

Arquitetura Hexagonal

Módulo 1 - Teoria e Fundamentos

Introdução

- O que é?
- Para que serve?
- Quando surgiu?
- Qual o objetivo?



Introdução

O que é?

1. É um padrão de projeto:

Na engenharia de software, um **padrão de projeto** (Design Pattern) é uma **solução geral reutilizável** para um **problema** que ocorre com **frequência**, dentro de um determinado contexto no projeto de software.

Introdução

2. É um padrão arquitetural:

Na engenharia de software, um padrão arquitetural consiste nas **regras** e **princípios** que definem como será a **organização estrutural** abstrata dos componentes significativos de um software, seus relacionamentos e suas interfaces externas.

Introdução

3. É um “Guideline”:

Uma diretriz é adotar **valores** para determinar o curso de uma ação. Visa simplificar processos específicos de acordo rotinas definidas ou práticas. Por definição, seguir uma diretriz na íntegra nunca é obrigatório. Você adota seguir a diretriz com o objetivo de **colher resultados** previamente direcionado pela determinada ação.

Introdução

Para que serve?

Projetar e construir aplicativos de software, estabelecendo uma arquitetura moderna, robusta e altamente flexível, orientadas pelas premissas básicas da filosofia de desenvolvimento ágil:

1. desenvolvimento orientado a TDD.
2. foco nos requisitos de negócio.
3. adiar decisões técnicas o máximo possível.

Introdução

Quando surgiu?

Elaborado e documentado em 2005 por Alistair Cockburn (https://en.wikipedia.org/wiki/Alistair_Cockburn):

- um dos autores do manifesto agil 2001.
- autor da metodologia Cristal Clear.

Foi idealizado para que as equipes de desenvolvimento pudessem aplicar premissas de ágil na elaboração da arquitetura de software. Perdeu força nos anos posteriores mas ressurgiu como um interesse global a partir de 2015.

Introdução

Qual objetivo?

Criar uma soluções de software que tenha uma arquitetura que permita igualmente ser executada por usuários, programas, testes automatizados, e que seja desenvolvido e testado isoladamente de seus dispositivos externos de execução eventuais, fazendo que com a equipe de desenvolvimento foque no desenvolvimento do requisitos de negócio, ignorando dependências externas técnicas e infra estruturais.

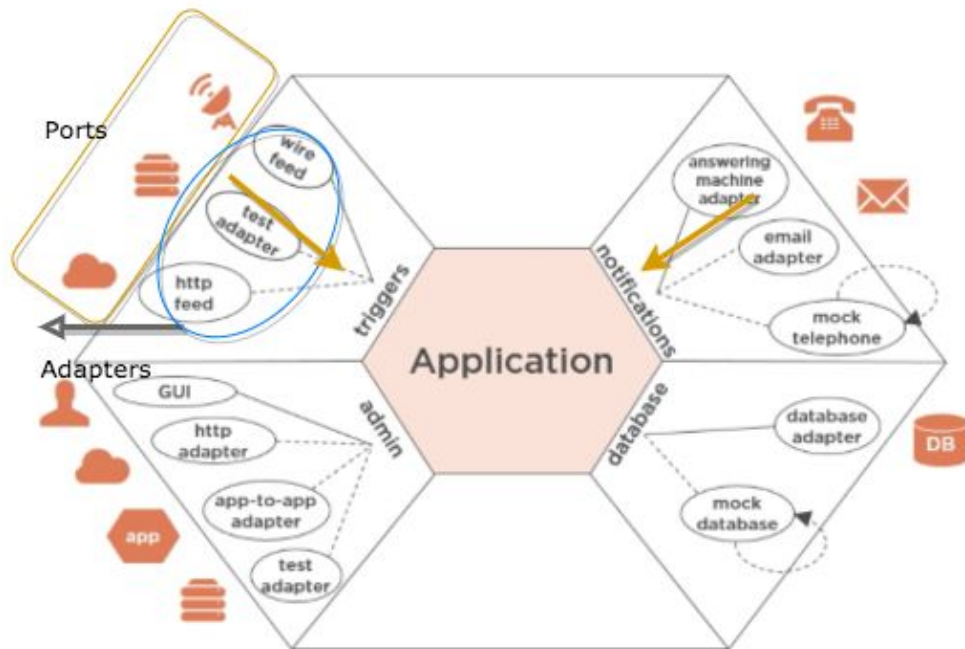
Introdução

Nomes Conhecidos:

Também chamados de "Ports and Adapters Patterns".

Introdução

Hexagonal Architecture



Introdução

Dúvidas e comentários?



Arquitetura Hexagonal

Princípios e Metáforas: Isolamento

Isolamento

Separation of concerns - SoC

É um princípio básico da engenharia de software, que visa separar preocupações, ou seja, **modularizar** uma solução de forma que cada módulo esteja focado em resolver apenas um **único problema**.

Separar preocupações é imprescindível para quem deseja elaborar uma arquitetura flexível, manutenível e sustentável.

Veja https://en.wikipedia.org/wiki/Separation_of_concerns

Isolamento

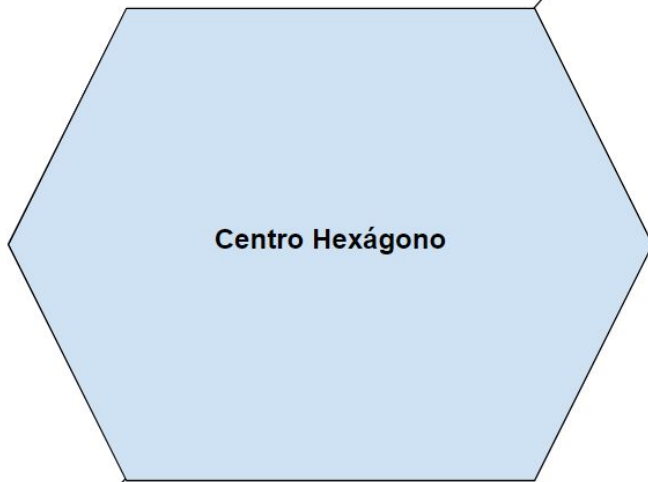
A arquitetura hexagonal aplica **SoC** e estabelece o princípio de modularizar a solução em **3 áreas distintas e isoladas**:

1. Centro como hexágono.
2. Lado superior esquerdo, fora do hexágono.
3. Lado inferior direito, fora do hexágono.

Lado superior esquerdo

Centro Hexágono

Lado inferior direito



Isolamento

Centro Hexágono

O desenho escolhido por Alistair Cockburn para desenhar o aplicativo foi um hexágono, por isso o nome. Dentro do hexágono, temos as coisas que **são importantes para o problema de negócio** que a solução está tentando resolver.

Isolamento

Centro Hexágono

É a parte mais importante do sistema que contém todo o código que relaciona com lógica de negócios do contexto da solução. Contém código de **oop/aop/fp** que abstrai as regras de negócio do mundo real em modelo de objetos. Esta é a parte que queremos isolar de todas as outras partes (lados esquerdo e direito).

