SOFTWARE ENGINEERING

UNIT-1

INTRODUCTION TO SOFTWARE ENGINEERING

1. Evolving Role of Software :

Software : Software is both a product and a vehicle that delivers a product.

Software delivers 'information'.

Def: Software is

a) Instructions that when executed provided desired features.

b) Data structures that enable the programs to adequately manipulate information.

c) Documents that describe the operation and use of the programs.

Software is developed or engineered.

The software is custom built.

2. Types of Software :

a) System Software : is a collection of programs written to service other programs. (eg. Compiler, editors and file management)

b) Application Software : consists of stand alone programs that solve a specific business need.

c) Engineering Software : is a set of number crunching programs.

d) Embedded Software : resides within a product or system and is used to implement and control features and functions for the end-user and for the system itself.

e) Product-line Software : designed to provide a specific capability for use by many different customers.

f) Web applications : span a wide array of applications

g) AI Software : makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis.

3. **Legacy Software** : have been the focus of continuous attention.

   Features of Legacy Software :

   a) The software must be adopted to meet the needs of new computing environment or technology.

   b) The software must be enhanced to implement new business requirements.

   c) The software must be extended to make it interoperable with more modern systems or databases.

   d) The software must be re-architected to make it viable within a network environment.

4. **Software Myths** : are beliefs about software and the process used to build it – can be traced to the earliest days of computing.

   a) Management Myths :

   Managers with software responsibility, like managers in most disciplines, are often under pressure to maintain budgets, keep schedules from slipping and improve quality.

   Myth : We already have a book that's full of standards and procedures for building software.

   Reality : The book of standards may very well exist, but is it used?

   Are software practitioners aware of its existence ?

   Does it reflect modern software engineering practice ?

   Is it complete ?

   Is it adaptable ?

Myth : If we get behind schedule, we can add more programmers.

Reality : Software development is not a mechanistic process like manufacturing.

"Adding people to a late software project makes it later".

b) CustomerMyths : A customer who request computer software may be a person in the next deck, a technical group down the hall, the marketing / sales department or an outside company that has requested software under contract.

Myth :A general statement of objectives is sufficient to begin writing programs.

Reality : Although a comprehensive and stable statement or require-ments is not always possible, an ambiguous statement of objectives is a recipe for disaster.

Myth : Project requirements continually change, but change can be easily accommodated because software is flexible.

Reality : It is true that software requirements change, but the impact of change varies with the time at which it is introduced.

c) **Practitioners Myths** : Myths that are still believed by software Practitioners have been fostered by over 50 years of programming culture.

Myth : Once we write the program and get it to work, our job is done.

Reality :Some one once said that the sooner you being writing code, the longer it will take you to get done.

Myth : Until I get the program running, I have no way of assessing its quality.

Reality : One of the most effective software quality assurance mechanisms can be applied from the inception of a project. This is called 'the formal technical review'.

5. The Problem at the development of Software :

a) Why does it take so long to get software finished.

b) Why are development costs so high.

c) Why can not we find all errors before we give the software to our customers.

d) Why do we spend so much time and effort maintaining existing programs.

e) Why do we continue to have difficulty in measuring progress as software is being developed and maintained.

6. Summary :

Software has become the key element in the evolution of computer-based systems and products and one of the most important technologies on the world stage.

Over the past 50 years, software has evolved from a specialized problem solving and information analysis tool to an industry in itself.

Yet we still have trouble developing high-quality software on time within budget.

Software – data, programs, and documents – addresses wide array of technology and application areas, yet all software evolves according to a set of laws that have remained the same for over 30 years.

“The intent of ‘software engineering’ is to provide a frame work for building higher quality software.”

* * * * *

7. Software Engineering – A Layered Technology :

Def : Software Engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

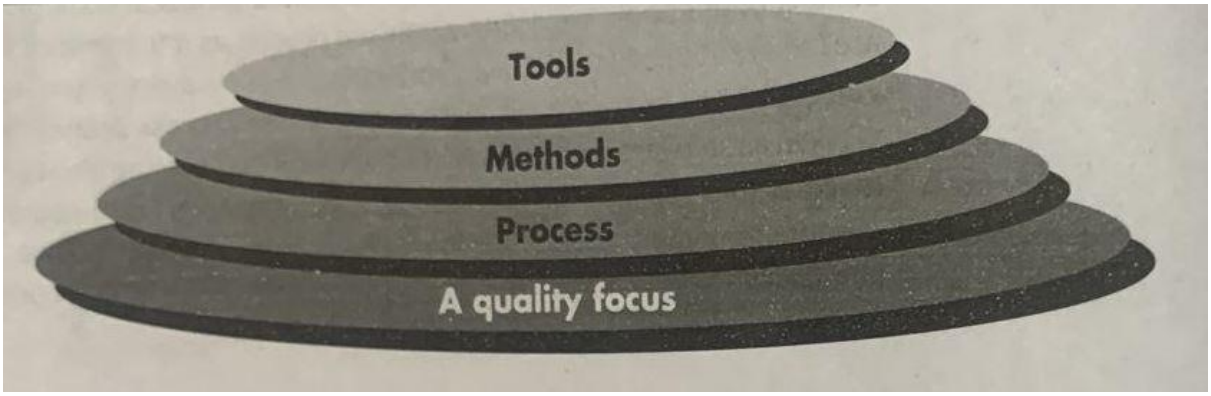
IEEE Definition :

"Software Engineering' is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software."

Software Engineering Layers :

Software Engineering is a layered technology.

Any engineering approach must rest on an organizational commitment to quality.



The foundation for the software engineering is the process layer.

Process defines a frame work that must be established for effective delivery of software engineering technology

The software process forms basis for management control of software projects.

Software Engineering methods provide the technical "how to 's'" for building software.

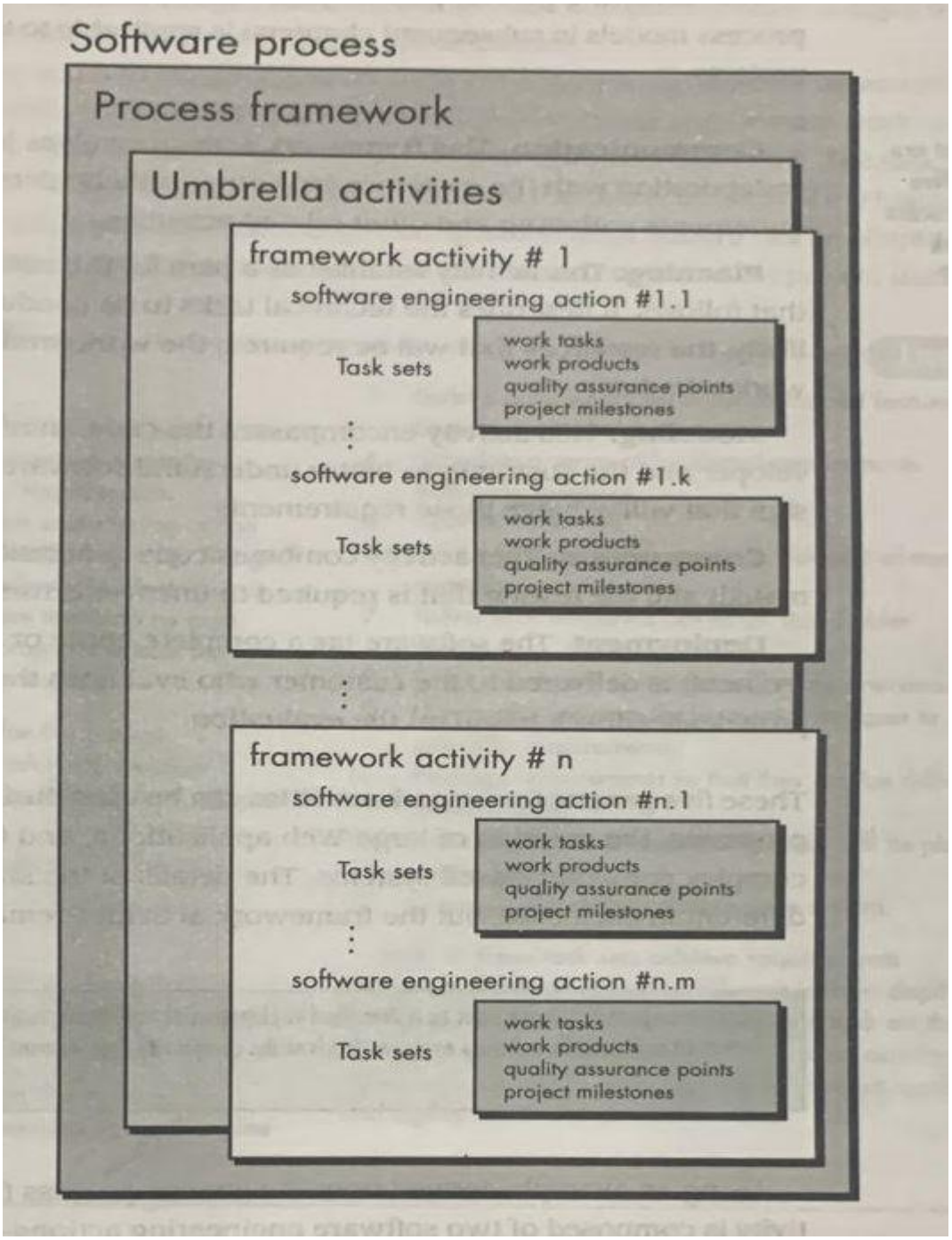
Methods encompass a broad array of tasks that include communication, requirement analysis, design modeling, program construction, testing and support.

Software Engineering Tools provide automated or semi-automated support for the process and the methods.

8. A Process Framework :

A process framework establishes the foundation for a complete software process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.

In addition, the process framework encompasses a set of 'umbrella activities' that are applicable across the entire software process :



In the above figure, each framework activity is populated by a set of software engineering actions – a collection of related tasks that produces a major software engineering work product. (Ex : design is a software engineering action).

Each action is populated with individual work tasks that accomplish some part of the work implied by the action.

Def: A process defines who is doing what, when and how to reach a certain goal.

The following Generic Process Framework is applicable to the vast majority of software projects.

The five Generic Process Framework Activities :

a) **Communication** : This framework activity involves heavy communication and collaboration with the customer and encompasses requirements gathering and other related activities.

b) **Planning :** This activity establishes a plan for the software engineering work that follows. It describes the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be developed, and a work schedule.

c) **Modeling** : This activity encompasses the creation of models that allow the developer and the customer to better understand software requirements and the design that will achieve those requirements.

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

e) **Deployment** : The software is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

The above five generic framework activities can be used

- during the development of small programs
- the creation of large web applications
- for the engineering of large, complex computer-based systems.

In the above process frame work, each software engineering action is represented by a number of different task sets – each a collection of

- software engineering work tasks
- related work products
- quality assurance points
- project milestones.

Task Set :defines the actual work to be done to accomplish the objectives of a software engineering action.

Typical activities in 'Umbrella Activities' :

a) **Software project tracking and control** – allows the software team to assess progress against the project plan and take necessary action to maintain schedule.

b) **Risk Management –** assesses risks that may effectthe outcome of the project or the quality of the product.

c) **Software Quality Assurance** – defines and conducts the activities required to ensure software quality.

d) **Formal Technical Reviews** – assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next action or activity.

e) **Measurement**- defines and collects process, project and product measures that assist the team in delivering software that meets customer needs.

f) **Software Configuration Management –** manages the effects of change throughout the software process.

g) **Reusability Management** – defines criteria for work product reuse and establishes mechanisms to achieve reusable components.

h) **Work Product Preparation and Production** – encompasses the activities required to create work products such as models, documents, logs, forms and lists.

9. The Capability Maturity Model Integration (CMMI) :

The Software Engineering Institute (SEI) has developed a comprehensive process meta-model that is predicated on a set of system and software engineering capabilities that should be present as organizations reach different levels of process capability and maturity.

To achieve these capabilities, the SEI contends that an organization should develop a process model that conforms to the 'capability maturity model integration (CMMI)' guidelines.

The CMMI represents a process meta-model in two different ways.

a) Continuous model
b) Staged model

The continuous CMMI meta-model describes a process in two dimensions.

