

MODULE 1

1. Introduction to software Engineering:

Software Engineering is a systematic, disciplined, quantifiable study and approach to the design, development, operation, and maintenance of a software system.

Software is a program or set of programs containing instructions that provide desired functionality. And Engineering is the process of designing and building something that serves a particular purpose and finds a cost-effective solution to problems.

1.1 The evolving nature of software engineering:

Software Evolution is a term which refers to the process of developing software initially, then timely updating it for various reasons, i.e., to add new features or to remove obsolete functionalities etc.

The evolution process includes fundamental activities of change analysis, release planning, system implementation and releasing a system to customers.

The necessity of Software evolution:

Software evaluation is necessary just because of the following reasons.

a) Change in requirement with time:

With the passes of time, the organization's needs and modus Operandi of working could substantially be changed so in this frequently changing time the tools(software) that they are using need to change for maximizing the performance.

b) Environment change:

As the working environment changes the things(tools) that enable us to work in that environment also changes proportionally same happens in the software world as the working environment changes then, the organizations need

reintroduction of old software with updated features and functionality to adapt the new environment.

c) Errors and bugs:

As the age of the deployed software within an organization increases their precision or impeccability decrease and the efficiency to bear the increasing complexity workload also continually degrades. So, in that case, it becomes necessary to avoid use of obsolete and aged software. All such obsolete Software need to undergo the evolution process in order to become robust as per the workload complexity of the current environment.

d) Security risks:

Using outdated software within an organization may lead you to at the verge of various software-based cyber attacks and could expose your confidential data illegally associated with the software that is in use. So, it becomes necessary to avoid such security breaches through regular assessment of the security patches/modules are used within the software. If the software isn't robust enough to bear the current occurring Cyber attacks so it must be changed (updated).

e) For having new functionality and features:

In order to increase the performance and fast data processing and other functionalities, an organization need to continuously evolute the software throughout its life cycle so that stakeholders & clients of the product could work efficiently.

1.2 Changing nature of software engineering:

The software is instruction or computer program that when executed provide desired features, function, and performance.

A data structure that enables the program to adequately manipulate information and document that describes the operation and use of the program.

Characteristic of software:

There is some characteristic of software which is given below:

1. Functionality
2. Reliability
3. Usability

4. Efficiency
5. Maintainability
6. Portability

Changing Nature of Software:

Nowadays, seven broad categories of computer software present continuing challenges for software engineers which are given below:

1. System Software:

System software is a collection of programs which are written to service other programs. Some system software processes complex but determinate, information structures. Other system application process largely indeterminate data. Sometimes when, the system software area is characterized by the heavy interaction with computer hardware that requires scheduling, resource sharing, and sophisticated process management.

2. Application Software:

Application software is defined as programs that solve a specific business need. Application in this area process business or technical data in a way that facilitates business operation or management technical decision making. In addition to conventional data processing application, application software is used to control business function in real time.

3. Engineering and Scientific Software:

This software is used to facilitate the engineering function and task. However modern application within the engineering and scientific area are moving away from the conventional numerical algorithms. Computer-aided design, system simulation, and other interactive applications have begun to take a real-time and even system software characteristic.

4. Embedded Software:

Embedded software resides within the system or product and is used to implement and control feature and function for the end-user and for the system itself. Embedded software can perform the limited and esoteric function or provide significant function and control capability.

5. Product-line Software:

Designed to provide a specific capability for use by many different customers, product line software can focus on the limited and esoteric marketplace or address the mass consumer market.

6. Web Application:

It is a client-server computer program which the client runs on the web browser. In their simplest form, Web apps can be little more than a set of linked hypertext files that present information using text and limited graphics. However, as e-commerce and B2B application grow in importance. Web apps are evolving into a sophisticated computing environment that not only provides a standalone feature, computing function, and content to the end user.

7. Artificial Intelligence Software:

Artificial intelligence software makes use of a non numerical algorithm to solve a complex problem that is not amenable to computation or straightforward analysis. Application within this area includes robotics, expert system, pattern recognition, artificial neural network, theorem proving and game playing.

1.3 Software engineering layers:

Software engineering is fully a layered technology, to develop software we need to go from one layer to another. All the layers are connected and each layer demands the fulfillment of the previous layer.

Layered technology is divided into four parts:

- 1 .A quality focus**
- 2. Process**
- 3. Method**
- 4. Tools**

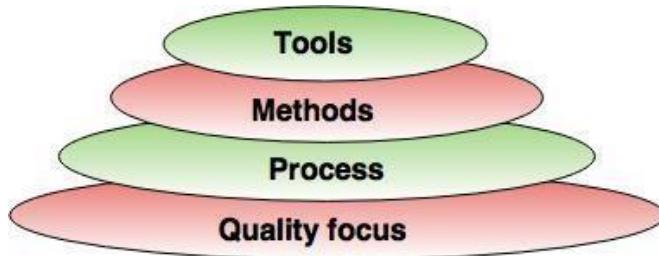
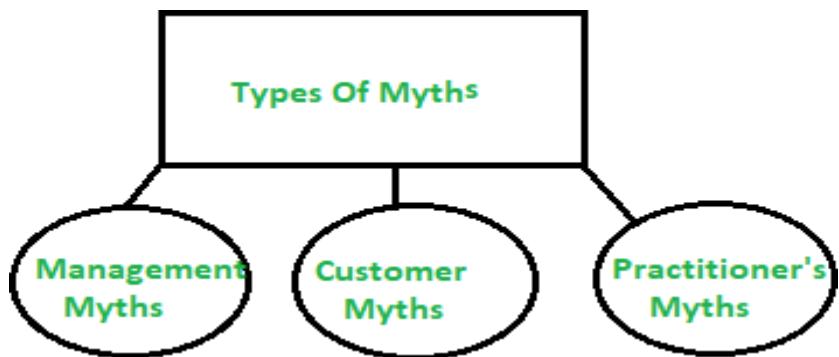


Fig. - Software Engineering Layers

1. **A quality focus:** It defines the continuous process improvement principles of software. It provides integrity that means providing security to the software so that data can be accessed by only an authorized person, no outsider can access the data. It also focuses on maintainability and usability.
2. **Process:** It is the foundation or base layer of software engineering. It is key that binds all the layers together which enables the development of software before the deadline or on time. Process defines a framework that must be established for the effective delivery of software engineering technology. The software process covers all the activities, actions, and tasks required to be carried out for software development.
3. **Method:** During the process of software development the answers to all “how-to-do” questions are given by method. It has the information of all the tasks which includes communication, requirement analysis, design modeling, program construction, testing, and support.
4. **Tools:** Software engineering tools provide a self-operating system for processes and methods. Tools are integrated which means information created by one tool can be used by another.

