremental Evolutionary model
(4) RAD model
(5) Prototype model
(6) Spiral model

(1) Classical Waterfall model:

- This model was originally proposed by Royce (1970). It is also called 'linear sequential life cycle model'.
- It is also called 'traditional waterfall model' or 'conventional waterfall model'.
- It's just a theoretical way of developing software All other life cycle models are essentially derived from it.
- This model breaks down the life cycle into set of phases like:
  - o Feasibility study
  - o Requirements analysis and specification
  - o Design
  - o Coding and unit testing
  - o Integration and system testing
  - o Maintenance

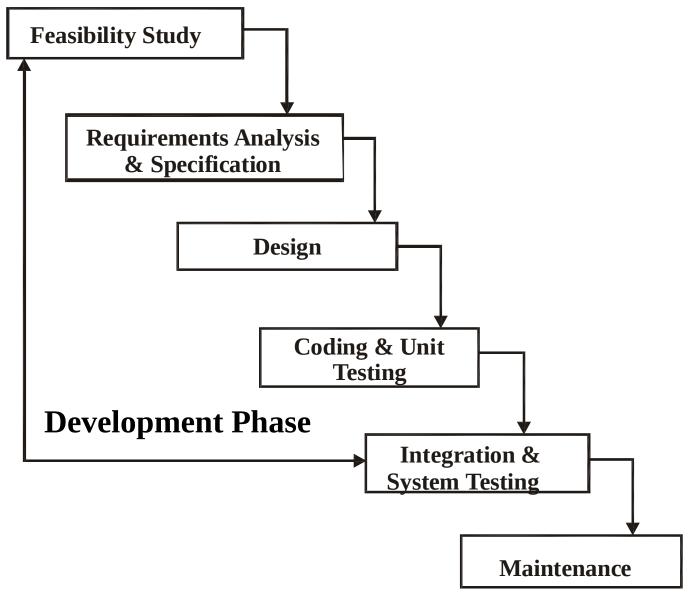


Figure: Classical Waterfall model

Now let's discuss the characteristics and working of every phase one by one.

1. **Feasibility Study:**

- Aim of this phase is to determine whether the system would be financially and technically feasible to develop the product.
- This is an abstract definition of the system.
- It includes the analysis of the problem definition and collection of relevant information of input, processing and output data.
- Collected data are analysed for:
  - → abstract definition Formulation of
  - → different solutions Analysis of
  - → alternative solutions
- Feasibility is evaluated from developer and customer's point of view.
- Cost/Benefit analysis is also performed at this stage.
- Three main issues are concerned with feasibility study:
  - → **Technical Feasibility** →contains hardware, software, network capability, reliability and availability.
  - → **Economical Feasibility** →contains cost/benefit analysis.
  - → **Operational Feasibility** →checks the usability. Concerned with technical performance and acceptance within the organization.

2. **Requirement Analysis and Specification:**

- Aim of this phase is to understand the exact requirements of the customer and to document them properly.
- It also reduces communication gap between developers and customers.
- Two different activities are performed during this phase:
  1. Requirements gathering and analysis
  2. Requirements specification
- **Requirement gathering:** The goal of this activity is to collect all relevant information from the customer regarding the product to be developed.
- **Requirements analysis:** Analyse all gathered requirements to determine exact needs of the customers/clients and hence improve the quality of the product.
- **Requirements specification:** During this activity, the user requirements are systematically organized into a Software Requirements Specification (SRS) document.
- The important components of SRS are →the functional requirements, the non-functional requirements and the constraints of the system.
- Output of this phase is →SRS document, which is also called Black box specification of the problem.
- This phase concentrates on "what" part, not "how".

 Requirement gathering and requirement analysis & specification collectively called **'Requirement Engineering'.**

3. Design:

- The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language.
- This phase affecting the quality of the product.
- Two main approaches are concerned with this phase:
  1. Traditional design approach
  2. Object oriented design approach

Traditional design approach:

- This approach divided into two parts: structure analysis and structure design.

