Next-Gen PHP

Aula 06

Separe camadas para organização

O que é Arquitetura?

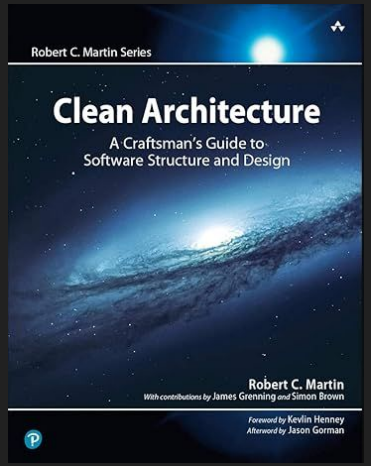
Arquitetura de Software?

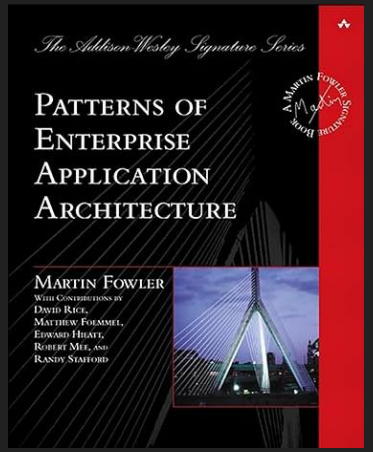
Arquitetura de Soluções?

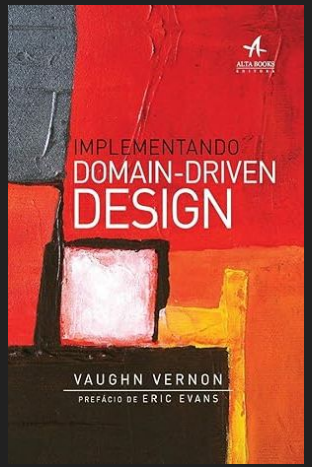
As dores

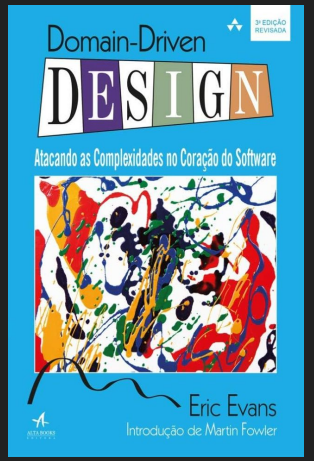
1. Projeto desorganizado.
2. Mistura de conceitos.
3. Difícil de testar partes.
4. Dificuldade na separação de serviços.
5. Módulos complexos de isolar.
6. Regras de negócio difíceis de mudar.

