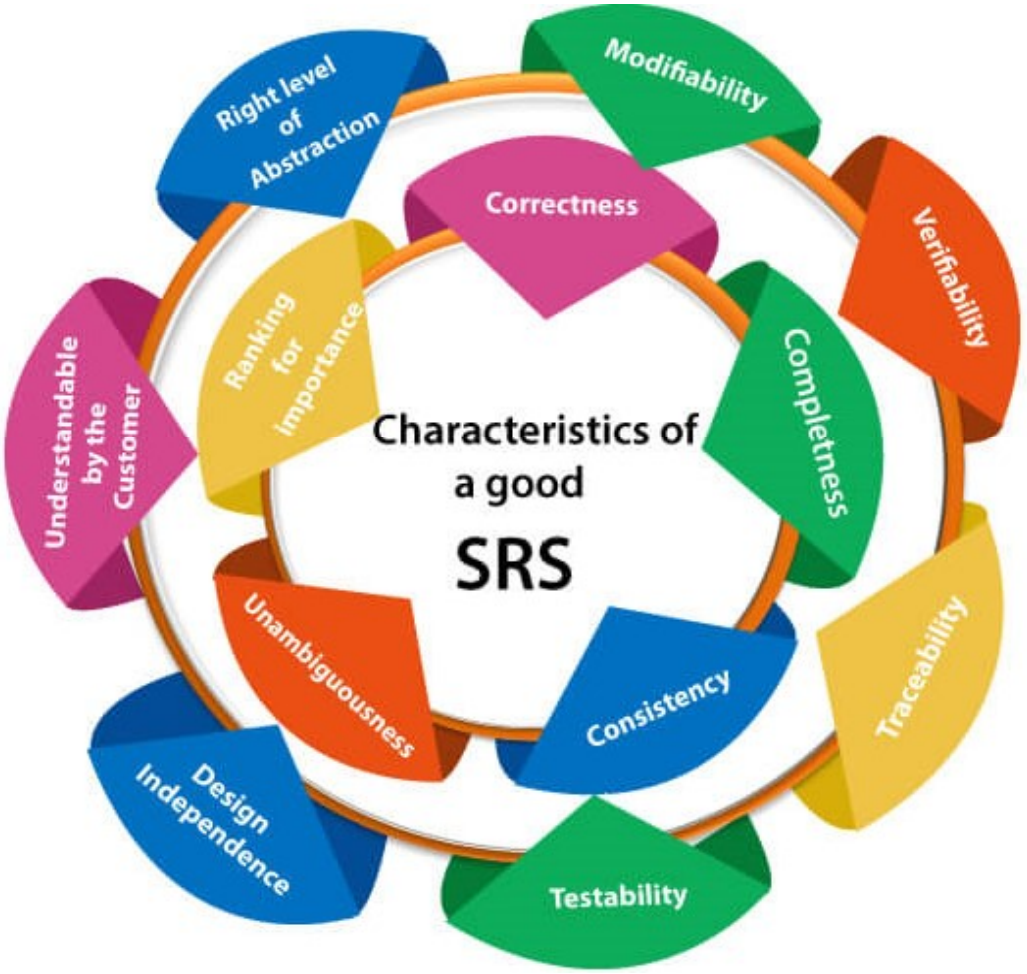
Unit-4
Software Requirement and Specification

- Functional and Non-functional requirements, Requirement Engineering Process (feasibility studies, requirements elicitation and analysis, requirement validation, requirement management)
- Data Modeling and Flow Diagram
- Software Prototyping techniques
- Requirement definition and specifications.

Software Requirement Specification

- Software Requirement Specification (SRS) Format as the name suggests, is a complete specification and description of requirements of the software that need to be fulfilled for the successful development of the software system.
- The Software Requirements Specification (SRS), also known as a requirements document, is a critical artefact produced during the requirements stage of the software development process.
- The interaction between different customers and contractors is done because it is necessary to fully understand the needs of customers.
- These requirements can be functional as well as non-functional depending upon the type of requirement.

Characteristics of good SRS



Following are the features of a good SRS document:

1. **Correctness:** User review is used to provide the accuracy of requirements stated in the SRS. SRS is said to be perfect if it covers all the needs that are truly expected from the system.
2. **Completeness:** The SRS is complete if, and only if, it includes the following elements:

- All essential requirements, whether relating to functionality, performance, design, constraints, attributes, or external interfaces.
- Definition of their responses of the software to all realizable classes of input data in all available categories of situations.
- Full labels and references to all figures, tables, and diagrams in the SRS and definitions of all terms and units of measure.

3. Consistency

- An SRS is consistent if no subset of individual requirements conflicts. Conflicts can arise in three ways:

  A. Conflicting Characteristics of Real-World Objects:

  - Example: One requirement specifies a tabular report format, while another specifies a textual format.

  B. Logical or Temporal Conflicts in Actions:

  - Example: One requirement specifies adding two inputs, while another specifies multiplying them.

  C. Inconsistent Terminology:

  - Example: A program's request for user input is called a "prompt" in one requirement and a "cue" in another.

4. Unambiguousness

- SRS is unambiguous when every fixed requirement has only one interpretation.
- This suggests that each element is uniquely interpreted.

5. Ranking for importance and stability

- The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement.
- Typically, all requirements are not equally important. Some prerequisites may be essential, especially for life-critical applications, while others may be desirable. Each element should be identified to make these differences clear and explicit.

6. Modifiability

- SRS should be made as modifiable as likely and should be capable of quickly obtain changes to the system to some extent.
- Modifications should be perfectly indexed and cross-referenced.

7. Verifiability

- SRS is correct when the specified requirements can be verified with a cost-effective system to check whether the final software meets those requirements.
- The requirements are verified with the help of reviews.

8. Traceability

- The SRS is traceable if the origin of each of the requirements is clear and if it facilitates the referencing of each condition in future development or enhancement documentation.

There are two types of Traceability:

i. **Backward Traceability:** Forward traceability involves tracking the flow of information from one stage of the software development lifecycle to the next. It helps ensure that all requirements are implemented correctly and that the software system meets the specified needs.

ii. **Forward Traceability:** Backward traceability involves tracking the origin of a particular software artifact or defect. It helps identify the root cause of issues and determine the impact of changes on previous stages of the development lifecycle.

9. Design Independence

- There should be an option to select from multiple design alternatives for the final system.
- More specifically, the SRS should not contain any implementation details.

10. Testability

- An SRS should be written in such a method that it is simple to generate test cases and test plans from the report.

11. Understandable by the customer

- An end user may be an expert in his/her explicit domain but might not be trained in computer science. Hence, the purpose of formal notations and symbols should be avoided too as much extent as possible.
- The language should be kept simple and clear.

12. The right level of abstraction

- If the SRS is written for the requirements stage, the details should be explained explicitly. Whereas, for a feasibility study, fewer analysis can be used. Hence, the level of abstraction modifies according to the objective of the SRS.

Properties of a good SRS document

- **Concise:** The SRS report should be concise and at the same time, unambiguous, consistent, and complete.
- **Structured:** It should be well-structured. A well-structured document is simple to understand and modify.
- **Black-box view:** It should only define what the system should do and refrain from stating how to do these.
- **Conceptual integrity:** It should show conceptual integrity so that the reader can merely understand it.
- **Verifiable:** All requirements of the system, as documented in the SRS document, should be correct.

Purpose SRS

- Acts as a foundation for subsequent software engineering activities.
- Ensures all requirements are elicited, analyzed, and formally documented.
- Provides a representation of the software that allows customers to validate whether it aligns with their expectations and needs.
- Contains user requirements and detailed system specifications.