Isolamento

Centro Hexágono

O hexágono deve ser **totalmente agnóstico** a qualquer tecnologia, framework e infraestrutura relacionado a **interfaces gráficas**, **interfaces comunicações** e **dispositivos externos** do mundo real.

Entretanto, o hexágono pode ter dependências de frameworks de serviços gerais, como por exemplos: Logg, IoC e etc.

Isolamento

Lado Superior Esquerdo

Este é o lado intercambiável e flexível através do qual um ator externo irá interagir e estimular a solução. Conterá **código de tecnologia específica** que irá disparar eventos na solução, normalmente um pessoa usando uma GUI ou programa externo via end point web services.

Isolamento

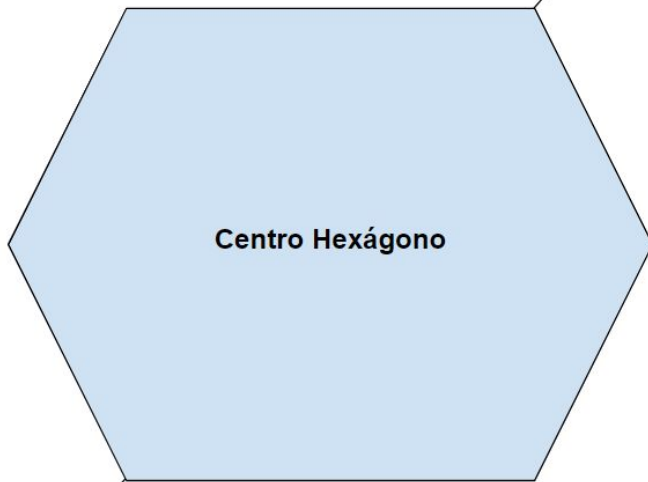
Lado Inferior Direito

Este é o lado intercambiável e flexível que fornecerá os **serviços de infraestrutura** que a solução precisa para existir. Conterá **código de tecnologia específica**, detalhes infraestruturais, normalmente código que interage com o banco de dados, faz chamadas para o sistema de arquivos, chamadas HTTP e outros aplicativos dos quais depende.

Lado superior esquerdo

Centro Hexágono

Lado inferior direito



Isolamento

Dúvidas e comentários?



Arquitetura Hexagonal

Princípios e Metáforas: Atores

Atores

Fora do hexágono, temos qualquer coisa do mundo real com a qual o aplicativo interage. Essas coisas incluem seres humanos, outros aplicativos ou qualquer dispositivo de hardware ou software. Eles são chamados de atores.

Eles são divididos em **2 grupos** dependendo de quem desencadeia a interação entre a solução e o ator:

1. Ator Primário Condutor (Driver)
2. Ator Secundário Conduzido (Driven)

Atores

1. Ator Primário Condutor

Atores no lado esquerdo / superior são chamados de atores primários condutores (Drivers). A interação **é acionada** pelo ator. É aquele que interage com o aplicativo para atingir um objetivo.

Exemplos: Suites de testes, front-end desktop, front-end web, end-point rest e etc...

Atores Primários Condutores



Test Cases



Human User



Remote Application



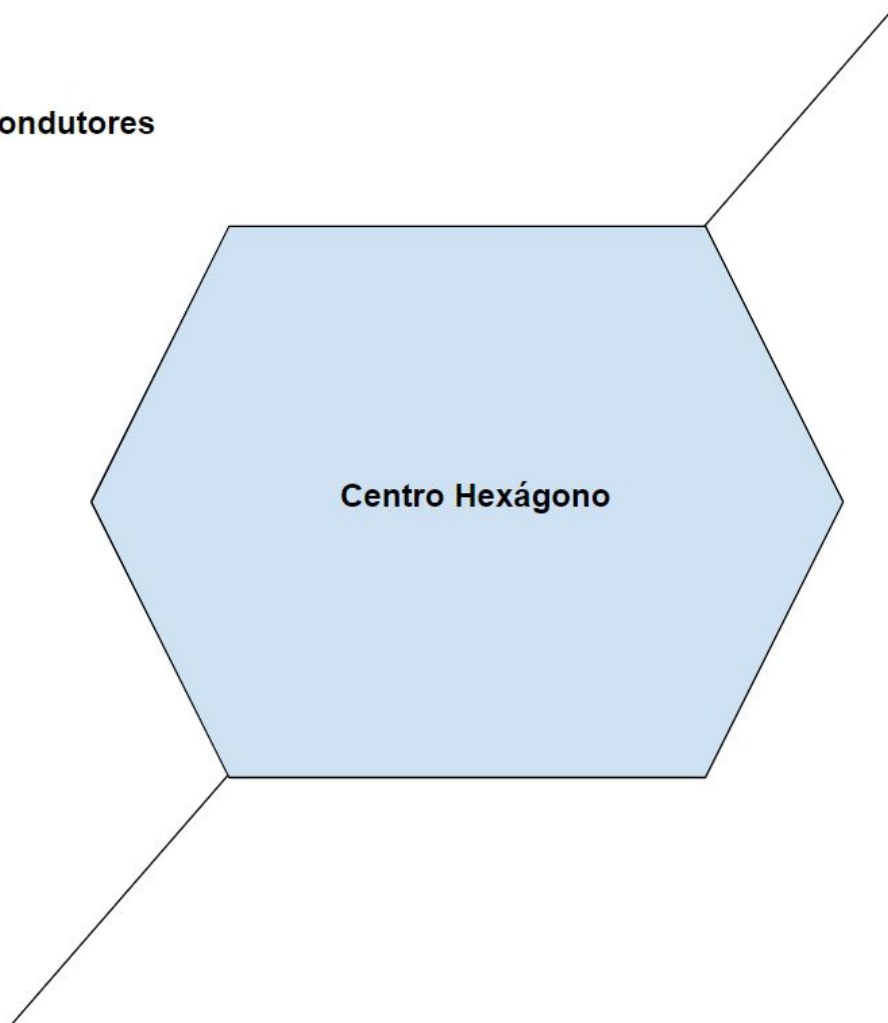
Single Page App



Mobile App



Message Queue



Atores

2. Ator Secundário Conduzido

Atores no lado direito / inferior chamados atores secundários conduzidos (Driven). A interação é acionada pelo hexágono. Um ator secundários **fornece funcionalidades necessárias** ao hexágono para processar a lógica de negócios.

Exemplos: banco de dados relacionais, nosql, serviços web http, stmp, sistema de arquivos e etc.