1.4 software myths:

Most, experienced experts have seen myths or superstitions (false beliefs or interpretations) misleading attitudes (naked users) which creates Major problems for management and technical people. The opposite Types of software-related myths are listed below.



(i) Management Myths :

Myth 1:

We have all the standards and procedures available for software development i.e. the software developer has all the reqd.

Fact:

- Software experts do not know that there are all of them levels.
- Such practices may or may not be expired at present / modern softwareengineering methods.
- And all existing processes are incomplete. Myth

2:

The addition of the latest hardware programs will improve the software development.

Fact:

- The role of the latest hardware is not very high on standard software development; instead (CASE) Engineering tools help the computer they are more important than hardware to produce quality and productivity.
- Hence, the hardware resources are misused. Myth

3:

Managers think that, with the addition of more people and program planners to Software development can help meet project deadlines (Iflagging behind).

Fact:

- Software development is not, the process of doing things like production; here the addition of people in previous stages can reduce the time it will be used for productive development, as the newcomers would take time existing developers of definitions and understanding of the file project. However, planned additions are organized and organized It can help complete the project.

(ii) Customer Myths :

The customer can be the direct users of the software, the technical team, marketing / sales department, or other company. Customer has myths

Leading to false expectations (customer) & that's why you created dissatisfaction with the developer.

Myth 1 :

A general statement of intent is enough to start writing plans (software development) and details of objectives can be done over time.

Fact:

- Official and detailed description of the database function, ethical performance, communication, structural issues and the verification process are important.
- It is happening that the complete communication between the customer and the developer is required.

Myth 2 :

- Project requirements continue to change, but, change, can be easy location due to the flexible nature of the software.

Fact :

- Changes were made to the final stages of software development but cost to make those changes grow through the latest stages of
- Development. A detailed analysis of user needs should be done to minimize change requirement. Figure shows the transition costs in respect of the categories of development.(iii) Practitioner's Myths :

Myths 1 :

They believe that their work has been completed with the writing of the plan and they received it to work.

Fact:

- It is true that every 60-80% effort goes into the maintenance phase (as of the latter software release). Efforts are required, where the product is available first delivered to customers.

Myths 2 :

There is no other way to achieve system quality, behind it done running. Fact:

- Systematic review of project technology is the quality of effective software verification method. These updates are quality filters and more accessible than test.

Myth 3:

An operating system is the only product that can be successfully exported project.

Fact:

- A working system is not enough, it is just the right document brochures and booklets are also reqd. To provide for guidance & software support.

Myth4 :

Engineering software will enable us to build powerful and unnecessary document & always delay us.

Fact:

- Software engineering does not deal with text building, rather while creating better quality leads to reduced recycling & this is being studied for rapid product delivery.

Chapter 2

2.1 process models:

A software process model is an abstraction of the software development process. The models specify the stages and order of a process. So, think of this as a representation of the **order of activities** of the process and the **sequence** in which they are performed.

The goal of a software process model is to provide guidance for controlling and coordinating the tasks to achieve the end product and objectives as effectively as possible.

A model will define the following:

- The tasks to be performed
- The input and output of each task
- The pre and post conditions for each task
- The flow and sequence of each task

Types of software process models

As we mentioned before, there are multiple kinds of software process models that each meet different requirements. Below, we will look at the top seven types of software process models that you should know.

2.1 A GENERIC PROCESS MODEL:

The Generic process model is an abstraction of the [software development process](#). It specifies the stages and order of a process

Generic Process Model will define the following:

Generic Process Framework Activities:

It establishes the foundation for a complete software process by identifying a small number of framework activities.

It also includes a set of umbrella activities that are applicable across the entire software process.

Each framework activity is populated by a set of software engineering actions. A Generic Process Framework for Software Engineering encompasses five activities.

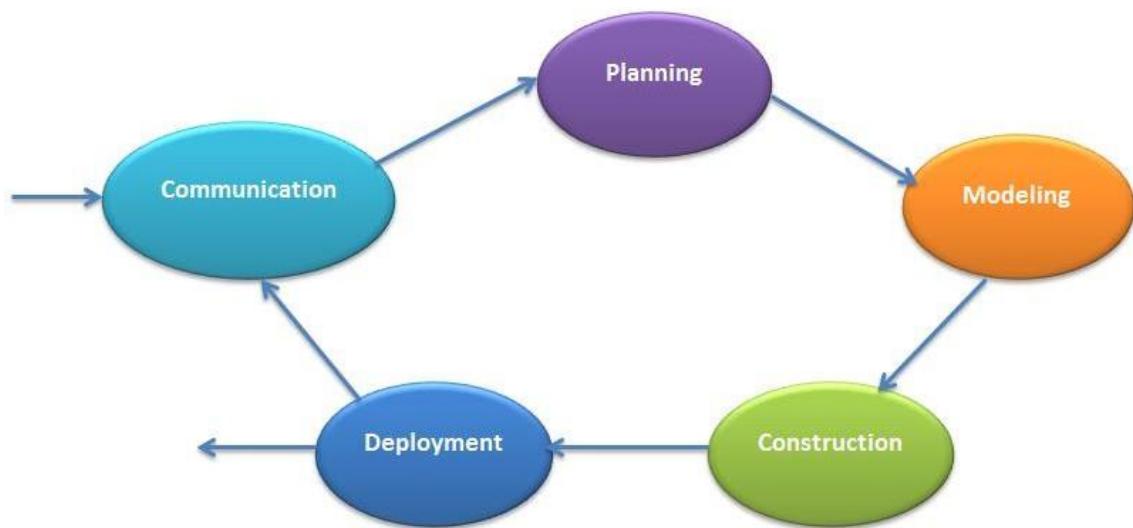


Fig: Generic Process Model

1. **Communication:** This framework activity involves heavy communication and collaboration with the customer. It encompasses requirements gathering and other related activities.
2. **Planning:** This activity establishes a plan for the software engineering work that follows. It describes the technical tasks which are conducted. The resources required and the work products are produced with a work schedule.

3. Modeling: It encompasses the creation of models that allow the developer and the customer. It is easy to better understand Software requirements and the design that will achieve those requirements.

4. Construction: This activity combines code generation and the testing that is required to uncover errors in the code.

5. Deployment: The software delivered to the customer who evaluates the delivered product. It also provides feedback based on the evaluation.

2.2 WATERFALL MODEL:

The Waterfall Model was the first Process Model to be introduced. It is also referred to as a software life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

The Waterfall model is the earliest SDLC approach that was used for software development.

The waterfall Model illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the phases do not overlap.

Waterfall Model – Design:

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 following illustration is a representation of the different phases of the Waterfall Mode.

