

5954014 - Engenharia de Software

Aula 11 - Princípios de Projeto de Software

Prof. Dr. Denis Martins

DCM | FFCLRP | USP

Níveis de Abstração em Engenharia de Software

INTENÇÃO

Requisitos

Dizem principalmente **o que e como precisamos** construir.



PONTE / TRANSIÇÃO

Modelos

Ajudam a **transitar** entre os requisitos e o código concreto.



EXECUÇÃO

Código

Mostra **concretamente** o que e como construímos.



Projeto de Software

Projeto de software é a atividade de definir estruturas, responsabilidades e interfaces que permitam atender aos requisitos e às restrições do sistema.

Objetivos de Aprendizagem:

- Justificar a organização do software.
- Avaliar coesão, acoplamento e ocultamento de informação.
- Distribuir responsabilidades usando GRASP.

Pergunta

"Saber o que o sistema deve fazer basta para decidir como organizá-lo?"

Retomando o Sistema de Reserva de Salas

Cenário

Uma universidade oferece um sistema web para reservar salas.

O usuário informa sala, data e horário.

O sistema usa autenticação institucional, verifica disponibilidade, registra a reserva e envia confirmação.

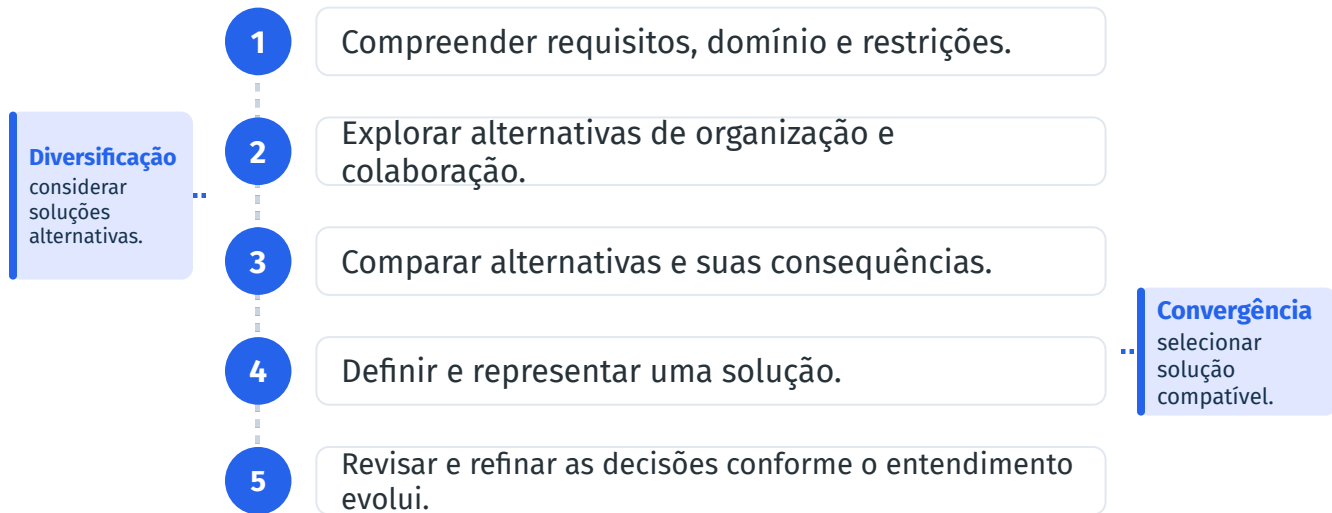
Regras do Caso

1. Início anterior ao término; reservas ativas da mesma sala não se sobrepõem.
2. Reservas consecutivas podem compartilhar o instante de término/início.
3. Autenticação realizada por serviço institucional externo.
4. Pedidos simultâneos não podem confirmar reservas conflitantes.
5. Falha no e-mail não desfaz uma reserva confirmada.

Comparação Requisito x Decisão de Projeto

Impedir reservas conflitantes → Definir quem consulta e quem garante o registro sem conflito.

Projeto Transforma Necessidades em Soluções



Exemplo Prático

A tela pode concentrar toda a operação de reserva ou delegar regras, armazenamento e comunicação a colaboradores.