Atores

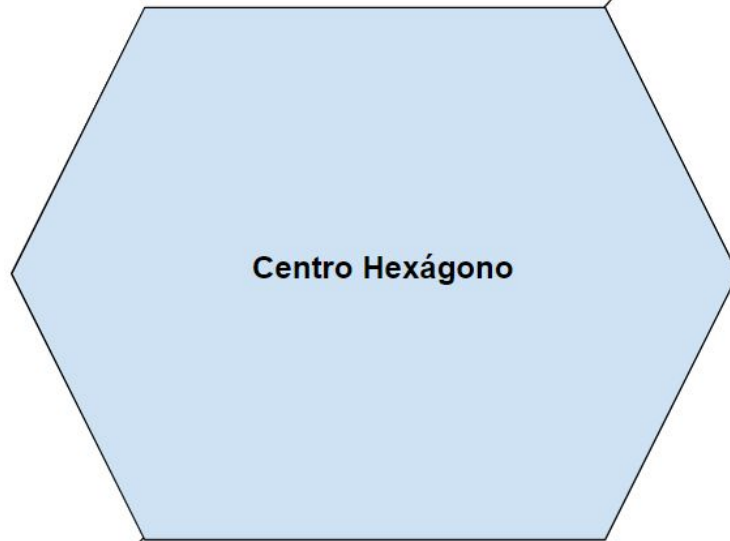
2. Ator Secundário Conduzido

Existem 2 tipos de atores conduzidos:

1.Repositório (Repository): É aquele com o objetivo de **enviar e receber** informações. Por exemplo, um banco de dados ou qualquer outro dispositivo de armazenamento.

2.Destinatário (Recipient): É aquele com o objetivo de **somente enviar** informações e esquecer. Por exemplo, um envio de email SMTP ou um post HTTP.

Atores Primários Condutores



Atores Secundários Conduzidos

Atores

2. Ator Secundário Conduzido

Como descobrir novos atores?

Para descobrir **qual é o tipo de ator** em uma interação, pergunte a si mesmo “quem” **aciona a conversa**. Se a resposta for "o ator", então é um condutor primário. Se a resposta for "o aplicativo", o ator é um ator conduzido secundário.

Atores

Dúvidas e comentários?



Arquitetura Hexagonal

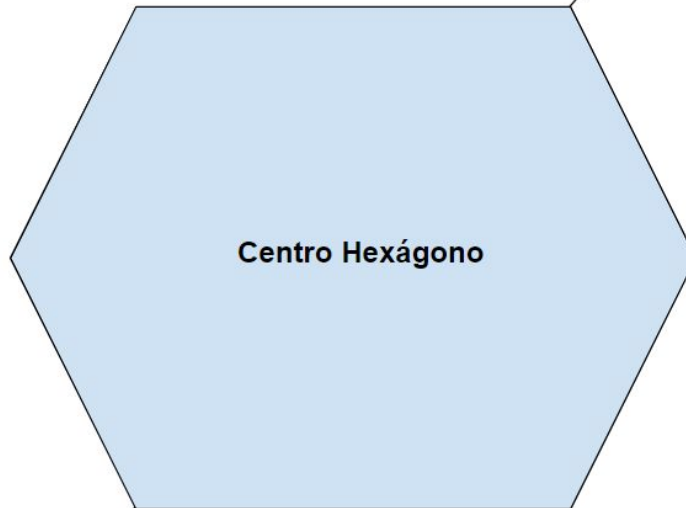
Princípios e Metáforas: Dependências

Dependências

A arquitetura hexagonal estabelece o seguinte princípio de dependências: "**somente de fora para dentro!**":

1. Lado esquerdo, os atores primários dependem do hexágono.
2. Lado direito, os atores secundários dependem do hexágono.
3. O centro, o hexágono não depende de ninguém, só dele mesmo.

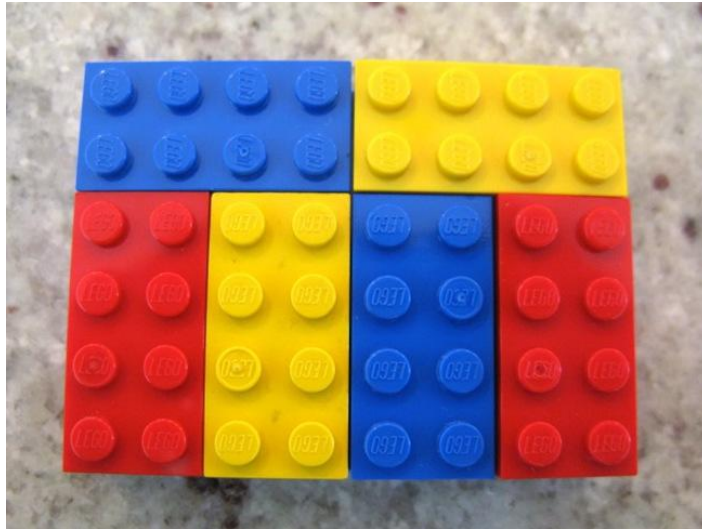
Atores Primários Condutores



Atores Secundários Conduzidos

Dependências

Sendo assim, o lado esquerdo e o direito se torna totalmente **flexível**, **intercambiável** e dependente do centro.



Dependências

Dúvidas e comentários?



