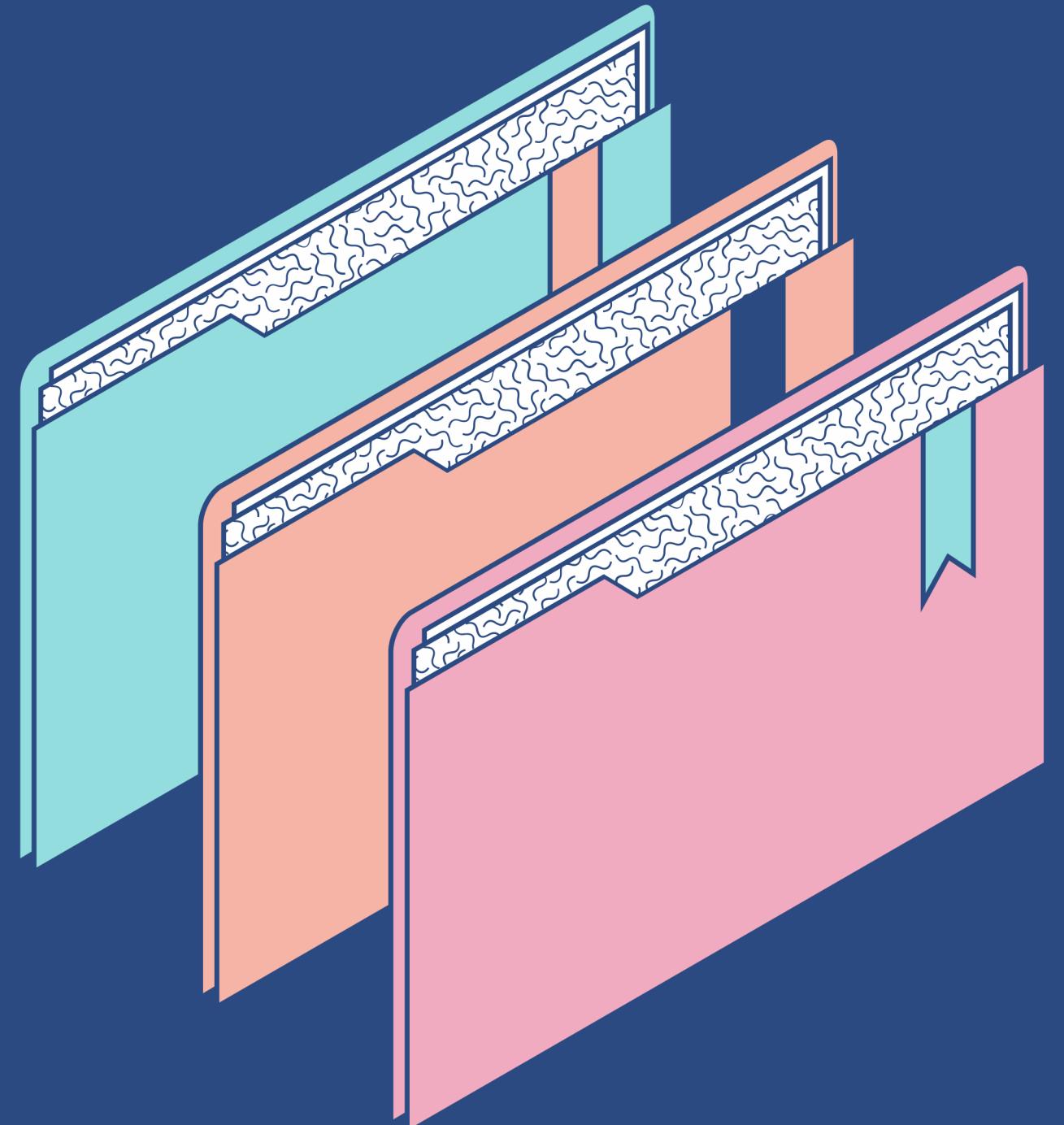




Desenvolvimento de software orientado a aspectos

Equipe: Thomas Henrique Carvalho Pinheiro,
Eric Lenin Lucena de Sousa Cruz, Gabriel da
Silva Vasconcelos, Tallys Cordeiro Prado,
Rodrigo Fernandes Lima