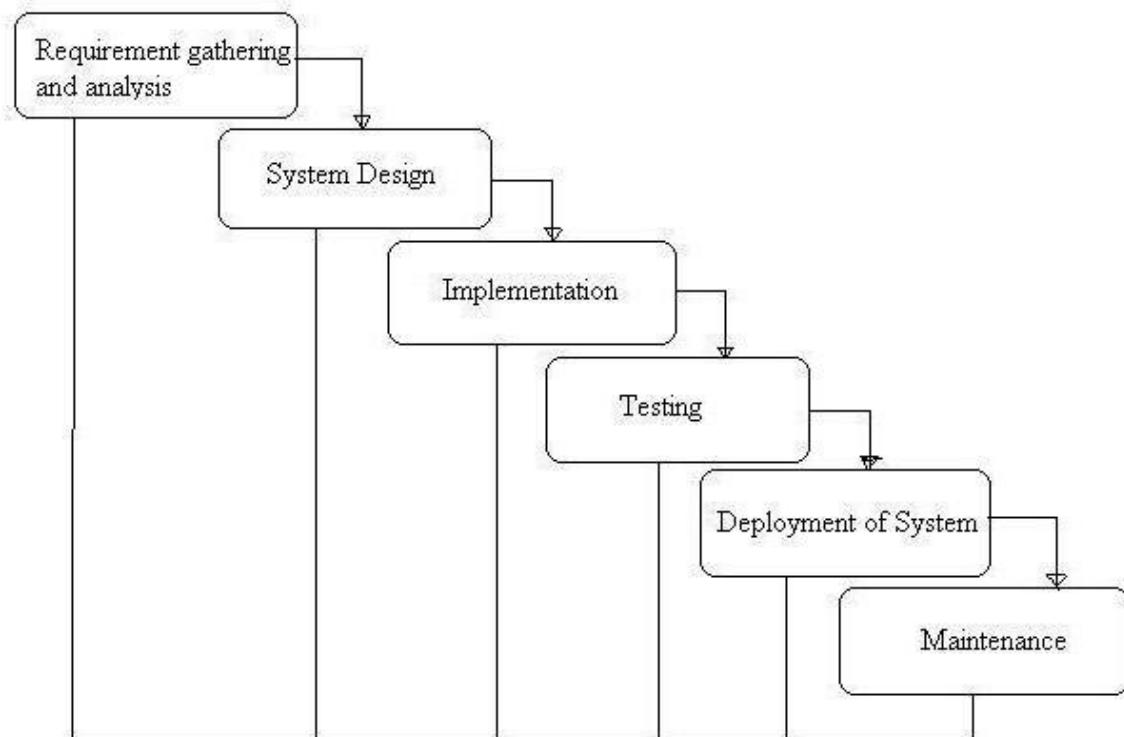
The sequential phases in Waterfall model are –

- **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.

General Overview of "Waterfall Model"



Waterfall Model - Application

Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are –

- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short.

Waterfall Model – Advantages:

Some of the major advantages of the Waterfall Model are as follows –

- 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.

Waterfall Model – Disadvantages:

The major disadvantages of the Waterfall Model are as follows –

- 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.

2.3 INCREMENTAL PROCESS MODELS:

Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle. In this model, each module goes through the requirements, design, implementation and testing phases. Every subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.

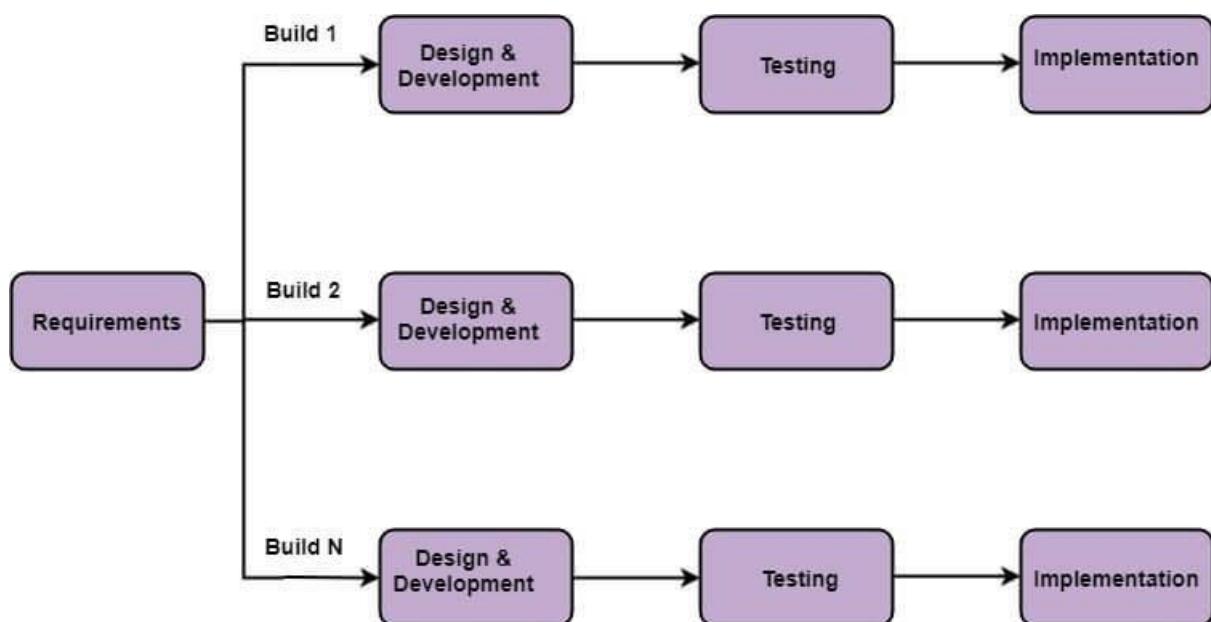


Fig: Incremental Model

The various phases of incremental model are as follows:

- 1. Requirement analysis:** In the first phase of the incremental model, the product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.
- 2. Design & Development:** In this phase of the Incremental model of SDLC, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.

3. Testing: In the incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.

4. Implementation: Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase. After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product

Types of incremental model:

1. Staged delivery model: construction of only one part of the project at a time.
2. Parallel development model: different subsystems are developed at the same time

Advantage of Incremental Model

- o Errors are easy to be recognized.
- o Easier to test and debug
- o More flexible.
- o Simple to manage risk because it is handled during its iteration.
- o The Client gets important functionality early.

Disadvantage of Incremental Model

- o Need for good planning
- o Total Cost is high.
- o Well defined module interfaces are needed.

