System Design

- **Context** − The context of a system has a static and a dynamic part. The static context of the system is designed using a block diagram of the whole system which is expanded into a hierarchy of subsystems. The subsystem model is represented by UML packages. The dynamic context describes how the system interacts with its environment. It is modelled using **use case diagrams**.
- **System Architecture** − The system architecture is designed on the basis of the context of the system in accordance with the principles of architectural design as well as domain knowledge. Typically, a system is partitioned into layers and each layer is decomposed to form the subsystems.

Object-Oriented Decomposition

- Object-oriented decomposition identifies individual autonomous objects in a system and the communication among these objects.
- Advantages −
  - The individual components are of lesser complexity, and so more understandable and manageable.
  - It enables division of workforce having specialized skills.
  - It allows subsystems to be replaced or modified without affecting other subsystems.

Identifying Concurrency

- Concurrency is identified and represented in the dynamic model.
- Each concurrent element is assigned a separate thread of control.
- Concurrency is associated with the problems of data integrity, deadlock, and starvation.
- It is to be identified at the design stage.

Identifying Patterns

- Some commonly accepted solutions are adopted for some categories of problems
- A pattern can be defined as a documented set of building blocks that can be used in certain types of application development problems.
- Commonly used design patterns
  - Façade pattern
  - Model view separation pattern
  - Observer pattern
  - Model view controller pattern
  - Publish subscribe pattern
  - Proxy pattern

Controlling Events

- An event is a specification of a significant occurrence that has a location in time and space.
- Four types of events
  - Signal Event − A named object thrown by one object and caught by another object.
  - Call Event − A synchronous event representing dispatch of an operation.
  - Time Event − An event representing passage of time.
  - Change Event − An event representing change in state.

Handling Boundary Conditions

- The start–up of the system, i.e., the transition of the system from non-initialized state to steady state.
- The termination of the system, i.e., the closing of all running threads, cleaning up of resources, and the messages to be sent.
- The initial configuration of the system and the reconfiguration of the system when needed.
- Foreseeing failures or undesired termination of the system.
- Boundary conditions are modelled using boundary use cases.

Object Design

- Object design includes the following phases,
  - Object identification
  - Object representation, i.e., construction of design models
  - Classification of operations
  - Algorithm design
  - Design of relationships
  - Implementation of control for external interactions
  - Package classes and associations into modules

Object Identification

- The functions of this stage are
  - Identifying and refining the classes in each subsystem or package
  - Defining the links and associations between the classes
  - Designing the hierarchical associations among the classes, i.e., the generalization/specialization and inheritances
  - Designing aggregations

Object Representation

- This stage involves constructing UML diagrams.
- Static Models − To describe the static structure of a system using class diagrams and object diagrams.
- Dynamic Models − To describe the dynamic structure of a system and show the interaction between classes using interaction diagrams and state–chart diagrams.

Classification of Operations

- The operation to be performed on objects are defined by combining the three models, object model, dynamic model, and functional model.
- An operation specifies what is to be done and not how it should be done.
- The tasks to be performed are
  - The state transition diagram of each object in the system
  - Operations are defined for the events received by the objects.
  - Cases in which one event triggers other events in same or different objects are identified.
  - The sub–operations within the actions are identified.
  - The main actions are expanded to data flow diagrams.

Algorithm Design

- Algorithms focus on how it is to be done.
- Optimal algorithm is selected for the given problem domain. The metrics for choosing the optimal algorithm are −
  - **Computational Complexity** − Complexity determines the efficiency of an algorithm in terms of computation time and memory requirements.
  - **Flexibility** − It determines whether the chosen algorithm can be implemented suitably, without loss of appropriateness in various environments.
  - **Understandability** − This determines whether the chosen algorithm is easy to understand and implement.

Design of Relationships

- Regarding associations −
  - Identify whether an association is unidirectional or bidirectional.
  - Analyse the path of associations and update them if necessary.
  - Implement the associations as a distinct object, in case of many–to-many relationships; or as a link to other object in case of one–to-one or one–to-many relationships.
- Regarding inheritances,
  - Adjust the classes and their associations.
  - Identify abstract classes.
  - Make provisions so that behaviours are shared when needed.

Implementation of Control

- In system design, a basic strategy for realizing the dynamic model is made.
- The approaches for implementation of the dynamic model are −
  - **Represent State as a Location within a Program** − In this approach, the location of control defines the program state. A finite state machine can be implemented as a program.
  - **State Machine Engine** − This approach directly represents a state machine through a state machine engine class. This class executes the state machine through a set of transitions and actions provided by the application.
  - **Control as Concurrent Tasks** − In this approach, an object is implemented as a task in the programming language or the operating system. Here, an event is implemented as an inter-task call. It preserves inherent concurrency of real objects.

Packaging Classes

- The different aspects of packaging
  - **Hiding Internal Information from Outside View** − It allows a class to be viewed as a "black box" and permits class implementation to be changed without requiring any clients of the class to modify code.
  - **Coherence of Elements** − An element, such as a class, an operation, or a module, is coherent if it is organized on a consistent plan and all its parts are intrinsically related so that they serve a common goal.
  - **Construction of Physical Modules** − The following guidelines help while constructing physical modules −
    - Classes in a module should represent similar things or components in the same composite object.
    - Closely connected classes should be in the same module.
    - Unconnected or weakly connected classes should be placed in separate modules.
    - Modules should have good cohesion, i.e., high cooperation among its components.
    - A module should have low coupling with other modules, i.e., interaction or interdependence between modules should be minimum.

Design Optimization

- Add redundant associations
- Omit non-usable associations
- Optimization of algorithms
  - Rearrangement of the order of computational tasks
  - Reversal of execution order of loops from that laid down in the functional model
  - Removal of dead paths within the algorithm
- Save derived attributes to avoid re-computation of complex expressions
  - With each update of the base attribute value, the derived attribute is also re-computed.
  - All the derived attributes are re-computed and updated periodically in a group rather than after each update.

Design Documentation

- Usage Areas
- Contents
  - High–level system architecture − Process diagrams and module diagrams
  - Key abstractions and mechanisms − Class diagrams and object diagrams.
  - Scenarios − Behavioural diagrams
- Features
  - Concise and at the same time, unambiguous, consistent, and complete
  - Traceable to the system's requirement specifications
  - Well-structured
  - Diagrammatic instead of descriptive