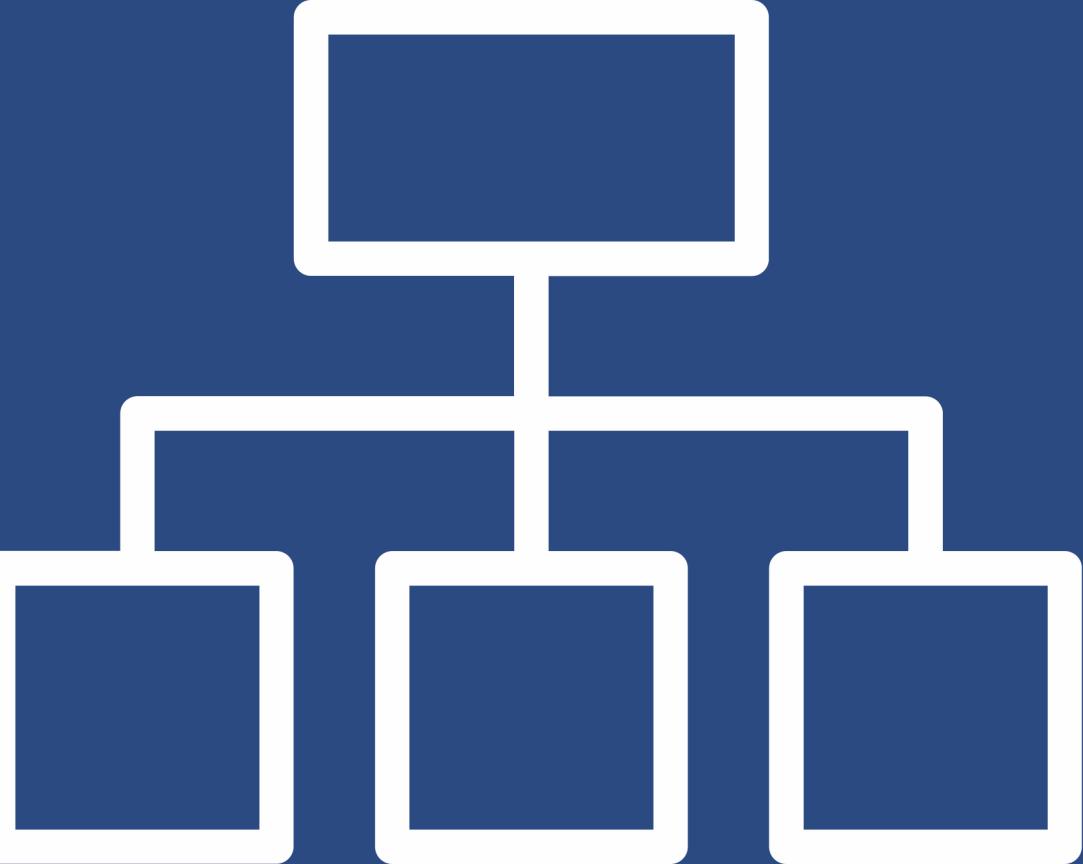
Tópicos

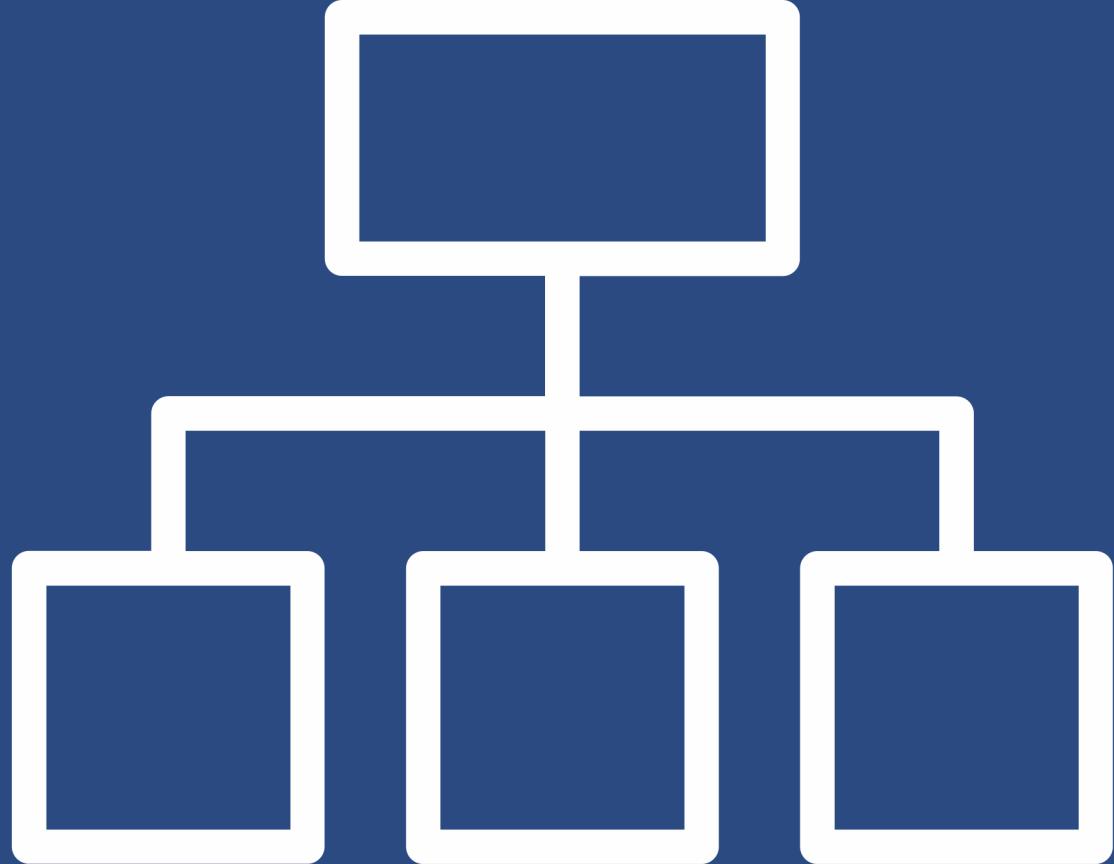
PRINCIPAIS TÓPICOS DISCUTIDOS
NESTA APRESENTAÇÃO

- Definição
- Padrões arquiteturais
- Abordagens de apoio ao Design Arquitetural
- Desafios na Arquitetura de software

• Definição e objetivo

- O objetivo da programação orientada a aspectos é oferecer suporte para o desenvolvedor na tarefa de separar componentes entre si e os aspectos entre si, utilizando-se de mecanismos que permitam a abstração e composição destas.
- Existem vários problemas que técnicas de programação estruturada ou orientada a objetos não são capazes de resolver. A orientação a aspectos ajuda nesse quesito, ajudando a organizar o código e minimizar problemas futuros.





• Padrões Arquiteturais

- A POA é uma abordagem de programação que permite a separação de preocupações (SoC) em um sistema de software, o que significa que diferentes preocupações podem ser tratadas separadamente e em paralelo. Essa separação pode ser alcançada por meio da introdução de aspectos, que são unidades modulares de código que encapsulam comportamentos que cortam através de múltiplas classes e métodos.

