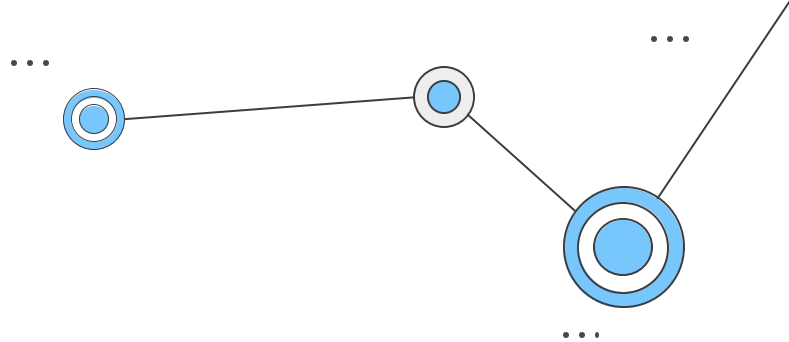


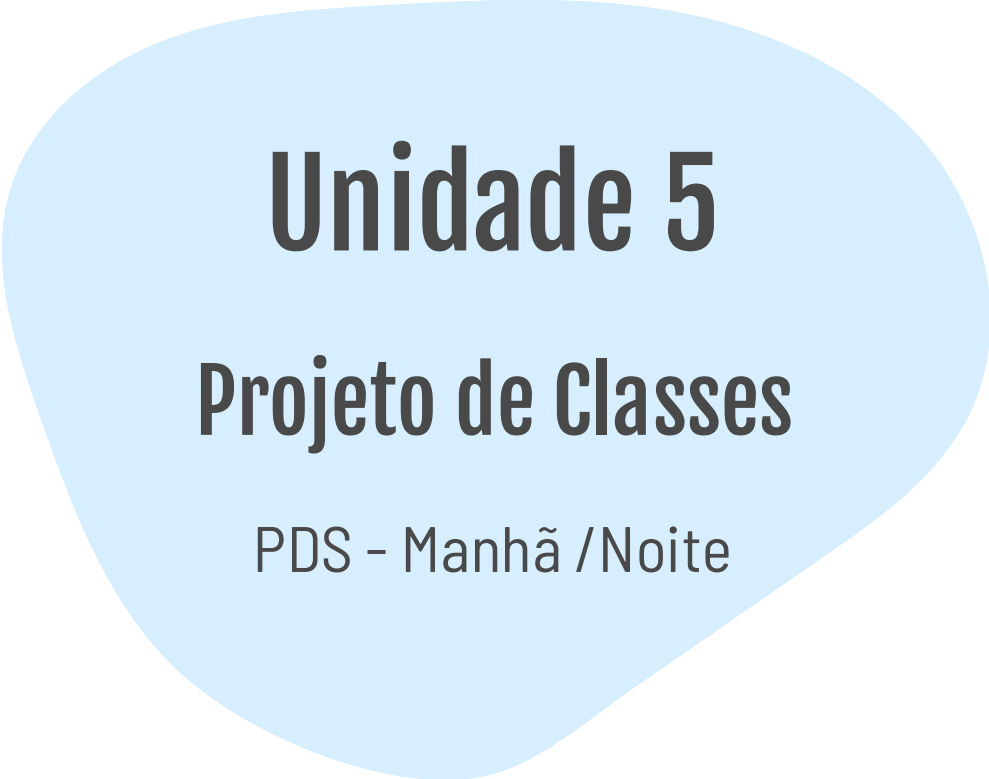
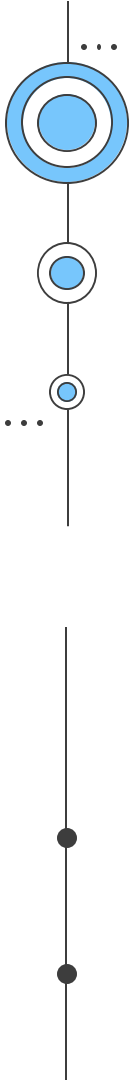


PUC Minas



Projeto de Software

Prof. Dr. João Paulo Aramuni



Unidade 5

Projeto de Classes

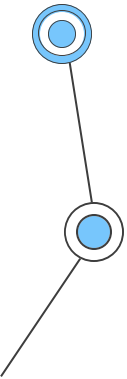
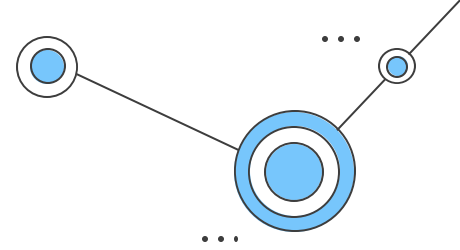
PDS - Manhã /Noite



Projeto de Classes

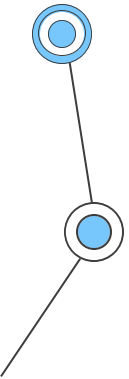
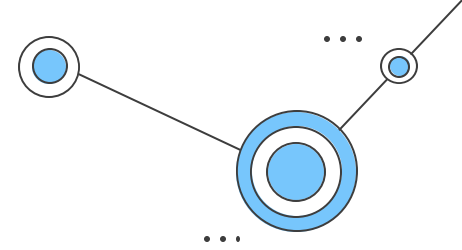
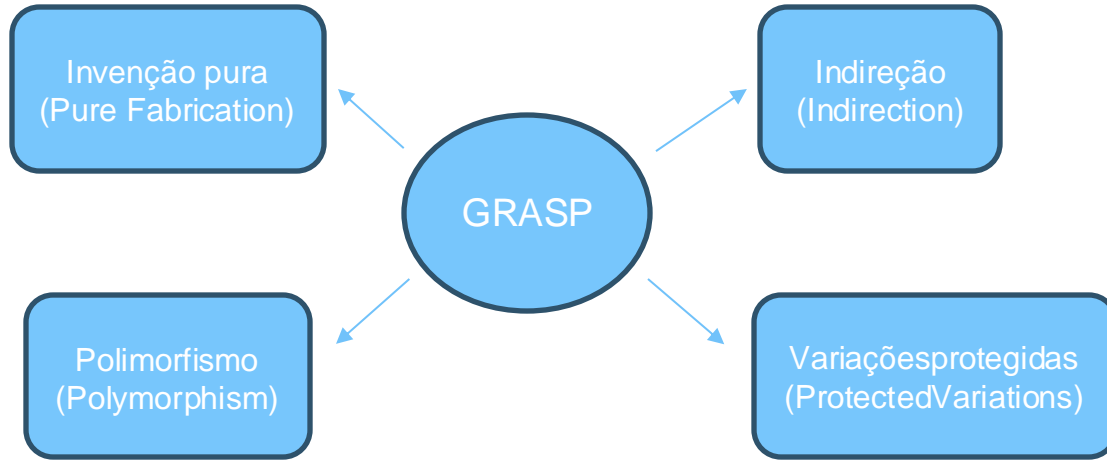
Sumário