Characteristics SRS

- Specifies a particular software product, program, or set of applications.
- Details the functions performed in a specific environment.

Goals of SRS

- Defines user needs and expectations.
- Documents user requirements for clarity and validation.
- Acts as a contractual document between the customer and the developer.
- Guides development processes to ensure alignment with client requirements.

Importance SRS

- Enables clear communication between stakeholders (clients, developers, and users).
- Establishes measurable criteria for software acceptance.
- Serves as a baseline to avoid misunderstandings and disputes.

❑ Functional Requirement

- These are the requirements that the end user specifically demands as basic facilities that the system should offer.
- It describes what the system should do, i.e., specific functionality or tasks.
- It focuses on the behavior and features of the system.
- It defines the actions and operations of the system.
- User authentication data input/output, transaction processing.

Examples:

- What are the features that we need to design for this system?
- What are the edge cases we need to consider, if any, in our design?

❑ Non-Functional Requirement

- ➢ These are the quality constraints that the system must satisfy according to the project contract.
- ➢ The priority or extent to which these factors are implemented varies from one project to another.
- ➢ They are also called non-behavioral requirements.

They deal with issues like:

- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

Examples:

- Each request should be processed with the minimum latency?
- System should be highly valuable.

Examples of Functional and Non-functional Requirements

Online Banking System

1. Functional Requirements:

- Users should be able to log in with their username and password.
- Users should be able to check their account balance.
- Users should receive notifications after making a transaction.

2. Non-functional Requirements:

- The system should respond to user actions in less than 2 seconds.
- All transactions must be encrypted and comply with industry security standards.
- The system should be able to handle 100 million users with minimal downtime.

Food Delivery App

1. Functional Requirements

- Users can browse the menu and place an order.
- Users can make payments and track their orders in real time.

2. Non-functional Requirements:

- The app should load the restaurant menu in under 1 second.
- The system should support up to 50,000 concurrent orders during peak hours.
- The app should be easy to use for first-time users, with an intuitive interface.

Difference between Functional requirements and Non-functional Requirements

Functional Requirements	Non-Functional Requirements
Describes what the system should do, i.e., specific functionality or tasks.	Describes how the system should perform, i.e., system attributes or quality.
Focuses on the behavior and features of the system.	Focuses on the performance, usability, and other quality attributes.
Defines the actions and operations of the system.	Defines constraints or conditions under which the system must operate.
User authentication, data input/output, transaction processing.	Scalability, security, response time, reliability, maintainability.
Easy to measure in terms of outputs or results.	More difficult to measure, often assessed using benchmarks or SLAs.
Drives the core design and functionality of the system.	Affects the architecture and overall performance of the system.
Directly related to user and business requirements.	Focuses on user experience and system performance.
Typically documented in use cases, functional specifications, etc.	Documented through performance criteria, technical specifications, etc.
Can be tested through functional testing (e.g., unit or integration tests).	Evaluated through performance testing, security testing, and usability testing.
Determines what the system must do to meet user needs.	Depends on how well the system performs the required tasks.

Requirement Engineering Process:

- 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.

- Feasibility Study
- Requirements elicitation
- Requirements specification
- Requirements for verification and validation
- 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.

❑ Technical Feasibility:

- In Technical Feasibility current resources both hardware software along required technology are analyzed to develop the project.
- This technical feasibility study reports whether there are correct required resources and technologies that will be used for project development.

❑ 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.

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.

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.

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.
- 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.

Techniques can be used to elicit requirements:

- **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.

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.
- 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.

Types of requirements commonly specified in this step

- **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.
- **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.
- **Constraints:** These describe any limitations or restrictions that must be considered when developing the software system.
- **Acceptance Criteria**: These describe the conditions that must be met for the software system to be considered complete and ready for release.

4. Requirements Verification and Validation

❑ Verification:

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

❑ Validation:

- 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.
- 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.

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.

Key activities involved in requirements management:

- **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.
- **Version control**: This involves keeping track of different versions of the requirements document and other related artifacts.
- **Traceability**: This involves linking the requirements to other elements of the development process, such as design, testing, and validation.
- **Communication:** This involves ensuring that the requirements are communicated effectively to all stakeholders and that any changes or issues are addressed promptly.
- **Monitoring and reporting**: This involves monitoring the progress of the development process and reporting on the status of the requirements.

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.

Stages in Software Engineering Process

- Requirements engineering is a critical process in software engineering that involves identifying, analyzing, documenting, and managing the requirements of a software system.
- The requirements engineering process consists of the following stages:
  - **Elicitation:** In this stage, the requirements are gathered from various stakeholders such as customers, users, and domain experts. The aim is to identify the features and functionalities that the software system should provide.
  - **Analysis:** In this stage, the requirements are analyzed to determine their feasibility, consistency, and completeness. The aim is to identify any conflicts or contradictions in the requirements and resolve them.
  - **Specification:** In this stage, the requirements are documented in a clear, concise, and unambiguous manner. The aim is to provide a detailed description of the requirements that can be understood by all stakeholders.
  - **Validation:** In this stage, the requirements are reviewed and validated to ensure that they meet the needs of all stakeholders. The aim is to ensure that the requirements are accurate, complete, and consistent.
  - **Management:** In this stage, the requirements are managed throughout the software development lifecycle. The aim is to ensure that any changes or updates to the requirements are properly documented and communicated to all stakeholders.

Data Modeling and flow diagram

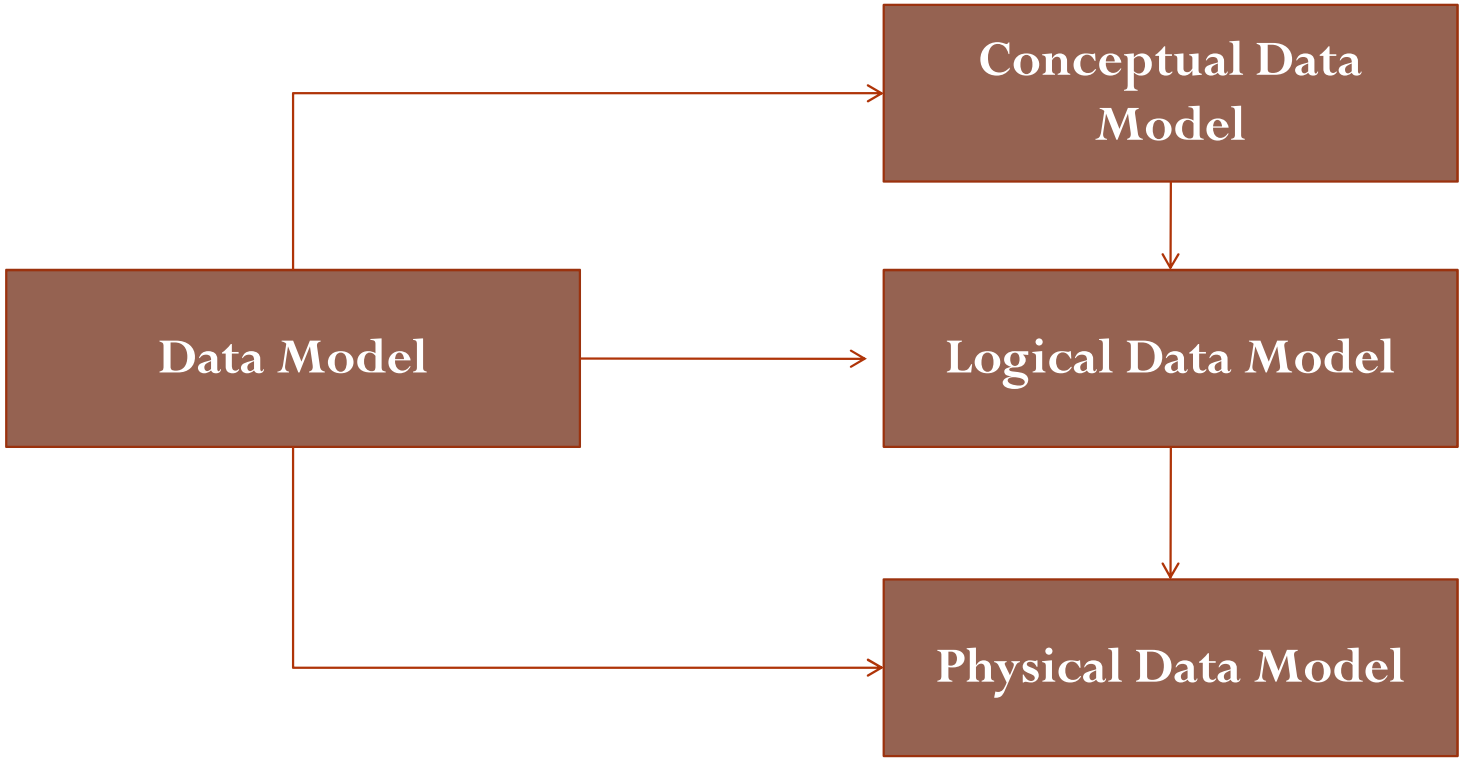
Data Modeling:

