**Patrones de diseño arquitectónico**

Como se componen, su estructura y como se relacionan sus elementos.

* **MVC** (Modelo – vista – controlador)

Modelo se encarga de los datos, notifica a las vistas de cambios. La vista consulta al modelo.

El controlador realiza los cambios

* El **modelo** contiene la información con la que el sistema trabaja, proporcionándosela a la vista para que pueda mostrarla y permitiendo realizar cambios en ella desde el controlador.
* El **controlador**responde a acciones del usuario, modificando el modelo cuando sea necesario. Además, se comunica con la vista para que se actualice con los últimos cambios del modelo.
* La **vista**presenta al usuario la información del modelo.

<https://ahorasomos.izertis.com/solidgear/componentes-de-arquitectura-de-android-de-mvc-a-mvvm/>

* **MVP:** El modelo es similar en todos los patrones de diseño teniendo los datos y la información de los estados de este. Este patrón es utilizado mayoritariamente para construir interfaces de usuario. El presentador asume como intermediario. La vista no sabe sobre el modelo.

La vista contiene los fragment.

El presentador provee los datos desde la vista al modelo y maneja tareas.

* **MVVM**

Model:

Una o mas clases de datos, Base de datos, u otras.

Proveedor de datos.

se encarga de los datos (acceso), no debe preocuparse por las vistas.

View model:

Capa intermedia que se comunica con modelo y con las vistas.

La vista notifica a view model, view model consigue los datos necesarios desde el modelo, una vez recibidos se los envía a la vista.

Obtener datos del modelo.

Intermediario entre la vista y el modelo, encapsula la lógica de la vista de los datos que produce el modelo.  
Controla las interacciones de las vistas (click en algún botón, cambio de textbook, deslizar un control, etc).

Traslada los datos de ingreso del usuario al modelo.

Validaciones intermedias.

Recupera y transforma los datos generados en el modelo y enlazarlos con la vista. Transformación de datos

Notificar a la vista de los cambios.

No debe existir código relacionado con las vistas.

Vista: producto que el usuario ve e interactúa, definir apariencia y estructura.

Un Activity o un fragment.

Se relaciona con el viewmodel a través de ***data bindings*** o enlaces de datos, la vista informa de cambios que hay en ella y el viewmodel comunica los cambios en el modelo o viewmodel.

el código de la vista es único a cada aplicación.

Beneficios: separación, desacople, pruebas unitarias.

Contras: requiere bastante código.