Arquitetura Hexagonal

Módulo 2 - Projeto e Design Java

Módulos

1. ~~Teoria, conceitos, valores e metáforas.~~
2. Projeto e Design Java.
3. Implementação com Java.

Abertura

Nesse módulo 2 veremos:

- Opções de design.
- Sequência de desenvolvimento.
- Sequência de build.
- Dicas de artefatos, serviços, classes, projeto, patterns e design.

Arquitetura Hexagonal

Opções de Design

Opções de Design

A arquitetura hexagonal **não define** diretivas de criação dos projetos, organização de pacotes, componentes, camadas, módulos, interfaces, classes, uso de padrões de projetos e tecnologias.

A arquitetura hexagonal **somente cobre princípios de isolamento e separação**, usando metáforas com o objetivo de fazer uma solução não depender de **front-end** e **serviço externos**, elaborando uma solução flexível e testável.

Opções de Design

Você fica livre para organizar seu projeto de forma que quiser, balanceando os **prós e contras** de cada opção, usando seu **know-how** e **os recursos** que sua plataforma oferece, desde que: cada opção não **fure** os princípios hexagonal.

Opções de Design

Para você **desenvolvedor sênior ou arquiteto**, seja livre e faça suas escolhas de projeto e design...

Opções de Design

Para você **desenvolvedor junior**, vou apresentar e comentar resumidamente as principais decisões:

Opções de Design

Organização de Projeto: Segue as opções do mais simples ao mais sofisticado:

1. Um projeto com os 3 módulos, separado por pacotes simples.
2. Um projeto com os 3 módulos, separado por java modules 9.
3. Três projetos diferentes, uma para cada módulo, usado jar manual.
4. Três projetos diferentes, uma para cada módulo, usando jar via maven system.
5. Três projetos diferentes, uma para cada módulo, usando jar via maven repositorio local jfrog - https://jfrog.com/open-source/
6. N projetos diferentes, uma para cada módulo, um para cada front-end, um para cada back-service: database, nosql, sms, webservice etc, usando jar manual, maven system ou maven repositório local jfrog.

Opções de Design

Organização de Projeto:

Não existe bala de prata, certo ou errado, pior ou melhor. Analise cada um e faça sua decisão baseado em prós e contras, contexto e necessidade.

Opções de Design

Organização de Pacotes:

1. Por tipos (Type)
2. Por camada (Layer)
3. Por serviço (Feature)
4. Por metáforas hexagonal (Ports, Adapters, etc)
5. Miscelânea

Opções de Design

Organização de Pacotes:

Links úteis de discussão sobre o assunto:

- https://proandroiddev.com/package-by-type-by-layer-by-feature-vs-package-by-layered-feature-e59921a4dffa
- https://dzone.com/articles/package-by-feature-is-demanded
- https://dzone.com/articles/package-your-classes-feature

Opções de Design

Organização de Pacotes:

Não existe bala de prata, certo ou errado, pior ou melhor. Analise cada um e faça sua decisão baseado em prós e contras, contexto e necessidade.

Opções de Design

Implementação das regras de negócio dentro do hexágono:

1. Transactions Script - EAA Pattern.
2. Domain Model - EAA Pattern.
3. Service Layer - EAA Pattern.
4. Anemic Domain Model - EAA Pattern.
5. Domain Driven Design - DDD.
6. Clean Architecture (Entities e User cases)

Opções de Design

Implementação das regras de negócio dentro do hexágono:

Links:

- https://martinfowler.com/eaaCatalog/transactionScript.html
- https://martinfowler.com/bliki/AnemicDomainModel.html
- https://martinfowler.com/eaaCatalog/serviceLayer.html
- https://martinfowler.com/eaaCatalog/domainModel.html
- https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

Opções de Design

Implementação das regras de negócio dentro do hexágono:

Livros:

- https://www.amazon.com.br/Padr%C3%B5es-Arquitetura-Aplica%C3%A7%C3%B5es-Corporativas-Martin/dp/8536306386
- https://www.amazon.com.br/Domain-Driven-Design-Eric-Evans/dp/8550800651
- https://www.amazon.com.br/Clean-Architecture-Craftsmans-Software-Structure/dp/0134494164

Opções de Design

Implementação das regras de negócio dentro do hexágono:

Não existe bala de prata, certo ou errado, pior ou melhor. Analise cada um e faça sua decisão baseado em prós e contras, contexto e necessidade.

Opções de Design

Organização das Portas Primárias - Casos de Usos

Segue as opções do mais simples ao mais sofisticado:

1. Uma única interface com várias operações distintas.
2. Várias interfaces, cada uma agrupando operações relacionadas.
3. Uso do padrão de projeto: Command Bus
4. Uso do padrão de projeto: Command Query.

Opções de Design

Organização das Portas Primárias - Casos de Usos

Links:

- https://medium.com/eleven-labs/cqrs-pattern-c1d6f8517314
- https://tactician.thephpleague.com/
- https://barryvanveen.nl/blog/49-what-is-a-command-bus-and-why-should-you-use-it

Opções de Design

Organização das Portas Primárias - Casos de Usos

Não existe bala de prata, certo ou errado, pior ou melhor. Analise cada um e faça sua decisão baseado em prós e contras, contexto e necessidade.

Opções de Design

Transferência de dados

Segue as opções do mais simples ao mais sofisticado para transferir dados do lado esquerdo (front-end) para dentro do hexágono.

A interfaces de casos de uso podem ser projetados usando as seguintes opções:

Opções de Design

Transferência de dados

1. Variáveis simples máximo de 4 [Joshua Bloch Java Effective item 40].
2. Padrão de Projeto Parameter Object.
3. Padrão de Projeto Value Object.
4. Expor objeto de domínio como parâmetro.
5. Objetos expansíveis como HashMap.
6. Padrão de projeto “Typesafe Heterogeneous Container Pattern”.
7. Objeto dinâmicos no uso de linguagens dinâmicas: groovy Expando.

