UNIT-2 REQUIREMEN ANALYSIS AND REQUIREMENT ENGINEERING

Requirement Analysis:

- Classical Systems Development Life Cycle (SDLC) method
- Requirement determination
- System Requirement Specification (SRS)
- Fact finding techniques
- Process models - waterfall model, incremental, Evolutionary process models - prototype, spiral and concurrent development model.

Requirement Engineering:

- Problem recognition
- Requirement engineering tasks
- Use cases and functional specification
- Requirements validation

SDLC – Classical Waterfall Model

The classical waterfall model is elegant and intuitive. However, it is not a practical model because it cannot be used in actual software development projects. Thus, this model can be considered to be a theoretical way of developing software. The classical waterfall model divides the life cycle into the following phases.

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

Feasibility Study

The main aim of the feasibility study is to determine whether developing the software is financially worthwhile, technically feasible, or not.

Roughly understand what customer wants:

Data which would be input into the system
The processing needed on these data
Output data to be produced by the system
Various constraints on the behavior of the system
Activities during the feasibility study

Workout an overall understanding of the problem
Formulate different solution strategies
Examine alternate solution strategies in terms of resource required, cost of development, and development time.
Perform a cost/benefit analysis to determine which solution is the best. It may be possible that none of the solutions is feasible due to high cost, resource constraints, and technical reasons.
Cost-benefit analysis (CBA) analyzes the following:
Development cost: It roughly estimates the entire development cost of the software product.
Setup cost: To start any development, various resources like development systems and software are needed. It also depends on the type of software development. For example: suppose we want to make an app for iOS, then we need at least a few apple smartphones and apple desktop systems.
Operational costs: The developers, managers, and other supporting staff require tea, coffee, rented building, electricity, etc. There is various kind of operational cost that must be included during the estimation.

Identify the value of benefits, and benefits should always be greater than costs (investment of software or product development). The benefits include cost of development + operational + setup + (quantifiable + non-quantifiable profits). Feasibility study learning helps in writing a business case.

Requirements Analysis and Specification

The aim of this phase to understand the exact requirements of the customer and document them properly. It consists of three distinct activities:

Requirements gathering
Requirement analysis
Requirement specification
The goal of the requirements gathering activity is to collect all relevant information from the customer regarding the product to be developed.

The requirements analysis activity is begun by collecting all relevant data regarding the product to be developed from the product's user and from the customer through interviews and discussions. The data collected from such a group of users usually contain several contradictions and ambiguities since each user typically has only a partial and incomplete view of the system. Therefore, it is necessary to identify all ambiguities and contradictions in the requirement and resolve them through further discussions with the customer.

After all the ambiguities, inconsistencies, and incompleteness have been resolved, and the requirements are properly understood, the requirements specification activity can be started. During the activity, the user requirements are systematically organized into a Software Requirements Specification document.

The important components of this document are functional requirements, non-functional requirements, and the goal of the implementation.

Design

The goal of the design phase is to transform the requirements specified in the SRS document into a suitable structure or implemented in some programming language. In technical terms, during the design phase, the software architecture is derived from the SRS document.

There are two commonly used design approaches:

Traditional approach
Object-oriented approach
Traditional design approach: It consists of two different activities.

A structured analysis of the requirements specification is carried out where the detailed structure of the problem is examined. A structured design activity follows this.
During structured design, the results of the structured analysis are transformed into the software design.
To understand this, we need to understand structured analysis and design techniques.

Object-oriented design approach: In this technique, various objects in the problem domain and the solution domain are is further refined to obtain the detailed design.
first identified. The different relationships that exist among these objects are identified. This object structure

Coding and Unit Testing

The purpose of the coding phase of software development is to translate the software design into source code. Each component of the design is implemented as a program module. The end product of this phase is a set of program modules that have been individually tested. During this phase, each module is unit tested to determine the correct working of all the individual modules. It involves testing each module in isolation, as this is the most efficient way to debug the errors identified at this stage.

Integration and System Testing

Integration of different modules is undertaken once they have been coded and unit tested. During the integration and system testing phase, the modules are integrated in a planned manner. Finally, when all the modules have been successfully integrated and tested, system testing is carried out.

System testing aims to ensure that the developed system conforms to its requirements laid out in the SRS document. System testing usually consists of three different kinds of testing activities:

Alpha testing: It is system testing performed by the development team.
Beta testing: It is the system testing performed by a friendly set of customers.
Acceptance Testing: It is a system testing performed by the customer himself after the product delivery to determine whether to accept or reject the delivered product.

Maintenance

Maintenance of a typical software product requires much more than the effort necessary to develop the product itself. Many studies carried out in the past confirm this and indicate that the relative effort of developing a typical software product to its maintenance effort is roughly in the 40:60 ratios.

Maintenance involves the following four kinds of activities:

Corrective errors that were not discovered during the product development phase. This is called corrective maintenance.
Improving the system's implementation and enhancing the system's functionalities according to the customer's requirements. This is called perfective maintenance.
Porting the software to work in a new environment. This is called adaptive maintenance.
Maintenance is regularly performed on the solution to lessen the likelihood of it failing. It is performed while the solution is still working so that it does not break down unexpectedly. This is called Preventive maintenance.

What is Requirements Determination?

A requirement is a vital feature of a new system which may include processing or capturing of data, controlling the activities of business, producing information and supporting the management.
Requirements determination involves studying the existing system and gathering details to find out what are the requirements, how it works, and where improvements should be made.

Major Activities in requirement Determination

Requirements Anticipation
 It predicts the characteristics of system based on previous experience which include certain problems or features and requirements for a new system.
 It can lead to analysis of areas that would otherwise go unnoticed by inexperienced analyst. But if shortcuts are taken and bias is introduced in conducting the investigation, then requirement Anticipation can be halfbaked.

Requirements Investigation
 It is studying the current system and documenting its features for further analysis.
 It is at the heart of system analysis where analyst documenting and describing system features using fact-finding techniques, prototyping, and computer assisted tools.