2.4 EVOLUTIONARY PROCESS MODELS:

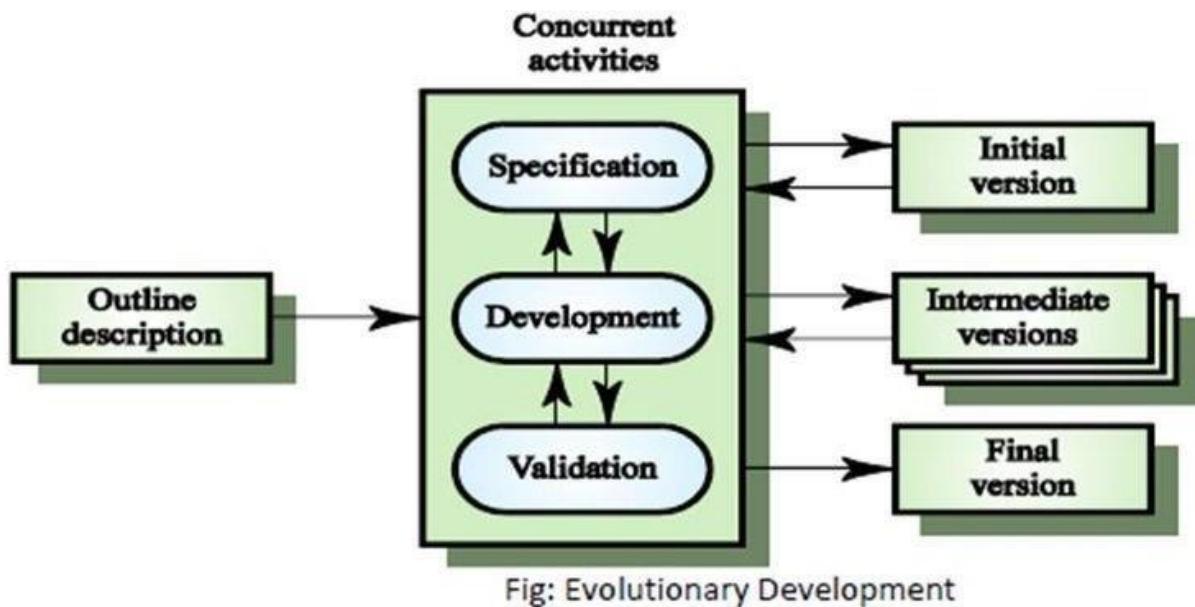
Evolutionary model is also referred to as the successive versions model and sometimes as the incremental model.

In Evolutionary model, the software requirement is first broken down into several modules (or functional units) that can be incrementally constructed and delivered.

The development first develops the core modules of the system. The core modules are those that do not need services from the other modules.

The initial product skeleton is refined into increasing levels of capability by adding new functionalities in successive versions.

Each evolutionary model may be developed using an iterative waterfall model of development.



Evolutionary model is used when the customer prefers to receive the product in increments so that he can start using the different features as and when they are developed rather than waiting all the time for the full product to be developed and delivered.

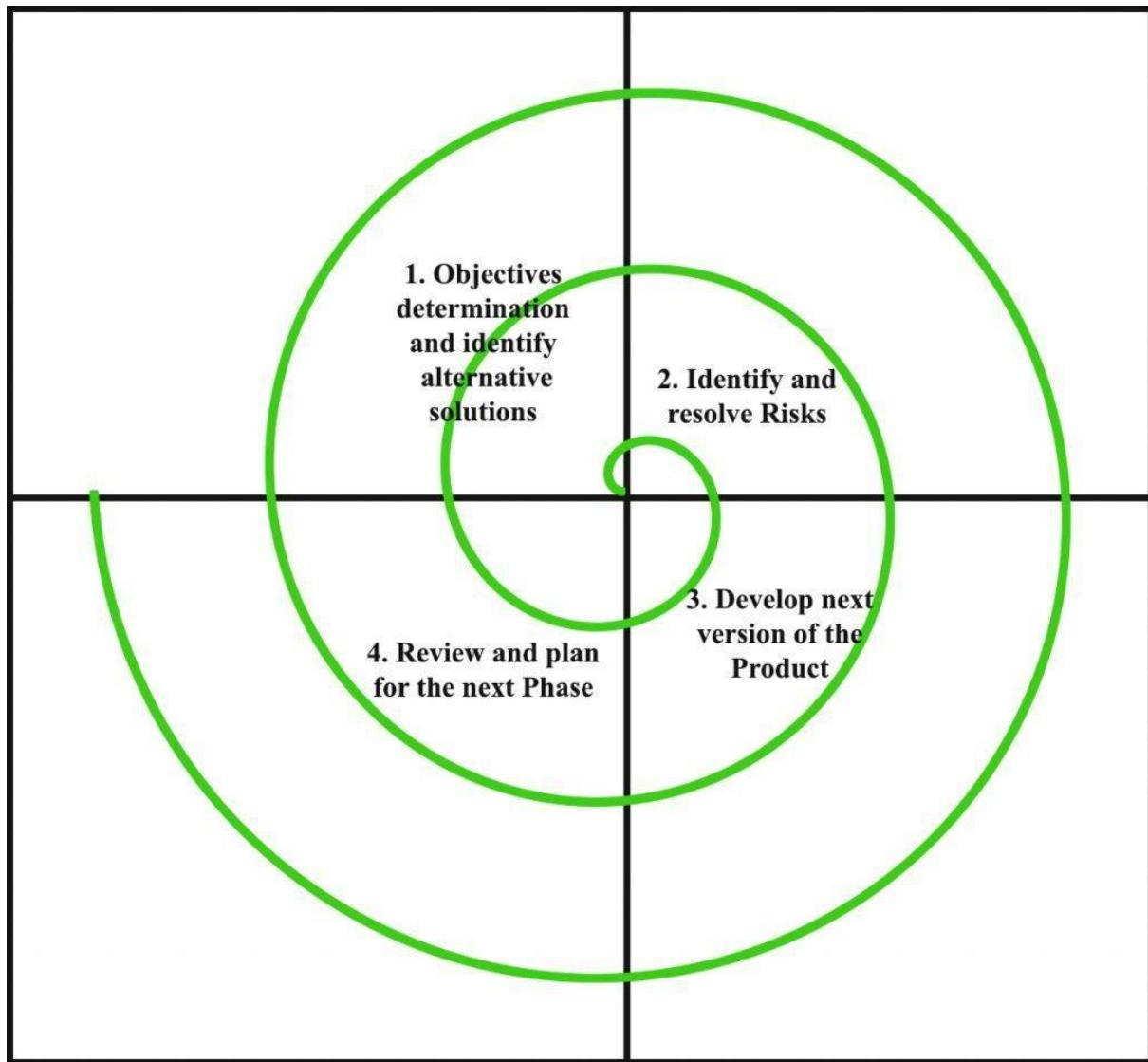
Advantages of Evolutionary Model

- Large project: Evolutionary model is normally useful for very large products.
- User gets a chance to experiment with a partially developed software much before the complete version of the system is released.
- Evolutionary model helps to accurately elicit user requirements during the delivery of different versions of the software.
- The core modules get tested thoroughly, thereby reducing the chances of errors in the core modules of the final products.
- Evolutionary model avoids the need to commit large resources in one go for development of the system.