 Structural Analysis:

- Where the detailed structure of the problem is examined.
- Identify the processes and data flow among these processes.
- DFD is used to perform the structure analysis and to produce result.
- In structure analysis functional requirement specified in SRS are decomposed and analysis of data flow is represented diagrammatically by DFD.

 Structure Design:

- During structured design, the results of structured analysis are transformed into the software design.
- Two main activities are associated with it :

  → **Architectural design (High-level design)** - decomposing the system into modules and build relationship among them.

  → **Detailed design (Low-level design)** - identified individual modules are design with data structure and Algorithms.

Object oriented design approach:

- In object-oriented approach, objects available in the system and relationships between them are identified.
- This approach provides lower development time and effort.
- Several tools and techniques are used in designing like:

Flow chart	DFD	Data Dictionary
Decision Table	Decision Tree	Structured English

4. Coding and Unit testing:

- It is also called the implementation phase.
- The aim of this phase is to translate the software design into source code and unit testing is done module wise.
- Each component of the design is implemented as a program module, and each unit is tested to determine the correct working of the system.
- The system is being operational (working conditions) at this phase.
- This phase affects testing and maintenance.
- Simplicity and clarity should be maintained.
- Output of this phase is → set of program modules that have been individually tested.

5. **Integration and system testing:**

- Once all the modules are coded and tested individually, integration of different modules is undertaken.
- The modules are integrated in a planned manner.
- This phase is carried out incrementally over a number of steps and during each integration step, the partially integrated system is tested and a set of previously planned modules are added to it.
- Finally, when all the modules have been successfully integrated and tested, system testing is carried out.
- Goal of this phase is → to ensure that the developed system works well to its requirements described in the SRS document.
- Basic function of testing is to detect errors. So, it is also called 'quality control measure'.
- Testing procedure is carried out using test data: Program test and System test. Program test is done on test data and system test is done actual data.
- Proper test cases should be selected for successful testing.
- It consists of three different kinds of testing activities.
  - → **α- testing:** It is the system testing performed by the development team.
  - → **β-testing:** It is the system testing performed by a friendly set of customers.
  - → **Acceptance testing:** It is the system testing performed by the customer himself after the product delivery to determine whether to accept or reject the delivered product.

Output of this phase is →system test plan and test report (error report).

6. **Maintenance:**

- It requires maximum efforts among all the phases to develop software product.
- This phase is needed to keep system operational.
- Generally, maintenance is needed due to change in the environment or the requirement of the system.
- Maintenance involves performing following four kinds of activities:
  - → **Corrective maintenance** - Correcting errors that were not discovered during the product development phase. It is done after product development.
  - → **Perfective maintenance** - Improving and enhancing the functionalities of the system according to the customer's requirements.

    It mainly deals with implementing new or changed user requirements and improving system performance.
  - → **Adaptive maintenance** - Porting the software to work in a new environment and ensure working. It also consists of adaption of software in the changes of environment.
  - → **Preventive maintenance** - is scheduled, routine maintenance to keep equipment running as well as prevent downtime and expensive repair cost. It reduces the likelihood of failure.

In this phase, the system is reviewing for studying the performance and knowing the full capabilities of the system.

▪ **Advantages of waterfall model:**

- It is simple and easy to understand and use.
- Clearly defined stages. As each phase has well defined input and outputs.
- It helps project personnel in planning.
- Waterfall model works well for smaller projects where requirements are very well understood.

- Well understood milestones.
- Results are well documented.

▪ **Disadvantages of waterfall model:**

- High amounts of risk and uncertainty.
- It is a theoretical model, as it is very difficult to strictly follow all the phases in all types of projects.
- Not so good for complex and object-oriented projects. Also, it cannot be used for the projects where requirements are changed frequently.
- It may happen that the error may be generated at any phase and encountered in later phase. So it is not possible to go back and solve the error in this model.

