Unit 2 Software Engineering

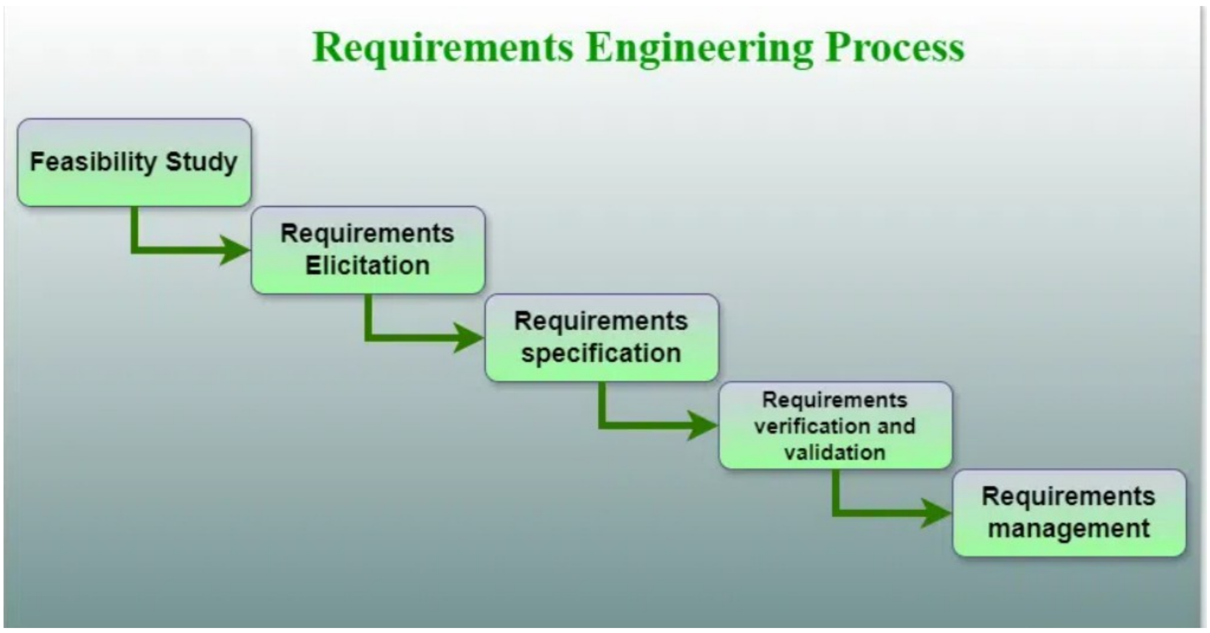
Requirements Engineering Process in Software Engineering

Requirements Engineering is the process of identifying, eliciting, analysing, specifying, validating, and managing the needs and expectations of stakeholders for a software system.

What is Requirements Engineering?

A systematic and strict approach to the definition, creation, and verification of requirements for a software system is known as requirements engineering. To guarantee the effective creation of a software product, the requirements engineering process entails several tasks that help in understanding, recording, and managing the demands of stakeholders.

Requirements Engineering Process


Requirements Engineering Process
Feasibility Study
Requirements Elicitation
Requirements specification
Requirements verification and validation
Requirements management

Requirements Engineering Process

1. **Feasibility Study**
2. **Requirements elicitation**
3. **Requirements specification**
4. **Requirements for verification and validation**

5. **Requirements management**

1. Feasibility Study

The feasibility study mainly concentrates on below five mentioned areas below. Among these Economic Feasibility Study is the most important part of the feasibility analysis and the Legal Feasibility Study is less considered feasibility analysis.

1. **Technical Feasibility**: In Technical Feasibility current resources both hardware software along required technology are analyzed/assessed to develop the project. This technical feasibility study reports whether there are correct required resources and technologies that will be used for project development. Along with this, the feasibility study also analyzes the technical skills and capabilities of the technical team, whether existing technology can be used or not, whether maintenance and up-gradation are easy or not for the chosen technology, etc.
2. **Operational Feasibility:** In Operational Feasibility degree of providing service to requirements is analyzed along with how easy the product will be to operate and maintain after deployment. Along with this other operational scopes are determining the usability of the product, Determining suggested solution by the software development team is acceptable or not, etc.
3. **Economic Feasibility:** In the Economic Feasibility study cost and benefit of the project are analyzed. This means under this feasibility study a detailed analysis is carried out will be cost of the project for development which includes all required costs for final development hardware and software resources required, design and development costs operational costs, and so on. After that, it is analyzed whether the project will be beneficial in terms of finance for the organization or not.
4. **Legal Feasibility:** In legal feasibility, the project is ensured to comply with all relevant laws, regulations, and standards. It identifies any legal constraints that could impact the project and reviews existing contracts and agreements to assess their effect on the project's execution. Additionally, legal feasibility considers issues related to intellectual property, such as patents and copyrights, to safeguard the project's innovation and originality.
5. **Schedule Feasibility:** In schedule feasibility, the project timeline is evaluated to determine if it is realistic and achievable. Significant milestones are identified, and deadlines are established to track progress

effectively. Resource availability is assessed to ensure that the necessary resources are accessible to meet the project schedule. Furthermore, any time constraints that might affect project delivery are considered to ensure timely completion. This focus on schedule feasibility is crucial for the successful planning and execution of a project.

2. Requirements Elicitation

It is related to the various ways used to gain knowledge about the project domain and requirements. The various sources of domain knowledge include customers, business manuals, the existing software of the same type, standards, and other stakeholders of the project. The techniques used for requirements elicitation include interviews, brainstorming, task analysis, Delphi technique, prototyping, etc. Some of these are discussed here. Elicitation does not produce formal models of the requirements understood. Instead, it widens the domain knowledge of the analyst and thus helps in providing input to the next stage.

Requirements elicitation is the process of gathering information about the needs and expectations of stakeholders for a software system. This is the first step in the requirements engineering process and it is critical to the success of the software development project. The goal of this step is to understand the problem that the software system is intended to solve and the needs and expectations of the stakeholders who will use the system.

Several techniques can be used to elicit requirements, including:

- **Interviews**: These are one-on-one conversations with stakeholders to gather information about their needs and expectations.
- **Surveys**: These are questionnaires that are distributed to stakeholders to gather information about their needs and expectations.
- **Focus Groups**: These are small groups of stakeholders who are brought together to discuss their needs and expectations for the software system.
- **Observation**: This technique involves observing the stakeholders in their work environment to gather information about their needs and expectations.
- **Prototyping**: This technique involves creating a working model of the software system, which can be used to gather feedback from stakeholders and to validate requirements.

It's important to document, organize, and prioritize the requirements obtained from all these techniques to ensure that they are complete, consistent, and accurate.

3. Requirements Specification

This activity is used to produce formal software requirement models. All the requirements including the functional as well as the non-functional requirements and the constraints are specified by these models in totality. During specification, more knowledge about the problem may be required which can again trigger the elicitation process. The models used at this stage include ER diagrams, data flow diagrams(DFDs), function decomposition diagrams(FDDs), data dictionaries, etc.

Requirements specification is the process of documenting the requirements identified in the analysis step in a clear, consistent, and unambiguous manner. This step also involves prioritizing and grouping the requirements into manageable chunks.

The goal of this step is to create a clear and comprehensive document that describes the requirements for the software system. This document should be understandable by both the development team and the stakeholders.

Several types of requirements are commonly specified in this step, including

1. **Functional Requirements:** These describe what the software system should do. They specify the functionality that the system must provide, such as input validation, data storage, and user interface.
2. **Non-Functional Requirements**: These describe how well the software system should do it. They specify the quality attributes of the system, such as performance, reliability, usability, and security.
3. **Constraints:** These describe any limitations or restrictions that must be considered when developing the software system.
4. **Acceptance Criteria**: These describe the conditions that must be met for the software system to be considered complete and ready for release.

To make the requirements specification clear, the requirements should be written in a natural language and use simple terms, avoiding technical jargon, and using a consistent format throughout the document. It is also important to use diagrams, models, and other visual aids to help communicate the requirements effectively.

Once the requirements are specified, they must be reviewed and validated by the stakeholders and development team to ensure that they are complete, consistent, and accurate.

4. Requirements Verification and Validation

Verification: It refers to the set of tasks that ensures that the software correctly implements a specific function.

Validation: It refers to a different set of tasks that ensures that the software that has been built is traceable to customer requirements. If requirements are not validated, errors in the requirement definitions would propagate to the successive stages resulting in a lot of modification and rework. The main steps for this process include:

1. The requirements should be consistent with all the other requirements i.e. no two requirements should conflict with each other.
2. The requirements should be complete in every sense.
3. The requirements should be practically achievable.

Reviews, buddy checks, making test cases, etc. are some of the methods used for this.

Requirements verification and validation (V&V) is the process of checking that the requirements for a software system are complete, consistent, and accurate and that they meet the needs and expectations of the stakeholders. The goal of V&V is to ensure that the software system being developed meets the requirements and that it is developed on time, within budget, and to the required quality.

1. Verification is checking that the requirements are complete, consistent, and accurate. It involves reviewing the requirements to ensure that they are clear, testable, and free of errors and inconsistencies. This can include reviewing the requirements document, models, and diagrams, and holding meetings and walkthroughs with stakeholders.
2. Validation is the process of checking that the requirements meet the needs and expectations of the stakeholders. It involves testing the requirements to ensure that they are valid and that the software system being developed will meet the needs of the stakeholders. This can include testing the software system through simulation, testing with prototypes, and testing with the final version of the software.