Opções de Design

Transferência de dados

Links:

- https://refactoring.guru/introduce-parameter-object
- https://deviq.com/value-object/
- https://mrhaki.blogspot.com/2009/10/groovy-goodness-expando-as-dynamic-bean.html
- https://fernandofranzini.wordpress.com/2013/02/28/generics-item-29/
- http://wavelino.coffeecup.com/pdf/EffectiveJava.pdf

Opções de Design

Transferência de dados

Não existe bala de prata, certo ou errado, pior ou melhor. Analise cada um e faça sua decisão baseado em prós e contras, contexto e necessidade.

Opções de Design

Uso de objetos polimórficos de backservices

Dentro do hexágono, existe duas opções para se consumir os serviços dos objetos de backservices:

1. Fazer IoC desses objetos em **qualquer outro objeto** dentro do hexágono, **por livre demanda**. Assim todos estes terão dependência com os backservices e deverão entrar no gerenciamento IoC. Isso pode ter prós e contras, dependendo de como foi organizado o hexágono.

Opções de Design

Uso de objetos polimórficos de backservices

2. **Limitar** o IoC desses objetos **somente nas implementações das portas primárias de caso de uso**. Assim, somente esses objetos terão dependência com os backservices e deverão entrar no gerenciamento IoC. Outros objetos, de outros serviços dentro do hexágono, não terão dependência com o backservices e poderão funcionar fora do IoC. Isso pode ter prós e contras dependendo de como foi organizado o hexágono.

No geral, **essa é a melhor opção**, pois organiza melhor a estrutura e reduz os acoplamentos no hexágono.

Opções de Design

Uso de objetos polimórficos de backservices

Não existe bala de prata, certo ou errado, pior ou melhor. Analise cada um e faça sua decisão baseado em prós e contras, contexto e necessidade.

Opções de Design

Dúvidas e comentários?

Arquitetura Hexagonal

Classes e Serviços Dentro do Hexágono

Classes e Serviços Dentro do Hexágono

Precisamos entender **como organizar** os grupos de serviços e as dependências **dentro do hexágono**:

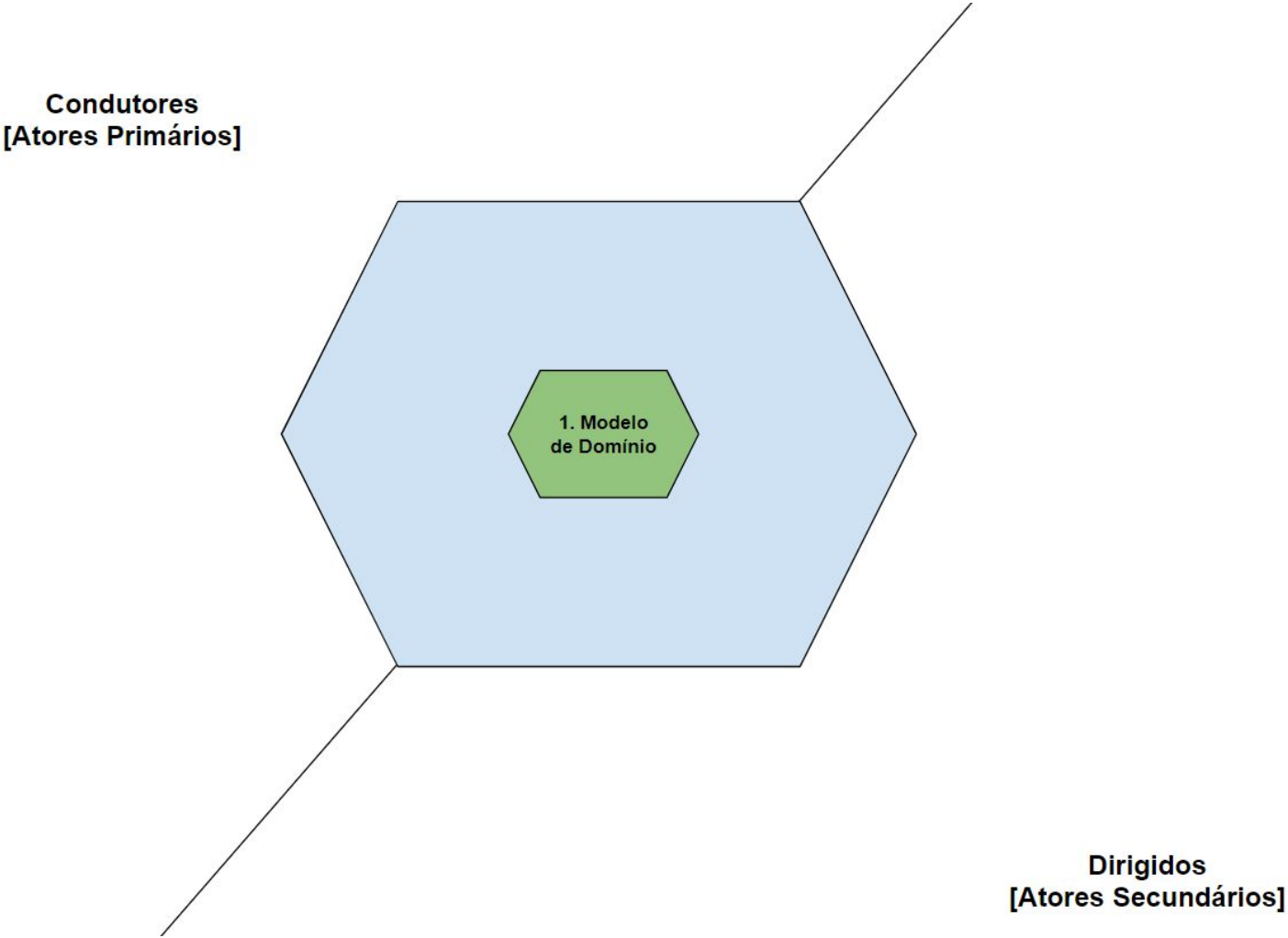
Dentro do hexágono faço a sugestão de **fazer 4 divisões** de serviços:

1. Modelo de Domínio [EAA - Domain Model].
2. Serviço de Domínio [EAA - Service Layer].
3. Portas Dirigidas + implementações de adaptadores mokados.
4. Portas Condutoras + implementações controladoras.

