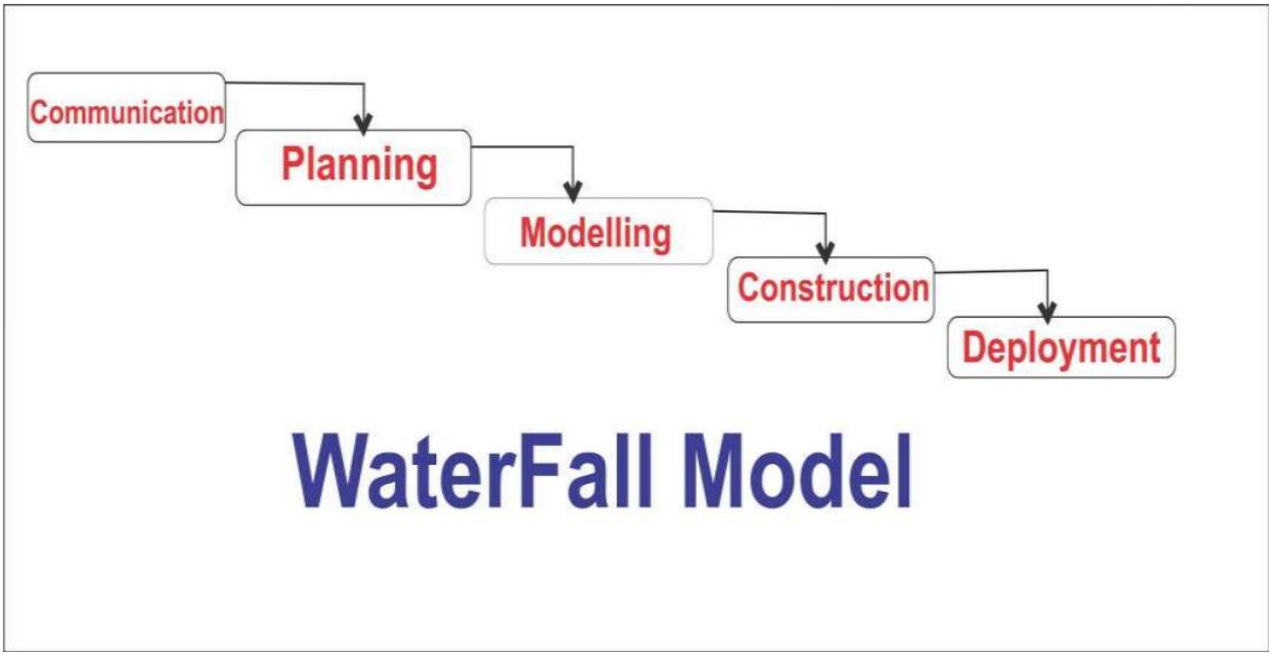
UNIT-II

PROCESS MODELS (Part-I)

Every software engineering organization should describe a unique set of framework activities for the software process it adopts.

THE WATER FALL MODEL:

- It is called classic life cycle or Linear model.
- Requirements are well defined and stable.
- It suggests a systematic, sequential approach to software development.
- It begins with customer specification of requirements and progresses.
  - Planning.
  - Modeling.
  - Construction and
  - Deployment.



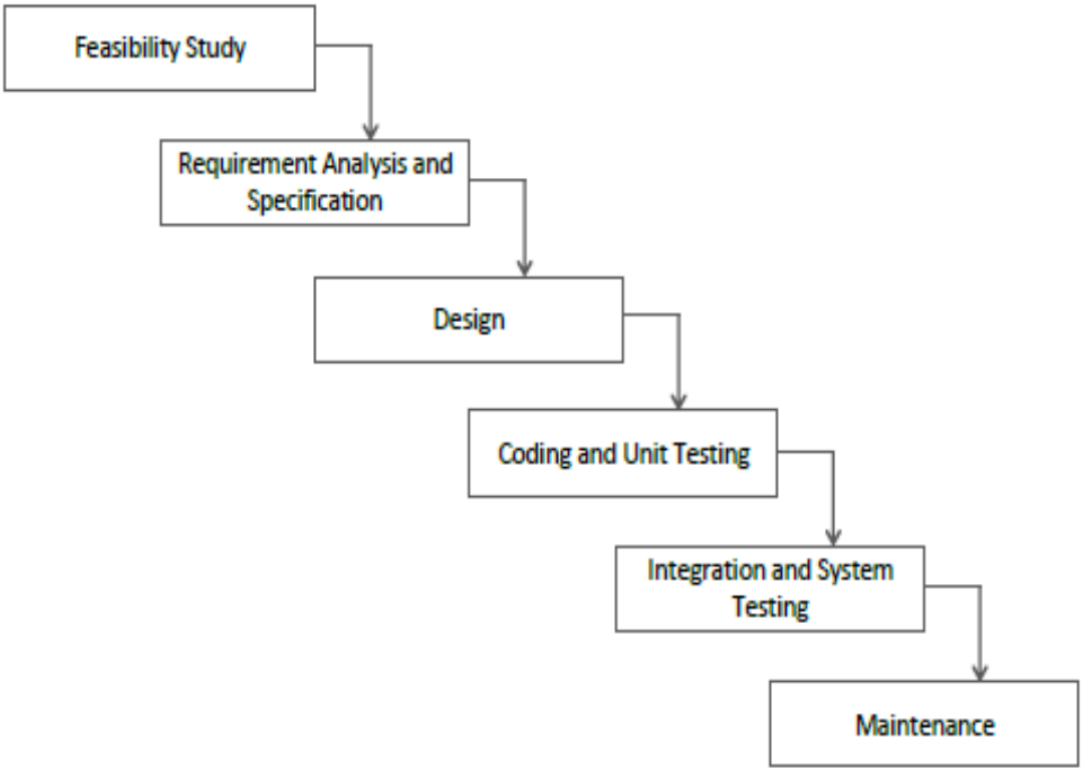
Advantages:

- Easy to understand.
- Each phase has well defined input and output.
- Helps project manager in proper planning of project.
- Provides a template into which methods of analysis, design, code and support can be placed.

Disadvantages:

- One way street.
- It lacks overlapping and interactions among phases.
- Model doesn't support delivery of system in pieces.

Phases of the Classical Waterfall Model:



Feasibility Study:

• It involves analysis of the problem and collection of all relevant information relating to the product.

• The collected data are analysed.

– Requirements of the Customer.

– Formulations of the different strategies for solving the problem.

– Evaluation of different solution strategies.

Requirements Analysis and Specification:

• It is understanding the exact requirements of the customer and to document them properly.

– Requirements gathering and analysis.

– Requirements specification.

Design:

• The design phase is to transform the requirements specified in the document into a structure that is suitable for implementation in some programming language.

• Traditional Design Approach.

• Object-Oriented Design Approach.

Coding and Unit Testing:

• The purpose of the coding and unit testing phase of software development is to translate the software design into source code.

Integration and System Testing:

• Integration of different modules is coded and unit tested.

• $\alpha - Testing$

• $\beta - Testing$

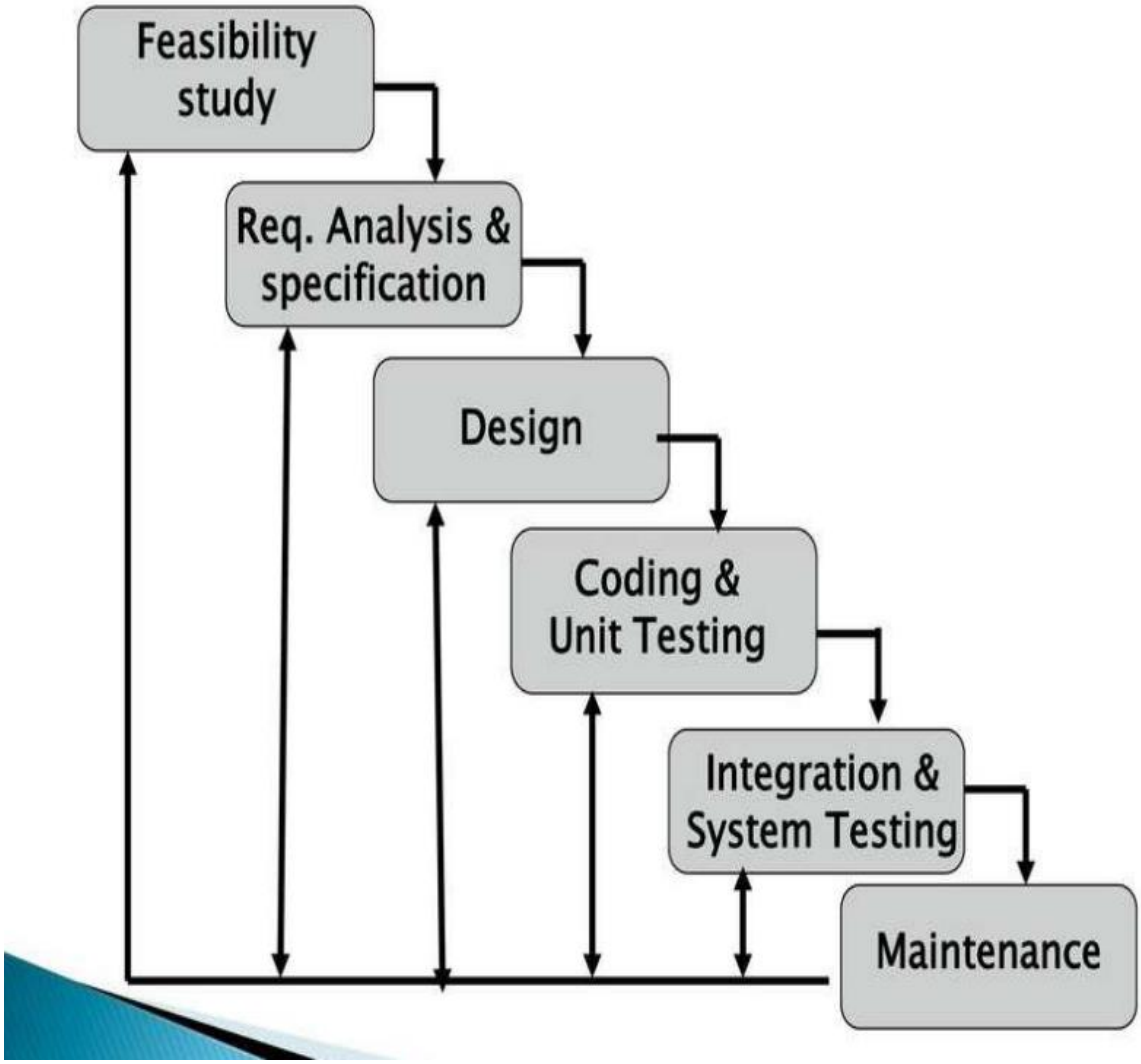
• Acceptance Testing.

