Architecture Design

Engineering design is the use of scientific principles and technical information in the creative development of a plan to bring about a man-made product to achieve a prescribed goal with certain specified constraints.
Software design is the process by which an agent creates a specification of a software artifact, intended to accomplish goals, using a set of primitive components and subject to constraints.

Why Design is important?
Design is the first stage in development errors in this stage will be costly
Design affect quality of product

Subtasks of Software Design
Architecture design
Architecture design determines how a software system is decomposed into components and how these components are interconnected.

Interface design
Interface design determines how the software interacts with its environment, such as human computer interactions (HCI).

Detailed design
Detail design determines the details of the implementation, such as the algorithms and data structures for implementing each functionality and component, and the programming language for coding.

Basic Elements of Design
01 .Statements of design problems and objectives
02.Constraints
03.Description of product
04.Rationale
05.Plan of production
06.Description of usage

Characteristics of Wicked or ill-structured problems
01 .No definitive formulation of the problem
02.No definitive solution to the problem
03.No definitve way of solving the problem

Factors affect designs
01 .Principle of Totality
02.Principle of Time
03.Principle of Value
04.Principle of Resources
05.Principle of Synthesis
06.Principle of Iteration
07.Principle of Change
08.Principle of Relationships
09.Principle of Competence
1 0.Principle of Service

1.Functional Requirements
Functional requirements are Statements of services that the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
2.Non Functional Requirements
Non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors.
Efficiency
Effectiveness
Capacity, current and forecast
Compliance
Documentation
Extensibility
Fault tolerance
Interoperability
Maintainability
Testability
Security

Views of Quality
01 .Transcendental view : something that can be recognized but not defined
02.User's point of view : fitness for purpose
03.value-based view : ability to provide what the customer requires at a price that they can afford
04.manufacturing point of view: conformance to specification
05.Product view

Key quality attributes of a software

Efficiency
Correctness and reliability
Portability
Maintainability
Reusability
Interoperability

Efficiency refers to the responsiveness of the system
.
Communication between components
Computation times of components
Functionality assigned to the components
Correctness is the property that software implements the specified users'requirements.
Reliability is the probability that a system performs user required functionality correctly at a specified environment in a given period of time.

Reliability can be measured in:
Mean time between failures
The probability of failure on demand
Availability

The effect of design on Reliability
A good architectural design
Good HCI design

Detailed design
Portability is the property of a software system that can be easily transported from one hardware/software platform to another.
Maintainability refers to the easiness of maintaining a software system.
Corrective maintenance: Modification of the system for correction of bugs.
Adaptive maintenance: Modification of the system due to environment changes.
Reusability is the property of a software system that its components can be easily reused in the development of other software systems.
Interoperability is the property of how easy a software system can be used with other software systems.

Design Objectives
Modularity
Portability
Malleability
Conceptual integrity

Requirements for good designs
Well structured
Simple
Efficient
Adequate
Flexible
Practical
Implementable
Standardised

Software quality models
Hierarchical models (McCall's Model)
Relational models (Perry's relational model)

McCall identified three main perspectives for characterizing the quality attributes of a software product.
Product revision (ability to change).
Product transition (adaptability to new environments).
Product operations (basic operational characteristics).

Brooks' four essences of difficulties of software development
Complexity
Conformity
Changeability
Invisibility

Vehicles to overcome difficulties
The Axiom of Separation of Concerns
The Axiom of Comprehension
The Axiom of Translation
The Axiom of Transformation

Basic rules of software design

The principle of modular designs
The principle of portable designs
The principle of malleable designs
The principle of intellectual control
The Principle of visualization

Waterfall Software Process

Requirements
Design
Implementation
Verification
Maintenance
V Model
Spiral Model

Design strategies: Decompositional methods
Compositional methods
Template-based
Evolutionary strategies

Benefits of the client-server architecture
Higher security
Centralized data access
Ease of maintenance

Disadvantages of Client–server architecture
Tendency for application data and business logic to be closely combined on the server.
Scalability issues
Its dependence on a central server, which can negatively impact system reliability.