Classes e Serviços Dentro do Hexágono

1. Modelo de Domínio

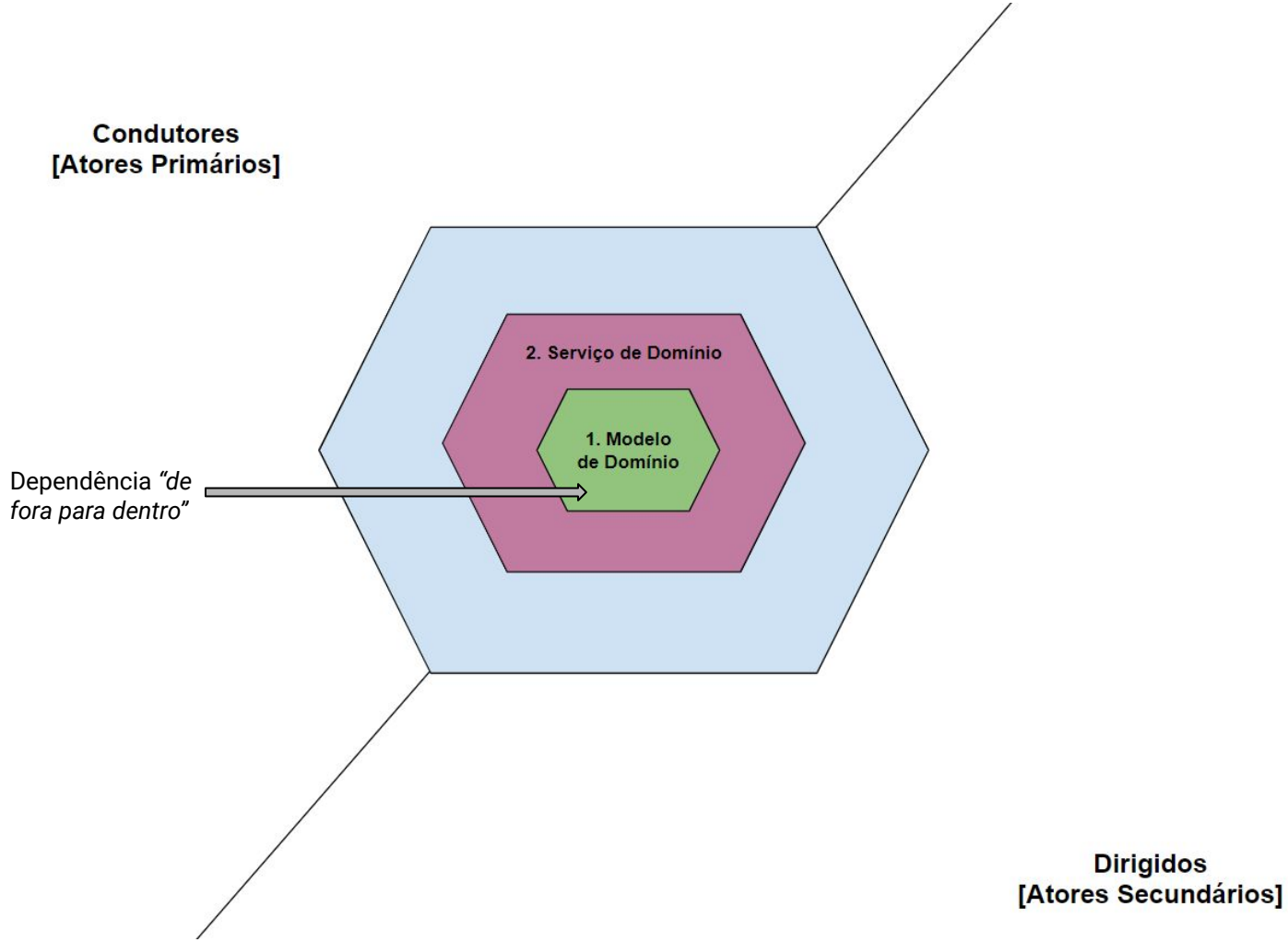
- Responsável por agrupar interfaces, classes, objetos de domínio, objetos de valor e enums que implementam exclusivamente as **regras de negócios**.
- Mapeia o mundo real em abstrações, estados e entidades usando os conceitos de POO.
- Não depende de nada, somente do **próprio serviço**.


Condutores
[Atores Primários]
1. Modelo
de Domínio
Dirigidos
[Atores Secundários]