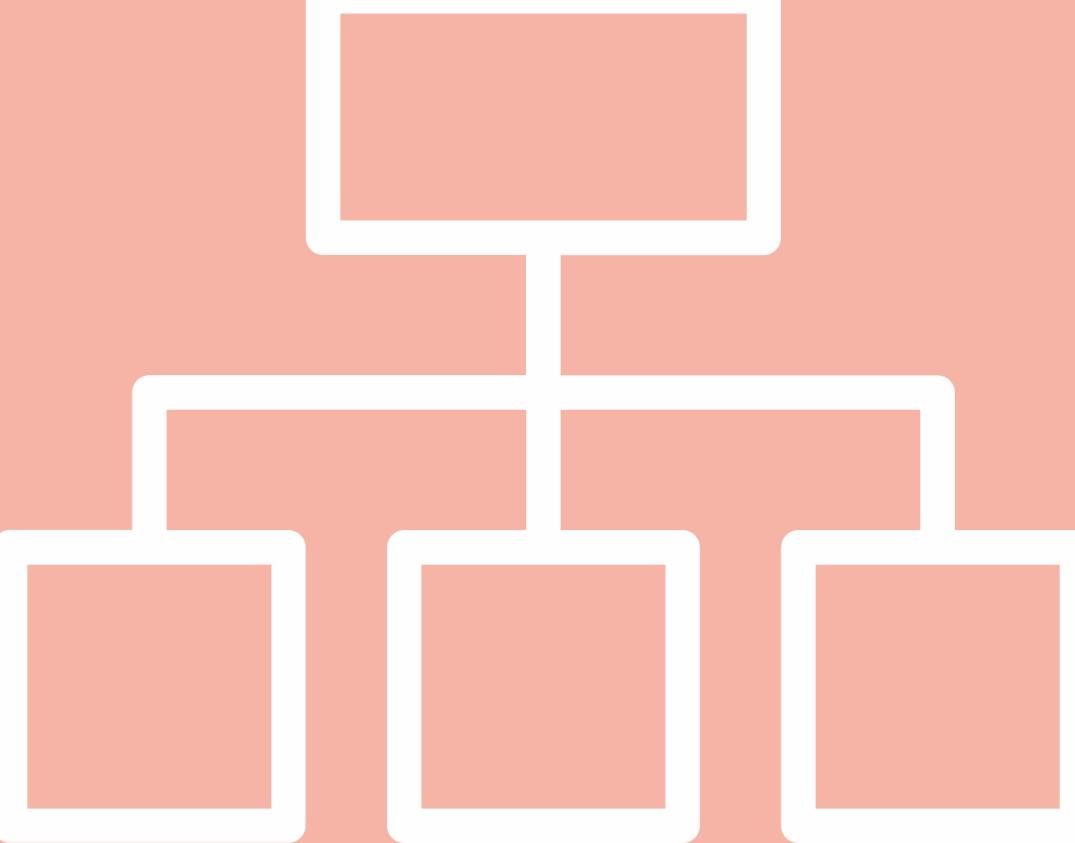
Padrões mais comuns

Padrão de Aspectos de Transações

Usado para garantir que as transações sejam gerenciadas de maneira consistente em todo o sistema, permitindo que diferentes aspectos (como registros de auditoria e validação de dados) sejam executados em diferentes estágios da transação.

Padrão de Aspectos de Segurança

Usado para garantir que as medidas de segurança apropriadas sejam aplicadas em todo o sistema, permitindo que diferentes aspectos (como autenticação e autorização) sejam aplicados a diferentes partes do sistema.