- Data modelling in analysis is the process of creating a visual representation , abstraction of data structures, relationships, and rules within a system or organization.
- Determining and analysing the data requirements required to support business activities within the bounds of related information systems in organisations is another process known as data modelling.
- The main objective of data modelling is to provide a precise and well-organized framework for data organisation and representation, since it enables efficient analysis and decision-making.

Data Model

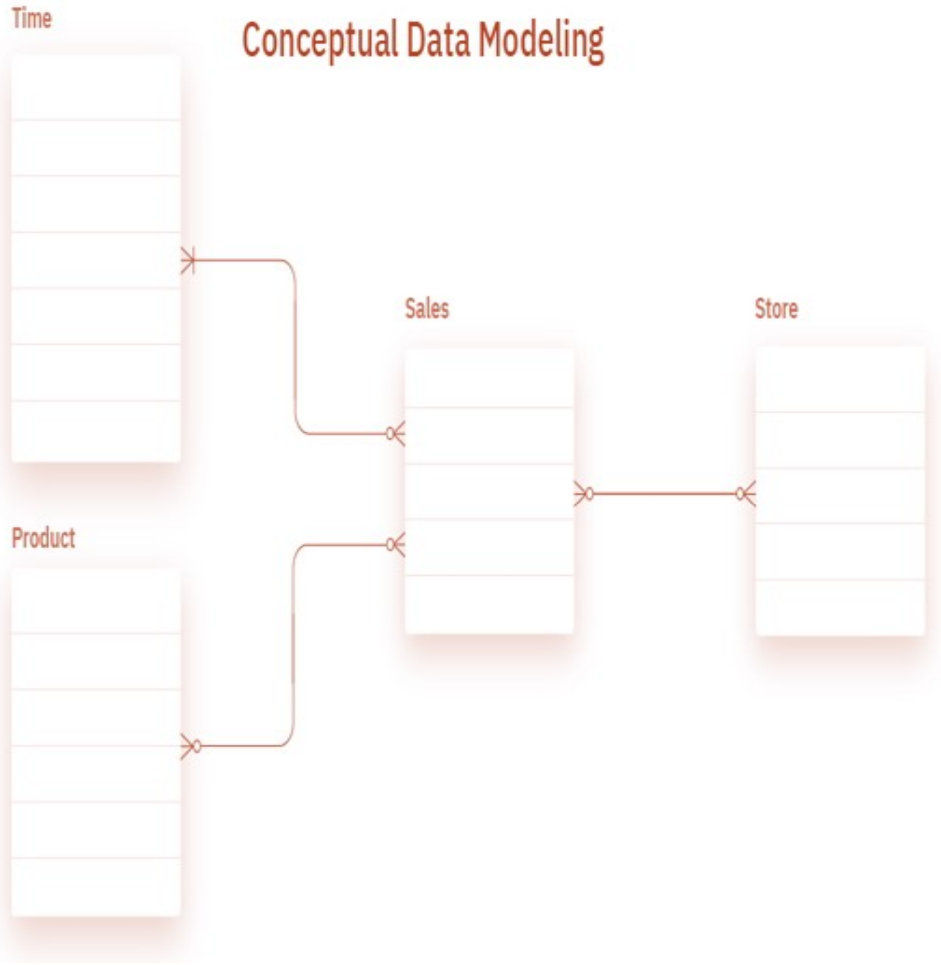
- Data models are visual representations of an enterprise's data elements and the connections between them.
- Models assist to define and arrange data in the context of key business processes, hence facilitating the creation of successful information systems.

Types of Data Model



Conceptual Data Model

- Conceptual Data Model is a representations of data Examine and describe in depth your abstract, high-level business concepts and structures.
- They are most commonly employed when working through high-level concepts and preliminary needs at the start of a new project.
- The main purpose of this data model is to organize, define business problems , rules and concepts.
- It helps business people to view any data like market data, customer data, and purchase data.

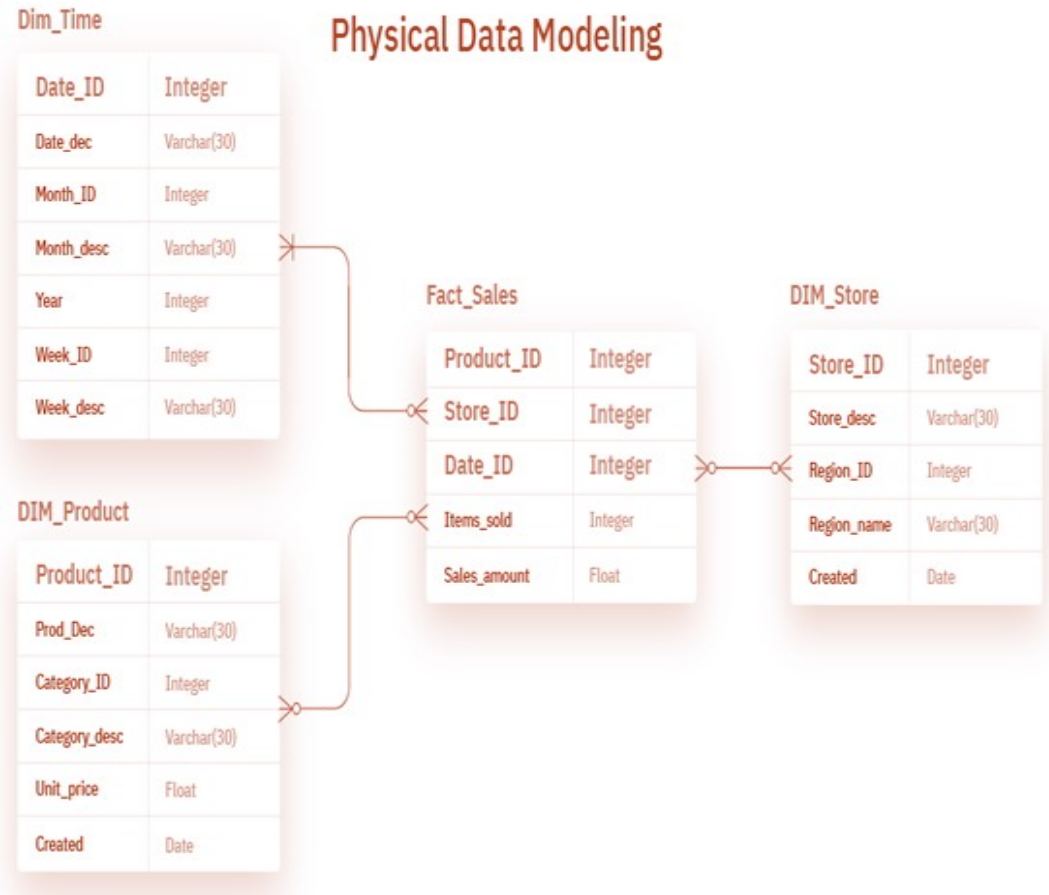


Logical Data Model:

- In the logical data model, By offering a thorough representation of the data at a logical level, the logical data model expands on the conceptual model.
- It outlines the tables, columns, connections, and constraints that make up the data structure.
- Although logical data models are not dependant on any particular database management system (DBMS), they are more similar to how data would be implemented in a database.
- The physical design of databases is based on this idea.

Physical Data Model:

- In Physical Data model ,The implementation is explained with reference to a particular database system.
- It is made with queries and the database language.
- Every table, column, and constraint—such as primary key, foreign key, NOT NULL, etc.—is represented in the physical data model.
- The creation of a database is the primary task of the physical data model.
- Developers and database administrators (DBAs) designed this model.
- This kind of data modelling aids in the creation of the schema and provides us with an abstraction of the databases.
- Constraints, RDBMS features, and database column keys are made possible by the physical data model.



Flow Diagram:

- Flowchart
- Data Flow Diagram

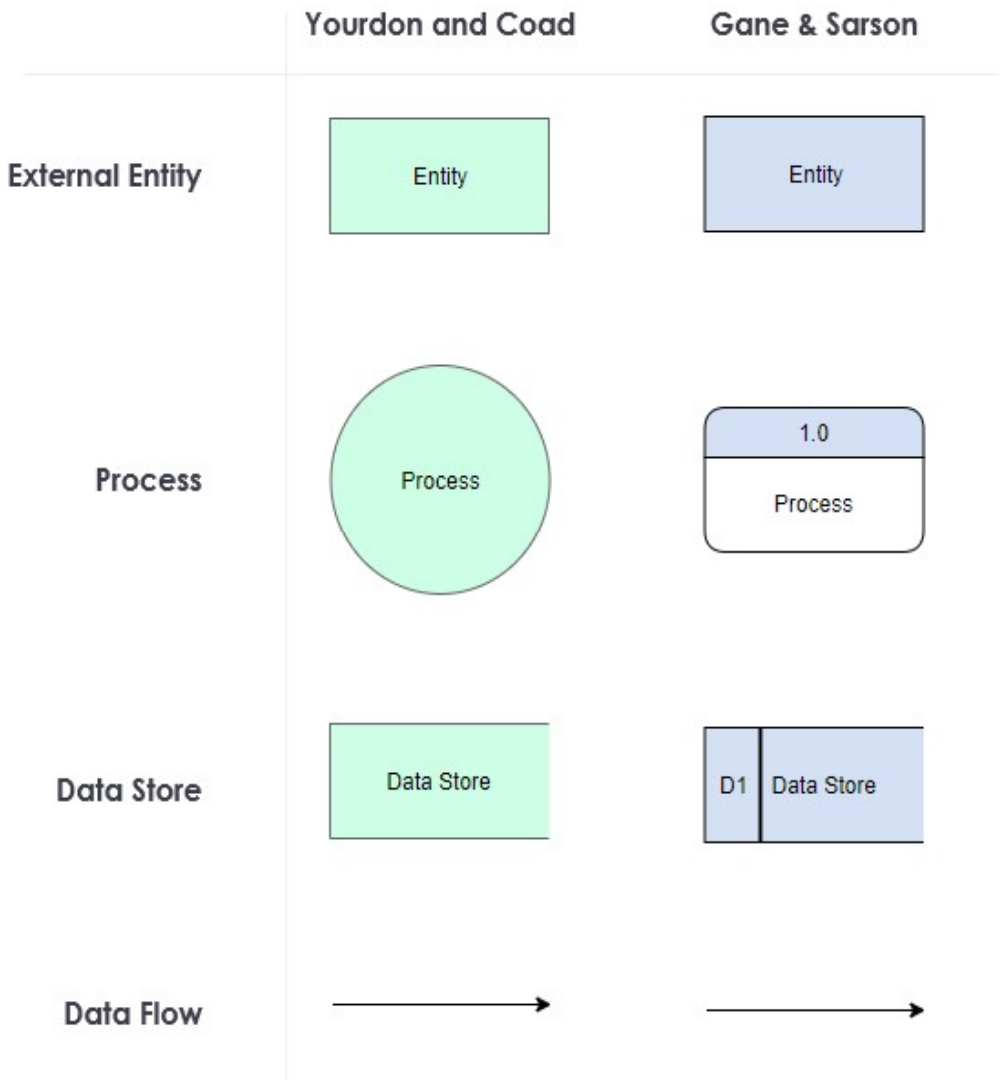
Data Flow Diagram(DFD):

- The flow of data in a system or process is represented by a Data Flow Diagram (DFD).
- It also gives insight into the inputs and outputs of each entity and the process itself.
- Data Flow Diagram (DFD) does not have a control flow and no loops or decision rules are present.
- Specific operations, depending on the type of data, can be explained by a flowchart.
- It is a graphical tool, useful for communicating with users, managers and other personnel. it is useful for analyzing existing as well as proposed systems.
- A DFD is the graphical representation of the flow of data from one component to another component in any system.
- DFD does not give detailed description of system but graphically describes system's data and how the data interact with the system.
- All data flow diagrams include four main elements: entity, process, data store and data flow.

Basic symbols of DFD:

Four basic elements are used to construct data-flow diagrams:

1. processes
2. data-flows
3. data stores
4. external entities



Characteristics of Data Flow Diagram

Below are some characteristics of Data Flow Diagram:

- **Graphical Representation**
  - Data Flow Diagram (DFD) use different symbols and notation to represent data flow within system. That simplify the complex model.
- **Problem Analysis**
  - Data Flow Diagram are very useful in understanding a system and can be effectively used during analysis.
  - Data Flow Diagram are quite general and are not limited to problem analysis for software requirements specification.
- **Abstraction**
  - Data Flow Diagram (DFD) provides a abstraction to complex model i.e. DFD hides unnecessary implementation details and show only the flow of data and processes within information system.
- **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.
- **Data Flow**
  - The primary objective of Data Flow Diagram (DFD) is to visualize the data flow between external entity, processes and data store.
  - Data Flow is represented by an arrow Symbol.
- **Ease of Understanding**
  - Data Flow Diagram (DFD) can be easily understand by both technical and non-technical stakeholders.
- **Modularity**
  - Modularity can be achieved using Data Flow Diagram (DFD) as it breaks the complex system into smaller module or processes.

Types of Data Flow Diagram (DFD)

- There are two types of Data Flow Diagram (DFD)
  - Logical Data Flow Diagram
  - Physical Data Flow Diagram

Logical Data Flow Diagram (DFD)

- 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 DFD 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.

Physical Data Flow Diagram (DFD)

- 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 DFD is more specific and close to implementation.