Each process area is formally assessed against specific goals and practices and is rated according to the following capability levels :

Level-0 : Incomplete

The process area (eg. requirements management) is either not performed or does not achieve all goals and objectives defined by the CMMI for level-1 capability.

Level-1 : Performed

All of the specific goals of the process area have been satisfied. Work tasks required to produce defined 'work products' are being conducted.

Level-2 : Managed

All level-1 criteria have been satisfied. In addition, all work associated with the process are conforms to an organizationally defined policy; all people doing the work have access to adequate resources to get the job done, stake holders are actively involved in the process area as required.

All work tasks and work products are "monitored, controlled and reviewed" and are evaluated for adherence to the process description.

Level-3 : Defined

All level-2 criteria have been achieved. In addition, the process is "tailored form the organization's set of standard processes according to the organization's tailoring guidelines, and contribute work products, measures and other process-improvement information to the organizational process assets.

Level-4 : Quantitatively Managed

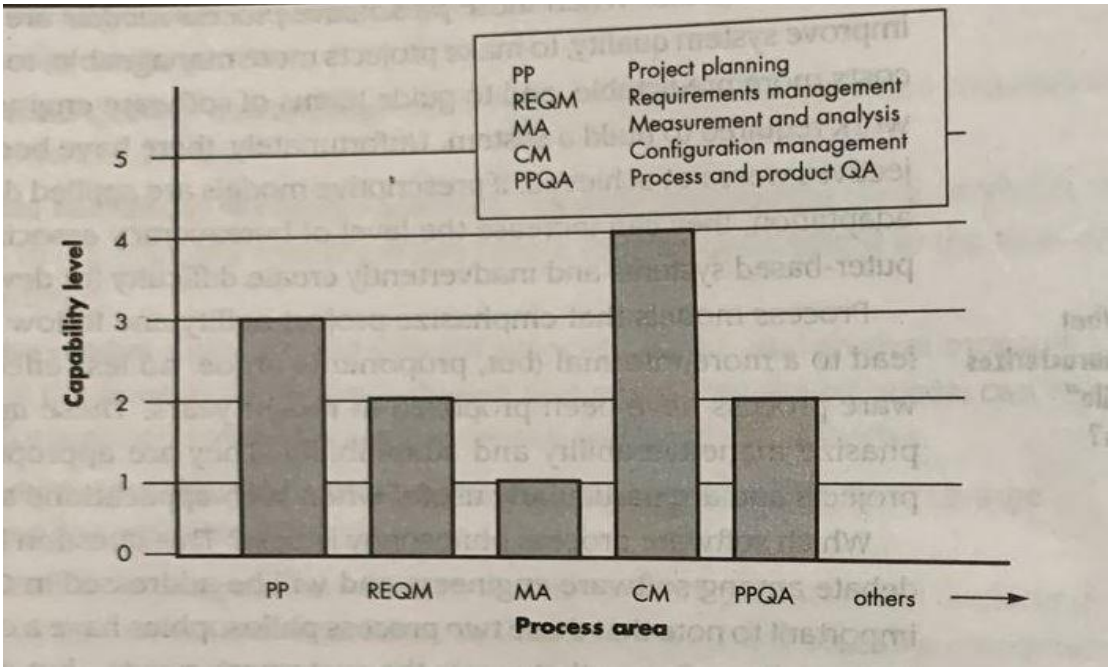
All level-3 criteria have been achieved. In addition, the process area is controlled and improved using measurement and quantitative assessment.

"Quantitative objectives for quality and process performance are established and used as criteria in managing the process".

Level-5 : Optimized

All capability level-4 criteria have been achieved. In addition, the process area is adapted and optimized using quantitative (statistical) means to meet changing customer needs and to continually improve the efficacy of the process area under consideration.

CMMI Process area capability Profile :



The CMMI defines each process area in terms of “specific Goals” and “specific practices” required to achieve these goals.

Specific Goals establish the characteristics that must exist if the activities implied by a process area are to be effective.

Specific Practices refine a goal into a set of process-related activities.

Project Planning is one of eight process areas defined by the CMMI for the ‘project management’ category :

a) SG 1 Establish Estimates

SP 1.1-1 Estimate the scope of the project.
SP1.2-1 Establish estimates of work product and task attributes.
SP 1.3-1 Define project life cycle
SP 1.4-1 Determine estimates of effort and cost.

b) SG 2 Develop a project Plan

SP 2.1-1 Establish the budget and schedule
SP 2.2-1 Identify Project risks
SP 2.3-1 Plan for data management
SP 2.4-1 Plan for Project resources
SP 2.5-1 Plan for needed knowledge and skills
SP 2.6-1 Plan stake holder involvement
SP2.7-1 Establish the Project Plan

c) SG 3 Obtain commitment to the plan.

SP 3.1-1 Review Plans that affect the project.
SP 3.2-1 Reconcile work and resource levels.
SP 3.3-1 Obtain Plan Commitment.

The CMMI also defines set of five generic goals and related practices for each process area.

Each of the five generic goals corresponds to one of the five capability levels.

Hence, to achieve a particular capability level, the generic goal for that level and the generic practices that correspond to that goal must be achieved.

The Generic Goals and Practices for Project Planning Process area :

GG1 Achieve Specific Goals

GP 1.1 perform base practices

GG 2 Institutionalize a managed Process

GP 2.1 Establish an organizational policy

GP 2.2 Plan the process

GP 2.3 Provide Resources

GP 2.4 Assign Responsibility

GP 2.5 Train People

GP 2.6 Manage Configurations

GP 2.7 Identify and involve relevant stake holders

GP 2.8 Monitor and control the process

GP 2.9 Objectively evaluate adherence

GP 2.10 Review status with higher level management

GG 3 Institutionalize a defined process

GP 3.1 Establish a defined process

GP 3.2 Collect improvement information

GG 4 Institutionalize a quantitatively managed process

GP 4.1 Establish quantitative objectives for the process

GP 4.2 Stabilize subprocess performance

GG 5Institutionalize anoptimizing process

GP 5.1 Ensure continuous process improvement.

GP 5.2 Correct root causes of problems

The staged CMMI model defines the same process areas, goals and practices as the continuous model.

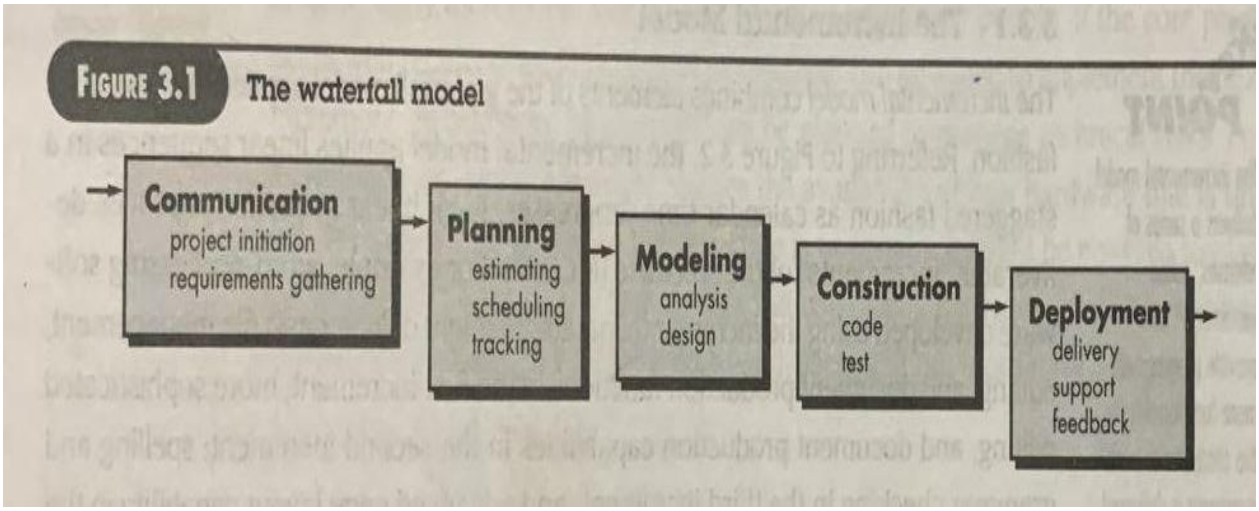
The primary difference is that the staged model defines five maturity levels, rather than five capability levels.

10.The Waterfall Model (WFM) :

The 'Waterfall Model' is also called 'Classic Life Cycle'.

This model suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment.

The waterfall model is the oldest paradigm for software engineering.



11.The Incremental Model (IM) :

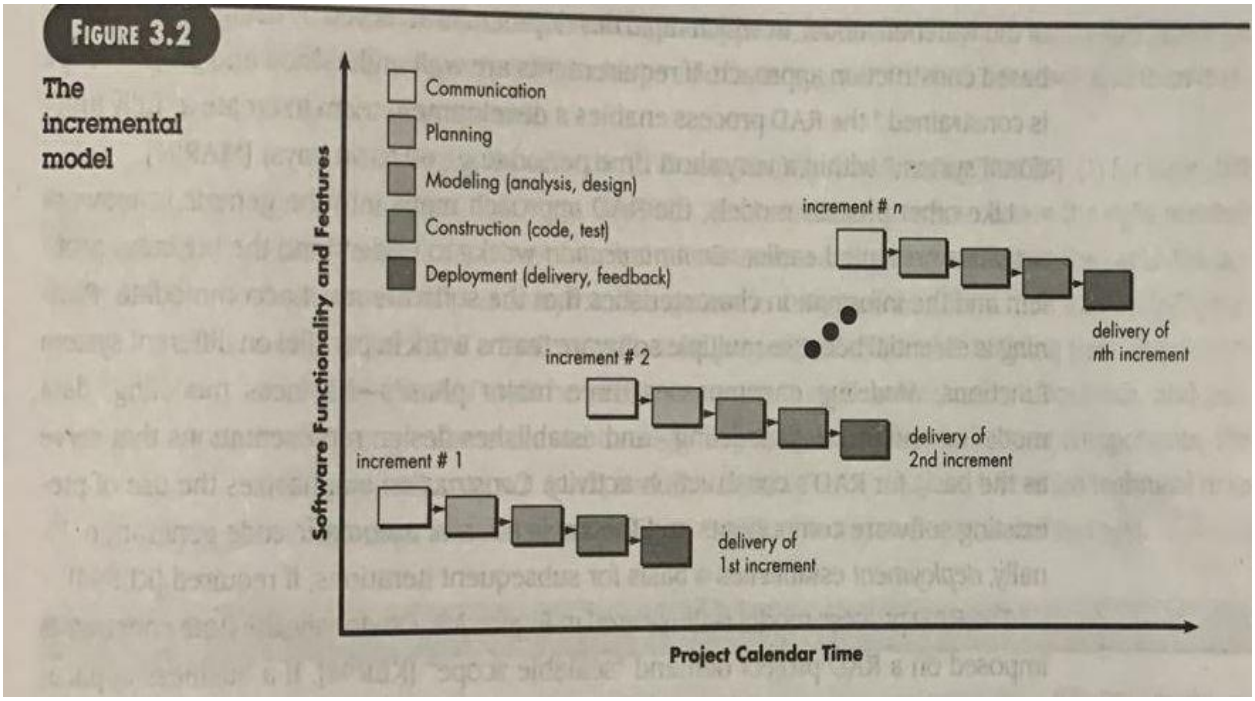
The 'IncrementalModel' combines elements of the waterfall model applied in iterative fashion.

The incremental model applies linear sequences in a staggered fashion as calendar time progresses.

Each linear sequence produces deliverable 'increments' of the software.

For example, word-processing software developed using the incremental paradigm might deliver

- basic file management, editing, and document production functions in the first increment;
- more sophisticated editing, and document production capabilities in the second increment;
- spelling and grammar checking in the third increment;
- advanced page layout capability in the fourth increment.



12.The RAD Model (RADM) :

The 'RADModel' is known as 'Rapid Application Development model'.

This model is an incremental software process model that emphasizes a short development cycle.

The 'RAD model' is a 'high-speed' adaptation of the waterfall model, in which rapid development is achieved by using a component-based construction approach.

