Engenharia de Software

Arquiteturas de software

Marta Martinho 2020

Organização da apresentação

- Definição de arquitetura de software;
- Princípios da arquitetura de software;
- Design de software;
- O que pode influenciar na escolha de uma arquitetura de software;
- Importância da arquitetura de software;
- Design patterns;
- Arquitectural styles: Client-Server; Layers; Orientado a eventos e Orientado a serviços;
- Arquitectural patterns: Tree-tiers; MVC; MVVM e Microserviços;
- Anti-patterns.

Arquitetura de software

- Pode ser definida como um conjunto de elementos arquiteturais que possuem alguma organização;
- Os elementos e a sua organização são definidos por decisões tomadas para cobrir os objetivos e as restrições impostas ao desenvolvimento de uma aplicação de software;

Princípios da arquitetura de software

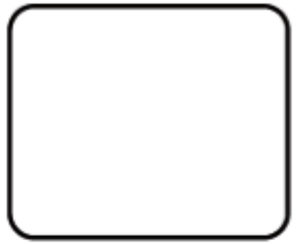
- **Abstração**: a modularização de um sistema deve ter em conta vários níveis de abstração, que podem ser definidos antes do início da implementação.
- **Acoplamento e coesão**: ser coeso significa que cada módulo deverá ter uma única responsabilidade. O acoplamento significa dependência entre módulos. Uma boa arquitetura deve ser coesa e de baixo acoplamento entre módulos.
- **Decomposição e modularização**: a partir de uma definição de alto nível de abstração, o problema vai-se decompondo de forma a ser obtido cada vez mais detalhe. A partir de um problema grande, recorrendo à modularização, este pode ser repartido em problemas menores, de forma a torná-lo mais simples.
- **Encapsulamento e omissão de informação**: a informação deverá estar disponível para módulos que efetivamente necessitem dela. O encapsulamento permite a quem use uma determinada classe, não saber muito acerca dela, simplesmente deve saber o que se espera da funcionalidade que a mesma oferece. Permite ainda que, caso uma classe seja alterada, não afete quem a está a usar.

Princípios da arquitetura de software

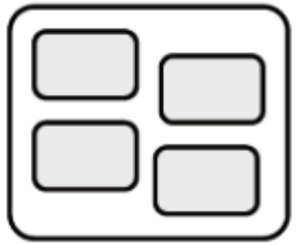
- **Separação de interface e implementação**: o conteúdo da implementação não deve estar disponível ao cliente.
- **Separação de assuntos**: dividir o software em módulos de forma a dividir as responsabilidades,
- **Necessário, integrável e primitivo**: as operações devem ser desenvolvidas facilmente, as características dos componentes devem ser apenas os necessários, mas completos.

Design

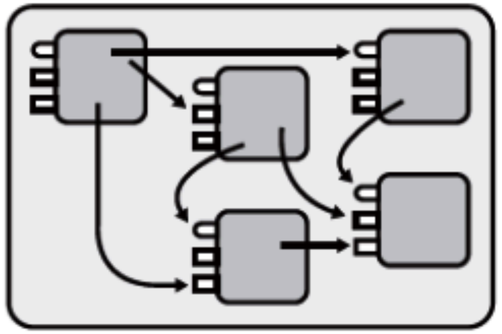
- Consiste na conceção de um esquema para transformar uma determinada especificação numa aplicação de software;
- A atividade de design liga os requisitos, ou seja as funcionalidades especificados, ao processo de desenvolvimento de código;
- Um bom design de alto nível providencia uma estrutura que pode conter múltiplos desings de mais baixo nível;
- Considera-se que um bom design é útil em projetos pequenos, mas indispensável em projetos grandes.