Disadvantages of Evolutionary Model

- Difficult to divide the problem into several versions that would be acceptable to the customer and which can be incrementally implemented and delivered.

2.5 SPIRAL MODEL:



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

Spiral Model - Design

The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

Objectives determination and identify alternative solutions:

This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

This phase also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral, the product is deployed in the identified market.

Identify and resolve risk:

In the risk analysis phase, a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.

Develop next version of the product:

In this phase software is developed, along with testing at the end of the phase. Hence in this phase the development and testing is done. At the end of the third quadrant the next version of the software is available.

Review and plan for the next phase:

This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

Spiral Model Application

The Spiral Model is widely used in the software industry as it is in sync with the natural development process of any product, i.e. learning with maturity which involves minimum risk for the customer as well as the development firms.

The following pointers explain the typical uses of a Spiral Model –

- When there is a budget constraint and risk evaluation is important.
- For medium to high-risk projects.
- Long-term project commitment because of potential changes to economic priorities as the requirements change with time.
- Customer is not sure of their requirements which is usually the case.
- Requirements are complex and need evaluation to get clarity.
- New product line which should be released in phases to get enough customer feedback.

- Significant changes are expected in the product during the development cycle.

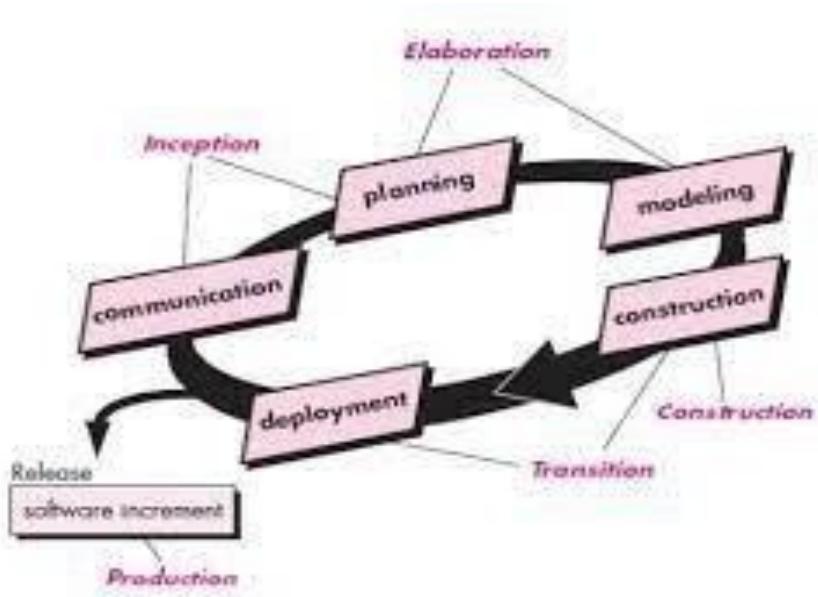
The advantages of the Spiral SDLC Model are as follows –

- 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.

The disadvantages of the Spiral SDLC Model are as follows –

- 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.

2.6 UNIFIED PROCESS:



Unified Process is based on the enlargement and refinement of a system through multiple iterations, with cyclic feedback and adaptation. The system

is developed incrementally over time, iteration by iteration, and thus this approach is also known as iterative and incremental software development. The iterations are spread over four phases where each phase consists of one or more iterations

Phases of the unified process:

Inception—the first and the shortest phase in the project. It is used to prepare basis for the project, including preparation of business case, establishing project scope and setting boundaries, outlining key requirements, and possible architecture solution together with design tradeoffs, identifying risks, and development of initial project plan—schedule with main milestones and cost estimates. If the inception phase lasts for too long, it is like an indicator stating that the project

vision and goals are not clear to the stakeholders. With no clear goals and vision the project most likely is doomed to fail. At this scenario it is better to take a pause at the very beginning of the project to refine the vision and goals. Otherwise it could lead to unnecessary make overs and schedule delays in further phases.

Elaboration—during this phase the project team is expected to capture a majority of system's requirements (e.g., in the form of use cases), to perform identified risk analysis and make a plan of risk management to reduce or eliminate their impact on final schedule and product, to establish design and architecture (e.g., using basic class

diagrams, package diagrams, and deployment diagrams), to create a plan for the next (construction) phase.

Construction—the longest and largest phase within Unified Process. During this phase, the design of the system is finalized and refined and the system is built using the basis created during elaboration phase. The construction phase is divided into multiple iterations, for each iteration to result in an executable release of the system. The final iteration of construction phase releases fully completed system which is to be deployed during transition phase, and

Transition—the final project phase which delivers the new system to its end-users. Transition phase includes also data migration from legacy systems and user trainings.

Each phase and its iteration consists of a set of predefined activities. The Unified Process describes work activities as disciplines—a discipline is a set of activities and related artifacts in one subject area (e.g., the activities

within requirements analysis). The disciplines described by Unified Process are as follows.

Business modeling—domain object modeling and dynamic modeling of the business processes,

Requirements—requirements analysis of system under consideration. Includes activities like writing use cases and identifying nonfunctional requirements,

Analysis and design—covers aspects of design, including the overall architecture,

Implementation—programming and building the system (except the deployment),

Test—involves testing activities such as test planning, development of test scenarios, alpha and beta testing, regression testing, acceptance testing, and

Deployment—the deployment activities of developed system.

2.7 PERSONAL AND TEAM PROCESS MODEL:

Personal software process:

The personal software process is a structured software development process that is designed to help software engineers better understand and improve their performance by developing of the code.

- It is a systematic application development method intended to help engineers understand and enhance their output by applying professionalism to the way they build software and monitoring their expected and actual code creation.
- It indicates developers how to control the value of their assets, what to draw up a sound plan as well as how to make promises. This also provides them the evidence to explain their proposals.
- The personal software method focuses on entities to enhance their results. It consists of methods, types and techniques that orient programmers in their technical work.

Team software process:

The team software process guides engineering teams in developing software intensive products

- The goal of the TSP is to enhance the quality and efficiency of the entire team application development project, in addition to helping us help meet the expense and timeline obligations of creating software.
- TSP is designed for use in education environments, concentrating on the process of creating a project management team, creating team goals, assigning team tasks, and several other collaboration activities.
- Team software relies on a community of people and seeks to improve team performance.