"O projeto prepara cada incremento e pode ser revisto durante a construção."

Modelo de Projeto reúne Perspectivas Complementares

O modelo de projeto reúne representações coerentes da solução em diferentes níveis de detalhe.

Perspectiva	Pergunta-Chave	Exemplo no Caso
Dados e classes	Que informações e responsabilidades existem?	Sala, Reserva, estado e intervalo
Arquitetura	Como as partes se organizam?	Apresentação, aplicação/domínio, infraestrutura
Interfaces	Como as partes interagem?	Contratos de reserva e autenticação
Componentes	Como cada parte realiza sua tarefa?	Validar intervalo e registrar sem conflito

Rastreabilidade:

Regra de não sobreposição



Responsabilidade definida



Operação representada

Nota: O ambiente de execução também pode ser representado por uma visão de implantação.

Abstração e Refinamento

Abstração: representar características relevantes e omitir detalhes que não interessam ao nível atual.

Refinamento: acrescentar detalhes progressivamente até orientar a implementação.

Níveis de Representação (Escada de Refinamento)

1. Domínio

Solicitar reserva de uma sala.

2. Responsabilidades

Identificar usuário; validar pedido;
registrar; comunicar resultado.

3. Operações

`solicitarReserva(...)`
`validarIntervalo()`
`registrarSeDisponivel(...)`

Abstração de Dados

Reserva representa sala, usuário, intervalo e estado.

Abstração Procedural

validarIntervalo() expressa uma tarefa sem revelar todos os passos detalhados de execução.

Arquitetura: Organização e Colaboração

Arquitetura de software descreve a organização dos principais elementos do sistema, suas responsabilidades e relações.

Exemplo de Organização em Camadas:

Apresentação

Interação com o usuário

Colaboração: A apresentação solicita uma reserva;



Aplicação / Domínio

Coordenação e regras do caso

...a aplicação usa regras de domínio...



Infraestrutura

Armazenamento e integração

...e serviços de infraestrutura.

Arquitetura: Organização e Colaboração

Arquitetura de software descreve a organização dos principais elementos do sistema, suas responsabilidades e relações.

Exemplo de Organização em Camadas:

Apresentação

Interação com o usuário

Colaboração: A apresentação solicita uma reserva;



Aplicação / Domínio

Coordenação e regras do caso

...a aplicação usa regras de domínio...



Infraestrutura

Armazenamento e integração

...e serviços de infraestrutura.

Por que importa?

Uma mudança visual pode preservar regras de reserva quando as responsabilidades estão separadas.

Integridade Conceitual

Integridade Conceitual é a coerência dos conceitos e das decisões de projeto ao longo do sistema.

Coerente	Inconsistente
Operações usam estados definidos: pendente, confirmada e cancelada	Uma operação chama a mesma situação de cancelada, outra de inativa, outra de excluída
Regras de intervalo têm a mesma interpretação nas operações	Cada operação interpreta de maneira diferente o horário de término

Efeito Prático

decisões previsíveis facilitam compreensão e evolução.

Pergunta:

Se cada módulo funciona isoladamente, o conjunto necessariamente é coerente?

Modularidade e Separação de Interesses

Modularidade: decompor o software em partes com responsabilidades e interfaces definidas.

Separação de Interesses: distinguir preocupações que podem ser compreendidas e modificadas de maneira relativamente independente.

Preocupações do Caso:

Interação

Regras de reserva

Armazenamento

Autenticação

Notificação

Comparação de Extremos:

Divisão insuficiente

Concentra complexidade e mudanças.

Divisão excessiva

Pode ampliar comunicação e integração.

Uma divisão útil depende de responsabilidades claras e relações controladas.

Ocultamento de Informação e Contratos

Ocultamento de Informação: Proteger decisões internas de um módulo, especialmente as sujeitas a mudança, oferecendo acesso por uma interface adequada.

Interface como Contrato: Operações disponíveis, entradas, resultados e condições de uso.

Exemplo Prático:

- Cliente chama `autenticar(credenciais)`.
- Componente de integração oculta protocolo e fornecedor.
- Trocar o fornecedor pode preservar a operação e sua semântica.