Maintenance:

• Maintenance of a typical software products requires much more than the effort necessary to develop the product itself.

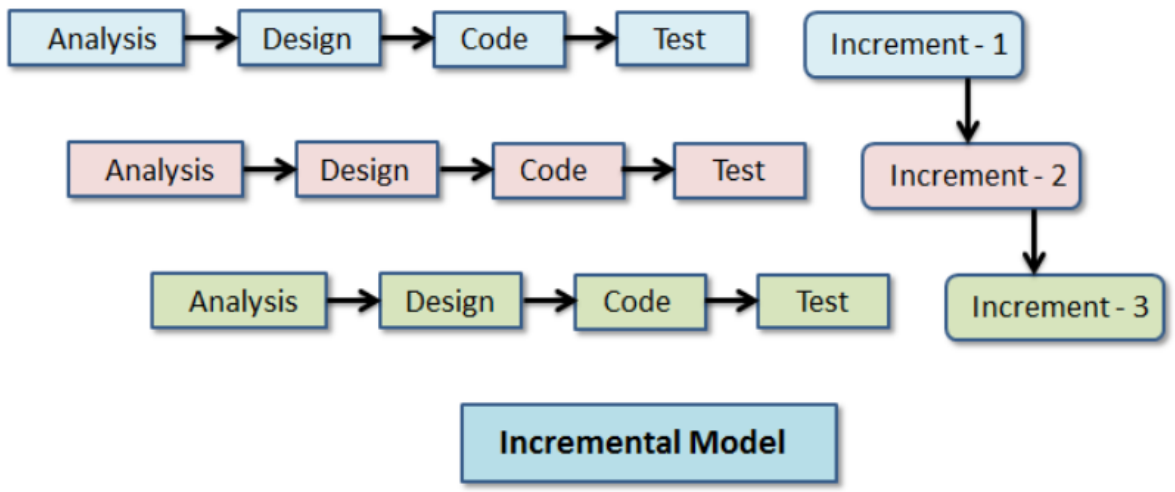
Iterative Waterfall life cycle model:

The main changes are done by providing feedback paths from every phase to its preceding phase.



INCREMENTAL PROCESS MODELS:

Incremental Model is one of the most adopted models of software development process where the software requirement is broken down into many standalone modules in the software development life cycle. Once the modules are split then incremental development will be carried out in steps covering all the analysis, designing, implementation, carrying out all the required testing or verification and maintenance.



Phases of Incremental Model:

Let us look at each stage in each incremental phase development. Let's imagine that we are developing second phase and first phase is already developed and 100% working

Phase in Each Increment	Description
Requirement Analysis	Complete analysis is performed on the requirement and make sure that this requirement will be compatible to previously developed
Design	Once the requirement for this particular increment understood and clear then design will be drafted on how to implement and archive this requirement.
Code	Now the coding is performed in accordance to achieve the purpose of the requirements. All the coding standards will be followed without any defaults and unnecessary hard codes

Test	This is the last in the incremental phase where aggressive testing is performed on the developed code and defects are reported and resolved

Advantages of the Incremental Model:

Let's see few of the advantages of this particular model.

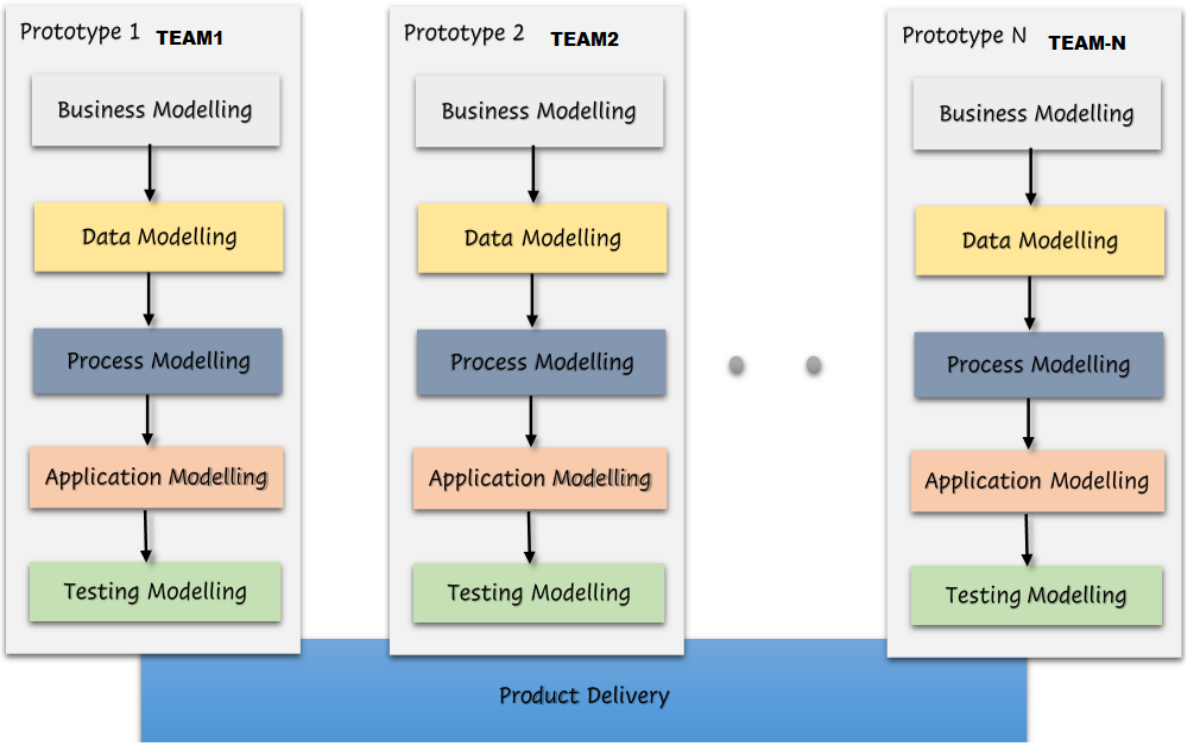
- Since the object will be divided into incremental stages, it will be made sure that 100% requirements are achieved and 100% objective of the software.
- Since there is testing at each incremental phase there will be multiple testing's for the software and more the testing better the result and fewer defects.
- By adopting this approach, we can lower the initial delivery cost.
- The user or the customer can provide feedback on each stage so work effort will be valued and sudden changes in the requirement can be prevented.
- Compared to the other model this model is tend to be cheaper on the pockets of the user.
- By following this models' errors can be identified quiet easily.

Rapid Application Development Model (RAD Model):

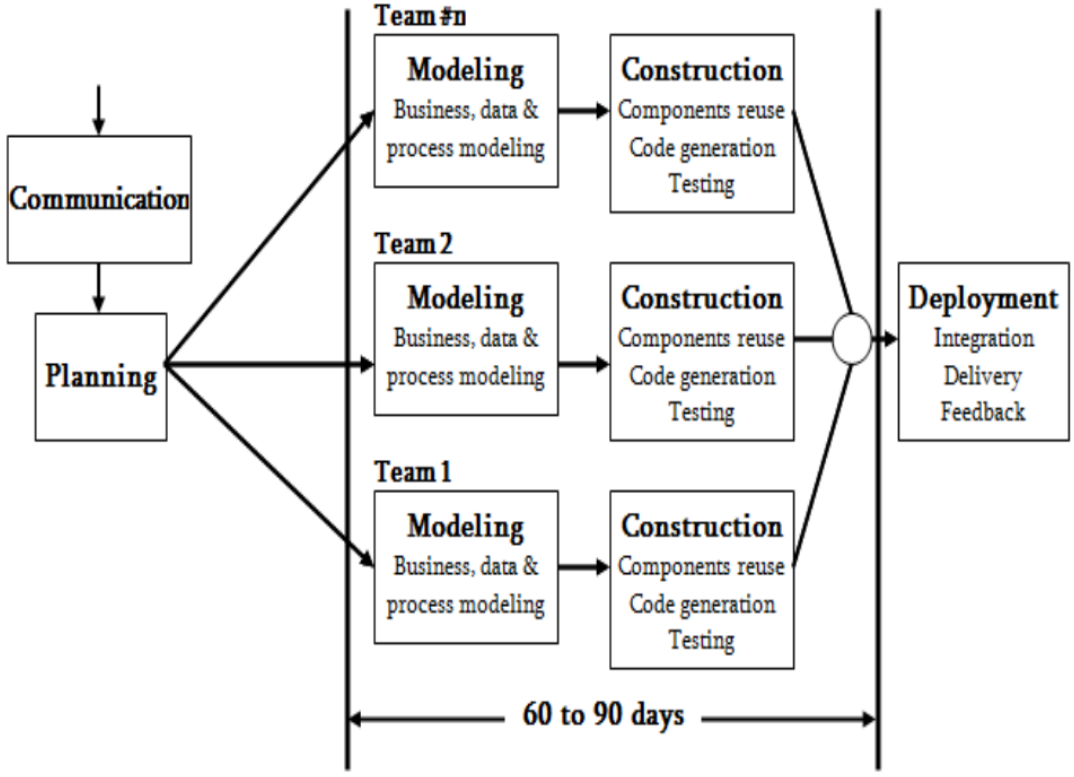
Rapid Application Development (RAD) is an incremental software model that a short development cycle.

• The RAD model is a "high-speed" of the waterfall model.

• The RAD process enables a development team to create a fully functional system within a very short time period.