Alguns livros









Primeiros Padrões de Arquitetura

Domain Logic Patterns

- Service Layer
- Domain Model

Data Source Architectural Patterns

- Active Record
- Data Mapper

Object-Relational Behavioral Patterns

- Unity of Work
- Lazy Load

Object-Relational Metadata Mapping Patterns

- Metadata Mapping
- Query Object
- Repository

Primeiros Padrões de Arquitetura

Web Presentation Patterns

- Template View
- Application Controller

Distribution Pattern

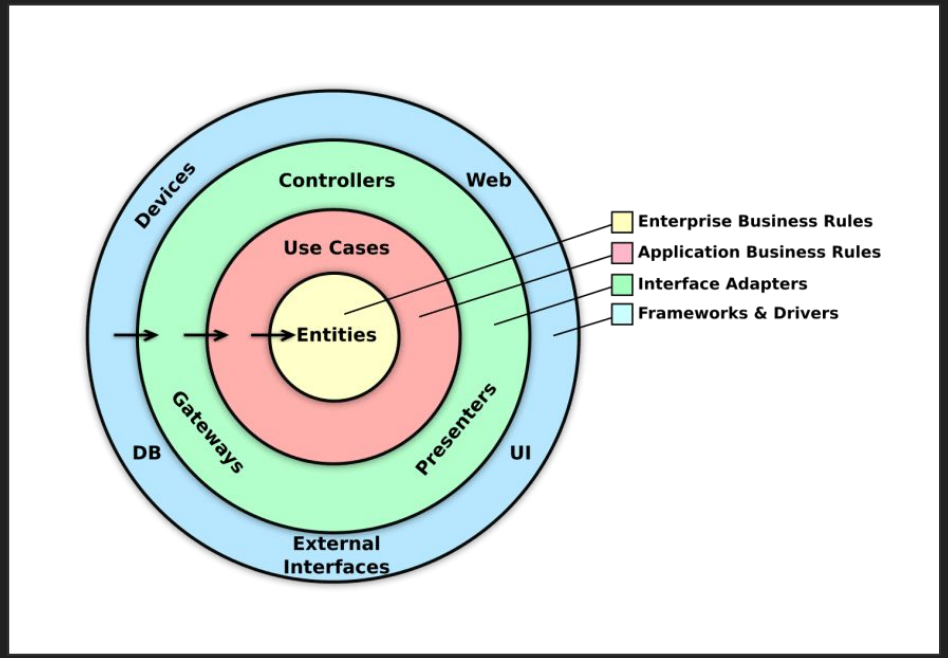
- Data Transfer Object
- Data Mapper

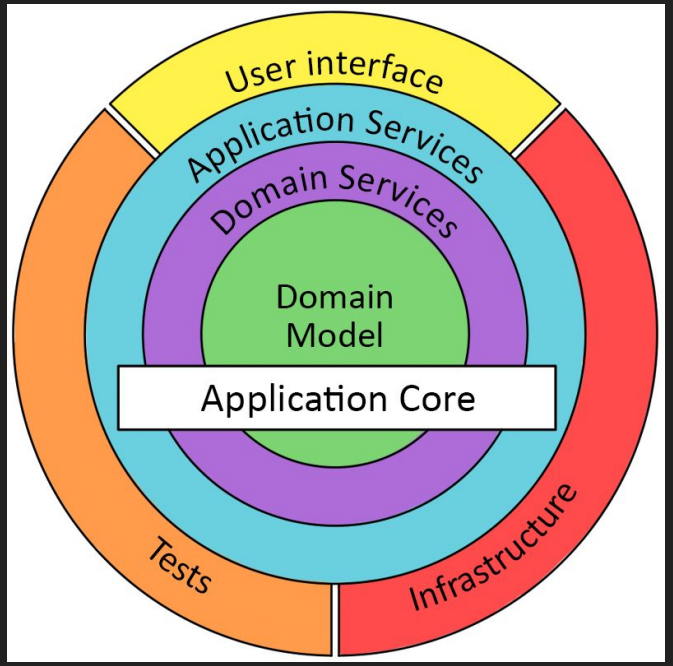
Base Patterns

- Gateway
- Mapper
- Registry
- Value Object

Clean Architecture

A Clean Architecture se propõe em separar sua aplicação em camadas, o padrão mais comum é o MVC, mas com o tempo ele foi não sendo de tanta ajuda como esperávamos, então, juntado de ideias sobre hexagonal e arquitetura por camadas em um artigo o Uncle Bob trouxe as primeira ideias em relação esse modelo de arquitetura.