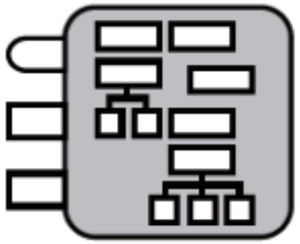
① Software system



② Division into subsystems/packages



③ Division into classes within packages

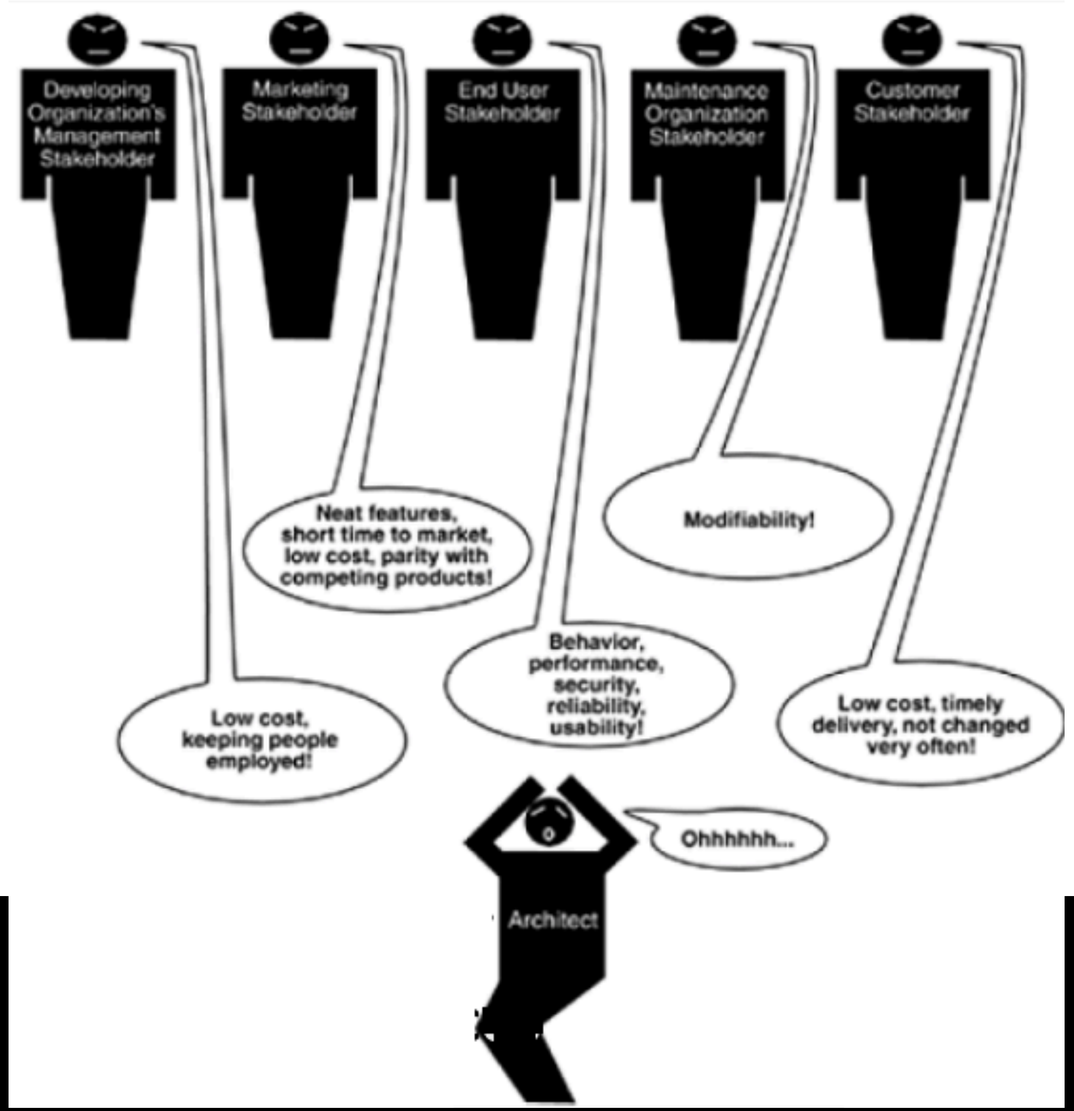


④ Division into data and routines within classes

⑤ Internal routine design

A arquitetura é influenciada por:

- Stakeholders;
- Organização do desenvolvimento;
- Background e experiência dos arquitetos;
- O ambiente tecnológico.



Arquitetura

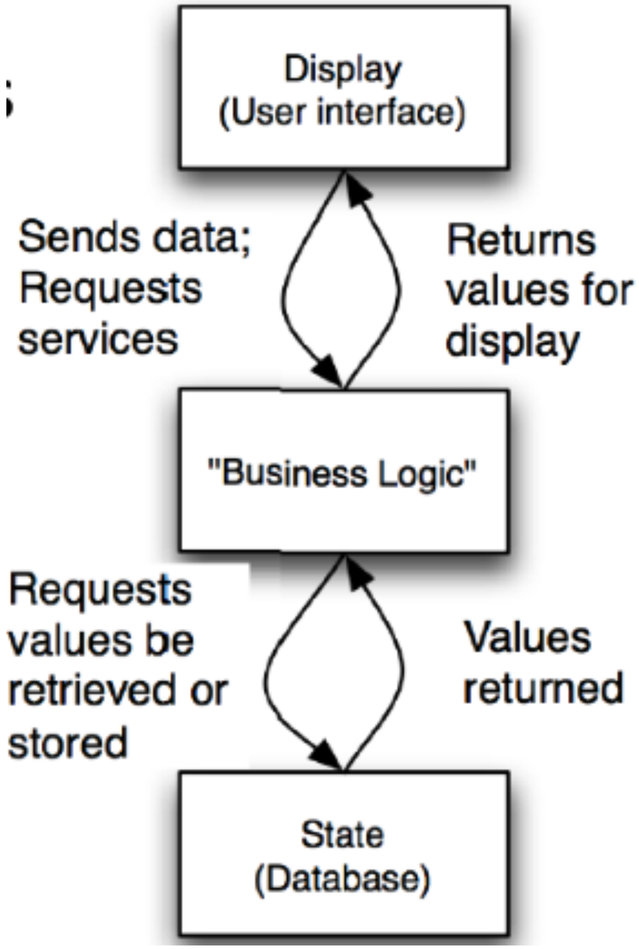
- É parte crucial do processo de desenvolvimento de software;
- Consiste na tomada de decisão precoce a nível de design de forma a dividir a aplicação a desenvolver em partes;
- A escolha de uma arquitetura não depende só do tipo de sistema a desenvolver em si, mas também do arquiteto, ou seja, os requisitos podem não influenciar a escolha da arquitetura;
- Se não for escolhida a arquitetura mais indicada, o resultado será um Big Ball of Mud - um sistema que carece de uma arquitetura que possa ser entendida;
- Isto acontece porque algumas equipas desenham à medida que implementam, sendo difícil, no final do processo de desenvolvimento, definir que arquitetura foi definida;
- Quanto maior for o problema, mais importância se deve dar às escolhas da arquitetura;

Arquitetura

- A arquitetura funciona como o “esqueleto” da aplicação, condicionando-o e afetando a qualidade dos atributos exigidos;
- A escolha da arquitetura tem grande impacto com a qualidade da aplicação a desenvolver;
- Cada componente tem um papel especifico no sistema, comunicando entre si de forma a providenciar a funcionalidade exigida;
- A escolha de uma correta arquitetura reduz o risco de falhas.

Importância da arquitetura de software

- Funciona como o “esqueleto” da aplicação:
    - de acordo com as funcionalidades, podemos dizer se uma arquitetura está bem ou mal desenhada.
- Habilita ou inibe os atributos de qualidade:
    - a arquitetura influencia a qualidade do software;
    - os programadores devem prestar atenção às funcionalidades do sistema para garantir a sua qualidade;
    - Podem ser habilitadas ou inibidas qualidades como segurança ou performance.



Importância da arquitetura de software

- Provoca restrições no sistema:
    - novos sistemas a serem desenvolvidos podem integrar ou incorporar sistema mais antigos, componentes desenvolvidos por outros programadores ou cumprir orçamentos;
    - As restrições impostas pela arquitetura pode dificultar o trabalho dos programadores.
- É transversal a toda a funcionalidade:
    - A escolha de uma arquitetura pobre pode comprometer a funcionalidade, fazendo com que os atributos de qualidade exigidos sejam difíceis de alcançar;
    - É possível escolher arquiteturas diferentes sem comprometer a funcionalidade e também é possível usar a mesma arquitetura em diferentes funcionalidades.
- Proporciona mais exatidão a nível de estimativas de custos e de tempos.

O que faz uma boa arquitetura de software

- Não existe uma arquitetura boa ou má;
- Uma arquitetura deve ser desenhada tendo em conta ou priorizando uma lista de atributos de qualidade exigidos;
- Uma arquitetura deve ser avaliada pela sua habilidade em conseguir garantir um sistema com qualidade;
- Uma boa arquitetura deve possibilitar o desenvolvimento incremental;
- A escolha de uma boa arquitetura gera código reutilizável, escalável e fácil de manter.

Exercício

Define os seguintes princípios de design de software

- abstração;
- Acoplamento e coesão;
- Decomposição e modularização;
- Encapsulamento e omissão de informação;
- Separação de interface e implementação;
- Necessário, integrável e primitivo;
- Separação de assuntos.