Arquitetura Hexagonal

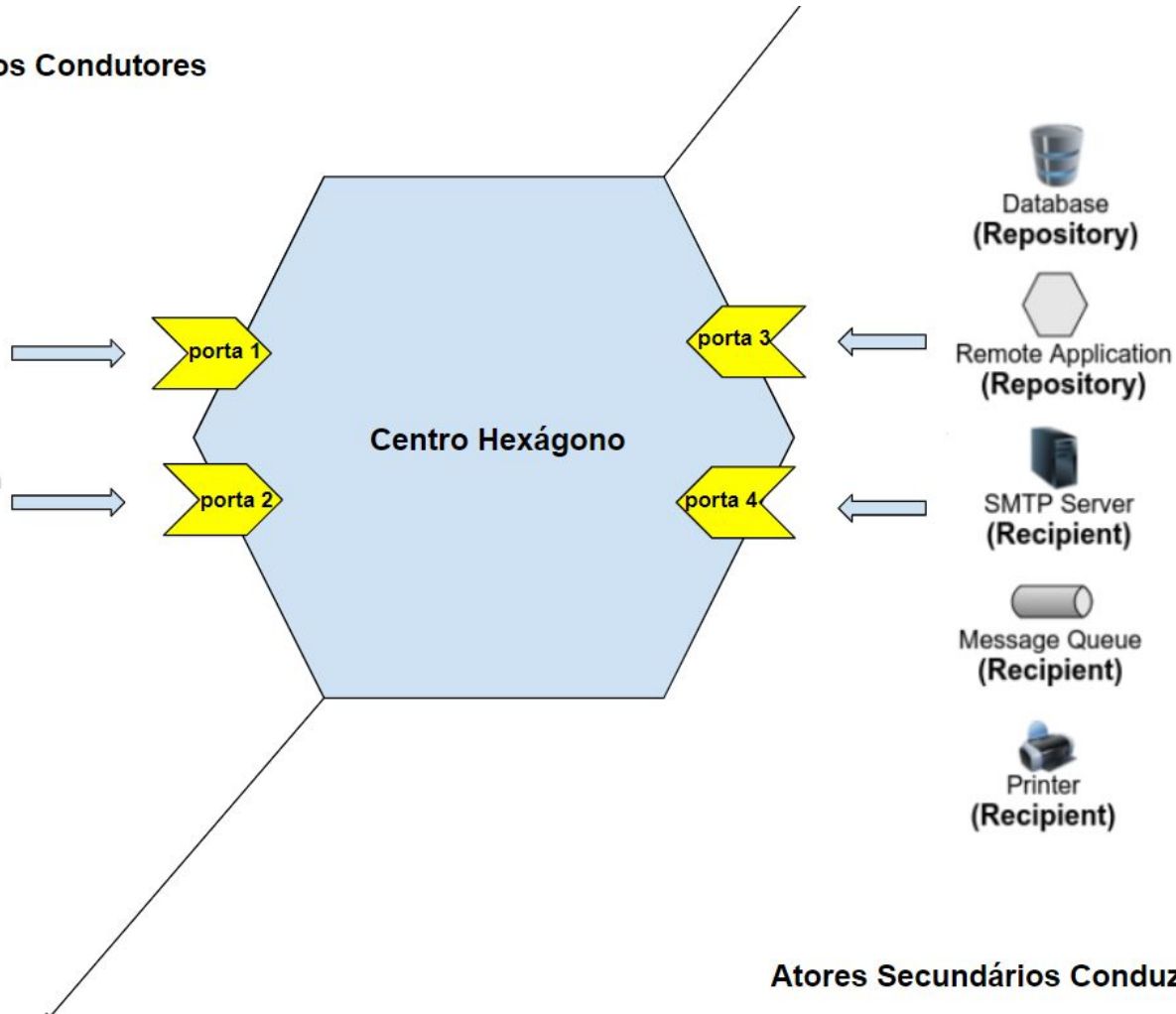
Princípios e Metáforas: Portas

Portas

A arquitetura hexagonal estabelece o princípio que, a comunicação "de fora para dentro" deve ser feita única e exclusivamente através de uma metáfora chamada "porta".

Assim, as interações entre todos os atores e o hexágono são organizadas no limite do hexágono (Edge) pelo motivo de estarem interagindo com o aplicativo. Cada grupo de interações com um determinado objetivo / intenção é uma porta.

Atores Primários Condutores



Atores Secundários Conduzidos

Na Prática: Portas

As portas são **interface OOP**, classes que contém apenas os protótipos dos métodos sem corpo, que polimorficamente definem um **contrato de uso**, e ao mesmo tempo isolam e encapsulam como aquele determinado serviço está sendo executado.

Portanto, a arquitetura deve seguir as regras de encapsulamento polimórfico.

Portas

Arquitetura hexagonal estabelece 2 tipos de portas:

1. Porta Primária Condutor (Driver)
2. Porta Secundária Dirigida (Driven)

Arquitetura Hexagonal

Princípios e Metáforas: Portas Primárias

Portas

Porta Primária Condutor (Driver)

Tem esse nome por que são **chamados pelos atores primários que formam o lado do usuário da solução**. Portas primárias formam a **API** (Application Programming Interface) da solução para entrada no hexágono.

Portas primárias são **funções e ou eventos ofertados** pela solução que permitem alterar objetos, ou atributos e execução de processos relacionado a lógica principal.

Exemplos: adicionar item carrinho de compra, pagar fatura, transferência bancária e etc.

Portas

Porta Primária Condutor (Driver)

São nomeadas como caso de usos (Use Case).

“Um caso de uso é um ferramenta para o levantamento dos requisitos funcionais do sistema. Ele descreve a funcionalidade proposta a ser desenvolvido, Um caso de uso é um documento narrativo que descreve a sequência de eventos de um ator que usa um sistema para completar um processo.” [WIKI]

Alistair Cockburn propôs que as portas condutores sejam mapeadas em formatos de caso de usos, uma vez que eles refletem exatamente as funções e ou eventos que solução precisa fazer.