- Padrões GRASP
 - Polimorfismo
 - Indireção
 - Invenção pura
 - Variações Protegidas

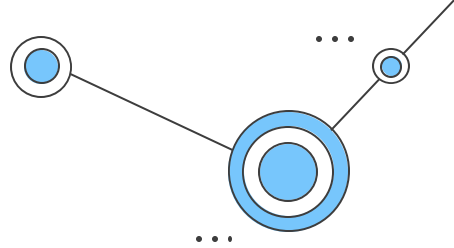


Projeto de Classes

Padrões GRASP Avançados

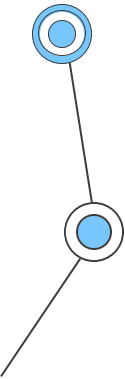


Projeto de Classes

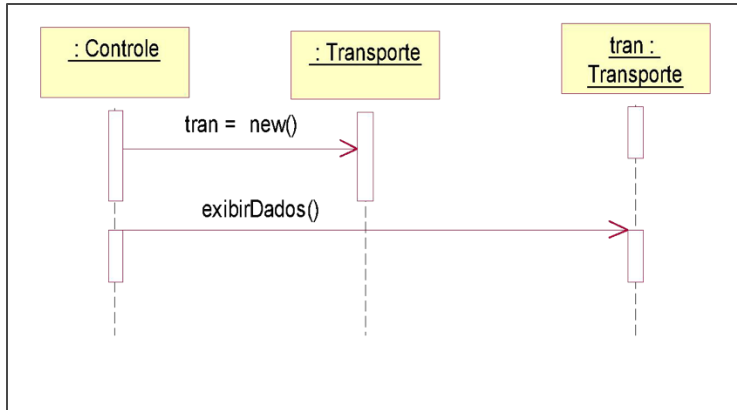
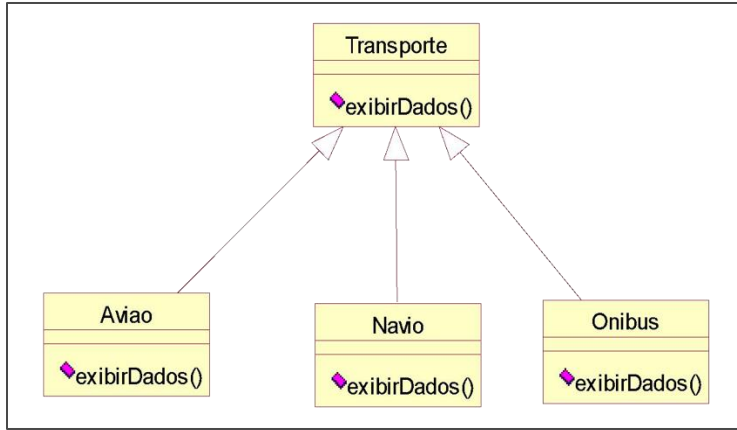


Padrões GRASP: Polimorfismo

- Problema:
 - Como tratar alternativas com base no tipo?
 - Como criar componentes de software interligáveis (plugáveis)?
 - Deseja-se evitar variação condicional (if-then-else): pouco extensível.
 - Deseja-se substituir um componente por outro sem afetar o cliente.
- Solução:
 - Quando alternativas ou comportamentos relacionados variam segundo o tipo (classe), deve-se atribuir a responsabilidade pelo comportamento (usando operações polimórficas) aos tipos para os quais o comportamento varia.
 - Corolário: não teste o tipo de um objeto e use lógica condicional para efetuar diferentes alternativas com base no tipo.



Padrões GRASP: Polimorfismo – Exemplos



Modelagem da chamada do método exibir dados

```
public static void main(String[] args) {
```

```
    Transporte tran[] = new Transporte[3];
```

```
    tran[0] = new Navio();
```

```
    // Objetos das subclasses
```

```
    tran[1] = new Aviao();
```

```
    // podem pertencer
```

```
    tran[2] = new Onibus();
```

```
    // ao tipo da Superclasse.
```

```
    System.out.println("exemplo polimorfismo .");
```

```
    System.out.println("nomes dos alunos: xxxxx.");
```

