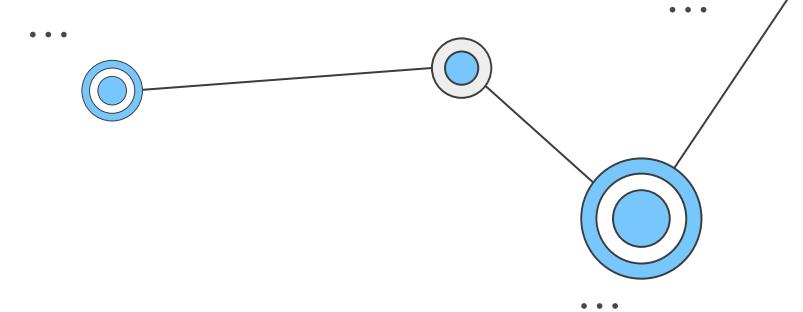
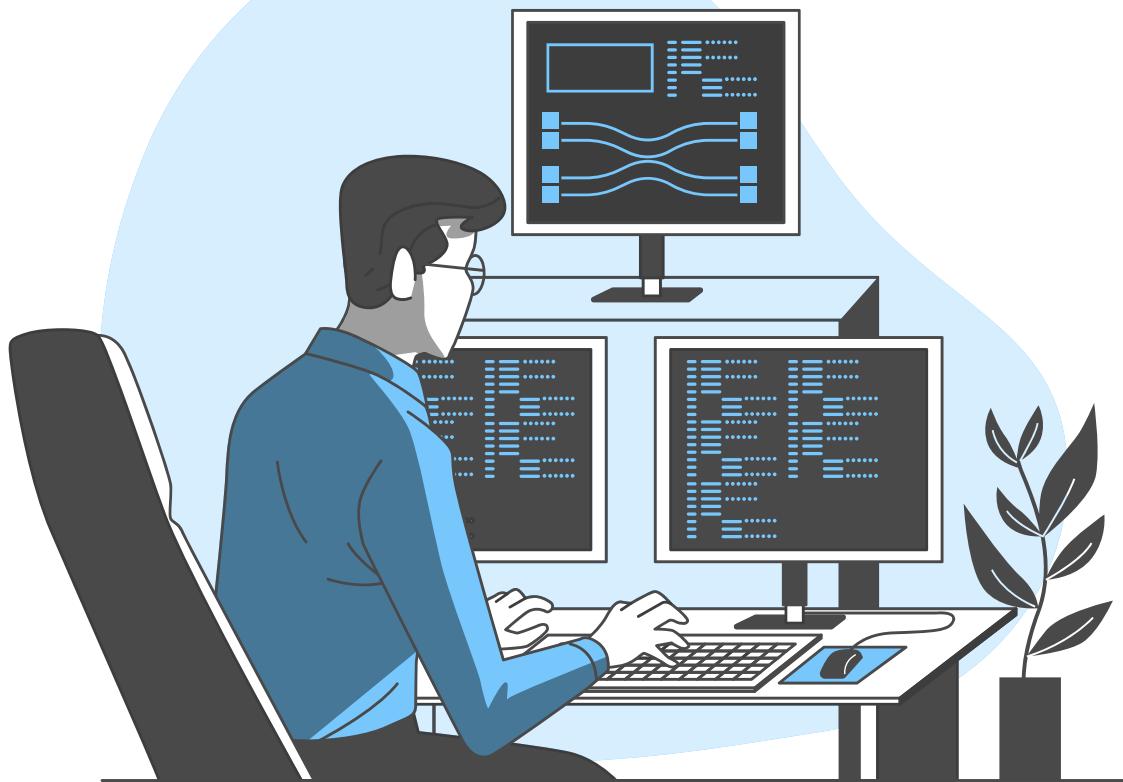




PUC Minas



Projeto de Software

Prof. Dr. João Paulo Aramuni

Unidade 2

Arquitetura de Software

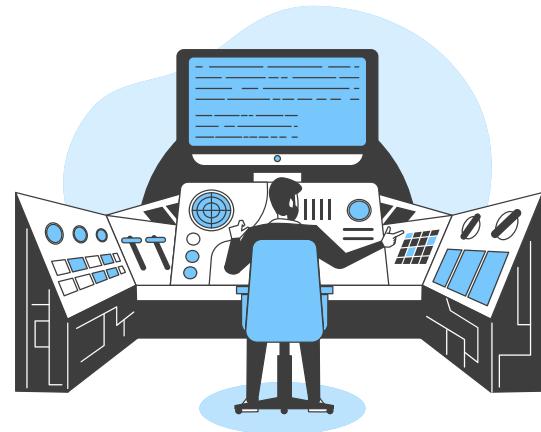
PDS - Manhã

Arquitetura de Software - Modelagem

Arquitetura de Software na UML (Unified Modeling Language)

