**What is Software Architecture:**

Software Companies have business goals and architecture can be understood as a bridge between those business goals and the system to be built.

The software architecture is a set of structures that describe a system and consists of software elements, relations among them and properties of both.

**Architecture is a set of software structures:**

Structures is a set of elements held together by a relation. There are three categories of architectural structures:

* Module Decomposition Structures: Modules can be understood as specific responsibilities that are assigned to respective teams based on their capabilities. For example team A might be working on the database side of things, Team B might be handling the business side, Team C might be handling the programming side. In larger projects, these elements are then subdivided into even smaller teams. One kind of modular structure comes from class diagrams if the modules in that are aggregated to layers.
* Component and Connector Structure: This is concerned with how elements interact with each other at runtime to carry out a specific task. Consists of services, the infrastructure that these services will interact with and the interaction relations between the services and infrastructure are all part of this.
* Allocation Structures: Concerned with the mapping of structures to different environments of the system. Modules are divided among teams and assigned to different locations in a file structure for implementation, integration and testing. This deployment of components onto hardware is called allocation structures.

A structure is architectural only if it is concerned with any reasoning about the system, for example responsiveness of the system to user requests, how the system behaves in case of an error. The reasoning can be considered as an attribute that is important to some stakeholder.

**Architecture is an abstraction:**

Architecture comprises of elements and the interaction between these elements. Elements interact with each other through partitions that divide the elements into their public and private parts. Private parts are those that are concerned with the internal implementation of an element and hence is not considered a part of the architecture. So only the public part is take into consideration here.

**Every System has a software architecture:**

A software architecture and its representation of this architecture are two different things. Perhaps the software was built a long time ago and the people who built it are long gone and all the documents have been lost. This does not mean that the software architecture is not there; it just means that the documentation has been lost and all we have left is the executing code. Hence the software architecture is not known to anyone.

**Architecture includes behavior:**

Behavior includes how elements interact with each other. This means that the box and line drawings that look like part of architectures are actually not. When looking at these boxes(database, GUI) one may imagine their behaviour but this information stems from the reader’s mind and not from information that has been explicitly stated.

System Architecture:

Concerned with the mapping of functionality onto hardware and software components, mapping of software architecture onto hardware architecture and interaction of humans with these components. For example a system architecture might define the functionality of processors and the type of network that connects these processors, this helps to reason about the system in terms of performance. Similarly the software architecture will define how exactly this functionality is implemented and how the processors exchange messages in the form of signals; this helps to reason about the system in terms of power consumption etc.

Enterprise Architecture:

Concerned with how an enterprise’s software systems are in line with its business processes and goals. How the software are used by business processes, the standards that determine the computational environment and the software itself are all part of this architecture. So the software infrastructure that supports communication between systems and with the external world is considered a part of the enterprise architecture.

**Structure and View:**

Structure is a set of elements. View is a representation of a coherent set of elements. In short, view is a representation of structure.

**Architectural Structures:**

**Useful module Structures:**

Decomposition Structure: The units are modules which are related using a is-a-submodule-of relation where each module is decomposed into smaller modules recursively until the modules are easily understood. It determines system’s modifiability by ensuring that likely changes are localized.

Uses Structure: The units are modules/classes. It is defined by a form of dependency called uses. A unit uses another unit if the correctness of the first unit is dependent upon the presence of a correctly working version of the second. Used in systems that can be extended to add functionality.

Layer Structure: The units are called layers. Layer is a cohesive set of services managed using an interface. It is managed using a very strict system in which a layer is only allowed to user the layer directly below it. It promotes portability.

Class/Generalization Structure: The units are classes. The relationship is is-an-instance-of or inherits-from. Follows object oriented method which supports collections of similar attributes and parametrized differences. Promotes reusability and incrementation.

Data Model: The units are data entities. Static information is displayed as these entities and the relationship between these entities. An entity can be an account and an account can have an attribute of account number. A relationship can dictate that one customer can have multiple accounts using a one-to-many relationship.

**C & C Structures:**

The relationship in all of these is attachment.

Service Structure: The units are services that interoperate with each other using service coordination mechanisms such as SOAP. Helpful in developing systems that had its components developed independent of each other.

Concurrency Structure: The units are components. The components are arranged into logical threads which can be described as a sequence of computations that could be allocated to a separate physical thread later in the design process. Promotes parallelism and allows architects to discover points of contention where concurrent execution might be a problem.

**Allocation Structures:**

Deployment Structure: The elements are software elements (a process from C & C), hardware entities (processor) and communication pathways. Relations are allocated-to which shows the mapping of software elements onto hardware, and migrates-to if the allocation is dynamic. Helps to reason about performance, data integrity, security.

Implementation Structure: Units are modules. Shows how software elements are mapped to the file structures in different environments of the system.

Work Assignment Structures: Assigns responsibility for distributing the modules to the teams who will make use of them. The architect knows the level of expertise on each team and arranges the units of commonality to a single team rather than distributing them to anyone who needs them.

**Relating Structures to Each other**

Even though these structures deal with different perspectives of the system, they are not independent. Elements in one structure might be used by elements of another structure.

**Fewer is better**

The larger the system, the more the structures and the bigger the differences between its structures. A structure should only be built if doing so brings about a positive return in terms of investment which is brought about by low development and maintenance costs.

**Which Structures to choose:**

When choosing a structure, the most desirable quality attributes must be thought about and then a structure which will be the most advantageous to deliver those quality attributes must be chosen.

**Architectural Patterns:**

Composition of architectural elements are called patterns.

**Module type pattern:**

Layered Pattern: When the uses relation between software elements is strictly uni-directional a pattern of layer is formed. Layers can only use the layers below them so the layers above are protected from the layers below them which supports the idea of the relationship being uni directional.

**C & C type patterns:**

Shared data/repository pattern: Takes the form of a database. The connectors are protocols for managing the data such as SQL. Consists of C&C that create, store and access persistent data.

Client Server Pattern: Components are clients and servers and connectors are the protocols and messages that they share amongst each other.

**Allocation type patterns:**

Multi-tier pattern: Describes how components of a system are distributed in specific subsets of hardware and software connected by a medium.

Competence center and platform: In competence center, work is allocated to a site based upon the expertise relating to that work at that site. In platform, one site is tasked to produce reusable core assets while the other sites develop applications that uses these assets.

**What makes a good architecture:**

**Process Recommendations:**

* The architecture should be made by a single architect or by a small group with a particular leader. Promotes conceptual integrity and consistency.
* The architecture should be based upon a list of clearly defined quality attribute requirements
* The architecture should be documented using views and these views should solve the concerns of the stakeholders
* During the early life cycle, the architecture should be evaluated to see its ability to deliver the quality attributes.
* The architecture should allow incrementation so that every thing does not need to be integrated at once.

**Structural Recommendations:**

* The architecture should have modules which are based upon functionalities with respect to separation of concerns and information hiding
* The quality attributes should be achieved using well known patterns and tactics
* The architecture should not depend on a particular version of a product or tool
* Modules that consume data should be separate from modules that produce data.
* Correspondence between modules and components must not be one-one
* There should be a small number of ways for components to interact with each other
* There should be specific set of areas for resource contention.