Flow Chart of RAD Model:



Contents of RAD Packages:

- Graphical user development environment.
- Reusable Components.
- Code generator.
- Programming Language.

Advantages:

- Fast products.
- Efficient Documentation.
- Interaction with user.

Disadvantages:

- User may not like fast activities.
- Not suitable for technical risks.

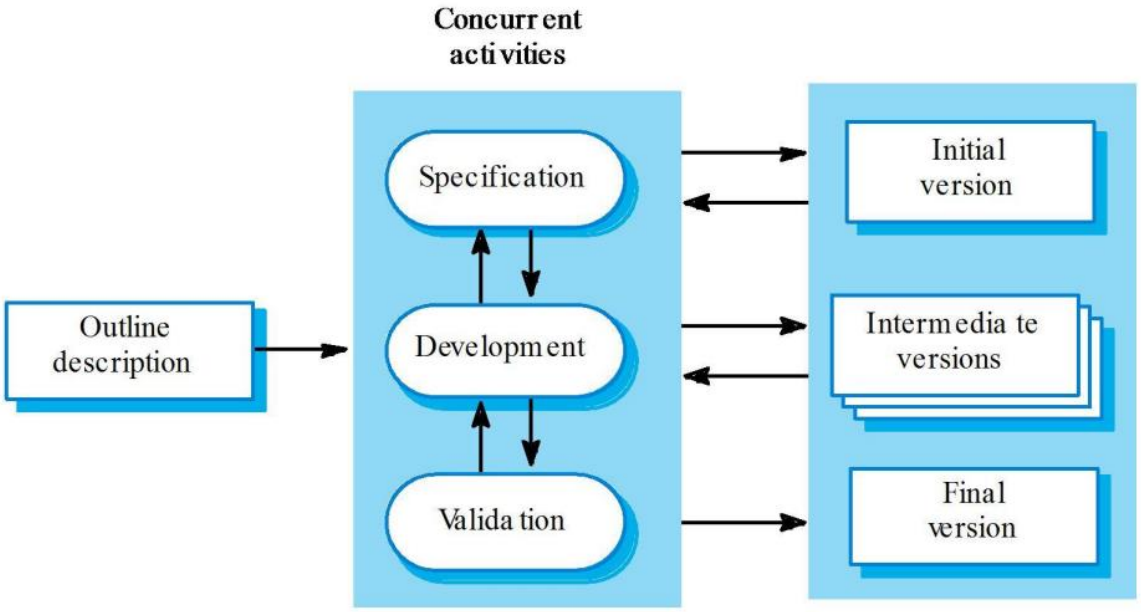
EVOLUTIONARY PROCESS MODELS:

Software evolves over a period of time Business and product requirements often change as development proceeds making a straight-line path to end product unrealistic Evolutionary models are iterative and as such are applicable to modern day applications

Types of Evolutionary Models:

- **Prototyping Model**
- **Spiral Model**
- **Concurrent Development Model**

Evolutionary Development:



Problems

- **Lack of process visibility;**
- **Systems are often poorly structured;**
- **Special skills (e.g. in languages for rapid prototyping) may be required.**

Applicability

- **For small or medium-size interactive systems;**
- **For parts of large systems (e.g. the user interface);**
- **For short-lifetime systems.**

Prototyping Model:

Process of developing a working replication of a product or system that has to be engineered. A prototype usually turns out to be a very crude version of the actual system, possibly exhibiting limited functional capabilities, low reliability, and inefficient performance as compared to actual software.

Prototyping Model is a software development model in which prototype is built, tested, and reworked until an acceptable prototype is achieved.

Used to build the initial prototype supporting only the basic functionality as desired by the customer. Once the problems are encountered by the customer, the

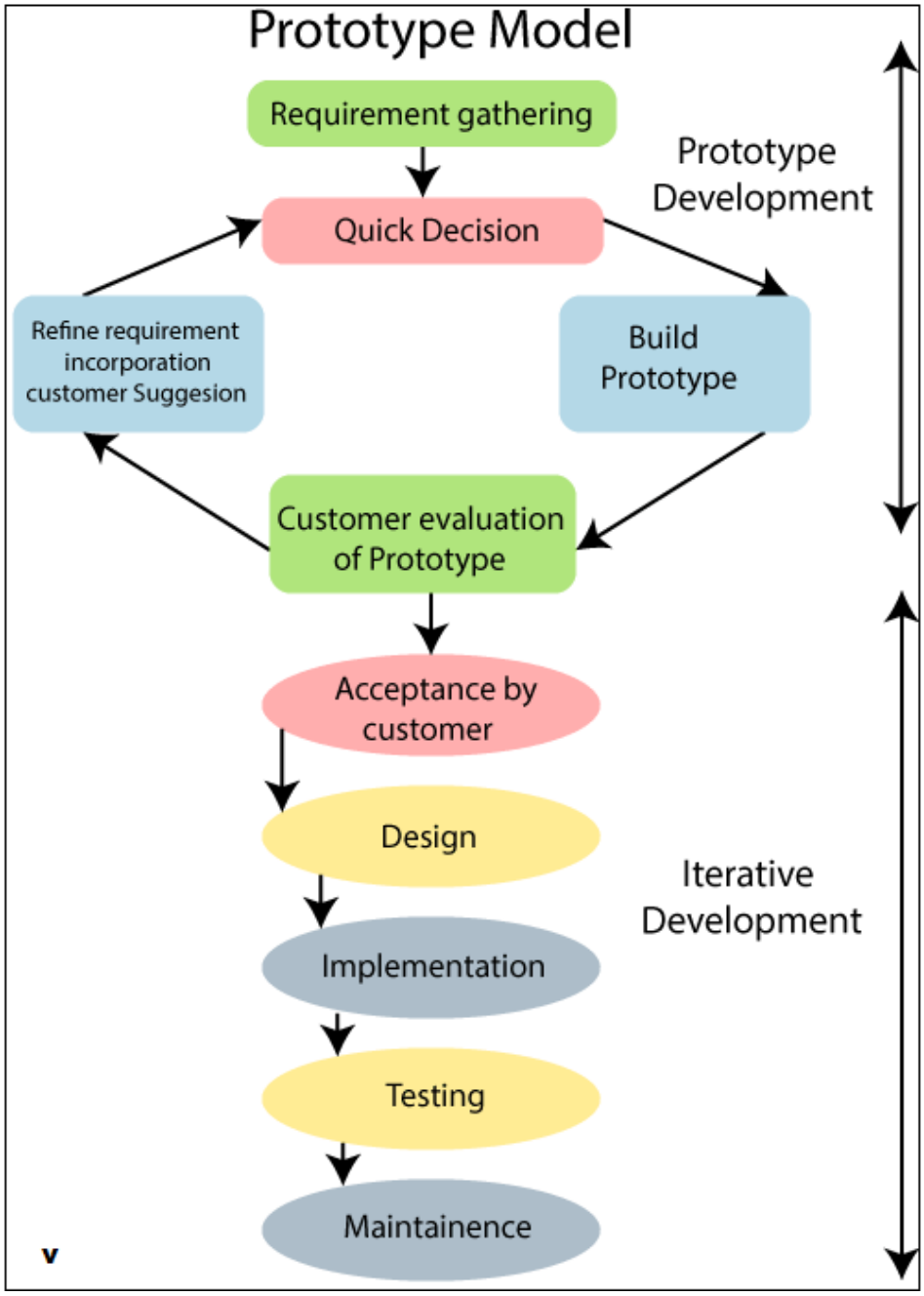
prototype is further refined to eliminate them. The process continues until the user approves the prototype and finds the working model to be satisfactory.

Advantages:

- In the development process of this model users are actively involved.
- The development process is the best platform to understand the system by the user.
- Errors are detected much earlier.

Disadvantages:

- An unstable implemented prototype often becomes the final product.
- The client involvement is more and it is not always considered by the developer.
- Require extensive customer collaboration
    - Costs customer money
    - Needs committed customer
    - Difficult to finish if customer withdraw
    - May be too customer specific, no broad market