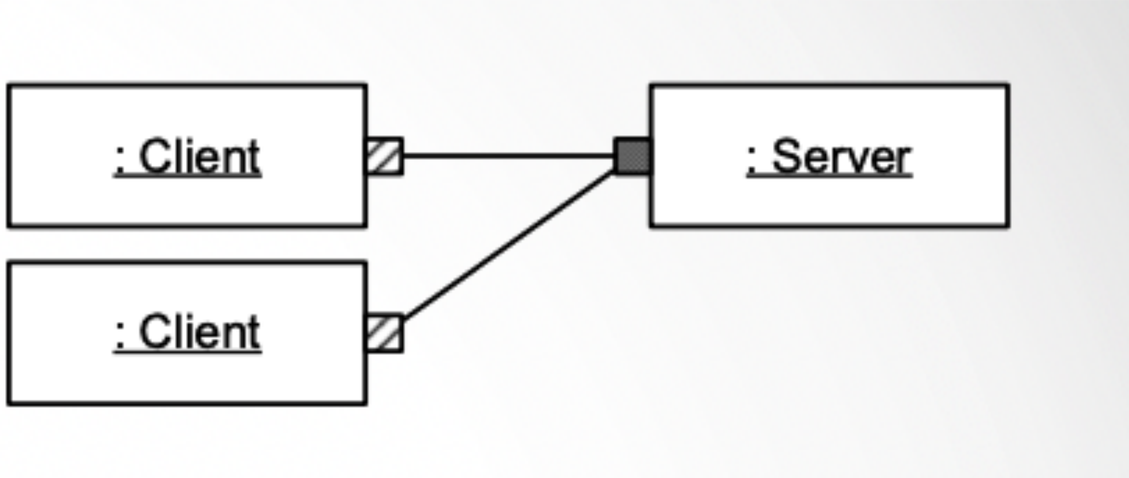
Indique as vantagens desses atributos.

Estilos de arquitetura de software

- É o mais alto nível de abstração a nível do design de uma aplicação
- De uma forma geral, os estilos de arquitetura de software, ditam como de deve organizar o código;
- É o mais alto nível de granulosidade especifica camadas e módulos de alto nível de uma aplicação;
- Especifica ainda a forma como essas camadas e módulos interagem entre si, ou seja, a relação entre eles;
- Um estilo de arquitetura de software pode ser implementado de várias formas, com estruturas ou práticas específicas;

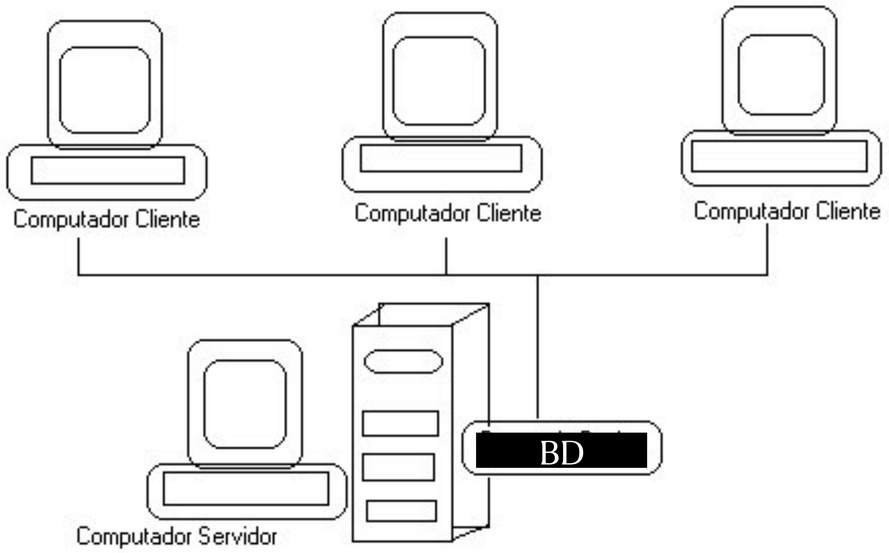
Client-Server

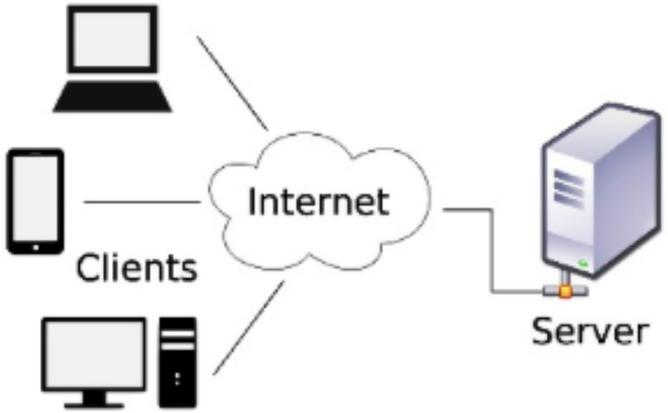
- Os clientes fazem pedidos a serviços do servidor;
- Só os clientes é que podem iniciar a comunicação, tendo que conhecer a identidade do mesmo;
- Os servidores não conhecem a identidade dos clientes, antes de serem contactados por estes;
- Os servidores fornecem um conjunto de serviços através de um ou mais portos;
- Os servidores podem ser clientes de outros servidores;
- A comunicações podem ser síncronas ou assíncronas;
- Os componentes podem ser organizados por camadas.



Client-Server: Exemplos

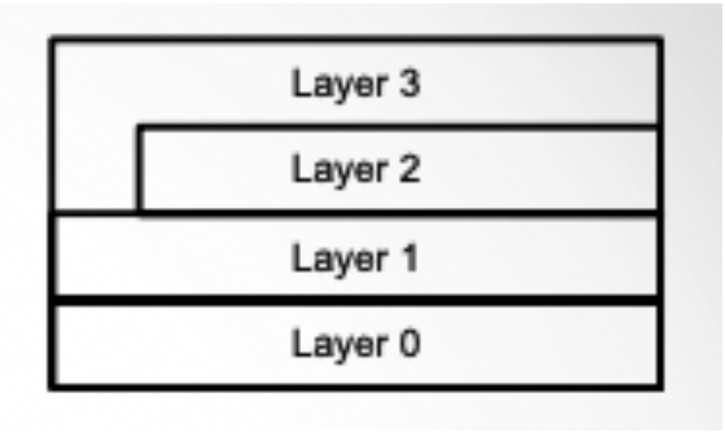
- Sistemas de informação em produção em redes locais, em que os clientes disponibilizam as interfaces gráficas ao utilizador que fazem pedidos ao servidor que contºem o SGBD - Sistema Gestor de Base de Dados;





- Uma aplicação Web em que os clientes efetuam os pedidos a partir do browser e os servidores são componentes que correm num servidor web.