Requirements Specifications
 It includes the analysis of data which determine the requirement specification, description of features for new system, and specifying what information requirements will be provided.
 It includes analysis of factual data, identification of essential requirements, and selection of Requirement-fulfillment strategies.

Fact Finding Techniques

To study any system the analyst needs to do collect facts and all relevant information. the facts when expressed in quantitative form are termed as data. The success of any project is depended upon the accuracy of available data. Accurate information can be collected with help of certain methods/ techniques. These specific methods for finding information of the system are termed as fact finding techniques. Interview, Questionnaire, Record View and Observations are the different fact finding techniques used by the analyst. The analyst may use more than one technique for investigation.

Interview
This method is used to collect the information from groups or individuals. Analyst selects the people who are related with the system for the interview. In this method the analyst sits face to face with the people and records their responses.

The interviewer must plan in advance the type of questions he/ she is going to ask and should be ready to answer any type of question. He should also choose a suitable place and time which will be comfortable for the respondent.

The information collected is quite accurate and reliable as the interviewer can clear and cross check the doubts there itself. This method also helps gap the areas of misunderstandings and help to discuss about the future problems. Structured and unstructured are the two sub categories of Interview. Structured interview is more formal interview where fixed questions are asked and specific information is collected whereas unstructured interview is more or less like a casual conversation where in-depth areas topics are covered and other information
apart from the topic may also be obtained.

Questionnaire
It is the technique used to extract information from number of people. This method can be adopted and used only by an skillful analyst. The Questionnaire consists of series of questions framed together in logical manner. The questions are simple, clear and to the point. This method is very useful for attaining information from people who are concerned with the usage of the system and who are living in different countries. The questionnaire can be mailed or send to people by post. This is the cheapest source of fact finding.

RecordView
The information related to the system is published in the sources like newspapers, magazines, journals, documents etc. This record review helps the analyst to get valuable information about the system and the organization.

Observation
Unlike the other fact finding techniques, in this method the analyst himself visits

the organization and observes and understand the flow of documents, working of
the existing system, the users of the system etc. For this method to be adopted it
takes an analyst to perform this job as he knows which points should be noticed and highlighted. In analyst may observe the unwanted things as well and simply cause delay in the development of the new system.

Feasibility Study

Feasibility Study can be considered as preliminary investigation that helps the management to take decision about whether study of system should be feasible for development or not.

 It identifies the possibility of improving an existing system, developing a new system, and produce refined estimates for further development of system.
 It is used to obtain the outline of the problem and decide whether feasible or appropriate solution exists or not.
 The main objective of a feasibility study is to acquire problem scope instead of solving the problem.
 The output of a feasibility study is a formal system proposal act as decision document which includes the complete nature and scope of the proposed system.

Steps Involved in Feasibility Analysis

The following steps are to be followed while performing feasibility analysis −

 Form a project team and appoint a project leader.
 Develop system flowcharts.
 Identify the deficiencies of current system and set goals.
 Enumerate the alternative solution or potential candidate system to meet goals.
 Determine the feasibility of each alternative such as technical feasibility, operational feasibility, etc.
 Weight the performance and cost effectiveness of each candidate system.
 Rank the other alternatives and select the best candidate system.
 Prepare a system proposal of final project directive to management for approval.

Types of Feasibilities

Economic Feasibility

 It is evaluating the effectiveness of candidate system by using cost/benefit analysis method.
 It demonstrates the net benefit from the candidate system in terms of benefits and costs to the organization.
 The main aim of Economic Feasibility Analysis (EFS) is to estimate the economic requirements of candidate system before investments funds are committed to proposal.
 It prefers the alternative which will maximize the net worth of organization by earliest and
highest return of funds along with lowest level of risk involved in developing the candidate system.

Technical Feasibility
 It investigates the technical feasibility of each implementation alternative.
 It analyzes and determines whether the solution can be supported by existing technology or not.
 The analyst determines whether current technical resources be upgraded or added it that
fulfill the new requirements.
 It ensures that the candidate system provides appropriate responses to what extent it can
support the technical enhancement.

Operational Feasibility
 It determines whether the system is operating effectively once it is developed and
implemented.
 It ensures that the management should support the proposed system and its working
feasible in the current organizational environment.
 It analyzes whether the users will be affected and they accept the modified or new business
methods that affect the possible system benefits.
 It also ensures that the computer resources and network architecture of candidate system
are workable.

Behavioral Feasibility
 It evaluates and estimates the user attitude or behavior towards the development of new
system.
 It helps in determining if the system requires special effort to educate, retrain, transfer, and
changes in employee's job status on new ways of conducting business.

Schedule Feasibility

 It ensures that the project should be completed within given time constraint or schedule.
 It also verifies and validates whether the deadlines of project are reasonable or not.

Cost benefit analysis: What is it?
A cost benefit analysis (also known as a benefit cost analysis) is a process by which organizations can analyze decisions, systems or projects, or determine a value for intangibles. The model is built by identifying the benefits of an action as well as the associated costs, and subtracting the costs from benefits. When completed, a cost benefit analysis will yield concrete results that can be used to develop reasonable conclusions around the feasibility and/or advisability of a decision or situation.

Why Use Cost Benefit Analysis?
Organizations rely on cost benefit analysis to support decision making because it provides an agnostic, evidence-based view of the issue being evaluated—without the influences of opinion, politics, or bias. By providing an unclouded view of the consequences of a decision, cost benefit analysis is an invaluable tool in developing business strategy, evaluating a new hire, or making resource allocation or purchase decisions.