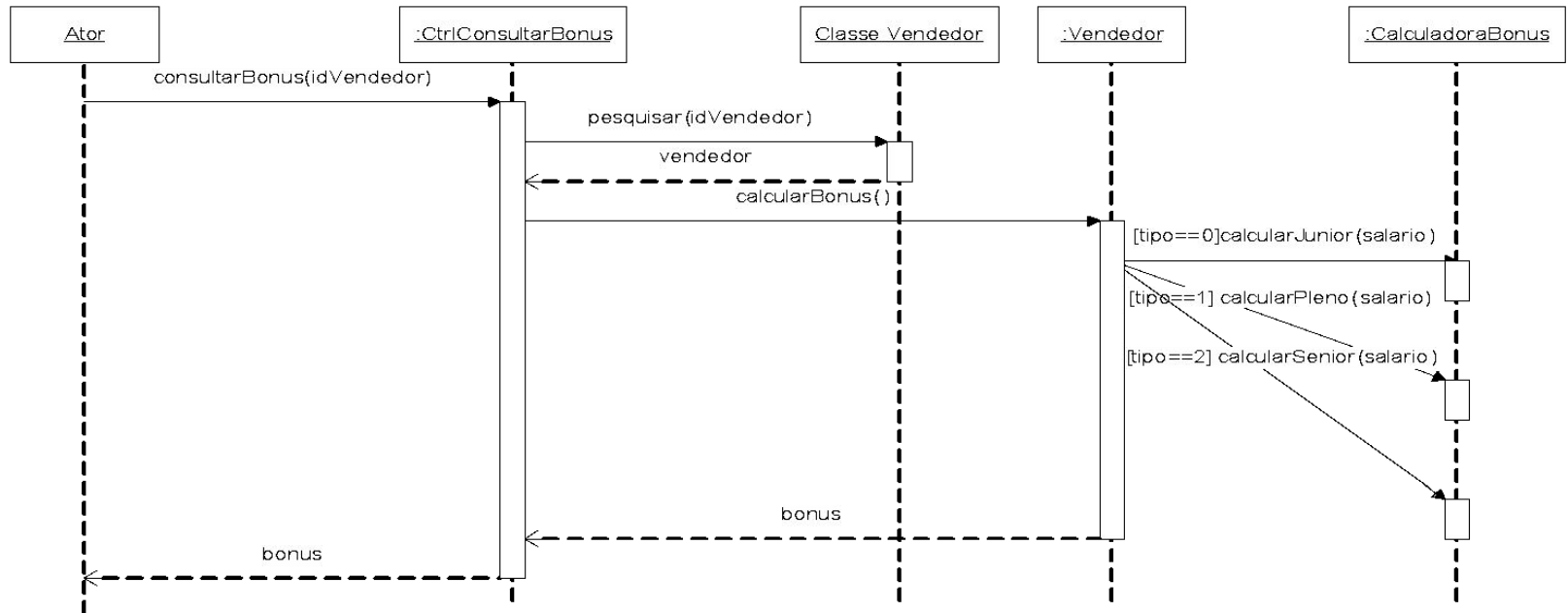
```
    for (int i = 0; i < tran.length; i++)
```

```
        tran[i].exibeDados();
```

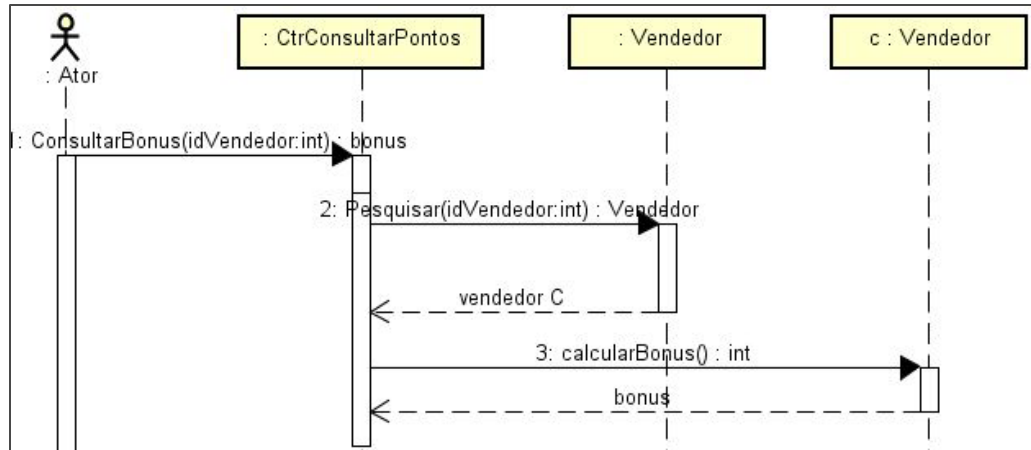
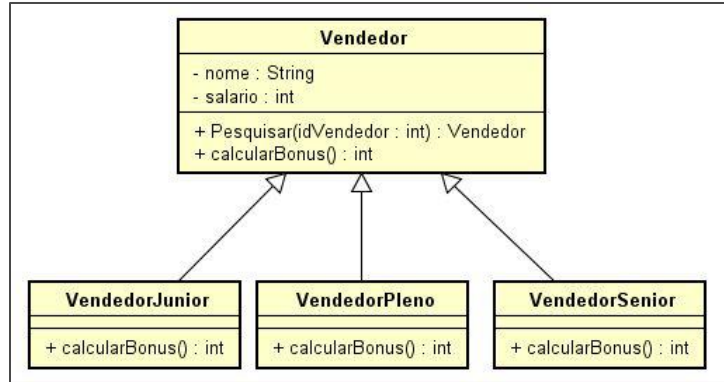
```
    // Chamada polimórfica do método.
```

Exemplo – aplicação

Dado o diagrama de sequência abaixo para um cenário do caso de uso “Consultar Bônus de Vendedor”, apresente sugestões de melhoria no desenho da solução. Desenhe o novo diagrama de sequência e o diagrama de classe de projeto proposto.



Exemplo – aplicação – solução



O método `pesquisar(idVendedor)` tipicamente realiza:

- 1) Pesquisa na tabela **VENDEDOR** do banco de dados o registro correspondente à chave primária `idVendedor`.
- 2) Verifica nos dados do registro pesquisado, na coluna **TIPO_VENDEDOR**, o tipo do vendedor.
- 3) Instancia o objeto **Vendedor** do tipo correspondente (**VendedorJunior**, **VendedorPleno** ou **VendedorSenior**).
- 4) Preenche os atributos do objeto com as informações do registro pesquisado e retorna este objeto

Projeto de Classes

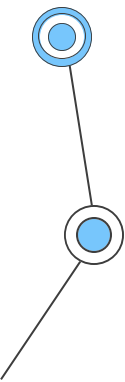
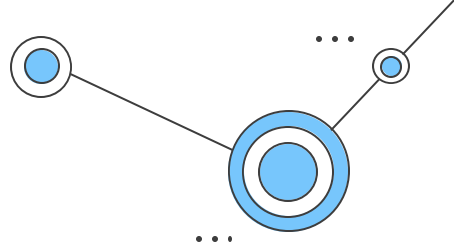
Vantagens e contraindicações

Vantagens

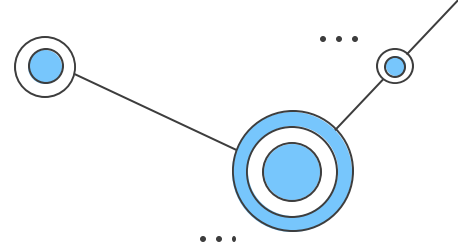
- as extensões exigidas para as novas variações são fáceis de adicionar
- novas implementações podem ser introduzidas sem afetar os clientes

Contraindicações

- não usar polimorfismo para adicionar uma flexibilidade para uma possível futura variação
 - ✓ o esforço pode não compensar