Entities - Camada de regras de negócio

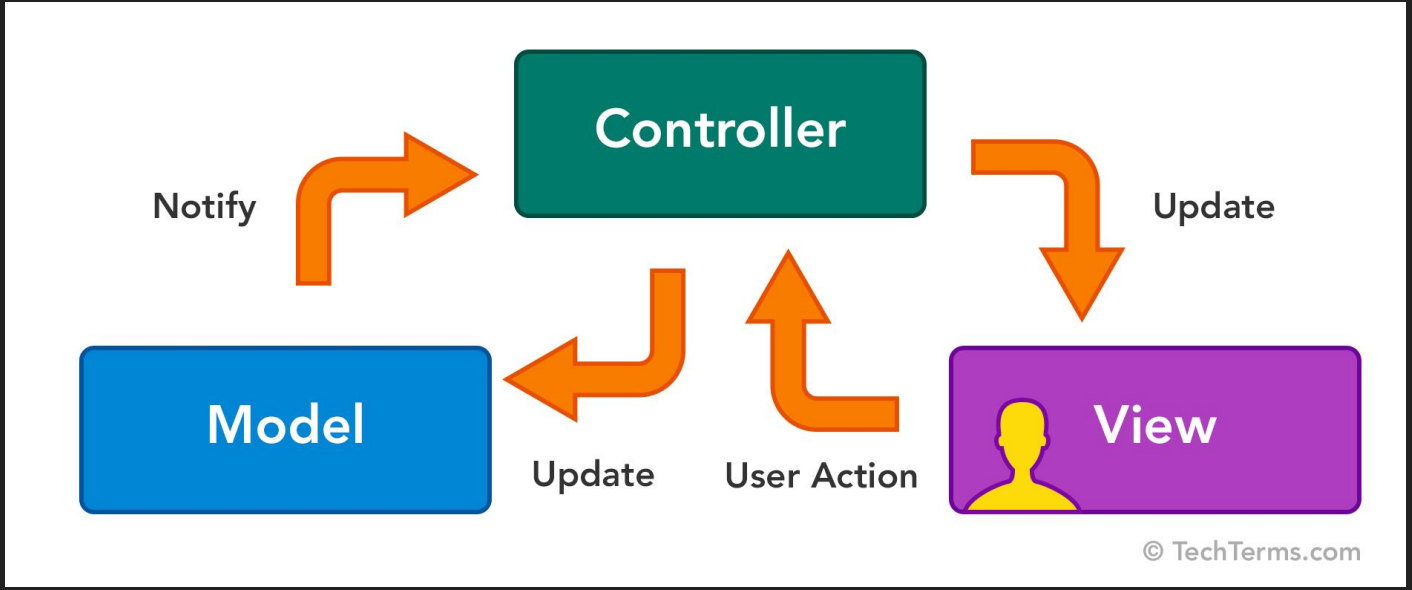
Camada onde ficam as regras de negócio de seu sistema, normalmente essas classes não devem fazer acesso a banco de dados, comunicação de API, ou qualquer mecanismo externo, é onde os dados devem ser passados e verificados em suas regras.

UseCases - Camada de regras de aplicação

Esta camada é a que liga a comunicação entre acesso a recursos externos com os Domínios, essa comunicação é focada nas regras de aplicação e ligar entre múltiplas ações de validação de regras invocando-os quando necessário.

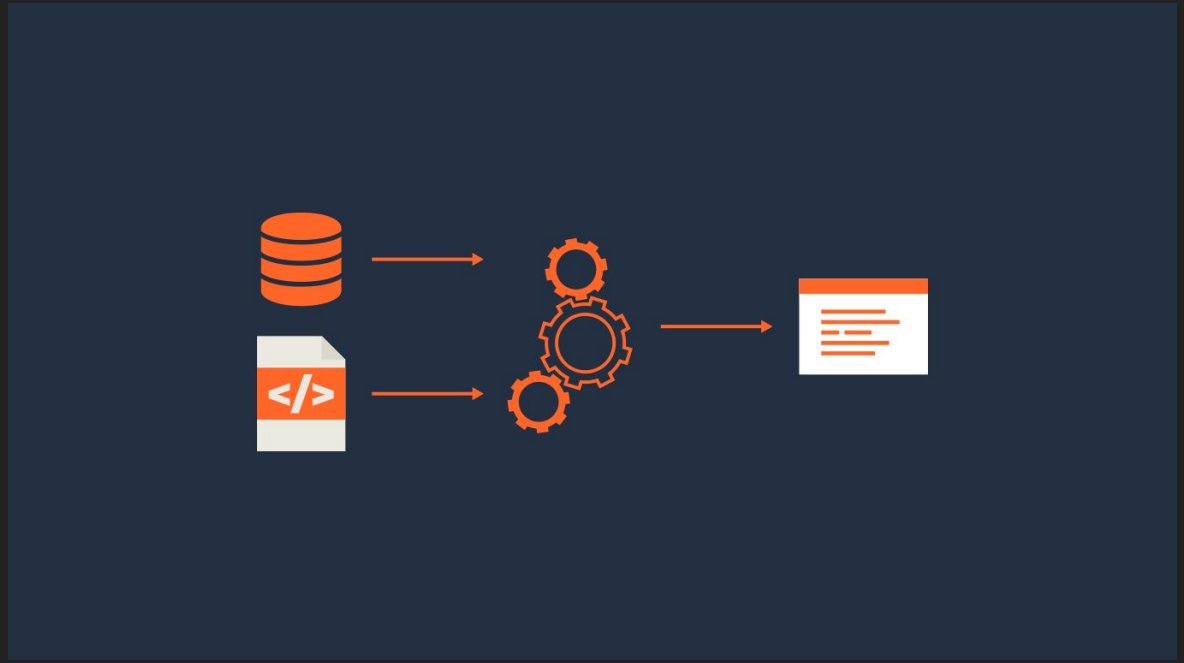
Controllers - Camada de interfaces e adapters