Scenarios Utilizing Cost Benefit Analysis
As mentioned previously, cost benefit analysis is the foundation of the decision-making process across a wide variety of disciplines. In business, government, finance, and even the nonprofit world, cost benefit analysis offers unique and valuable insight when:
 Developing benchmarks for comparing projects
 Deciding whether to pursue a proposed project
 Evaluating new hires
 Weighing investment opportunities
 Measuring social benefits
 Appraising the desirability of suggested policies
 Assessing change initiatives
 Quantifying effects on stakeholders and participants

How to Do a Cost Benefit Analysis
While there is no “standard” format for performing a cost benefit analysis, there are certain core elements that will be present across almost all analyses. Use the structure that works best for your situation or industry, or try one of the resources and tools listed at the end of this article. We’ll go through the five basic steps to performing a cost benefit analysis in the sections below, but first, here’s a high-level of overview:
1. Establish a framework to outline the parameters of the analysis
2. Identify costs and benefits so they can be categorized by type, and intent
3. Calculate costs and benefits across the assumed life of a project or initiative

4. Compare cost and benefits using aggregate information
5. Analyze results and make an informed, final recommendation

Identify and Categorize Costs and Benefits

Now that your framework is in place, it's time to sort your costs and benefits into buckets by type. The primary categories that costs and benefits fall into are direct/indirect, tangible/intangible, and real:

 Direct costs are often associated with production of a cost object (product, service, customer, project, or activity)

 Indirect costs are usually fixed in nature, and may come from overhead of a department or cost center

 Tangible costs are easy to measure and quantify, and are usually related to an identifiable source or asset, like payroll, rent, and purchasing tools

 Intangible costs are difficult to identify and measure, like shifts in customer satisfaction, and productivity levels

 Real costs are expenses associated with producing an offering, such as labor costs and raw materials.

]

Software Requirements Specification

for

<Project>

Version 1.0 approved

Prepared by <author>

<organization>

<date created>

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

Introduction

Purpose

<Identify the product whose software requirements are specified in this document, including the revision or release number. Describe the scope of the product that is covered by this SRS, particularly if this SRS describes only part of the system or a single subsystem.>

Document Conventions

<Describe any standards or typographical conventions that were followed when writing this SRS, such as fonts or highlighting that have special significance. For example, state whether priorities for higher-level requirements are assumed to be inherited by detailed requirements, or whether every requirement statement is to have its own priority.>

Intended Audience and Reading Suggestions

<Describe the different types of reader that the document is intended for, such as developers, project managers, marketing staff, users, testers, and documentation writers. Describe what the rest of this SRS contains and how it is organized. Suggest a sequence for reading the document, beginning with the overview sections and proceeding through the sections that are most pertinent to each reader type.>

Product Scope

<Provide a short description of the software being specified and its purpose, including relevant benefits, objectives, and goals. Relate the software to corporate goals or business strategies. If a separate vision and scope document is available, refer to it rather than duplicating its contents here.>

References

<List any other documents or Web addresses to which this SRS refers. These may include user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.>

Overall Description

Product Perspective

<Describe the context and origin of the product being specified in this SRS. For example, state whether this product is a follow-on member of a product family, a replacement for certain existing systems, or a new, self-contained product. If the SRS defines a component of a larger system, relate the requirements of the larger system to the functionality of this software and identify interfaces between the two. A simple diagram that shows the major components of the overall system, subsystem interconnections, and external interfaces can be helpful.>

Product Functions

<Summarize the major functions the product must perform or must let the user perform. Details will be provided in Section 3, so only a high level summary (such as a bullet list) is needed here. Organize the functions to make them understandable to any reader of the SRS. A picture of the major groups of related requirements and how they relate, such as a top level data flow diagram or object class diagram, is often effective.>

User Classes and Characteristics

<Identify the various user classes that you anticipate will use this product. User classes may be differentiated based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience. Describe the pertinent characteristics of each user class. Certain requirements may pertain only to certain user classes. Distinguish the most important user classes for this product from those who are less important to satisfy.>

Operating Environment

<Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications with which it must peacefully coexist.>

Design and Implementation Constraints

<Describe any items or issues that will limit the options available to the developers. These might include: corporate or regulatory policies; hardware limitations (timing requirements, memory requirements); interfaces to other applications; specific technologies, tools, and databases to be used; parallel operations; language requirements; communications protocols; security considerations; design conventions or programming standards (for example, if the customer's organization will be responsible for maintaining the delivered software).>

User Documentation

<List the user documentation components (such as user manuals, on-line help, and tutorials) that will be delivered along with the software. Identify any known user documentation delivery formats or standards.>

Assumptions and Dependencies

<List any assumed factors (as opposed to known facts) that could affect the requirements stated in the SRS. These could include third-party or commercial components that you plan to use, issues around the development or operating environment, or constraints. The project could be affected if these assumptions are incorrect, are not shared, or change. Also identify any dependencies the project has on external factors, such as software components that you intend to reuse from another project, unless they are already documented elsewhere (for example, in the vision and scope document or the project plan).>

External Interface Requirements

User Interfaces

<Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components

for which a user interface is needed. Details of the user interface design should be documented in a separate user interface specification.>

Hardware Interfaces

<Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware, and communication protocols to be used.>

Software Interfaces

<Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Refer to documents that describe detailed application programming interface protocols. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.>

Communications Interfaces

<Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.>

System Features

<This template illustrates organizing the functional requirements for the product by system features, the major services provided by the product. You may prefer to organize this section by use case, mode of operation, user class, object class, functional hierarchy, or combinations of these, whatever makes the most logical sense for your product.>

System Feature 1

<Don't really say "System Feature 1." State the feature name in just a few words.>

4.1.1 Description and Priority

<Provide a short description of the feature and indicate whether it is of High, Medium, or Low priority. You could also include specific priority component ratings, such as benefit, penalty, cost, and risk (each rated on a relative scale from a low of 1 to a high of 9).>

4.1.2 Stimulus/Response Sequences

<List the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.>

4.1.3 Functional Requirements

<Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present in order for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary. Use "TBD" as a placeholder to indicate when necessary information is not yet available.>

<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>

REQ-1:
REQ-2:

System Feature 2 (and so on)

Other Nonfunctional Requirements

Performance Requirements

<If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.>

Safety Requirements

<Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product's design or use. Define any safety certifications that must be satisfied.>

Security Requirements

<Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.>

Software Quality Attributes

<Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.>

Business Rules

<List any operating principles about the product, such as which individuals or roles can perform which functions under specific circumstances. These are not functional requirements in themselves, but they may imply certain functional requirements to enforce the rules.>

Other Requirements

<Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.>

Appendix A: Glossary

<Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.>

Appendix B: Analysis Models

<Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams.>

Appendix C: To Be Determined List

<Collect a numbered list of the TBD (to be determined) references that remain in the SRS so they can be tracked to closure.>

PROCESS MODELS

Process Model/Prescriptive Process Model

A Process was defined as a collection of work activities, actions, and tasks that are performed when some work product is to be created.