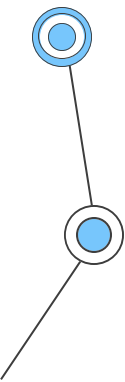


Projeto de Classes

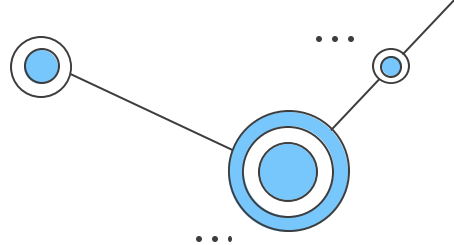


Padrão GRASP: Invenção Pura

- Problema:
 - Às vezes, durante o projeto é preciso atribuir responsabilidades que não são encaixam naturalmente em nenhuma das classes conceituais
 - Que objeto deve ter a responsabilidade quando você não quer violar "Alta Coesão" e "Baixo Acoplamento", mas as soluções oferecidas pelo "Especialista" não são apropriadas?
 - Atribuir responsabilidades apenas para classes do domínio conceitual pode levar a situações de maior acoplamento e menos coesão.
- Solução:
 - Criar uma classe artificial que não representa nenhuma entidade no domínio do problema. Uma classe fictícia que possibilite alta coesão, baixo acoplamento e o reuso.

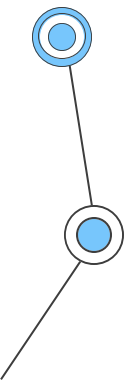


Projeto de Classes



Exemplo: problema

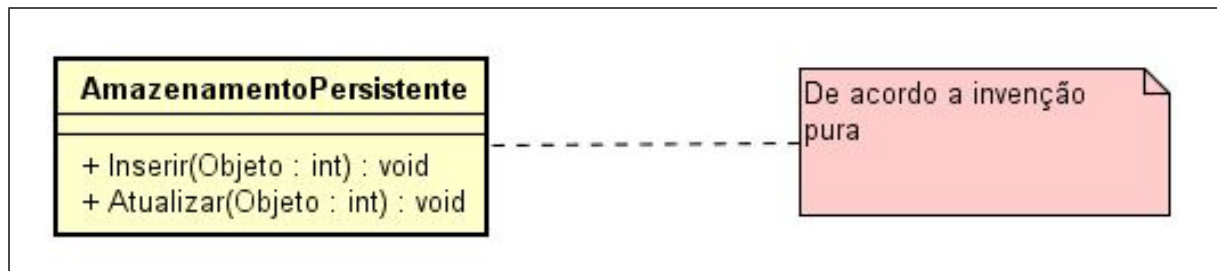
- Suporte para salvar as instâncias de Venda em um banco de dados relacional
 - Segundo o especialista, atribuir esta responsabilidade à Venda é justificável
 - ✓ Venda tem os dados que precisam ser salvos
 - Porém, a tarefa exige inúmeras operações de suporte relacionadas ao banco de dados
 - ✓ Venda pode se tornar não coesa
 - ✓ Venda precisa estar acoplada à interface do banco de dados
 - ✓ Outras classes precisam do mesmo suporte ao serviço de salvar objetos em um banco de dados



Projeto de Classes

Exemplo: solução

- Criar uma nova classe que seja responsável unicamente por salvar objetos em algum tipo de meio de armazenamento persistente



Projeto de Classes

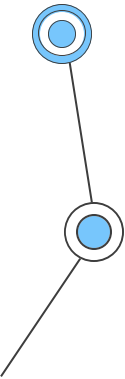
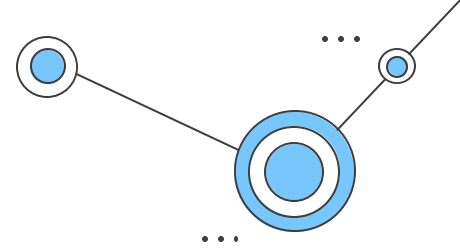
Vantagens e contraindicações

Vantagens

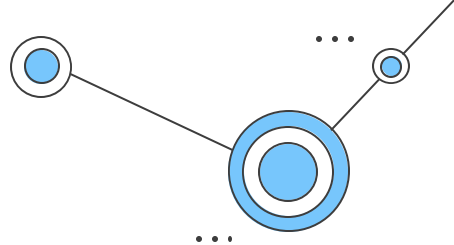
- coesão alta é favorecida.
- potencial de reutilização pode aumentar devido à presença de classes de Invenção Pura refinadas.

Contraindicações

- uso extremo – funções simples se tornam objetos.



Projeto de Classes



Padrão GRASP: Indireção

Problema:

- Onde colocar uma responsabilidade de modo a evitar o acoplamento direto entre duas ou mais classes? Como desacoplar objetos de modo a possibilitar o baixo acoplamento e manter alta a possibilidade de reuso?

Solução:

- Atribua a responsabilidade a um objeto intermediário que faça a mediação entre componentes ou serviços de modo que eles não sejam diretamente acoplados.
- Usar um objeto intermediário para ser o mediador entre componentes para que eles não sejam diretamente acoplados.

Vantagem:

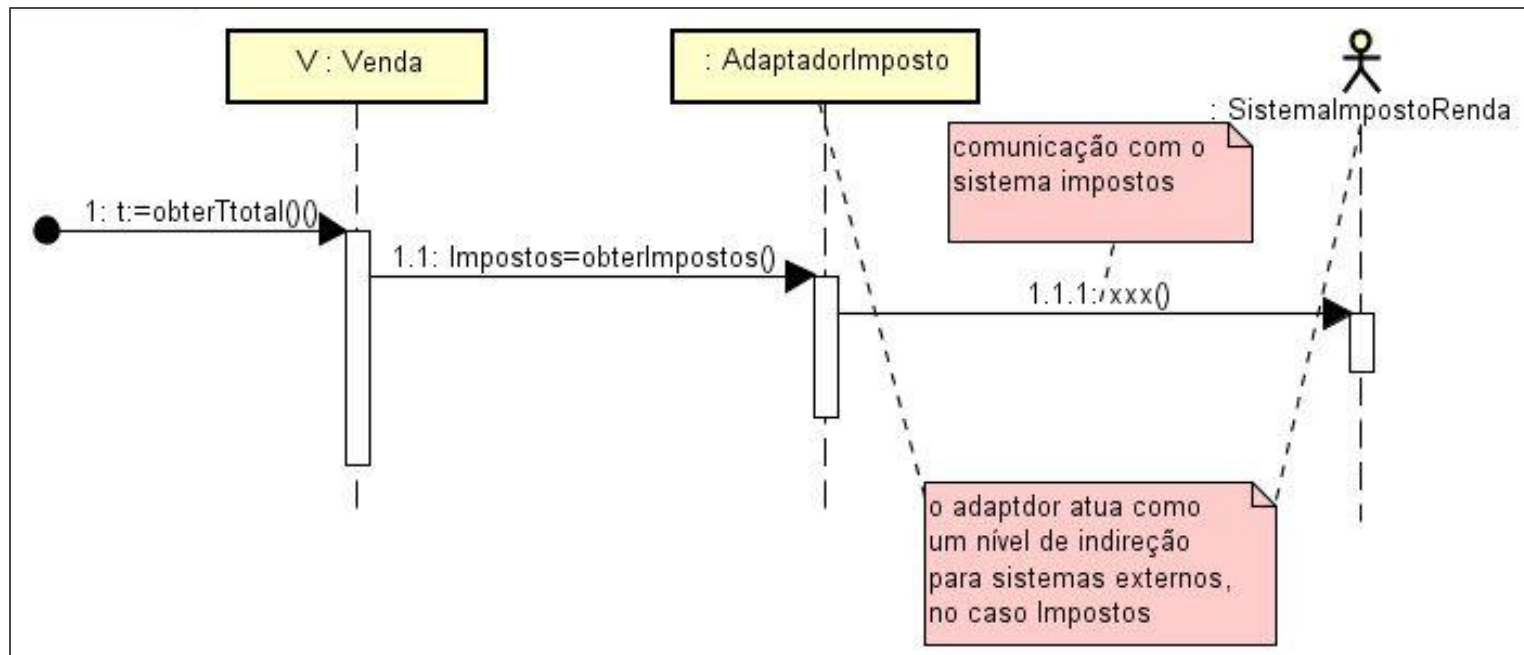
- Acoplamento mais fraco entre os componentes.



Projeto de Classes

Exemplo

- Indireção por meio do adaptador



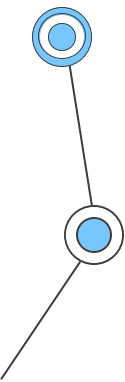
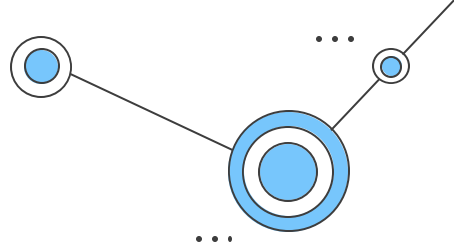
Projeto de Classes

Indireção

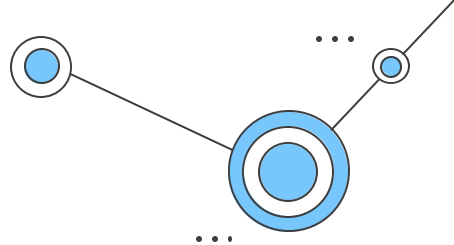
"A maior parte dos problemas em Ciência da Computação pode ser resolvida por um nível adicional de indireção"

Velho provérbio com especial relevância para sistemas orientados a objetos.

"A maior parte dos problemas de desempenho pode ser resolvida removendo-se algumas camadas de indireção"



Projeto de Classes



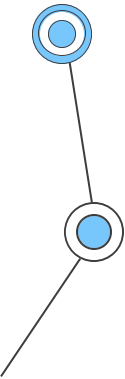
Padrão GRASP: Variações Protegidas

Problema:

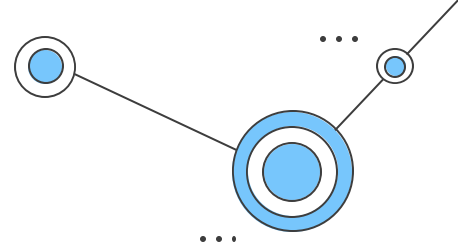
- Como projetar objetos, subsistemas e sistemas de modo que as variações ou instabilidades nesses elementos não tenham um impacto indesejável sobre outros elementos

Solução:

- Identificar pontos de variação ou instabilidade previsível; atribuir responsabilidades para criar uma interface estável em torno deles.
- Encapsulamento, interfaces, polimorfismo, indireção e padrões; máquinas virtuais e brokers são motivados por este princípio.
- Evite enviar mensagens a objetos muito distantes.