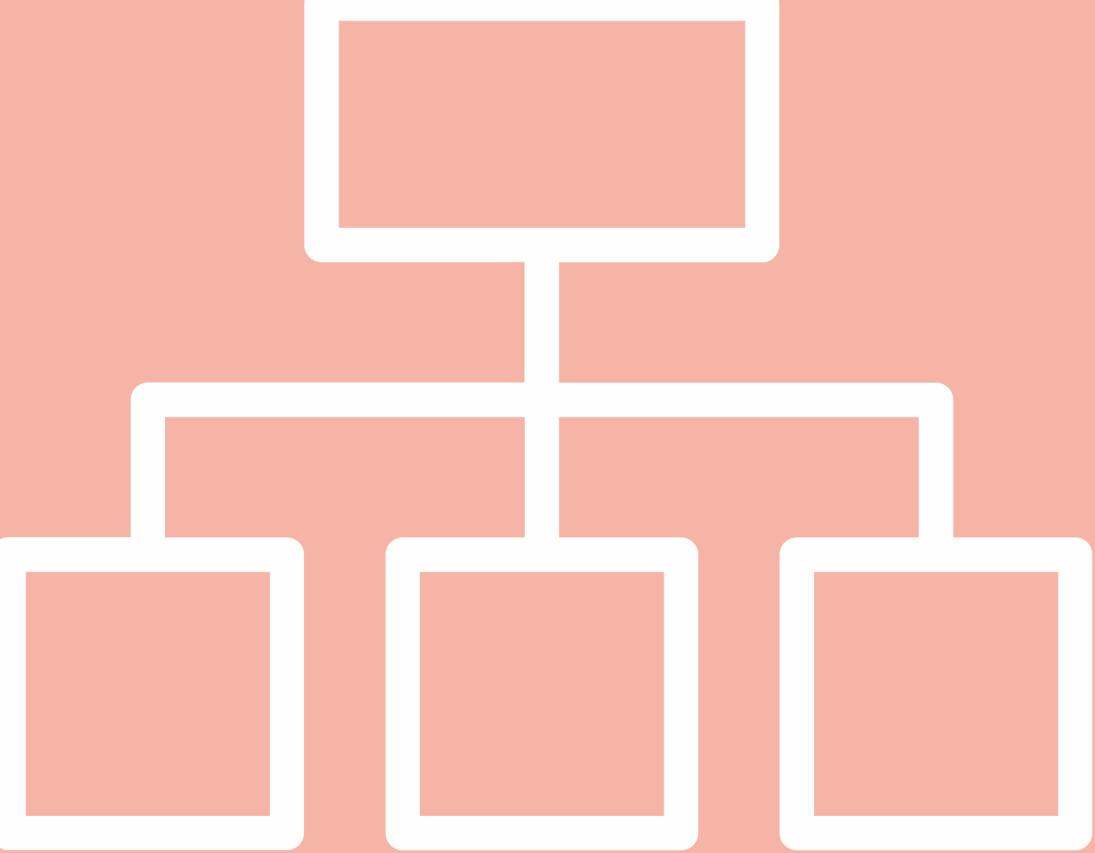
Padrões mais comuns

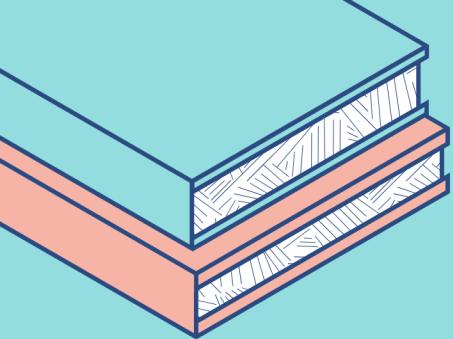
Padrão de Aspectos de Registro

Usado para coletar e registrar informações relevantes em todo o sistema, permitindo que diferentes aspectos (como rastreamento de erros e análise de desempenho) sejam aplicados a diferentes partes do sistema.

Padrão de Aspectos de Cache

Usado para melhorar o desempenho do sistema, permitindo que diferentes aspectos (como armazenamento em cache e gerenciamento de memória) sejam aplicados a diferentes partes do sistema.





Abordagens de apoio Design Arquitetural

- Métodos
- Ferramentas

Abordagens de apoio



Métodos

Os métodos de engenharia de requisitos orientados a aspectos são abordagens que buscam identificar e tratar as preocupações transversais em todas as etapas do ciclo de vida de desenvolvimento de software. Eles se baseiam na utilização de técnicas e ferramentas para facilitar a captura, a análise e a especificação dos requisitos do sistema de software, levando em conta as preocupações transversais.

Métodos

Aspect-oriented Requirements Engineering (AORE)

Objetivo principal identificar e especificar as preocupações transversais dos requisitos do sistema. Ele se baseia em uma abordagem iterativa e incremental, onde são identificados e tratados os aspectos relevantes ao longo do ciclo de vida de desenvolvimento de software.

Feature-Oriented Requirements Engineering (FORE)

Concentra na identificação e especificação de funcionalidades específicas do sistema. Ele utiliza o conceito de Feature para representar uma funcionalidade específica, considerando também as preocupações transversais ao longo do ciclo de vida.



Métodos

Goal-Oriented Requirements Engineering (GORE)

Método que se concentra na identificação e especificação dos objetivos do sistema. Ele utiliza o conceito de Goal para representar um objetivo do sistema

Problem Frames (PF)

Objetivo principal identificar e especificar os problemas do sistema. Ele se concentra na identificação de problemas específicos e utiliza o conceito de Frames para representar as soluções para esses problemas