Layers



- É a arquitetura mais comum;
- Os componentes são organizados em camadas horizontais;
- Cada camada tem uma responsabilidade específica, que usam outras camadas adjacentes, com pouco ou nenhum conhecimento sobre as mesmas.
- Tem como vantagem a facilidade de modificabilidade, portabilidade e reusabilidade;
- Alterações estão normalmente localizadas numa única camada;
- Possibilidade de recuso de camadas entre sistemas;
- Pode ocorrer degradação de desempenho se forem implementadas várias camadas.

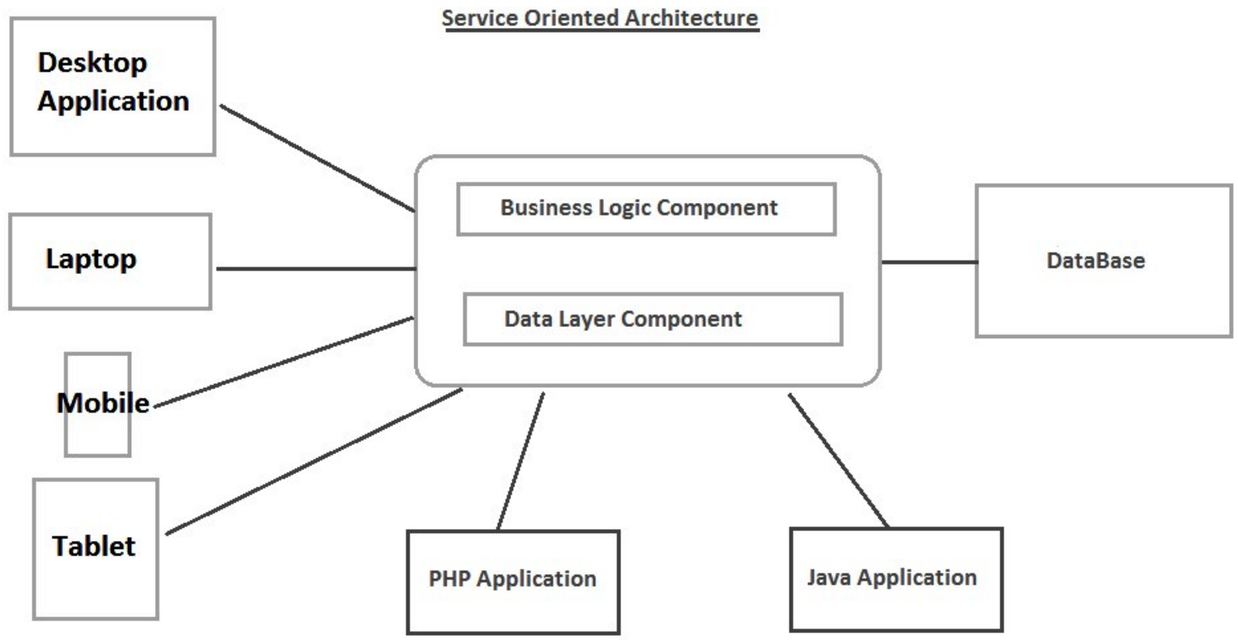
Orientado a serviços (SOA)

- Consiste numa coleção de componentes distribuídos que fornecem e/ou consomem serviços;
- Os componentes do serviço - providers - e de quem consome o serviço - consumers - podem usar diferentes linguagens de programação;
- Os serviços fornecidos não autónomos;
- O serviço providers, disponibilizam um ou

vários serviços através da publicação de interfaces;

- O serviço que consomem, invocam os serviços diretamente

ou através de um intermediário.

