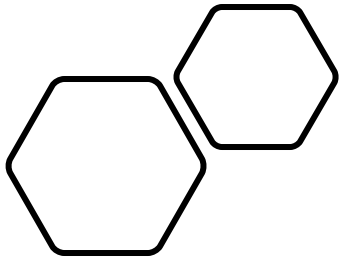
# Arquitetura de Software

Prof. Bernardo Copstein

# Arquitetura CLEAN



Leituras recomendadas:

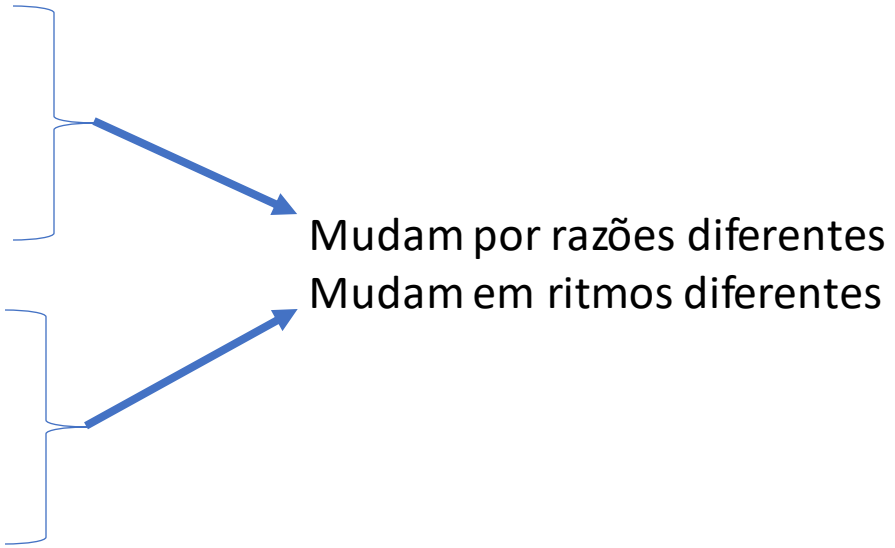
- Martin, Robert C.. Clean Architecture (Robert C. Martin Series) . Pearson Education.

# Relembrando:

- A arquitetura de um sistema de software é a “forma” que é dada ao sistema por aqueles que o constroem. Esta “forma” aparece na divisão do sistema em seus componentes, na forma como estes componentes se organizam e se comunicam entre si. O propósito desta “forma” é facilitar o desenvolvimento, a instalação (deploy), operação e manutenção do sistema de software.
- **A estratégia por traz desta facilitação é deixar a maior quantidade de opções em aberto pelo maior tempo possível.**

# Agrupar o que muda pela mesma razão

- Interface com o usuário
  - Mudam por razões diversas das regras de negócio
  - Mudam por razões diversas dos casos de uso
- Regras de negócio
  - Ligadas a aplicação
    - validação de campos
    - geração de relatórios
    - casos de uso
  - Ligadas ao domínio
    - cálculo de juros
    - cálculo estrutural
    - análise de solo



# Isolar o que muda por razões diferentes

- Os mecanismos de persistência
  - Servidor de BD
  - Esquema do banco
  - Linguagem de consulta
- Interface com o usuário
  - Modelo de interface
  - Tecnologia de desenvolvimento
- O conjunto de casos de uso entre si
  - São alterados por razões diversas e de forma isolada
  - Podem surgir novos sem necessidade de alterar os antigos

- Todo o desacoplamento facilita:
  - O desenvolvimento independente
  - A separação em caso de necessidade

- **Desacoplamento mantém as opções em aberto !!**

São “plug-ins” que podem ser desacoplados a qualquer momento

# *A good architect maximizes the number of decisions* ***not made***

- Não é necessário escolher o gerenciador de banco de dados no início do projeto porque o objetivo geral é “persistir os dados” (não importa como)
- Não é necessário escolher o servidor web a ser usado no início do projeto porque o objetivo geral é atender usuários distribuídos (Como? WEB? Celular? ???)
- Não é necessário adotar REST no início do projeto porque a interface com o mundo exterior deve ser agnóstica