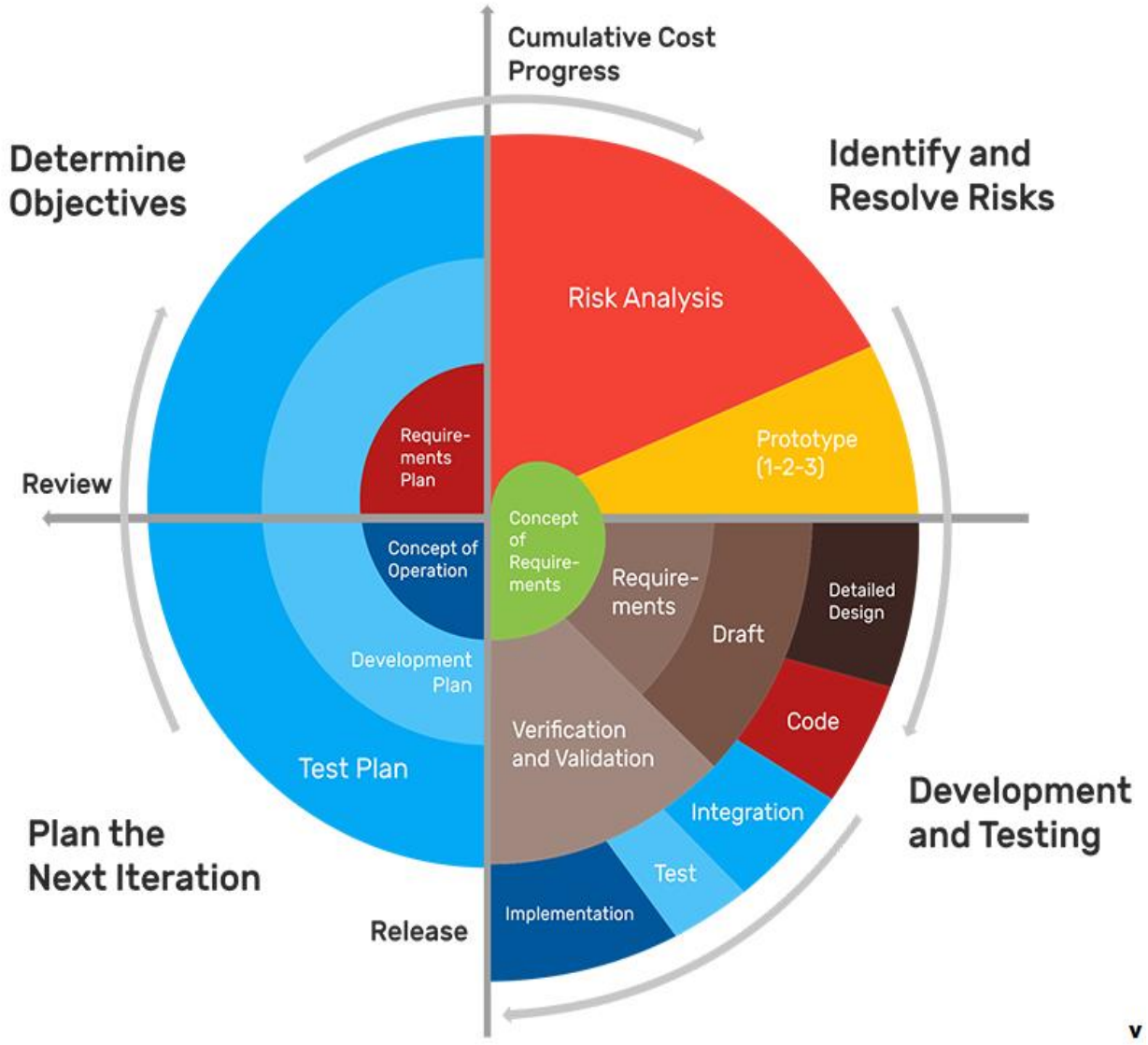


Spiral Model:

Spiral Methodology is unique as it incubates a working prototype at the end of each mini-spiral. 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.

Each spiral is composed of four stages:

- Planning
- Risk analysis
- Engineering
- Customer Review.



Planning	• It includes estimating the cost, schedule and resources for the iteration. It also involves understanding the system requirements for continuous communication between the system analyst and the customer
Risk Analysis	• Identification of potential risk is done while risk mitigation strategy is planned and finalized
Engineering	• It includes testing, coding and deploying software at the customer site
Evaluation	• Evaluation of software by the customer. Also, includes identifying and monitoring risks such as schedule slippage and cost overrun

Advantages:

- Risk Identification at early stage.
- Suitable for high-risk projects.
- Flexibility for adding functionality.
- Cost estimation becomes easy as the prototype building is done in small fragments

Disadvantages:

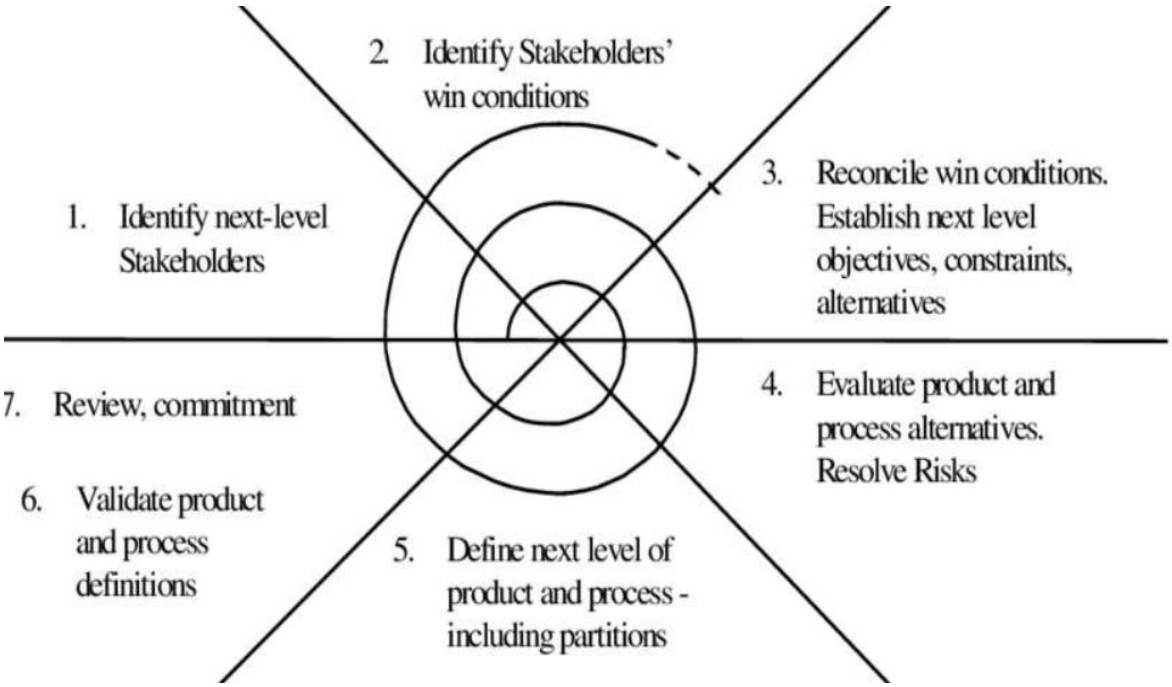
- Costly.
- Risk dependent.
- Not suitable for smaller projects.
- Difficult to meeting budget.

Win-Win Sprial Model:

- The customer wins by getting the system satisfying most of thier requirements and developers winsby working on achievable budgets and deadlines.
- **Advantages:**
- Lieghtweight methods suit small-medium size project.
- Produces good team.
- Test based approach to requirements and quality assurance

Diadvantages:

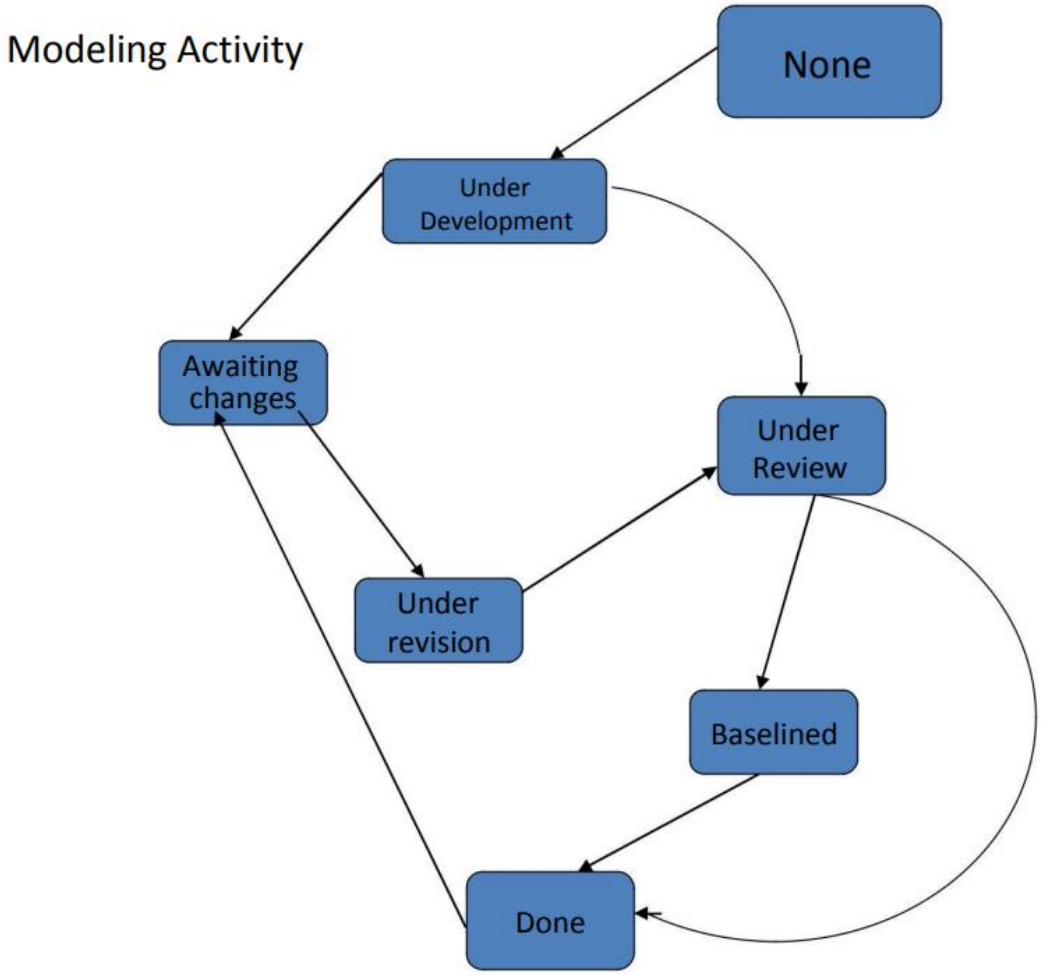
- Programming pairs is costly.
- Difficult to scale up to large projects where documentation.