Classes e Serviços Dentro do Hexágono

2. Serviço de Domínio

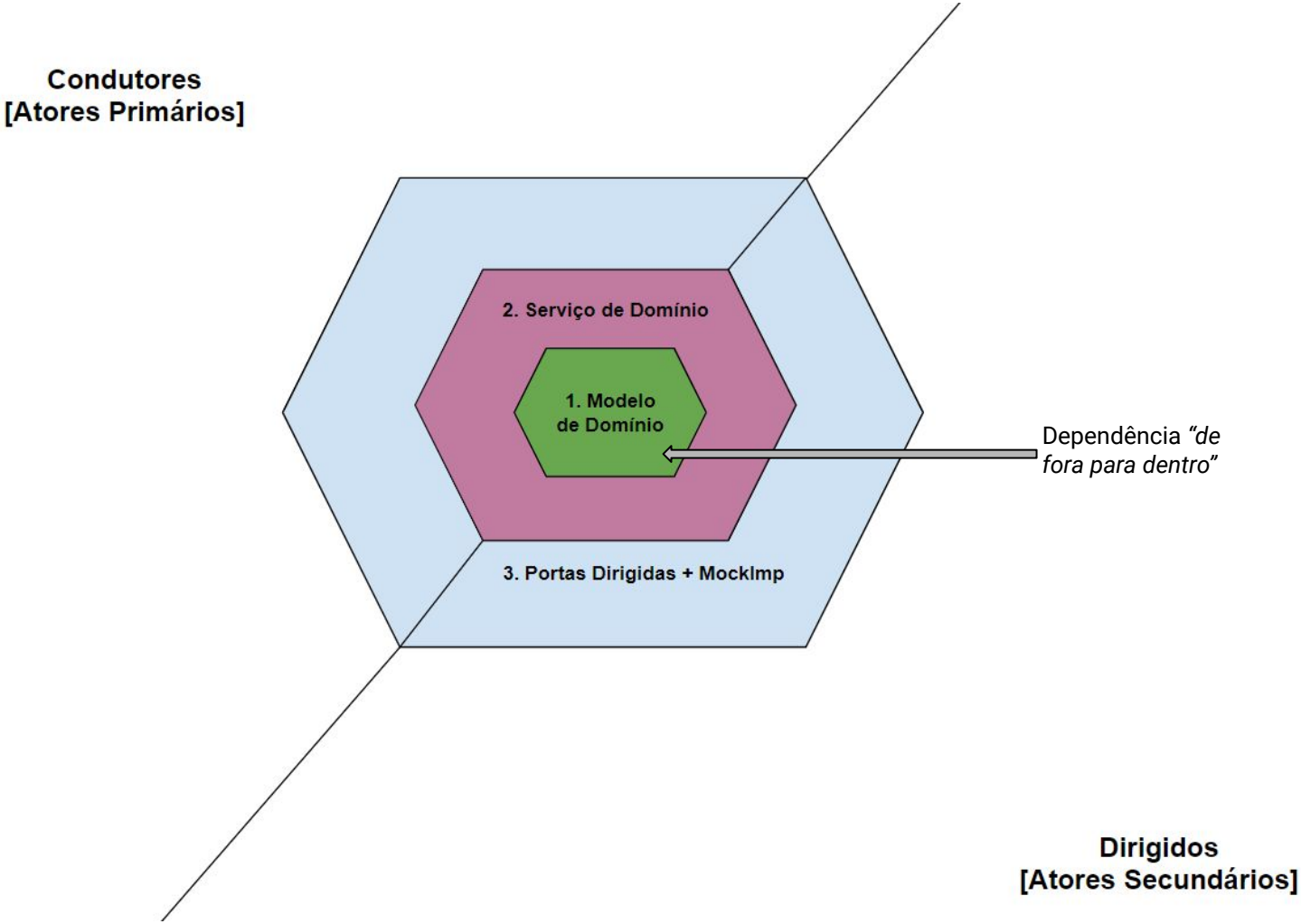
- Responsável por agrupar interfaces, classes, objetos de valor e enums que implementam exclusivamente os **processos de negócio**.
- Esse serviço recebe implementação de **processos compostos** que envolvem executar **várias operações de domínios diferentes** e que normalmente não se encaixam dentro do modelo de domínio.
- Mapeia o mundo real em abstrações, estados e entidades usando os conceitos de POO.
- Depende do serviço de **Modelo de Domínio** e do **próprio serviço**.


Condutores
[Atores Primários]
2. Serviço de Domínio
1. Modelo
de Domínio
Dependência *"de fora para dentro"*
Dirigidos
[Atores Secundários]