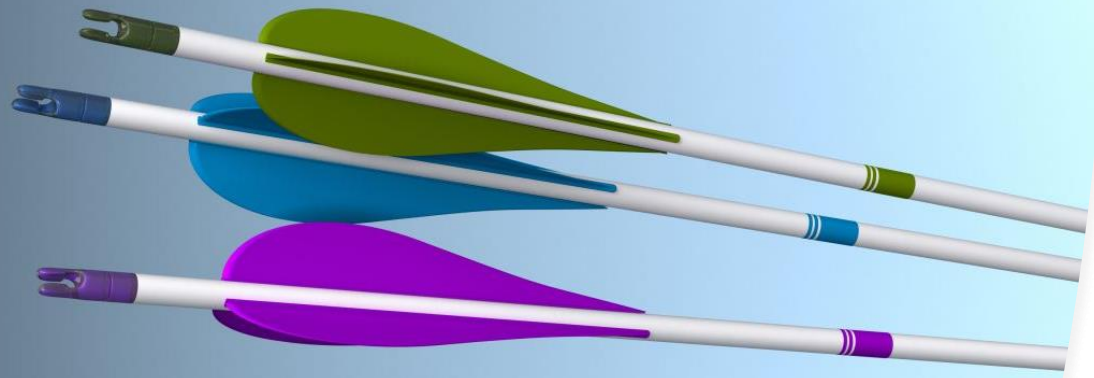
# Modos de desacoplamento

- No código fonte
  - Estrutura monolítica (organizada em camadas, módulos etc)
  - Facilita a manutenção e o desenvolvimento em paralelo
- Nível de implementação
  - jar, DLLs etc
  - Evita que mudanças em algumas partes impliquem em recompilar outras
- Nível de serviço
  - APIs independentes que se comunicam por rede
  - Permite parar apenas alguns serviços em caso de manutenção
  - Facilita o gerenciamento de escalabilidade

# Aplicação = regras de negócio + plug-ins

# Políticas e níveis: Robert Martin

- Um programa é uma descrição detalhada de uma política que coordena a transformação de entradas em saídas
  - As políticas são quebradas em políticas menores
    - Uma regra de negócio
    - Um controller
    - Um relatório
  - As políticas que mudam pelas mesmas razões devem ser agrupadas nos mesmos módulos e/ou camadas
  - Quanto mais longe uma política esta dos plug-ins maior seu nível
  - As políticas de alto nível não devem depender das de baixo nível
    - Use inversão de dependência



# Princípios de um arquiteto de software

- Bons arquitetos separam os detalhes dos objetivos e então decompõem os objetivos de maneira que os mesmos não tenham conhecimento dos detalhes de que não dependem.

- Bons arquitetos projetam os objetivos de maneira que as decisões sobre os detalhes possam ser postergadas