Physical Data Flow Diagrams (DFDs) are used in the following situations:

- **Detailed Design Phase:** Physical DFDs are helpful during the detailed design phase of a system.
- **Implementation Planning:** Physical DFD helps developer during Implementation planning as it provides a detailed view of how data flow within the system's physical components, such as hardware devices, databases, and software modules.
- **Integration with Existing Systems:** Physical DFD are important for understanding data flow when integrating a new system with existing systems or external entities,
- **Documentation and Maintenance:** Physical DFDs can be referred as documentation for system architecture and data flow patterns, that helps in system maintenance and troubleshooting.

Following are the rules which are needed to keep in mind while drawing a DFD(Data Flow Diagram).

- **Data can not flow between two entities.**
  - Data flow must be from entity to a process or a process to an entity.
  - There can be multiple data flows between one entity and a process.
- **Data can not flow between two data stores**
  - Data flow must be from data store to a process or a process to an data store.
  - Data flow can occur from one data store to many process.
- **Data can not flow directly from an entity to data store**
  - Data Flow from entity must be processed by a process before going to data store and vice versa.
- **A process must have at least one input data flow and one output data flow**
  - Every process must have input data flow to process the data and an output data flow for the processed data.
- **A data store must have at least one input data flow and one output data flow**
  - Every data store must have input data flow to store the data and an output data flow for the retrieved data.
- **Two data flows can not cross each other.**
- **All the process in the system must be linked to minimum one data store or any other process.**

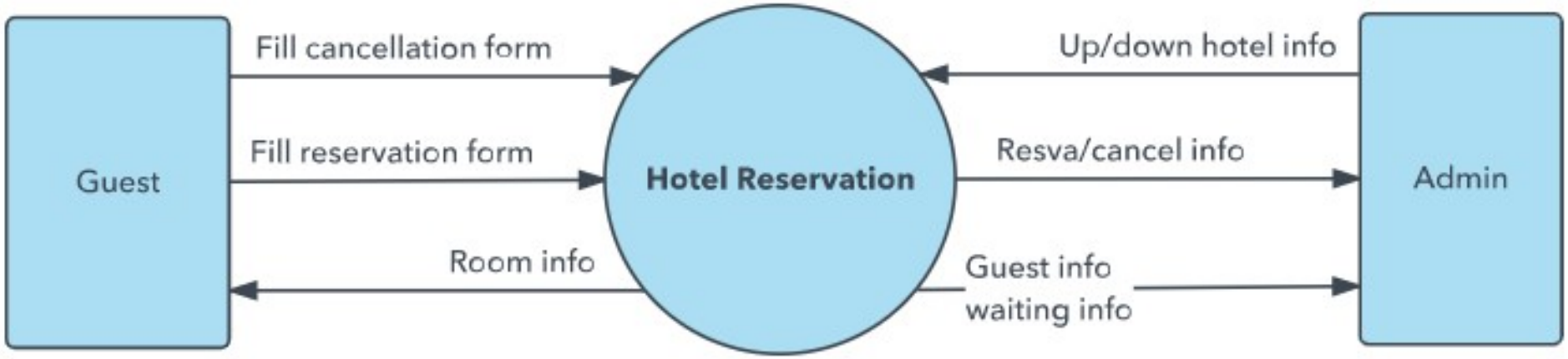
DFD levels :

- Levels or layers are used in DFDs to represent progressive degrees of detail about the system or process. These levels include:
  - **Level 0:** Also known as a "context diagram," this is the highest level and represents a very simple, top-level view of the system being represented.
  - **Level 1:** Still a relatively broad view of the system, but incorporates sub processes and more detail.
  - **Level 2:** Provides even more detail and continues to break down sub processes as needed.
  - **Level 3:** While this amount of detail is uncommon, complex systems can benefit from representation at this level.

Context Diagram/(0-Level DFD):

- Level 0 is the highest-level Data Flow Diagram (DFD), which provides an overview of the entire system.
- 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 a context diagram.
- It's designed to be an abstraction view, showing the system as a single process with its relationship to external entities.
- It represents the entire system as a single bubble with input and output data indicated by incoming/outgoing arrows.

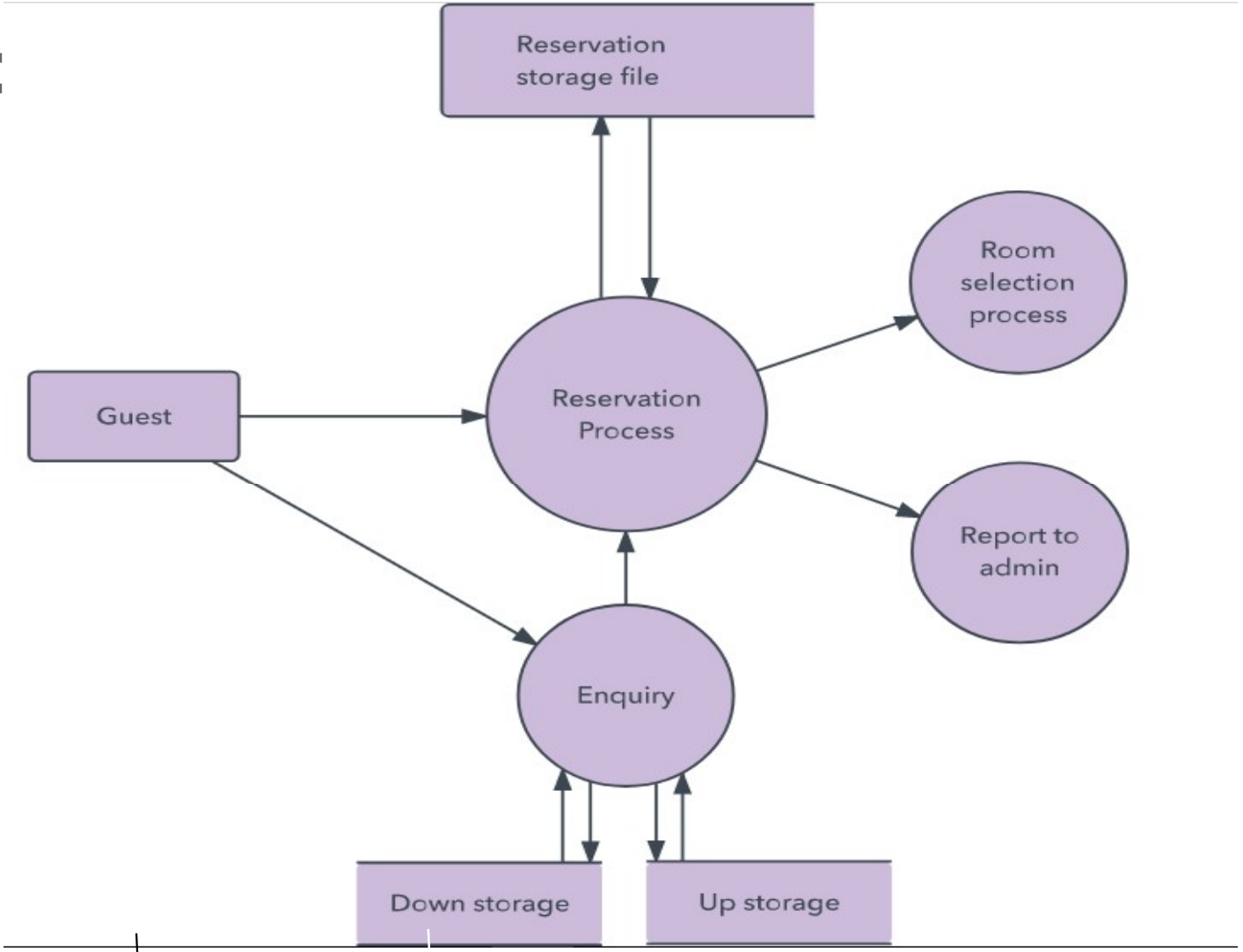
Level-0 DFD of Hotel Reservation system



Level 1 DFD

- 1-Level 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.
- In 1-level Data Flow Diagram (DFD), the context diagram is decomposed into multiple bubbles/processes.
- In this level, we highlight the main functions of the system and breakdown the high-level process of 0-level Data Flow Diagram (DFD) into subprocesses.