Visão de Arquitetura & Abstração



Exposição interna

`agenda.reservas.add(reserva)`

Cliente altera diretamente a coleção

Acesso por serviço

`agenda.solicitarReserva(pedido)`

Componente aplica as regras antes do registro

Ocultamento de Informação e Contratos

Ocultamento de Informação: Proteger decisões internas de um módulo, especialmente as sujeitas a mudança, oferecendo acesso por uma interface adequada.

Interface como Contrato: Operações disponíveis, entradas, resultados e condições de uso.

Exemplo Prático:

- Cliente chama `autenticar(credenciais)`.
- Componente de integração oculta protocolo e fornecedor.
- Trocar o fornecedor pode preservar a operação e sua semântica.

Visão de Arquitetura & Abstração



ALERTA DE DESIGN: Um atributo privado pode continuar exposto por um getter que devolve uma coleção mutável.
Pergunta para Reflexão: O que deve continuar estável para os clientes?

Coesão, Acoplamento e Independência Funcional

Conceito	Definição	Pergunta Chave
Coesão	Relação entre responsabilidades de um elemento.	<i>As tarefas pertencem a um propósito focado?</i>
Acoplamento	Dependência entre elementos.	<i>Uma mudança aqui afeta quais partes?</i>

Independência Funcional: responsabilidades focadas e interação controlada (**alta coesão + baixo acoplamento**).

Contraste: Responsabilidades Dispersas

TelaReserva valida horários, executa SQL e envia e-mail.

Contraste: Responsabilidades Relacionadas

Validador de intervalos concentra exclusivamente operações relacionadas ao horário.

Baixo acoplamento permite colaboração com dependências justificadas.

Exercício 1: Diagnóstico de Responsabilidades

Situação:

TelaReserva recebe dados, chama o fornecedor de autenticação, calcula conflitos, executa SQL, envia e-mails e monta a resposta. Se o e-mail falha, mostra 'reserva não realizada', mesmo depois de gravar a reserva.

Tarefas:

1. Identifiquem três problemas e relacionem-nos aos conceitos estudados.
2. Distribuem de 4 a 6 responsabilidades entre elementos da solução.
3. Escolham uma mudança: autenticação, armazenamento ou notificação. Expliquem seu impacto.

Produza uma tabela com responsabilidade, responsável e justificativa.

Refatoração e Correção de Comportamento

Reorganizar a estrutura interna de um projeto ou código **preservando seu comportamento externo**.

Responsabilidades de TelaReserva

ANTES

TelaReserva (UI + Lógica de Envio de E-mail)



DEPOIS (Extração)

TelaReserva
(Apenas UI)

ServicoEmail
(Colaborador)



Alteração de Código

Extrair o envio de e-mail para um colaborador mantendo resultados observáveis

Trocar 'reserva não realizada' por mensagem coerente com reserva confirmada

Classificação

Refatoração

Correção de comportamento

Pergunta: Qual detalhe sua proposta passou a ocultar?

Classe de Análise x Classe de Projeto

Classe de Análise

Representa conceito do domínio em nível abstrato.

Exemplo:

- **Reserva** (usuário, sala, período)

Refinamento
→

Classe de Projeto

Especifica atributos, operações e colaborações para implementação.

Exemplo:

- **Reserva** (usuariold, salald, inicio, termino, estado; **validarIntervalo()**)

Elemento Técnico

RepositorioReservas

Oferece operações de acesso e registro aos dados persistentes do sistema.

CrITÉrios de Qualidade

- **Serviços necessários:** atendimento aos requisitos.
- **Responsabilidade focada:** alta coesão.
- **Dependências justificadas:** baixo acoplamento.

Perguntas Importantes

1 Que responsabilidade está sendo atribuída?

2 Que informação precisa ficar interna?

3 De quais elementos a solução depende?

4 Que mudança pode exigir sua revisão?

**Abstração seleciona detalhes; refinamento desenvolve;
modularidade organiza; coesão e acoplamento orientam a avaliação.**

Pergunta para Discussão:

Que informação sua solução oculta e por que isso facilita uma mudança?