3. Verification and Validation is an iterative process that occurs throughout the software development life cycle. It is important to involve stakeholders and the development team in the V&V process to ensure that the requirements are thoroughly reviewed and tested.

It's important to note that V&V is not a one-time process, but it should be integrated and continue throughout the software development process and even in the maintenance stage.

5. Requirements Management

Requirement management is the process of analyzing, documenting, tracking, prioritizing, and agreeing on the requirement and controlling the communication with relevant stakeholders. This stage takes care of the changing nature of requirements. It should be ensured that the SRS is as modifiable as possible to incorporate changes in requirements specified by the end users at later stages too. Modifying the software as per requirements in a systematic and controlled manner is an extremely important part of the requirements engineering process.

Requirements management is the process of managing the requirements throughout the software development life cycle, including tracking and controlling changes, and ensuring that the requirements are still valid and relevant. The goal of requirements management is to ensure that the software system being developed meets the needs and expectations of the stakeholders and that it is developed on time, within budget, and to the required quality.

Several key activities are involved in requirements management, including:

1. **Tracking and controlling changes:** This involves monitoring and controlling changes to the requirements throughout the development process, including identifying the source of the change, assessing the impact of the change, and approving or rejecting the change.
2. **Version control**: This involves keeping track of different versions of the requirements document and other related artifacts.
3. **Traceability**: This involves linking the requirements to other elements of the development process, such as design, testing, and validation.
4. **Communication:** This involves ensuring that the requirements are communicated effectively to all stakeholders and that any changes or issues are addressed promptly.
5. **Monitoring and reporting**: This involves monitoring the progress of the development process and reporting on the status of the requirements.

Requirements management is a critical step in the software development life cycle as it helps to ensure that the software system being developed meets the needs and expectations of stakeholders and that it is developed on time, within budget, and to the required quality. It also helps to prevent scope creep and to ensure that the requirements are aligned with the project goals.

Tools Involved in Requirement Engineering

- Observation report
- Questionnaire (survey, poll)
- Use cases
- User stories
- Requirement workshop
- Mind mapping
- Roleplaying
- Prototyping

Advantages of Requirements Engineering Process

- Helps ensure that the software being developed meets the needs and expectations of the stakeholders
- Can help identify potential issues or problems early in the development process, allowing for adjustments to be made before significant
- Helps ensure that the software is developed in a cost-effective and efficient manner
- Can improve communication and collaboration between the development team and stakeholders
- Helps to ensure that the software system meets the needs of all stakeholders.
- Provides an unambiguous description of the requirements, which helps to reduce misunderstandings and errors.
- Helps to identify potential conflicts and contradictions in the requirements, which can be resolved before the software development process begins.

- Helps to ensure that the software system is delivered on time, within budget, and to the required quality standards.
- Provides a solid foundation for the development process, which helps to reduce the risk of failure.

Disadvantages of Requirements Engineering Process

- Can be time-consuming and costly, particularly if the requirements-gathering process is not well-managed
- Can be difficult to ensure that all stakeholders' needs and expectations are taken into account
- It Can be challenging to ensure that the requirements are clear, consistent, and complete
- Changes in requirements can lead to delays and increased costs in the development process.
- As a best practice, Requirements engineering should be flexible, adaptable, and should be aligned with the overall project goals.
- It can be time-consuming and expensive, especially if the requirements are complex.
- It can be difficult to elicit requirements from stakeholders who have different needs and priorities.
- Requirements may change over time, which can result in delays and additional costs.
- There may be conflicts between stakeholders, which can be difficult to resolve.
- It may be challenging to ensure that all stakeholders understand and agree on the requirements.

Information Modelling

Information Modelling in Software Engineering is needed because it helps us understand, organize, and represent data and its relationships before building a software system.
Here are the main reasons why we need it:

1. Clear Representation of Information
   - Converts complex real-world data into a structured, understandable model.
   - Shows entities, attributes, and relationships (e.g., customer–orders–products).

---

2. Bridge Between Users and Developers
   - Provides a common language between stakeholders and technical teams.
   - Helps users validate whether the model reflects their business needs.

---

3. Reduce Ambiguity and Errors
   - Avoids confusion in requirements by formally defining data.
   - Ensures that every data element has a clear meaning and purpose.

---

4. Support for Database and System Design
   - Acts as the foundation for designing databases, file structures, and APIs.
   - Example: Entity-Relationship (ER) models guide the schema of relational databases.

---

5. Consistency and Reusability
   - Promotes data consistency across different modules of the system.
   - Encourages reuse of well-defined data structures in future projects.

---

6. Improve Quality and Maintainability

- With a structured information model, systems are easier to extend, debug, and maintain.
- Enhances system quality by ensuring correct, complete, and non-redundant data.

Types of Information Models

There are several types of information models, each serving different purposes and levels of abstraction:

1. Conceptual Data Model: This high-level model focuses on abstract business concepts and structures. It is used to define and organize business problems, rules, and concepts without delving into technical details

2. Logical Data Model: This model provides a detailed representation of the data at a logical level, including tables, columns, relationships, and constraints. It serves as a blueprint for the physical design of databases

3. Physical Data Model: This model describes the implementation details of the data model in a specific database system, including tables, columns, primary keys, foreign keys, and other constraints

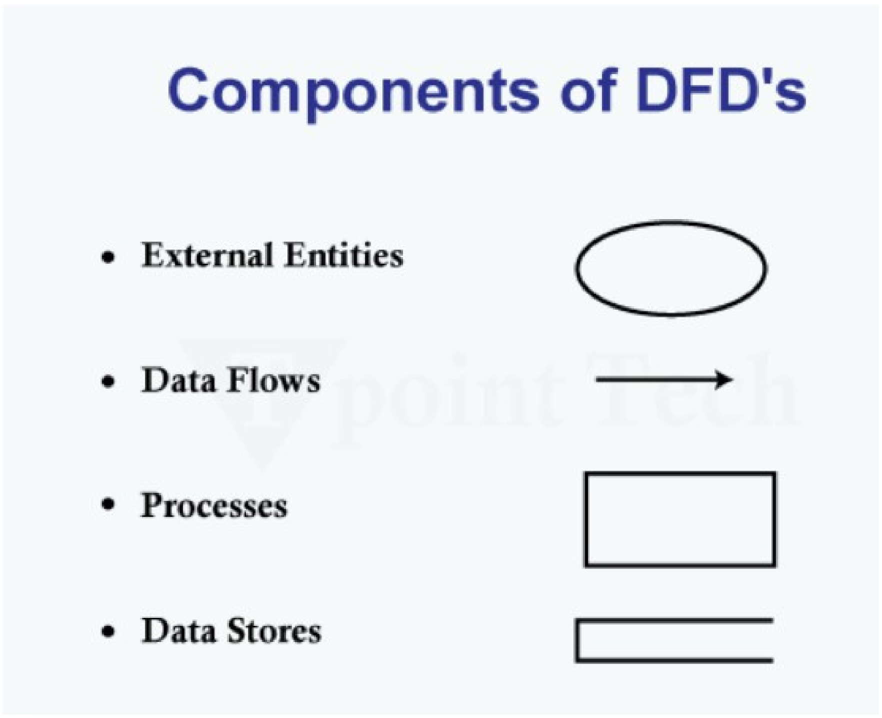
Data Flow Diagram

Data Flow Diagram is a visual representation of the flow of data within the system. It help to understand the flow of data throughout the system, from input to output, and how it gets transformed along the way. The models enable software engineers, customers, and users to work together effectively during the analysis and specification of requirements.

Data flow Diagram Components

A DFD consists of the following four major parts:

1. External entities
2. Processes
3. Data stores
4. Data flows



Characteristics of Data Flow Diagram (DFD)

Below are some characteristics of Data Flow Diagram (DFD):

1. Graphical Representation: Data Flow Diagram (DFD) use different symbols and notation to represent data flow within system. That simplify the complex system into understandable visual elements. This makes them easier to interpret by both **technical** and **non-technical stakeholders**.

2. Problem Analysis: Data Flow Diagram **(**DFDs) are very useful in understanding a system and can be effectively used during analysis. Data Flow Diagram (DFDs) are quite general and are not limited to problem analysis for software requirements specification.

3. Abstraction: DFDs abstract away the **implementation details** and focus on the **data flow and processes** within a system. They provide a high-level overview and omit unnecessary technical information.

4. **Hierarchy**: Data Flow Diagram (DFD) provides a hierarchy of a system. High- level diagram i.e. 0-level diagram provides an overview of entire system while lower-level diagram like 1-level DFD and beyond provides a detailed data flow of individual process.

Levels in Data Flow Diagram (DFD)

DFDs are categorized into various levels, with each level providing different degrees of detail. The levels are numbered from **0** and onward. The higher the level, the more detailed the diagram becomes. The following are the four levels of DFDs:

0-Level Data Flow Diagram (DFD)

Level 0 DFD is the **highest-level diagram**, representing the system as a **single process** with its interactions with external entities. It shows the major processes, data flows, and data stores in the system, without providing any details about the internal workings of these processes. It is also known as the **Context Diagram**, which abstracts the system's operations and shows how data enters and leaves the system.