2.8 THE CAPABILITY MATURITY MODEL INTEGRATION:

The Capability Maturity Model (CMM) is a procedure used to develop and refine an organization's software development process.

The model defines a five-level evolutionary stage of increasingly organized and consistently more mature processes.

Capability Maturity Model is used as a benchmark to measure the maturity of an organization's software process.

OBJECTIVES OF CMMI:

1. Fulfilling customer needs and expectations.
2. Value creation for investors/stockholders.
3. Market growth is increased.
4. Improved quality of products and services.
5. Enhanced reputation in industry.

Chapter 3

REQUIREMENTS ENGINEERING

Functional vs Non Functional Requirements

Requirements analysis is very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: *Functional* and *Non-functional requirements*.

Functional Requirements: These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Non-functional requirements: These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements.

They basically deal with issues like:

Functional Requirements	Non Functional Requirements
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies “What should the software system do?”	It places constraints on “How should the software system fulfill the functional requirements?”
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.
It is mandatory.	It is not mandatory.
It is captured in use case.	It is captured as a quality attribute.
Defined at a component level.	Applied to a system as a whole.
Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Functional Testing like System, Integration, End to End, API testing, etc are done.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc are done.

Functional Requirements	Non Functional Requirements
-------------------------	-----------------------------

Usually easy to define.

Example

- 1) Authentication of user whenever he/she logs into the system.
- 2) System shutdown in case of a cyber attack.
- 3) A Verification email is sent to user whenever he/she registers for the first time on some software system.

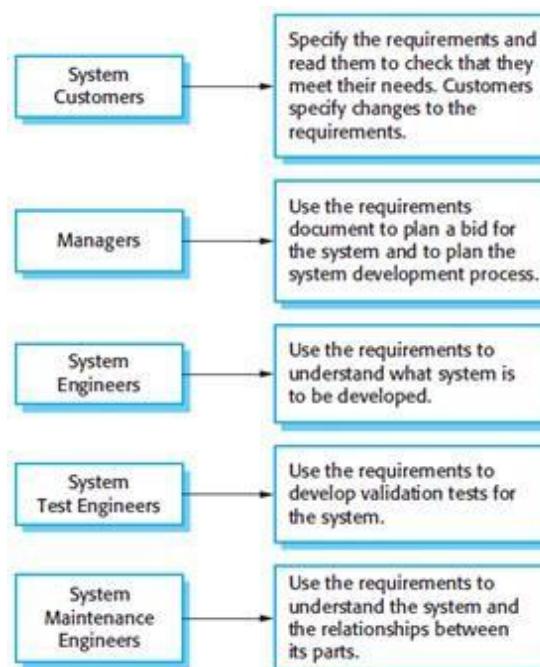
Usually more difficult to define.

Example

- 1) Emails should be sent with a latency of no greater than 12 hours from such an activity.
- 2) The processing of each request should be done within 10 seconds
- 3) The site should load in 3 seconds when the number of simultaneous users are > 10000

The software requirements document

- The software requirements document (sometimes called the software requirements specification or SRS) is an official statement of what the system developers should implement.
- It should include both the user requirements for a system and a detailed specification of the system requirements. Sometimes, the user and system requirements are integrated into a single description.
- In other cases, the user requirements are defined in an introduction to the system requirements specification. If there are a large number of requirements, the detailed system requirements may be presented in a separate document.



The software requirements document (sometimes called the software requirements specification or SRS) is an official statement of what the system developers should implement. It should include both the user requirements for a system and a detailed specification of the system requirements. Sometimes, the user and system requirements are integrated into a single description. In other cases, the user requirements are defined in an introduction to the system requirements specification. If there are a large number of requirements, the detailed system requirements may be presented in a separate document.

Requirements documents are essential when an outside contractor is developing the software system. However, agile development methods argue that requirements change so rapidly that a requirements document is out of date as soon as it is written, so the effort is largely wasted. Rather than a formal document, approaches such as Extreme Programming (Beck, 1999) collect user requirements incrementally and write these on cards as user stories.

The user then prioritizes requirements for implementation in the next increment of the system.

For business systems where requirements are unstable, I think that this approach is a good one. However,

I think that it is still useful to write a short supporting document that defines the business and dependability requirements for the system; it is easy to forget the requirements that apply to the system as a whole when focusing on the functional requirements for the next system release.

The requirements document has a diverse set of users, ranging from the senior management of the organization that is paying for the system to the engineers responsible for developing the software.

REQUIREMENTS SPECIFICATION:

- Requirements specification is the process of writing down the user and system requirements in a requirements document.
- Ideally, the user and system requirements should be clear, unambiguous, easy to understand, complete, and consistent. In practice, this is difficult to achieve as stakeholders interpret the requirements in different ways and there are often inherent conflicts and inconsistencies in the requirements.
- The user requirements for a system should describe the functional and non-functional requirements so that they are understandable by system users who don't have detailed technical knowledge.
- Ideally, they should specify only the external behavior of the system.
- The requirements document should not include details of the system architecture or design.
- Consequently, if you are writing user requirements, you should not use software jargon, structured notations, or formal notations.
- You should write user requirements in natural language, with simple tables, forms, and intuitive diagrams.
- System requirements are expanded versions of the user requirements that are used by software engineers as the starting point for the system design.
- They add detail and explain how the user requirements should be provided by the system. They may be used as part of the contract for the implementation of the system and should therefore be a complete and detailed specification of the whole system.

- Ideally, the system requirements should simply describe the external behavior of the system and its operational constraints.

They should not be concerned with how the system should be designed or implemented. However, at the level of detail required to completely specify a complex software system, it is practically impossible to exclude all design information.

There are several reasons for this:

You may have to design an initial architecture of the system to help structure the requirements specification. The system requirements are organized according to the different sub-systems that make up the system. In most cases, systems must interoperate with existing systems, which constrain the design and impose requirements on the new system.

The use of a specific architecture to satisfy non-functional requirements (such as N-version programming to achieve reliability, discussed in Chapter 13) may be necessary. An external regulator who needs to certify that the system is safe may specify that an already certified architectural design be used.

User requirements are almost always written in natural language supplemented by appropriate diagrams and tables in the requirements document.

System requirements may also be written in natural language but other notations based on forms, graphical system models, or mathematical system models can also be used.

Graphical models are most useful when you need to show how a state changes or when you need to describe a sequence of actions.

UML sequence charts and state charts, described in Chapter 5, show the sequence of actions that occur in response to a certain message or event.