Each of these activities, actions, and tasks reside within a framework or model that defines their relationship with the process and with one another.

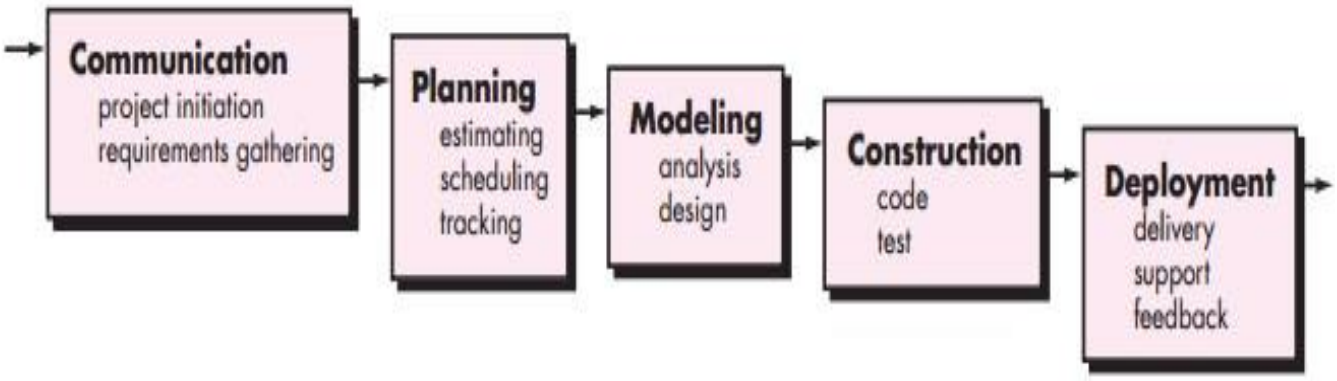
These models are also known as Prescriptive Process Model. **"Prescriptive"** because they prescribe a set of process elements—framework activities, software engineering actions, tasks, work products, quality assurance, and change control mechanisms for each project. Each process model also prescribes a process flow (also called a work flow)—that is, the manner in which the process elements are interrelated to one another.

There are **FOUR** types of Process Models:

1) Waterfall Model
2) Incremental Model
3) Evolutionary Models
    i) Prototyping Model
    ii) Spiral Model
4) Concurrent Models

1) Waterfall Model

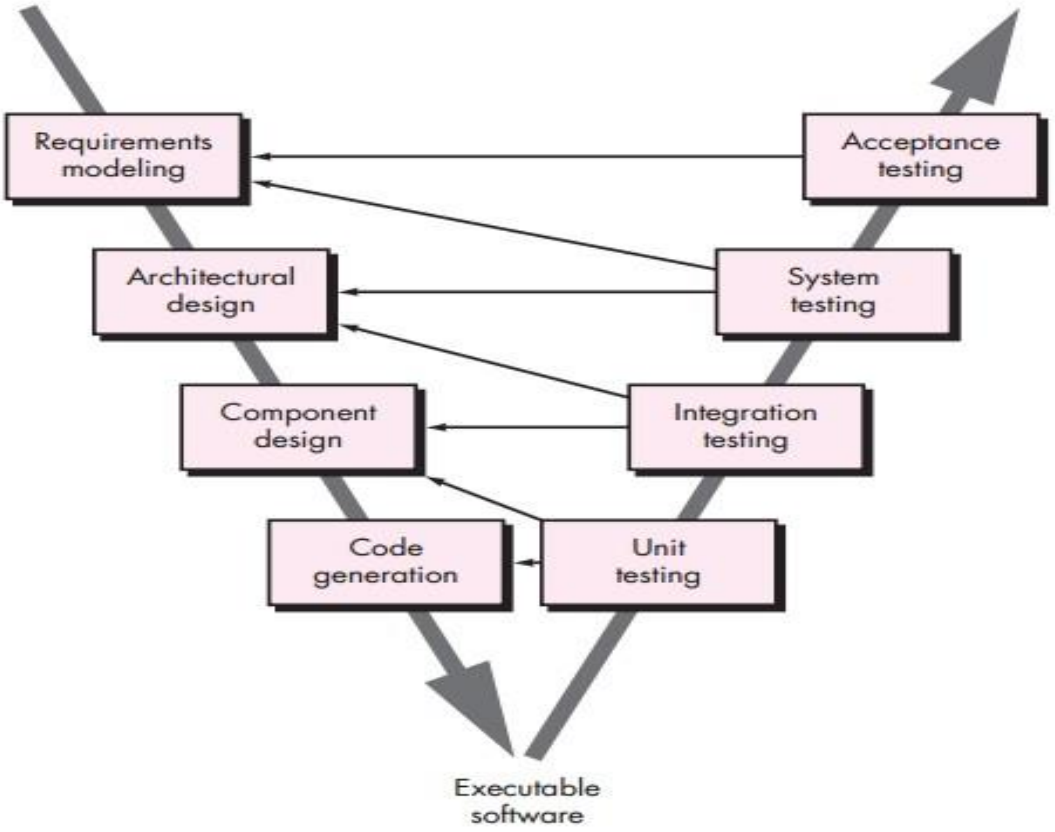
The waterfall model, sometimes called the classic life cycle, suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in ongoing support of the completed software **Figure-D.**



[Figure-D- Waterfall Model]

V-model: A variation in the representation of the waterfall model is called the **V-model.**

- Represented in Figure-E, the V-model depicts the relationship of quality assurance actions to the actions associated with communication, modeling, and early construction activities.
- As a software team moves down the left side of the V, basic problem requirements are refined into progressively more detailed and technical representations of the problem and its solution.
- Once code has been generated, the team moves up the right side of the V, essentially performing a series of tests (quality assurance actions) that validate each of the models created as the team moved down the left side.
- In reality, there is no fundamental difference between the classic life cycle and the V-model.
- The V-model provides a way of visualizing how verification and validation actions are applied to earlier engineering work.