If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a 'fully functional system' within a very short time period. Here,

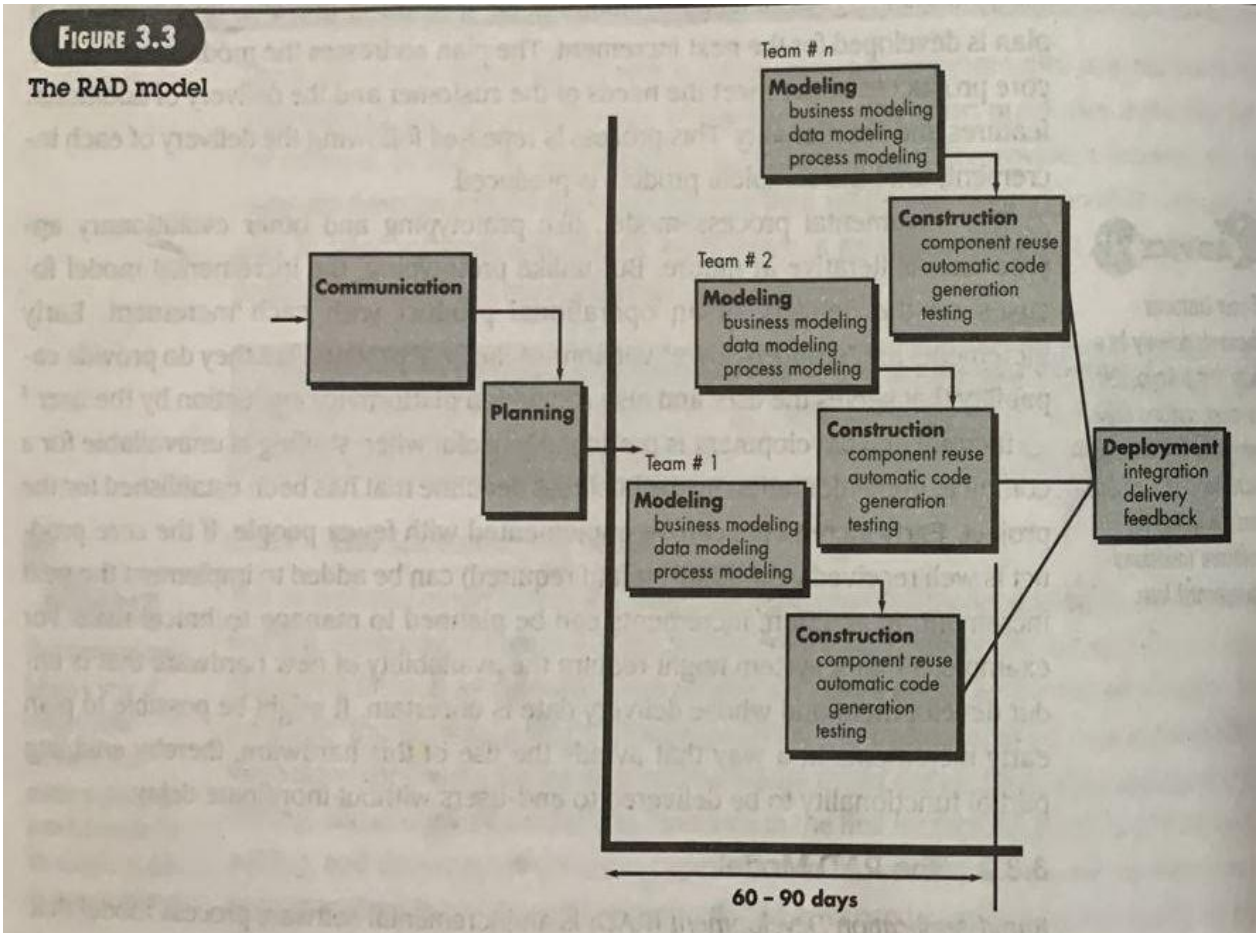
Communication – works to understand the business problem and the information characteristics that the software must accommodate.

Planning – is essential because multiple software teams work in parallel on different system functions.

Modeling – encompasses three major phases business modeling, data modeling and process modeling.

Construction – emphasizes the use of preexisting software components and the application of automatic code generation.

Deployment – establishes a basis for subsequent iterations.



13. Prototyping :

A ‘prototyping’ is an early sample, model or release of a product built o test a concept or process.

‘Paradigm’ is a typical example or pattern of something.

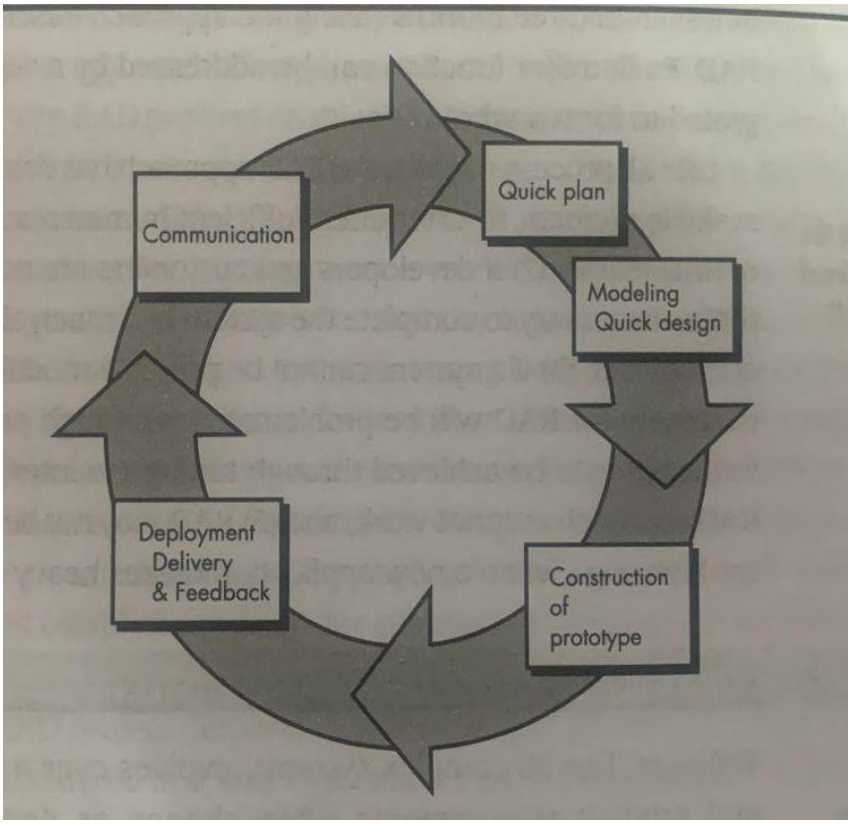
A customer defines a set of general objectives for software, but does not identify detailed input, processing, or output requirements.

In this context, a ‘prototyping paradigm’ may offer the best approach.

Here, prototyping can be used as a standalone process model.

This can be used as a technique that can be implemented within the context of any one of the process models.

The ‘prototyping paradigm’ begins with ‘communication’ and goes through the planning, modeling, construction and ends with ‘deployment’.



14. The Spiral Model :

The 'Spiral Model' is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model.

This provides the potential for rapid development of increasingly more complete versions of the software.

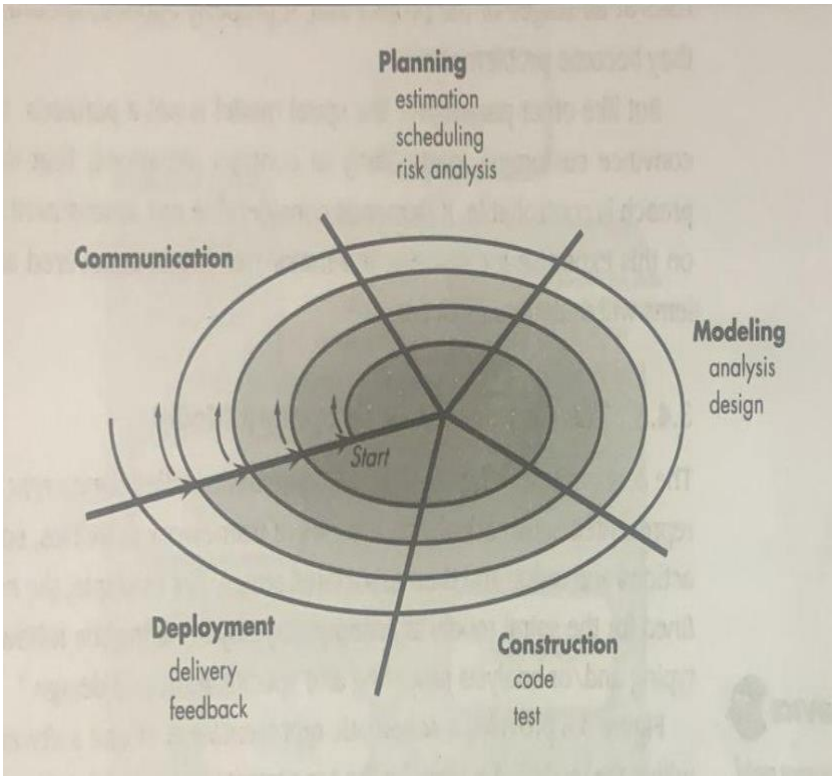
Using the spiral model, software is developed in a series of evolutionary releases.

A spiral model is divided into a set of framework activities defined by the software engineering team.

As this evolutionary process begins, the software team performs activities that are implied by a circuit around the spiral in clockwise direction, beginning at the centre.

Risk is considered at each revolution is made.

This model consider all the phases : communication, planning, modeling, construction and deployment.



15. : The Concurrent Development Model :

The 'Concurrent Development Model' can be represented schematically as a series of framework activities , software engineering actions and tasks, and their associated states.

In the following figure, we have taken the activity – modeling.

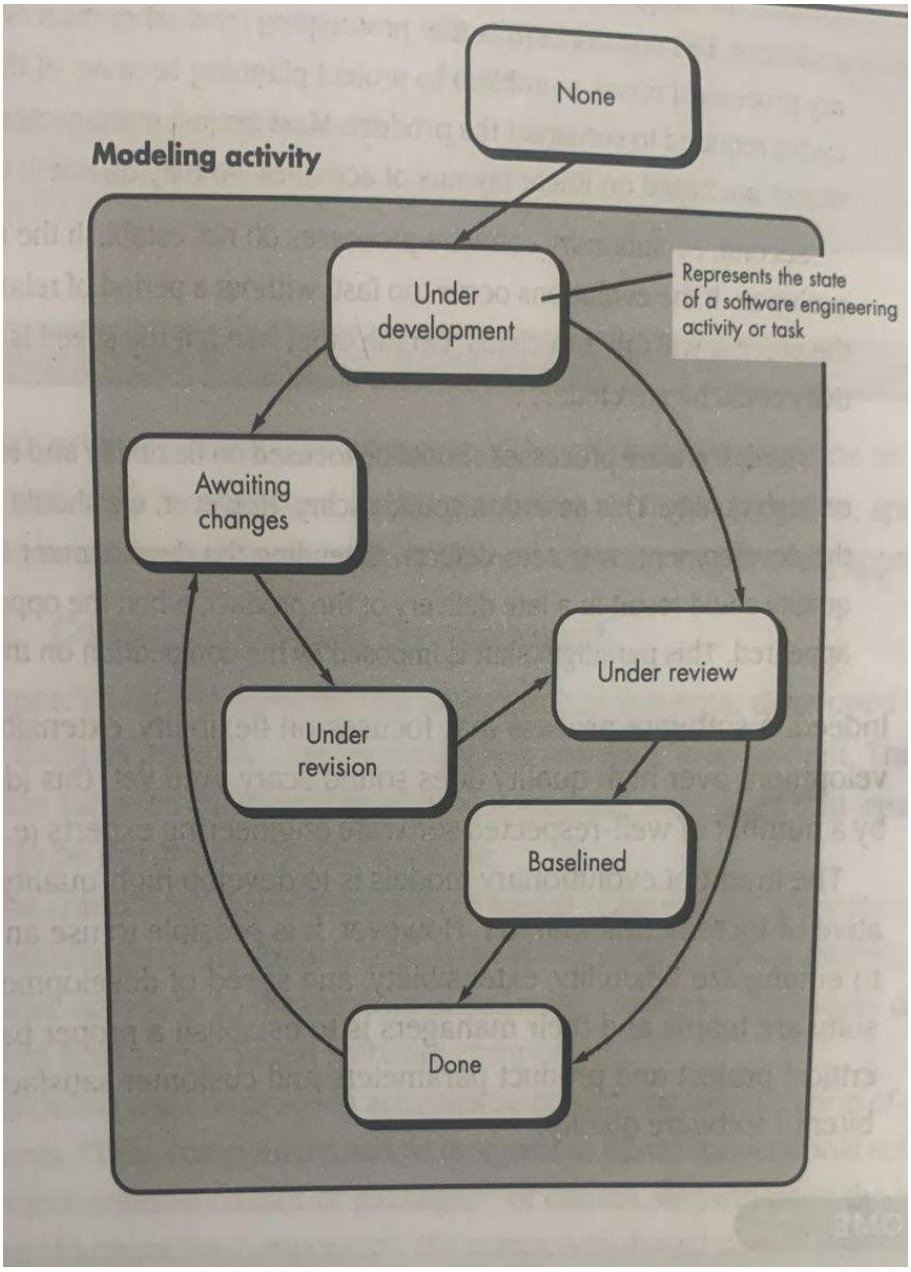
All activities exist concurrently, but resides in different states.