# Políticas e camadas

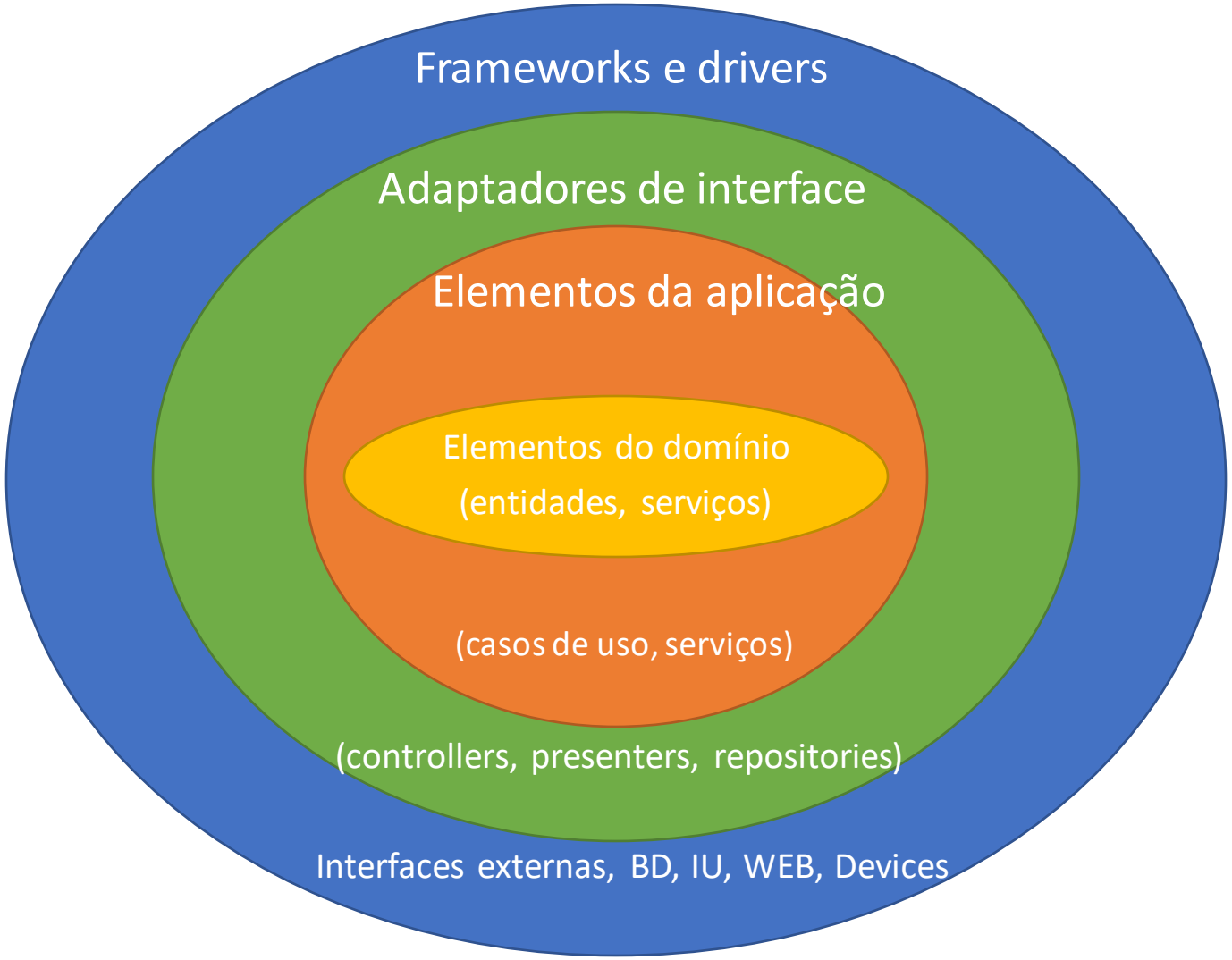
**DDD**

- Modelagem do domínio antes de tudo
- Organização do sistema em um modelo de camadas:
  - Interface
  - Aplicação
  - Domínio
  - Infraestrutura
- A dependência é de cima para baixo

**Robert Martin**

- As políticas de mais alto nível não devem depender das políticas de mais baixo nível (ou de plug-ins)
- Como aplicar inversão de controle sem burlar o modelo?
- Para resolver estas contradições:
  - “arquitetura limpa”

# A Arquitetura Limpa
(Clean Architecture)



# A Regra da Dependência

Os círculos representam diferentes áreas do software

Os círculos mais externos contém “plug-ins”, enquanto que os mais internos contém “políticas”.