[Figure-E- V-Model]

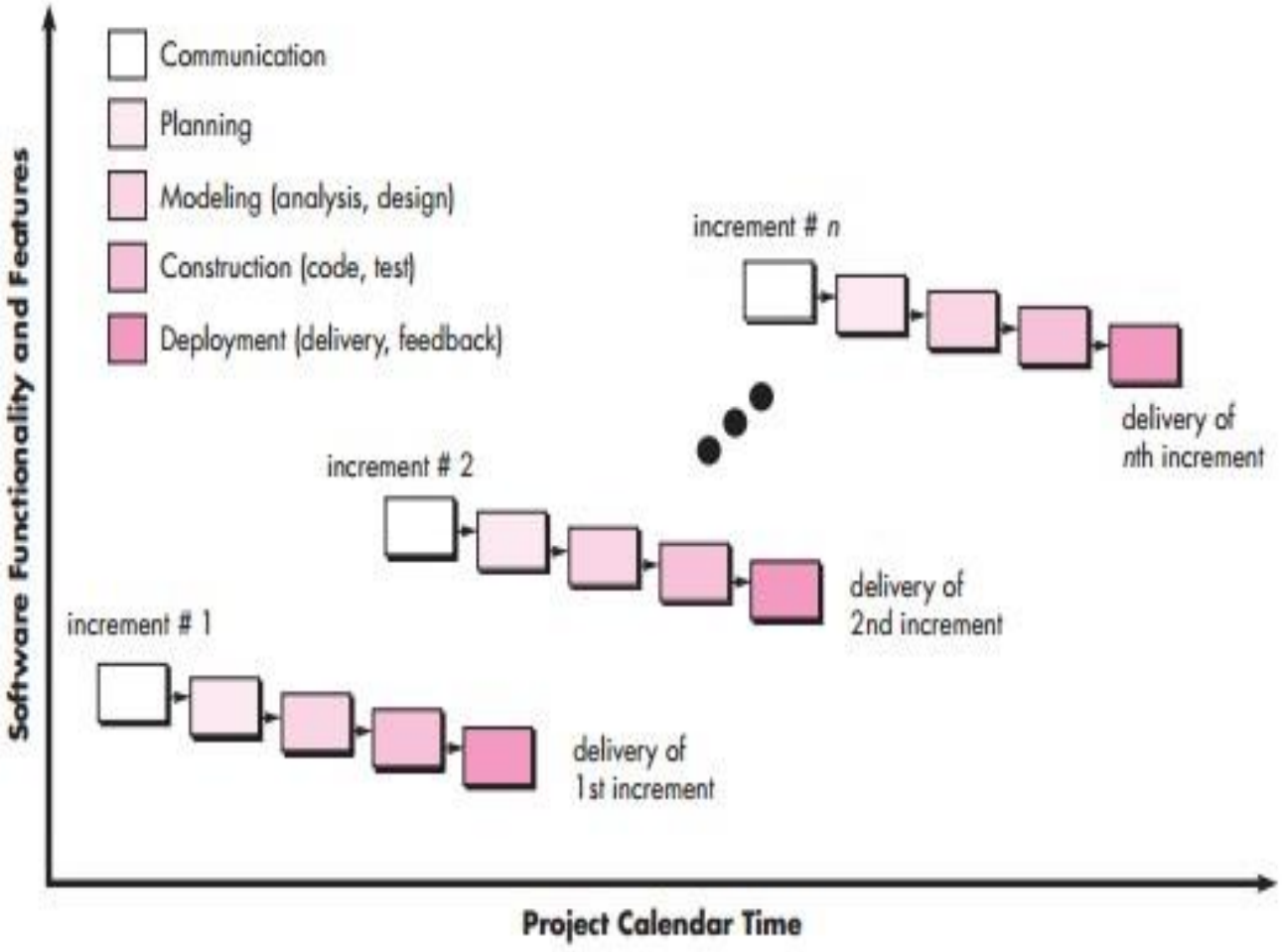
2) Incremental Model

The incremental model combines elements of linear and parallel process flows.

Referring to **Figure-F,** the incremental model applies linear sequences in a staggered fashion as calendar time progresses.

Each linear sequence produces deliverable "increments" of the software in a manner that is similar to the increments produced by an evolutionary process flow.

When an incremental model is used, the first increment is often a **core product.** That is, basic requirements are addressed but many supplementary features (some known, others unknown) remain undelivered. The core product is used by the customer (or undergoes detailed evaluation).



[Figure-F- Incremental Model]

As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality.

This process is repeated following the delivery of each increment, until the complete product is produced.

3) Evolutionary Models:

Evolutionary models are iterative. They are characterized in a manner that enables you to develop increasingly more complete versions of the software.

There are **two common evolutionary process models:**

i) Prototyping Model

ii) Spiral Model

i) Prototyping Model

Prototyping paradigm is best suites in below situations:

- A customer defines a set of general objectives for software, but **does not identify detailed requirements** for functions and features.
- In other cases, the **developer may be unsure of the efficiency** of an algorithm, the adapt-ability of an operating system, or the form that human-machine interaction should take.
- The prototyping paradigm assists you and other stakeholders [customers] to better understand what is to be built when **requirements are fuzzy.**

Prototype models includes **FIVE** phases as shown in above **figure-G**:

1) Communication:

- Starting phase of process model. You meet with other stakeholders to define the overall objectives for the software, identify whatever requirements are known.