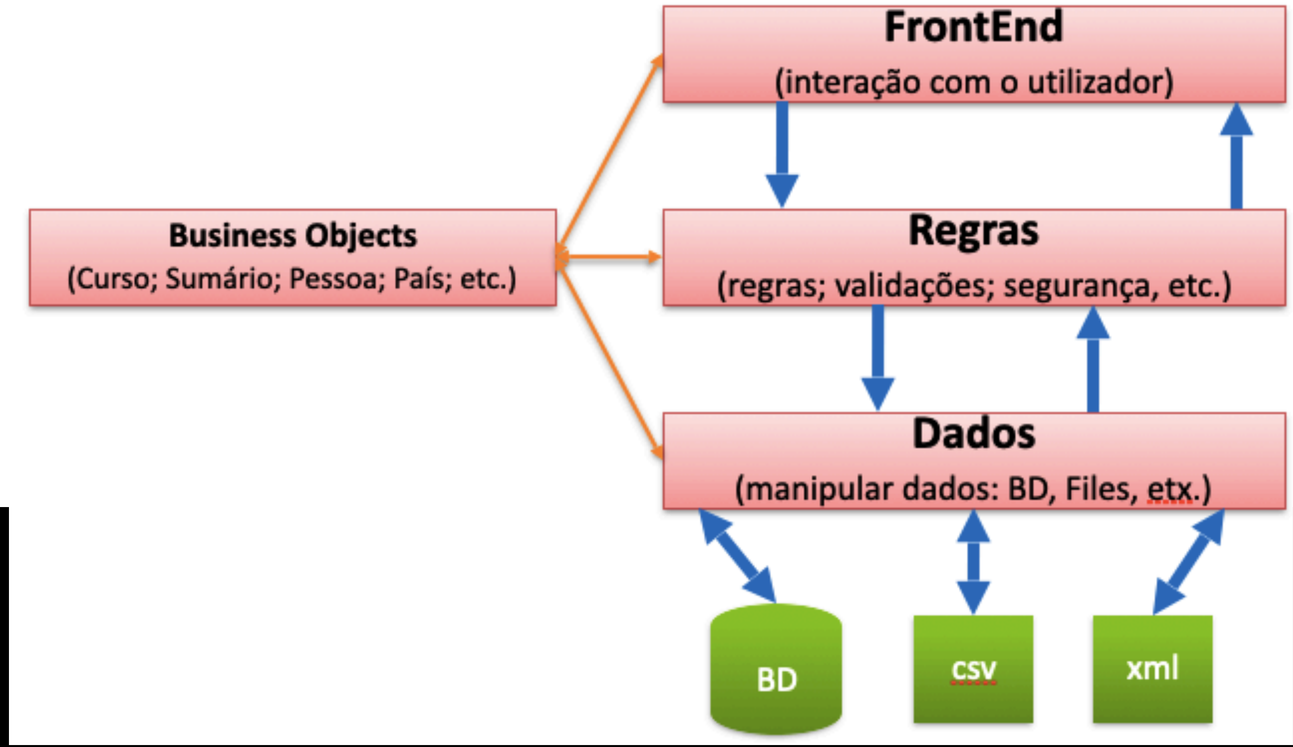


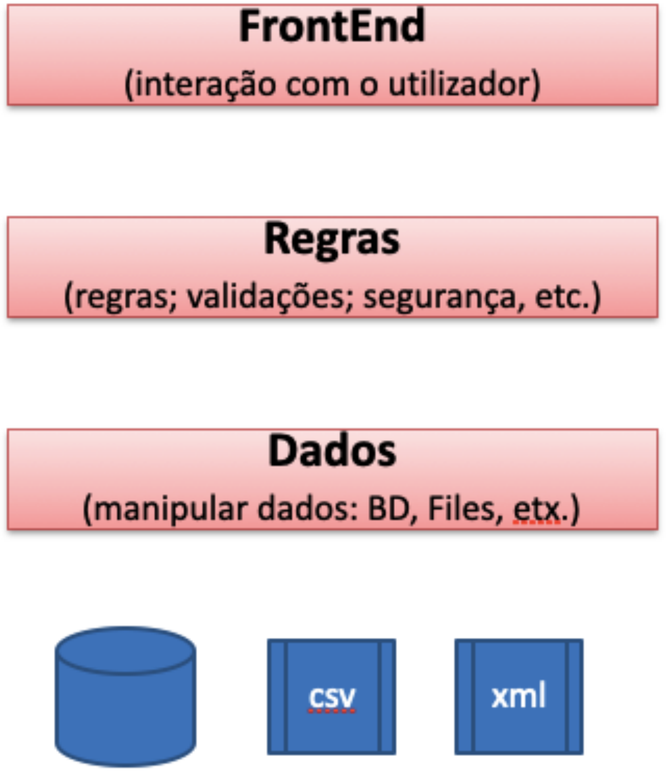
Padrões de arquitetura de software

- Um padrão é uma solução recorrentes para um problema recorrente;
- Os padrões de arquitetura resolvem problemas relacionados com os estilos de arquitetura;
- São uma forma de implementar um estilo de arquitetura de software;
- Estes definem, por exemplo, para o estilo de arquitetura em lares, quantas camadas terão que ser implementadas para um determinado sistema, ou para uma arquitetura orientada a serviços, quantos módulos existirão e como estes vão comunicar entre si;
- Os padrões de arquitetura têm impacto na base geral do código, afetando toda a aplicação transversalmente.

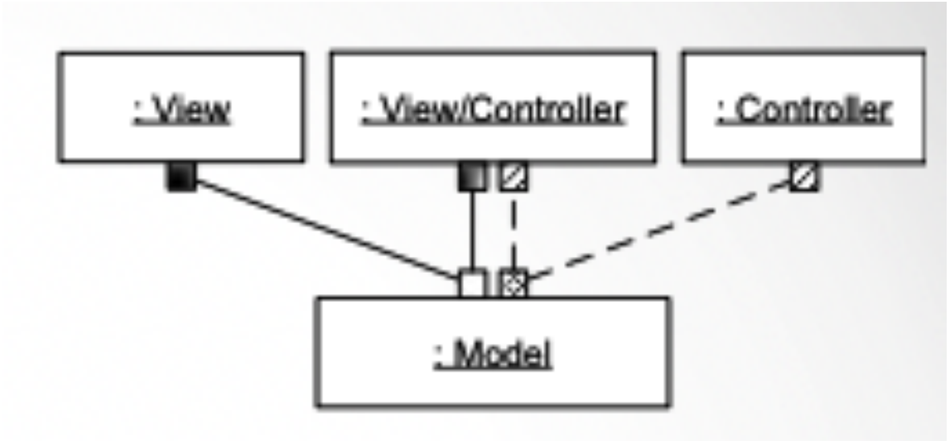
Tree-tiers

- Camada de apresentação
- Camada de regras de negócio
- Camada de acesso a dados





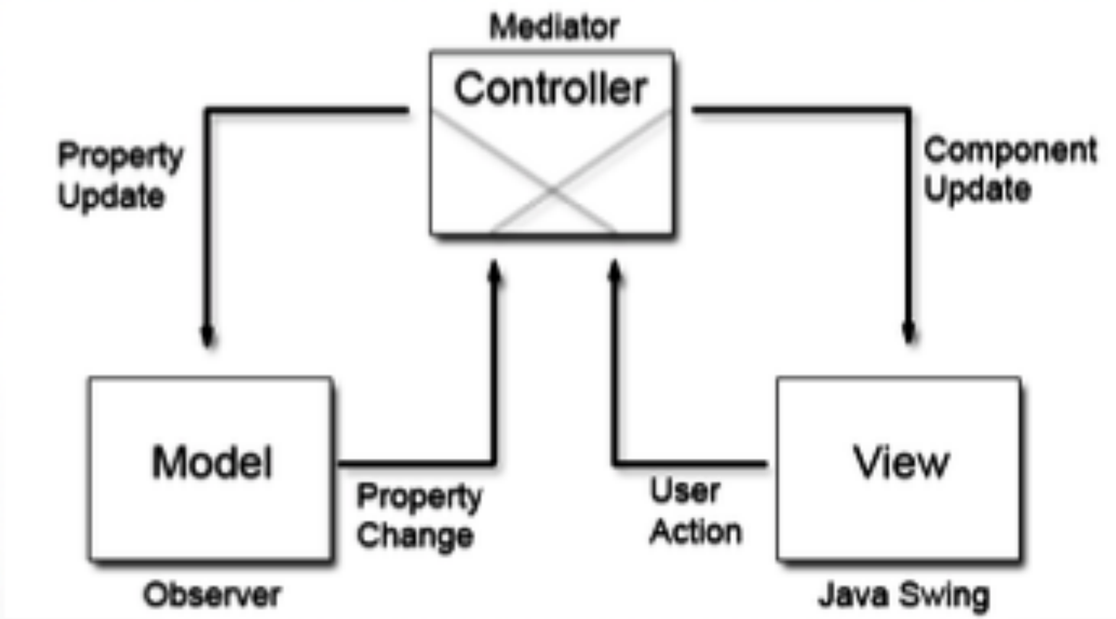
MVC - Model-View-Controller



- Os componentes interagem com um model central;
- Existe uma única central de model que representa o estado;
- O model é apresentado ao utilizador a partir de uma ou várias views;
- Os controllers interagem com o model central;
- Os componentes view e o controller são independentes um do outro, no entanto são dependentes do model central
- Tem como vantagem a facilidade de modificabilidade, extensibilidade e concorrência;
- Os componentes controller e view podem ser fáceis de alterar ou de serem adicionados;
- Processamento concorrente nas view e controllers.

