Vertical Slice Architecture

Harã Heique dos Santos

Por que usamos padrões/estilos arquiteturais?

- **“Resolver” problemas já conhecidos** na arquitetura de software;
- **Soluções comprovadas** e amplamente aceitas;
- Princípios e conceitos que **definem a estrutura básica do software**;
- Como os componentes do sistema **devem interagir e se comunicar entre si**;

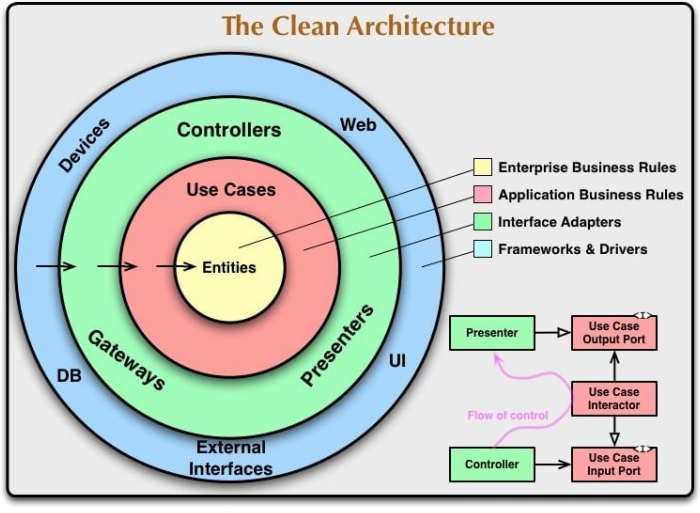
- Exemplos: MVC, Hexagonal Architecture, Onion Architecture, Clean Architecture, SOA, Client-Server, Microservices e afins.

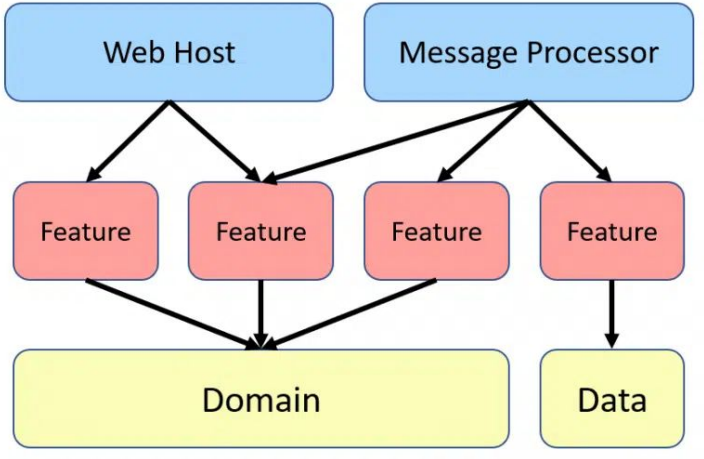
O que é Vertical slice architecture (VSA)?

- É um padrão/**estilo** arquitetural;
- Assim como arquitetura em camadas (e variações) foca no quesito de acoplamento, mas com **alta ênfase em coesão**;
  - **Acoplamento** = grau de dependência entre elementos em um software;
  - **Coesão** = grau em que os elementos dentro de um software pertencem juntos.
- Se preocupa demasiadamente nas **capacidades do negócio;**
  - O que a aplicação faz;
  - O que meu sistema provê para o negócio;
- Ideias para sistemas que **atacam o negócio/domínio.**
  - Sistemas que resolvem **problemas e necessidades de negócio** (geralmente empresariais);
  - Sistemas que atuam como **produto final para nosso cliente final**.
- A nível de código a ideia é organizar o código **por features.**
  - Features = **use cases do negócio**. Uma ação. Algo que posso fazer.