1-Level Data Flow Diagram (DFD)

Level 1 DFD provides a more detailed view of the system by breaking down the major processes identified in the level 0 Data Flow Diagram (DFD) into sub-processes. Each sub-process is depicted as a separate process on the level 1 Data Flow Diagram (DFD). The data flows and data stores associated with each sub-process are also shown.

Level 1 DFD provides a more **detailed view** of the system, focusing on key functional aspects. The **Context Diagram** from **Level 0** is expanded into **multiple bubbles/processes**.

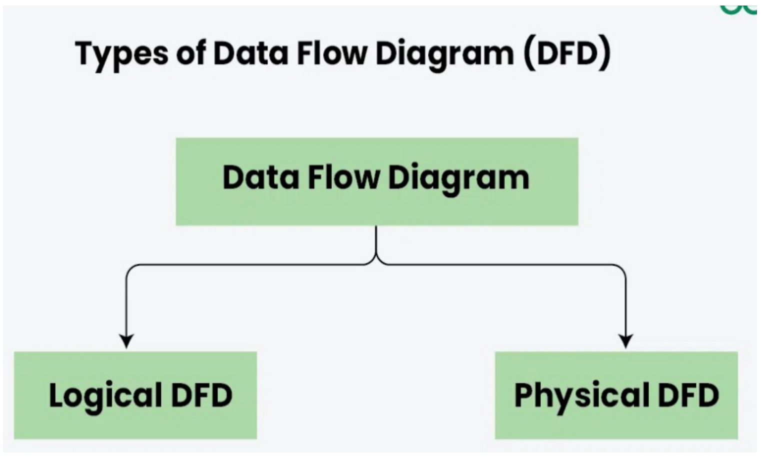
2-Level Data Flow Diagram (DFD)

Level 2 DFD further breaks down the sub-processes from **Level 1 DFD** into **additional sub-processes**, providing an even more **detailed view**. This level is useful when dealing with **specific requirements** or parts of the system that need a closer examination of their processes and interactions.

3-Level Data Flow Diagram (DFD)

3-Level is the **most detailed level** of Data Flow Diagram (DFDs), which provides a detailed view of the processes, data flows, and data stores in the system. This level is typically used for complex systems, where a high level of detail is required to understand the system. It includes **detailed descriptions** of each process, data flow, and data store, and is usually used when there is a need for a comprehensive understanding of the system.

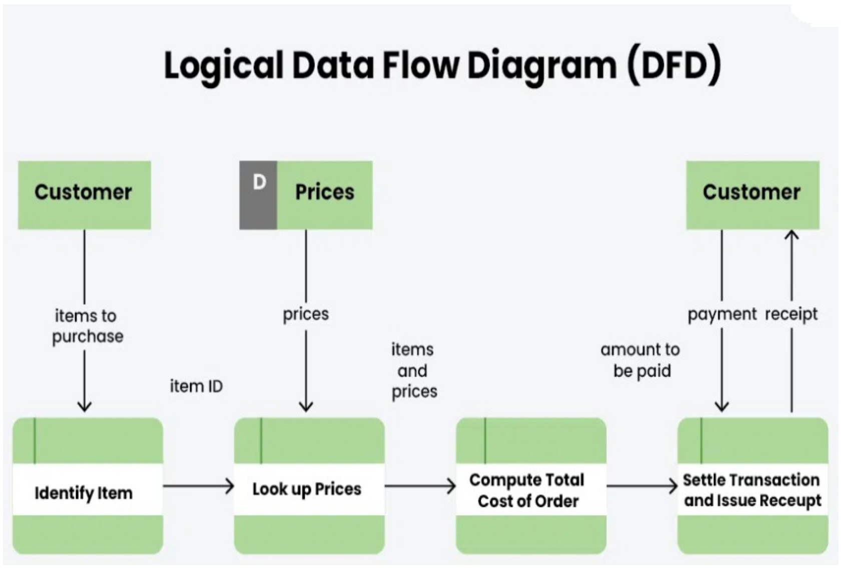
Types of Data Flow Diagram (DFD)
DFDs can be classified into two main types, each focusing on a different perspective of system design:



1. Logical Data Flow Diagram (DFD)

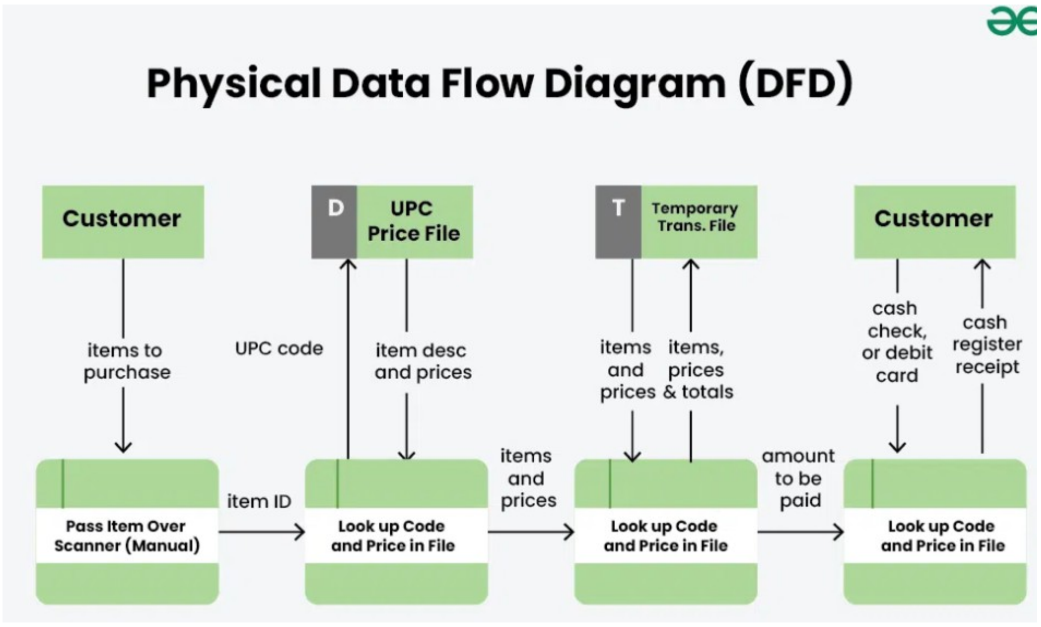
The **Logical Data Flow Diagram** mainly focuses on the **system process**. It illustrates how data flows in the system. Logical Data Flow Diagram (DFD) mainly focuses on high level processes and data flow without diving deep into technical implementation details.

Logical DFDs is used in various organizations for the smooth running of system. Like in a Banking software system, it is used to describe how data is moved from one entity to another.



2. Physical Data Flow Diagram

Physical data flow diagram shows **how the data flow is actually implemented** in the system. In the Physical Data Flow Diagram (DFD), we include additional details such as data storage, data transmission, and specific technology or system components. **Physical DFDs** are more detailed and provide a closer look at the **actual implementation** of the system, including the hardware, software, and physical aspects of data processing.



Introduction of ER Model

The Entity-Relationship Model (ER Model) is a conceptual model for designing a databases. This model represents the logical structure of a database, including entities, their attributes and relationships between them.

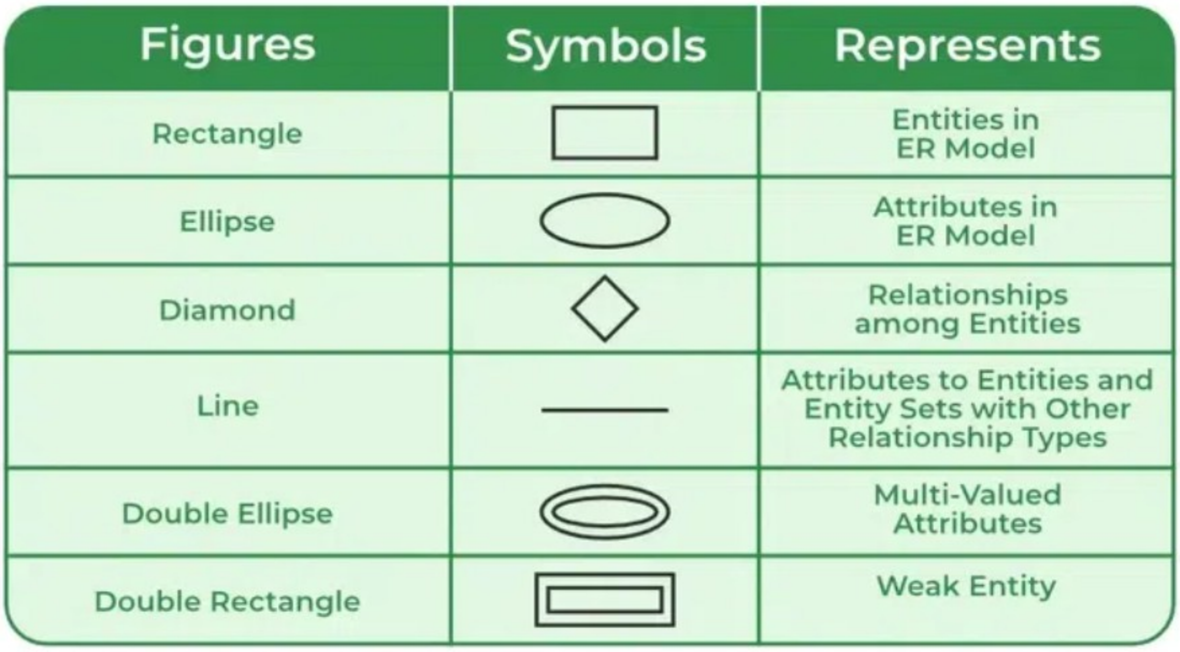
- **Entity:** An objects that is stored as data such as *Student*, *Course* or *Company*.
- **Attribute:** Properties that describes an entity such as *StudentID*, *CourseName*, or *EmployeeEmail*.
- **Relationship:** A connection between entities such as "a *Student* enrolls in a *Course*".