For example, early the 'communication' activity has completed its first iteration and exists in the 'awaiting changes'.

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

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

Similarly other phases also can be drawn.


Modeling activity
None
Under development
Represents the state of a software engineering activity or task
Awaiting changes
Under review
Under revision
Baselined
Done

* * * * *

SOFTWARE ENGINEERING

UNIT-2

AGILE PROCESS MODEL

1. **Introduction :**

Agility :is ability to move quickly and easily.

is also ability to thing and understand quickly.

Def: is a rapid whole-body movement with change of velocity or direction in response to a stimulus.

Agile Software Engineering :

" ASE combines a philosophy and a set of development guidelines."

The philosophy encourages

- customer satisfaction
- early incremental delivery of software.
- small, highly motivated project teams
- informal methods
- minimal software engineering work products
- overall development simplicity.

Agile Team :is a team that is self-organizing and in control of its own destiny.

This fasters communication and collaboration among all who serve on it.

2. **Agile Principles :**

There are 12 principles for those who want to achieve agility.

a) Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

b) Welcome changing requirements, even late in development.

c) Deliver working software frequently from a couple of weeks to a couple of months.

d) Business people and developers must work together daily throughout the project.

e) Build projects around motivated individuals.

f) The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

g) Working software is the primary measure of progress.

h) Agile processes promote sustainable development.

i) Continuous attention to technical excellence and good design enhances agility.

j) Simplicity – that art of maximizing the amount of work not done - is essential.

k) The best architectures, requirements, and designs emerge from self-organizing teams.

l) At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

3. Agile Software Process :

Agile Software Process addresses three key assumptions about the majority of software projects :

a) It is difficult to predict in advance which software requirements will persist and which will change. And also it is equally difficult to predict how customer priorities will change as project proceeds.

b) For many types of software, design and construction are interleaved. That is, both activities should be performed in tandem so that design models are proven as they are created.

c) Analysis, design, construction and testing are not as predictable as we might like.

So, how do we create a process that can manage unpredictability ?

The answer for this is : Process Adaptability.

i.e., The agile process must be adaptable.

4. Human Factors :

Proponents of agile software development take great pains to emphasize the importance of 'human factors'. They are :

a) Competence

b) Common Focus

c) Collaboration

d) Decision making ability

e) Fuzzy Problem-Solving ability

f) Mutual trust and respect

g) Self-Organization.

5. Extreme Programming(XP):

'Extreme Programming' is a software development discipline that organizes people to produce higher-quality software more productively.

In other words :

"Extreme Programming is an agile framework that emphasizes both the broader philosophy of agile –

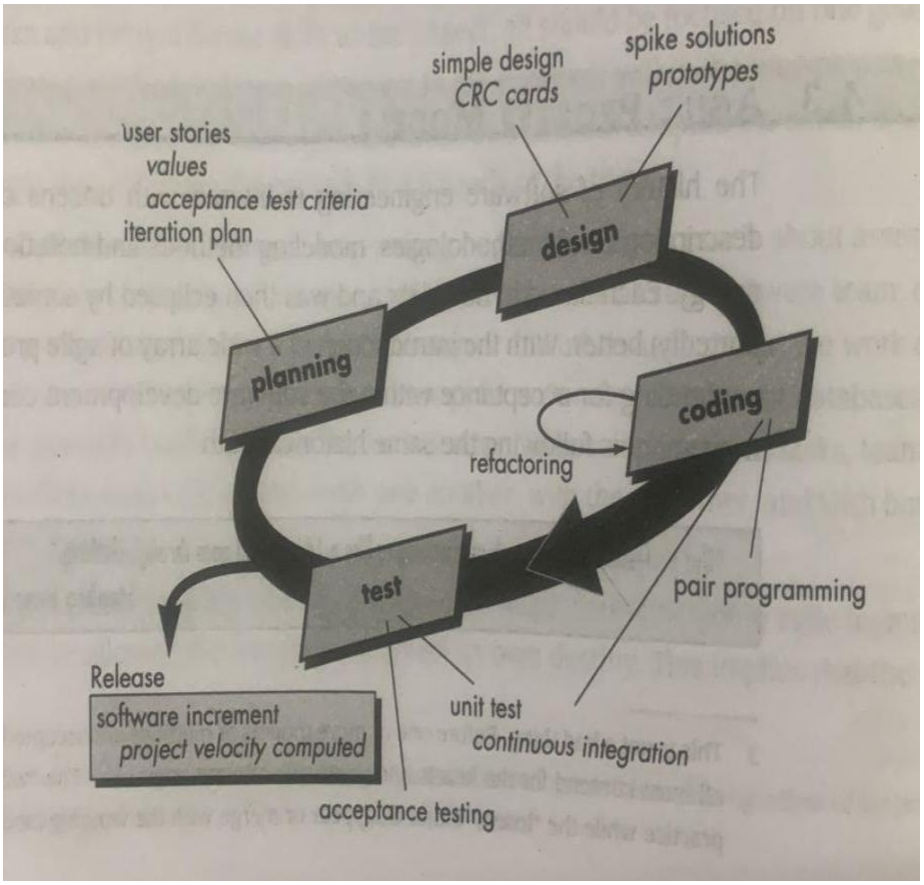
- to produce higher-quality software to please customers
- the more specific goal of making life better for the engineers developing it. "

Extreme Programming involves the client in the process of software development.

XP encompasses a set of rules and practices that occur within the context of four frame work activities : planning, design, coding and testing .

The following figure illustrates the XP process and notes some of the key ideas and tasks that are associated with each framework activity.

The Extreme Programming Process :



Planning : The planning activity begins with the creation of a set of stories that describe required features and functionality for software to be built.

Each story is written by the customer and is placed on an index card. The customer assigns a value based on the overall business value of the feature of function.

Members of the XP team then assess each story and assign a cost – measured in development weeks - to it. If the story will require more than three development weeks, the customer is asked to split the story into smaller stories, and the assignment of value and cost occurs again.

Design : XP encourages the principle 'keep it simple' (KIS). A simple design is always preferred over a more complex representation. In addition, the design provides implementation guidance for a story as it is written – nothing less, nothing more.

XP encourages the use of CRC cards. (class-responsibility collaborator) as an effective mechanism for thinking about the software in an object-oriented context.

If a difficult design problem is encountered as part of the design of a story, XP recommends the immediate creation of an operational prototype of that portion of the design, called '**spike solution'** – the design prototype is implemented and evaluated .

XP encourages 'refactoring' - a construction technique that is also a design technique.

"**Refactoring**" is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves the internal structure.

Code :XP recommend that after stories are developed and preliminary design work is done, the team should not move to code, but rather develop a series of unit tests that will exercise each of the stories that is to be included in the current release.

Once the unit test has been created, the developer is better able to focus on what must be implemented to pass the unit test.

A key concept during the coding activity is '**pair programming**'.

XP recommends that two people work together at one computer work station to create code for a story. This provides a mechanism for real time problem solving (two heads are often better than one) and real time quality assurance.

Testing : The unit tests that are created should be implemented using a framework that enables them to be automated.

As the individual unit tests are organized into a 'universal testing suite', integration and validation testing of the system can occur on a daily basis.

XP acceptance, also called 'customer tests', are specified by the customer and focus on overall system features and functionality that are visible and reviewable by the customer.

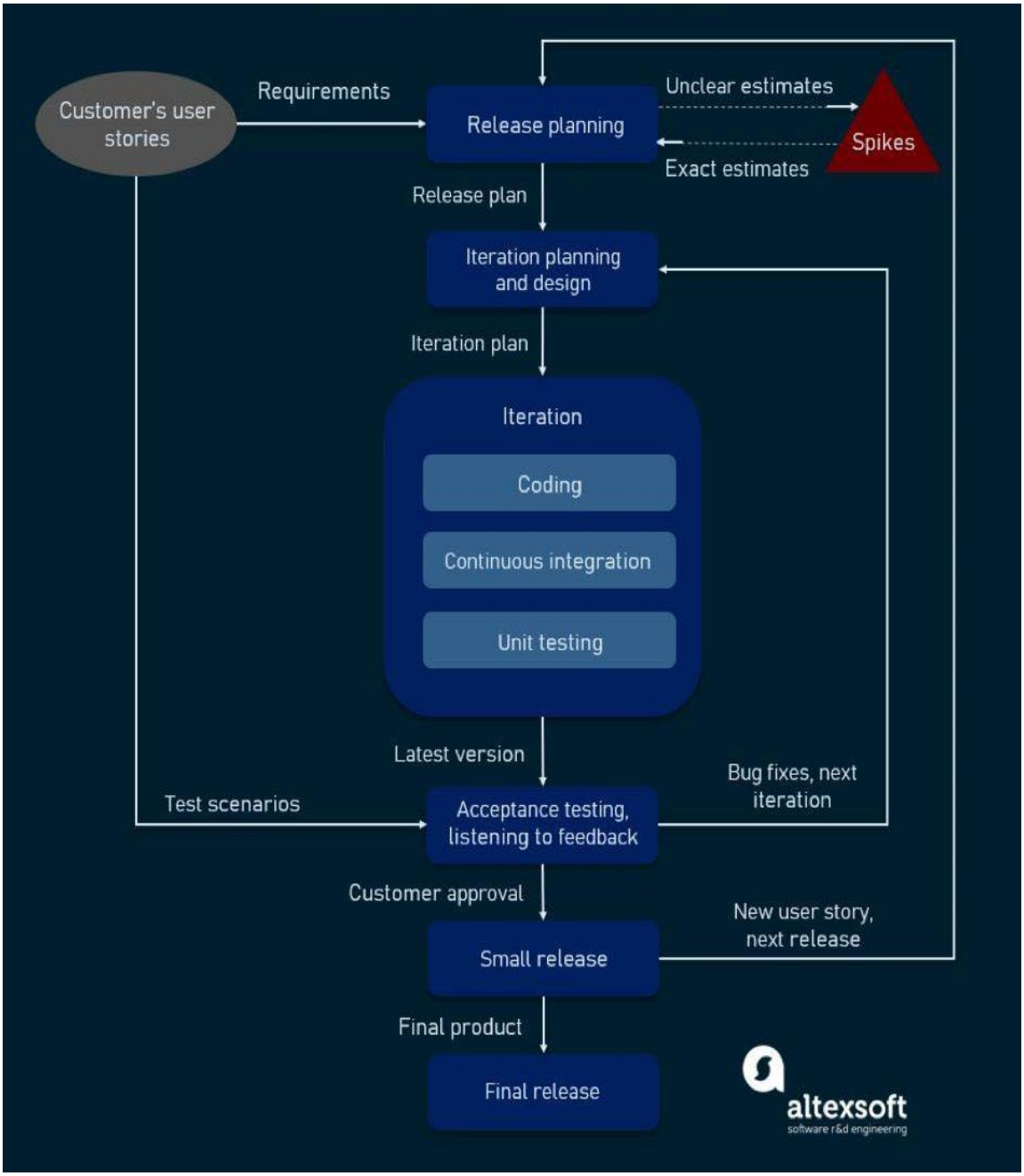
"Fixing small problems every few hours takes less time than fixing huge problems just before the deadline".