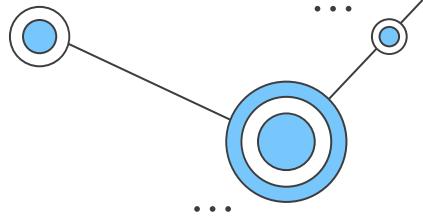
Na UML os seguintes conceitos são usados para modelar a arquitetura:

- Visão Lógica (projeto)
 - ✓ Pacotes
 - ✓ Subsistemas
 - ✓ Componentes
 - ✓ Interfaces
 - ✓ Camadas
- Visão Implementação
 - ✓ Diagramas de componentes
- Visão Implantação
 - ✓ Diagramas de implantação



Arquitetura de Software - Modelagem

Modelagem Arquitetura Lógica

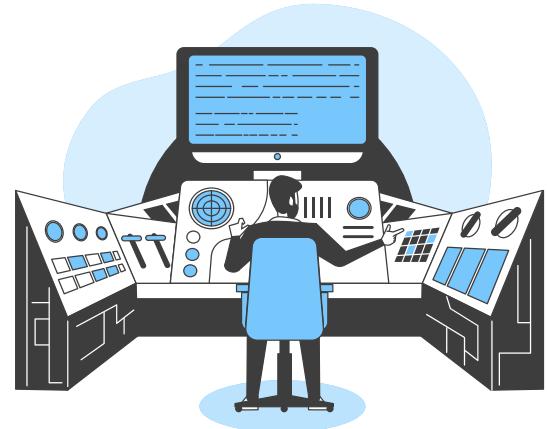
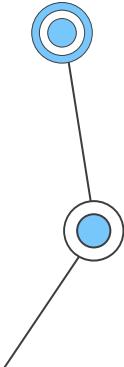


Objetivo:

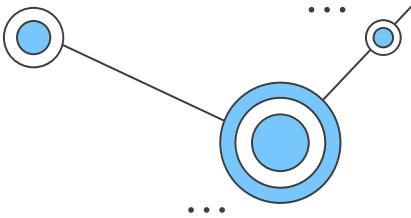
- Capturar a organização dos principais *Elementos* de um sistema e seus relacionamentos.

Diagramas básicos:

- Diagramas de pacotes e classes.

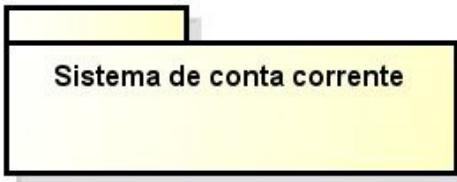


Arquitetura de Software - Modelagem

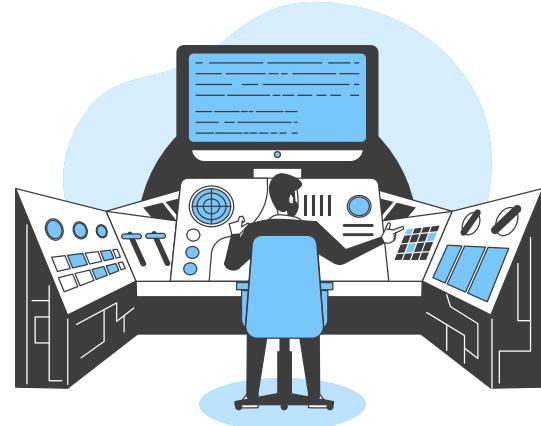


Pacotes

- Um mecanismo de agrupamento geral que pode ser utilizado para agrupar vários artefatos de um modelo.
- Notação: uma pasta com uma aba.