Atribuição de Responsabilidades

Temos responsabilidades identificadas. Agora precisamos justificar sua distribuição.

Responsabilidade	Pergunta de Decisão (Quem?)
Validar o intervalo	Quem possui os dados?
Criar Reserva	Quem reúne os dados iniciais?
Coordenar a solicitação	Quem recebe a operação do sistema?
Consultar conflitos	Quem acessa o conjunto de reservas?

Desafio

Uma solução que funciona também precisa ser compreensível e modificável.

Padrões: problema, contexto, solução e consequências

Definição de Padrão: Um padrão descreve uma solução recorrente para um problema de projeto em determinado contexto, incluindo suas consequências.

Descrição Mínima: problema, contexto, solução, consequências.

Escala de Padrão	Foco / Escopo	Exemplo
Arquitetural	Organização geral	Ex.: Camadas
Projeto	Responsabilidades e colaboração	Ex.: GRASP
Idioma de código	Implementação em linguagem específica	Ex.: Gerenciador de contexto em Python

Padrões: problema, contexto, solução e consequências

Definição de Padrão: Um padrão descreve uma solução recorrente para um problema de projeto em determinado contexto, incluindo suas consequências.

Descrição Mínima: problema, contexto, solução, consequências.

Escala de Padrão	Foco / Escopo	Exemplo
Arquitetural	Organização geral	Ex.: Camadas
Projeto	Responsabilidades e colaboração	Ex.: GRASP
Idioma de código	Implementação em linguagem específica	Ex.: Gerenciador de contexto em Python

Framework

Conceito: Infraestrutura reutilizável.

Fornecer uma estrutura genérica e extensível que dita a arquitetura padrão para aplicações de um determinado domínio.

GRASP

General Responsibility Assignment Software Patterns.

Conjunto de princípios diretivos para atribuir responsabilidades a objetos e classes em projeto orientado a objetos.

GRASP Especialista: responsabilidade e informação

Problema: Quem deve assumir uma responsabilidade?

Solução: Atribuí-la a quem possui as informações necessárias para cumpri-la.

Venda

`obterTotal()`

Conhece os itens da venda e soma os subtotais calculados.

ItemLinhaVenda

`obterSubtotal()`

Conhece a quantidade do item e calcula o subtotal.

EspecificacaoProduto

`obterPreco()`

Fornece o preço unitário do produto.

GRASP Especialista: responsabilidade e informação

Problema: Quem deve assumir uma responsabilidade?

Solução: Atribuí-la a quem possui as informações necessárias para cumpri-la.

Venda

`obterTotal()`

Conhece os itens da venda e soma os subtotais calculados.

ItemLinhaVenda

`obterSubtotal()`

Conhece a quantidade do item e calcula o subtotal.

EspecificacaoProduto

`obterPreco()`

Fornece o preço unitário do produto.

Cálculo: $2 \text{ un} \times \text{R\$ } 30,00 + 1 \text{ un} \times \text{R\$ } 20,00 = \text{R\$ } 80,00$

GRASP Especialista: responsabilidade e informação

Problema: Quem deve assumir uma responsabilidade?

Solução: Atribuí-la a quem possui as informações necessárias para cumpri-la.

Venda

`obterTotal()`

Conhece os itens da venda e soma os subtotais calculados.

ItemLinhaVenda

`obterSubtotal()`

Conhece a quantidade do item e calcula o subtotal.

EspecificacaoProduto

`obterPreco()`

Fornece o preço unitário do produto.

Cálculo: $2 \text{ un} \times \text{R\$ } 30,00 + 1 \text{ un} \times \text{R\$ } 20,00 = \text{R\$ } 80,00$

Aplicação ao Caso de Reservas:

Reserva conhece seu início e término; consultar conflitos requer informação sobre outras reservas.

GRASP Criador: quem instancia o objeto?

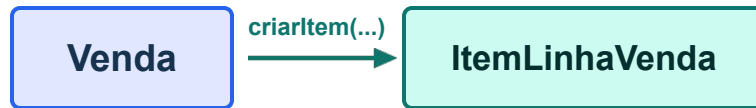
Problema: Quem deve criar uma instância de A?