▪ **Applications of waterfall model (When to use waterfall model):**

→ This model is used only when the requirements are very well known, clear and fixed.
→ When environment is stable.
→ When product definition is stable.
→ When technology is understood.
→ When the project is small.

(2) Iterative Waterfall model:

- Classical waterfall model is idealistic: It assumes that no defect is introduced during any development activity.
- But in practice defects do get introduced in almost every phase of the life cycle. Even defects may get at much later stage of the life cycle. So, solution of this problem is iterative waterfall model.
- Iterative waterfall model is by far the most widely used model. Almost every other model is derived from the waterfall model.
- The principle of detecting errors as close to its point of introduction as possible - is known as "phase containment of errors."
- Phase containment of errors can be achieved by reviewing after every milestone.

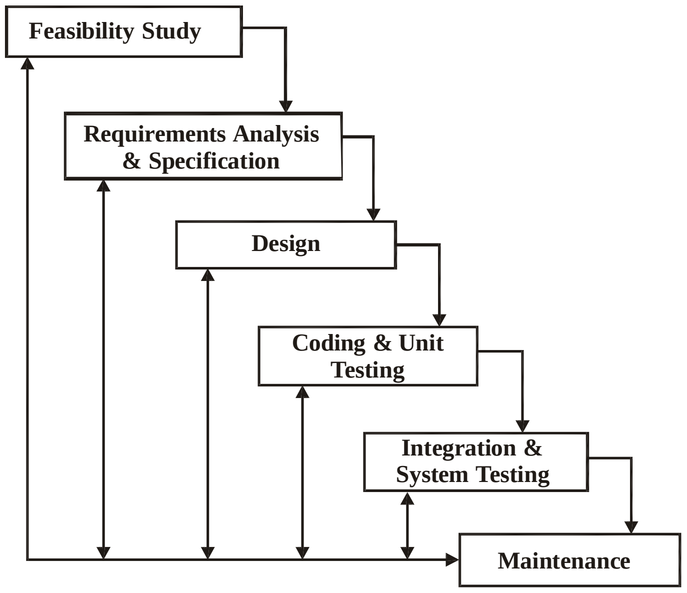

Feasibility Study
Requirements Analysis & Specification
Design
Coding & Unit Testing
Integration & System Testing
Maintenance

Figure: Iterative Waterfall Model

(3) Incremental Model:

- It is also referred as the successive version of waterfall model using incremental approach and evolutionary model.
- In this model, the system is broken down into several modules which can be incrementally implemented and delivered.
- First develop the core product of the system. The core product is used by customers to evaluate the system.
- The initial product skeleton is refined into increasing levels of capability : by adding new functionalities in successive versions.

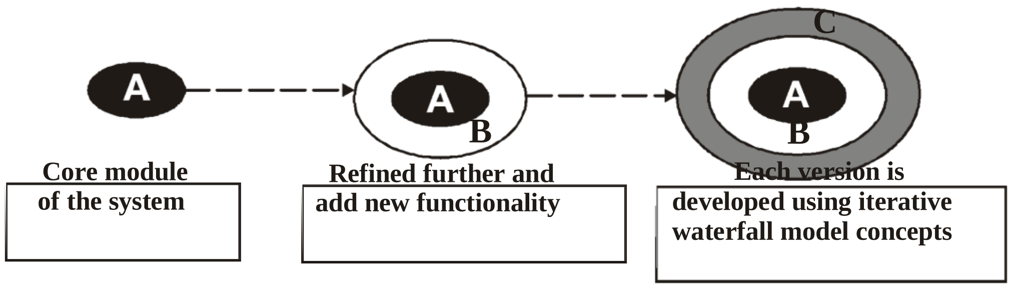

A
A
B
C
A
B
Core module of the system
Refined further and add new functionality
Each version is developed using iterative waterfall model concepts

Figure: Incremental model

- From above figure, we can analyse that at the initial stage, core module/product is developed. Then by adding customer requirements or new functionalities the incremental version is built. Each version is developed using iterative waterfall model concepts.