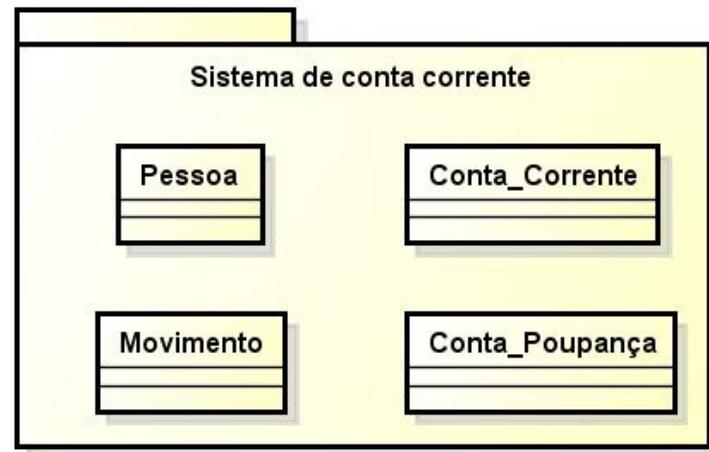
- Um pacote pode ser usado:
 - Para organizar modelos em desenvolvimento.
 - Como uma unidade de gerencia de configuração.



Arquitetura de Software - Modelagem

Pacotes

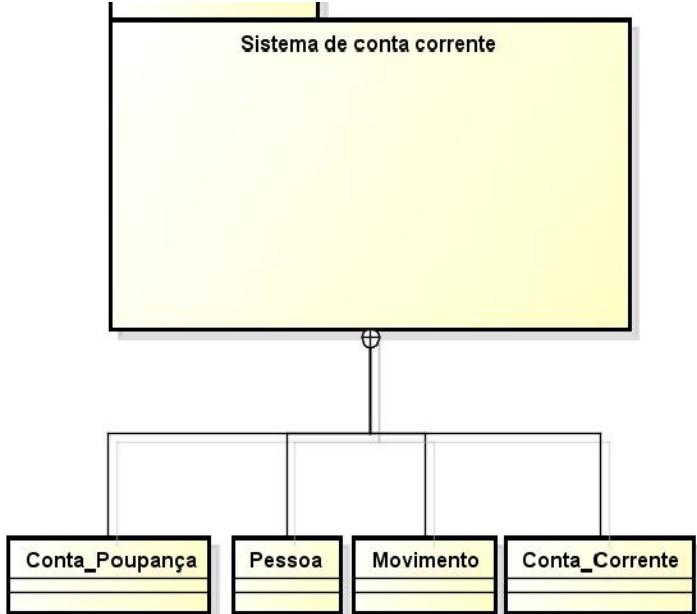
- Conteúdo, duas formas de representar graficamente:
 - Primeira forma: exibir o conteúdo dentro do pacote.
- ✓ Identificamos os elementos contidos pelo pacote, sem definir as características ou
- ✓ Definimos o diagrama completo.



Arquitetura de Software - Modelagem

Pacotes

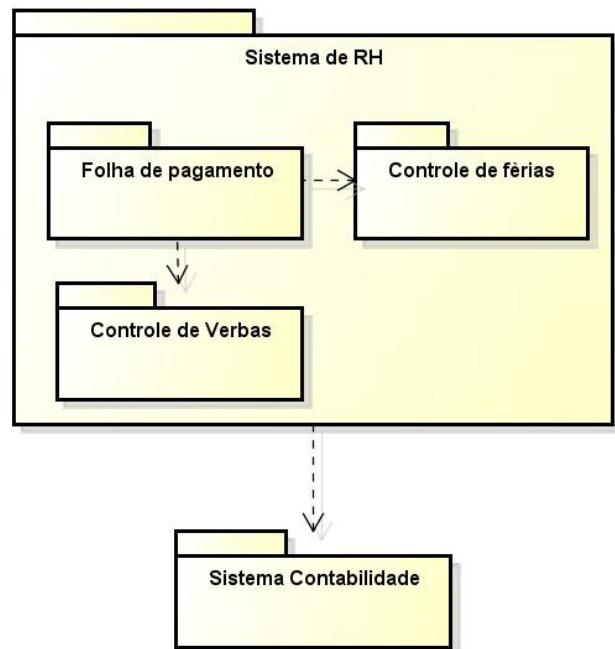
- Segunda forma:
 - Exibir os membros do pacote por meio do conector de aninhamento.



Arquitetura de Software - Modelagem

Diagrama de Pacotes

- Pacotes podem ser agrupados dentro de outros pacotes, formando uma hierarquia de contenção.