Abordagens de apoio

Ferramentas

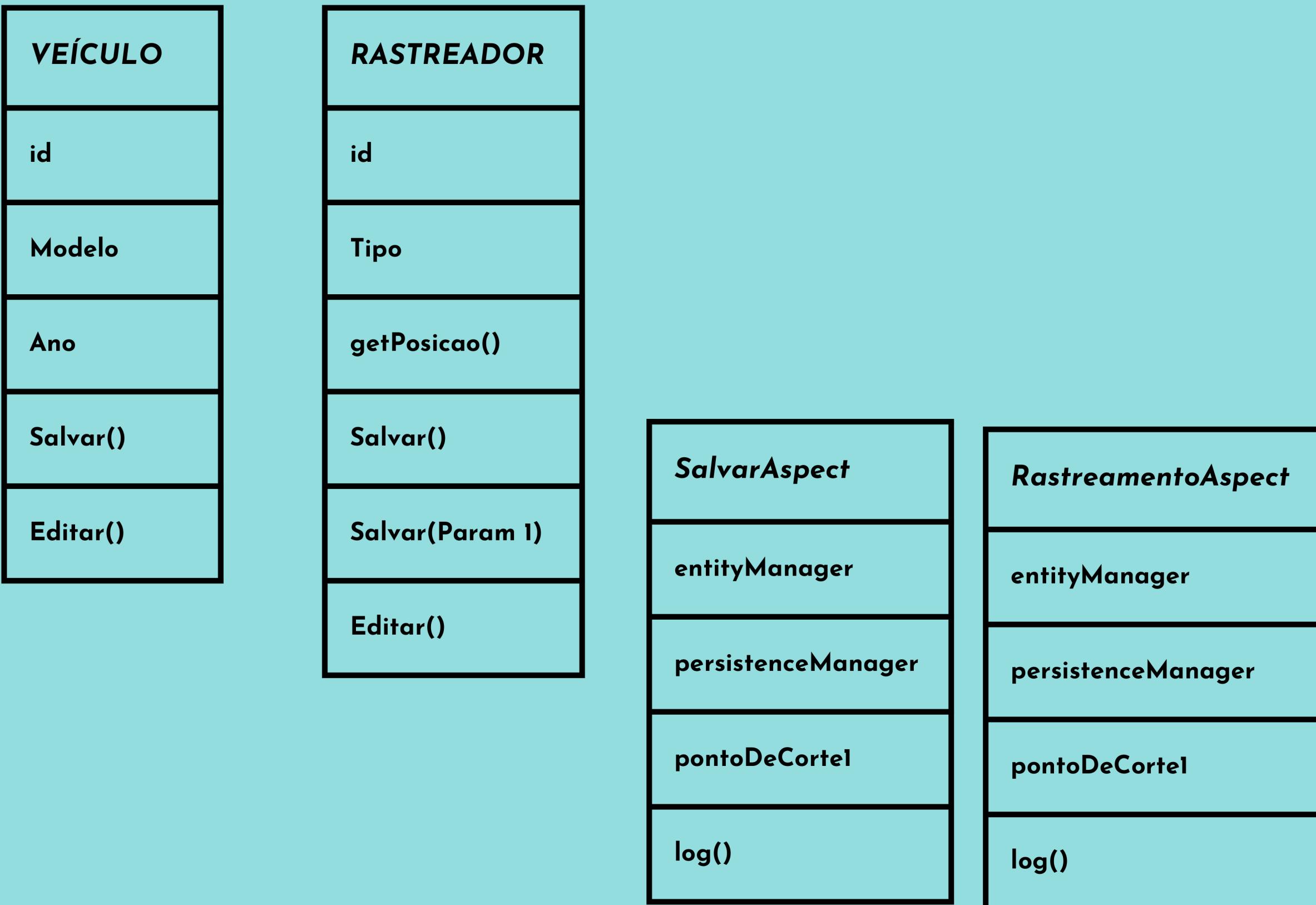
- ASPECT J



Abordagens de apoio

ASPECT J

- Aspecto
- Joinpoints
- Flags
- Advices



Veiculo.java

```
// Classe Veiculo
public class Veiculo {

    private String modeloSelecionado;

    public void salvar() {
        // Implementação da Lógica de salvamento do veículo
    }

    public void editar() {
        // Implementação da Lógica de edição do veículo
    }

}
```

Rastreador.java

```
// Classe Rastreador
public class Rastreador {

    public void getPosicao() {
        // Implementação da Lógica para obter a posição do rastreador
    }

    public void salvar() {
        // Implementação da Lógica de salvamento do rastreador
    }

    public void salvar(Rastreador rastreadorDados) {
        // Implementação da Lógica de salvamento do rastreador com o nome especificado
    }

    public void editar() {
        // Implementação da Lógica de edição do rastreador
    }
}
```

```
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;

@Aspect
public class RastreamentoAspect {

    @Pointcut("execution(public void Rastreador.salvar())")
    public void salvarPointcut() {}

    @Pointcut("execution(public void Rastreador.salvar(Rastreador)) && args(rastreadorDados)")
    public void salvarComDadosPointcut(Rastreador rastreadorDados) {}

    @Pointcut("execution(public void Veiculo.editar()) || execution(public void Rastreador.editar())")
    public void editarPointcut() {}

    @Before("salvarPointcut()")
    public void rastrearSalvamentoRastreador(JoinPoint jp) {
        System.out.println("Rastreamento: salvando rastreador...");
        String nomeMetodo = jp.getSignature().getName(); // Obtém o nome do método que está sendo executado
        System.out.println("Método: " + nomeMetodo);
    }

    @Before("salvarComDadosPointcut(rastreadorDados)")
    public void rastrearSalvamentoRastreadorComDados(Rastreador rastreadorDados) {
        System.out.println("Rastreamento: salvando rastreador com dados específicos..."+rastreadorDados);
    }

    @Before("editarPointcut()")
    public void rastrearEdicao(JoinPoint jp) {
        System.out.println("Rastreamento: editando veículo ou rastreador...");
        Object objetoDestino = jp.getTarget(); // Obtém o objeto de destino da chamada de método
        System.out.println("Objeto de destino: " + objetoDestino);
    }
}
```