Arquitetura Hexagonal

Princípios e Metáforas: Portas Secundárias

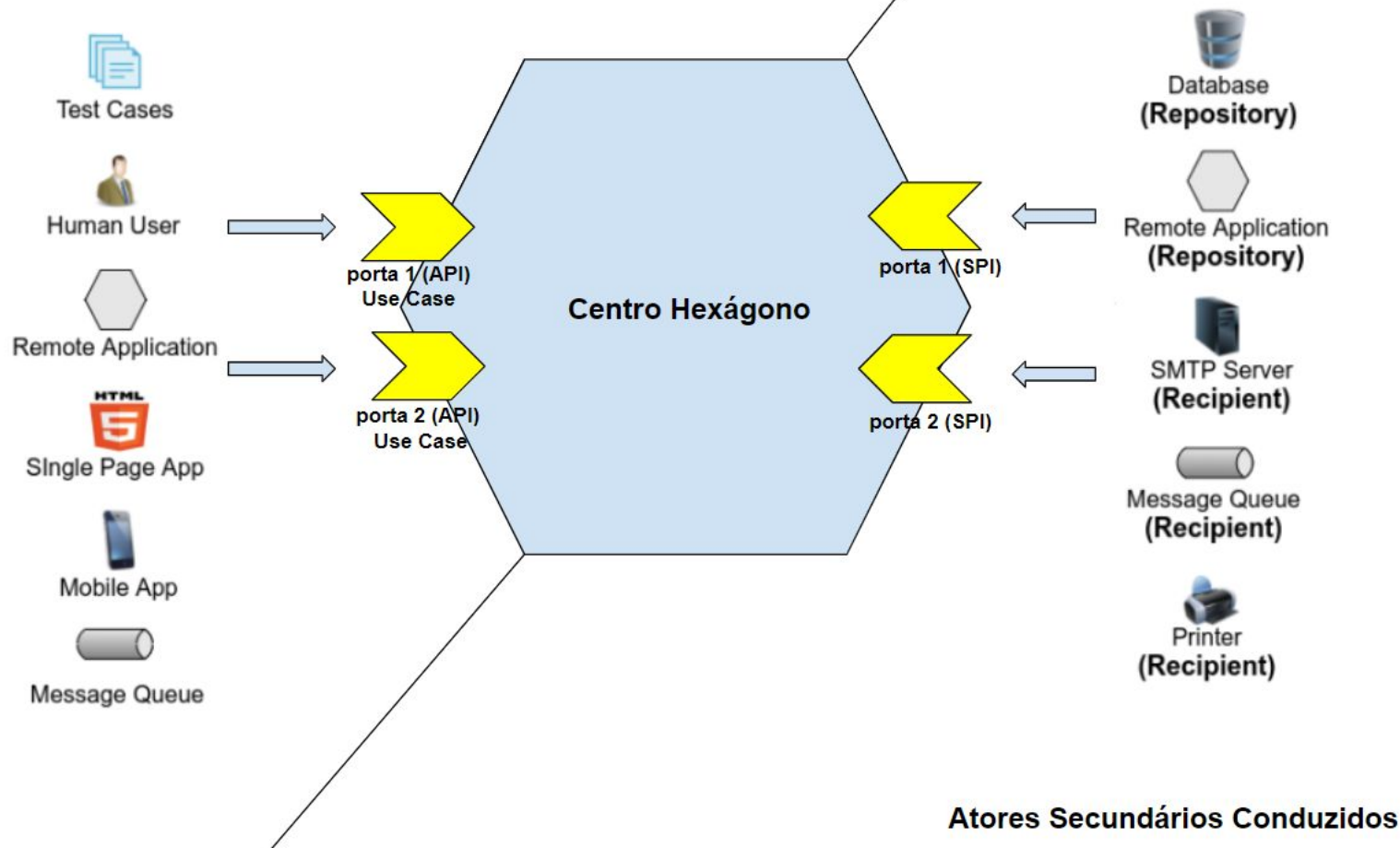
Portas

Porta Secundária Dirigida (Driven)

Tem esse nome por que são chamados pelo hexágono para executar serviços externos. Portas secundárias formam a **SPI** (Service Provider Interface) que é requerida pela solução.

Portas secundárias possibilitam acessar dados em banco de dados relacionais, nosql, serviços web http, stmp, sistema de arquivos, por exemplos inserir cliente, pesquisar fatura, procurar cliente inativos e etc.

Atores Primários Condutores



Na Prática: Portas

Todas as classes OOP que implementam a interfaces de **portas primárias** (casos de uso) devem estar **dentro do hexágono**, pois elas são agnósticos a tecnologias e são usadas definir e redirecionar as chamadas externas para **dentro das operações** de negócio oferecidas pela solução.

Todas as classes OOP que implementam a interfaces de **portas secundárias** devem estar **fora do hexágono**, pois elas usam alguma **tecnologia específica** e convertem chamada de negócio, agnóstico a tecnologia em alguma necessidade infraestrutural e externa a solução.

Portas

Hexagono 100% Isolado

Assim, essas interfaces atuam como isoladores explícitos entre o interior e o exterior da solução, tanto para o lado esquerdo de entrada, quanto o lado direito de saída, cumprindo o principal objetivo da visão hexagonal que é ter “core” da solução 100% isolado, independente de tecnologia de GUI, dispositivos externos, serviços infraestruturais e podendo ser orientado a TDD.

Portas

Dúvidas e comentários?



Arquitetura Hexagonal

Princípios e Metáforas: Adaptadores

Adaptadores

A arquitetura hexagonal estabelece uma metáfora chamada de "**adaptador**".

Um adaptador é um componente de software que permite uma **tecnologia externa** interaja com uma porta do hexágono. Dada uma porta, deve haver um adaptador para cada tecnologia que se deseja usar.

Adaptadores

Arquitetura hexagonal estabelece 2 tipos de adaptadores:

1. Adaptador Condutor (Driver)
2. Adaptador Dirigido (Driven)

Adaptadores

Adaptador Condutor (Driver)

É o componente usado para converter uma solicitação de tecnologia específica em uma solicitação agnóstica e pura de sistema para uma porta condutora, traduzindo dados de entradas externos para dentro da solução.

Objetivo: Responsável por fazer integração do lado de fora para dentro do hexágono. Segue exemplos:

Adaptadores

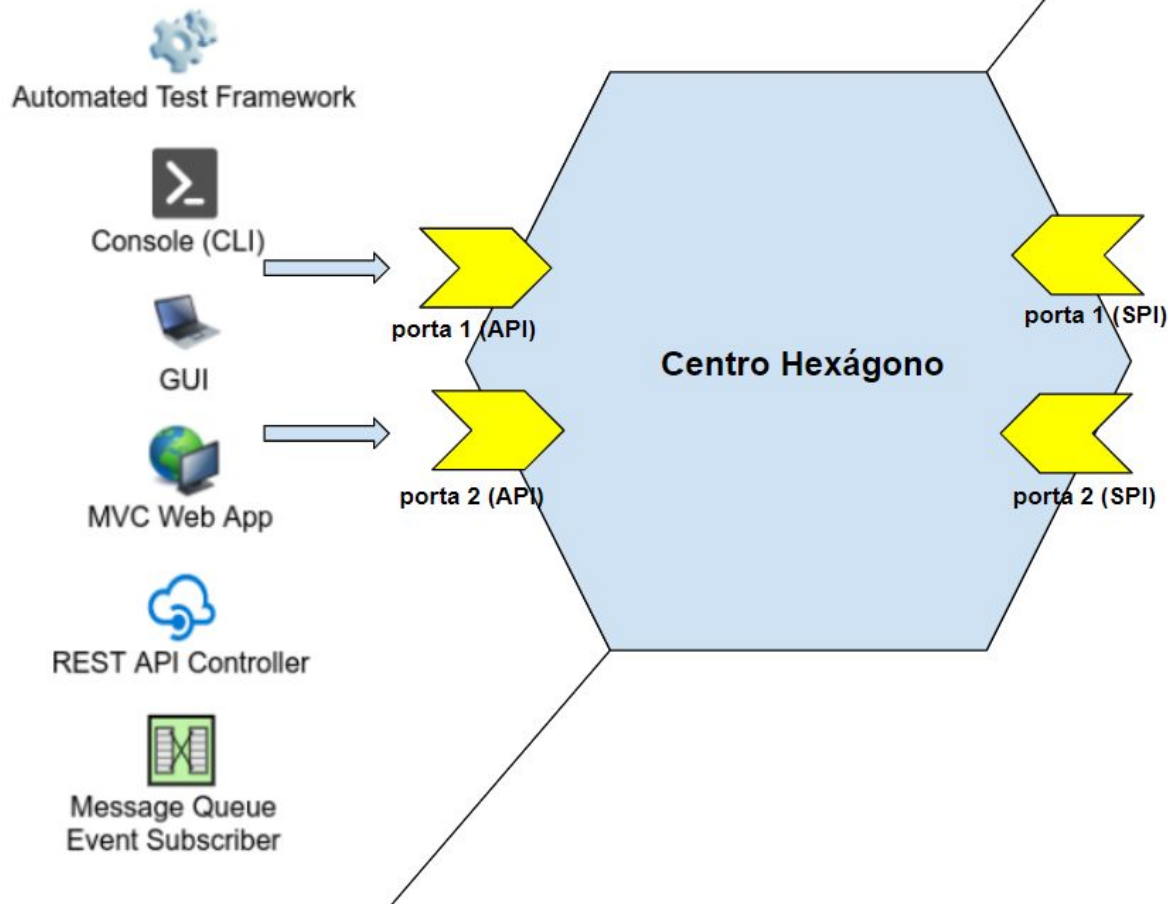
1. **Suite de testes automatizada TDD**: converte casos de teste em solicitações para a porta condutora.
2. **GUI de um aplicativo desktop**: converte eventos acionados por componentes gráficos para a porta condutora.
3. **GUI de um aplicativo Web MVC**: o controlador recebe do view a ação solicitada pelo usuário e o converte em uma solicitação para a porta condutora.
4. **Fila de Mensageria**: uma mensagem da fila é recebida do serviço de mensagens e converte em uma solicitação para a porta condutora.