2) Quick Plan:

- Prototyping iteration is planned quickly, and modeling (in the form of a "quick design") occurs.

3) Modeling Quick Design:

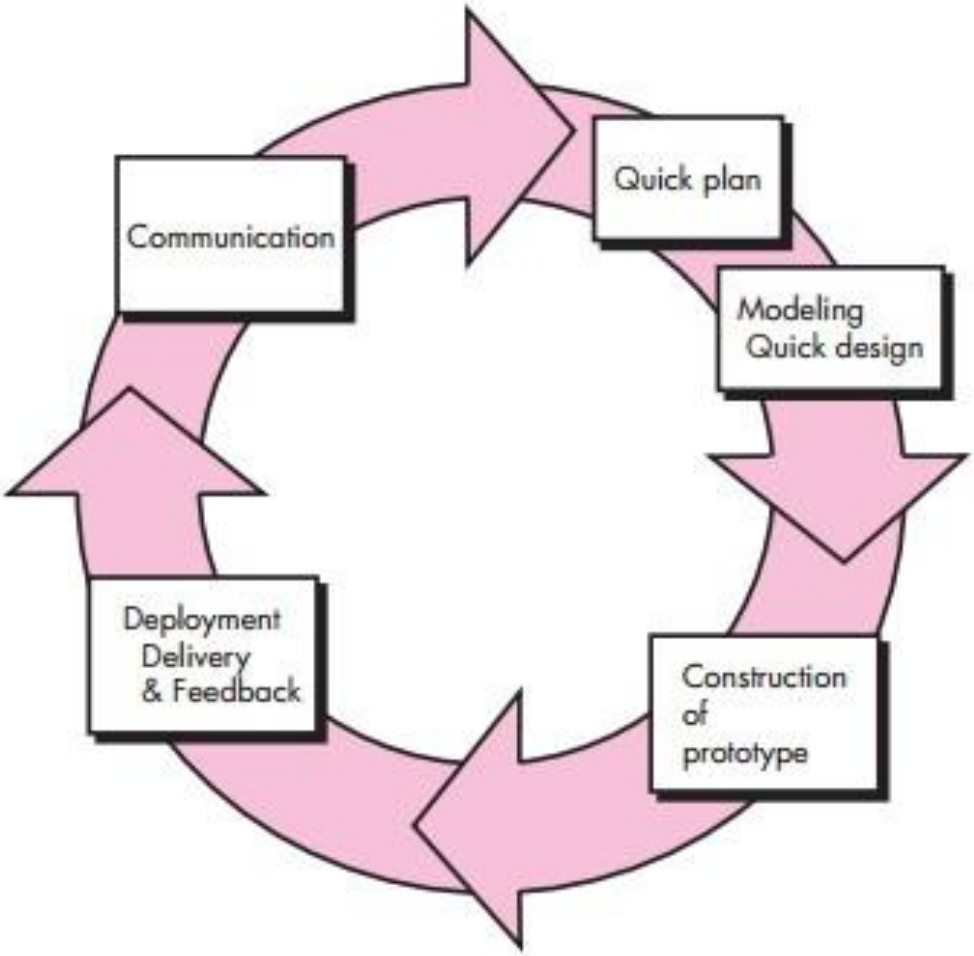
- A quick design focuses on a representation of those aspects of the software that will be visible to end users (e.g., human interface layout or output display formats).

4) Construction of Prototype:

- The quick design leads to the construction of a prototype. Coding of different modules will be done in this phase.

5) Deployment Delivery & Feedback:

- The prototype is deployed and evaluated by stakeholders, who provide feedback that is used to further refine requirements.



[Figure-G- Prototype Model]

ii) Spiral Model

It is originally proposed [developed] by Barry Boehm.

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.**

So, the spiral model is the combination of two process models waterfall model and prototyping model.

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

Boehm describes the model in the following manner:

- The spiral development model is **a risk-driven process model** generator that is used to guide multi-stakeholder concurrent engineering of software intensive systems.
- It has **two** main distinguishing features.
  a) Cyclic approach: for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk.
  b) Anchor point milestones: for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.

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

Each of the framework activities represent one segment of the spiral path illustrated in **Figure-H**.

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

Risk is considered as each revolution is made.