Concurrent Development Model:

Constitutes a series of events that will trigger transitions from state to state for each of the software engineering activities framework activities, actions or tasks It can be represented as a series of framework activities, software engineering actions and tasks and their associated states

- •All activities exist concurrently but reside in different states
- •Applicable to all types of software development
- •Event generated at one point in the process trigger transitions among the states

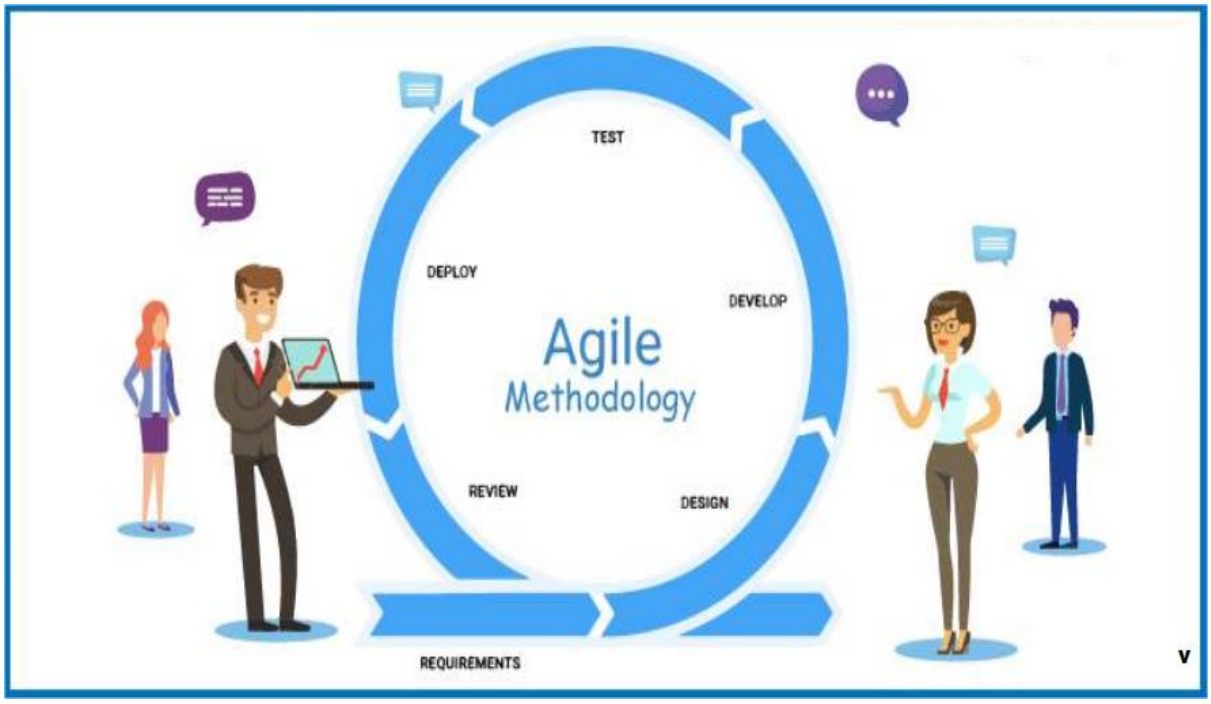


The Modeling activity which existed in the 'none' state while initial communication was completed, now makes a transition into the 'under development' state.

If the customer indicates that changes in requirements must be made, the Modeling activity moves from under development state into awaiting changes state.

AGILE PROCESS MODEL:

Agile Modelling defines Agile modelling as "…a practice-based methodology for effective modelling and documentation of software-based systems. Simply put, Agile Modelling (AM) is a collection of values, principles, and practices for modelling software that can be applied on a software development project in an effective and lightweight manner."



What Are Agile Modeling's Core Principles?

The modelling embraces 11 core Principles.

1. Model with a Purpose:

Ask why you're developing the models and who you are developing for them. (Developer or contractor)

2. Adopt Simplicity:

Keep the models as straight forward and uncomplicated as possible, and believe the simplest solution is also the best solution.

3. Embrace Change:

The more your understanding of a project grows, the more likely it will change. Instead of fighting change, accept them and have the courage to readjust and rebuild.

4. Your Secondary Goal is Enabling the Next Effort:

Your successors might have to improve or enhance your project after you depart. Leave them enough documentation and models to expedite possible changes or improvements.

5. Incremental Change:

It's rare for a model to be complete on the first try. Models evolve as the project grows and develops. You can cushion against the shock of change by making minor changes to the models as needed.

6. Maximize Stakeholder Investment:

The team must make the best effort to develop software that meets the stakeholder's needs. Bear in mind, the whole purpose of producing the software is to maximize the return for the client.

7. Remember the Existence of Multiple Models:

There are many modelling solutions available, so pick the ones that fit the current situation best. Additionally, there are many different methods of software delivery.

8. Produce Quality Work:

Nobody wants careless, rushed work. The developer doesn't like it because they know it's not something they can be proud of deep down. The teams that come later to check the work don't like it because sloppy work is challenging to understand and means more hours spent fixing it.

And finally, the end-users won't like the sub-par work because it most likely won't function properly or doesn't meet their expectations.

9. Provide Rapid Feedback:

Receiving timely feedback on the model closes the model's loop of understanding. Model a small portion—show it to the appropriate parties for review—then model again.

10. Make Working Software Your Primary Goal:

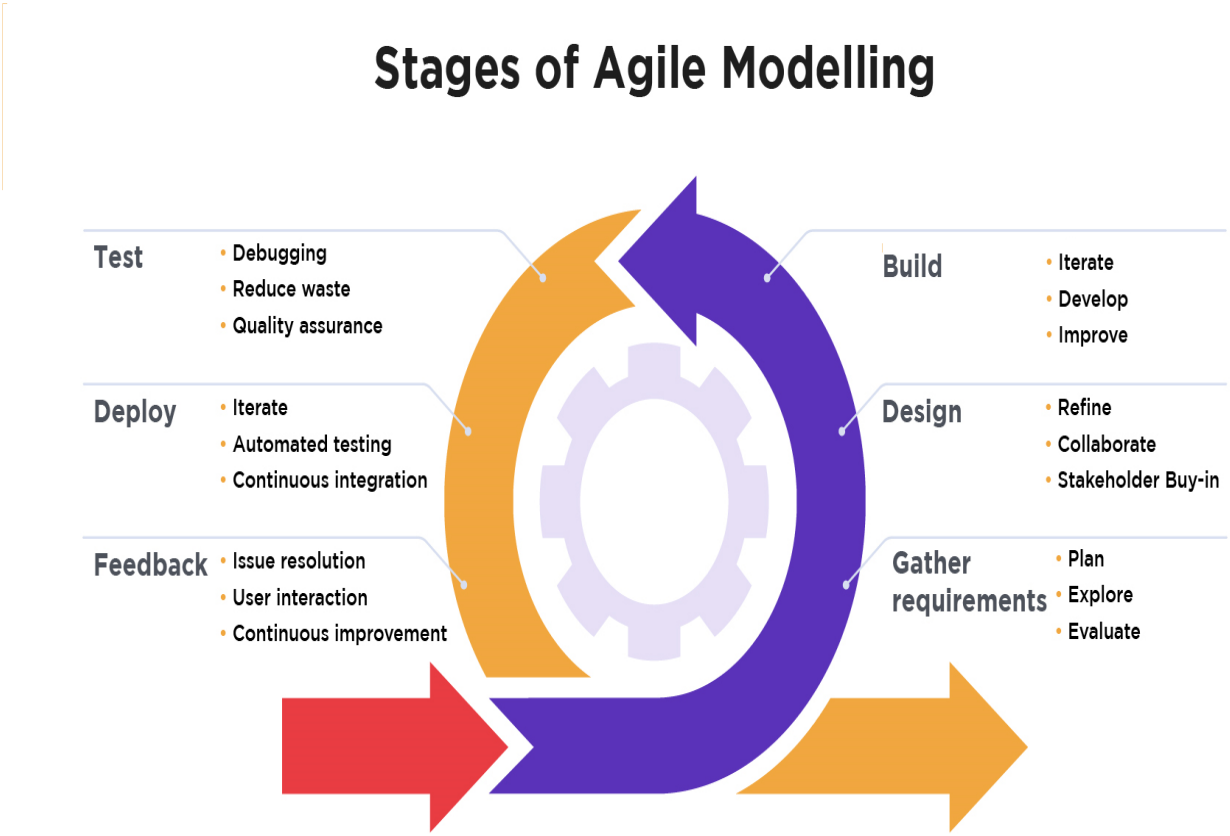
Models are just a means to the end, which is building great software for your customer. Make sure that documentation and modelling directly support the goal of your software development project.

11. Travel Light:

Traveling light is another way of saying that you have sufficient documentation about the models you're developing, but no more than that.

If you have too little documentation, the developing team might lose its way—if you have too much, the development team may forget that the primary goal is not writing documentation but instead building software and the right models!

Phases of the Agile Model:



- **Requirements Gathering:**

Here is where you define the project's requirements. This phase includes explaining business opportunities and planning the time and effort required for the project. Once you quantify this information, you can evaluate the technical and economic feasibility of your project

- **Design the Requirements:**

Once you've identified the project parameters, work with the stakeholders to define the requirements.

- **Construction/Iteration:**

After the team defines and designs the requirements, the real work begins. Product, design, and developer teams start working on related projects, ultimately deploying a product or service that is not static.

- **Testing:**

The quality assurance (QA) team examines and evaluates the product's performance, looking for bugs and other flaws.

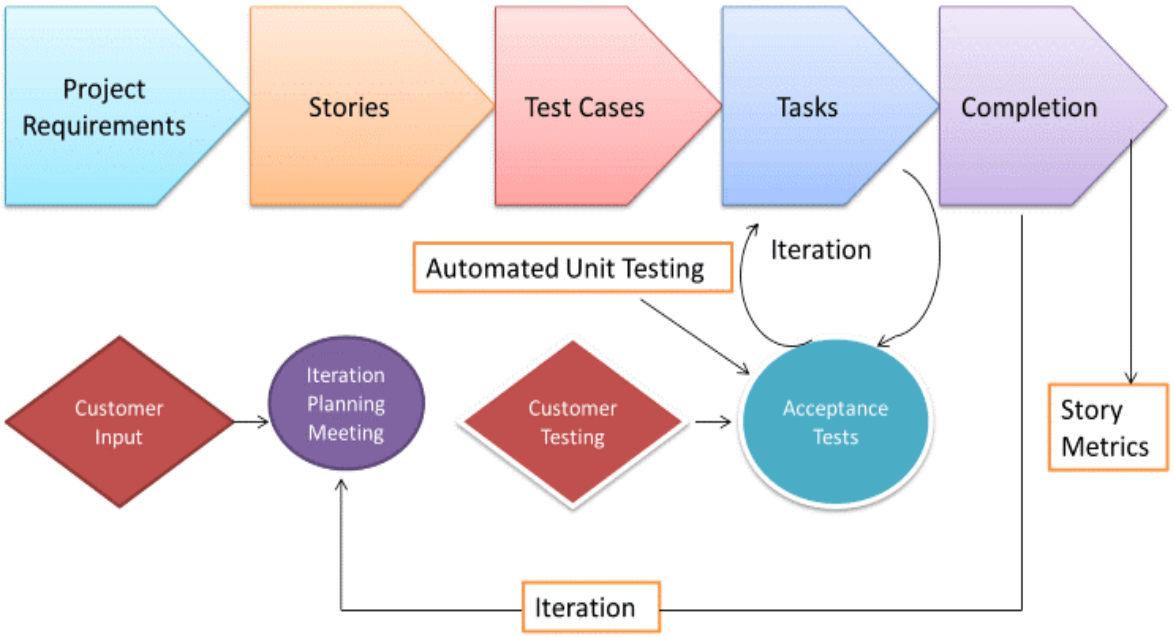
- **Deployment:**

The team deploys the product in a working environment.