Symbols Used in ER Model

ER Model is used to model the logical view of the system from a data perspective which consists of these symbols:

- **Rectangles:** Rectangles represent entities in the ER Model.
- **Ellipses:** Ellipses represent attributes in the ER Model.
- **Diamond:** Diamonds represent relationships among Entities.

- **Lines:** Lines represent attributes to entities and entity sets with other relationship types.
- **Double Ellipse:** Double ellipses represent multi-valued Attributes, such as a student's multiple phone numbers
- **Double Rectangle:** Represents weak entities, which depend on other entities for identification.



Figures	Symbols	Represents
Rectangle	▭	Entities in ER Model
Ellipse	⬭	Attributes in ER Model
Diamond	◇	Relationships among Entities
Line	—	Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse	◎	Multi-Valued Attributes
Double Rectangle	⧈	Weak Entity

What is an Entity?

An Entity represents a real-world object, concept or thing about which data is stored in a database. It act as a building block of a database. Tables in relational database represent these entities.

Example of entities:

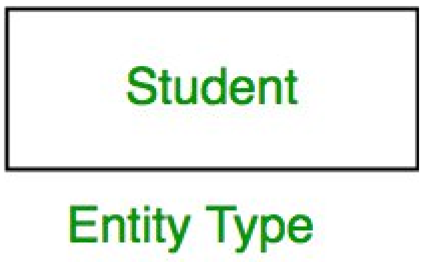
- **Real-World Objects:** Person, Car, Employee etc.
- **Concepts:** Course, Event, Reservation etc.
- **Things:** Product, Document, Device etc.

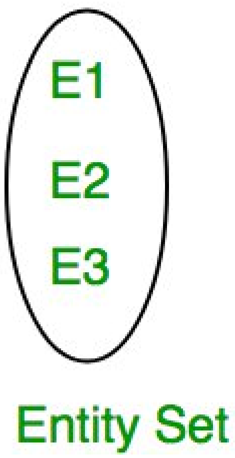
The entity type defines the structure of an entity, while individual instances of that type represent specific entities.

What is an Entity Set?

An entity refers to an individual object of an entity type, and the collection of all entities of a particular type is called an entity set. For example, E1 is an entity that belongs to the entity type "Student," and the group of all students forms the entity set.

In the ER diagram below, the entity type is represented as:





Entity Set

We can represent the entity sets in an ER Diagram but we can't represent individual entities because an entity is like a row in a table, and an ER diagram shows the structure and relationships of data, not specific data entries (like rows and columns). An ER diagram is a visual representation of the data model, not the actual data itself.

Types of Entity

There are two main types of entities:

1. Strong Entity

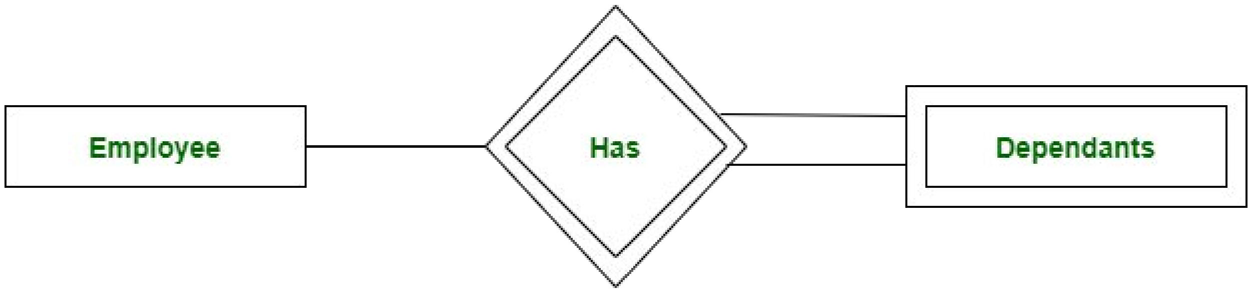
A Strong Entity is a type of entity that has a key Attribute that can uniquely identify each instance of the entity. A Strong Entity does not depend on any other Entity in the Schema for its identification. It has a primary key that ensures its uniqueness and is represented by a rectangle in an ER diagram.

2. Weak Entity

A Weak Entity cannot be uniquely identified by its own attributes alone. It depends on a strong entity to be identified. A weak entity is associated with an identifying entity (strong entity), which helps in its identification. A weak entity are represented by a double rectangle. The participation of weak entity types is always total. The relationship between the weak entity type and its identifying strong entity type is called identifying relationship and it is represented by a double diamond.

Example:

A company may store the information of dependents (Parents, Children, Spouse) of an Employee. But the dependents can't exist without the employee. So dependent will be a Weak Entity Type and Employee will be identifying entity type for dependent, which means it is Strong Entity Type.



Strong Entity and Weak Entity

Attributes in ER Model

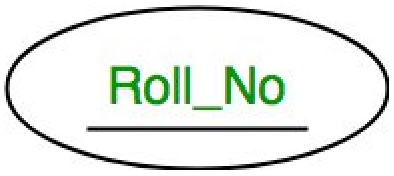
Attributes are the properties that define the entity type. For example, for a Student entity Roll_No, Name, DOB, Age, Address, and Mobile_No are the attributes that define entity type Student. In ER diagram, the attribute is represented by an oval.

Attribute

Types of Attributes

1. Key Attribute

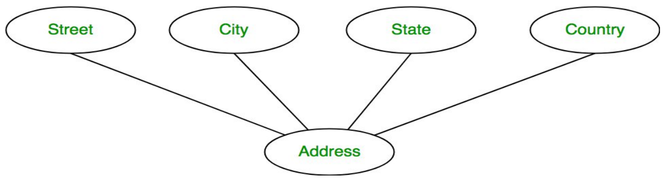
The attribute which uniquely identifies each entity in the entity set is called the key attribute. For example, Roll_No will be unique for each student. In ER diagram, the key attribute is represented by an oval with an underline.



Key Attribute

2. Composite Attribute

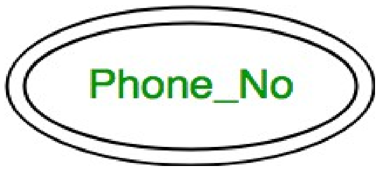
An attribute composed of many other attributes is called a composite attribute. For example, the Address attribute of the student Entity type consists of Street, City, State, and Country. In ER diagram, the composite attribute is represented by an oval comprising of ovals.



Composite Attribute

3. Multivalued Attribute

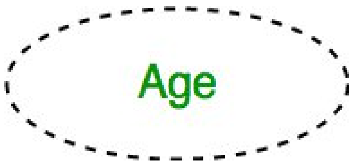
An attribute consisting of more than one value for a given entity. For example, Phone_No (can be more than one for a given student). In ER diagram, a multivalued attribute is represented by a double oval.



Multivalued Attribute

4. Derived Attribute

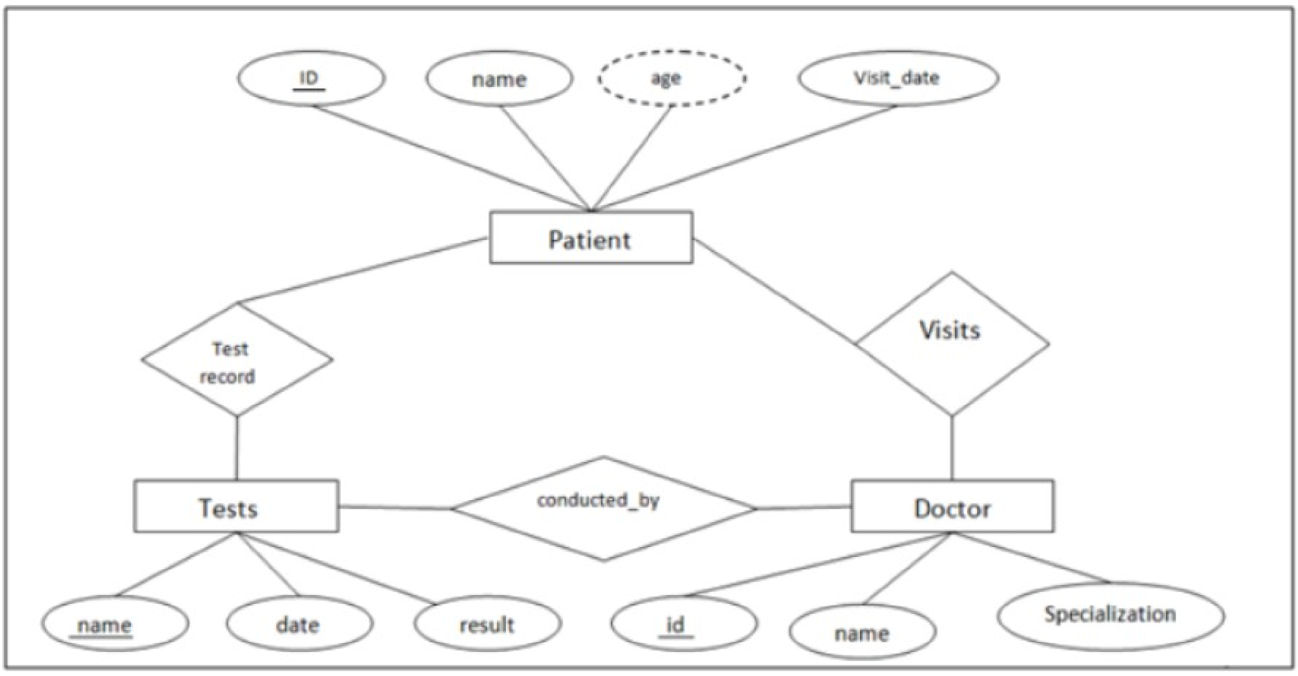
An attribute that can be derived from other attributes of the entity type is known as a derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, the derived attribute is represented by a dashed oval.