RastreamentoAspect.java

```
// Define o pointcut que seleciona as chamadas ao método salvar(String nome) da classe Rastreador
@Pointcut("execution(public void Rastreador.salvar(String)) && args(nome)")
public void salvarComNomePointcut(String nome) {}

@Before("salvarPointcut()")
public void rastrearSalvamento(JoinPoint jp) {
    System.out.println("Rastreamento: salvando rastreador...");
    String nomeMetodo = jp.getSignature().getName(); // Obtém o nome do método que está sendo executado
    System.out.println("Método: " + nomeMetodo);

    Object objetoDestino = jp.getTarget(); // Obtém o objeto de destino da chamada de método
    System.out.println("Objeto de destino: " + objetoDestino);

    Object[] argumentos = jp.getArgs(); // Obtém os argumentos passados para o método
    System.out.println("Argumentos: " + Arrays.toString(argumentos));
}
```

Função em Rastreador.java

```
public void salvar(String nome) {
    // Implementação da Lógica de salvamento do rastreador com o nome especificado
}
```

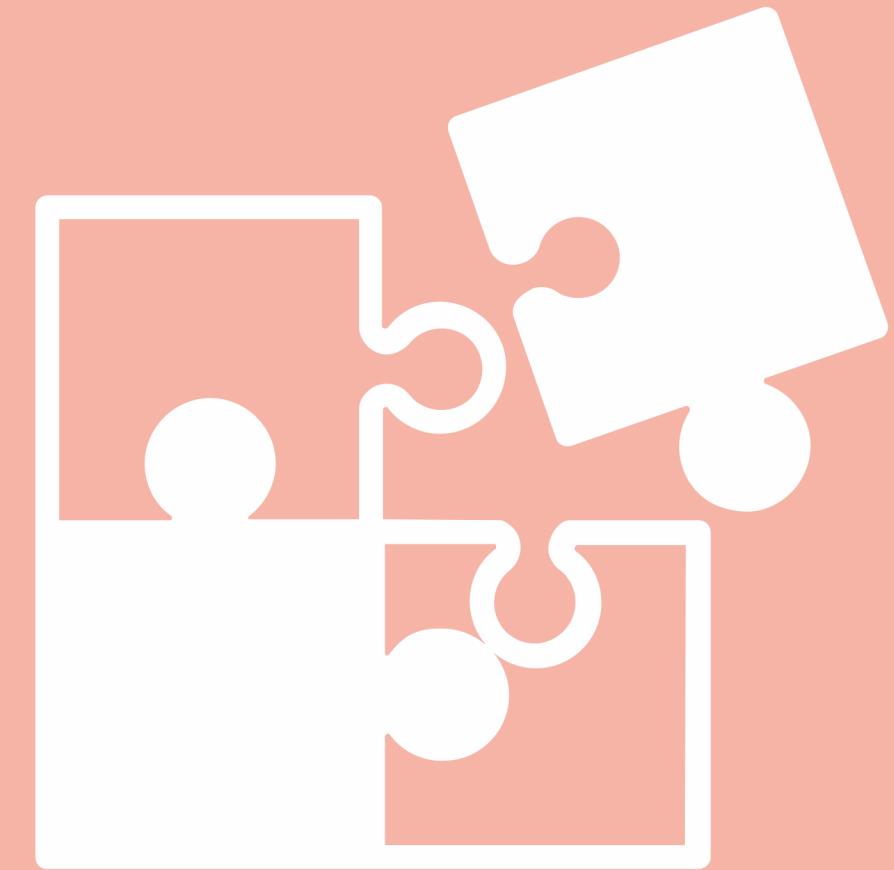
Para que serve?