Formal mathematical specifications are sometimes used to describe the requirements for safety- or security- critical systems, but are rarely used in other circumstances.

REQUIREMENTS ENGINEERING:

Requirements engineering (RE) refers to the process of defining, documenting, and maintaining requirements in the engineering design process. Requirement engineering provides the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system. Thus, requirement engineering is the disciplined application of proven principles, methods, tools, and notation to describe a proposed system's intended behavior and its associated constraints.

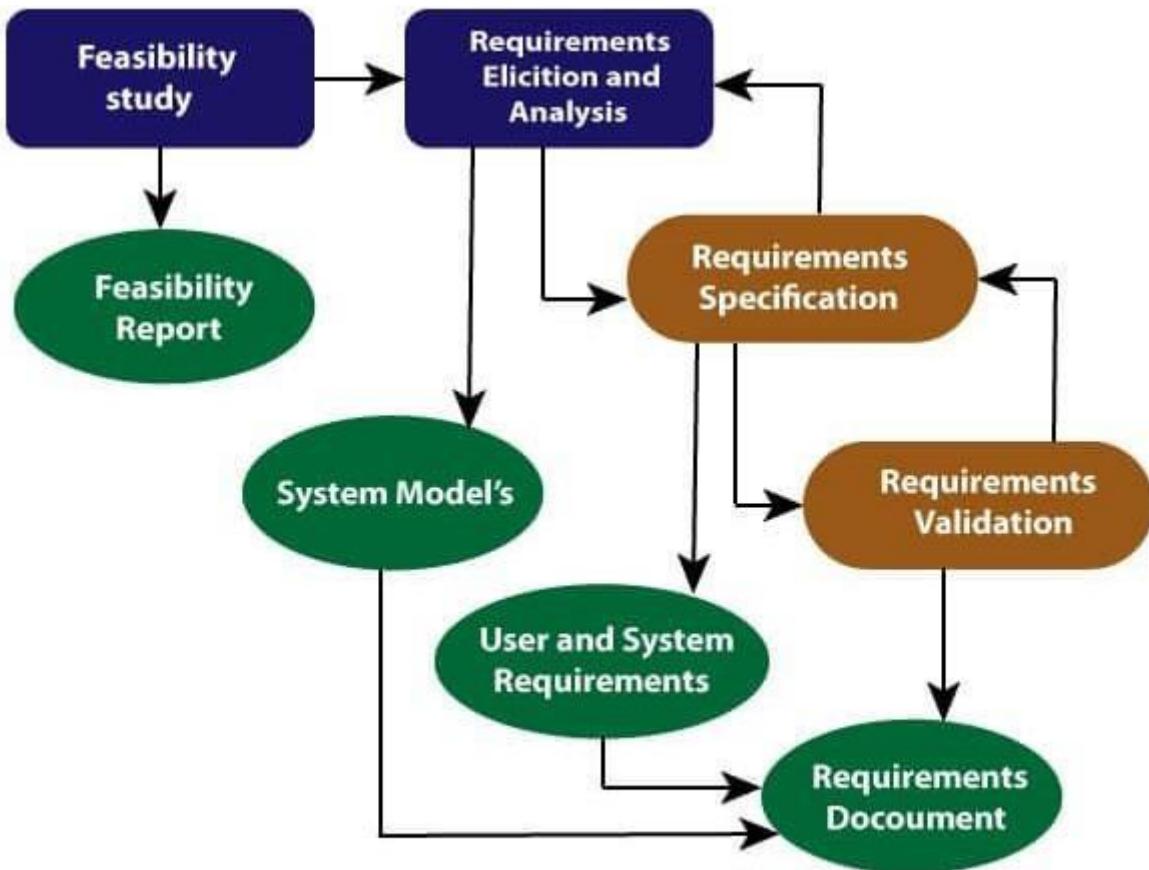
Requirement Engineering Process

It is a four-step process, which includes -

Feasibility Study

Requirement Elicitation and Analysis

Software Requirement Specification



Requirement Engineering Process

1. Feasibility Study:

The objective behind the feasibility study is to create the reasons for developing the software that is acceptable to users, flexible to change and conformable to established standards.

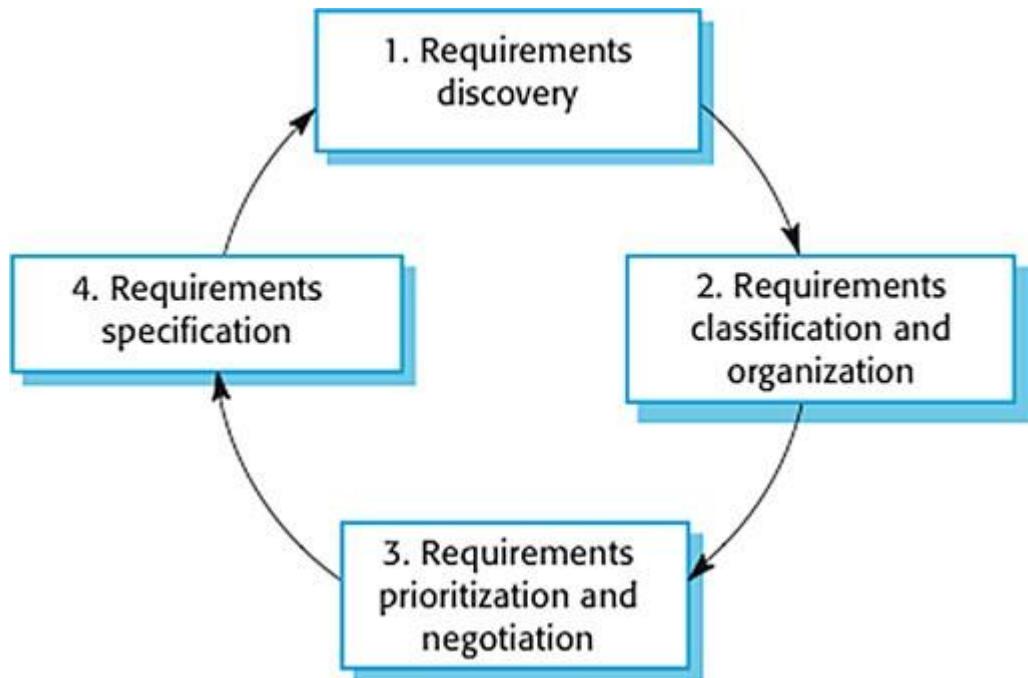
Types of Feasibility:

1. **Technical Feasibility** - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.
2. **Operational Feasibility** - Operational feasibility assesses the range in which the required software

performs a series of levels to solve business problems and customer requirements.

3. **Economic Feasibility** - Economic feasibility decides whether the necessary software can generate financial profits for an organization.