Na Prática: Adaptadores

Os **adaptadores condutores** são classes OOP que usam **frameworks e tecnológicas específicas**, convertendo dados vindos dessas filosofias externas para dentro do hexágono, repassando as operações para a porta primária. Exemplos:

- classes teste junit
- classes desktop Swing ou JavaFx
- classes web JSF managed beans
- classes json end point rest jax-rs
- classes json end point rest spring mvc

Adaptadores Condutores



Adaptadores

Adaptador Condutor (Driver)

Para cada porta condutora, deve haver pelo **menos dois adaptadores**:

1. um para testar o comportamento via TDD.
2. um usando a tecnologia real requerida pela solução.

Adaptadores

Adaptador Dirigido (Driven)

É o componente usado para converter chamadas **de dentro do
solução para fora**, usando serviços de infraestrutura tecnológicos externos a solução.

Objetivo: Responsável por fazer integração de dentro do hexágono para o lado de fora. Exemplos:

Adaptadores

- **Um adaptador SQL**: Implementa uma porta orientada para persistir dados acessando um banco de dados SQL.
- **Um adaptador de email**: Implementa uma porta orientada para notificar as pessoas enviando um email para elas.
- **Um adaptador App-To-App**: Implementa uma porta controlada para obter alguns dados, solicitando-os a um aplicativo remoto.
- **Um adaptador fila de mensageria**: Implementa uma porta controlada para enviar mensagens a outras soluções.

Na Prática: Adaptadores

Os **adaptadores dirigidos** são classes OOP que implementam as interfaces de portas dirigidos e que usam **frameworks e tecnologias específicas**, dando o suporte para aquelas necessidades de chamadas externas. Exemplos:

- Classe DAO via JDBC.
- Classe EAO via JPA.
- Classe envio via JavaMail usando SMTP.
- Classe envio de sms consumidor de um web services jax-ws.
- Classes cliente consumidor de um rest end point via jax-rs.
- Classes cliente consumidor de um rest end point via Sprint RestTemplate.

Adaptadores Condutores


Automated Test Framework


Console (CLI)



GUI


MVC Web App


REST API Controller


Message Queue
Event Subscriber

porta 1 (API)

porta 2 (API)

Centro Hexágono

porta 1 (SPI)

porta 2 (SPI)


Mock Adapter


SQL Adapter


Email Adapter


App-To-App
Adapter


Message Queue
Event Publisher

Adaptadores Conduzidos

Adaptadores

Adaptador Conduzido (Driven)

Para cada porta dirigidos devemos escrever pelo **menos dois adaptadores**:

1. um para o dispositivo do mundo real.
2. outro para um simulado que imita o comportamento real, chamado de mock.

Filosofia - Adaptadores

Funcionam com um “adaptador de tomada” que fazem “ponte” para que o hexágono possa ter input de dados do lado esquerdo e ter acesso aos serviços de infraestrutura do lado direito.



Adaptadores

Dúvidas e comentários?



Arquitetura Hexagonal

Princípios e Metáforas: Adaptadores Simulados

Adaptadores Simulados - Mock

Arquitetural hexagonal promove que a solução seja desenvolvida **independentemente de seus dispositivos externos**, fazendo que com a equipe de desenvolvimento foque no desenvolvimento do requisitos de negócio, ignorando dependências externas técnicas e infra estruturais.

Um mock é componente usado **emular ao hexágono os serviços reais ofertados de um dispositivos externo** de modo que, o módulo do hexágono possa ser **100% desenvolvido, testado e homologado sem precisar** instalar, configurar ou usar serviços, tecnologias ou infra estruturas pendentes a solução.

Na Prática: Adaptadores Simulados - Mock

Mocks são classes OOP que implementam as interfaces de portas dirigidos com objetivo de **assinar o contrato de serviços emulando aqueles determinados serviços**. Exemplos:

- Classe DAO que ao invés de usar JDBC, faz a persistência em memória usando HashMap.
- Classe DAO que ao invés de usar JDBC, faz a persistência em um sgdb embutido Hsqldb.
- Classe envio de email que ao invés de usar JavaMail, faz `System.out.println`.

Adaptadores Simulados - Mock

Dúvidas e comentários?



Arquitetura Hexagonal

Princípios e Metáforas: Fluxo de Execução

Adaptadores Condutores



Automated Test Framework



Console (CLI)



GUI



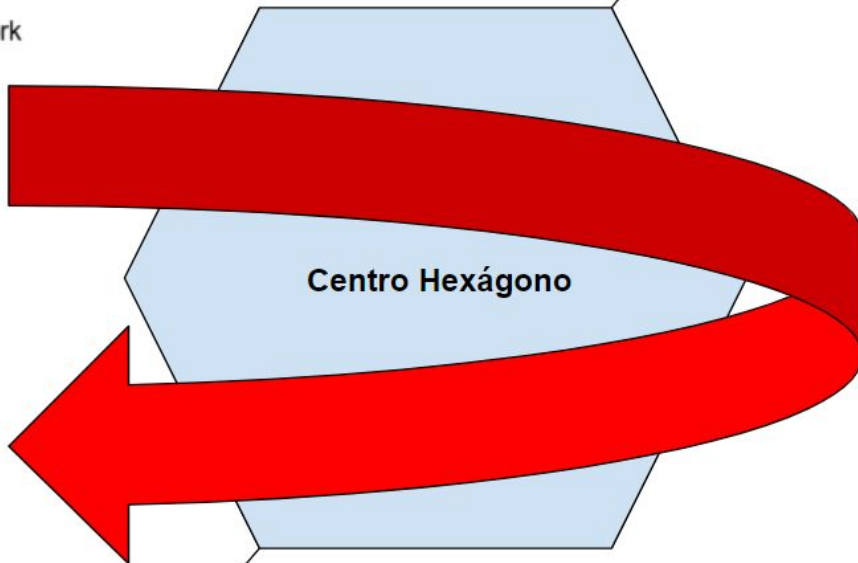
MVC Web App



REST API Controller



Message Queue
Event Subscriber



Mock Adapter



SQL Adapter



Email Adapter



App-To-App
Adapter



Message Queue
Event Publisher

Adaptadores Conduzidos

Fluxo de Execução

O que isso quer dizer?

Na teoria, queríamos que o core do sistema, o hexágono ficasse isolado e não dependente de ninguém, mas na prática (Runtime), **o hexágono na verdade depende do lado direito.** Assim, estamos furando a 2º e 3º regra de dependência.