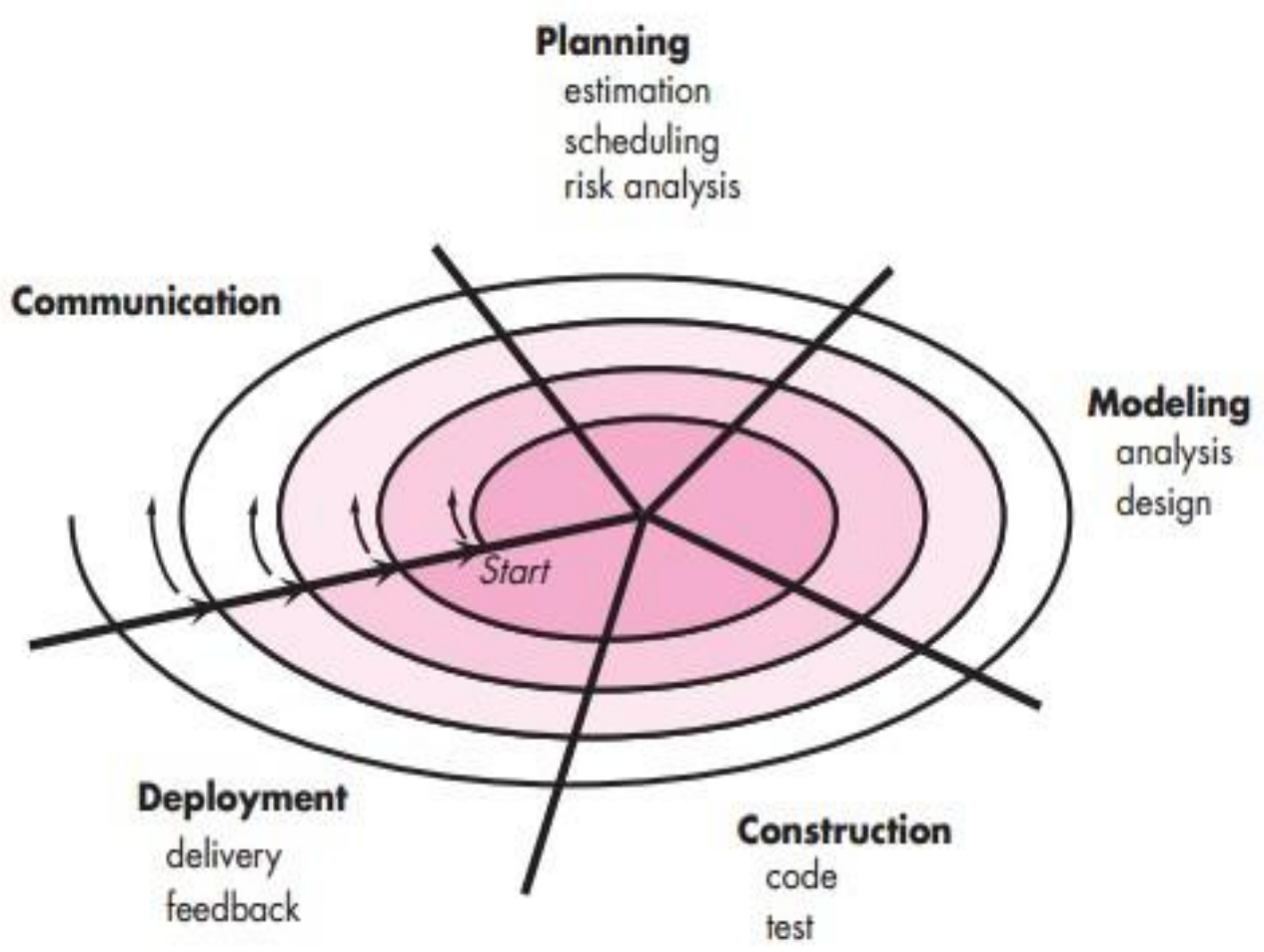
Anchor point milestones—a combination of work products and conditions that are attained along the path of the spiral—are noted for each evolutionary pass.

The first circuit around the spiral might result in the **development of a product specification;**

Subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software.

Each pass through the planning region results in adjustments to the **project plan.**

Cost and schedule are adjusted based on **feedback** derived from the customer after **delivery.**



[Figure-H- Spiral Model]

Advantages:

Unlike other process models that end when software is delivered, the **spiral model can be adapted to apply throughout the life of the computer software.**

The spiral model is a realistic approach to the development of large-scale systems and software. Because software evolves as the process progresses, the developer and customer better understand and react to risks at each evolutionary level.

The spiral model demands a direct consideration of technical risks at all stages of the project and, if properly applied, should reduce risks before they become problematic.

Disadvantages:

It may be difficult to convince customers (particularly in contract situations) that the evolutionary approach is controllable.

It demands considerable risk assessment expertise and relies on this expertise for success.

If a major risk is not uncovered and managed, problems will undoubtedly occur.

4) Concurrent Models:

The concurrent development model, sometimes called concurrent engineering, allows a software team to represent iterative and concurrent elements of any of the process models.

For example, the modeling activity defined for the spiral model is accomplished by invoking one or more of the following software engineering actions: prototyping, analysis, and design.

Figure-I provides a schematic representation of one software engineering activity within the modeling activity using a concurrent modeling approach.

The activity—**modeling**—may be in any one of the states noted at any given time.

Similarly, other activities, actions, or tasks (e.g., **communication or construction**) can be represented in an analogous manner.

All software engineering activities exist concurrently but reside in different states.

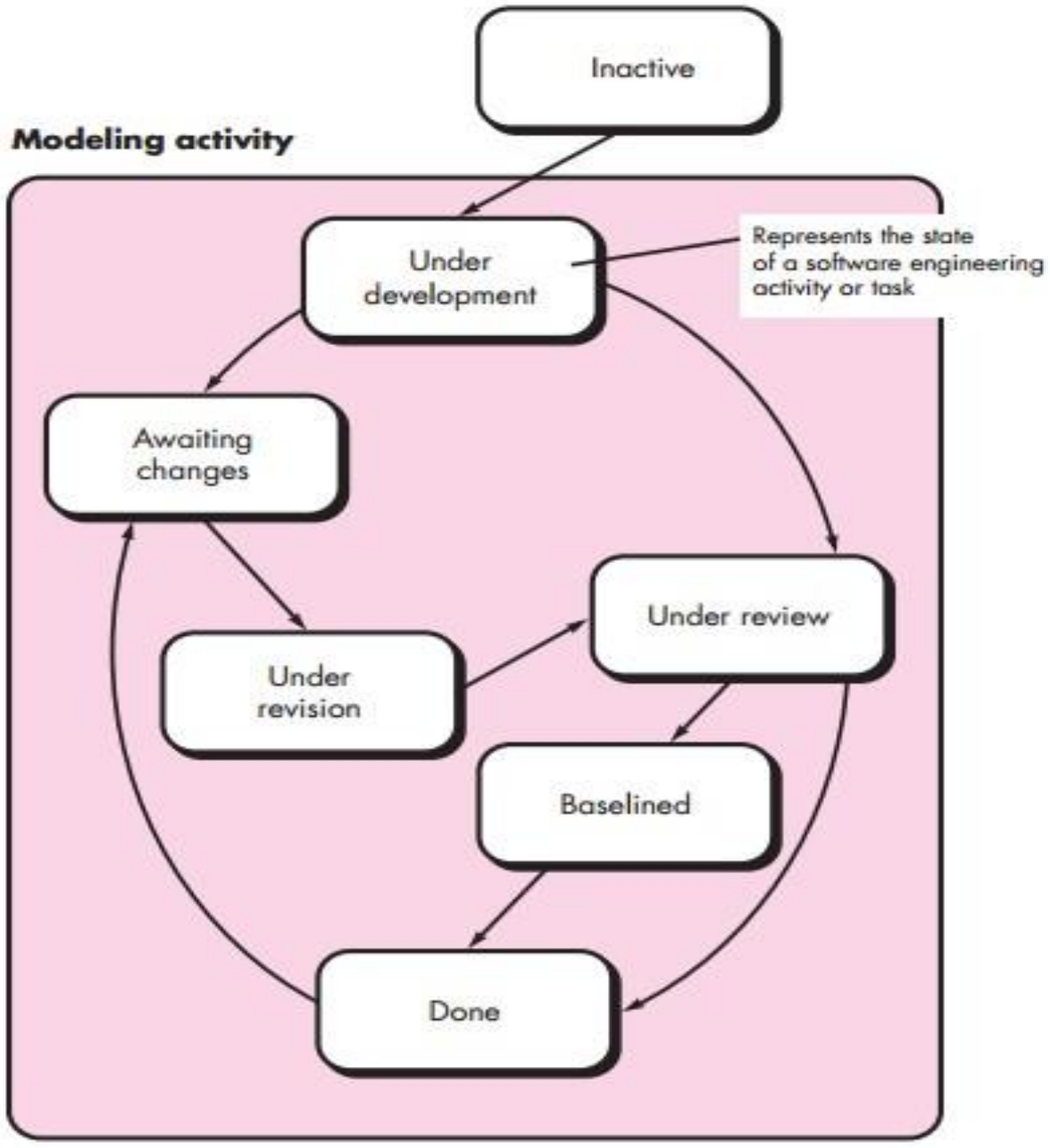
Concurrent modeling defines a series of events that will trigger transitions from state to state for each of the software engineering activities, actions, or tasks.