É a camada que geralmente recebe a entrada de dados da aplicação, também ela normalizar entrada e saída chamando agentes para isso, os Controllers são camadas que vão ter acessos a elementos do Framework também.



Frameworks and Drivers - UI, Web e Database

É a camada de configuração, também considerada a camada onde fica a Infraestrutura, destinada a ficar as configurações dos elementos externos como: libs de terceiros, container de injeção de dependência, banco de dados e também a saída de dados, muito comum em aplicações fullstack onde a saída cai para uma camada de Presentation.



A Camada mais alta o SetUp

Dependency Injection Container

Conhecido também como IoC do conceito Inversion of Control

```php
class RiakServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     */
    public function register(): void
    {
        $this->app->singleton(Connection::class, function (Application $app) {
            return new Connection(config('riak'));
        });
    }
}
```

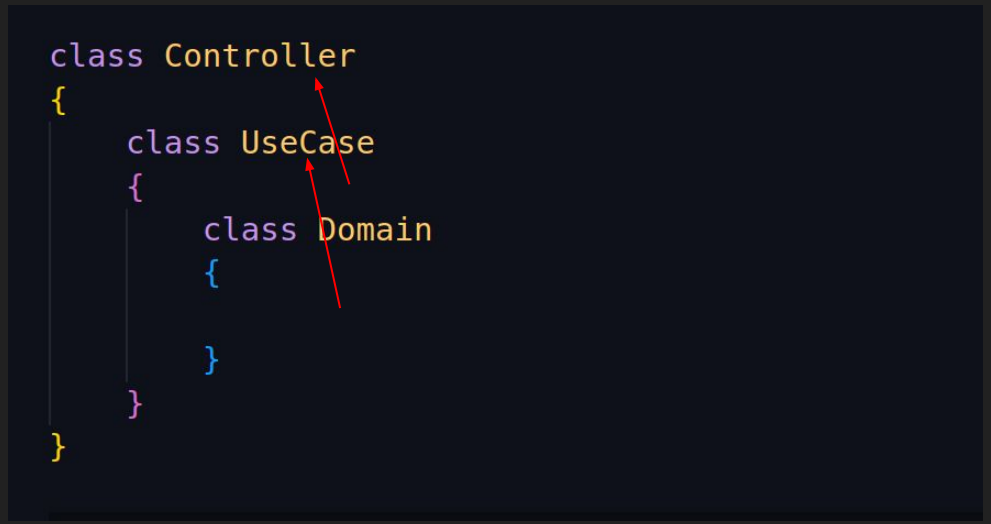
```php
$container = new Container([
    // Dependency parent with dependency child

    // all dependencies should be involved by a Closure(function() or fn())
    Dependency::class              => fn() => new Dependency(),

    ComponentThatHasAnotherDependency::class => function($container) {
        return new ComponentThatHasAnotherDependency(
            $container->get(Dependency::class)
        );
    },

    // or simply
    ComponentThatHasAnotherDependency::class =>
        fn($c) => new ComponentThatHasAnotherDependency($c->get(Dependency::class)),

    // more complex injections
    ComponentThatHasTwoDeps::class => fn($c) => new ComponentThatHasTwoDeps(
        $c->get(Dependency::class),
        $c->get(AnotherDependency::class),
    )
]);
```