Projeto de Classes



Padrão GRASP - exemplo

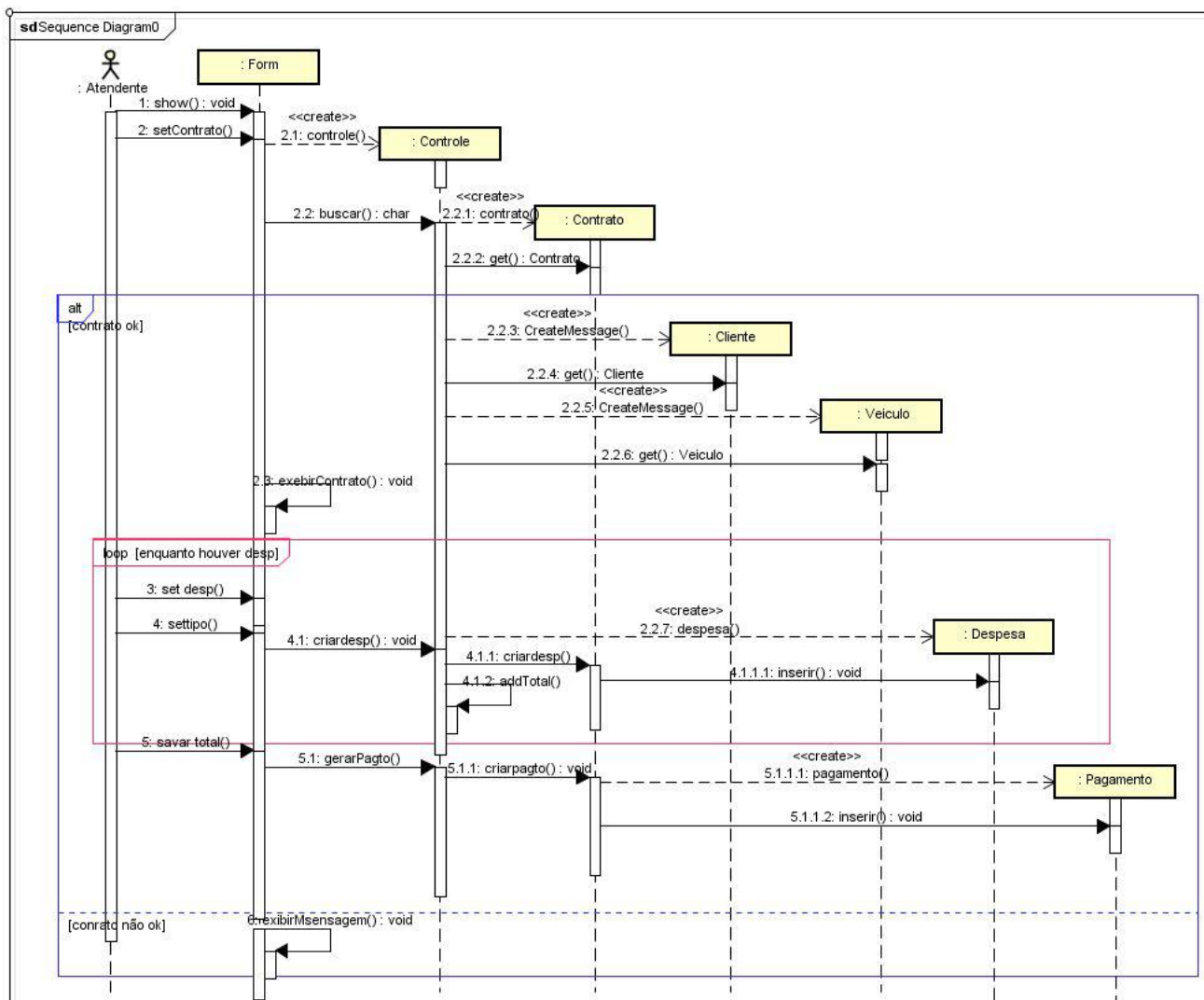
Considere o caso de uso “Efetuar pagamento locação Veículo” descrito a seguir.

- Um veículo é locado para um cliente. Um cliente pode fazer diversas locações. Um mesmo veículo pode ser alugado diversas vezes pelo mesmo cliente. As datas de início e fim da locação devem ser consideradas.
- Para o cálculo do valor da locação o atendente informa o número do contrato da locação. O sistema busca os dados do contrato (nome cliente, dados do veículo e período de locação). Se for um contrato válido, ele informa as despesas que ocorreram com o veículo. As despesas podem ser de dois tipos: quilometragem e manutenção. Para quilometragem deve ter a qte de quilômetros e para manutenção uma descrição. Ele armazena as despesas e mostra no final o valor total da locação.