Derived Attribute

The Complete Entity Type Student with its Attributes can be represented as:

Hospital ER Model



This is an ER model of a Hospital. The entities are represented in rectangular boxes and are Patient, Tests and Doctor.

Each of these entities have their respective attributes which are –

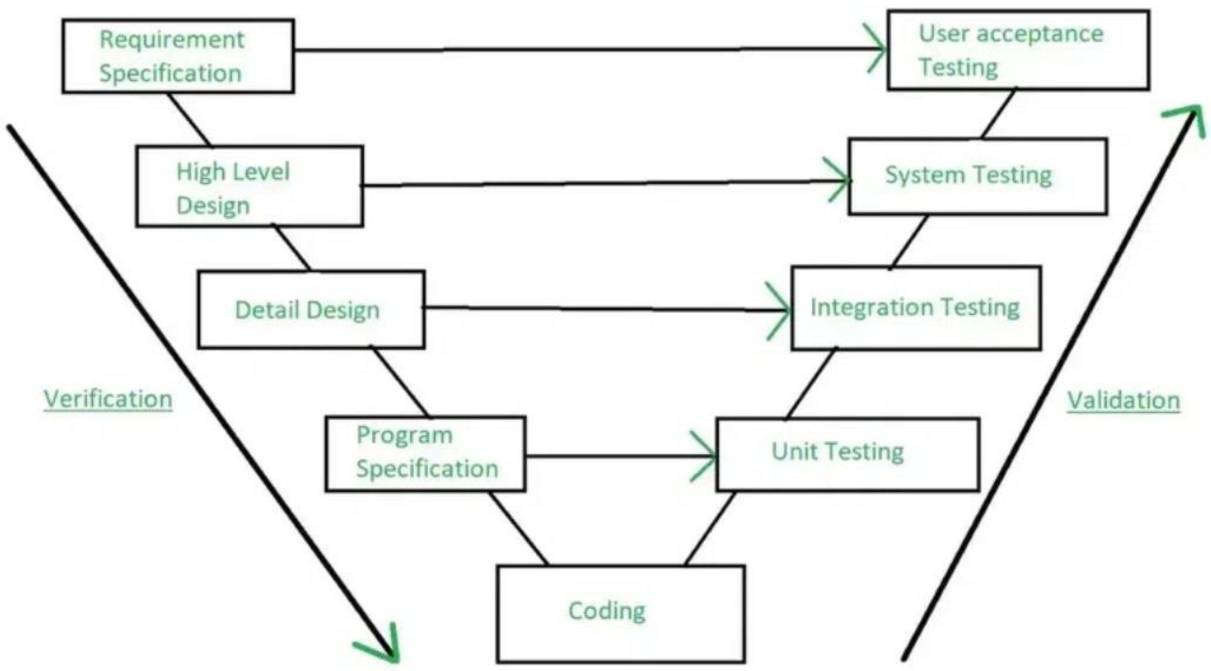
- **Patients** - ID(primary key), name, age,visit_date
- **Tests**- Name(primary key), date, result
- **Doctor**- ID(primary key), name, specialization

Verification and Validation in Software Engineering

Verification and Validation are the processes of investigating whether a software system satisfies specifications and standards and fulfills the required purpose. Verification and Validation both play an important role in developing good software. Verification helps in examining whether the product is built right according to requirements, while validation helps in examining whether the right product is built to meet user needs.

What is Verification?

Verification is the process of checking that software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have. Verification is static testing. Verification means **Are we building the product right?**



Verification and Validation

Static Testing

Verification Testing is known as Static Testing, and it can be simply termed as checking whether we are developing the right product or not, and also whether our software is fulfilling the customer's requirement or not. Here are some of the activities that are involved in verification.

- **Inspections**: This is when we carefully look over the software's design or code to spot any potential problems or areas that could be improved.
- **Reviews**: This is a team effort, where everyone gets together to assess the software's requirements, design, or code. The goal is to make sure everything is on track and meets the specified objectives.
- **Walkthroughs**: In a walkthrough, the developer or designer gives a casual presentation of their work, explaining how it meets the project's requirements. It's a great way to get everyone on the same page.
- **Desk-Checking**: This is when a developer manually reviews their own code or design to catch any mistakes or issues before moving forward.

Verification and Validation

What is Validation?

Validation is the process of checking whether the **Software Product** is up to the mark, or in other words product has high-level requirements. It is the process of checking the validation of the product i.e., it checks whether what we are developing is the right product. It is a validation of the actual and expected products. Validation is dynamic testing. Validation means **Are we building the right product?**

Dynamic Testing

Validation Testing is known as **Dynamic Testing** in which we examine whether we have developed the product right or not and also about the business needs of the client. Here are some of the activities that are involved in Validation.

- **Black Box Testing**: This is all about testing the software from the user's perspective. You don't need to know how the code works internally. Instead, you just focus on whether the software behaves as expected when you use it, like clicking buttons or entering information.
- **White Box Testing**: In this case, you have access to the internal workings of the software. You're testing the logic, the flow, and how the code is structured to make sure everything is functioning as it should behind the scenes.
- **Unit Testing**: This involves testing individual parts or functions of the software to make sure each piece works properly on its own, before they're combined with other parts of the system.
- **Integration Testing**: Once individual pieces are tested, integration testing checks how well different parts of the software work together. It's like making sure all the components play nicely when combined.

Example Scenario of Verification and Validation

Verification: During development, the team takes a step back to check the system's design and code. They focus on key features like user registration, the shopping cart, and payment processing. Through inspections and walkthroughs, they make sure each part of the software is being built according to the initial plan and specifications.

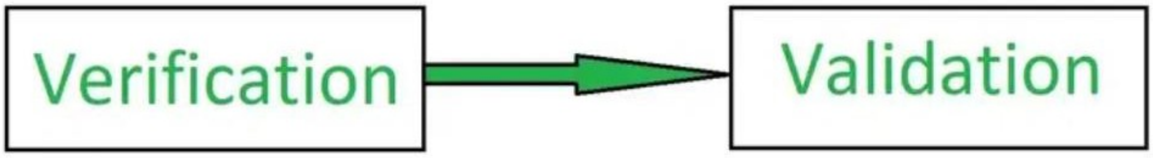
Validation: Once the software is finished, it's time to test it. First, the team runs **unit tests** to check individual features, like making sure the shopping cart calculates prices correctly. Then, **integration tests** ensure all features work well together. After that, **system testing** checks if everything functions as a whole. Finally, the client does **acceptance testing** to confirm that the software meets their needs and expectations.

Differences between Verification and Validation

Here is the **Differences Between Verification and Validation**.

	Verification	**Validation**
Definition	Verification refers to the set of activities that ensure software correctly implements the specific function	Validation refers to the set of activities that ensure that the software that has been built is traceable to customer requirements.
Focus	It includes checking documents, designs, codes, and programs.	It includes testing and validating the actual product.
Type of Testing	Verification is the **Static testing**.	Validation is **Dynamic testing.**
Execution	It does *not* include the execution of the code.	It includes the execution of the code.
Methods Used	Methods used in verification are reviews, walkthroughs, inspections and desk-checking.	Methods used in validation are **Black Box Testing**, **White Box Testing** and **Non- Functional testing**.

Note: Verification is followed by Validation.



verification and validation

Verification and validation

Why Verification and Validation Important?

Both verification and validation are key to delivering high-quality software, and while they focus on different things, they work together to make sure your product is as per requirements. Here's the reason why its important:

1. **Verifying Correctness**: Verification helps catch mistakes early in development. It ensures that you're building the software the right way, according to the original plan and specifications.
2. **Meeting User Needs**: Validation is about making sure the final product actually solves the problem for the user. It checks that the software delivers what the customer wants and performs as expected in real-world conditions.
3. **Preventing Bugs and Issues**: Verification helps find bugs or errors before the software even gets tested. That means you can fix problems early on. Validation, on the other hand, ensures the product works well in real-world use, preventing surprises later.
4. **Improving Efficiency**: By using both verification and validation, teams can catch issues sooner and reduce the time spent on bug fixes later in the process. This makes development more efficient and helps release products faster.