Advantages:

→ Each successive version performing more useful work than previous versions.

→ Initial product deliver is faster.

→ The core modules get tested thoroughly, thereby reducing chance of errors in final product.

→ The model is more flexible and less costly to change the scope and requirements.

→ User gets a chance to experiment with partially developed software.

→ This model helps finding exact user requirements.

→ Feedback providing at each increment is useful for determining the better final product.

Disadvantages:

→ Sometimes it is difficult to subdivide problems into functional units.

→ Model can be used for very large problems.

→ It needs good planning and design.

→ Resulting product cost may exceed.

Applications:

→ When the problem is very large and user requirements are not well specified at initial stage.

→ When new technology is used in development.

(4) RAD model:

- Full form of RAD is - Rapid Application Development. This model was proposed by IBM in 1980.
- Rapid application development (RAD) is an incremental software development process model that emphasize on extremely short development cycle.
- It emphasizes on reuse.
- The RAD model is a "high-speed" adaptation of the linear sequential model in which rapid development is achieved by using component-based construction.
- If requirements are well understood and project scope is limited, the RAD process enables a development team to create a "fully functional system" within very short time periods (e.g., 60 to 90 days).
- Figure of this model is shown below. Many development teams are working parallel to complete the task.

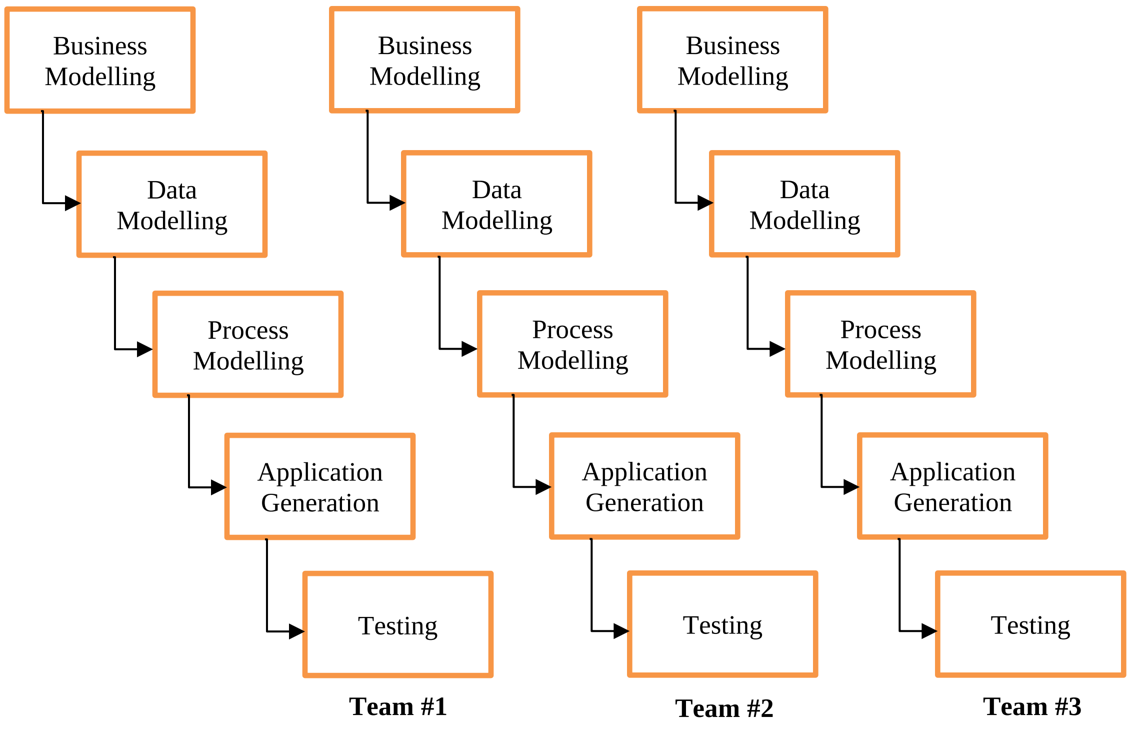


Figure: RAD Model

- **Phases of RAD model are:**

1. Business modeling

- The information flow among business functions is carried out in this phase. Business functions are modelled.