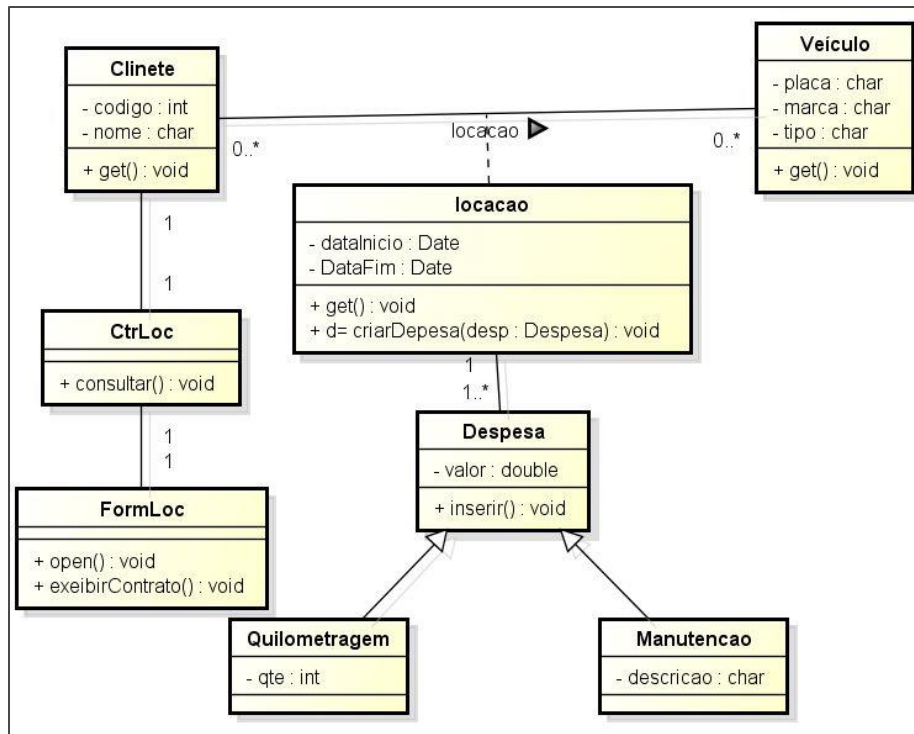
Projeto de Classes

Padrão GRASP - exemplo

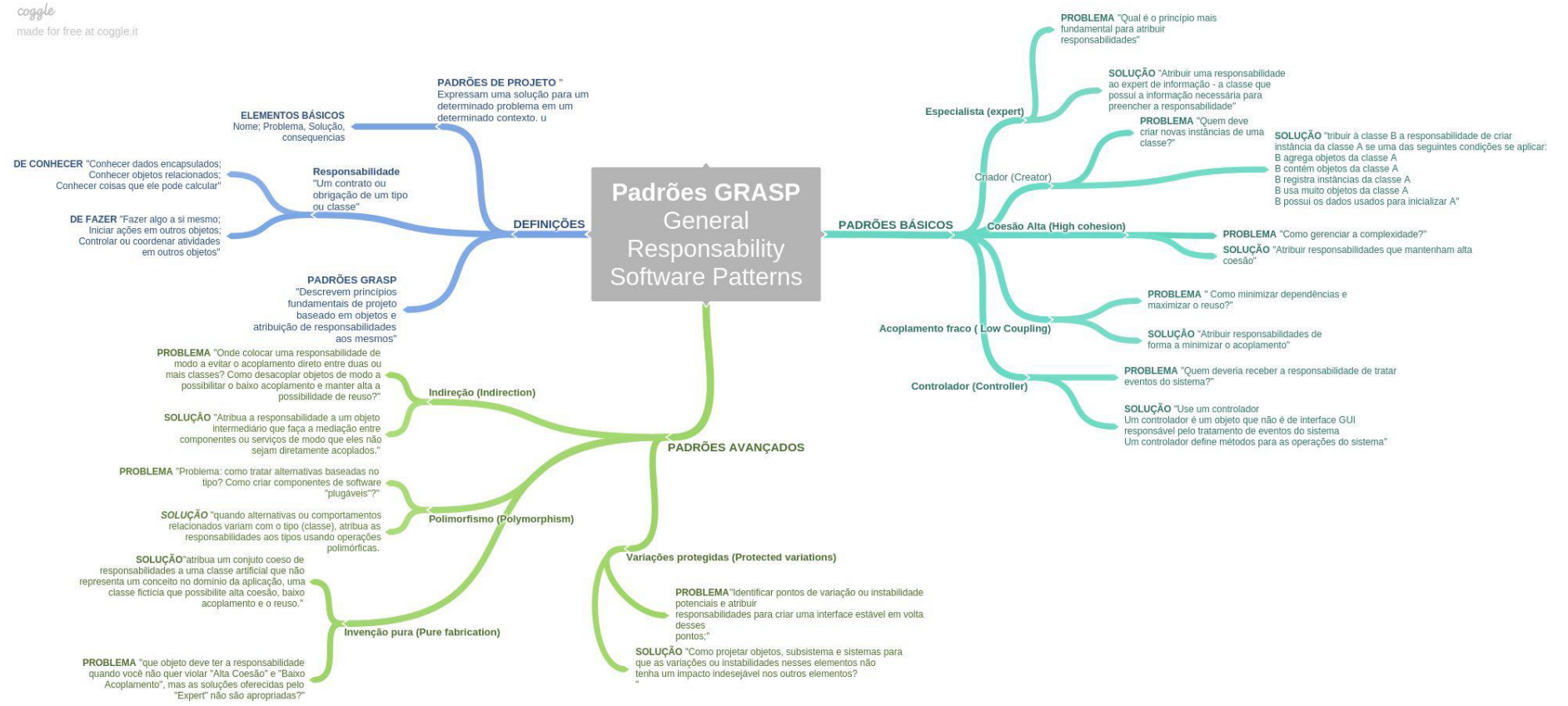


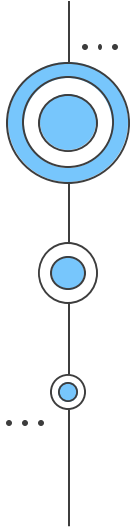
Projeto de Classes

Padrão GRASP -
exemplo



Projeto de Classes

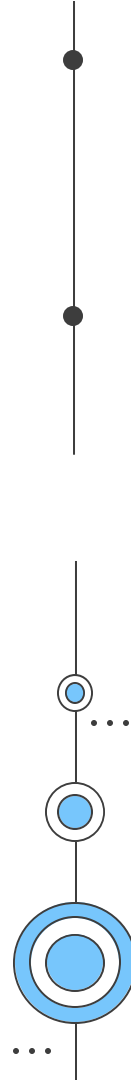


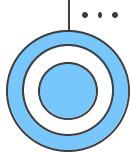


Projeto de Software

Referências básicas:

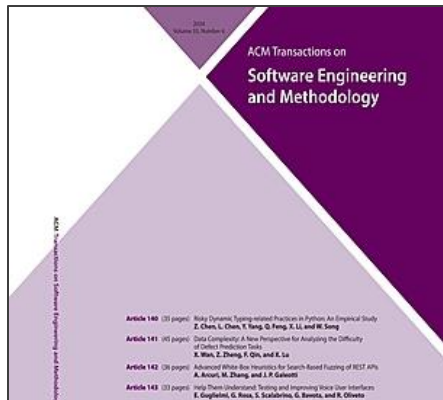
- **ACM TRANSACTIONS ON SOFTWARE ENGINEERING AND METHODOLOGY**. New York, N.Y., USA: Association for Computing Machinery, 1992-. Trimestral. ISSN 1049-331X. Disponível em: <https://dl.acm.org/toc/tosem/1992/1/2>. Acesso em: 19 jul. 2024. (Periódico On-line).
- LARMAN, Craig. **Utilizando UML e padrões**: uma introdução á análise e ao projeto orientados a objetos e desenvolvimento iterativo. 3. ed. Porto Alegre: Bookman, 2007. E-book. ISBN 9788577800476. (Livro Eletrônico).
- SILVEIRA, Paulo et al. **Introdução à arquitetura e design de software**: uma visão sobre a plataforma Java. Rio de Janeiro, RJ: Elsevier, Campus, 2012. xvi, 257 p. ISBN 9788535250299. (Disponível no Acervo).
- VERNON, Vaughn. **Implementando o Domain-Driven Design**. Rio de Janeiro, RJ: Alta Books, 2016. 628 p. ISBN 9788576089520. (Disponível no Acervo).



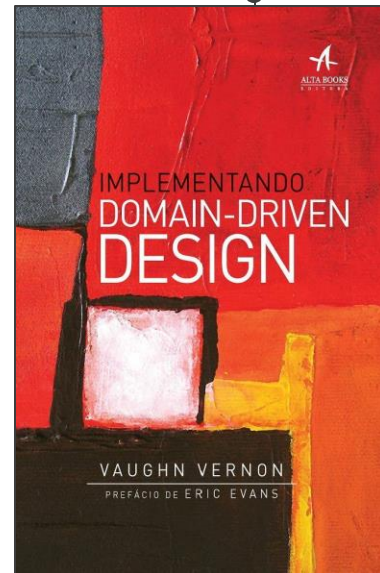
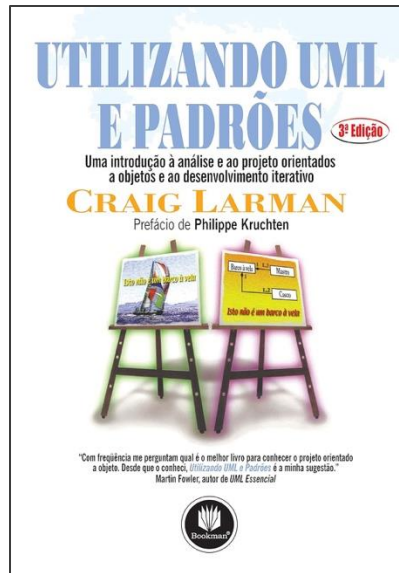


Projeto de Software

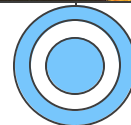
Referências básicas:

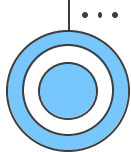


ACM Transactions on
Software Engineering
and Methodology



...