Level 1 DFD:


Reservation storage file
Room selection process
Guest
Reservation Process
Report to admin
Enquiry
Down storage
Up storage

Level 2 DFD:

- 2-Level provides an even more detailed view of the system by breaking down the sub-processes identified in the level 1 Data Flow Diagram (DFD) into further sub-processes.
- Each sub-process is depicted as a separate process on the level 2 DFD.
- The data flows and data stores associated with each sub-process are also shown.
- 2-Level Data Flow Diagram (DFD) goes one step deeper into parts of 1-level DFD.
- It can be used to plan or record the specific/necessary detail about the system's functioning.

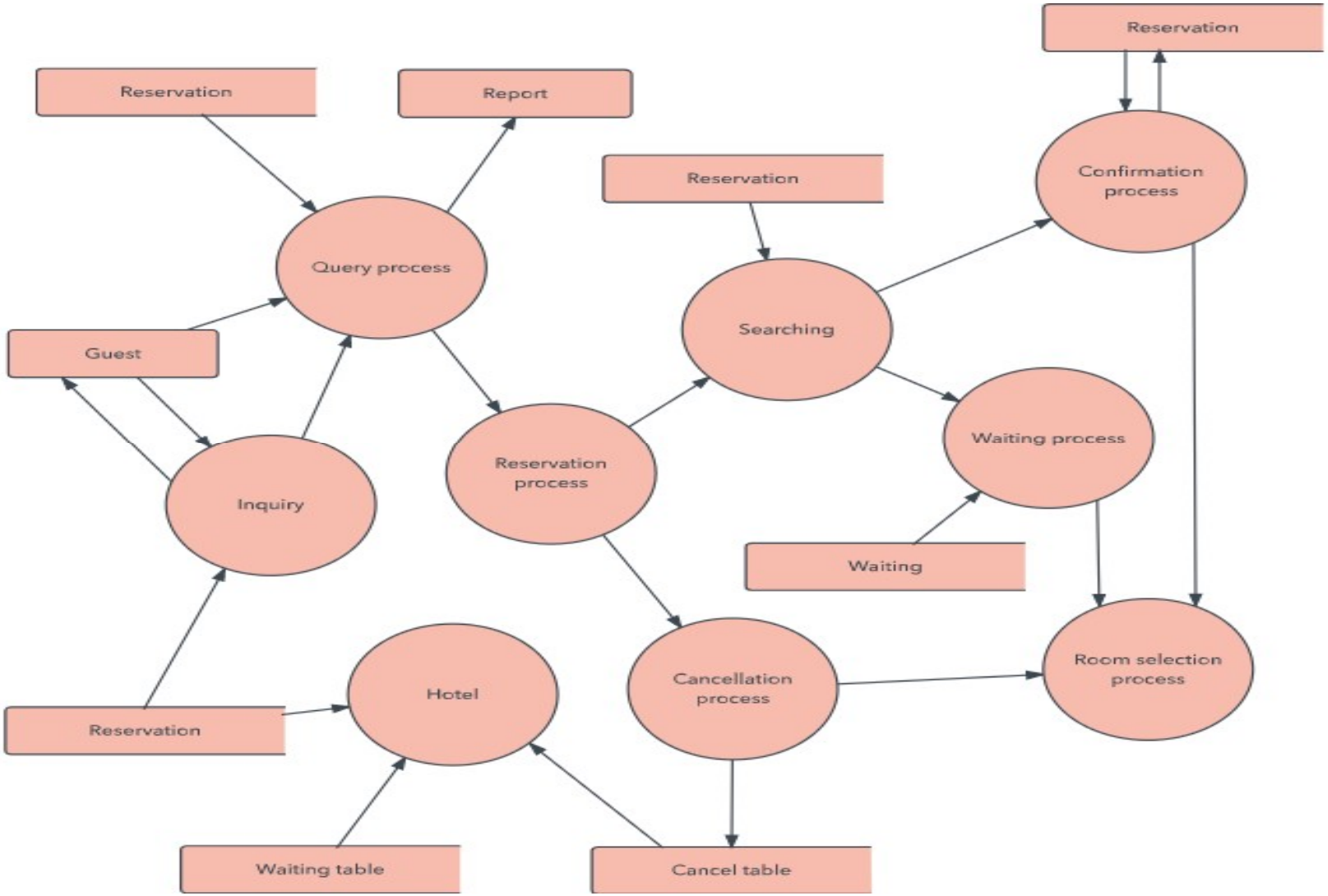


fig: level 2 DFD

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.
- Each process on the level 3 DFD is depicted with a detailed description of its input, processing, and output.
- The data flows and data stores associated with each process are also shown.

Advantages of using Data Flow Diagrams (DFD)

- **Easy to understand:** DFDs are graphical representations that are easy to understand and communicate, making them useful for non-technical stakeholders and team members.
- **Improves system analysis:** DFDs are useful for analyzing a system's processes and data flow, which can help identify inefficiencies, redundancies, and other problems that may exist in the system.
- **Supports system design:** DFDs can be used to design a system's architecture and structure, which can help ensure that the system is designed to meet the requirements of the stakeholders.
- **Enables testing and verification:** DFDs can be used to identify the inputs and outputs of a system, which can help in the testing and verification of the system's functionality.
- **Facilitates documentation:** DFDs provide a visual representation of a system, making it easier to document and maintain the system over time.