Solução: Escolher quem já tem relação com o objeto ou reúne seus dados iniciais.

Candidato B (deve criar A se):

- Contém ou agrega **A**
- Registra **A**
- Usa **A** de perto
- Possui os dados de inicialização de **A**

Exemplo do Padrão



Venda contém itens e possui os dados de contexto necessários, tornando-se a candidata ideal a criar **ItemLinhaVenda**.

GRASP Criador: quem instancia o objeto?

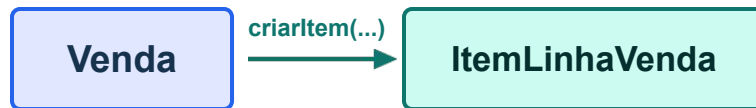
Problema: Quem deve criar uma instância de A?

Solução: Escolher quem já tem relação com o objeto ou reúne seus dados iniciais.

Candidato B (deve criar A se):

- Contém ou agrega **A**
- Registra **A**
- Usa **A** de perto
- Possui os dados de inicialização de **A**

Exemplo do Padrão



Venda contém itens e possui os dados de contexto necessários, tornando-se a candidata ideal a criar **ItemLinhaVenda**.

Aplicação ao Caso de Reservas:

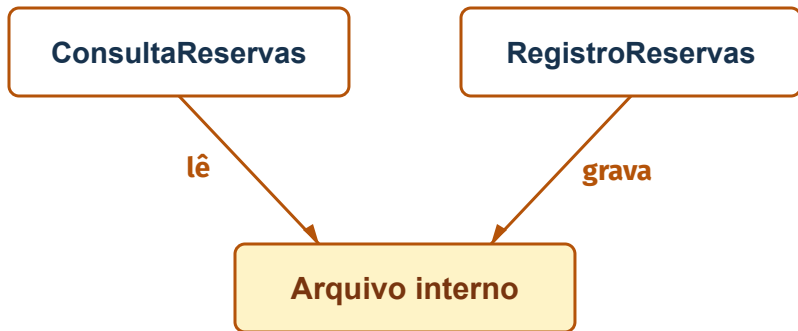
Um serviço que reúne usuário, sala e intervalo pode ser candidato a criar **Reserva**.

Pergunta: Essa criação aproveita uma relação necessária ou acrescenta uma dependência evitável?

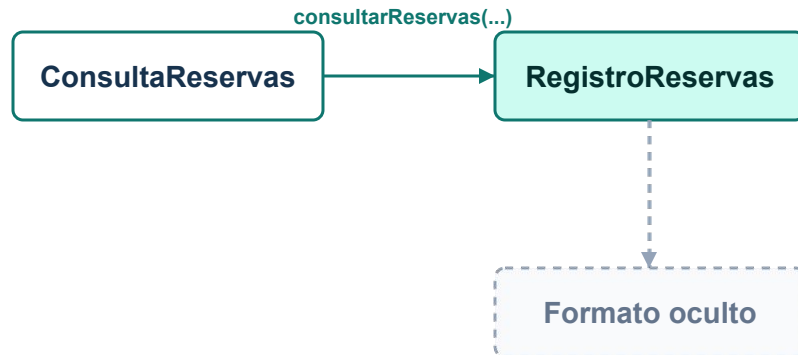
Baixo Acoplamento: dependências explícitas e escondidas

Baixo Acoplamento: dependências reduzidas e justificadas.

Dependência de detalhe interno (Oculto)



Dependência por contrato (Explícita)



Risco de Acoplamento Indireto:

Uma mudança no arquivo pode afetar a consulta mesmo sem uma chamada direta.

Alta Coesão: responsabilidades relacionadas

Baixo Acoplamento: Distribuir responsabilidades para manter os elementos focados em tarefas fortemente relacionadas.

Comparação de Abordagens (Exemplo TPV):

Coordenação focada (Alta Coesão)

TPV recebe operação e delega à Venda.

O objeto mantém escopo claro e limitado de atuação, atuando apenas como intermediador inicial.