REQUIREMENT ELICITATION AND ANALYSIS:



It's a process of interacting with customers and end-users to find out about the domain requirements, what services the system should provide, and the other constraints.

Domain requirements reflect the environment in which the system operates so, when we talk about an application domain we mean environments such as train operation, medical records, e-commerce etc.

It may also involve different kinds of stockholders; end-users, managers, system engineers, test engineers, maintenance engineers, etc.

A stakeholder is anyone who has direct or indirect influence on the requirements.

The requirements elicitation and analysis has 4 main process

We typically start by gathering the requirements, this could be done through a general discussion or interviews with your stakeholders, also it may involve some graphical notation.

Then you organize the related requirements into sub-components and prioritize them, and finally, you refine them by removing any ambiguous requirements that may arise from some conflicts.

Here are the 4 main processes of requirements elicitation and analysis.

It shows that it's an iterative process with feedback from one activity to another. The process cycle starts with requirements discovery and ends with the requirements document. The cycle ends when the requirements document is complete.

Requirements Discovery

It's the process of interacting with, and gathering the requirements from, the stakeholders about the required system and the existing system (if exists).

It can be done using some techniques, like interviews, scenarios, prototypes, etc, which help the stockholders to understand what the system will be like.

Gathering and understanding the requirements is a difficult process

That's because stakeholders may not know what exactly they want the software to do, or they may give unrealistic requirements.

They may give different requirements, which will result in conflict between the requirements, so we have to discover and resolve these conflicts.

Also, there might be some factors that influence the stakeholder's decision, like, for example, managers at a company or professors at the university want to take full control over the management system.

Requirements Classification & Organization

It's very important to organize the overall structure of the system.

Putting related requirements together, and decomposing the system into sub-components of related requirements. Then, we define the relationship between these components.

What we do here will help us in the decision of identifying the most suitable architectural design patterns.

Requirements Prioritization & Negotiation

We previously explained why eliciting and understanding the requirements is not an easy process.

One of the reasons is the conflicts that may arise as a result of having different stakeholders involved. *Why?* because it's hard to satisfy all parties if it's not impossible.

This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiations until you reach a situation where some of the stakeholders can compromise.

We shouldn't reach a situation where a stakeholder is not satisfied because his requirements are not taken into consideration.

Prioritizing your requirements will help you later to focus on the essentials and core features of the system, so you can meet the user expectations.

It can be achieved by giving every piece of function a priority level. So, functions with higher priorities need higher attention and focus.

Requirements Specification

Software requirement specification is a kind of document which is created by a software analyst after the requirements collected from the various sources - the requirement received by the customer written in ordinary language.

It is the job of the analyst to write the requirement in technical language so that they can be understood and beneficial by the development team.

The models used at this stage include ER diagrams, data flow diagrams (DFDs), function decomposition diagrams (FDDs), data dictionaries, etc.

Data Flow Diagrams: Data Flow Diagrams (DFDs) are used widely for modeling the requirements. DFD shows the flow of data through a system. The system may be a company, an organization, a set of procedures, a computer hardware system, a software system, or any combination of the preceding. The DFD is also known as a data flow graph or bubble chart.

Data Dictionaries: Data Dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirements stage, the data dictionary should at least define customer data items, to ensure that the customer and developers use the same definition and terminologies.

Entity-Relationship Diagrams: Another tool for requirement specification is the entity- relationship diagram, often called an "*E-R diagram*." It is a detailed logical representation of the data for the organization and uses three main constructs i.e. data entities, relationships, and their associated attributes.

REQUIREMENT VALIDATION:

After requirement specifications developed, the requirements discussed in this document are validated. The user might demand illegal, impossible solution or experts may misinterpret the needs. Requirements can be checked against the following conditions -

- If they can practically implement
- If they are correct and as per the functionality and specially of software
- If there are any ambiguities
- If they are full
- If they can describe

Requirements Validation Techniques

Requirements reviews/inspections: systematic manual analysis of the requirements.

Prototyping: Using an executable model of the system to check requirements.

Test-case generation: Developing tests for requirements to check testability.

Automated consistency analysis: checking for the consistency of structured requirements descriptions.

Requirements validation is the process of checking that requirements defined for development, define the system that the customer really wants. To check issues related to requirements, we perform requirements validation. We usually use requirements validation to

check error at the initial phase of development as the error may increase excessive rework when detected later in the development process.

In the requirements validation process, we perform a different type of test to check the requirements mentioned in the [Software Requirements Specification \(SRS\)](#), these checks include:

- Completeness checks
- Consistency checks
- Validity checks
- Realism checks
- Ambiguity checks
- Verifiability

The output of requirements validation is the list of problems and agreed on actions of detected problems. The lists of problems indicate the problem detected during the process of requirement validation. The list of agreed action states the corrective action that should be taken to fix the detected problem.

There are several techniques which are used either individually or in conjunction with other techniques to check to check entire or part of the system:

Test case generation:

Requirement mentioned in SRS document should be testable, the conducted tests reveal the error present in the requirement. It is generally believed that if the test is difficult or impossible to design than, this usually means that requirement will be difficult to implement and it should be reconsidered.

Prototyping:

In this validation techniques the prototype of the system is presented before the end-user or customer, they experiment with the presented model and check if it meets their need. This type of model is generally used to collect feedback about the requirement of the user.

Requirements Reviews:

In this approach, the SRS is carefully reviewed by a group of people including people from both the contractor organisations and the client side, the reviewer systematically analyses the document to check error and ambiguity.

Automated Consistency Analysis:

This approach is used for automatic detection of an error, such as nondeterminism, missing cases, a type error, and circular definitions, in requirements specifications.

First, the requirement is structured in formal notation then CASE tool is used to check in- consistency of the system, The report of all inconsistencies is identified and corrective actions are taken.

Walk-through:

A walkthrough does not have a formally defined procedure and does not require a differentiated role assignment.

Checking early whether the idea is feasible or not.

Obtaining the opinions and suggestion of other people.

Checking the approval of others and reaching an agreement.

REQUIREMENT MANAGEMENT:

Requirements Management is the process of gathering, analyzing, verifying, and validating the needs and requirements for the given product or system being developed.

Successful requirements management ensures that completed deliverables meet the expectations of the stakeholders.

Requirements Management is an iterative set of activities that help ensure that elicitation, documentation, refinement, and changes of requirements is adequately dealt with during a lifecycle, with a view toward satisfying the overall mission or need in a quality manner and to the customers' satisfaction.

Whether part of a project management plan or standalone, a Requirements Management Plan (RMP) describes the requirements artifacts, requirement types (including attributes), the requirements management process, and the metrics and tools to be used for measuring, reporting, and controlling change.