Arquitetura de Software - Modelagem

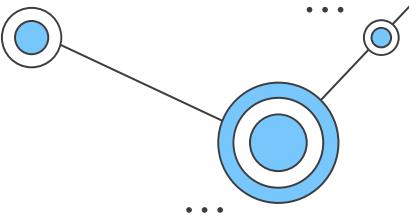
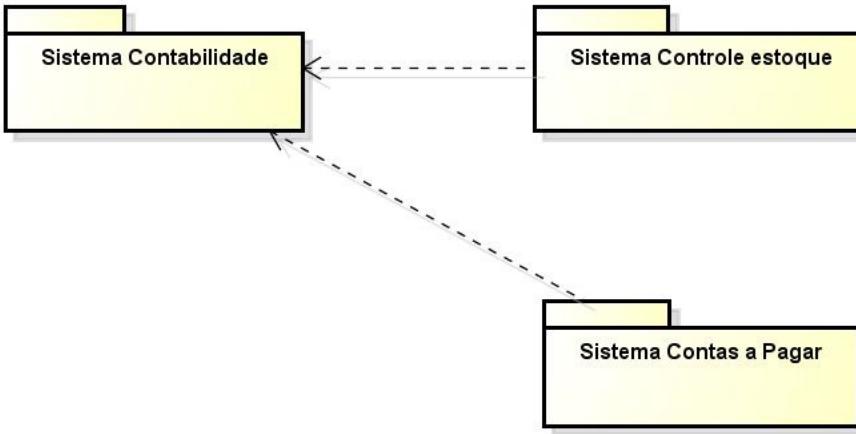


Diagrama de Pacotes - Dependências

- Pode haver relacionamentos de dependência entre pacotes.
 - Um pacote P1 é dependente de outro, P2, se houver qualquer dependência entre quaisquer dois elementos de P1 e P2.



O relacionamento de dependência no diagrama de pacotes pode ter dois esteriótipos:

✓ <<merge>> significando que os elementos do pacote que utiliza essa dependência serão unidos aos elementos do outro pacote

✓ <<import>> significando que o pacote que utiliza essa dependência está importando alguma característica ou elemento do outro pacote

Arquitetura de Software - Modelagem

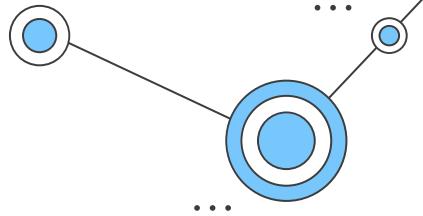
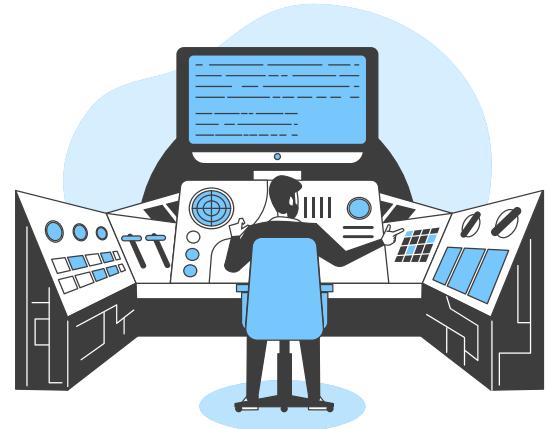
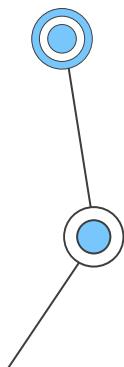
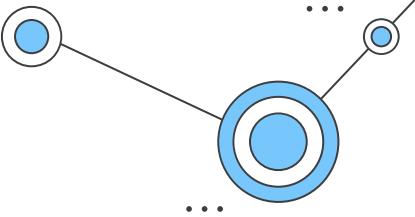


Diagrama de Pacotes - Estereótipos

- É possível aplicar estereótipos aos pacotes, deixando claros o que eles representam.
- Alguns tipos:
 - System
 - Subsystem
 - Model
 - Framework
 - Layer - camada