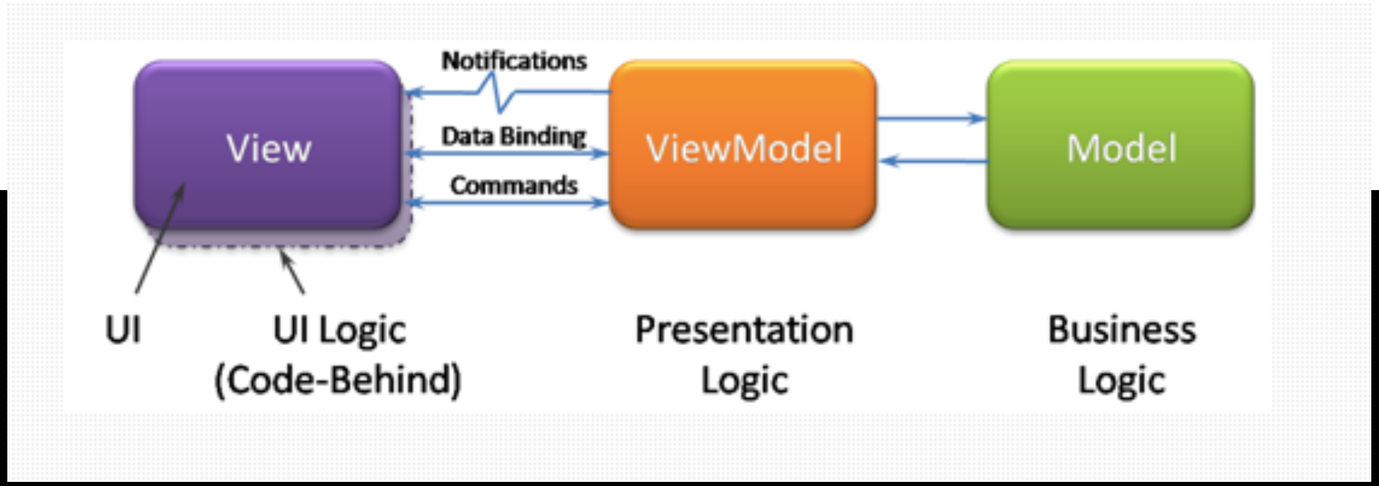
MVC - Model-View-Controller

- Example (UI with Java Swing)



MVVM - Model-View-ViewModel

- Um padrão usado para desenvolvimento em tecnologia WPF, muito semelhante ao MVC;
- A camada model não iterage com View e vice-versa;
- A view interage com o ViewModel, usando o mecanismo de binding na comunicação;
- A view através do databinding comunica com o viewmodel nootificando-o da ocorrência de eventos;
- O viewmodel responde a esta notificação comunicando com o modelo, quer para obtenção, alteração ou inserção de dados;
- Neste este padrão existe uma separação total entre a camada de apresentação, a camada de lógica e os dados;
- Existe separação total de responsabilidades;
- Facilidade de manutenção, testabilidade extensibilidade.