Concentração de interesses (Baixa Coesão)

TPV acumula múltiplas responsabilidades:

- calcula regras
- cria objetos
- envia mensagens e persiste dados

Alta Coesão: responsabilidades relacionadas

Baixo Acoplamento: Distribuir responsabilidades para manter os elementos focados em tarefas fortemente relacionadas.

Comparação de Abordagens (Exemplo TPV):

Coordenação focada (Alta Coesão)

TPV recebe operação e delega à Venda.

O objeto mantém escopo claro e limitado de atuação, atuando apenas como intermediador inicial.

Concentração de interesses (Baixa Coesão)

TPV acumula múltiplas responsabilidades:

- calcula regras
- cria objetos
- envia mensagens e persiste dados

Aplicação a Reservas

Reserva valida seu intervalo; o envio de confirmação pertence a uma preocupação distinta.

Pergunta: Adicionar outra forma de notificação deveria alterar a regra do intervalo?

Controlador: receber e coordenar operações

Problema: Quem recebe e coordena uma operação do sistema?

Solução: Atribuir a responsabilidade a um objeto fora da interface com usuário, capaz de coordenar e delegar a execução.

Fluxo de Colaboração Parcial



Alternativas de Design do Controlador

1. Controlador Fachada (Facade Controller)

Representa o sistema como um todo, um subsistema ou uma empresa (ex: **SistemaReserva**, **Loja**). Indicado para sistemas com poucas operações.

2. Controlador Por Caso de Uso (Use Case Controller)

Representa um cenário de caso de uso específico onde a operação ocorre (ex: **HandlerCriarReserva**). Evita classes sobrecarregadas ("bloated").

Controladores com responsabilidades excessivas

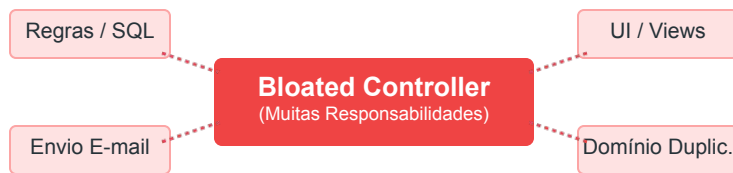
Sinais de Problema (Bloated Controller)

- Coordena processos pouco relacionados.
- Executa regras, SQL, envio de e-mail e formatação visual.
- Duplica informações do domínio.

Decisões Possíveis

- Delegar responsabilidades para classes especialistas.
- Separar coordenações distintas.
- Manter no controlador apenas o estado do caso de uso.

Arquitetura: Sobrecarregado vs. Delegação



Comparação Estratégica de Abordagens

Controlador Fachada

Serve a um conjunto **limitado** e coeso de eventos do sistema. Eficiente enquanto mantém foco.

Por Caso de Uso

Recomendado quando a fachada começa a crescer e **perder o foco**, isolando a coordenação.

Controladores com responsabilidades excessivas

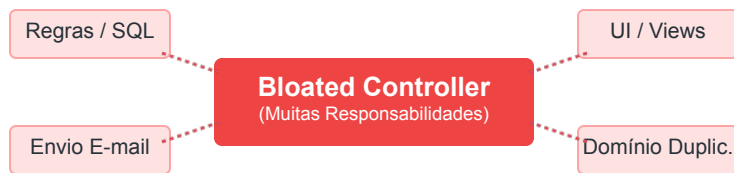
Sinais de Problema (Bloated Controller)

- Coordena processos pouco relacionados.
- Executa regras, SQL, envio de e-mail e formatação visual.
- Duplica informações do domínio.

Decisões Possíveis

- Delegar responsabilidades para classes especialistas.
- Separar coordenações distintas.
- Manter no controlador apenas o estado do caso de uso.

Arquitetura: Sobrecarregado vs. Delegação



Comparação Estratégica de Abordagens

Controlador Fachada

Serve a um conjunto **limitado** e coeso de eventos do sistema. Eficiente enquanto mantém foco.

Por Caso de Uso

Recomendado quando a fachada começa a crescer e **perder o foco**, isolando a coordenação.