Conclusion

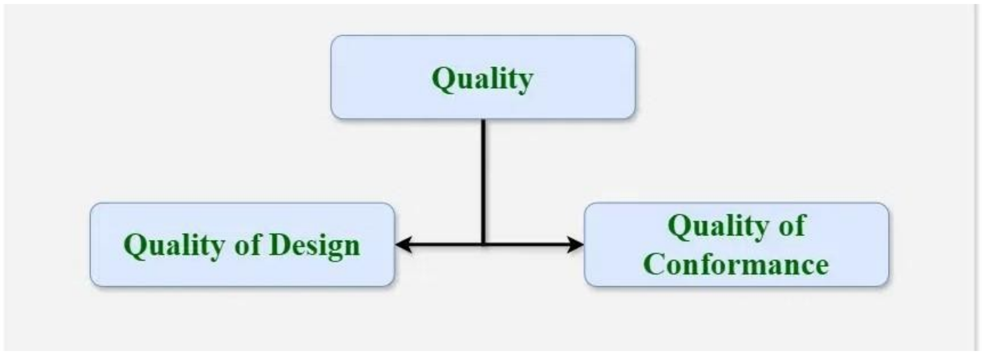
Verification and validation are essential processes in software development that ensure the quality and effectiveness of a software product. Verification checks if the software is built correctly according to specifications, while validation ensures the software meets user needs and performs well in real- world conditions. Together, they help identify and fix issues early, improving the reliability and user satisfaction of the final product.

Software Quality Assurance - Software Engineering

Software Quality Assurance (SQA) is simply a way to assure quality in the software. It is the set of activities that ensure processes, procedures as well as standards are suitable for the project and implemented correctly. It is a process that works parallel to Software Development. It focuses on improving the process of development of software so that problems can be prevented before they become major issues. Software Quality Assurance is a kind of Umbrella activity that is applied throughout the software process.
For those looking to deepen their expertise in SQA and elevate their professional skills, consider exploring a specialized training program **Manual to Automation Testing: A QA Engineer's Guide** . This program offers practical, hands-on experience and advanced knowledge that complements the concepts covered in this guide.

Quality

Quality in a product or service can be defined by several measurable characteristics. Each of these characteristics plays a crucial role in determining the overall quality.
Generally, the quality of the software is verified by third-party organizations like international standard organizations .



Quality in Software Engineering

Elements of Software Quality Assurance (SQA)

- **Standards:** The IEEE, ISO, and other standards organizations have produced a broad array of software engineering standards and related documents. The job of SQA is to ensure that standards that have been adopted are followed and that all work products conform to them.
- **Reviews and audits:** Technical reviews are a quality control activity performed by software engineers for software engineers. Their intent is to uncover errors. Audits are a type of review performed by SQA personnel (people employed in an organization) with the intent of ensuring that quality guidelines are being followed for software engineering work.
- **Testing:** Software testing is a quality control function that has one primary goal to find errors. The job of SQA is to ensure that testing is properly planned and efficiently conducted for primary goal of software.

- **Error/defect collection and analysis** : SQA collects and analyzes error and defect data to better understand how errors are introduced and what software engineering activities are best suited to eliminating them.
- **Change management:** SQA ensures that adequate change management practices have been instituted.
- **Education:** Every software organization wants to improve its software engineering practices. A key contributor to improvement is education of software engineers, their managers, and other stakeholders. The SQA organization takes the lead in software process improvement which is key proponent and sponsor of educational programs.
- **Security management:** SQA ensures that appropriate process and technology are used to achieve software security.
- **Safety:** SQA may be responsible for assessing the impact of software failure and for initiating those steps required to reduce risk.
- **Risk management:** The SQA organization ensures that risk management activities are properly conducted and that risk-related contingency plans have been established.

Focus of Software Quality Assurance (SQA)

The Software Quality Assurance (SQA) focuses on the following:
Software Quality Assurance (SQA)

- **Software's Portability:** Software's portability refers to its ability to be easily transferred or adapted to different environments or platforms without needing significant modifications. This ensures that the software can run efficiently across various systems, enhancing its accessibility and flexibility.
- **Software's Usability:** Usability of software refers to how easy and intuitive it is for users to interact with and navigate through the application. A high level of usability ensures that users can effectively accomplish their tasks with minimal confusion or frustration, leading to a positive user experience.
- **Software's Reusability:** Reusability in software development involves designing components or modules that can be reused in multiple parts of the software or in different projects. This promotes efficiency and reduces development time by eliminating the need to reinvent the wheel for similar functionalities, enhancing productivity and maintainability.
- **Software's Correctness:** Correctness of software refers to its ability to produce the desired results under specific conditions or inputs. Correct software behaves as expected without errors or unexpected behaviors, meeting the requirements and specifications defined for its functionality.
- **Software's Maintainability:** Maintainability of software refers to how easily it can be modified, updated, or extended over time. Well maintained software is structured and documented in a way that allows developers to make changes efficiently without introducing errors or compromising its stability.
- **Software's Error Control:** Error control in software involves implementing mechanisms to detect, handle, and recover from errors or unexpected situations gracefully. Effective error
control ensures that the software remains robust and reliable, minimizing disruptions to users and providing a smoother experience overall.

Major Activities in Software Quality Assurance (SQA)

- **SQA Management Plan:** Make a plan for how you will carry out the SQA throughout the project. Think about which set of software engineering activities are the best for project. check level of SQA team skills.
- **Set The Check Points:** SQA team should set checkpoints. Evaluate the performance

of the project on the basis of collected data on different check points.

- **Measure Change Impact:** The changes for making the correction of an error sometimes re introduces more errors keep the measure of impact of change on project. Reset the new change to check the compatibility of this fix with whole project.
- **Multi testing Strategy:** Do not depend on a single testing approach. When you have a lot of testing approaches available use them.
- **Manage Good Relations:** In the working environment managing good relations with other teams involved in the project development is mandatory. Bad relation of SQA team with programmers team will impact directly and badly on project.
- **Maintaining records and reports:** Comprehensively document and share all QA records, including test cases, defects, changes, and cycles, for stakeholder awareness and future reference.
- **Reviews software engineering activities:** The SQA group identifies and documents the processes. The group also verifies the correctness of software product.
- **Formalize deviation handling:** Track and document software deviations meticulously. Follows established procedures for handling variances.

Benefits of Software Quality Assurance (SQA)

- SQA produces high quality software.
- High quality application saves time and cost.
- SQA is beneficial for better reliability.
- SQA is beneficial in the condition of no maintenance for a long time.
- High quality commercial software increase market share of company.
- Improving the process of creating software.
- Improves the quality of the software.
- It cuts maintenance costs. Get the release right the first time, and your company can forget about it and move on to the next big thing. Release a product with chronic issues, and your business bogs down in a costly, time consuming, never-ending cycle of repairs.

Disadvantage of Software Quality Assurance (SQA)

There are a number of disadvantages of quality assurance.**Cost:** Some of them include adding more resources, which cause the more budget its not, addition of more resources for betterment of the product.

- **Time Consuming:** Testing and Deployment of the project taking more time which cause delay in the project.
- **Overhead** : SQA processes can introduce administrative overhead, requiring documentation, reporting, and tracking of quality metrics. This additional administrative burden can sometimes outweigh the benefits, especially for smaller projects.
- **Resource Intensive** : SQA requires skilled personnel with expertise in testing methodologies, tools, and quality assurance practices. Acquiring and retaining such talent can be challenging and expensive.
- **Resistance to Change** : Some team members may resist the implementation of SQA processes, viewing them as bureaucratic or unnecessary. This resistance can hinder the adoption and effectiveness of quality assurance practices within an organization.
- **Not Foolproof** : Despite thorough testing and quality assurance efforts, software can still contain defects or vulnerabilities. SQA cannot guarantee the elimination of all bugs or issues in software products.
- **Complexity** : SQA processes can be complex, especially in large-scale projects with multiple stakeholders, dependencies, and integration points. Managing the complexity of quality assurance activities requires careful planning and coordination.

What is a Software Quality Assurance Plan?

A Quality Assurance Plan (QAP) is a document or set of documents that outlines the systematic processes, procedures, and standards for ensuring the quality of a product or service. It is a key component of quality management and is used in various industries to establish and maintain a consistent level of quality in deliverables. For a software product or service, an SQA plan will be used in conjunction with the typical development, prototyping, design, production, and release cycle. An SQA plan will include several components, such as purpose, references, configuration and management, tools, code controls, testing methodology, problem reporting and remedial measures, and more, for easy documentation and referencing.

SQA-Plan

Importance of Software Quality Assurance Plan

- **Quality Standards and Guidelines:** The SQA Plan lays out the requirements and guidelines to make sure the programme satisfies predetermined standards for quality.
- **Risk management:** It is the process of recognizing, evaluating and controlling risks in order to reduce the possibility of errors and other problems with quality.
- **Standardization and Consistency:** The strategy guarantees consistent methods, processes, and procedures, fostering a unified and well-structured approach to quality assurance.
- **Customer Satisfaction:** The SQA Plan helps to ensure that the finished product satisfies customer needs, which in turn increases overall customer satisfaction.
- **Resource optimization:** It is the process of defining roles, responsibilities, and procedures in order to maximize resource utilization and minimize needless rework.
- **Early Issue Detection:** SQA Plans help identify problems early on, which lowers the expense and work involved in fixing them.

Objectives And Goals of Software Quality Assurance Plan:

The objectives and goals of a Quality Assurance Plan (QAP) are to ensure that the products or services meet specified quality standards and requirements. The plan serves as a roadmap for implementing quality assurance processes throughout a project or organizational activity. The specific objectives and goals can vary depending on the nature of the project or industry, but common elements include:

Compliance with Standards and Regulations:

- **Objective:** Ensure that the project or product complies with relevant industry standards, regulatory requirements, and any other applicable guidelines.
- **Goal:** Achieve and maintain adherence to established quality standards to meet legal and regulatory obligations.