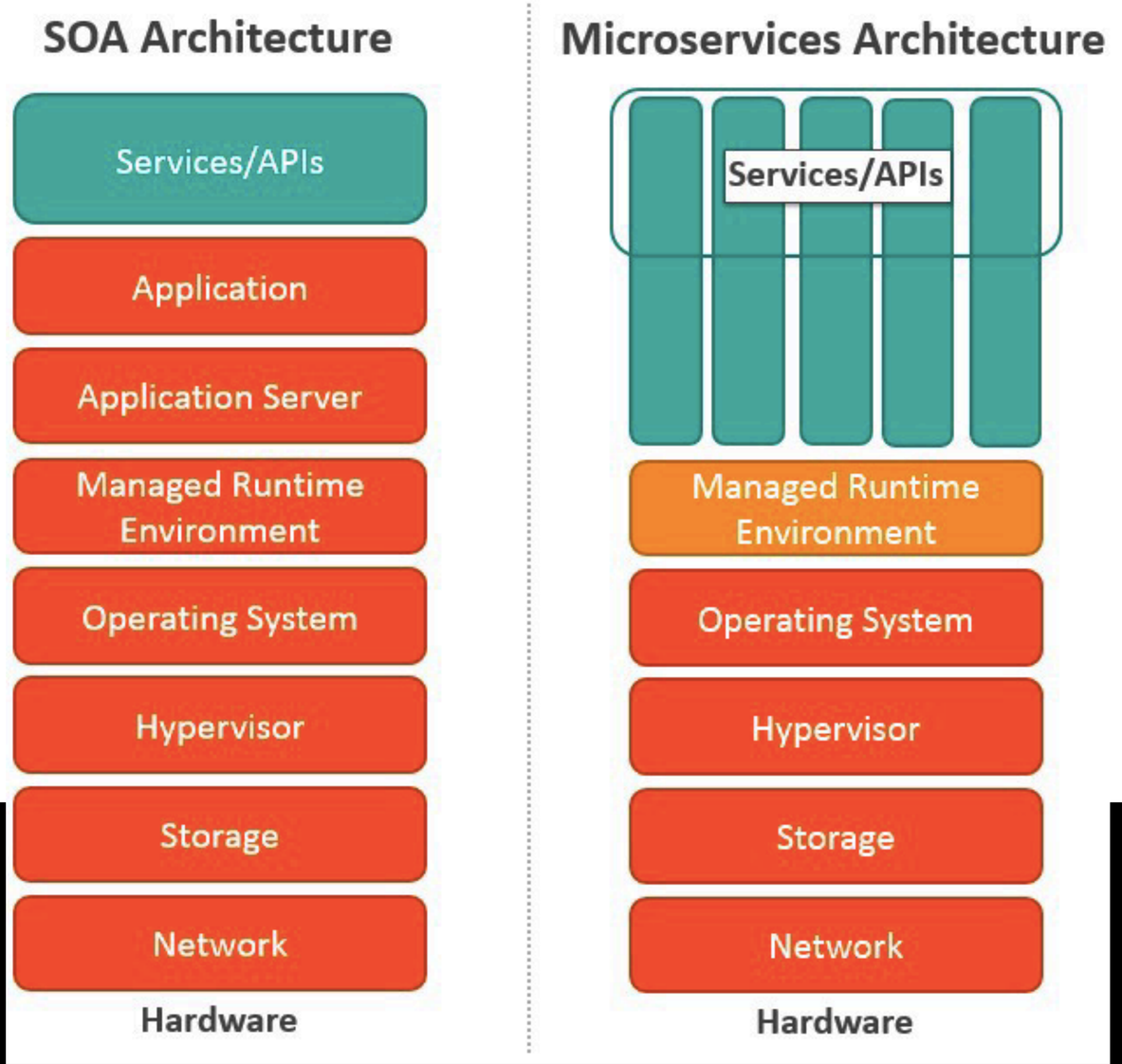


Microserviços

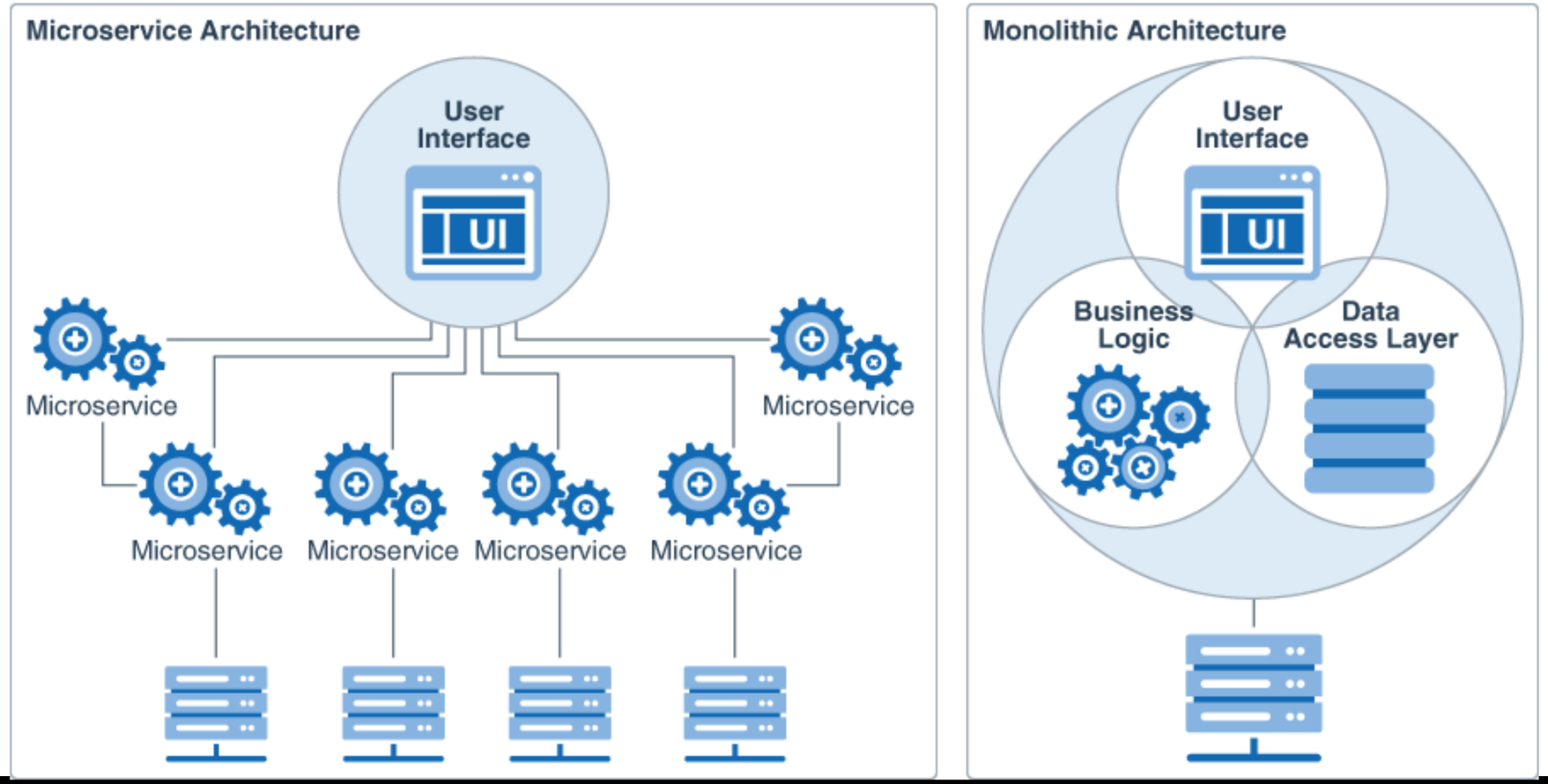
- É um padrão muito popular atualmente;
- São disponibilizado serviços que contêm um ou mais módulos da aplicação;
- Cada um destes serviços pode estar ligado a uma base de dados diferente;
- Este padrão é geralmente mais robusto e fornece maior escalabilidade;
- Suporta o desenvolvimento incremental com mais facilidade;
- Possui uma arquitetura distribuída, em que todos os componentes são independentes uns dos outros;
- Estes componentes são acedidos via remota através de REST, SOAP, RMI, etc.

Microserviços é um padrão de SOA

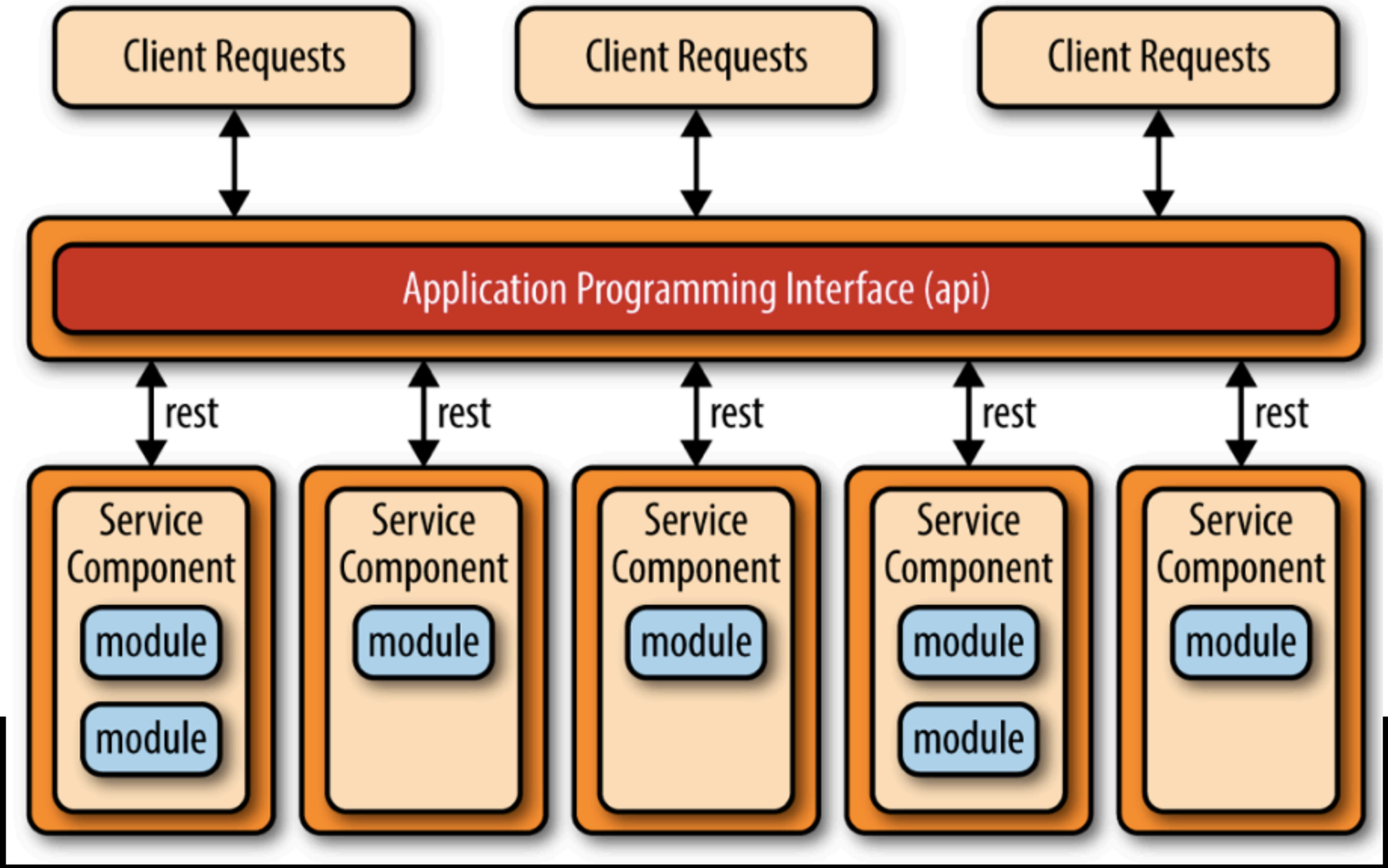
- Existem algumas divergências de opinião;
- Ambas abordam o conceito da decomposição em serviços;



