**Software Architecture**

**Introduction:**

The software architecture of a system depicts the system’s organization or structure, and provides an explanation of how it behaves. A system represents the collection of components that accomplish a specific function or set of functions. In other words, the [software architecture](https://en.wikipedia.org/wiki/Software_architecture) provides a sturdy foundation on which software can be built.

A series of architecture decisions and trade-offs impact quality, performance, maintainability, and overall success of the system. Failing to consider common problems and long-term consequences can put your system at risk.

There are multiple high-level architecture patterns and principles commonly used in modern systems. These are often referred to as architectural styles. The architecture of a software system is rarely limited to a single architectural style. Instead, a combination of styles often make up the complete system.

**Benefits Of Software Architecture:**

1. **Higher productivity:** It is easier to add new features to existing software, since the structure is already in place, and the location for every new piece of code is known beforehand.
2. **Better code maintainability:** It is easier to maintain software based on an architecture, as the structure of the code is visible and known, so it’s easier to extend the software or find bugs and anomalies.
3. **Higher adaptability:** New features, such as a different front end, or adding a process rule are easier to achieve, as the software architecture creates a clear separation of concerns.
4. **Quality:** More reliable assessment of system quality attributes like performance, security, interoperability, reliability, availability.

**Importance Of Software Architecture:**

**Meeting the Requirements:**

A software architecture comprises information from various stakeholders such as domain experts, business analysts, product owners, and end-users. This information helps you identify and meet different functional, non-functional, technical, and operational requirements. A [successful requirements management](https://www.kovair.com/blog/successful-requirements-management-eliminates-project-defects/) can help you eliminate many project defects.

**Ensuring Quality:**

Software architecture can be designed to focus on specific quality attributes of a system such as performance, features, security, and interoperability. Generally, these quality attributes do not always stay in accordance with one another.

A software architecture establishes an agreed-upon and validated quality requirements and standards for the products. It also lets you predict a software system’s qualities and avoid costly rework.

**Facilitating Communication among Stakeholders:**

Software architecture and its documentation are simple and comprehensive enough that any stakeholders can reason about the software system. It lets you communicate and explain the software system to others. It can be a basis for discussions and negotiations regarding various aspects of a project such as cost, quality and duration.

**Embracing Change:**

There can be many changes in a software system such as new requirements, market changes, changes to business processes, bug fixes,  technology advances, and many more; especially in the modern [agile development process](https://www.kovair.com/blog/how-enterprises-can-successfully-scale-agile-development-process/) change is the only constant.  Good software architecture can help the team anticipate and adapt to these changes without necessarily having to make architectural changes.

**Providing a Reusable Model:**

The code and early decisions that shaped the architecture are reusable for projects that have similar requirements and structures. Not only does this save us a lot of time and effort, but this tested and proven architecture also ensures and increases the quality of products.

**Estimating Cost and Effort:**

The design of the software architecture itself affects the kind of tasks necessary for the implementation. In this way, the project managers can break down the work as individual tasks based on the nature and size of the project.

The project managers break down final deliverables and goals into smaller packages of work. And the developers initially start with specific tasks and then group them into packages of work. By reducing these complexities, we can achieve more accurate cost and effort estimates.

**2- Architectural pattern**

Model View Controller (MVC): The **MVC pattern** suggests splitting the code into 3 components (Model, View, Controller).

MVC patterns separate the input, processing, and output of an application. This model divided into three interconnected parts called the model, the view, and the controller.

We decided to use MVC for its many benefits that helps us as developers, it provides

Faster development process, and easier on testing and support

The **Model-View-Controller (MVC)** framework is an architectural pattern that separates an application into three main logical components Model, View, and Controller. Hence the abbreviation MVC. Each architecture component is built to handle specific development aspect of an application. MVC separates the business logic and presentation layer from each other. It was traditionally used for desktop graphical user interfaces (GUIs). Nowadays, MVC architecture has become popular for designing web applications as well as mobile apps

### Consists of subsystems classified into one of:

### Model

The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. For example, a Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.

### View

The View component is used for all the UI logic of the application. For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.

### Controller

Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. For example, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller will be used to view the Customer data

In this system the component-and-connector view type enables the representation of software architecture from the point of view of its components, the principal unit of runtime interaction or data storage, and its connectors, the interaction mechanism among components and the “data flow” among them.