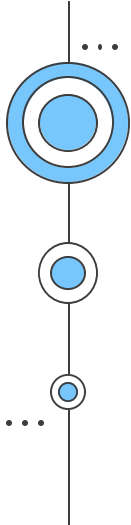


Projeto de Software

Referências complementares:

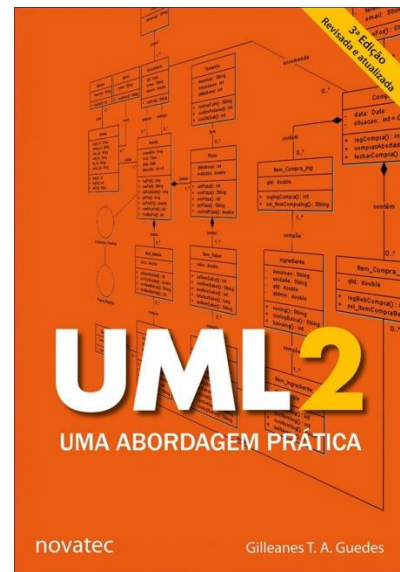
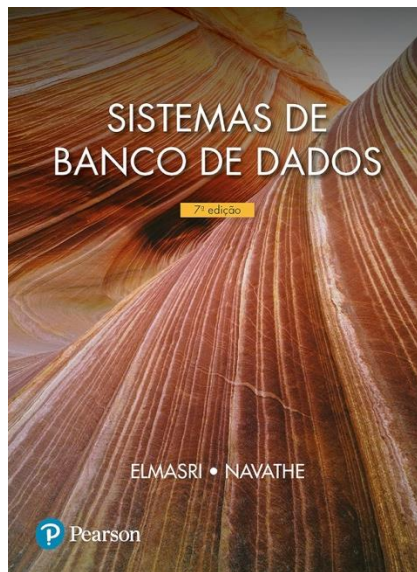
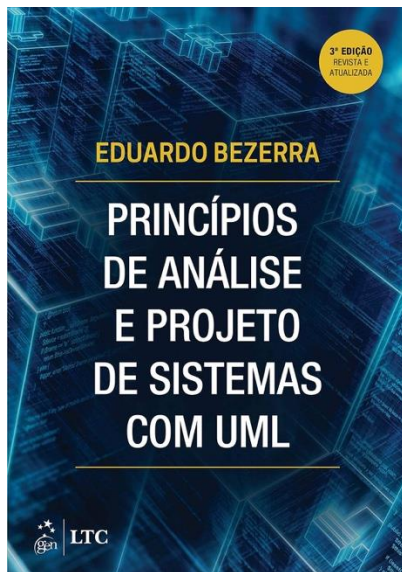
- BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML**. 3. ed. rev. e atual. Rio de Janeiro: Elsevier, 2015. xvii, 398 p. ISBN 9788535226263. (Disponível no Acervo).
- ELMASRI, Ramez; Navathe, Shamkant B. **Sistemas de banco de dados**, 7ª ed. Editora Pearson 1152 ISBN 9788543025001. (Livro Eletrônico).
- GUEDES, Gilleanes T. A. **UML 2**: uma abordagem prática. 2. ed. São Paulo: Novatec, c2011. 484 p. ISBN 9788575222812. (Disponível no Acervo).
- **IEEE TRANSACTIONS ON SOFTWARE ENGINEERING**. New York: IEEE Computer Society, 1975-. Mensal,. ISSN 0098-5589. Disponível em: <https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=32>. Acesso em: 19 jul. 2024. (Periódico On-line).
- SOMMERVILLE, Ian. **Engenharia de software**. 10. ed. São Paulo: Pearson Education do Brasil, c2019. xii, 756 p. ISBN 9788543024974. (Disponível no Acervo).
- WAZLAWICK, Raul Sidnei. **Análise e design orientados a objetos para sistemas de informação**: modelagem com UML, OCL e IFML. 3. ed. Rio de Janeiro, RJ: Elsevier, Campus, c2015. 462 p. ISBN 9788535279849. (Disponível no Acervo).



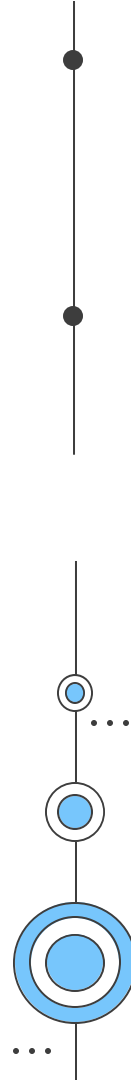


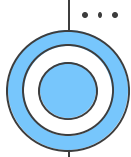
Projeto de Software

Referências complementares:



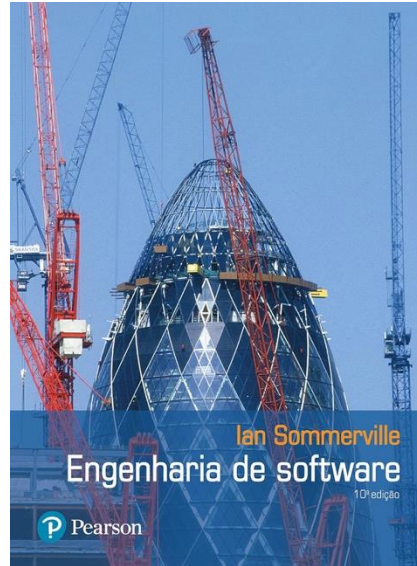
...





Projeto de Software

Referências complementares:



...

Obrigado!

Dúvidas?

joaopauloaramuni@gmail.com



[GitHub](#)



[LinkedIn](#)



[Lattes](#)

...