Injeção acontecendo

```php
class GetUsersUseCase implements CommandInterface
{
    /**
     * @param SearchDTO $searchDTO
     * @param QueryBuilder $queryBuilder
     * @param UserRepository $userRepository
     */
    public function __construct(
        protected SearchDTO $searchDTO,
        protected QueryBuilder $queryBuilder,
        protected UserRepository $userRepository
    ) {
    }
```

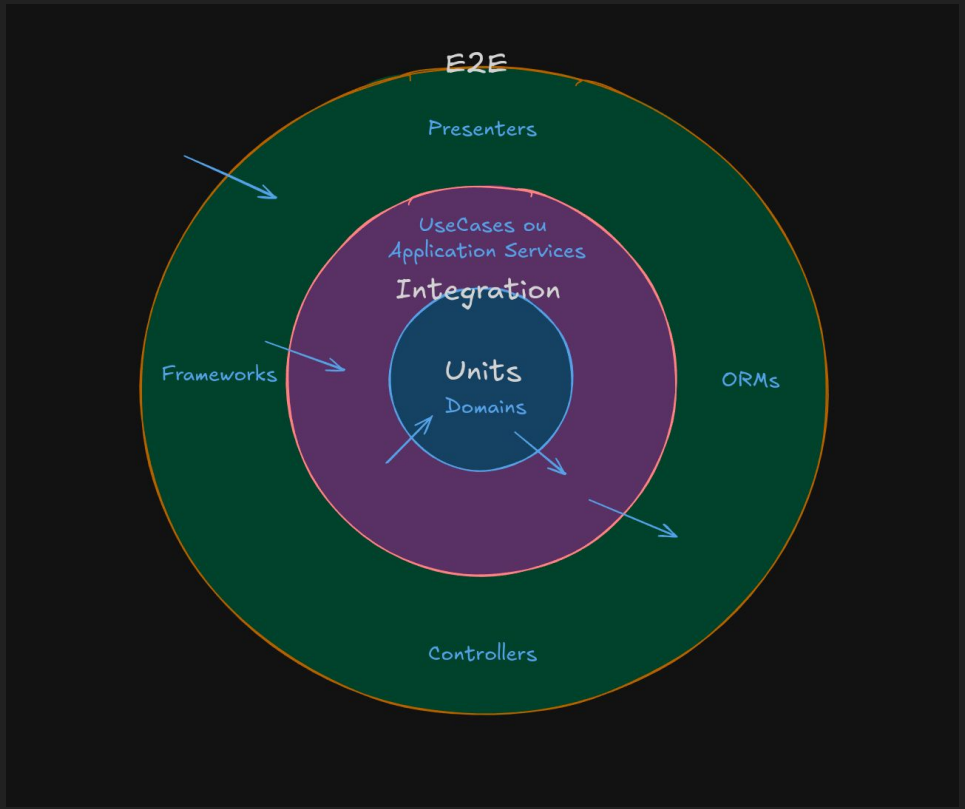
Camadas anti-corrupção

Em modelos Rich Domain Model




```
class Domain
{
    // Não chamar Repository
    // Não chamar Logs
    // Não chamar UseCases
    // Não chamar APIs
}
```

Arquitetura sobre Testes



Desafio

Presente no repositório do nosso curso:

https://github.com/DifferDev/NextGenPHP

Hands-on

Vamos ao mão na massa!!!

Clean Architecture no LARAVEL!

Boa noite!

Obrigado pela presença!