2. Data modeling

- The characteristics of objects (attributes) and their relationships are defined.

3. Process modeling

- The data objects defined in data modeling are transformed to implement business functions.

4. Application generation

- Automated tools are used to convert process models into code and the actual system. Tools like VB, VC++ and JAVA etc. are used.
- RAD works to reuse existing components or create new ones.

5. Testing and turn over

- As RAD uses the components that already been tested so the time for developing and testing is reduced.

Advantages:

→ Application can be developed in a quick time.
→ This model highly makes use of reusable components.
→ Reduce time for developing and testing.
→ Due to full customer involvement, customer satisfaction is improved.

Disadvantages:

→ Requirements must be cleared and well understood for this model.
→ It is not well suited where technical risk is high.
→ In it, highly skilled and expert developers are needed.
→ User involvement is necessary.

Applications:

→ The RAD model is mostly used when the system is modularized and all the requirements are well defined.
→ When a system needs to be produced in a short span of time (2-3 months).
→ RAD SDLC model should be chosen only if resources with high business knowledge are available.
→ When the technical risk is less.

(5) Prototype model:

- Prototype is a working physical system or sub system. Prototype is nothing but a 'toy implementation of a system'.
- In this model, before starting actual development, a working prototype of the system should be built first.
- A prototype is actually a partial developed product.
- A prototype usually turns out to be a very crude version of the actual system.
- Compared to the actual software, a prototype usually has:
  - o limited functional capabilities
  - o low reliability
  - o inefficient performance
- Prototype usually built using several shortcuts, and these shortcuts might be inefficient, inaccurate or dummy functions.
- Prototype model is very useful in developing GUI part of system. The prototype model is shown in below figure.

In working of the prototype model, product development starts with initial requirements gathering phase.

- Then, quick design is carried out and prototype is built.
- The developed prototype is then submitted to the customer for his evaluation.
- Based on customer feedback, the requirements are refined and prototype is modified.
- This cycle of obtaining customer feedback and modifying the prototype continues till the customers approve the prototype.
- Then the actual system is developed using the different phases of iterative waterfall model.
- There are two types of prototype model.

I. **Exploratory development prototype model:** in which the development work is done with the users to explore their requirements and deliver a final system.

II. **Throw away prototype model:** in which a small part of the system is developed and given to the customer to try out and evaluate. Then feedback is collected from customer and accordingly main system is modified. The prototype is then discarded.

Advantages:

→ A partial product is built at initial stage, so customers can get a chance to have a look of the product.
→ New requirements can be accommodated easily.
→ Missing functionalities identified quickly.
→ It provides better flexibility in design and development.
→ As the partial product is evaluated by the end users, more chance of user satisfaction.
→ Quick user feedback is available for better solution.