Benefits of the Three-tier/N-tier Architecture
Maintainability
Scalability
Flexibility
Flexibility
Availability

Data-centered architecture - Advantages
Data-centered architectures promote integrability

Data-flow architectures
Unix Shell Scripts
Traditional Compilers

Pipe and Filter Advantages
Easy to understand the overall input/output behavior of a system as a simple composition of the behaviors of the individual filters.

Encourages reuse of filters.
Systems can be easily maintained and enhanced, since new filters can be added to existing systems and old filters can be replaced by improved ones.
Often leads to batch-type processing
Not good for interactive applications where you often want to do incremental computations, e.g., incremental display updates
Data transmission critical for system performance

Call and return architectures
The components of a main program/subprogram architecture are distributed across multiple computers on a network.

Component-Based Architectural Style
A component is a software object, meant to interact with other components, encapsulating certain functionality or a set of functionalities
.
Key features of a component
Reusable.
Replaceable.
Extensible.
Encapsulated.
Independent.

Benefits of the component-based architectural style
Ease of deployment
Reduced cost
Ease of development
Reusable

Layered Architectural Style
Layered architecture focuses on the grouping of related functionality within an application into distinct layers that are stacked vertically on top of each other. Functionality within each layer is related by a common role or responsibility.
Tiers indicate a physical separation of components, which may mean different assemblies on the same server or multiple servers.
Layers refers to a logical separation of components, such as having distinct namespaces and classes for the Database Access Layer (DAL), Business Logic Layer (BLL) and User Interface Layer (UIL).

Advantages of the layered architectural style
Abstraction
Isolation
Manageability
Performance
Reusability
Testability

Service-Oriented Architectural Style
Service-oriented architecture (SOA) enables application functionality to be provided as a set of services, and the creation of applications that make use of software services.

Properties of a service
It logically represents a business activity with a specified outcome.
It is self-contained.
It is a black box for its consumers.
It may consist of other underlying services.

The key principles of the SOA architectural style
Services are autonomous
Services are distributable
Services are loosely coupled

A WSDL (Web Services Definition Language) is an XML document that describes a web service.
SOAP (Simple Object Access Protocol) is a protocol specification for exchanging structured information in the implementation of web services.
Representational state transfer (REST) or RESTful web services defines a set of functions which developers can perform requests and receive responsesvia HTTP protocol such as GET and POST

Benefits of the SOA architectural style
Abstraction
Discoverability
Interoperability

Microservices architecture
The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.

Limitations of Monolithic Architecture

Monolithic applications can evolve into a "big ball of mud"; a situation where no single developer (or group of developers) understands the entirety of the application.
Scaling monolithic applications can often be a challenge.
Limited re-use is realized across monolithic applications.
By definition monolithic applications are implemented using a single development stack (ie, JEE or.NET), which can limit the availability of "theright tool for the job."

Advantages of Microservices architecture

Services are encouraged to be small, ideally built by a handful of developers.
If microservices' interfaces are exposed with a standard protocol, such as a REST-ful API, they can be consumed and reused by other services and applications without direct coupling through language bindings or shared libraries.

Services exist as independent deployment artifacts and can be scaled independently of other services.

Developing services discretely allows developers to use the appropriate development framework for the task at hand.The tradeoffs of microservices architecture vs monolithic architecture

Types of Design Patterns
Creational Patterns Provide a way to create objects while hiding the creation logic
01 .Singleton Pattern
-used when only one instance of an object is needed throughout the lifetime of an application.(E.g: Clip Board,Windows Registry)
02.Factory Pattern
-Creates objects without exposing the instantiation logic to the client and Refers to the newly created object through a common interface.
03,Builder pattern
-Instead of using numerous constructors, the builder pattern
Builder Pattern is used when:
-The creation algorithm of a complex object is independent from the parts that actually compose the object
-The system needs to allow different representations for the objects that are being built

Structural Patterns Concerned with class and object composition.
01 .Decorator pattern
-Decorator pattern allows a user to add new functionality to an existing object without altering its structure.
(E.g:FileInputStream,FileOutputStream,BufferedReader)
02.Adapter pattern
-Convert the interface of a class into another interface clients expect

Behavioral Patterns Concerned with communication between objects.
01 .Observer Design pattern
-Define one to many dependency between objects
02.Strategy pattern
-Enables selecting an algorithm at runtime. The strategy pattern