HIGH LEVEL DESIGN

- There are certain common items that we need to specify in the high level design.

- Most important is **Security**: - login screen
- Operating system security
- Application security
- Data security
- Network security
- Physical security

- **Hardware**:

- we need to specify the hardware that will run our application.

- We need to specify additional hardware requirements too.
- Printers.
- Network components (cables, modems, gateways, and routers)
- Servers (database servers, web servers, and application servers)
- Audio and video hardware (webcams, headsets)

- **User Interface**

- Indicate the main methods for navigating through the application.

- We can specify the forms or windows that the application will include.
- we can then verify that they allow the user to perform the tasks defined in the requirements.

- In particular, we should walk through the user stories and use cases and make sure we have included all the forms needed to handle them.

- **Internal Interfaces**:

- When we divide the program into pieces, we should specify how the modules will interact.

- Then the teams assigned to the pieces can work separately.

- It’s important that the high-level design specifies the internal interactions clearly and unambiguously so that the teams can work as independently as possible.

- **External Interfaces:**
- Many applications must interact with external systems.

- For example, suppose we're building a program that assigns crews for a large chartered fishing company.

- The application needs to assign a captain, first mate, and cook for each trip.

- Our program needs to interact with the existing employee database to get information about crew members.

- **Architecture:**
- An application’s architecture describes how its pieces fit together at a high level.

- Architectures attempt to simplify development by reducing the interactions among the pieces of the system.

- For example, a component-based architecture tries to make each piece of the system as separate as possible so that different teams of developers can work on them separately.

- **monolithic architecture**:

- In a monolithic architecture, a single program does everything.

- Important drawbacks - In particular, the pieces of the system are tied closely together, so it doesn't give us a lot of flexibility.

- Advantage - Because everything is built into a single program, there's no need for complicated communication across networks.

- Monolithic architectures are also useful for small applications where a single programmer or team is working on the code.

- **Client/Server Architecture**:

- A client/server architecture separates pieces of the system that need to use a particular function (clients) from parts of the system that provide those functions (servers).

- For example, many applications rely on a database to hold information about customers, products, orders, and employees.

- The application needs to display that information in some sort of user interface.

- One way to do that would be to integrate the database directly into the application.

- One problem with this design is that multiple users cannot use the same data.

- To solve this issue we have **two-tier architecture** where a client (the user interface) is separated from the server (the database).

- The two-tier architecture makes it easier to support multiple clients with the same server, but it ties clients and servers relatively closely together.

- we can increase the separation between the clients and server by using **Three Tier Architecture**.

- The middle tier is separated from the clients and the server by networks.

- The database runs on one computer, the middle tier runs on a second computer, and the instances of the client run on still other computers.

- **Component-based architecture:**

- A component-based architecture decouples the pieces of code.

- Different pieces of the application as components providing services to each other.

- Eg:

- **Service-Oriented architecture:**

- A service-oriented architecture (SOA) is similar to a component-based architecture except the pieces are implemented as services.

- A service is a self-contained program that runs on its own and provides some kind of service for its clients.

- **Data-Centric architectures:**

- Data-centric or database-centric architectures come in a variety of flavors that all use data in some central way.

- Storing data in a relational database system.
- Using tables.
- Using stored procedures inside the database to perform calculations.

- **Event-Driven Architecture:**
- In an event-driven architecture (EDA), various parts of the system respond to events as they occur.

- For example, when a consumer purchases a car, the car's state changes from "for sale" to "sold". A car dealer's system architecture may treat this state change as an event.

- **Rule-Based architecture**
- A rule-based architecture uses a collection of rules to decide what to do next.

- These systems are sometimes called expert systems or knowledge-based systems.

- **Distributed architecture:**

- In a distributed architecture, different parts of the application run on different processors and may run at the same time.

- The processors could be on different computers scattered across the network, or they could be different cores on a single computer.

- **Reports:**

- Almost all the application generate reports.

- Business applications might include reports that deal with customers (who's buying, who has unpaid bills, where customers live), products (inventory, pricing, what's selling well), and users (which employees are selling a lot, employee work schedules).

- **Database design**

- Database Design is an important part of most applications.

- The first part of database design is to decide what kind of database the program will need.

- We need to specify whether the application will store data in text files, XML files, a full-fledged relational database.

- **Audit Trails**
- An audit trail keeps track of each user who modifies a specific record.
- Auditing can be as simple as creating a history table that records a user's name, a link to the record that was modified, and the date when the change occurred.

- **User Access**
- Provides different levels of access to different kinds of data.
- Build a table listing the users and the privileges they should be given. The program can then disable or remove the buttons and menu items that a particular user shouldn't be allowed to use.

- **Database Maintenance**

- If we use audit trails and the records require a lot of changes, the database will start to fill up with old versions of records that have been modified.

- Even if we don't use audit trails, over time the database can become cluttered with outdated records.

- We can move the older data into a data warehouse, a secondary database that holds older data for analysis.

Data flows and States

- In order to describe the system and the way processes interact with the data, many data flows and state transition diagrams are used.

- A data flow diagram shows how data such as a customer order flows through various processes.

Training

- Deciding whether users want to attend courses taught by instructors, read printed manuals, watch instructional videos, or browse documentation online.

LOW LEVEL DESIGN

- Provide extra details that is necessary before developers can start writing code.
- Gives more specific guidance for how various parts of the system interact.
- Refines the definition of database, major classes and internal & external interfaces.

High-level design focuses on **what**.

Low-level design begins to focus on **how.**

- There are two important factors in the low-level design.

- They are:

- **Object-oriented design**
- **Database design**

OO Design

- Refining the classes that are identified by HLD.

  - **Identifying classes**

  - **Building inheritance hierarchies**

  - **Refinement**

    It is the process of breaking a parent class into multiple subclasses to capture some difference between objects in the class.

    Eg: parent class is “car”. When we derive the mercedez, range rover, Toyota, Ford, from super class car, we call it refinement.

- **Generalization**:

Refinement starts with a single class and creates child classes to represent differences between objects.

Generalization does the opposite:

It starts with several classes and creates a parent for them to represent common features.

Eg: car can be of two types Automatic and Manual – but both having some common features.

Database Design

Database design determines what tables the database contains and how they are related.

There are different kinds of databases which stores hierarchical data, documents, graphs and networks, key/value pairs, and objects.

However, the most popular kind of databases are relational databases.

Relational database

- A relational database stores related data in tables.
- Each table holds records that contain pieces of data that are related.
- Sometimes records are called tuples to emphasize that they contain a set of related values.
- The pieces of data in each record are called fields.
- Each field has a name and a data type.

- Database should be normalized.

- Database normalization is a process of rearranging a database to put it into a standard (normal) form that prevents different kinds of anomalies.

- If not normalized?

- Duplicate data
- We may be unable to delete a particular piece of data
- Unnecessary data may exist – waste space