Extreme Programming Life Cycle :

'Extreme Programming Life Cycle' is the journey that takes us from the planning stage to the delivery of the final product.

As an Agile methodology, XP works on the principle of iterative cycles, so the life cycle iterates several times until the product is ready.

XPLC is the time required to perform a typical XP project.



6. Dynamic Systems Development Method (DSMD) :

The 'Dynamic Systems Development Method' is an agile software development approach that "provides a framework for building and maintaining systems which meet tight time constraints through the use of incremental prototyping in a controlled project environment".

In other words, DSDM is an associate degree agile code development approach that provides a framework for building and maintaining systems.

Like XP, DSDM suggests an iterative software process.

The DSDM approach to each iteration follows the 80 % rule.

That is, only enough work is required for each increment to facilitate movement to the next increment.

The remaining detail can be completed later when more business requirements are known or changes have been requested and accommodated.

The DSDM life cycle defines three different iterative cycles, preceded by two additional life cycle activities :

a) **Feasibility Study** : establishes the basic business requirements and constraints associated with the application to be built and then assesses whether the application is a viable candidate for the DSDM process.

b) **Business Study** : establishes the functional and information requirements that will allow the application to provide business value; also, defines the basic application architecture and identifies the maintainability requirements for the application.

c) **Functional Model Iteration** : produces a set of incremental prototype that demonstrate functionality for the customer. The intent during this iterative cycle is to gather additional requirements by eliciting feedback from users as they exercise the prototype.

d) **Design and build iteration** : revisits prototypes built during the functional model iteration to ensure that each has been engineered in

a manner that will enable it to provide operational business value for end-users.

e) **Implementation** : places the latest software increment into the operational environment.

It should be noted that :

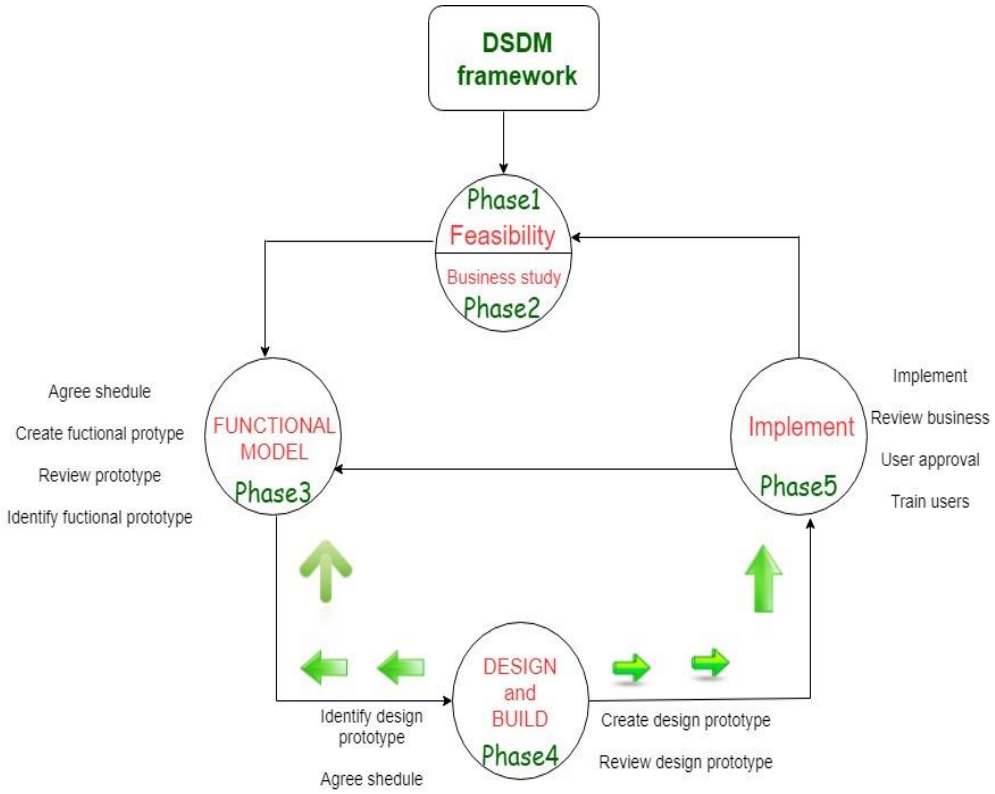
i) the increment may not be 100 % complete or

ii) changes may be requested as the increment is put into place.

In either case, DSDM development work continues by returning to the function model iteration activity.

DSDM is often combined with XP to supply a mixed approach that defines a solid method model, known as DSDM life cycle.

DSDM Life Cycle :



Dynamic Systems Development Method life cycle

7. Feature Driven Development (FDD) :

The 'Feature Driven Development' is a practical process model for object-oriented software engineering.

'Feature Driven Development' is an iterative and incremental software development process.

This is a light weight or Agile method for developing software.

FDD is a customer-centric software development methodology known for short iterations and frequent releases.

In the context of FDD, a feature "is a client valued function that can be implemented in two weeks or less".

The benefits of features :

a) Because features are small blocks of deliverable functionality, users can describe them more easily, understand how they related to one another more readily.

b) Features can be organized into a hierarchical business-related grouping.

c) Since a feature is the FDD deliverable software increment, the team develops operational features every two weeks.

d) Because features are small, their design and code repre-sentations are easier to inspect effectively.

e) Project planning, scheduling, and tracking are driven by the feature hierarchy.

The template for defining a feature :

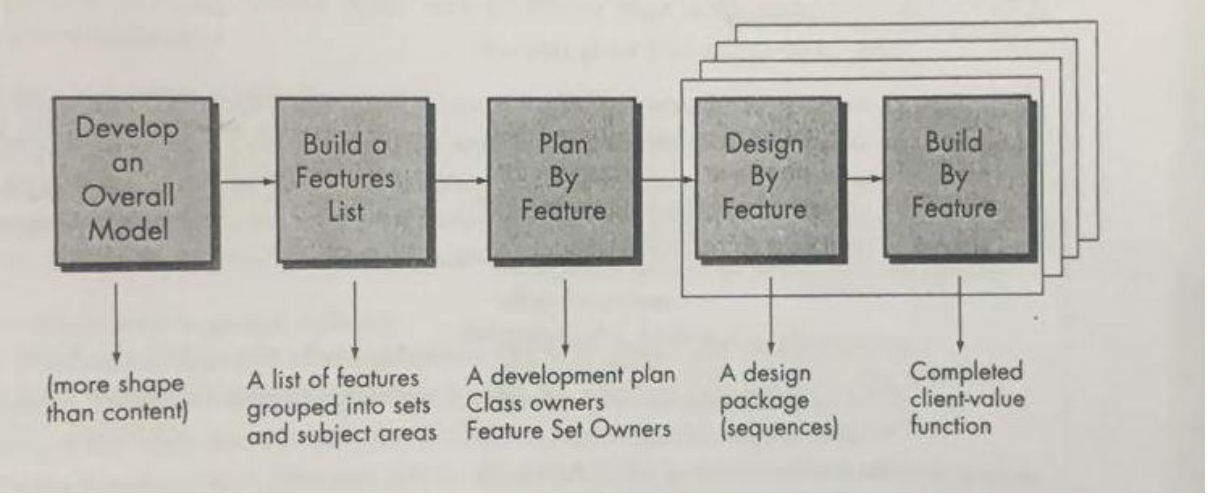
<action> the <result>< by | for | of | to> a <object>

Example : a) Add the product to a shopping cart.

b) Display the technical specifications of a product.

c) Store the shipping-information for a customer.

FDD Framework :



The above figure defines five 'collaborating' framework activities (in FDD they are called 'processes').

It is essential for developers, their managers, and the customer to understand project status – what accomplishments have been made and problems have been encountered.

If deadline pressure is significant, it is critical to determine, if software increments are properly scheduled.

To accomplish this, FDD defines six milestones during the design and implementation of a feature :

design walkthrough	design
design inspection	code
code inspection	promote to build

Advantages of FDD :

- Gives the team a very good understanding of the project's scope and context.
- Requires few meetings.
- Uses a use-centric approach.
- Works well with large-scale, long-term, or ongoing projects.

8. SCRUM :

'Scrum' is an agile process model.

'Scrum' is a framework for product management commonly used in software development.

Def: 'Scrum' is a management framework that teams use to self-organize and work towards a common goal.

This describes a set of meetings, tools, and roles for efficient project delivery .

Principles of Scrum :

a) Small working teams are organized to 'maximize communication, minimize overhead, and maximize sharing of tacit, informal knowledge'.

b) The process must be adaptable to both technical and business changes 'to ensure the best possible product is produced'.

c) The process yields frequent software increments 'that can be inspected, adjusted, tested, documented and built on'.

d) Development work and the people who perform it are partitioned 'into clean, low coupling partiiones, or packets'.

e) Constant testing and documentation is performed as the product is built.

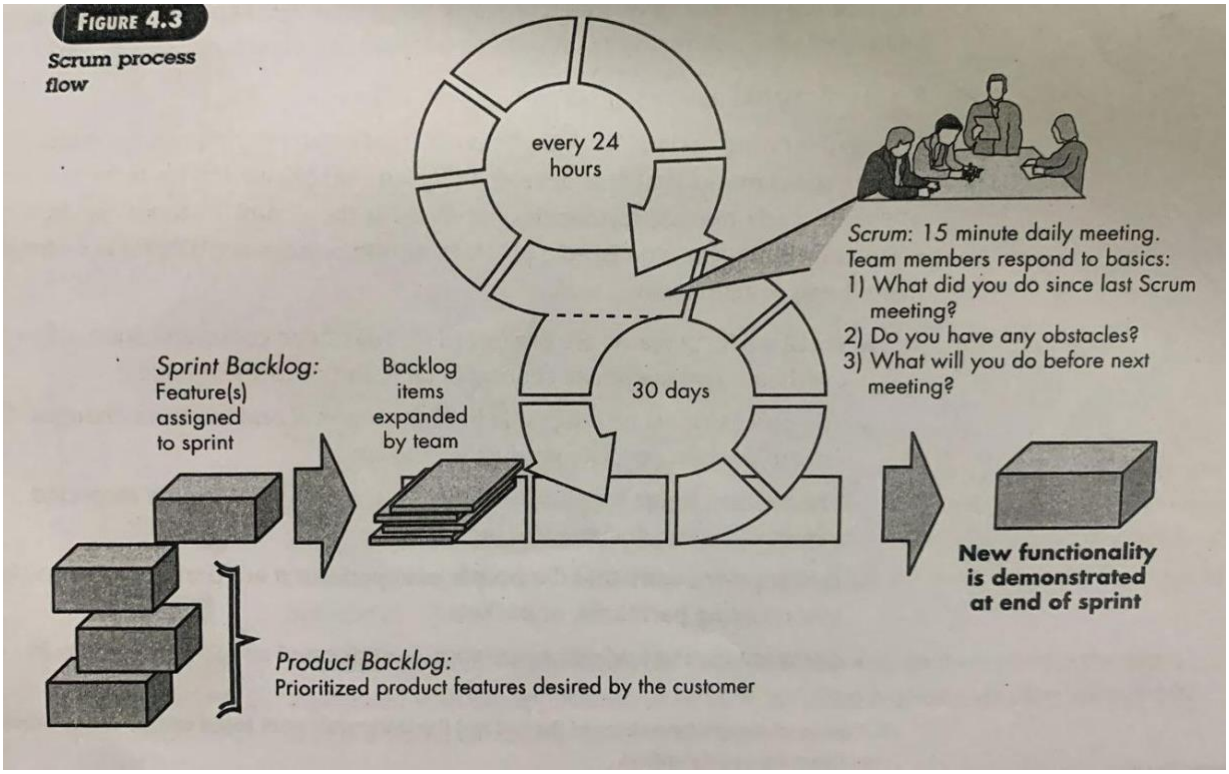
f) The scrum process provides the 'ability to declare a product "done" whenever required.