Classes e Serviços Dentro do Hexágono

3. Portas Dirigidas

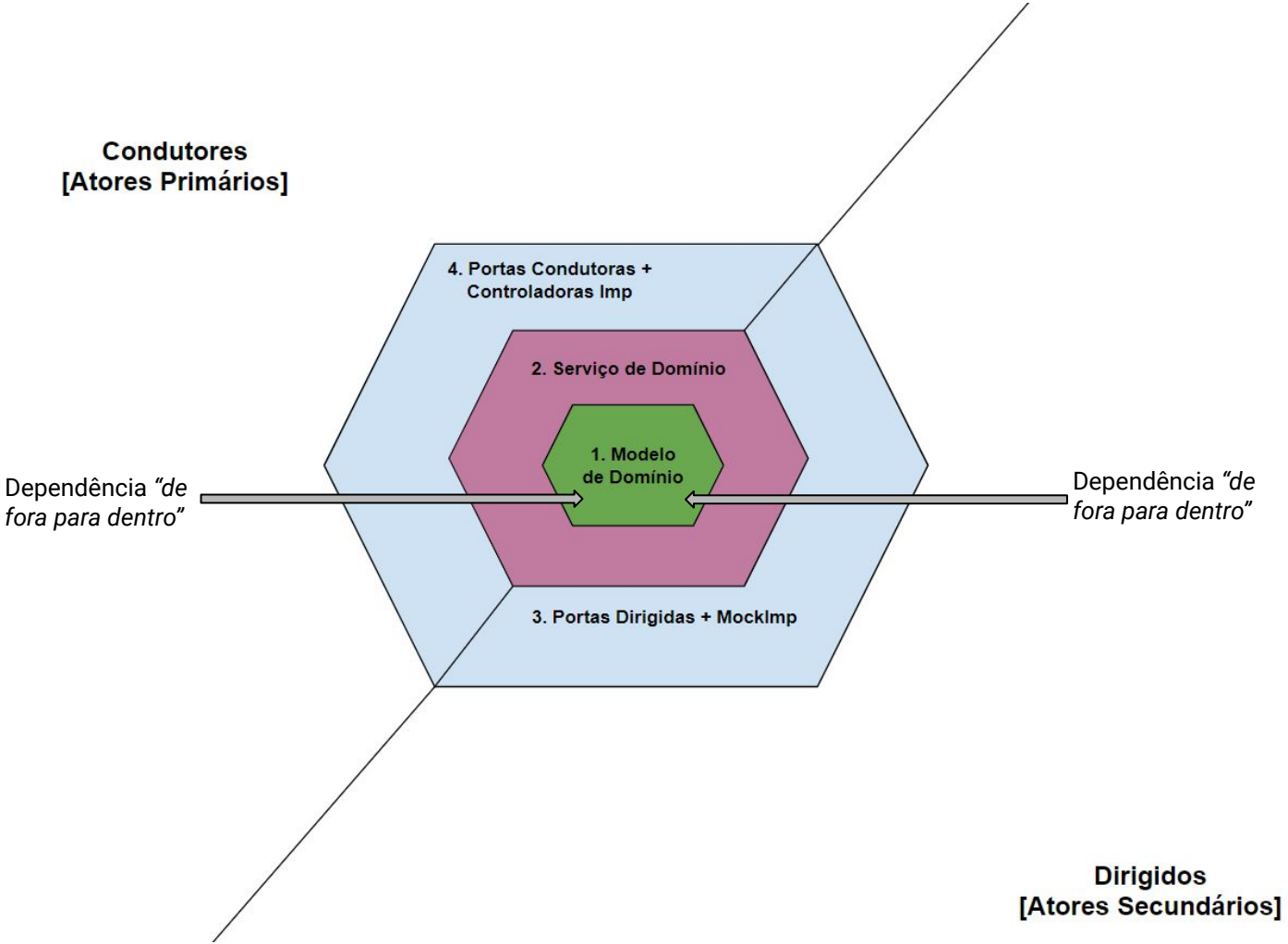
- Responsável por agrupar as **interfaces** (SPI), classes e enums que definem polimorficamente todos os serviços “backservices” que a sistema precisa para funcionar.
- Mapeia os serviços externos em abstrações, usando os conceitos de POO.
- Responsável por agrupar **implementações das interfaces "mocks"**, emulando todos os serviços infra estruturais, de forma com que o sistema possa ser executado e testado sem ter infraestrutura.
- Pode depender do **Modelo de Domínio** , **Serviço de Domínio** e do **próprio serviço**.


Condutores
[Atores Primários]
2. Serviço de Domínio
1. Modelo
de Domínio
3. Portas Dirigidas + MockImp
Dependência *"de fora para dentro"*
Dirigidos
[Atores Secundários]

Classes e Serviços Dentro do Hexágono

4. Portas Condutoras - Casos de Uso

- Responsável por agrupar as **interfaces** (API), classes e enums que definem as **funções e ou eventos** ofertados pelo sistema.
- Mapeia os serviços ofertados em abstrações, usando os conceitos de POO.
- Responsável por agrupar as classes implementações das interfaces controladoras, que traduzem um pedido externo do hexágono para dentro, executando as operações.
- **Único e exclusivamente responsável por executar operações nas interfaces de portas dirigidas, em prol de processar funções e ou eventos**.
- Pode depender do **Modelo de Domínio**, **Serviço de Domínio** e **Portas Dirigidas**.


Condutores
[Atores Primários]
4. Portas Condutoras +
Controladoras Imp
2. Serviço de Domínio
1. Modelo
de Domínio
3. Portas Dirigidas + MockImp
Dependência *"de fora para dentro"*
Dependência *"de fora para dentro"*
Dirigidos
[Atores Secundários]

Classes e Serviços Dentro do Hexágono

Dúvidas e comentários?

Arquitetura Hexagonal

Serviços Fora do Hexágono - Lado Inferior Direito

Classes e Servicos Lado Direito

Precisamos entender **como organizar** os agrupadores de serviços e as dependências fora do hexágono: **no lado direito inferior**:

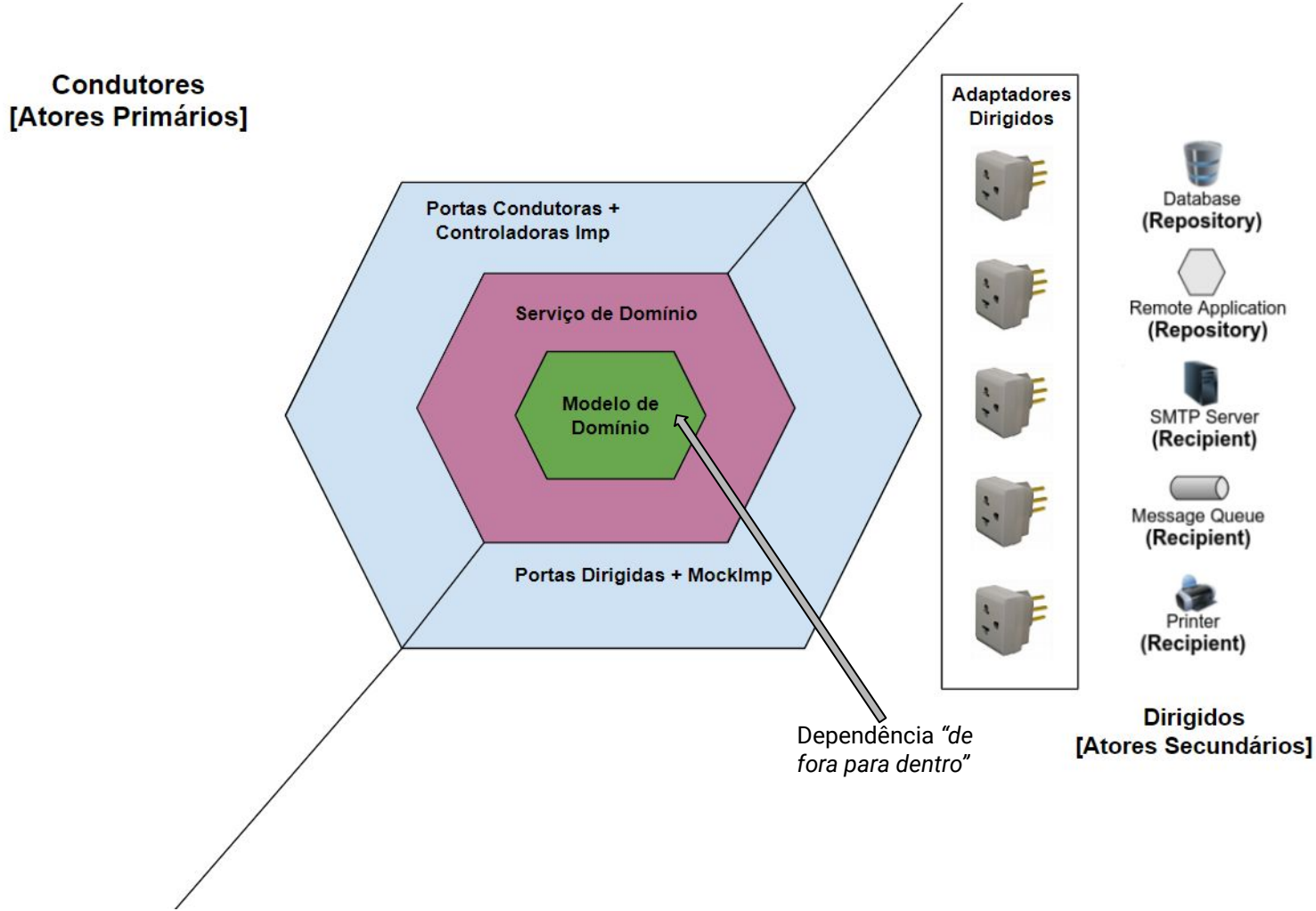
Teremos um único serviço:

1. Adaptadores Dirigidos.

Classes e Servicos Lado Direito

Adaptadores Dirigidos:

- Responsável por agrupar as classes implementações das interfaces SPI das portas dirigidas, usando as tecnologias, frameworks, protocolos e interagindo efetivamente com os serviços e ou dispositivos externos.
- Pode depende do serviço de **Modelo de Domínio** e **Serviço de Domínio**.
- Para cada dispositivo de interação, haverá um serviço adaptador.


Condutores
[Atores Primários]
Portas Condutoras +
Controladoras Imp
Serviço de Domínio
Modelo de
Domínio
Portas Dirigidas + MockImp
Adaptadores
Dirigidos
Database
(Repository)
Remote Application
(Repository)
SMTP Server
(Recipient)
Message Queue
(Recipient)
Printer
(Recipient)
Dependência "de
fora para dentro"
Dirigidos
[Atores Secundários]

Classes e Servicos Lado Direito

Dúvidas e comentários?

Arquitetura Hexagonal

Serviços Fora do Hexágono - Lado Superior Esquerdo

Classes e Servicos Lado Esquerdo

Precisamos entender **como organizar** os agrupadores de serviços e as dependências fora do hexágono: **lado esquerdo superior**:

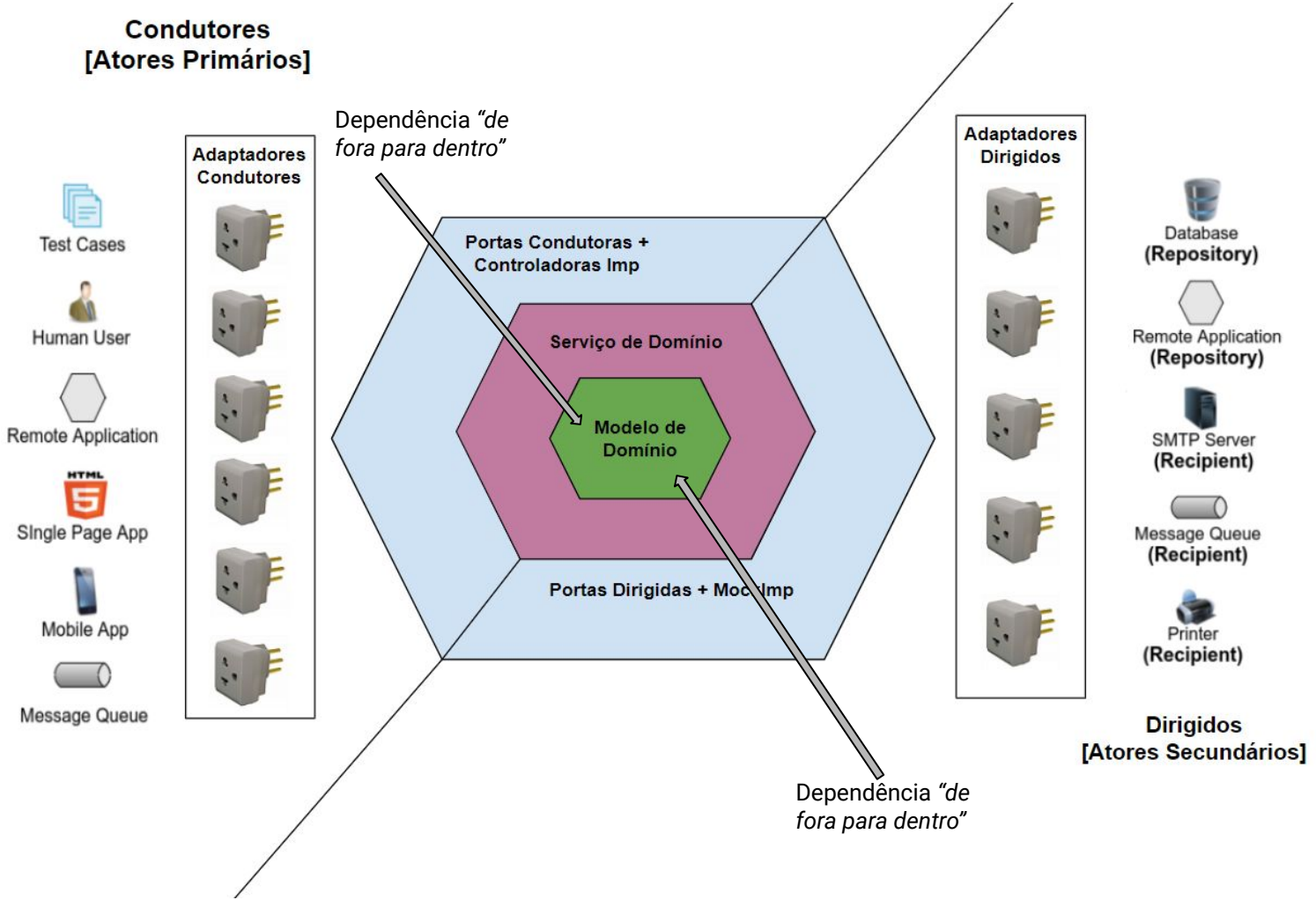
Teremos um único serviço:

1. Adaptadores Condutores.

Classes e Servicos Lado Esquerdo

Adaptadores Condutores:

- Responsável por agrupar as classes adaptadoras, usando as tecnologias, frameworks, widget específicos de **front-end** ou **input de dados**, traduzindo as informações vindo destes mecanismos para o sistema.
- Depende somente das interfaces das portas Condutoras.
- Para cada tipo de front-end, haverá um serviço adaptador.


Condutores
[Atores Primários]
Adaptadores
Condutores
Test Cases
Human User
Remote Application
SIngle Page App
Mobile App
Message Queue
Dependência "de
fora para dentro"
Portas Condutoras +
Controladoras Imp
Serviço de Domínio
Modelo de
Domínio
Portas Dirigidas + Mock Imp
Dependência "de
fora para dentro"
Adaptadores
Dirigidos
Database
(Repository)
Remote Application
(Repository)
SMTP Server
(Recipient)
Message Queue
(Recipient)
Printer
(Recipient)
Dirigidos
[Atores Secundários]

Classes e Servicos Lado Esquerdo

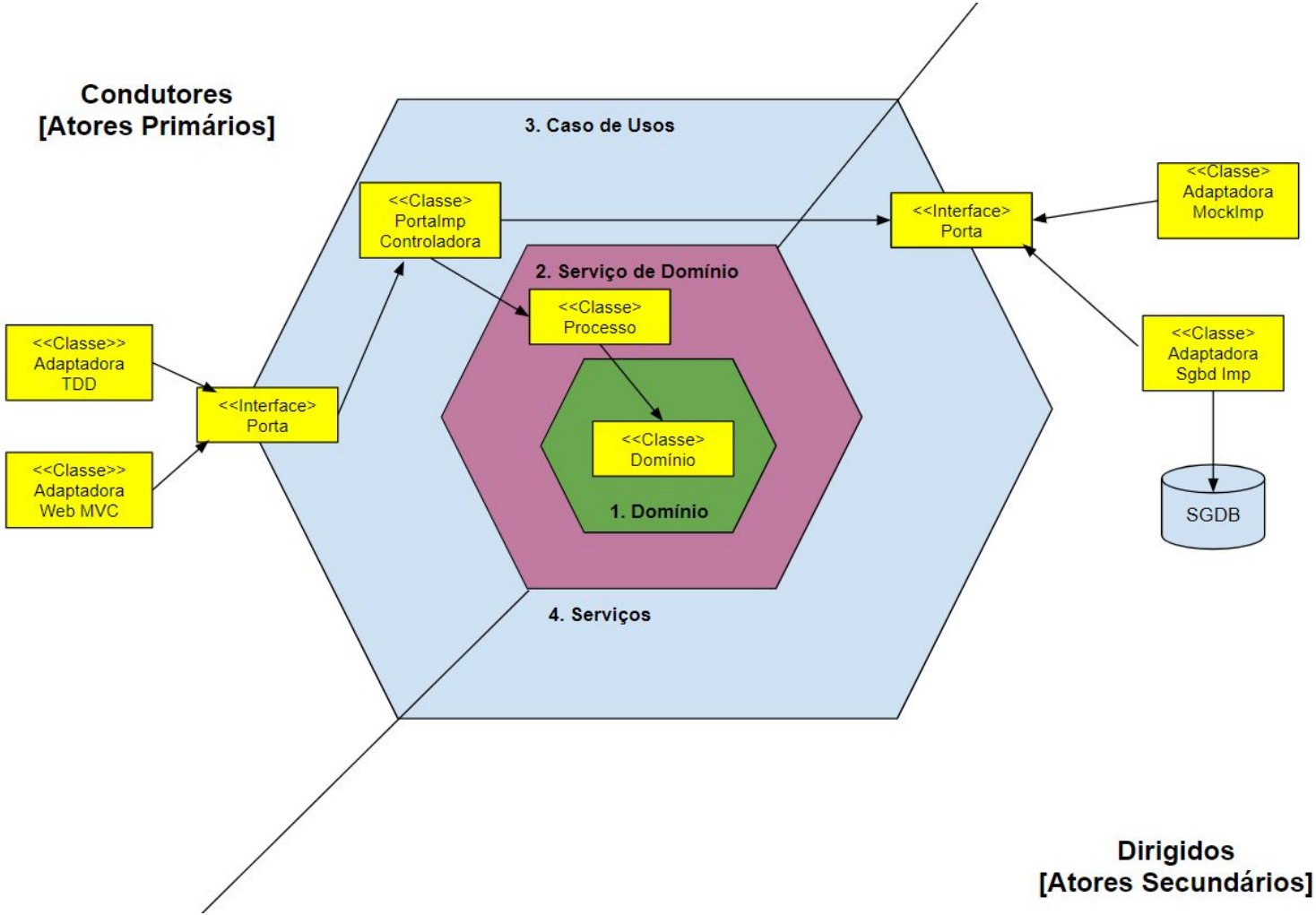
Dúvidas e comentários?