Microserviços

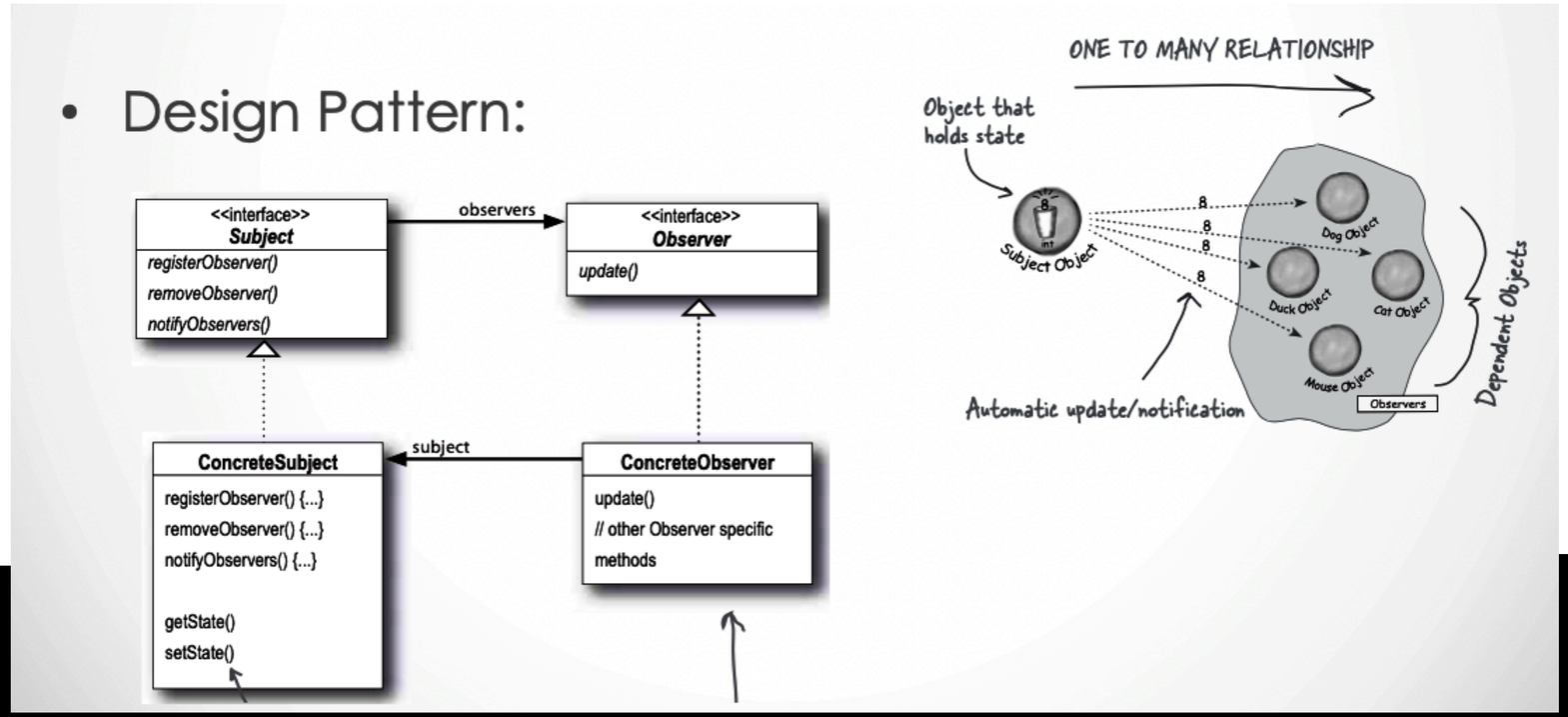


Microserviços



Design Patterns

- Os padrões de design são soluções para problemas recorrentes em sistemas de software;
- Resolve problemas mais localizados;
- É uma descrição de uma solução de design já comprovada que é a solução ideal para resolver um determinado problema;
- Diferem dos padrões arquiteturais pelo seu âmbito, são mais localizados e com impacto numa determinada secção do código base e não nele todo;
- Um exemplo de um padrão de design é o Observer, que implementa um interface onde existe dependência entre objetos de um para muitos, onde os objetos dependentes são notificados quando o objeto que “segura” o estado muda de estado.



Anti-patterns

- Expõe a verdade acerca da industria do desenvolvimento de software;
- Um anti-pattern, é em tudo semelhante a um padrão de aquitetura/design, que no entanto a sua aplicação gera consequências negativas;
- Uma anti-pattern pode ser consequência de:
    - Não saber mais ou melhor;
    - Não ter conhecimento ou experiência para resolver um determinado tipo de problema;
    - Aplicar um determinado padrão no contexto errado.
- Causam bloqueios no desenvolvimento;
- Podem gerar a oportunidade para criação de soluções ainda não provadas.
- Existem vários tipos de anti-pattern e estas podem descrever formas úteis de fazer refactoring ao código.

Engenharia de Software

Arquiteturas de software

Marta Martinho 2020