Scrum principles are used to guide development activities withing a process that incorporates the framework activities :

requirements analysis design evolution delivery

Within each framework activity, work tasks occur within a process pattern, called a 'sprint'.

The overall flow of scrum process is illustrated in the following figure :



Sprints : consist of work units that are required to achieve requirement defined in the backlog that must be fit into a predefined time-box (typically 30 days).

The sprint allows team members to work in a short-term, but stable environment.

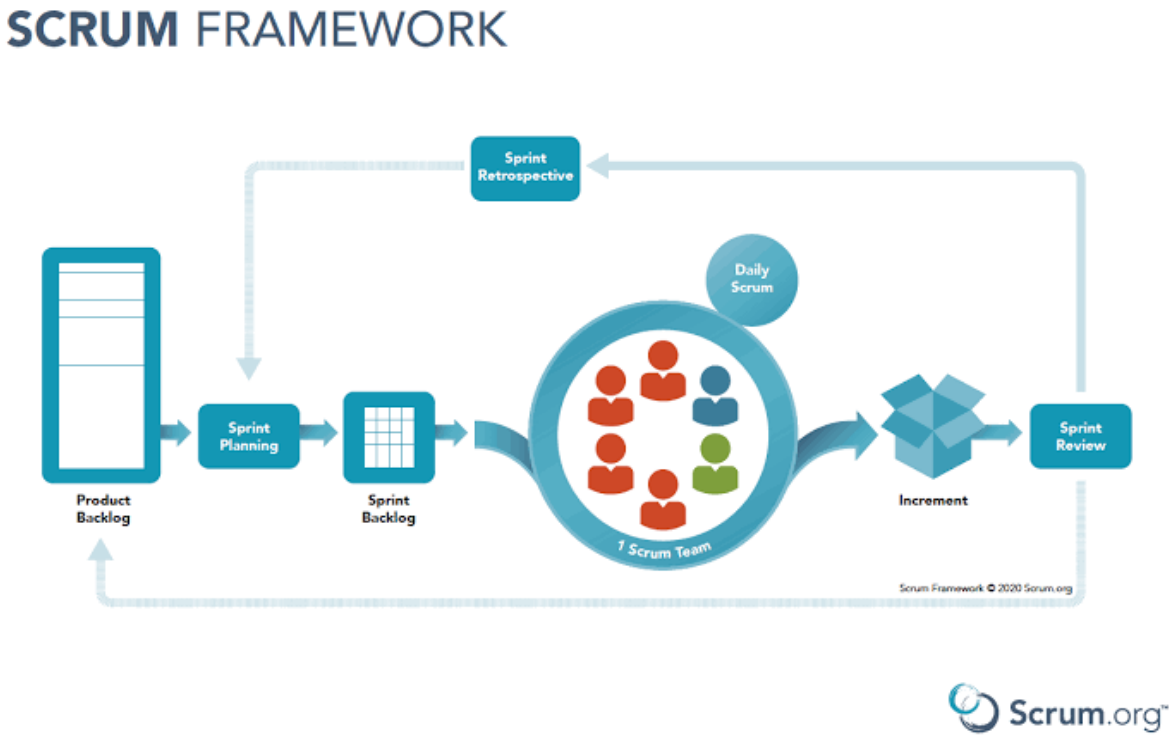
Scrum Meetings : are short meetings held daily by the scrum team.

Three key questions are asked and answered by all the team members.

- What did you do since the last team meeting.
- What obstacles are you encountering.
- What do you plan to accomplish by the next team meeting.

A team is called a '**scrum master**', leads the meeting and assesses the responses from each person.

Demos : deliver the software increment to the customer so that functionality that has been implemented can be demonstrated and evaluated by the customer



Roles of Scrum Master :

There are eight roles of Scrum Master.

a) Servant-Leader
b) Faculitator
c) Coach
d) Conflict Navigator
e) Manager
f) Mentor
g) Teacher
h) Change Agent

The Scrum Master teaches scrum team to keep the daily scrum within the 15-minute time-box.

The daily scrum is an internal meeting for the scrum team.

The scrum master is responsible for identifying and removing any impediments that may be preventing the team from delivering value.

Scrum master also implement changes and steps to increase the team's productivity.

9. SOFTWARE REQUIREMENTS :

The requirements for a system are the descriptions of the services provided by the system and its operational constraints.

These requirements reflect the needs of customers for a system that helps solve some problem such as controlling a device, placing an order or finding information.

The process of finding out, analyzing, documenting and checking these services and constraints is called '**Requirements Engineering**' (RE).

There are two types of requirements :

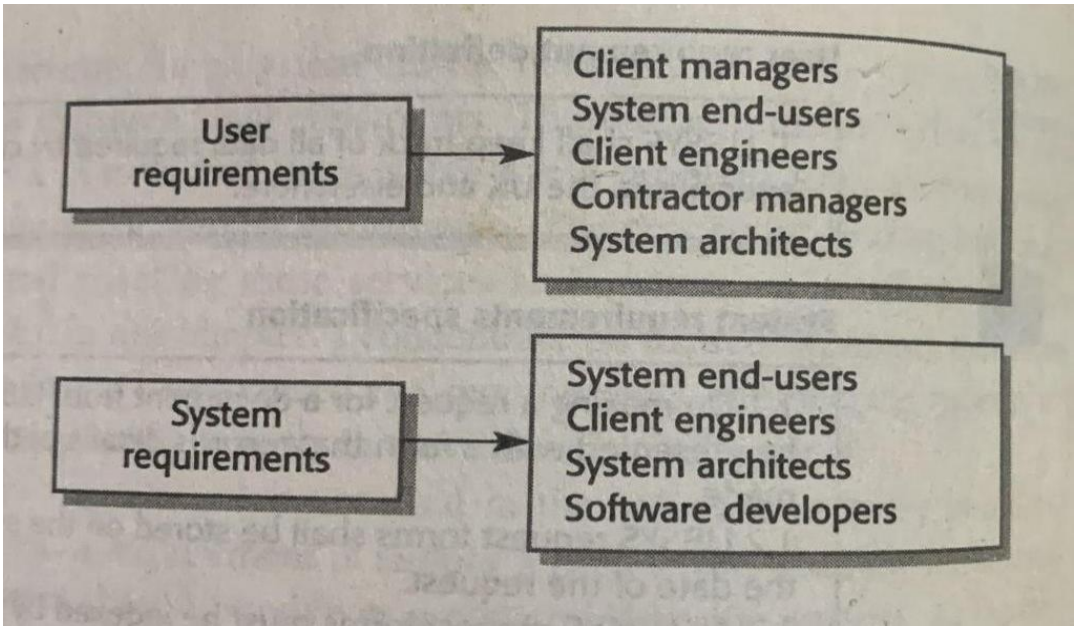
a) **User Requirements** : to mean high level abstract requirements.

   User requirements are statements, in a natural language plus diagrams, of what service the system is expected to provide and the constraints under which is must operate.

b) **System Requirements** : to mean the detailed description of what the system should do.

   System requirements set out the system's functions, services and operational constraints in detail.

Readers of different types of specification :



10.FUNCTIONAL REQUIREMENTS :

The Functional Requirements for a system describe **what the system should do.**

In other words, these statements are statements of

- what services the system should provide,
- how the system should react to particular inputs
- how the system should behave in particular situations.

These requirements depend on

- the type of software being developed,
- the expected users of the software
- the general approach taken by the organization when writing requirements.

Example : University Library System **(LIBSYS)**, used by students and faculty to order books and document from other libraries.

- The user shall be able to search either all of the initial set of databases or select a subset from it.
- The system shall provide appropriate viewers for the user to read documents in the document store.
- Every order shall be allocated a unique identifier (Order-Id), which the user shall be able to copy to the account's permanent storage area.

The above are the '**Functional User Requirements**', define specific facilities to be provided by the system.

Functional Requirements Specification of a system should be both complete and consistent :

'**Completeness**' means that all services required by the user should be defined.

'**Consistency**' means that requirement should not have contradictory definitions.

It is practically impossible to achieve requirements consistency and completeness.

11.Non-FUNCTIONAL REQUIREMENTS :

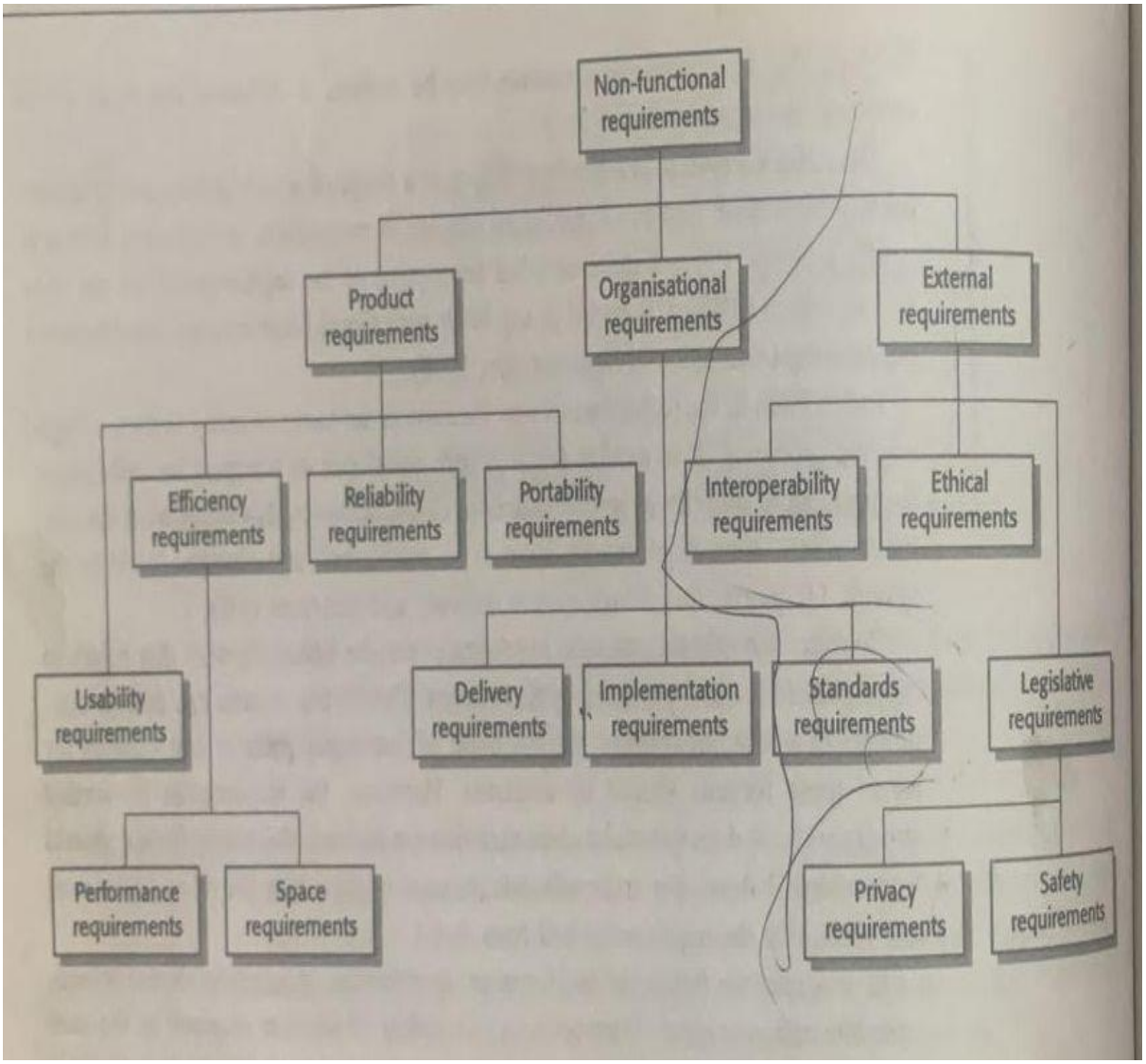
'Non-functional requirements' are requirement that are not directly concerned with the specific functions delivered by the system.

These may related to emergent system properties such are

- reliabilityresponse time store occupancy.

