Mid 1

Architectural Design

- This is the process of designing the overall structure and behavior of a software system.

- It involves breaking down the system into smaller parts (components) and specifying how these parts are connected (connectors) to achieve the system's objectives.

- The design should consider system requirements, contextual information, and user stories to ensure that the resulting system meets its intended purpose.

Components

- These are the building blocks of a software system.

- Components encapsulate specific functionality or data and provide a well-defined interface for interacting with them.

- They are designed to be reusable and provide application-specific services.

**Connectors**

- These are the "glue" that holds the components together.

- Connectors facilitate interactions between components, allowing them to communicate and share data.

- They can be simple, like procedure call connectors, or more complex, like distribution connectors for distributed systems.

**Configuration:**

- This refers to how the components and connectors are arranged and connected in a software system's architecture.

- The configuration is designed to achieve the system's objectives and can vary depending on the specific requirements and constraints of the system.

In summary, architectural design in software architecture involves designing the overall structure and behavior of a software system by breaking it down into components and connectors, arranging them in a specific configuration to achieve the system's objectives. Components encapsulate functionality or data, while connectors facilitate interactions between components.

Sure, let's summarize the content you provided:

\*\*Fundamentals of Architectural Styles and Patterns\*\*:

- Architectural Styles: These are collections of design decisions that are applicable in a given development context and help in achieving beneficial qualities in each resulting system.

- Architectural Patterns: These are sets of design decisions that are applicable to recurring design problems and can be adapted to different software development contexts.

\*\*Architectural Styles vs Patterns\*\*:

- Scope: Architectural styles apply to development contexts, while patterns apply to specific design problems.

- Abstraction: Styles are more abstract and general, while patterns are more specific and concrete.

- Relationship: A system designed according to a style may involve the use of multiple patterns, and a single pattern can be applied to systems designed according to multiple styles.

\*\*Architectural Patterns\*\*:

- State-Logic-Display: Used in business applications, multiplayer games, and web applications.

- Model-View-Controller (MVC): Separates information, presentation, and user interaction.

- Sense-Compute-Control (SCC): Dynamically transforms data into processing components.

\*\*Architectural Styles Fundamentals\*\*:

- Vocabulary of design elements: Components, connectors, data elements.

- Set of configuration rules: Topological constraints.

- Semantic interpretation: Compositions of design elements have well-defined meanings.

\*\*Architectural Styles Fundamentals -- Benefits\*\*:

- Design reuse: Well-understood solutions applied to new problems.

- Code reuse: Shared implementations of invariant aspects of a style.

- Understandability of system organization: A phrase such as “client-server” conveys a lot of information.

- Interoperability: Supported by style standardization.

\*\*Architectural Styles Fundamentals -- Types\*\*:

- Traditional, language-influenced styles: Main program and subroutines, Object-oriented.

- Layered: Virtual machines, Client-server.

- Data-flow styles: Batch sequential, Pipe and filter.

- Shared memory: Blackboard, Rule-based.

- Interpreter: Interpreter, Mobile code.

- Implicit invocation: Event-based, Publish-subscribe.

- Peer-to-peer.

- “Derived” styles: C2, CORBA.

\*\*Architectural Styles -- Traditional\*\*:

- Main program and subroutines: Instantly familiar to anyone who has programmed in a language such as C.

- Object-oriented: Components are objects (instances of a class), and connectors are method invocations.

- Layered: Consists of ordered sequence of layers, each offering a set of services that may be accessed by programs residing within the layer above it.

- Client-Server: Two-level topology with multiple clients making requests to the server.

\*\*Architectural Styles -- Data Flow\*\*:

- Batch Sequential: Separate programs are executed in order, and data is passed as an aggregate from one program to the next.

- Pipe-and-Filter: Separate programs are executed, potentially concurrently, and data is passed as a stream from one program to the next.

- Shared Memory -- Blackboard: Independent programs access and communicate exclusively through a global data repository, known as a blackboard.

- Shared Memory -- Rule-based: Inference engine parses user input and determines whether it is a fact/rule or a query.

\*\*Architectural Styles -- Interpreter\*\*:

- Interpreter: Parses and executes input commands, updating the state maintained by the interpreter.

- Mobile-Code Style: A data element is dynamically transformed into a data processing component.

\*\*Architectural Styles -- Implicit Invocation\*\*:

- Publisher-Subscribe: Subscribers register/deregister to receive specific messages or specific content.

- Event-based: Independent components asynchronously emit and receive events communicated over event buses.

\*\*Architectural Styles -- Peer-to-Peer\*\*:

- Peer-to-Peer: State and behavior are distributed among peers, which can act as either clients or servers.

\*\*Architectural Styles -- REST\*\*:

- REST: Representation State Transfer Style. The key abstraction of information is a resource, named by an URL. All interactions are context-free.

\*\*Architectural Styles -- REST Components\*\*:

- User agent, Origin server, Proxy, Gateway.

\*\*Architectural Styles -- REST Connectors\*\*:

- Modern Web Examples: Client libwww, Server libwww, Cache browser cache, Resolver bind, Tunnel SOCKS.

In summary, architectural styles and patterns are collections of design decisions that help in achieving beneficial qualities in each resulting system. Architectural patterns are applicable to recurring design problems, while architectural styles apply to development contexts. Architectural styles and patterns can be used together to design and implement software systems effectively.

Software architecture is a crucial aspect of software development, as it provides a blueprint for the overall structure, modules, and interactions of a software system. It involves making high-level design decisions that address non-functional requirements such as reliability and performance. Architecture is difficult to change once implemented, so it's important to get it right from the start.

Software design, on the other hand, is concerned with the detailed interfaces, algorithms, and procedures of the design elements. It focuses on the functional requirements and is more flexible than architecture, as it can be modified to accommodate changes in requirements.

The architecture of the World Wide Web (WWW) is a good example of how architecture is separate from the code. There is no single piece of code that implements the architecture of the web; instead, there are multiple pieces of code that implement various components of the architecture.

In summary, software architecture provides a high-level blueprint for a software system, while software design focuses on the detailed implementation of that blueprint. Architecture is difficult to change once implemented, while design is more flexible and can be modified to accommodate changes in requirements.