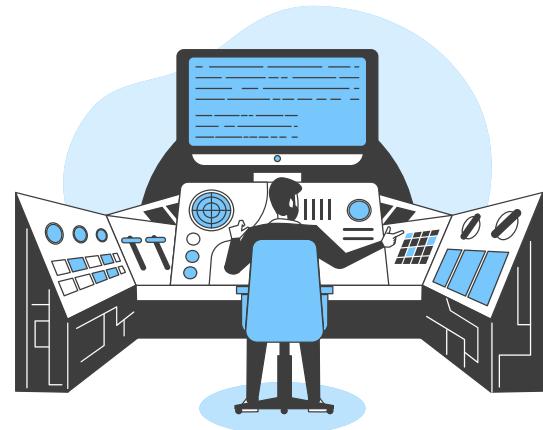


Arquitetura de Software - Modelagem



Classificador

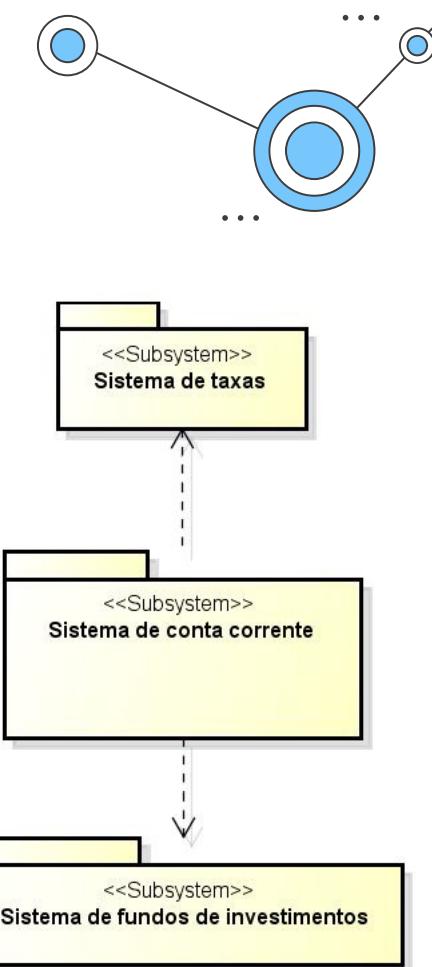
- Classificador é uma entidade de software que implementa os métodos dos serviços especificados em um ou mais interfaces.
 - Um classificador possui comportamento, ou seja, implementa um conjunto de operações.
- Na UML, classificador é um nome genérico que pode denotar:
 - uma classe
 - um subsistema
 - um componente
 - uma interface



Arquitetura de Software - Modelagem

Subsistemas

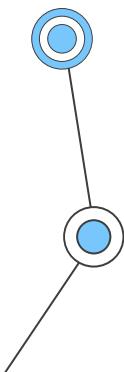
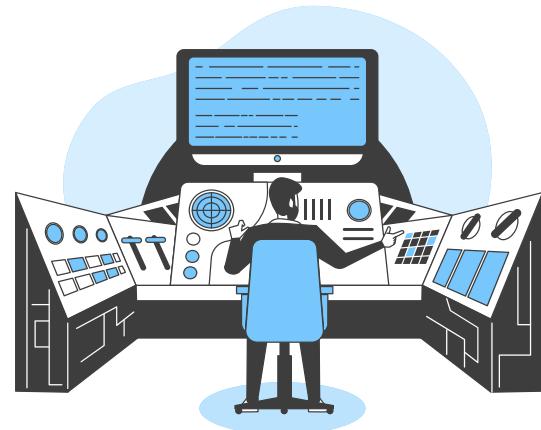
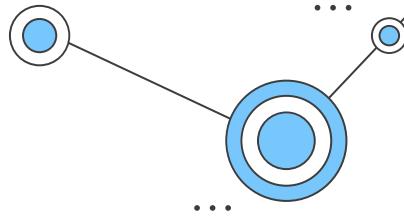
- Um subsistema é um classificador que corresponde a um agrupamento de classes.
 - Sendo um tipo de classificador, um subsistema realiza uma ou mais interfaces.
- Na UML, um subsistema é representado como um pacote com o estereótipo <<subsystem>>.
 - Importante: subsistemas podem realizar interfaces enquanto que pacotes são somente mecanismos de agrupamentos.



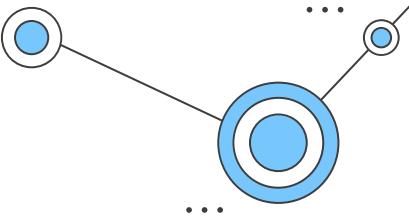
Arquitetura de Software - Modelagem

Alocando classes a subsistemas

- Durante o desenvolvimento de um sistema orientado a objetos, seus subsistemas devem ser identificados, juntamente com as interfaces entre eles.
- Cada classe do sistema é então alocada aos subsistemas.
- Finalmente, um diagrama de subsistemas é construído.
- Uma vez feita a sua identificação e a definição de sua interface, cada subsistema pode ser desenvolvido quase que de forma independente um do outro.