Fluxo de Execução

Na teoria:

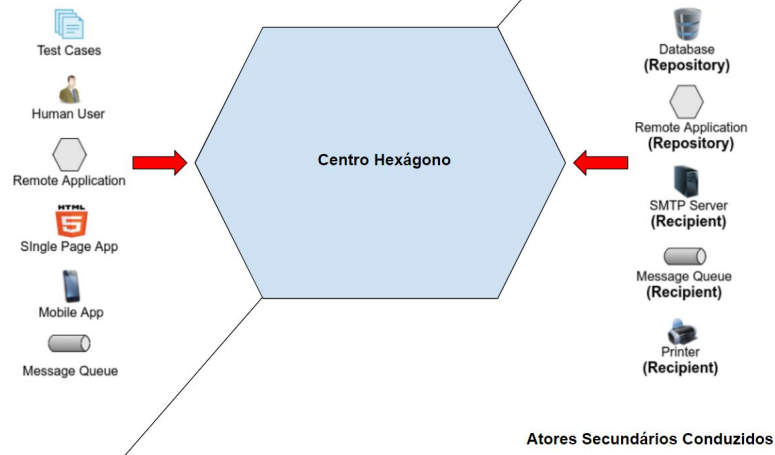
1. Lado esquerdo, os atores primários dependem do hexágono.
2. Lado direito, os atores secundários dependem do hexágono.
3. O centro, o hexágono não depende de ninguém, só dele mesmo.

Na prática:

- O centro, o **hexágono depende do lado direito**.
- Lado direito, os **atores secundários não dependem** do hexágono.

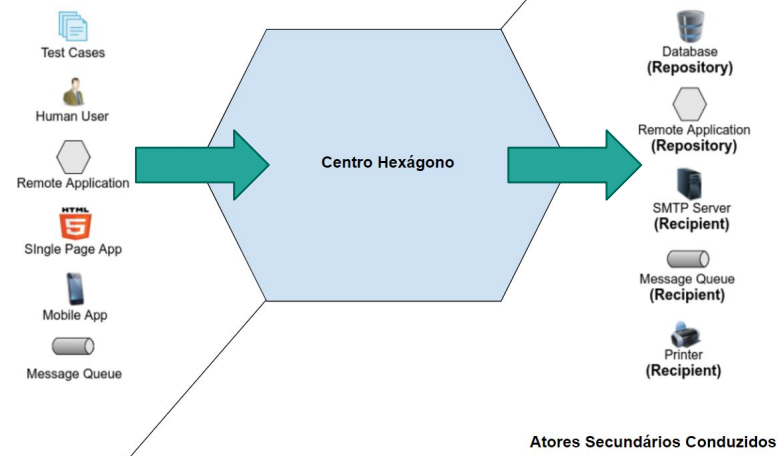
TEORIA

Atores Primários Condutores



PRÁTICA

Atores Primários Condutores



Fluxo de Execução

Como resolver isso?

É possível ou vamos viver na utopia?



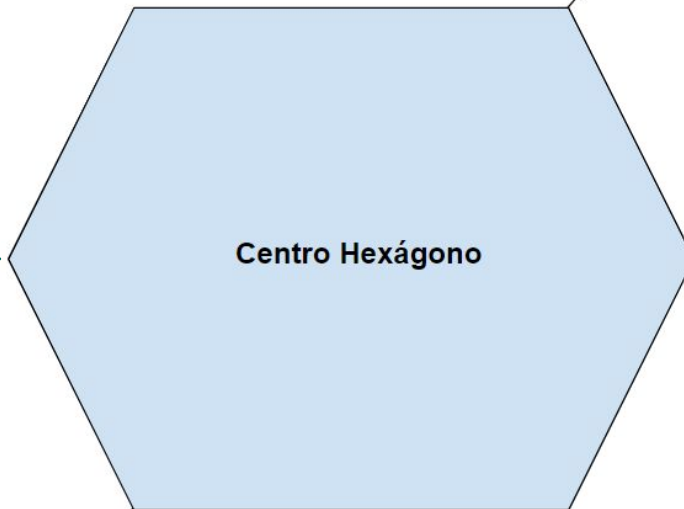
Fluxo de Execução

Inversão de Controle (Inversion of Control - IoC)

É um padrão arquitetural, uma técnica de arquitetura de software usada para **inverter uma linha de dependência em um bloco arquitetural**. Se o componente A [->depende->] B, usando IoC é possível fazer inverter a seta, fazendo com que A [<- dependa IoC<-] B.

A arquitetura hexagonal aplica IoC, estabelecendo o princípio modular que **o lado de fora direito tem dependência ao hexágono via IoC!**

Atores Primários Condutores



IoC



Atores Secundários Conduzidos

Fluxo de Execução

Problema resolvido na teoria, na documentação e prática (runtime)

1. Lado esquerdo, os atores primários dependem do hexágono
2. Lado direito, os atores secundários dependem do hexágono **via IoC**.
3. O centro, o hexágono não depende de ninguém, só dele mesmo.



Fluxo de Execução

Dúvidas e comentários?



Arquitetura Hexagonal

Princípios e Metáforas: Dependências Configuráveis

Dependências Configuráveis

Com a uso de portas polimórficas, IoC e o princípio de flexibilidade de intercâmbio de adaptadores, a arquitetura hexagonal estabelece o uso de um gerenciador de dependências de forma dinâmica e configurável em ambos o lados.

Dependências Configuráveis

Lado condutor primário

A conversa é iniciada pelo driver (ator primário), portanto, o adaptador do driver tem uma dependência configurável na porta do driver, que é uma interface implementada pelo aplicativo.

Dependências Configuráveis

Lado dirigido secundário

A conversa é iniciada pelo aplicativo, portanto, o aplicativo tem uma dependência configurável na porta acionada, que é uma interface implementada pelo adaptador acionado do ator secundário.

Dependências Configuráveis

Dentro do hexágono

De forma opcional, dentro do hexágono pode haver uma sub organização que pode fazer uso de dependência configurável entre os próprios componentes internos.

Dependências Configuráveis

Dependência configurável é o princípio mais importante em que se baseia a arquitetura hexagonal, pois permite que o hexágono **seja dinamicamente dissociado de qualquer tecnologia de front-end e infra-estrutura.**

E esse desacoplamento é o que possibilita o principal objetivo da arquitetura, ou seja, ter um aplicativo que possa **ser executado por vários drivers e testado isoladamente** de destinatários / repositórios.

Na Prática: Dependências Configuráveis

O projeto hexagonal **deve fazer uso de qualquer serviços de IoC** de sua plataforma de preferência para montar a rede de objetos (wiring) de execução da solução, tando os build de desenvolvimento, testes, homologação e produção. Exemplos Java:

- CDI
- Spring IoC
- Pico Container

Dependências Configuráveis

Dúvidas e comentários?