For example, during early stages of design (a major software engineering action that occurs during the modeling activity), an inconsistency in the requirements model is uncovered.

This generates the event analysis model correction, which will trigger the requirements analysis action from the **done state** into the **awaiting changes state.**

Concurrent modeling is applicable to all types of software development and pro-vides an accurate picture of the current state of a project. Rather than

confining soft-ware engineering activities, actions, and tasks to a sequence of events, it defines a process network.



[Concurrent Model]

Requirement Engineering

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

Requirement Engineering Process

It is a four-step process, which includes -

1. Feasibility Study
2. Requirement Elicitation and Analysis
3. Software Requirement Specification
4. Software Requirement Validation
5. Software Requirement Management

1. Feasibility Study:

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

Types of Feasibility:

1. **Technical Feasibility** - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.
2. **Operational Feasibility** - Operational feasibility assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.
3. **Economic Feasibility** - Economic feasibility decides whether the necessary software can generate financial profits for an organization.

2. Requirement Elicitation and Analysis:

This is also known as the **gathering of requirements**. Here, requirements are identified with the help of customers and existing systems processes, if available.

Analysis of requirements starts with requirement elicitation. The requirements are analyzed to identify inconsistencies, defects, omission, etc. We describe requirements in terms of relationships and also resolve conflicts if any.

Problems of Elicitation and Analysis

- Getting all, and only, the right people involved.
- Stakeholders often don't know what they want
- Stakeholders express requirements in their terms.
- Stakeholders may have conflicting requirements.
- Requirement change during the analysis process.
- Organizational and political factors may influence system requirements.

3. Software Requirement Specification:

Software requirement specification is a kind of document which is created by a software analyst after the requirements collected from the various sources - the requirement received by the customer written in ordinary language. It is the job of the analyst to write the requirement in technical language so that they can be understood and beneficial by the development team.

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

- **Data Flow Diagrams:** Data Flow Diagrams (DFDs) are used widely for modeling the requirements. DFD shows the flow of data through a system. The system may be a company, an organization, a set of procedures, a computer hardware system, a software system, or any combination of the preceding. The DFD is also known as a data flow graph or bubble chart.
- **Data Dictionaries:** Data Dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirements stage, the data dictionary should at least define customer data items, to ensure that the customer and developers use the same definition and terminologies.

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

4. Software Requirement Validation:

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

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

Requirements Validation Techniques

- **Requirements reviews/inspections:** systematic manual analysis of the requirements.
- **Prototyping:** Using an executable model of the system to check requirements.
- **Test-case generation:** Developing tests for requirements to check testability.
- **Automated consistency analysis:** checking for the consistency of structured requirements descriptions.

Software Requirement Management:

Requirement management is the process of managing changing requirements during the requirements engineering process and system development.

New requirements emerge during the process as business needs a change, and a better understanding of the system is developed.

The priority of requirements from different viewpoints changes during development process.

The business and technical environment of the system changes during the development.

Prerequisite of Software requirements

Collection of software requirements is the basis of the entire software development project. Hence they should be clear, correct, and well-defined.

A complete Software Requirement Specifications should be:

- Clear
- Correct
- Consistent
- Coherent
- Comprehensible
- Modifiable
- Verifiable
- Prioritized
- Unambiguous
- Traceable
- Credible source

Software Requirements: Largely software requirements must be categorized into two categories:

1. **Functional 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 are describing the behavior of the system as it correlates to the system's functionality.
2. **Non-functional Requirements:** This can be the necessities that specify the criteria that can be used to decide the operation instead of specific behaviors of the system. Non-functional requirements are divided into two main categories:
    - **Execution qualities** like security and usability, which are observable at run time.
    - **Evolution qualities** like testability, maintainability, extensibility, and scalability that embodied in the static structure of the software system.

References:

1. https://notepub.io/notes/software-engineering/software-development-life-cycle/sdlc-classical-waterfall-model/
2. https://www.nielit.gov.in/
3. IEEE SRS Template for Software
4. https://www.javatpoint.com/software-engineering-requirement-engineering