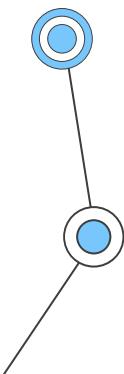
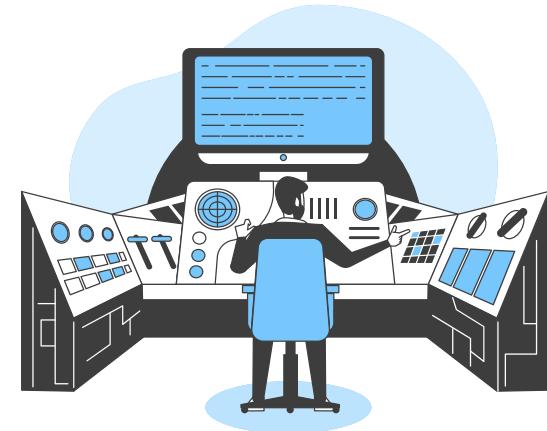


Arquitetura de Software - Modelagem



Alocando classes a subsistemas

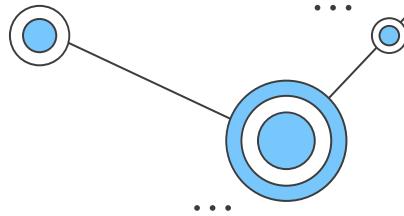
- Modelo de classes de domínio: ponto de partida para a identificação dos subsistemas.
 - As classes devem ser agrupadas segundo algum critério para formar subsistemas.
- Um critério de agrupamento possível: considerar as classes mais importantes do modelo de classes de domínio.
 - Para cada uma dessas classes, um subsistema é criado.
 - Outras classes menos importantes e relacionadas a uma classe considerada importante são posicionadas no subsistema desta última.



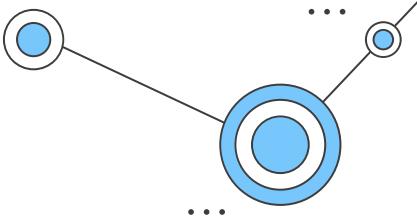
Arquitetura de Software - Modelagem

Dicas para a definição de subsistemas

- Subsistemas devem ser minimamente acoplados.
- Subsistemas devem ser maximamente coesivos.
- Dependências cíclicas entre subsistemas devem ser evitadas.
 - A alternativa para eliminar ciclos é quebrar um subsistema pertencente ao ciclo em dois ou mais. Uma outra solução é combinar dois ou mais subsistemas do ciclo em um único.
- Uma classe deve ser definida em um único subsistema (embora possa ser utilizada em vários).
 - O subsistema que define a classe deve mostrar todas as propriedades da mesma.
 - Outros subsistemas que fazem referência a essa classe podem utilizar a notação simplificada da mesma.

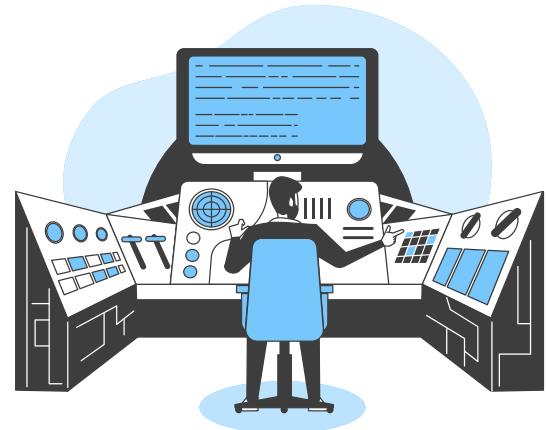
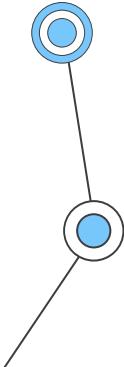


Arquitetura de Software - Modelagem



Componentes

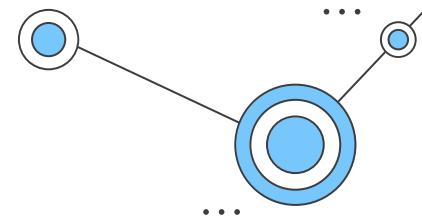
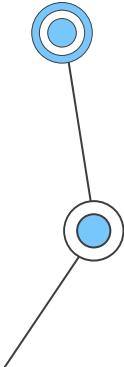
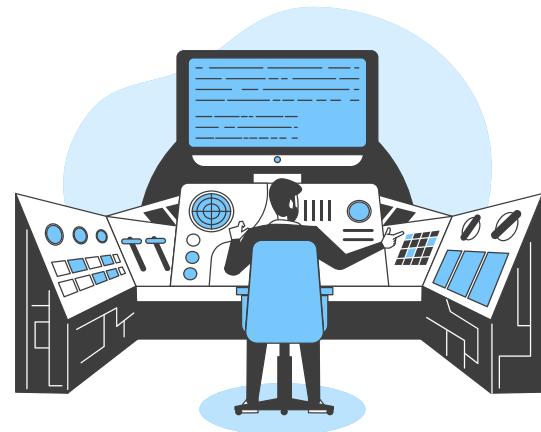
- Além dos subsistemas e das classes, um outro tipo de classificador é o componente.
- Detalhes no item Visão implementação.