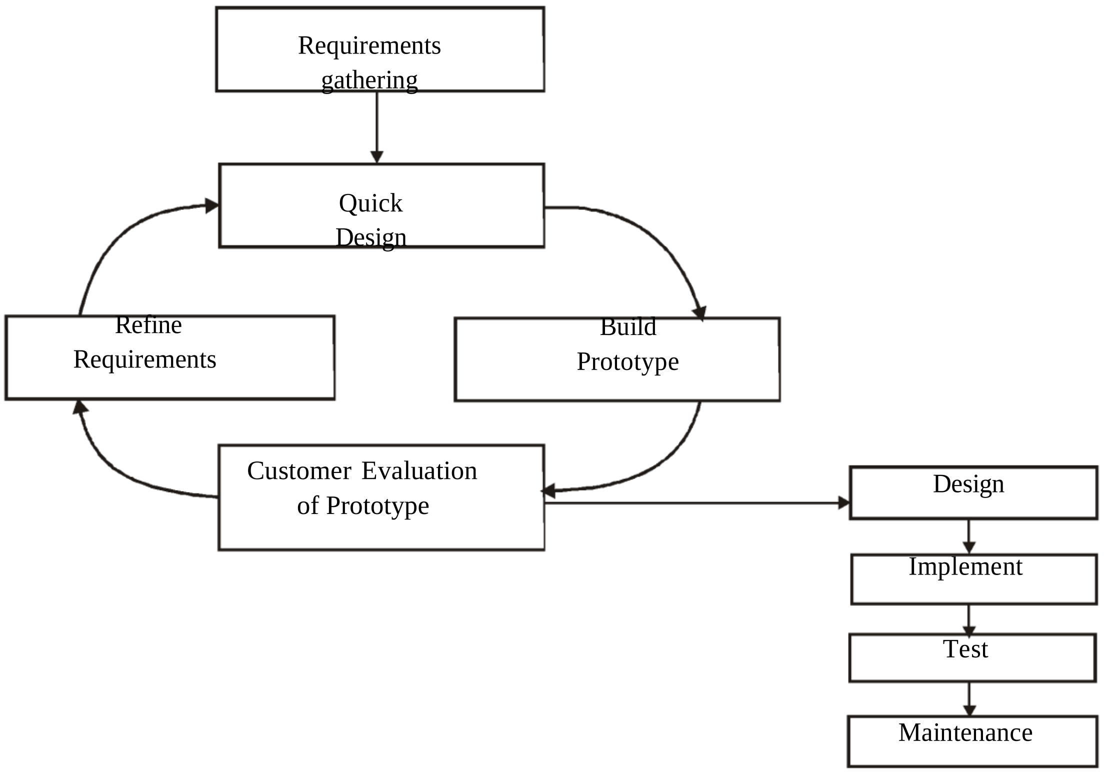


Figure: The Prototype Model

Disadvantages:

→ The code for prototype model is usually thrown away. So, wasting of time is there.

→ The construction cost of developing the prototype is very high.

→ If end user is not satisfied with the initial prototype, he may lose interest in the final product.

→ This model requires extensive participation and involvement of the customers that is not possible every time.

Applications:

→ This model used when desired system needs to have a lot of interactions with end users.

→ This type of model generally used in GUI (Graphical User Interface) type of development.

(6) Spiral model:

- Spiral model is proposed by Boehm in 1986.
- In application development, spiral model uses fourth generation languages (4GL) and developments tools.
- The diagrammatic representation of this model appears like a spiral with many loops.

Figure of spiral model is shown below.

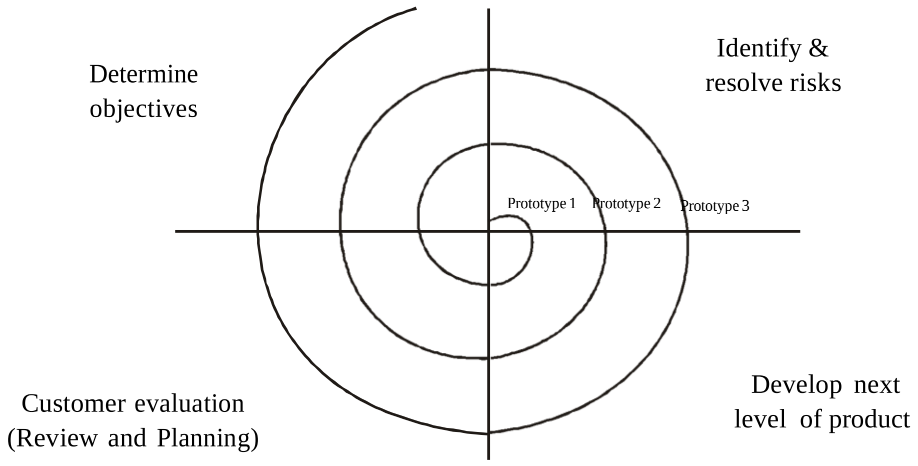


Figure: Spiral Model

Each loop of the spiral represents a phase of the software process:

- The innermost loop might be concerned with system feasibility,
- The next loop with system requirements definition,
- The next one with system design, and so on
- Number of phases is not fixed in this model.
- This model is more flexible compared with other models.
- Each loop in the spiral is split into four sectors (quadrants).