These are rarely associated with individual system features.

Types of Non-Functional Requirements :



Majorly, the types of non-functional requirements are :

Product Requirements : These requirements specify product behavior.

Organizational Requirements : These requirements are derived from policies and procedures in the customer's and developer's organization.

External Requirements : This broad heading covers all requirements that are derived from factors external to the system and its development process.

Metrics for specifying non-functional requirements :

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	K bytes Number of RAM 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

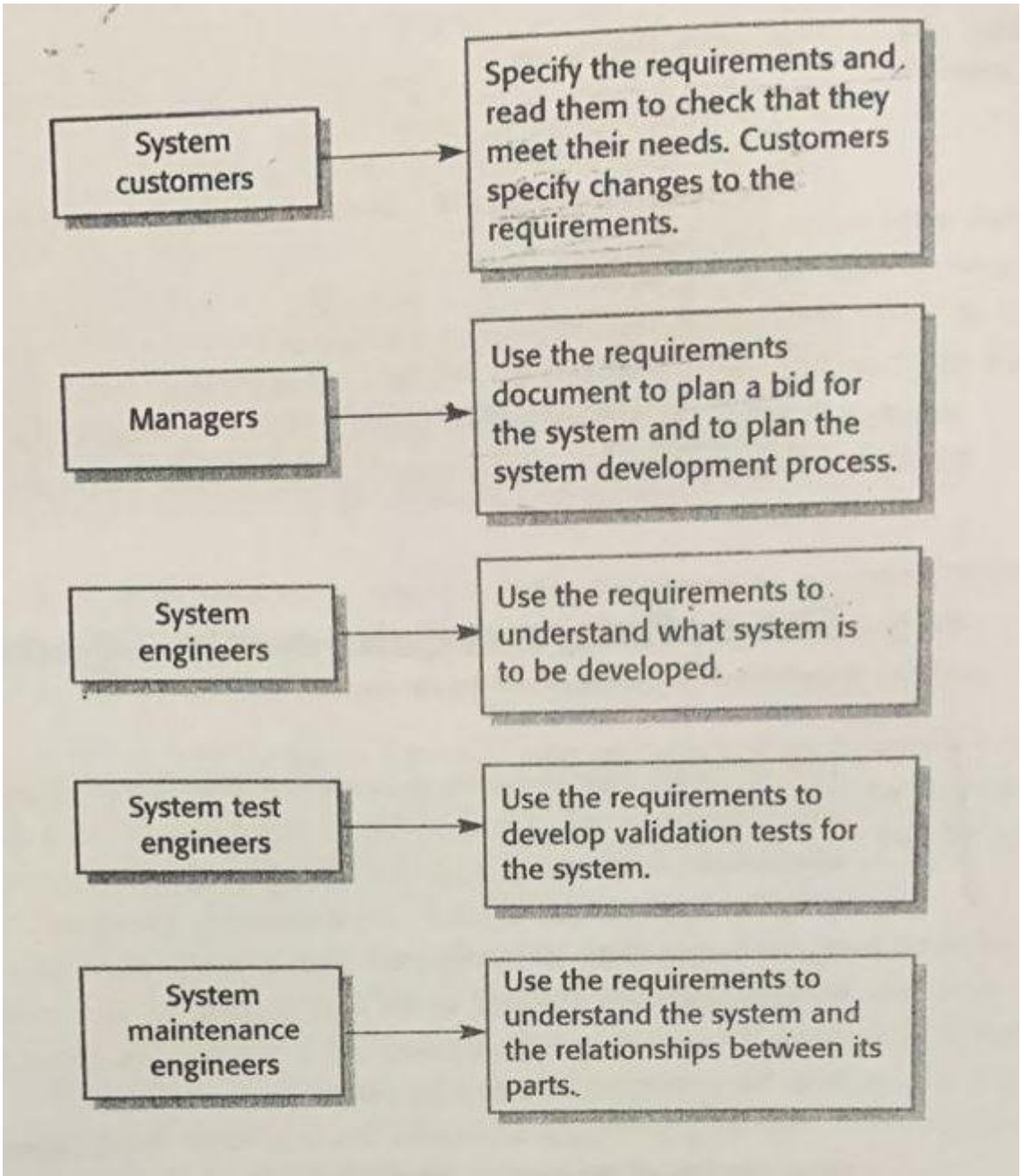
12. SOFTWARE REQUIREMENT DOCUMENT (SRD) :

This is also known as 'Software Requirement Specification (SRS)'.

This is the official statement of what the system developers should implement.

This should include both the user requirements for a system and a detailed specification of the system requirements.

Users of a Requirement Document :



The requirements document has a diver 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.

The diversity of possible users means that the requirements document has to be a compromise between communicating the requirements to customers, defining the requirements in precise detail for developers and testers, and including information about possible system evaluation.

The **IEEE standard** suggest the following structure for requirements document :

a) Introduction :

- Purpose of the requirements document.
- Scope of the Product
- Definitions, Acronyms and abbreviations
- References
- Overview of the remainder of the document.

b) General Description :

- Product perspective
- Product functions
- User Characteristics
- General Constraints
- Assumptions and dependencies

c) Specific Requirements :

- Functional, non-functional and interface requirements
- Document external interfaces
- System functionality and performance
- Logical database requirements
- Design constraints
- Emergent System Properties
- Quality Characteristics

d) Appendices

e) Index

A 'software requirements document' (also known as SRS) is a document or set of documentation that outlines the features and intended behavior of a software application.

The RE process is :

- Feasibility Study
- Requirement Gathering
- Software Requirement Specification
- Software Requirement Validation

The complete SRS are :

- Clear
- Correct
- Consistent
- Coherent
- Comprehensible
- Modifiable
- Verifiable
- Prioritized

The Structure of Requirements Document :

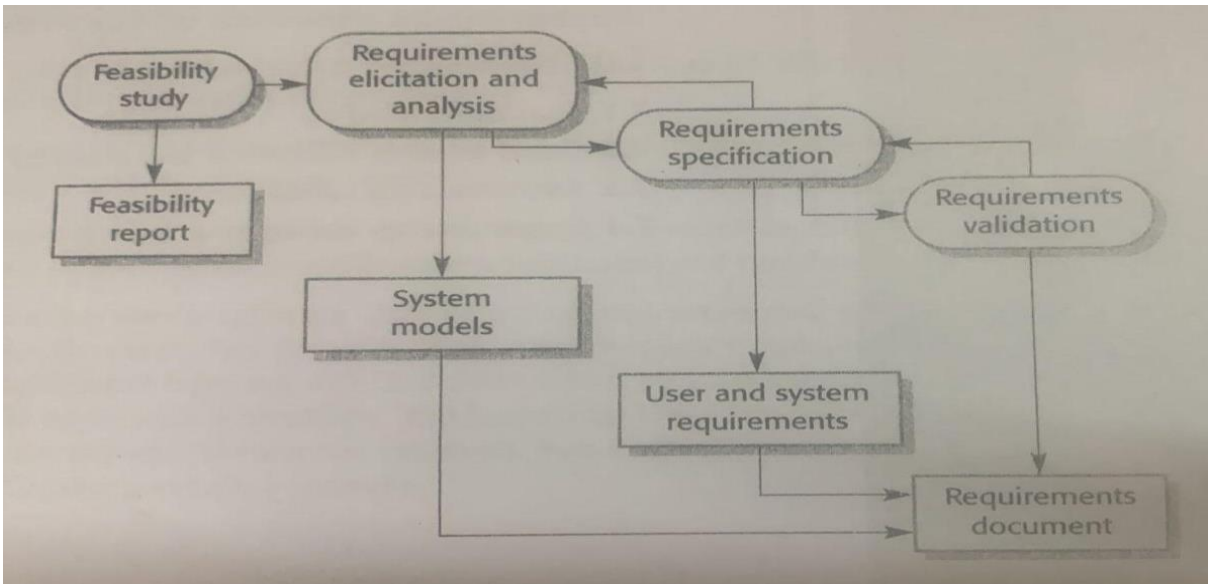
Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe its functions and explain how it will work with other systems. It should describe how the system fits into the overall business or strategic objectives of the organisation commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	The services provided for the user and the non-functional system requirements should be described in this section. This description may use natural language, diagrams or other notations that are understandable by customers. Product and process standards which must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.

System requirements specification	This should describe the functional and non-functional requirements in more detail. If necessary, further detail may also be added to the non-functional requirements, e.g. interfaces to other systems may be defined.
System models	This should set out one or more system models showing the relationships between the system components and the system and its environment. These might be object models, data-flow models and semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based and anticipated changes due to hardware evolution, changing user needs, etc.
Appendices	These should provide detailed, specific information which is related to the application which is being developed. Examples of appendices that may be included are hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organisation of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, etc.

13. **Requirement Engineering Process (REP) :**

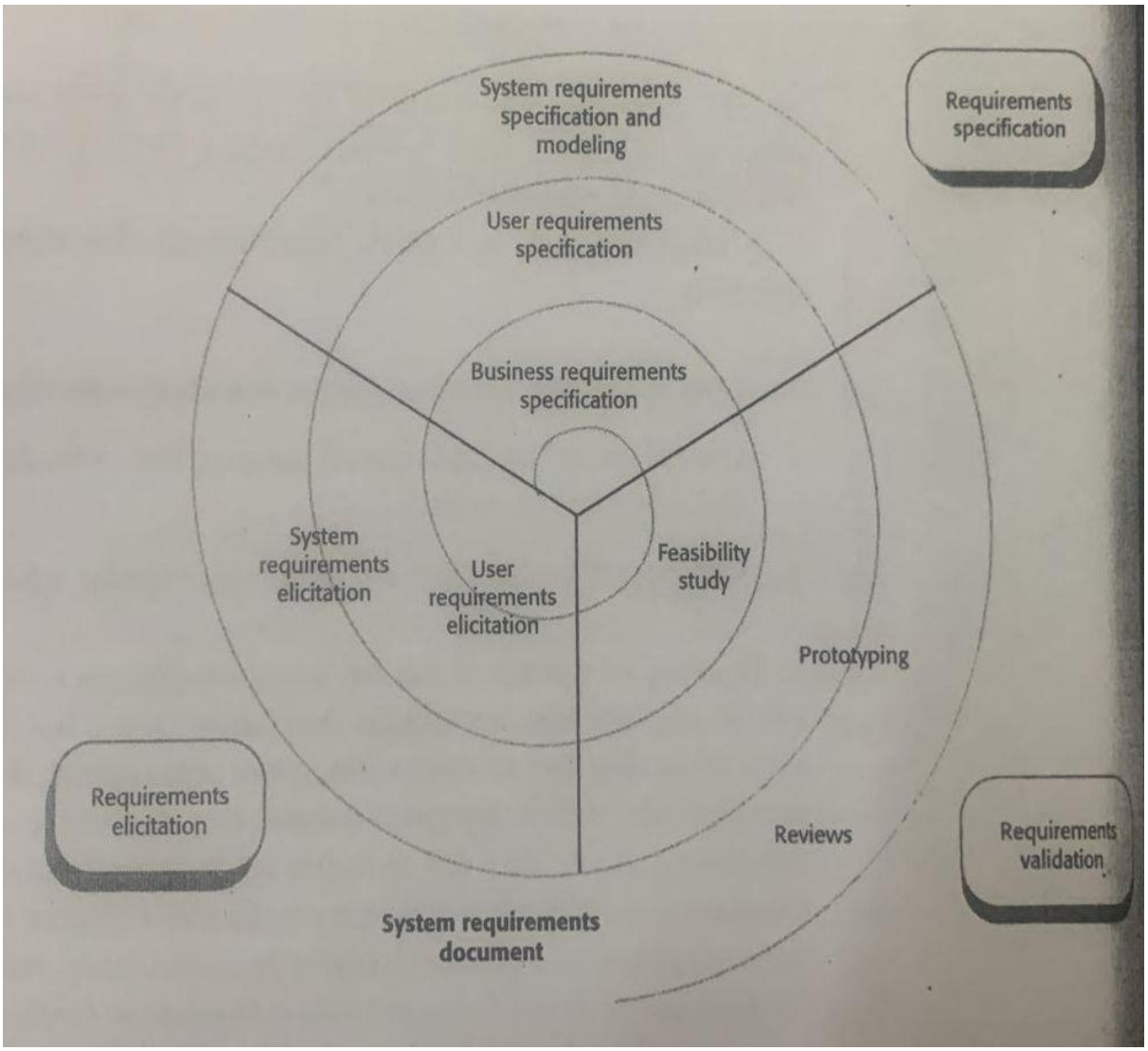
'Requirement Engineering' (RE) refers to the process of defining, documenting, and maintaining requirements in the engineering design process.