Arquitetura de Software - Modelagem

Interfaces

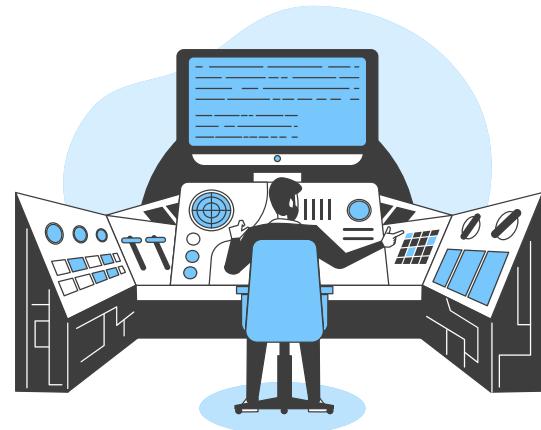
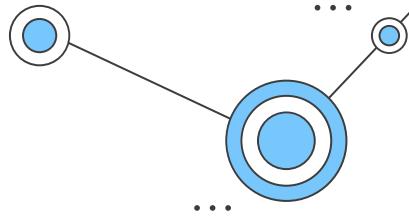
- Definição (**serviço**): um serviço é alguma tarefa que uma entidade de software realiza para outra. Um serviço é composto de sua especificação e de seu método.
 - Especificação: define o que o serviço realiza, além de documentar as informações que são necessárias para a sua realização.
 - Método: corresponde ao modo de realizar um serviço. Dada a especificação de um serviço, podem existir diversos métodos para realizá-lo.



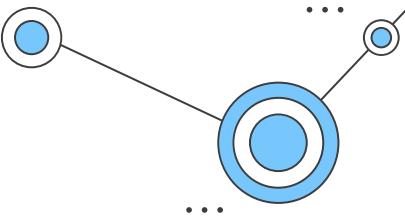
Arquitetura de Software - Modelagem

Interfaces

- Uma interface corresponde a um conjunto especificações de serviços.
- Semelhante ao conceito de classe abstrata (possui um nome, não gera instâncias).
 - Mas, uma interface não contém estrutura interna (atributos e associações).
 - Além disso, todas as operações (serviços) de uma interface só possuem especificações; o método não é definido na interface.