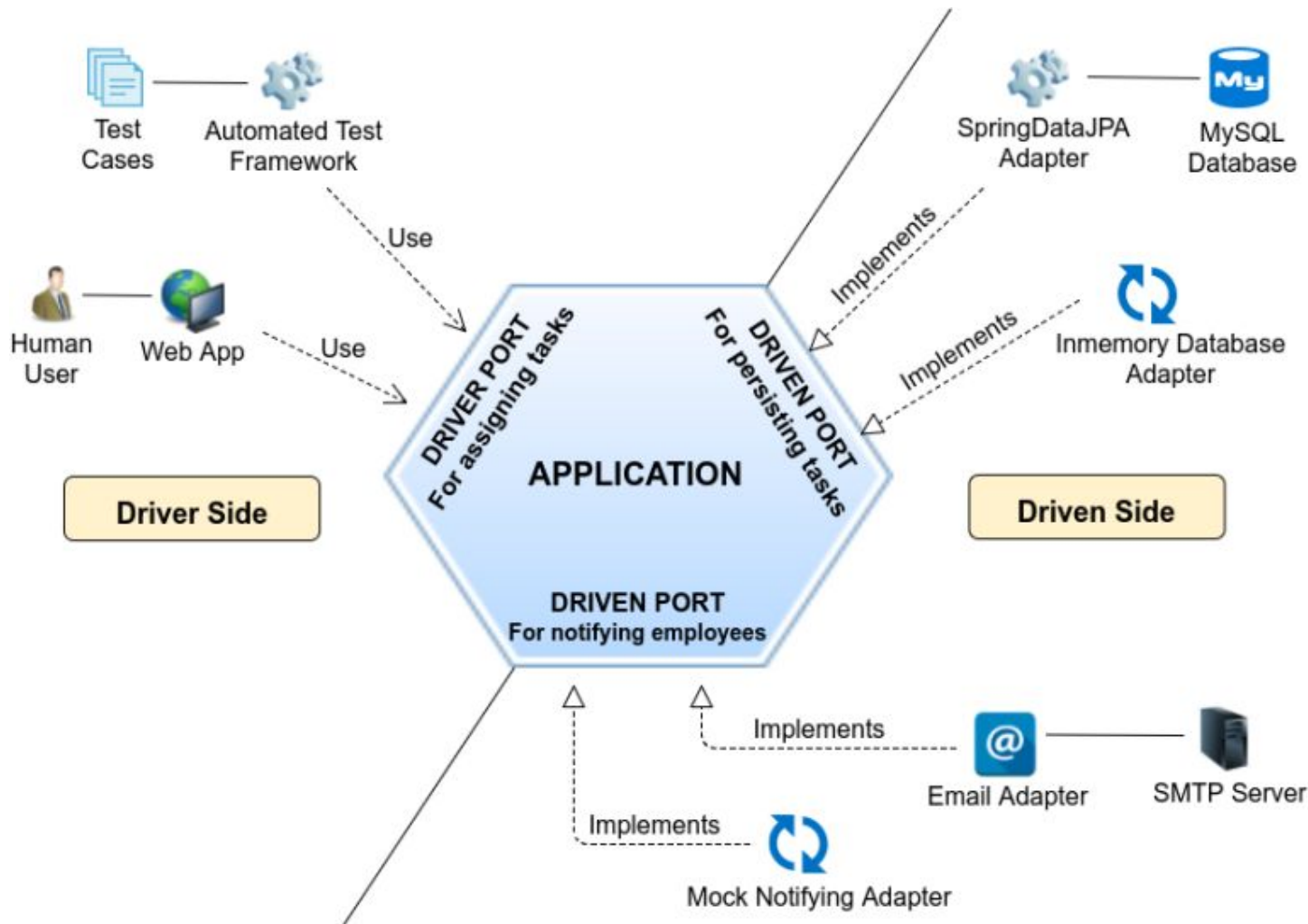
Arquitetura Hexagonal

Princípios e Metáforas: Exemplo Real

Exemplo Real

Aplicativo de Gerenciador de tarefas

É um aplicativo simples com interface web, utilizado pelos funcionários de uma empresa para atribuição de tarefas entre si. Quando um funcionário recebe uma tarefa, o aplicativo envia um email para ele. Quando um funcionário move uma tarefa para outro, o outro recebe um email.



Exemplo Real

Dúvidas e comentários?



Arquitetura Hexagonal

Princípios e Metáforas: Quando usar ou não usar?

Quando usar ou não usar?

Projetos **temporais**, de **menor porte** ou **legados**, no qual não se tenha previsão de alteração de tipo de front-end ou de back services, poderia ser justificável não o uso dessa abordagem.

Quando usar ou não usar?

Projetos que fazem uso de **regras de negócio dentro de banco de dados**, amarrados na tecnologia, marca e provedor proprietário não se justifica uso dessa abordagem.

Quando usar ou não usar?

Projetos de médio e grande porte, que supostamente possuem um longo ciclo de vida, e que precisaram ser modificados muitas vezes durante sua vida útil, que sofreram de erosão arquitetural e dívida técnica, justifica o uso dessa abordagem.

Pois em curto prazo, o investimento se reverte em lucro pela quantidade e velocidade das mudanças.

Quando usar ou não usar?

Projetos de **qualquer porte** que precisam ser desenvolvidos sem **amarrações** com **front-end** e **back services**, justificam o uso dessa abordagem.

Quando usar ou não usar?

Dúvidas e comentários?



Arquitetura Hexagonal

Princípios e Metáforas: Leitura Complementar

Leitura Complementar

Atualmente 06/2019 não existem livros sobre o assunto.

Blog oficial do autor está em reconstrução -

<https://alistair.cockburn.us>.

Cópia da documentação oficial -

https://web.archive.org/web/20080914152611/http://alistair.cockburn.us/index.php/Hexagonal_architecture

Leitura Complementar

Eu traduzi a documentação do pattern no blog acrescentando algumas melhorias:

<https://fernandofranzini.wordpress.com/2019/04/09/arquitura-hexagonal/>

Arquitetura Hexagonal

Teoria e Fundamentos: Conclusão

Conclusão

Não existe bala de prata. A arquitetura hexagonal tem se destacado como uma ótima opção de arquitetura, pois o custo é baixo (poucos pontos negativos) e retorno é alto é rápido (muitos pontos positivos).

É uma das várias opções de padrões arquiteturais encontradas no mercado hoje: DDD, Onion e Clear Architecture.

Conclusão

Como eu faço para implementar tudo isso em Java agora?



Conclusão

Este curso é o **módulo 1** foi a **teoria** e **conceitos**.

Os próximos módulos serão responsáveis por ensinar a implementação da arquitetura hexagonal usando JDK, TDD, IntelliJ, Java 12, Módulos Java, JUnit, Spring Framework (Test, IoC, Transactions e JDBC), JavaFX e HSQDB.

Aguardo você!

Fechamento