Customer Satisfaction:

- **Objective:** Enhance customer satisfaction by delivering products or services that meet or exceed customer expectations.
- **Goal:** Identify and prioritize customer requirements, and incorporate them into the quality assurance processes to create a positive customer experience.

Defect Prevention:

- **Objective:** Implement measures to prevent defects, errors, or issues in the early stages of the project lifecycle.
- **Goal:** Identify potential sources of defects, analyze root causes, and take proactive steps to eliminate or minimize the occurrence of defects.

Consistency and Reliability:

- **Objective:** Establish a consistent and reliable process for the development or delivery of products and services.
- **Goal:** Ensure that the quality of deliverables is consistent over time and across different phases of the project, promoting reliability and predictability.

Process Improvement:

- **Objective:** Continuously improve processes to enhance efficiency, effectiveness, and overall quality.
- **Goal:** Implement feedback mechanisms, conduct regular process assessments, and identify opportunities for improvement to optimize the quality assurance process.

Risk Management:

- **Objective:** Identify and manage risks that could impact the quality of the project or product.
- **Goal:** Develop strategies to assess, mitigate, and monitor risks that may affect the achievement of quality objectives.

Clear Roles and Responsibilities:

- **Objective:** Clearly define roles and responsibilities related to quality assurance activities.
- **Goal:** Ensure that team members understand their roles in maintaining and improving quality, fostering accountability and collaboration.

Documentation and Traceability:

- **Objective:** Establish a robust documentation process to track and trace quality-related activities and decisions.

- **Goal:** Create comprehensive records that enable transparency, accountability, and the ability to trace the development or production process.

Training and Competence:

- **Objective:** Ensure that team members are adequately trained and possess the necessary competencies to perform quality assurance tasks.
- **Goal:** Provide ongoing training to enhance the skills and knowledge of individuals involved in quality assurance.

Continuous Monitoring and Reporting:

- **Objective:** Monitor quality metrics and report on the status of quality assurance activities.
- **Goal:** Implement regular monitoring and reporting mechanisms to track progress, identify issues, and make data-driven decisions to maintain or improve quality.

By aligning the Quality Assurance Plan with these objectives and goals, organizations can establish a robust framework for consistently delivering high-quality products or services while adapting to changes and opportunities for improvement.

Steps to Develop Software Quality Assurance Plan:

Developing a Quality Assurance Plan (QAP) involves a systematic process to ensure that the plan effectively addresses the quality requirements of a project or process. Here are the steps to develop a Quality Assurance Plan:

Define Project Objectives and Scope:

- Clearly articulate the objectives and scope of the project or process for which the Quality Assurance Plan is being developed. Understand the goals, deliverables, and stakeholders involved.

Identify Quality Standards and Criteria:

- Determine the relevant quality standards, regulations, and criteria that are applicable to the project. This may include industry standards, legal requirements, and internal organizational standards.

Identify Stakeholders:

- Identify and involve key stakeholders who have an interest in or will be affected by the quality of the project or process. This includes project managers, team members, customers, and any regulatory bodies.

Define Roles and Responsibilities:

- Clearly define the roles and responsibilities of individuals or teams involved in quality assurance activities. This ensures accountability and clarity in the execution of quality- related tasks.

Conduct a Risk Assessment:

- Identify potential risks to the quality of the project. Assess the impact and likelihood of these risks and develop strategies for risk mitigation and management.

Establish Quality Assurance Activities:

- Determine the specific activities and tasks that will be carried out to ensure quality throughout the project lifecycle. This may include reviews, inspections, audits, testing, and other quality control measures.

Develop Testing and Inspection Procedures:

- If applicable, create detailed testing and inspection procedures. This includes developing test plans, test cases, and acceptance criteria to ensure that the product or service meets quality standards.

Document Processes and Procedures:

- Document the processes and procedures that will be followed to implement quality assurance activities. This documentation serves as a reference for team members and provides a basis for consistency.

Establish Documentation and Reporting Mechanisms:

- Define the documentation requirements for tracking and reporting quality-related information. This may include reports, checklists, and records of quality assessments.

Allocate Resources and Training:

- Specify the resources, tools, and training that will be provided to team members to support quality assurance efforts. Ensure that team members have the necessary skills and knowledge.

Define Change Control Processes:

- Establish a process for managing changes to the project or product to ensure that they do not negatively impact quality. This includes a change control board and a structured process for reviewing and approving changes.

Review and Approval:

- Seek input and feedback from relevant stakeholders on the draft Quality Assurance Plan. Revise the plan based on feedback and obtain formal approval before implementation.

Monitoring and Continuous Improvement:

- Continuously monitor quality metrics, track progress, and identify opportunities for improvement. Regularly review and update the Quality Assurance Plan to adapt to changes and enhance its effectiveness.

Communication and Training:

- Communicate the Quality Assurance Plan to all relevant stakeholders and provide training as necessary. Ensure that everyone involved understands their roles and responsibilities in maintaining and improving quality.

SQA implementation phase:

Developers work with the SQA team to construct a development plan before beginning to build an application. The Software Quality Assurance (SQA) team produces the Software Quality Assurance

plan, and the developers write the Software Development Life Cycle (SDLC) plan. If the documentation created by the developers and the SQA are well-written and structured, the application that is going to be developed is already 50% finished.
The SQA team's duty during this stage is to guarantee that the intended application's suggested features are implemented. The application will monitor not only the implementation process but also the development process. Certain design techniques, such language or framework, can be applied. In order to assist the development team in choosing the appropriate framework language.
The software's documentation is reviewed by the SQA team. The purpose of the application is stated in detail in the documentation.

Best Practices for Creating Software Quality Assurance Plan:

Creating a Software Quality Assurance (SQA) plan is essential to ensure the development of high- quality software. Here are some best practices to consider when creating a Software Quality Assurance Plan:

Understand Project Objectives:

- Clearly understand the objectives, goals, and scope of the software development project. This understanding is critical for tailoring the SQA plan to the specific needs of the project.

Define Quality Standards:

- Identify and document the relevant quality standards, guidelines, and best practices that the software must adhere to. This may include industry standards, regulatory requirements, and internal organizational standards.

Involve Stakeholders:

- Engage key stakeholders, including developers, testers, project managers, and customers, in the creation of the SQA plan. Ensure that their perspectives and requirements are considered.

Set Clear Roles and Responsibilities:

- Clearly define the roles and responsibilities of individuals or teams involved in SQA activities. This includes roles such as SQA manager, testers, developers, and project managers.

Establish Testing Processes:

- Define the testing processes, including types of testing (e.g., unit testing, integration testing, system testing, etc.), testing techniques, and tools to be used. Clearly articulate the criteria for success in each testing phase.

Incorporate Automated Testing:

- Integrate automated testing into the testing processes where applicable. Automated testing can improve efficiency, coverage, and the repeatability of tests, especially for repetitive and regression testing.

Document Test Plans and Cases:

- Develop comprehensive test plans and test cases that cover the functional and non- functional requirements of the software. Test documentation serves as a reference for testers and ensures comprehensive coverage.

Implement Code Reviews:

- Integrate code reviews as part of the development process. Code reviews help identify defects early in the development lifecycle, improving code quality and reducing the likelihood of defects reaching later stages.

Utilize Continuous Integration (CI) and Continuous Deployment (CD):

- Implement CI/CD practices to automate the integration and deployment processes. This ensures that changes are regularly and consistently tested, reducing the risk of integration issues and improving overall software quality.

Implement Configuration Management:

- Use configuration management practices to manage changes to the software, including version control, baseline management, and change control. This helps maintain consistency and traceability in software development.

Why is SQA Plan Important for Software Companies?

A Software Quality Assurance (SQA) plan is crucial for software companies for several reasons, as it helps ensure the delivery of high-quality software products and contributes to the overall success of the company. Here are some key reasons why an SQA plan is important for software companies:

- **Quality Assurance and Customer Satisfaction**: An SQA plan establishes processes and standards to ensure that software products meet or exceed customer expectations. High- quality software contributes to customer satisfaction, enhances the company's reputation, and fosters customer loyalty.
- **Compliance with Standards and Regulations**: Many industries have specific standards and regulations that software products must adhere to. An SQA plan helps ensure compliance with these standards, reducing the risk of legal issues and demonstrating the company's commitment to quality.
- **Early Detection and Prevention of Defects**: The SQA plan includes processes for early detection and prevention of defects. This involves activities such as code reviews, testing, and inspections, which help identify and address issues in the early stages of the development lifecycle, reducing the cost and impact of defects.
- **Cost Reduction**: Identifying and fixing defects early in the development process is more cost- effective than addressing them in later stages or after the software is released. An SQA plan helps minimize the number of defects that reach production, reducing maintenance costs and improving overall efficiency.
- **Improved Collaboration and Communication**: The SQA plan defines roles, responsibilities, and communication channels within the development team. This clarity fosters better collaboration among team members, reducing misunderstandings and improving the overall efficiency of the development process.

- **Risk Management**: An SQA plan includes risk management strategies to identify, assess, and mitigate risks that could impact software quality. This proactive approach helps the company address potential issues before they become significant problems.
- **Enhanced Productivity and Efficiency**: Standardized processes outlined in the SQA plan contribute to improved productivity and efficiency. Consistent methodologies and practices help streamline development, testing, and release processes, leading to faster and more reliable software delivery.

