**UNIT:2**

***Software Requirements: Functional and Non-functional, User requirements, System requirements, Software Requirements Document,***

***Requirement Engineering Process: Feasibility Studies, Requirement elicitation and analysis, Requirement validation, Requirements Management .***

***System models: Context models, behavioral models, data models, object models, structured methods***

**What Is a Software Requirements Specification (SRS) Document?**

* A software requirements specification (SRS) is a document that describes what the software will do and how it will be expected to perform. It also describes the functionality the product needs to fulfill the needs of all stakeholders (business, users).

The elements that comprise an SRS can be simply summarized into four Ds:

1. [**Define** your product's purpose.](https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document#purpose)
2. [**Describe** what you're building.](https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document#describe)
3. [**Detail** the requirements.](https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document#detail)
4. [**Deliver** it for approval.](https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document#approve)

**Why Use an SRS Document?**

* An SRS gives you a complete picture of your entire project. It provides a single source of truth that every team involved in development will follow. It is your plan of action and keeps all your teams — from development and testing to maintenance — on the same page.



# **Requirements Engineering Process**

* **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**



**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:**

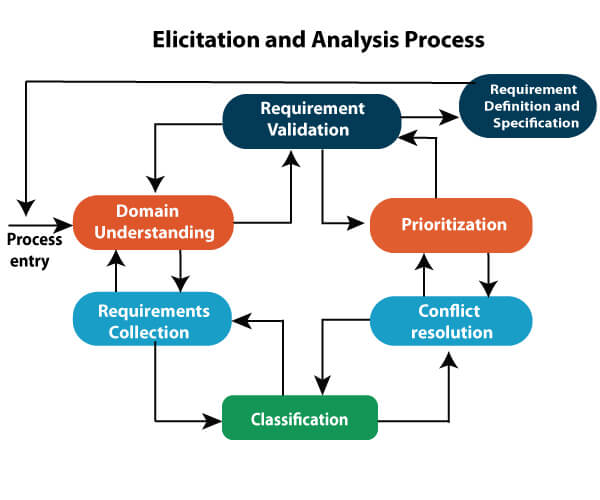
1. **Technical Feasibility** - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.
2. **Operational Feasibility** - Operational feasibility assesses the range in which the required software performs a series of levels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

1. **Getting all, and only, the right people involved.**
2. **Stakeholders often don't know what they want**
3. **Stakeholders express requirements in their terms.**
4. **Stakeholders may have conflicting requirements.**
5. **Requirement change during the analysis process.**
6. **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 -

1. **If they can practically implement**
2. **If they are correct and as per the functionality and specially of software**
3. **If there are any ambiguities**
4. **If they are full**
5. **If they can describe**

**Requirements Validation Techniques**

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

### **Software Requirement Management:**

1. Requirement management is the process of managing changing requirements during the requirements engineering process and system development.
2. New requirements emerge during the process as business needs a change, and a better understanding of the system is developed.
3. The priority of requirements from different viewpoints changes during development process.
4. 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:

1. **Clear**
2. **Correct**
3. **Consistent**
4. **Coherent**
5. **Comprehensible**
6. **Modifiable**
7. **Verifiable**
8. **Prioritized**
9. **Unambiguous**
10. **Traceable**
11. **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.

# **Classical Analysis**

* **Classical analysis in software engineering refers to the traditional and well-established methods and techniques used to analyze, design, and develop software systems.**
* **It encompasses a set of structured and systematic approaches to understanding and solving complex problems in software development.**
* While modern software engineering has evolved to include agile, DevOps, and other methodologies, classical analysis methods are still valuable and applicable in many scenarios.

Some key aspects of classical analysis in software engineering include:

**Requirements Analysis:** This is the initial phase of software development where the requirements of the system are gathered, documented, and analyzed. Classical methods often involve techniques like interviews, surveys, and documentation analysis to elicit and specify requirements.

**System Design:** Classical analysis places a strong emphasis on system design. It involves creating detailed architectural and structural blueprints for the software system. Common techniques include data flow diagrams, entity-relationship diagrams, and structured design methods like Structured Analysis and Structured Design (SAD/STD).

**Structured Programming:** Classical analysis often promotes structured programming techniques, which emphasize modular code design and the use of control structures like loops and conditionals. This helps improve code maintainability and readability.

**Waterfall Model:** The classical analysis approach is often associated with the waterfall model of software development. In this model, each phase of development (requirements, design, implementation, testing, maintenance) follows a linear and sequential progression.

**Formal Methods:** Classical analysis sometimes involves the use of formal methods to mathematically prove the correctness of software systems. These methods are particularly useful for critical systems where correctness is paramount.

**Quality Assurance:** Classical analysis places a strong emphasis on quality assurance through techniques like code reviews, inspections, and formal testing. The goal is to ensure that the software meets its specifications and functions as intended.

**Documentation:** Comprehensive documentation is a hallmark of classical analysis. It includes requirements documents, design specifications, and user manuals. This documentation serves as a reference for developers and helps with system maintenance.

**Predictive Planning:** Classical analysis often relies on detailed project planning and estimation techniques to predict the schedule, cost, and resources required for a software project .

# 

*THE END*

*Thank You*