- **Feedback:**

Once the product is released, the team receives feedback about the product and handles any issues that may have arisen.

The modelling adds to existing Agile methodologies such as the Rational Unified Process (RUP) or extreme programming (XP).



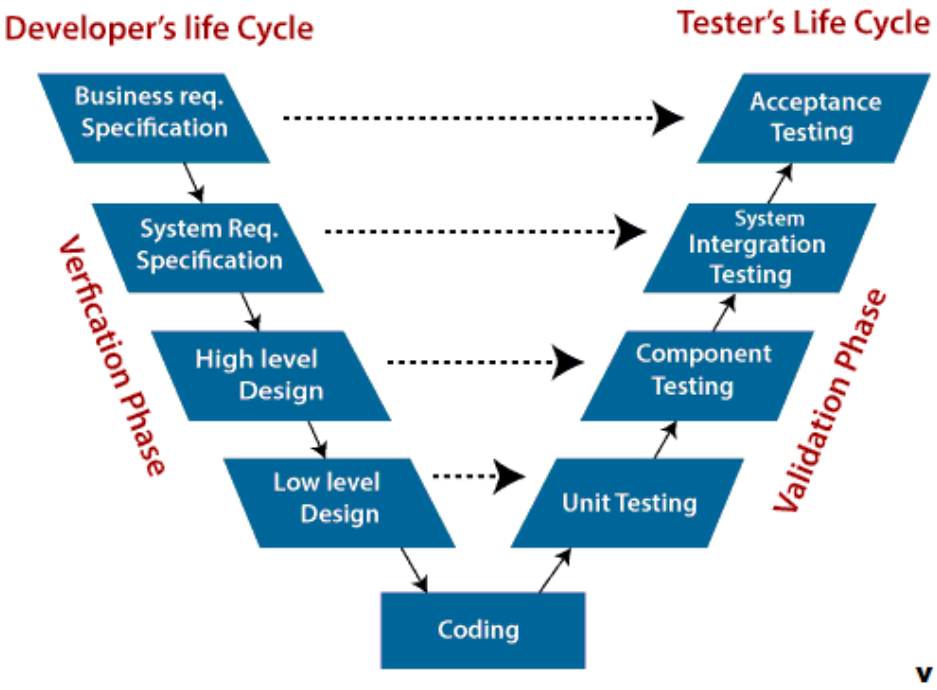
Extreme Programming (XP)

V MODEL:

V-Model also referred to as the Verification and Validation Model. In this, each phase of SDLC must complete before the next phase starts. It follows a sequential design process same as the waterfall model. Testing of the device is planned in parallel with a corresponding stage of development.

Verification: It involves a static analysis method (review) done without executing code. It is the process of evaluation of the product development process to find whether specified requirements meet.

Validation: It involves dynamic analysis method (functional, non-functional), testing is done by executing code. Validation is the process to classify the software after the completion of the development process to determine whether the software meets the customer expectations and requirements.



There are the various phases of Verification Phase of V-model:

1. **Business requirement analysis:** This is the first step where product requirements understood from the customer's side. This phase contains detailed communication to understand customer's expectations and exact requirements.
2. **System Design:** In this stage system engineers analyze and interpret the business of the proposed system by studying the user requirements document.
3. **Architecture Design:** The baseline in selecting the architecture is that it should understand all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology detail, etc. The integration testing model is carried out in a particular phase.
4. **Module Design:** In the module design phase, the system breaks down into small modules. The detailed design of the modules is specified, which is known as Low-Level Design
5. **Coding Phase:** After designing, the coding phase is started. Based on the requirements, a suitable programming language is decided. There are some guidelines and standards for coding. Before checking in the repository, the

final build is optimized for better performance, and the code goes through many code reviews to check the performance.

There are the various phases of Validation Phase of V-model:

1. **Unit Testing:** In the V-Model, Unit Test Plans (UTPs) are developed during the module design phase. These UTPs are executed to eliminate errors at code level or unit level. A unit is the smallest entity which can independently exist, e.g., a program module. Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/ units.
2. **Integration Testing:** Integration Test Plans are developed during the Architectural Design Phase. These tests verify that groups created and tested independently can coexist and communicate among themselves.
3. **System Testing:** System Tests Plans are developed during System Design Phase. Unlike Unit and Integration Test Plans, System Tests Plans are composed by the client's business team. System Test ensures that expectations from an application developer are met.
4. **Acceptance Testing:** Acceptance testing is related to the business requirement analysis part. It includes testing the software product in user atmosphere. Acceptance tests reveal the compatibility problems with the different systems, which is available within the user atmosphere. It conjointly discovers the non-functional problems like load and performance defects within the real user atmosphere.

Advantages of V-Model in Software Engineering

1. Very simple and easy to use SDLC V model.
2. Very easy to manage as the development happens in a sequential manner and the next phase starts only when the previous phase is complete.
3. Ideal model for short and rigid projects.
4. Verification and validation at each step of development ensures that the final product will be bug-free.
5. Project management is quite easy as compared to other software development models.

Disadvantages OF V-Model in Software Engineering

1. Not suitable for complex projects.
2. Not suitable for projects having unclear or changing requirements.
3. Not suitable for projects wanting concurrency in the development phase.
4. Very risky and uncertain SDLC V model.

SOFTWARE REQUIREMENTS (Part – II)

Functional and Non-Functional Requirements:

What is a Functional Requirement?

A **Functional Requirement** (FR) is

- a description of the service.
- It describes a software system or its component.
- A function is nothing but inputs to the software system, its behavior, and outputs.
- It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform.
- Functional Requirements in Software Engineering are also called **Functional Specification**.

NON-FUNCTIONAL REQUIREMENTS:

The 10 Main Types of Non-Functional Requirements:

Where most teams make mistakes with NFRs.

Your team won't "forget" to think about performance issues or test reliability. But are you planning for localization, regulatory issues, or maintainability?

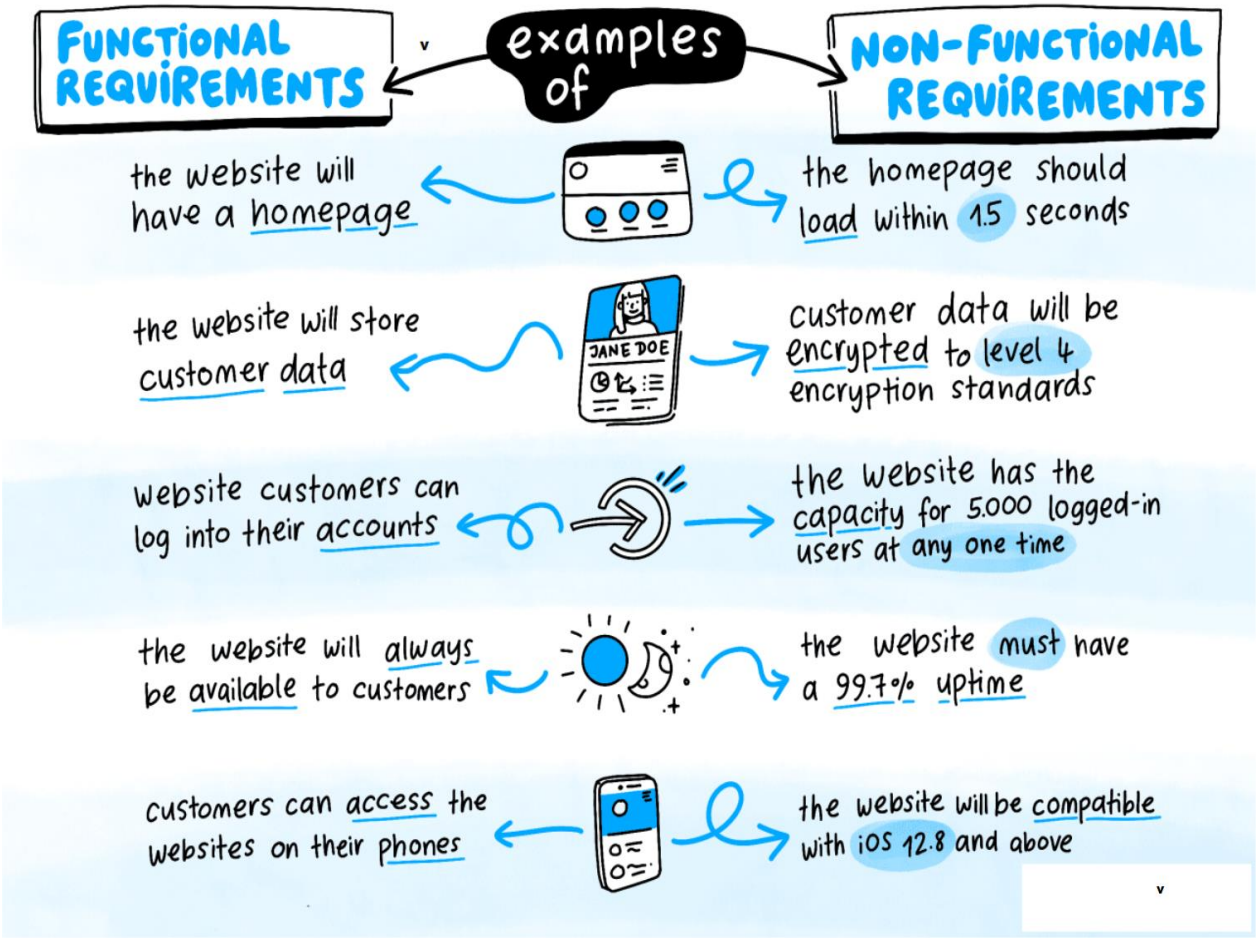
A good product manager needs to consider all ten non-functional requirements during the development phase. That includes:

Non-Functional requirement types and what to consider:	
1. Performance 👉	Focuses on the system's speed, efficiency, and workload. I.e., how fast does the system respond?
2. Scalability 👉	Ensures the system can respond to changes in demand. I.e., how will the system pull on additional resources?