Arquitetura Hexagonal

Fluxo de Execução

Fluxo de Execução

Para esclarecer todas as possíveis dúvidas, segue agora a **sequência de execução do modelo hexagonal,** usando os agrupamentos e nomenclaturas utilizadas nas explicações anteriores:


Condutores
[Atores Primários]
3. Caso de Usos
<<Classe>>
Adaptadora
TDD
<<Classe>>
Adaptadora
Web MVC
<<Interface>
Porta
<<Classe>
PortaImp
Controladora
2. Serviço de Domínio
<<Classe>
Processo
<<Classe>
Domínio
1. Domínio
4. Serviços
<<Interface>
Porta
<<Classe>
Adaptadora
MockImp
<<Classe>
Adaptadora
Sgbd Imp
SGDB
Dirigidos
[Atores Secundários]

Fluxo de Execução

Dúvidas e comentários?

Arquitetura Hexagonal

Sequência de Desenvolvimento

Sequência de Desenvolvimento

- PASSO 1 - Hexágono (Centro)
- PASSO 2 - Front-End (Lado Esquerdo)
- PASSO 3 - Back-Services (Lado Direito)
- PASSO 4 - Build de Produção

Sequência de Desenvolvimento

PASSO 1 - Hexágono (Centro)

O ponto de partida é implementar o hexágono como uma **caixa preta**, com as interfaces de portas definidas em torno dela, tanto no lado esquerdo, quanto no lado direito. Ou seja, implementar, testar e validar os serviços do sistema, sem tela, sem infraestrutura, usando TDD e mocks.

Build 1: Adaptador Testes -> Sistema <- Adaptadores Mocks.

Sequência de Desenvolvimento

PASSO 2 - Front-End (Lado Esquerdo)

Implementar os adaptadores primários condutores, plugando a na porta do hexágono. Ou seja, implementar, testar e validar a tecnologia e frameworks de front-end, sem serviços de infraestrutura.

Build 2:

Adaptador Interface Gráfica -> Sistema <- Adaptadores Mocks.

Sequência de Desenvolvimento

PASSO 3 - Back-Services (Lado Direito)

Implementar os adaptadores secundários dirigidos, plugando a na porta do hexágono. Ou seja, implementar, testar e validar a tecnologia e frameworks de referente as serviços necessários para a solução.

Build 3:

Adaptador Interface Gráfica -> Sistema <- Adaptadores Serviços Homologação.

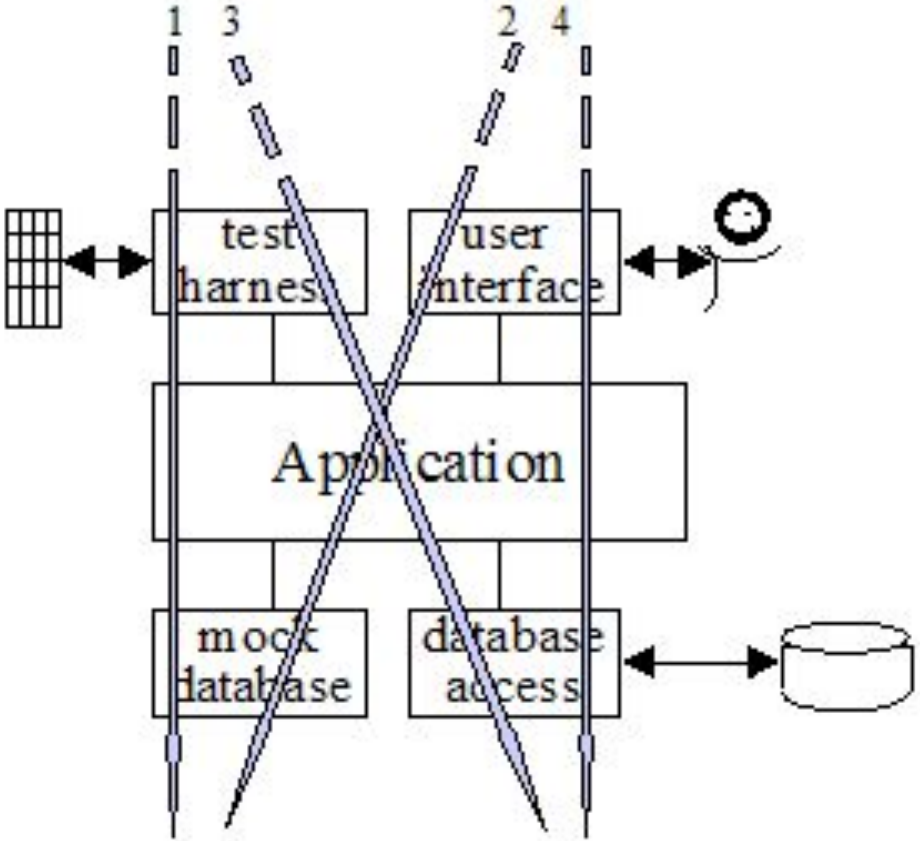
Sequência de Desenvolvimento

PASSO 4 - Build de Produção

Configurar o ambiente de produção e fazer o build oficial final:

Build 4:

Adaptador Interface Gráfica -> Sistema <- Adaptadores Serviços Produção.


1
3
2
4
test harnes
user interface
Application
mock database
database access

Sequência de Desenvolvimento

Dúvidas e comentários?

Arquitetura Hexagonal

Projeto e Design Java: Fechamento

Fechamento