Requirement Engineering Process :



The goal of requirement engineering process is create and maintain a system requirements document.

Spiral Model of Requirements Engineering Process :



14. **Feasibility Study :**

'Feasibility Study' is a preliminary exploration of a proposed project or undertaking to determine its merits and viability.

A feasibility study aims to provide an independent assessment that examines all aspects of a proposed project, including technical, economic, financial, legal, and environmental considerations.

The input to feasibility study is a set of preliminary requirements, an outline description of the system and how the system is intended to support processes.

The result of the feasibility study should be report that recommends whether or not it is worth carrying on with the requirements engineering and systems development process.

A feasibility study is a short, focused study that aims to answer a number of questions like :

a) Does the system contribute to the overall objectives of the organization.

b) Can the system be implemented using current technology and within given cost and schedule constraints.

c) Can the system be integrated with other systems which are already in place.

15. Requirements Elicitation and Analysis :

Here, software engineers work with customers and system end-users to find out about the application domain, what services the system should provide, the required performance of the system, hardware constraints and so on.

The term 'stakeholder' is used to refer to any person or group who will be affected by the system, directly or indirectly.

The process activities are FOUR.

a) **Requirements Classification and Organization** : This activity takes the unstructured collection of requirements, groups related equirements and organizes them into coherent clusters.

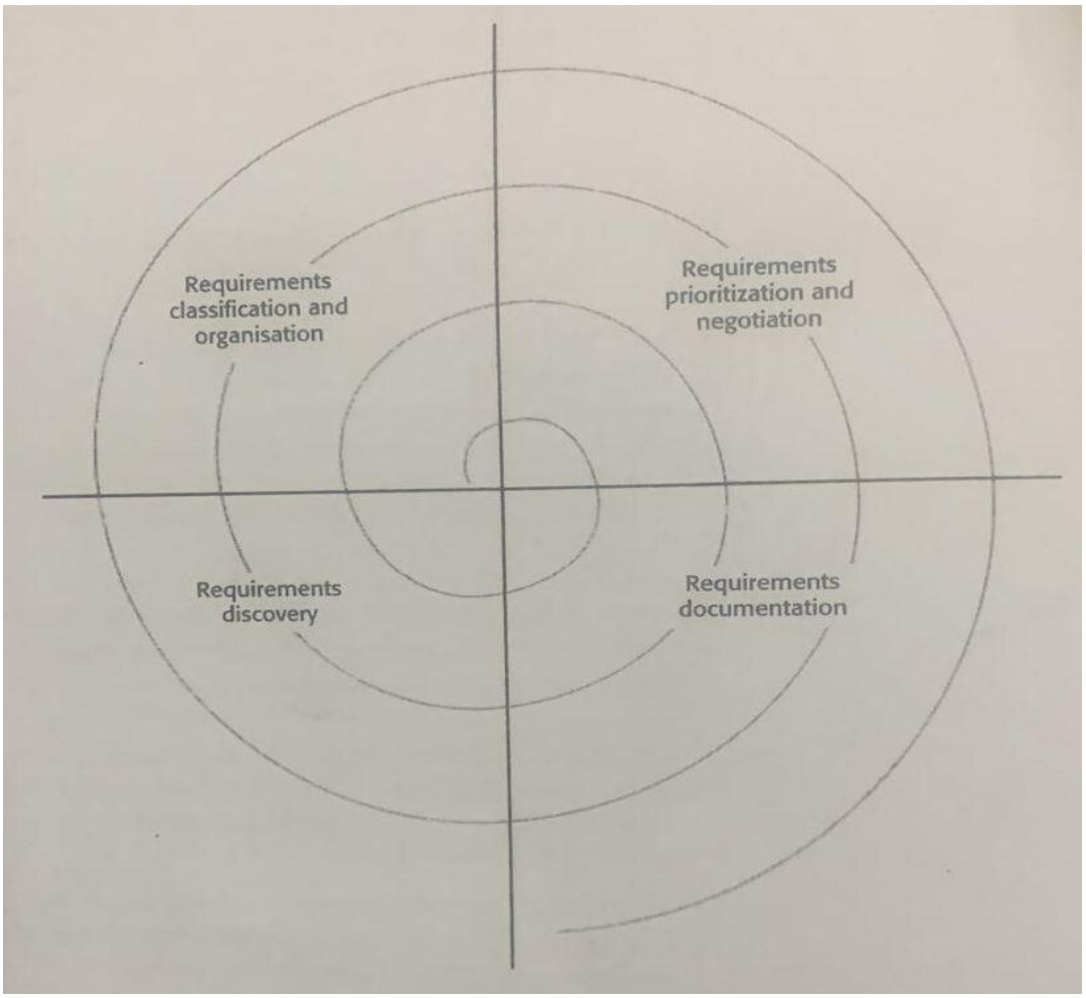
b) **Requirements Prioritization and negotiation** : Inevitably, where multiple stakeholders are involved, requirements will conflict. This activity is concerned with prioritizing requirements, and finding and resolving requirements conflict through negotiation.

c) **Requirements Documentation** : The requirements are documented and input in the next round of the spiral. Formal and informal requirements documents may be produced.

d) **Requirements Discovery** : This is the process of interacting with stakeholders in the system to collect their requirements. Domain requirements from stakeholders and documentation are also discovered during this activity.

The requirement elicitation and analysis process :

****



Ethnography : is an observational technique that can be used to understand social and organizational requirements.

Ethnography is particularly effective at discovering two types of requirements :

a) Requirements that are derived from the way in which people actually work rather than the way in which process definitions say they ought to work.

b) Requirements that are derived from cooperation and awareness of other people's activities.

16. Requirements Validation :

'Requirements Validation' is the process of confirming that the written requirements agree with the stakeholders' requests.

In other words, verification is about checking whether the requirements are complete, correct and consistent.

During the 'requirements validation process', checks should be carried out on the requirements document.

These checks are :

a) **Validity Checks :** Each requirement should be checked for its validity.

b) **Consistency Checks** : Requirements in the document should not conflict.

c) **Completeness checks** : The requirements document should include requirement, which define all document, and constraints intended by the system user.

d) **Realism Checks** : Using knowledge of existing technology, the requirements should be checked to ensure that they could actually be implemented.

e) **Verifiability** : To reduce the potential for dispute between customer and contractor, system requirements should always be written so that they are verifiable.

Validation Techniques can be used in conjunction or individually :

a) Requirement Reviews : The requirements are analyzed systematically by a team of viewers.

b) Prototyping : In this approach to validation, an executable model of the system is demonstrated to end-users and customers.

c) Test-case Generation : Requirements should be testable. If the tests for the requirements are devised as part of the validation process, this often reveals requirements problems.

17. Requirements Reviews :

'Requirements Review' is manual process that involves people from both client and customer.

They check the requirements document for anomalies and omissions.

The review team should check each requirement for consistency as well as check the requirements as a whole for completeness.

Reviewers may also check for :

a) **Verifiability** : Is the requirement as stated realistically testable.

b) **Comprehensibility** : Do the procurers or end-users of the system properly understand the requirement.

c) **Traceability** : Is the one which allows the impact of change on the rest of the system to be assessed.

d) **Adaptability** : Is the requirement adaptable ? That is, can the requirement be changed without large-scale effects on other system requirements.

18. Requirements Management :

'Requirements Management' is the process of understanding and controlling changes to system requirement.

The requirements for large software systems are always changing.

During the software process, the stakeholders' understanding of the problem is constantly changing.

Once a system is installed, new requirement inevitably emerge.

It is hard for users and system customers to anticipate what effects the new system will have on the organization.

Once end-users have experience of a system they discover new needs and priorities.

One should keep track of individual requirements and maintain links between dependent requirements.

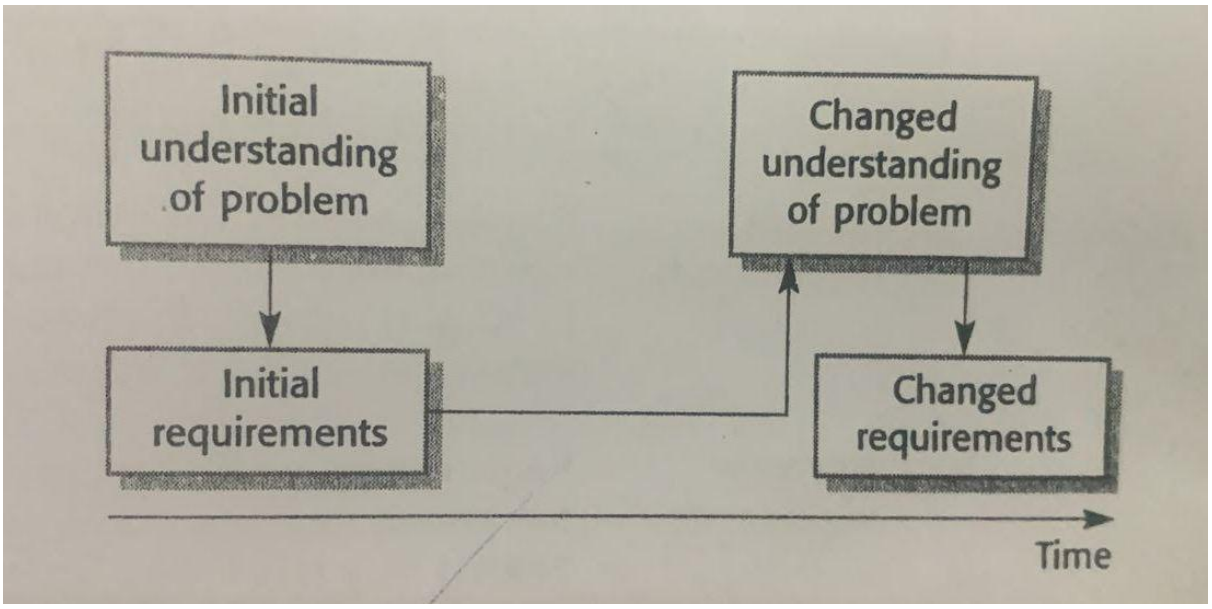
Enduring and Volatile Requirements :

Requirements evolution during the RE process and after a system has gone into service is inevitable.

Developing software requirements focuses attention on software capabilities, business objectives and other business systems.

After developing a better understanding of user needs, there is every scope to change the set of requirements.

Requirements Evolution :



Requirement Evolution falls into two categories :

a) **Enduring Requirements** : These are relatively stable requirements that derive from the core activity of the organization and which related directly to the domain of the system.

   For example, in a hospital, there will always be requirements concerned with patients, doctors, nurses and treatments.

b) **Volatile Requirements** : These are requirements that are likely to change during the system development process or after the system has been operational.

   An example would be requirements resulting from government healthcare policies.

19. Requirements Management Planning :

'Planning' is an essential first stage in the requirements management process.

During the requirements management stage, one has to decide on

a) **Requirements Identification** : Each requirement must be uniquely identified.

b) **A change management process** : This is the set of activities that assess the impact and cost of changes.

c) **Traceability Policies** : These policies define the relationships between requirements.

d) **CASE Tool :** support requirements management involves the processing of large amounts of information about the requirements.

Three types of traceability information :

a) **Source Traceability** : information links the requirements to the stakeholders who proposed the requirements and to the rationale for these requirements.

b) **Requirement Traceability** : information links dependent requirements within the requirement document.

c) **Design Traceability** : information links the requirements to the design modules where these requirements are implemented.

20. Requirements Change Management :

This should be applied to all proposed changes to the requirements.

Three principal stages in RCM :

Identified Problem → Problem Analysis & Change Specification →

Change Analysis & Costing →Change Implementation →

Revised Requirements.

* * * * *