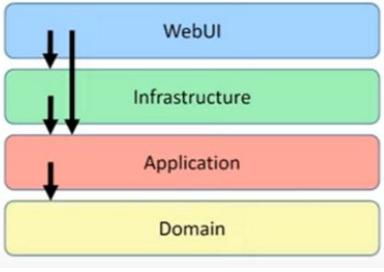
Comparação entre CA e VSA

Separation of concerns

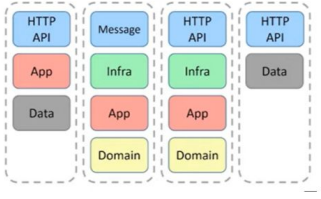


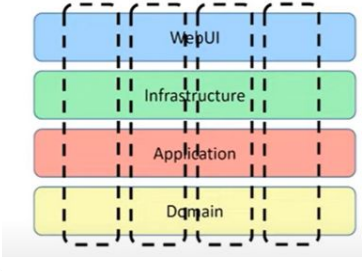


Coupling
Dependencies



Boundaries
Cohesion





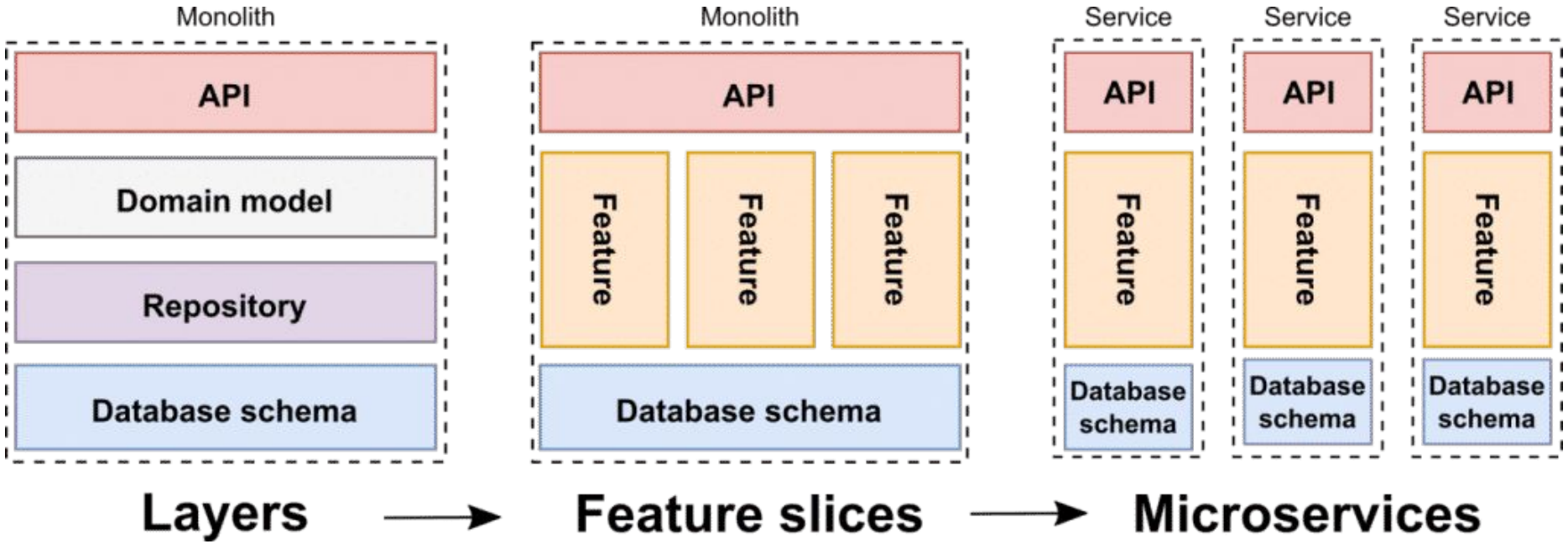
Comparação entre CA e VSA

N Layers (Clean, Hexagonal, Onion afins)	Vertical Slice Architecture
Foco demasiado em acoplamento	Acoplamento limitado no escopo da feature com foco demasiado em coesão
Separation of concerns (acoplamento e dependencies directions)	Coesão funcional (grupo relacionado de operações baseado em **tarefas**)
Separação em cada camada para permitir que elas mudem independentemente. Porém as alterações de único use case (feature) são feitas em todas as camadas juntas.	Tudo que muda junto (features) fica/pertence junto.
Médio/Alto uso de abstrações para manter as camadas internas “limpas” e isoladas do “mundo externo”.	Baixo ou nenhum uso de abstrações. Caso use é mais p/ modularização e fácil uso de uma interface limpa e amigável.
Bom em cenários que há maior complexidade em produtos mais técnicos e que são usados por outros produtos	Bom em cenários que focam no domínio (core), onde as funcionalidades específicas do negócio são fundamentais (produto para cliente final).

Comparação entre CA e VSA

N Layers (Clean, Hexagonal, Onion afins)	Vertical Slice Architecture
Foca mais em *technical concerns* (preocupação técnicas)	Foca mais em *business concerns* e *capabilities* (preocupações nas capacidades do negócio)
Organização estrutural do código **por camadas**	Organização estrutural do código **por features**

Similaridades com serviços micro-componentizados



Fonte: https://www.betterask.erni/news-room/slices-vs-layers/

Trade-offs do VSA (vantagens)

- Minimiza o acoplamento entre features e maximiza o acoplamento na feature;
- Melhoria na manutenibilidade (infrequência de conflitos);
  - Se preciso mexer num use case eu só mexo nele sem impactar outros
- Implementação de cada feature baseado na sua complexidade;
  - Se uma feature é mais complexa então adote complexidade nela. Caso seja mais simples, seja mais pragmático e direto (decisão por feature set).
- Bom controle de versionamento das features (funcionalidades);
- CQS/CQRS out-of-box;
  - Como divido por features acabo segregando em use cases de leitura e escrita me fornecendo as Queries & Commands.
- Ênfase na coesão com divisão e agrupamento de features;
- Redução de abstrações;
  - Consequentemente reduz a complexidade, compreensão e melhor manutenção.
- Pragmatismo nas implementações;
- Preocupação em atacar as necessidades e problemas do negócio
  - Código é projetado e estruturado pelas necessidades e funcionalidades que o negócio deseja;
  - Usa a linguagem do negócio para definir as features.

Trade-offs do VSA (desvantagens)

- Maturidade da equipe;
  - Um estilo/padrão arquitetural novo pode causar resistência (o que é natural);
  - Mudança de olhar para o código foge do modelo padrão;
  - A flexibilidade entregue pode ser ruim ou bom, pois teríamos que adotar um “padrão”;
  - Pensamento crítico em olhar o cenário dado e perseguir uma implementação compatível à situação.
- Possível duplicação de código entre features;
- Compartilhamento de recursos entre features;
- Testes automatizados (Neon);
  - Quando aplicado o ideal é trabalhar com testes de integração além dos unitários;
  - Testes de integração/funcionais p/ orquestradores e bordas da aplicação (componentes menos críticos e estáveis);
  - Testes unitários p/ onde há lógica de fato e não precisa de mocks/stubs (componentes mais críticos e estáveis).
- Quantidade de arquivos gerados no código fonte
  - Sinceramente isto não deveria ser a “preocupação principal”.

Referências

- Baixar todas as referências neste link do pdf: https://drive.google.com/file/d/19-_KlgpbgFcTl1nY6WSq5ao3d_jEcrxr/view?usp=sharing

SHOW ME SOME CODE!!!

Código fonte aqui

Conclusão

- O VSA tem foco no valor ao negócio;
- Em todos aspectos de software sempre perseguimos: “**a alta coesão e o baixo acoplamento**”.