Regra geral da arquitetura:

**As dependências do código-fonte devem apontar apenas para dentro, em direção a políticas de nível superior.**

Nada em um círculo mais interno deve conhecer alguma coisa em um círculo mais externo

# Elementos do domínio

- Correspondem ao círculo mais interno
- As entidades encapsulam os objetos do domínio (as abstrações pertinentes ao domínio
- Dificilmente sofrem alterações em função de solicitações relativas a aplicação. Só mudam quando os objetivos da empresa ou conhecimento sobre uma certa área mudam
- Podem conter entidades de domínio ou serviços de domínio
- Exemplos:
  - Em uma universidade atual existem **alunos**, **professores** e **técnicos**
  - Em um aeroporto chegam e partem **aviões** de **companhias aéreas**
  - O aeroporto usa um **serviço de alocação de portões** para determinar que portão deve ser ocupado por cada aeronave que chega
- DICA: se não existisse TI essas entidades fariam parte do domínio de qualquer forma

# Elementos da aplicação

- O software deste círculo contém os elementos **específicos da aplicação** (sistema sendo desenvolvido)
- Encapsulam e implementam todos os casos de uso da aplicação
- Organizam o fluxo **de** e **para** as entidades e serviços de domínio de maneira a atingir os objetivos dos casos de uso
- Não é esperado que alterações neste nível alterem as entidades ou serviços de domínio
- Também não se espera que alterações na persistência, no banco de dados ou qualquer tecnologia externa afete este nível

# Adaptadores de interface

- O software deste nível é composto por um conjunto de adaptadores que converte os dados entre o formato mais conveniente para os casos de uso e entidades, e os formatos mais convenientes para os níveis externos tais como persistência e GUI.
- Neste nível, também, iremos localizar as classes que fazem a interface com o mundo externo.

# Frameworks e drivers

- Normalmente composto pelos frameworks de persistência e GUI, entre outros
- Quaisquer serviços dependentes de uma tecnologia específica (API externa, RPC, ...) se encontram neste nível

# Dicas

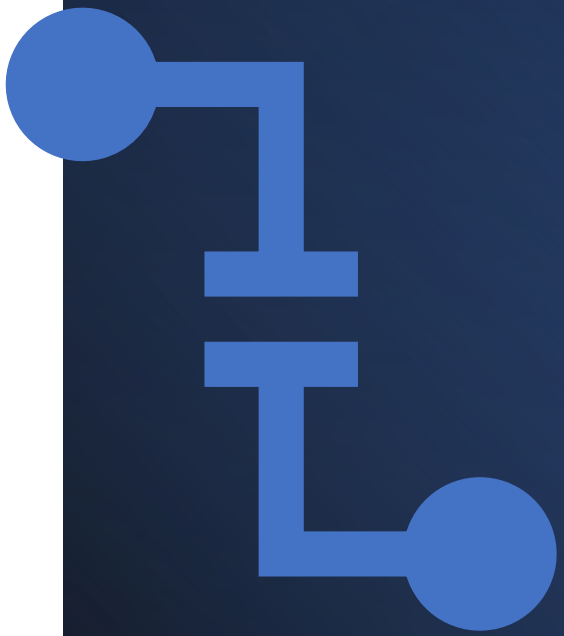
- Para saber que classes fazem parte da camada de domínio procure imaginar como funcionaria o negócio se ele fosse controlado usando-se lápis e papel (sem o uso de computadores).
- De uma maneira geral, todos os recursos que seriam necessários nesse cenário hipotético fazem parte do domínio da aplicação
- Exemplo da universidade:
    - Alunos, professores, disciplinas (conteúdos), turmas (listas de alunos e professor responsável), regras de aprovação/reprovação etc.

# Resumindo

- Iniciamos modelando o domínio do negócio
- Em seguida modelamos a lógica da aplicação
- Postergam-se ao máximo as decisões relativas aos “detalhes”: tecnologia de persistência é um detalhe; interface com o usuário é um detalhe; Web é um detalhe
- Organizamos tudo nos círculos da arquitetura limpa

# Arq. CLEAN x Opinionated Frameworks

- Normalmente os frameworks terminam por influenciar a forma pela qual modelamos os sistemas.
- Eles fornecem um conjunto de abstrações a partir da qual devemos modelar a solução
- Opinionated frameworks fazem com que não possamos postergar tanto o quanto necessário certas decisões sobre tecnologia
- O Spring é um exemplo típico: a necessidade de anotações em todas as camadas termina por antecipar a opção pelo seu uso ou não



# Exercícios de fixação

Veja as listas de exercícios sobre arquitetura limpa