A escolha depende das responsabilidades. Não exige um controlador por caso de uso.

Inversão de Dependências

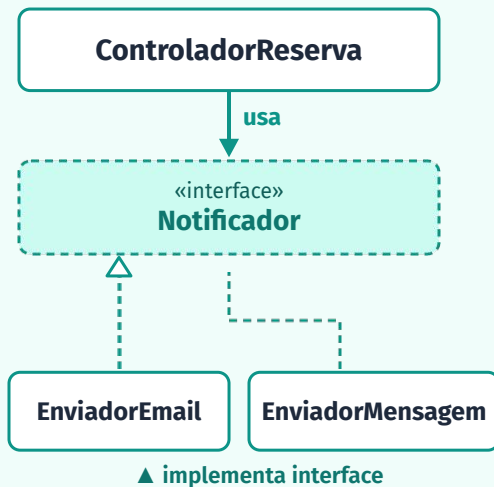
Princípio: Alto nível e detalhes devem depender de abstrações. As abstrações devem preservar independência dos detalhes.

Dependência Direta Alto Acoplamento



Problema: O módulo de alto nível está rigidamente acoplado a uma implementação concreta. Substituir a forma de notificação exige alterar o código do controlador.

Integração por Abstração Flexível (DIP)



Contrato de Operação:

`enviarConfirmacao(reserva)`

Benefício Prático:

Falha na comunicação preserva a reserva confirmada.

Permite adicionar novos canais de notificação sem alterar o controlador.

Exercício 2: Modelo de Projeto para Solicitar Reserva

Missão

Projetem a colaboração para Solicitar Reserva, atendendo às regras do caso e considerando uma mudança na autenticação, no armazenamento ou na notificação.

Entregáveis

1. Desenho de colaboração/sequência (até 7 elementos e operações nomeadas).
2. Tabela: Responsabilidade | Responsável | Justificativa GRASP.
3. Explicação de coesão e acoplamento.
4. Impacto da mudança escolhida.

Conferência de Regras: 1. Sem sobreposição simultânea 2. Autenticação externa 3. Falha de e-mail preserva reserva

Responsabilidade	Responsável	Justificativa com GRASP
[Ex: Validar disponibilidade]	[Ex: ReservaService]	[Ex: Information Expert / Controller]
[Ex: Autenticar usuário]	[Ex: AuthAdapter]	[Ex: Low Coupling / Protected Variations]
[Ex: Notificar confirmação]	[Ex: EmailService]	[Ex: Indirection / High Cohesion]

Conclusão

Um bom projeto organiza responsabilidades e controla dependências para atender aos requisitos e facilitar mudanças.

1. Coerência

Manter conceitos, regras e decisões **consistentes** em todo o projeto.

2. Modularidade

Separar responsabilidades e proteger detalhes internos sujeitos a mudança via ocultamento.

3. Coesão e Acoplamento

Manter responsabilidades relacionadas e **dependências justificadas** (alta coesão, baixo acoplamento).

4. GRASP e Abstrações

Orientar quem executa, cria e coordena, **protegendo integrações variáveis** por contratos.

Exemplo de aplicação ao Sistema de Reserva de Salas

Trocar o canal de notificação deve **preservar as regras de reserva** e o comportamento acordado no contrato (ex: falha de envio não desfaz reserva).

Pergunta para Reflexão:

Qual decisão do seu projeto facilita uma mudança? Que custo ou dependência essa decisão introduz?

Referências

Bibliografia Básica da Disciplina

Roger S. Pressman & Bruce R. Maxim

Engenharia de Software: Uma Abordagem Profissional

Capítulos 9 (completo)

Ian Sommerville

Engenharia de Software
Capítulo 7 (completo)

Referências Complementares de Apoio

Marco Tulio Valente

Engenharia de Software Moderna
Capítulo 5 (completo)

<https://engsoftmoderna.info/cap5.html>

Maps Mental do Capítulo 9 (Pressman & Maxim)

<https://coggle.it/diagram/XhnVryTykD4D4-DF/t/pressman-engenharia-de-software-cap-conceitos-de-projeto>

Dúvidas e Discussão