Para facilitar o desenvolvimento, a manutenção e a escalabilidade de sistemas complexos, padrões de arquitetura de software fornecem estruturas e princípios para organizá-los. Alguns dos padrões mais importantes incluem:

- O **modelo em camadas** organiza o sistema em várias camadas, cada uma com suas próprias responsabilidades (apresentação, negócios, etc.);

- **Cliente-servidor**: separa a lógica do sistema (servidor) da interface do usuário (cliente);

- A **SOA** organiza a aplicação como um conjunto independente de serviços. A aplicação é dividida em microserviços, que são ainda mais menores e mais independentes;

- **Baseada em Eventos**: Desconectando o sistema, os componentes reagem a eventos. Um evento é algo que aconteceu no sistema, como a criação de um novo usuário, a conclusão de um pedido ou uma alteração em um dado;

- **Microserviços**: dividem uma aplicação monolítica em pequenos serviços independentes, cada um responsável por uma única funcionalidade bem definida.

- **Arquitetura hexagonal**: visa separar a lógica de negócio de um sistema das suas interfaces externas, como bancos de dados, APIs e interfaces de usuário. A lógica de negócio reside no núcleo do sistema, enquanto as interfaces externas são conectadas através de "portas" e "adaptadores".

Padrões de Projeto Mobile são específicos para desenvolvimento de aplicativos móveis e visam organizar o código de forma eficiente. Os mais comuns são:

- **MVC:** Separa os dados (Model), a interface (View) e a lógica de controle (Controller):

  Model: Representa a lógica de negócios e os dados da aplicação. É sua responsabilidade administrar a persistência dos dados, as regras de negócios e as notificações de mudanças. Ao ocorrer uma alteração no Model, ele notifica a View para que seja atualizada.

  View: É a interface visual da aplicação que coleta entradas e apresenta os dados ao usuário. A View mostra apenas os dados do Model, sem lógica comercial.

Controller: Funciona como um intermediário entre a View e o Model. Ele coleta as informações do usuário, modifica o Model e avisa a View para que seja redesenhada.

- **MVP:** Similar ao MVC, mas com a lógica de apresentação centralizada no Presenter.

  Model: Mesma função do MVC.

  View: Similar ao MVC, mas geralmente possui uma interface mais simples, focando apenas na exibição dos dados.

  Presenter: Assume a maior parte da lógica de apresentação, que no MVC ficava distribuída entre o Controller e a View. Ele recebe as entradas do usuário, atualiza o Model e informa a View sobre as mudanças que devem ser feitas.

- **MVVM:** Utiliza databinding para conectar a View aos dados do ViewModel, simplificando a atualização da interface.

  Model: Mesma função do MVC e MVP.

  View: Utiliza um mecanismo de *data binding* para se conectar aos dados do ViewModel. Isso significa que a View é automaticamente atualizada quando os dados no ViewModel mudam.

  ViewModel: É uma representação da View no mundo do Model. Ele expõe os dados e os comandos necessários para a View, de forma a facilitar a ligação entre as duas.