Fazer auditoria em sistemas CRUD e microserviços.

Coletar métricas e dados para análise e tomada de decisão em sistemas de Big Data.

Coletar dados para análise e melhoria dos modelos de machine learning.

Desafios na arquitetura de software



- **Complexidade do projetos**

A introdução de aspectos pode aumentar a complexidade do código, o que acaba sendo difícil de entender e manter o mesmo.

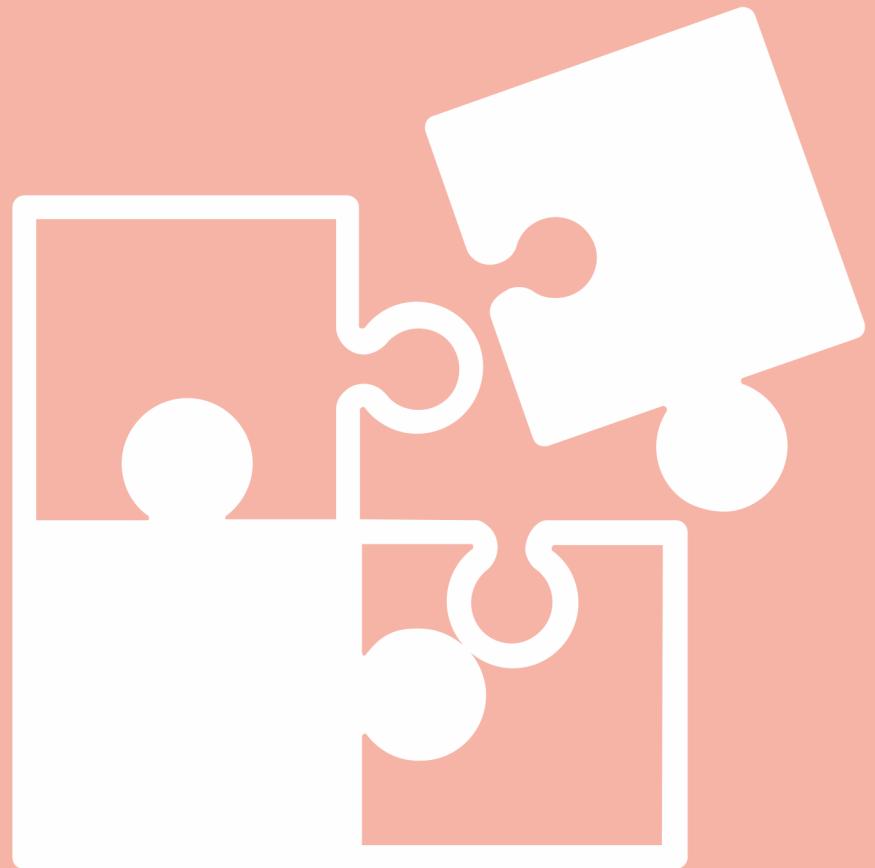
- **Compatibilidade em outras arquiteturas**

O DSOA por ser algo "novo", pode haver dificuldade em integrar sistemas existentes que não foram projetados com DSOA em mente.

- **Testes e depurações**

É necessário garantir que todos os aspectos funcionem corretamente sem atrapalhar nas funcionalidades principais do sistema. Além disso a depuração pode ser mais difícil em sistemas com muitos aspectos interconectados.

Desafios na arquitetura de software



- **Manutenção e evolução**
- **Ferramentas de desenvolvimento**

A medida que o projeto evolui, pode ser difícil manter a coesão entre os aspectos e as funcionalidades principais.

Nem todas as ferramentas de desenvolvimento suportam nativamente aspectos, o que pode tornar a implementação mais difícil.

Referências

- Albahar, MA 2015. Engenharia de Software Orientada a Aspectos. Em International Journal of Engineering, Management & Sciences (IJEMS) , vol. 2(4)
- Baniassad, E. and Clarke, S. 2004. Finding aspects in requirements with THEME/Doc. In Proc. of Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design Workshop, pp.15--22.
- CAMPOS, F. Programação Orientada a Aspectos. Juiz de Fora, 2006. Monografia (Bacharelado) - Departamento de Ciência da Computação, UFJF - MG.



Perguntas?