3. Reliability 👉	Defines the system's availability and the tolerance for failure I.e., what's the target uptime?
4. Resilience 👉	Defines how quickly a system can recover if it fails. I.e., how does the reset process work?
5. Security 👉	Focuses on how the system is kept secure, stores data, and responds to attacks. I.e., what are the security protocols of the site?
6. Usability 👉	Specifies how systems should operate for the customer/end-user. I.e., how many clicks to get to a certain place?
7. Maintainability 👉	Ensures the system is easy to upgrade and troubleshoot. I.e., what format is the error log?
8. Modifiability 👉	Defines how the system can be changed if required. I.e., how can users customize certain features? How can developers adjust the code?

9. Localization 👉	Specifies how a system will adapt to the user's location. I.e., how will the system detect and display different languages/currencies?
10. Regulatory 👉	Ensures the system is legally compliant. I.e., how does this system comply with ISO27001?

Examples of Functional and Non-Functional Requirements:



Non-Functional Requirements measures

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	M Bytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Now, let's see the comparison chart between the functional and non-functional requirements.

Functional Requirements	Non-functional requirements
Functional requirements help to understand the functions of the system.	They help to understand the system's performance.
Functional requirements are mandatory.	While non-functional requirements are not mandatory.
They are easy to define.	They are hard to define.
They describe what the product does.	They describe the working of product.
It concentrates on the user's requirement.	It concentrates on the expectation and experience of the user.
It helps us to verify the software's functionality.	It helps us to verify the software's performance.

These requirements are specified by the user.	These requirements are specified by the software developers, architects, and technical persons.
There is functional testing such as API testing, system, integration, etc.	There is non-functional testing such as usability, performance, stress, security, etc.
Examples of the functional requirements are - Authentication of a user on trying to log in to the system.	Examples of the non-functional requirements are - The background color of the screens should be light blue.
These requirements are important to system operation.	These are not always the important requirements; they may be desirable.
Completion of Functional requirements allows the system to perform, irrespective of meeting the non-functional requirements.	While system will not work only with non-functional requirements.

FUCNTIONAL REQUIREMENTS:

Functional requirements define a function that a system or system element must be qualified to perform and must be documented in different forms. The functional requirements describe the behavior of the system as it correlates to the system's functionality.

Functional requirements should be written in a simple language, so that it is easily understandable. The examples of functional requirements are authentication, business rules, audit tracking, certification requirements, transaction corrections, etc.

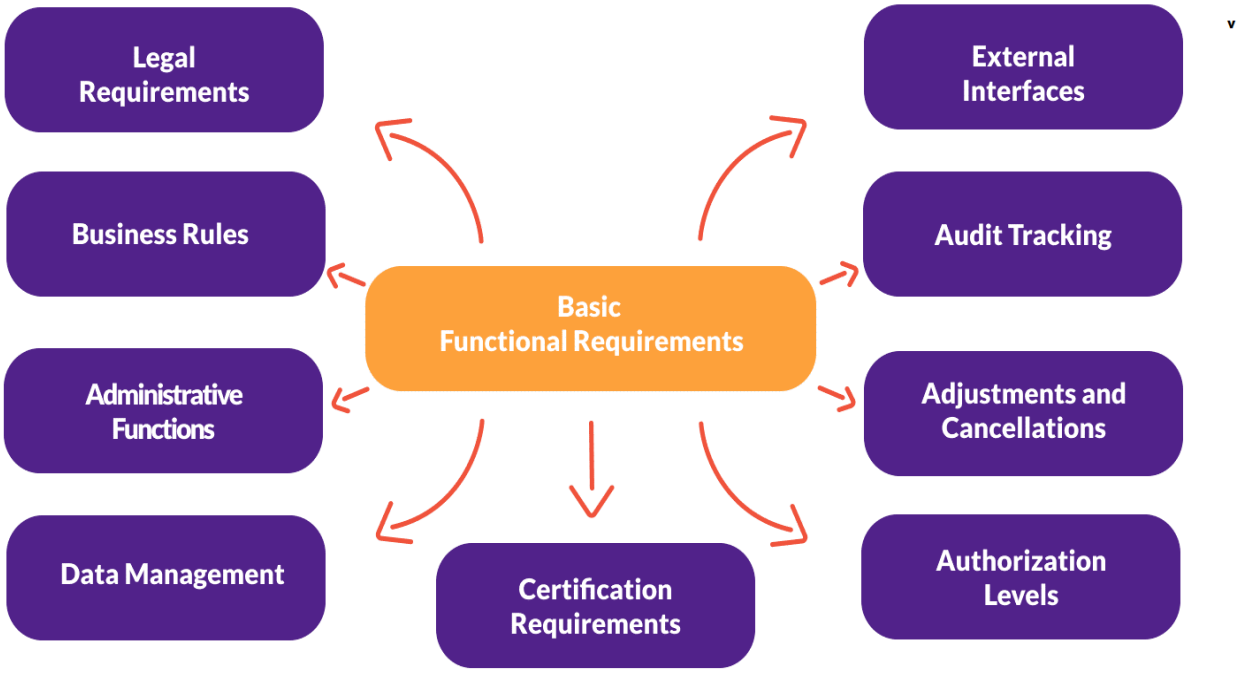
These requirements allow us to verify whether the application provides all functionalities mentioned in the application's functional requirements. They support tasks, activities, user goals for easier project management.

There are a number of ways to prepare functional requirements. The most common way is that they are documented in the text form. Other formats of preparing the functional requirements are use cases, models, prototypes, user stories, and diagrams.

Example of Functional Requirements:

Functional Requirement No.	Function Requirement Description
FR 1	User should be able to enter Sales Data
FR 2	Sales Reports should be generated every 24 hours
FR 3	API interface to Invoice System

Types of Functional Requirements:



Functional Requirements of a system should include the following things:

- Details of operations conducted in every screen
- Data handling logic should be entered into the system
- It should have descriptions of system reports or other outputs
- Complete information about the workflows performed by the system
- It should clearly define who will be allowed to create/modify/delete the data in the system

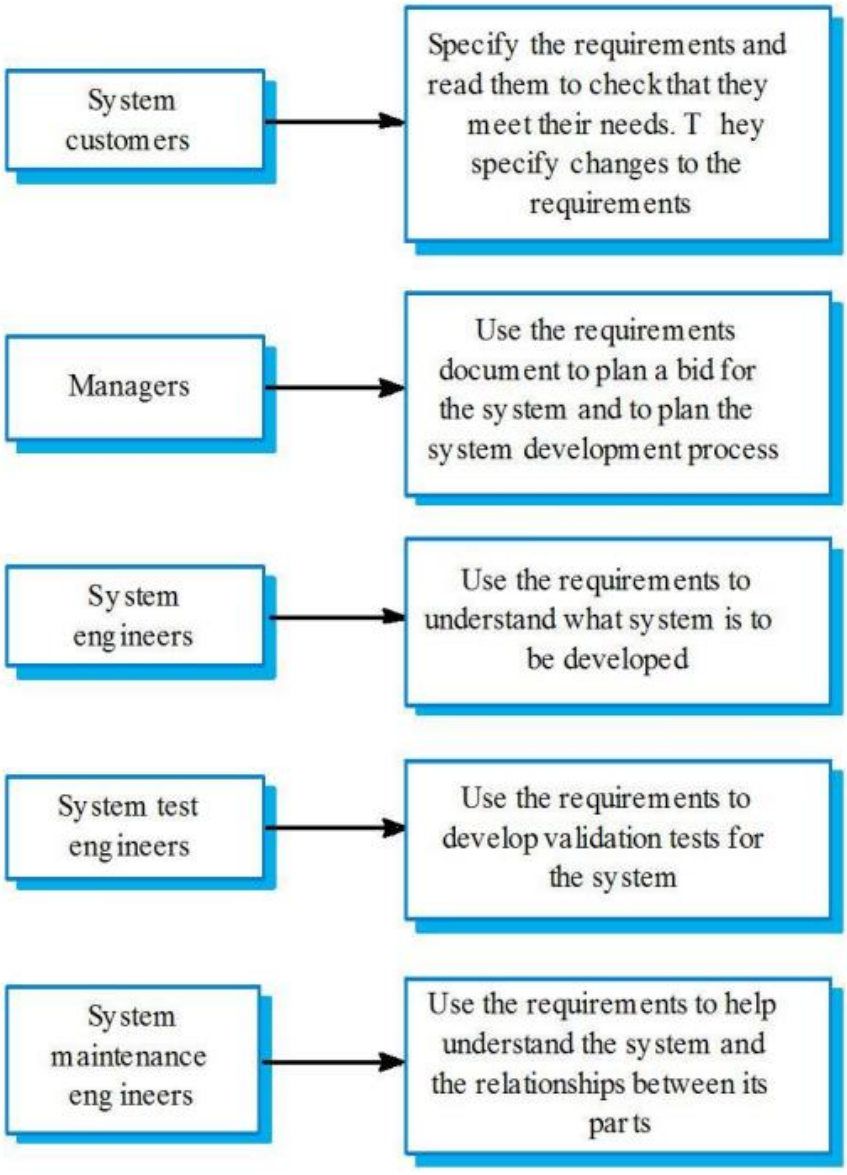
- How the system will fulfil applicable regulatory and compliance needs should be captured in the functional document

Software Requirements Document (SRS):

The Software requirements document is the official statement of what is required of the system developers. Should include both a definition of user requirements and a specification of the system requirements.

It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.

Users of SRS Document:



Purpose of SRS:

Communication between the Customer, Analyst, System Developers, Maintainers Firm foundation for the design phase Support system testing activities Support project management and control Controlling the evolution of the system.

Structure of Requirements Document (IEEE Standard):

Defines a generic structure for a requirements document that must be instantiated for each specific system.

- Introduction.
- General description.
- Specific requirements.
- Appendices.
- Index.