1st Quadrant- Determine objectives:

- Identifying the objectives, identifying their relationships and find the possible alternative solutions. Objectives like: performance, functionality, hardware software interface etc.
- Also examine the risks associated with these objectives.

2nd Quadrant- Identify and resolve risks:

- In this phase, detailed analysis is carried out of each identified risk
- Alternate solutions are evaluated and risks are reduces at this quadrant.

3rd Quadrant - Develop next level product:

- Develop and validate the next level of the product after resolving the identified risks.
- Activities like design, code development, code inspection, testing and packaging are performed.
- Resolution of critical operations and technical issues of next level product are performed.

4th Quadrant - Review and Planning (Customer evaluation):

- In this part, review the results achieved so far.
- Plan the next iteration around the spiral. Different plans like development plan, test plan, installation plan are performed.
- With each iteration around the spiral; progressively more complete version of the software gets built.
- In spiral model at any point, Radius represents cost and Angular dimension represents progress of the current phase.
- At the end, all risks are resolved and software is ready to use.

Advantages:

- It is more flexible, as we can easily deal with changes.
- Due to user involvement, user satisfaction is improved.
- It provides cohesion between different stages.
- Risks are analyzed and resolved so final product will be more reliable.
- New idea and additional functionalities can be easily added at later stage.

Disadvantages:

- It is applicable for large problem only.
- It can be more costly to use.
- It is more complex to understand.
- More number of documents are needed as a greater number of spirals.
- Risk analysis phase requires much higher expertise.

Applications (when to use spiral model):

- Used when medium to high-risk projects.
- When users are unsure for their needs.
- When requirements are complex.

Spiral model can be viewed as a Meta model.

- Spiral model subsumes almost all the life cycle models.
- Single loop of spiral represents Waterfall Model.
- Spiral model uses a prototyping approach by first building the prototype before developing actual product.
- The iterations along the spiral model can be considered as supporting the Evolutionary Model.
- The spiral model retains the step-wise approach of the waterfall model.

- **Comparison of different Life cycle models :**
- **The Classical waterfall model**
  - → Can be considered as a basic model as all other models are derived from this model.
  - → But, it is not used in practical development as no mechanism to handle errors committed during the phase.
- **The Iterative waterfall model**
  - → Most widely used model.
  - → But, suitable only for well-understood problems.
- **The Prototype model**
  - → Suitable for projects in which user requirements and technical aspects are not well understood.
  - → Especially popular for user interface part of the project (GUI based projects).
- **The Evolutionary model**
  - → It is suitable for large problems: as it can be decomposed into a set of modules that can be incrementally implemented.
  - → Also used for object-oriented development projects.
  - → But can be used only if the incremental delivery of the system is acceptable by the customer.
- **The Spiral model**
  - → Used to develop the projects those have several kinds of risks.
  - → But, it is much complex than other models.
- **Program versus software product.**

Parameter	Program	Software product
Size	It is usually small in size.	It is large in size.
Developer	It has single developer.	Here, team of developers.
User	Author himself is sole user.	Large number of users.
Interface	It lacks proper user interface.	Here, well designed interface.
Documentation	It lacks proper documentation.	Here, well documented and user manual prepared.
Functionality	It provides limited functionality and less features.	It provides more functionality.
Development	No need of systematic, organized and planned development.	Require proper systematic, organized and planned development.