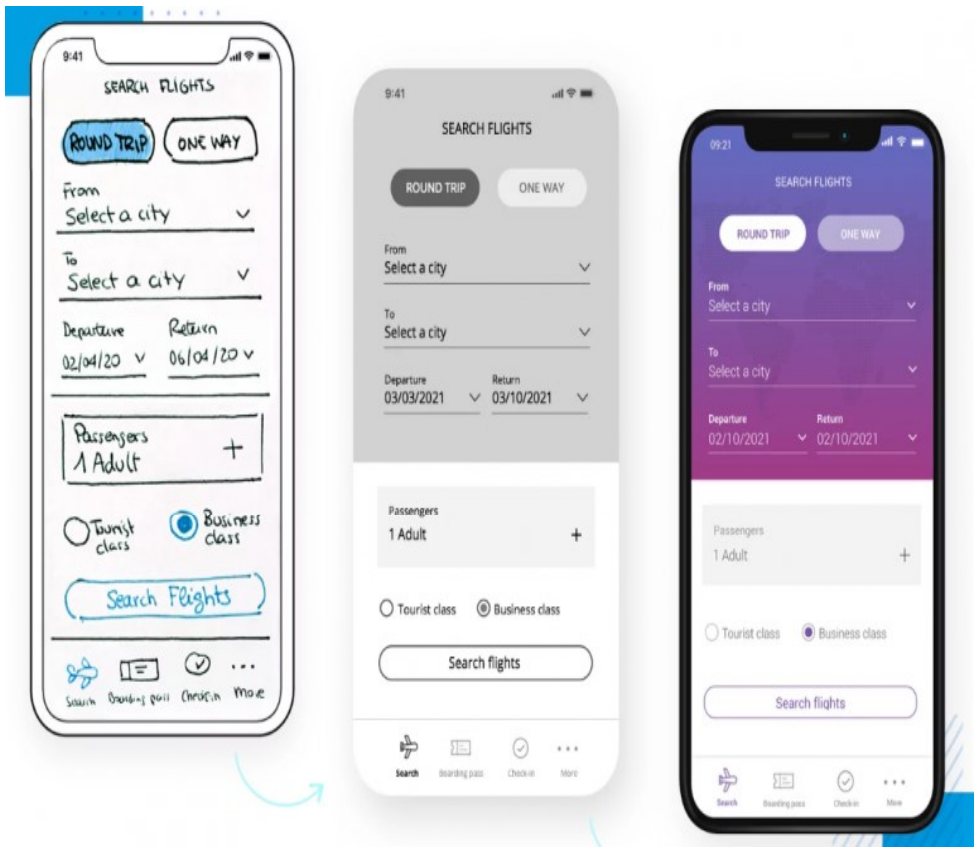
Disadvantages of using DFD

- **Can be time-consuming:** Creating DFDs can be a time-consuming process, especially for complex systems.
- **Limited focus:** DFDs focus primarily on the flow of data in a system, and may not capture other important aspects of the system, such as user interface design, system security, or system performance.
- **Can be difficult to keep up-to-date:** DFDs may become out-of-date over time as the system evolves and changes.
- **Requires technical expertise:** While DFDs are easy to understand, creating them requires a certain level of technical expertise and familiarity with the system being analyzed.

Software Prototyping Technique:

- A software prototype is an incomplete version of a software program. It is for evaluate whether the idea is feasible, whether the code is working fine, and to ensure that the basic functionalities are working. All this without expending vast resources or man-hours.
- Software prototyping is a crucial technique in software development that involves creating a preliminary model or version of the final product. This model, known as a prototype, is used to gather feedback from users and stakeholders, refine the design, and ultimately improve the quality of the final software.
- The objective of a software prototype is simple – correct smaller mistakes before they snowball into bigger irreversible issues.

- A software prototype can be compared to a rough skeleton of an idea. In fact, it has three stages during which the final software product takes form and shape.
  - ***Low-fidelity prototype*** – a rough skeleton of the application or the product based on requirements
  - ***Mid-fidelity prototype*** – A slightly presentable version of the prototype for discussions and user feedback
  - ***High-fidelity prototype*** – A fully-functional and visually appealing version that can be circulated to larger masses for user testing and analysis.



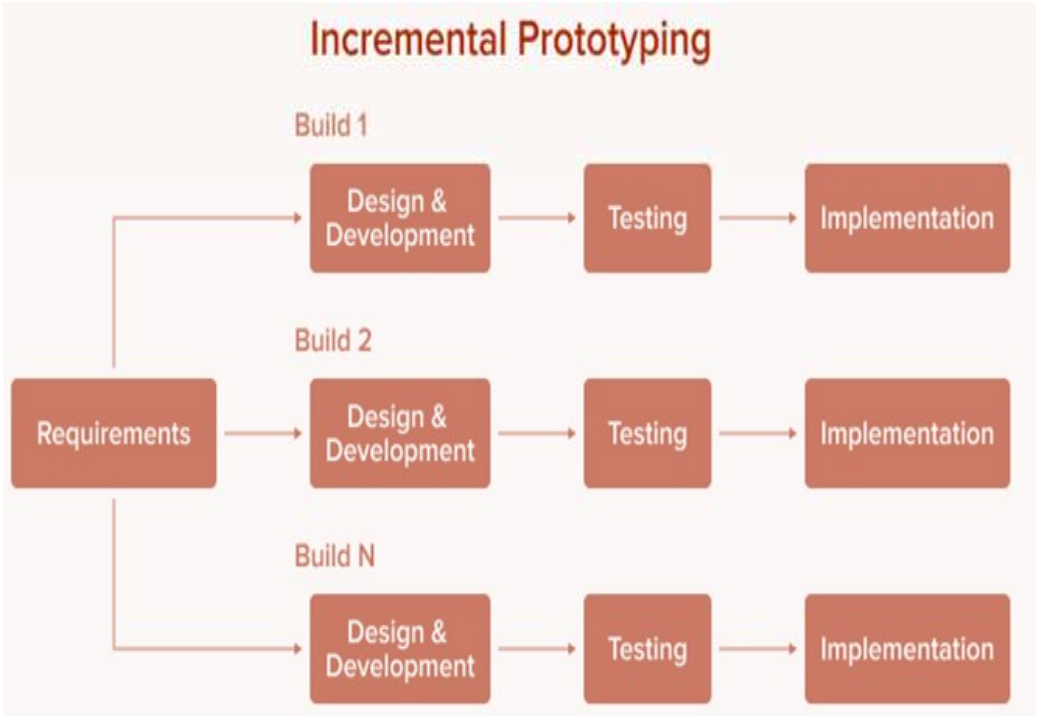
There are four major software prototyping methods you can follow to build your software prototype.

1. Incremental prototyping
2. Throwaway prototyping
3. Extreme prototyping
4. Evolutionary prototyping

- Each method delivers a slightly different end result, however, all leading to building a better software prototype.