IEEE Requirements Standard:

1.Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions, Acronyms and Abbreviations

1.4 References

1.5 Overview

2. General description

2.1 Product perspective

2.2 Product function summary

2.3 User characteristics

2.4 General constraints

2.5 Assumptions and dependencies

3. Specific Requirements

3.1 Functional requirements

3.2 External interface requirements

3.3 Performance requirements

3.4 Design constraints

3.5 Attributes example: security, availability, maintainability, transferability/conversion

3.6 Other requirements

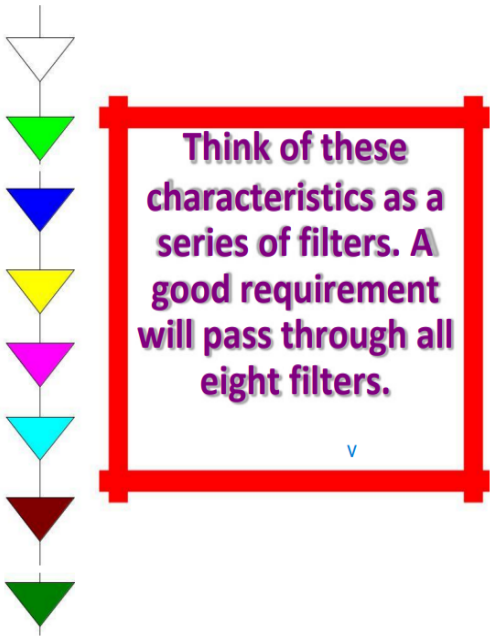
4. Appendices

5. Index

Characteristics of Good User & Software Requirement:

A user requirement is good if it is:

1. Verifiable
2. Clear and concise
3. Complete
4. Consistent
5. Traceable
6. Viable
7. Necessary
8. Implementation free



A verifiable requirement …

is stated in such a way that it **can be tested** by:
inspection,
analysis, or
demonstration.

makes it **possible to evaluate** whether the system met the requirement, and

is verifiable by **means that will not contaminate** the product **or compromise** the data integrity.

A clear & concise requirement …

must consist of a **single requirement**,

should be no more than **30-50 words** in length,

must be **easily read and understood** by non technical people,

must be **unambiguous** and not susceptible to multiple interpretations,

must **not contain** definitions, descriptions of its use, or reasons for its need, and

must **avoid** subjective or open-ended terms.

A complete requirement …

contains **all the information** that is needed to define the system function,

leaves **no one guessing** (For how long?, 50 % of what?), and

includes **measurement units** (inches or centimeters?).

A consistent requirement …

does not conflict with other requirements in the requirement specification,

uses the **same terminology** throughout the requirement specification, and

does not duplicate other URs or pieces of other URs or create redundancy in any way.

A traceable requirement …

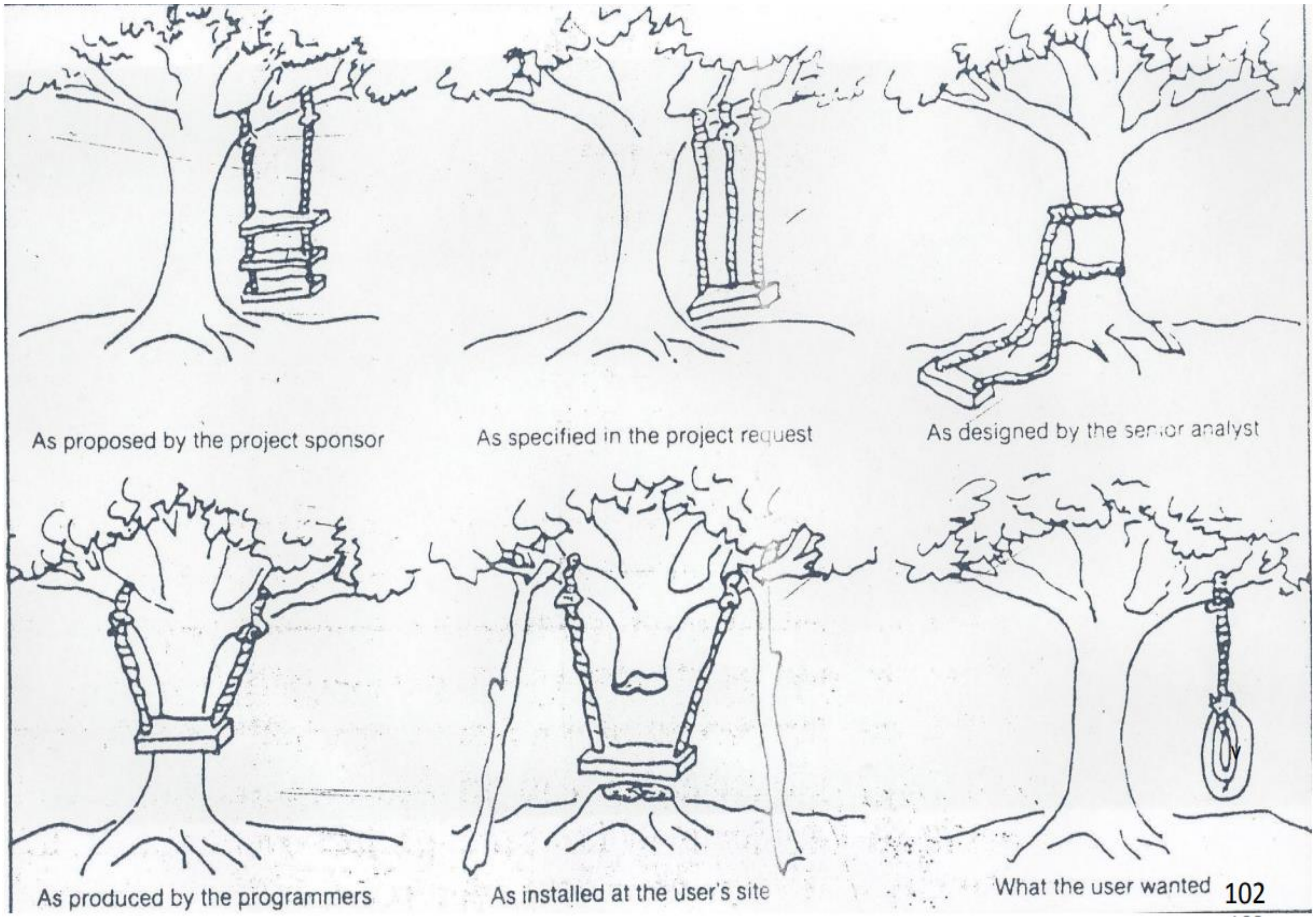
has a **unique identity** or number,

cannot be separated or broken into smaller requirements,

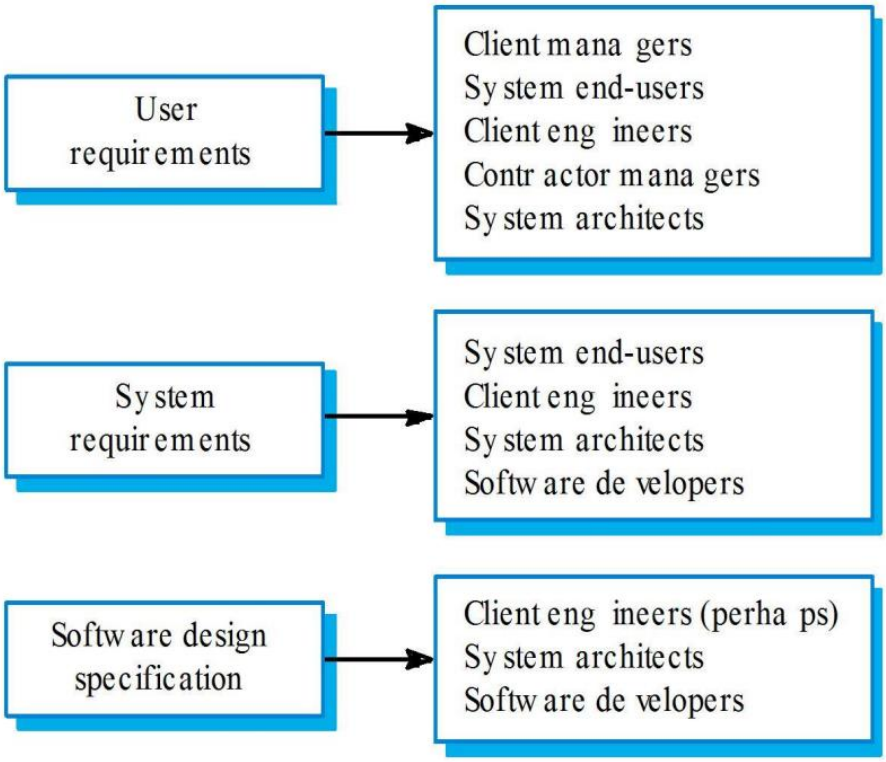
can **easily be traced** through to specification, design, and testing.

Change Control on UR level.

Example of Requirements Mismatch:



Requirements Readers:



Importance of SRS Document:

Steps to write Good SRS Document:

Tips for writing SRS Documentation:

SRS Vs FRS Vs BRD:

	Business Requirements Document (BRD)	Software Requirement Specifications (SRS)	Functional Requirement Specifications (FRS)
Other names		Product Requirements Document (PRD) and System Requirements Specification	Functional Specifications Document (FSD), Product Specification Document (PSD), Functional Specs (FS)
Created By	Business Analyst	Business/System Analyst	Business/System Analyst/Implementation Leads
Contains	High level business requirements and stakeholder requirements	Detailed functional requirements, non-functional requirements and use cases	Granular functional requirements, data flow and UML diagrams
Used By	Upper and middle management	Project managers, SMEs (subject matter experts), technical and implementation lead	Technical leads, development teams and testing teams.
Prepared in	Initiation phase	Planning phase	Planning phase
Answers	'Why' the requirements are being undertaken	'What' requirements must be fulfilled to satisfy business needs	'How' exactly the system is expected to function
Example	Improve efficiency by tracking the employee time in office	Proposed software will contain following modules: Login, Administrator, Employee and Reporting	Login module will contain fields like: Enter username, Enter password, Submit button