Pros of Software Quality Assurance Plan:

- **Early Defect Identification**: SQA strategies lessen the possibility of expensive problems later in the software development life cycle by facilitating the early discovery and resolution of flaws.
- **Enhanced Quality of Product:** SQA plans help to improve the overall quality of the software product by setting quality standards and procedures, which raises customer satisfaction.
- **Adherence to the standards**: It plans guarantee adherence to industry norms, legal mandates, and optimal methodologies, offering a structure for producing software of superior quality.
- **Improved Cooperation Among Teams:** The SQA plan's well-defined roles and duties encourage productive teamwork and guarantee that everyone is on the same page with the quality goals.

Cons of Software Quality Assurance Plan:

- **Overhead in Small Projects:** The cost of developing and upholding a detailed SQA plan may be excessive for small projects compared to their scope and complexity.
- **Opposition to Change:** An SQA strategy may involve changes that teams accustomed to their current workflows find difficult to accept, which could result in a time of adjustment and resistance.
- **Documentation Complexity:** SQA plans with high documentation requirements run the risk of adding complexity and coming off as bureaucratic to teams, which makes it difficult to keep documentation up to date.
- **Reliance on Human Elements:** An SQA plan's performance depends on human elements like procedure adherence and attention to detail, which makes human mistake a possibility.

Software Quality Framework - Software Engineering

Software Quality Framework is a model for software quality that ensures quality by connecting and integrating the different views of software quality. This article focuses on discussing the Software Quality Framework.

What is a Software Quality Framework?

Software Quality Framework connects the customer view with the developer's view of software quality and it treats software as a product.

1. The software product view describes the characteristics of a product that bear on its ability to satisfy stated and implied needs.

2. This is a framework that describes all the different concepts relating to quality in a common way measured by a qualitative scale that can be understood and interpreted commonly.

Therefore, the most influential factor for the developers is the customer perception. This framework connects the developer with the customer to derive a common interpretation of quality.

Developers View

Validation and verification are two independent methods used together to check that a software product meets the requirements and that it fulfills its intended purpose. Validation checks that the product design satisfies the purposeful usage and verification checks for errors in the software.

1. The primary concern for developers is in the design and engineering processes involved in producing software.
2. Quality can be measured by the degree of conformance to predetermined requirements and standards, and deviations from these standards can lead to poor quality and low reliability.
3. While validation and verification are used by the developers to improve the software, the two methods don't represent a quantifiable quality measurement.
4. The developer's view of software quality and the customer's view of software quality are both different things.

For example, the customer understands or describes the quality of operation as meeting the requirement while the developers use different factors to describe the software quality. The developer view of quality in the software is influenced by many factors. This model stresses on 3 primary ones:

The code: *It is measured by its correctness and reliability.*

The data: *The application integrity measures it.*

Maintainability: *It has different measures the simplest is the mean time to change.*

Users View

When the user acquires software, he/she always expect a high-quality software. When end users develop their software then quality is different. End-user programming, a phrase popularized by which is programming to achieve the result of a program primarily for personal, rather than public use.

1. The important distinction here is that software itself is not primarily intended for use by many users with varying needs.
2. For example, a teacher may write a spreadsheet to track student's test scores.
3. In these end-user programming situations, the program is a means to an end that could be used to accomplish a goal.
4. In contradiction to end-user programming, professional programming has the goal of producing software for others to use.
5. For example, the moment a novice Web developer moves from designing a web page for himself to designing a Web page for others, the nature of this activity has changed.
6. Users find software quality as a fit between their goals and software's functionality.

7. The better the quality, the more likely the user will be satisfied with the soft-ware.
8. When the quality is bad, developers must meet user needs or face a diminishing demand for their software.

Therefore, the user understands quality as fitness for purpose. Avoiding complexity and keeping software simple, considerably lessens the implementation risk of software. In some instances, users abandoned the implementation of a complex software because the software developers were expecting the users to change their business and to go with the way the software works.

Product View

The product view describes quality as correlated to inherent characteristics of the product. Product quality is defined as the set of characteristics and features of a product that gives contribution to its ability to fulfill given requirements.

1. Product quality can be measured by the value-based view which sees the quality as dependent on the amount a customer is willing to pay for it.
2. According to the users, a high-quality product is one that satisfies their expectations and preferences while meeting their requirement.
3. Satisfaction of end users of the product represents craft to learn, use, upgrade the product and when asked to participate in rating the product, a positive rating is given.

ISO9000

ISO9000 is an international standard of quality management and quality assurance. It certifies the companies that they are documenting the quality system elements which are needed to run a efficient and quality system.

The International organization for Standardization is a worldwide federation of national standard bodies. The **International standards organization (ISO)** is a standard which serves as a for contract between independent parties. It specifies guidelines for development of **quality system**. Quality system of an organization means the various activities related to its products or services. Standard of ISO addresses to both aspects i.e. operational and organizational aspects which includes responsibilities, reporting etc. An ISO 9000 standard contains set of guidelines of production process without considering product itself.

Why ISO Certification required by Software Industry?

There are several reasons why software industry must get an ISO certification. Some of reasons are as follows :

- This certification has become a standards for international bidding.
- It helps in designing high-quality repeatable software products.
- It emphasis need for proper documentation.
- It facilitates development of optimal processes and totally quality measurements.

Features of ISO 9001 Requirements :

- Document control - All documents concerned with the development of a software product should be properly managed and controlled.
- Planning - Proper plans should be prepared and monitored.

- Review - For effectiveness and correctness all important documents across all phases should be independently checked and reviewed .
- Testing - The product should be tested against specification.
- Organizational Aspects - Various organizational aspects should be addressed e.g., management reporting of the quality team.

SEI-CMM

SEI (Software Engineering Institute) - Capability Maturity Model (CMM) is specifically for software organizations to certify them at which level, they are following and maintaining the quality standards.

What is the Capability Maturity Model (CMM?)

Capability Maturity Model (CMM) was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987. It is not a software process model. It is a framework that is used to analyze the approach and techniques followed by any organization to develop software products. It also provides guidelines to enhance further the maturity of the process used to develop those software products.

It is based on profound feedback and development practices adopted by the most successful organizations worldwide. This model describes a strategy for software process improvement that should be followed by moving through 5 different levels. Each level of maturity shows a process capability level. All the levels except level 1 are further described by Key Process Areas (KPA).

Importance of Capability Maturity Model

- **Optimization of Resources:** CMM helps businesses make the best use of all of their resources, including money, labor, and time. Organizations can improve the effectiveness of resource allocation by recognizing and getting rid of unproductive practices.
- **Comparing and Evaluating:** A formal framework for benchmarking and self-evaluation is offered by CMM. Businesses can assess their maturity levels, pinpoint their advantages and disadvantages, and compare their performance to industry best practices.
- **Management of Quality**: CMM emphasizes quality management heavily. The framework helps businesses apply best practices for quality assurance and control, which raises the quality of their goods and services.
- **Enhancement of Process**: CMM gives businesses a methodical approach to evaluate and enhance their operations. It provides a road map for gradually improving processes, which raises productivity and usefulness.
- **Increased Output:** CMM seeks to boost productivity by simplifying and optimizing processes. Organizations can increase output and efficiency without compromising quality as they go through the CMM levels.

Principles of Capability Maturity Model (CMM)

- People's capability is a competitive issue. Competition arises when different organizations are performing the same task (such as software development). In such a case, the people of an organization are sources of strategy and skills, which in turn results in better performance of the organization.

- The people's capability should be defined by the business objectives of the organization.
- An organization should invest in improving the capabilities and skills of the people as they are important for its success.
- The management should be responsible for enhancing the capability of the people in the organization.
- The improvement in the capability of people should be done as a process. This process should incorporate appropriate practices and procedures.
- The organization should be responsible for providing improvement opportunities so that people can take advantage of them.
- Since new technologies and organizational practices emerge rapidly, organizations should continually improve their practices and develop the abilities of people.

Following are the important differences between ISO9000 and SEI-CMM.

Sr. No.	**Key**	**ISO9000**	**SEI-CMM.**
1	Definition	ISO9000 is an international standard of quality management and quality assurance. It certifies the companies that they are documenting the quality system elements which are needed to run an efficient and quality system.	SEI-CMM is specifically for software organizations to certify them at which level, they are following and maintaining the quality standards.
2	Focus	Focus of ISO9000 is on customer supplier relationship, and to reduce the customer's risk.	Focus of SEI-CMM is to improve the processes to deliver a quality software product to the customer.
3	Target Industr y	ISO9000 is used by manufacturing industries.	SEI-CMM is used by software industry.
4	Recognition	ISO9000 is univesally accepted accross lots of countries.	SEI-CMM is mostly used in USA.
5	Guidelines	ISO9000 guides about concepts, priciples and safeguards to be in place in a workplace.	SEI-CMM specifies what is to be followed at what level of maturity.
6	Levels	ISO9000 has one acceptance level.	SEI-CMM has five acceptance levels.
Sr. No.	**Key**	**ISO9000**	**SEI-CMM.**

7	Validity	ISO9000 certificate is valid for three years.	SEI-CMM certificate is valid for three years as well.
8	Level	ISO9000 has no levels.	SEI-CMM has five levels, Initial, Repeatable, Defined, Managed and Optimized.
9	Focus	ISO9000 focuses on following a set of standards so that firm's delivery are successful everytime.	SEI-CMM focuses on improving the processes.