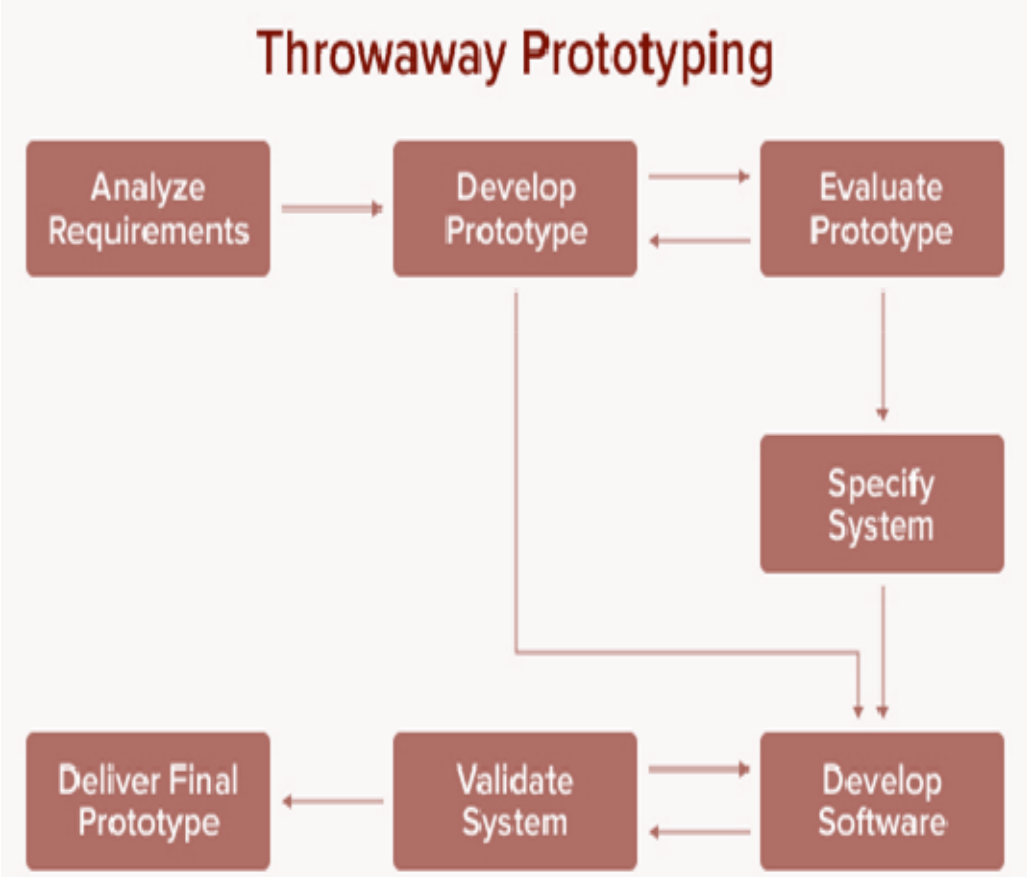
❑ Incremental Prototyping

- Incremental prototyping is useful for enterprise software that has many modules and components which may be loosely related to one another.
- In incremental prototyping, separate small prototypes are built in parallel.
- The individual prototypes are evaluated and refined separately, and then merged into a comprehensive whole, which can then be evaluated for consistency in look, feel, behavior, and terminology.
- The risk with incremental programming is the look and feel can vary so much among the prototypes, the modules feel like completely different pieces of software. The design team must come up with some guiding principles in advance and keep the designers on a short leash to ensure consistency.



❑Throwaway prototyping

- Rapid throwaway or rapid prototyping is one the most important type of prototyping models.
- It is based on the preliminary requirement of the software applications.
- The throwaway prototype can be quickly developed by the team to show how the initial requirements of the system will look visually.
- In this model, the feedback from the customers helps in driving changes in the requirements and after the user feedback is received, the prototype is again created to meet the client's needs.



❑Extreme prototyping

- Extreme Prototyping is a specialized prototyping technique primarily used in web development and web application design.
- It focuses on rapidly creating a functional prototype by dividing the development process into three distinct phases.
- This approach is particularly useful for projects where the user interface (UI) and user experience (UX) are critical, and the backend functionality can be developed iteratively.
- The primary benefit of this method is that it helps developers catch bugs and issues well before it reaches the production stage. As a result, it saves costs that would otherwise require on rebuilding the product to solve the issue or to make it functional for the user.

Phases of Extreme Prototyping

1. Static Prototype (HTML/CSS)

- In this phase, a static prototype of the user interface is created using HTML and CSS.

2. Simulated Functionality (JavaScript)

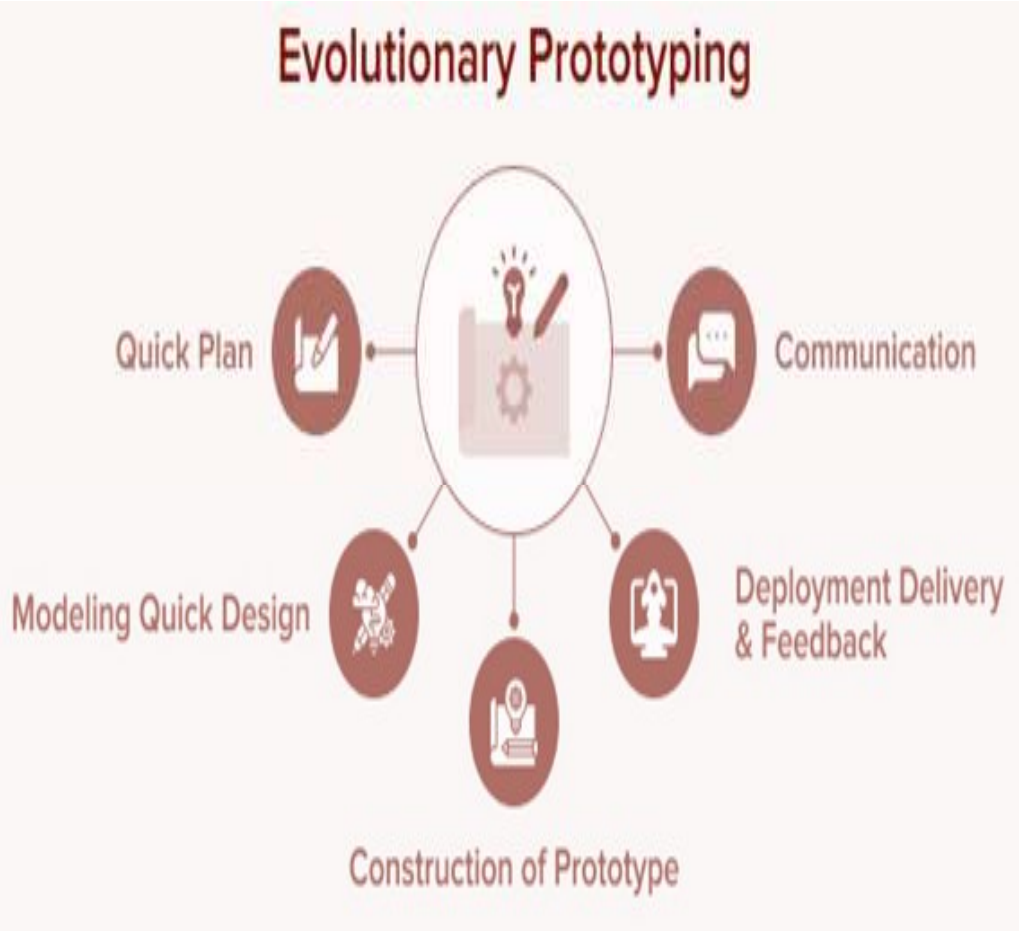
- In the second phase, simulated functionality is added to the static prototype using JavaScript.

3. Backend Integration

- In the final phase, the backend functionality is integrated into the prototype.

❑Evolutionary Prototyping

- Evolutionary prototypes are created by the team in an incrementally refined manner after accepting the customer's final feedback.
- This type of prototyping enables the team to save a lot of development time and effort.
- The prototype is gradually refined and extended into the final product.
- The prototype serves as the foundation for the actual system.
- Main reason behind it is that the development of a prototype from scratch for every different interaction can be very frustrating.



❑ Requirement Definition and Specification

- Requirement Definition and Specification are critical stages in the software development process. They establish a clear understanding of what a system or project must do and how it will function.

1. Requirement Definition

- Requirement definition is the process of gathering, analyzing, and documenting the needs and expectations of stakeholders for a software system.
- It focuses on understanding **what** the system should do from the perspective of users, customers, and other stakeholders.
- The requirement definition is a high-level process where stakeholders identify and agree on what the system or product is supposed to accomplish.

Activities in Requirement Definition

- **Elicitation:**
  - Gathering requirements from stakeholders through interviews, surveys, workshops, and observation.
- **Analysis:**
  - Analyzing the gathered requirements to identify inconsistencies, ambiguities, and conflicts.
- **Prioritization:**
  - Prioritizing requirements based on their importance, feasibility, and impact on the system.
- **Documentation:**
  - Documenting the requirements in a clear and understandable format (e.g., user stories, use cases, or natural language descriptions).

2. Requirement Specification

- Requirement specification is the process of translating the high-level requirements gathered during the requirement definition phase into a detailed, technical document.
- It describes **how** the system will achieve its goals.
- It serves as a blueprint for developers, testers, and other stakeholders.

Activities in Requirement Specification

- **Refinement:**
  - Breaking down high-level requirements into detailed, actionable specifications.
- **Technical Detailing:**
  - Adding technical details such as data formats, system architecture, and interfaces.
- **Validation:**
  - Ensuring that the specified requirements align with stakeholder needs and are feasible to implement.
- **Documentation:**
  - Creating a formal document (e.g., Software Requirements Specification or SRS) that serves as a reference for the development team.