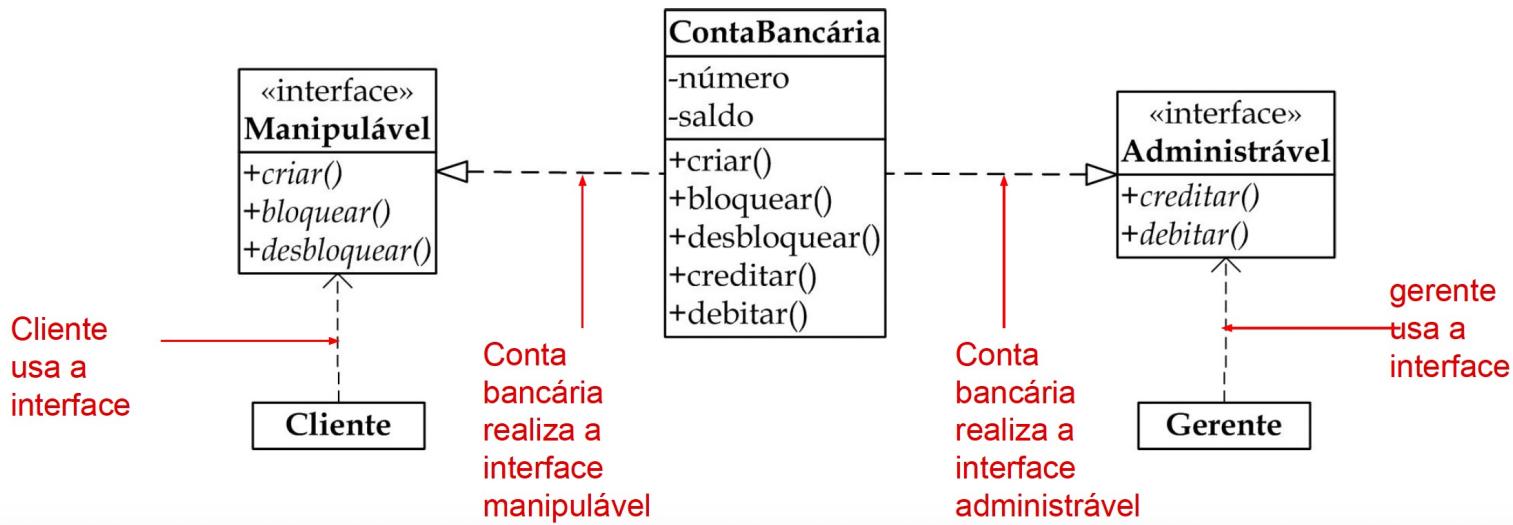


Arquitetura de Software - Modelagem



Interfaces - Notação

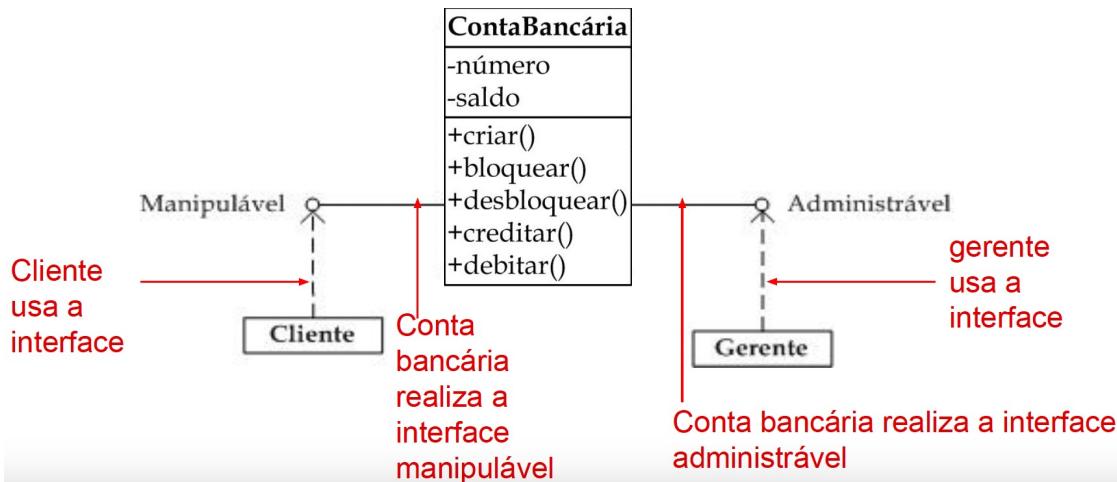
- A primeira notação: é a mesma para classes.
 - ✓ São exibidas as operações que a interface especifica.
 - ✓ Deve ser usado o estereótipo <<interface>>.



Arquitetura de Software - Modelagem

Interfaces - Notação

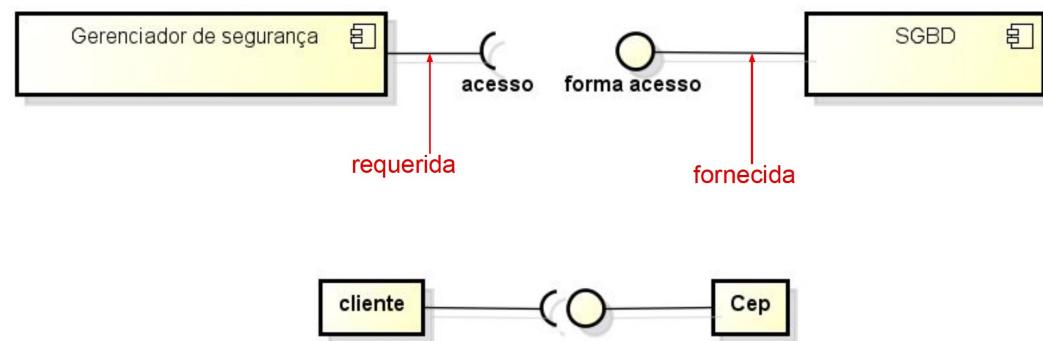
- Segunda notação: através de um segmento de reta com um pequeno círculo em um dos extremos e ligado ao classificador que a realiza no outro extremo.
 - ✓ Classes clientes conectadas à interface através de um relacionamento de dependência.



Arquitetura de Software - Modelagem

Interfaces - Tipos

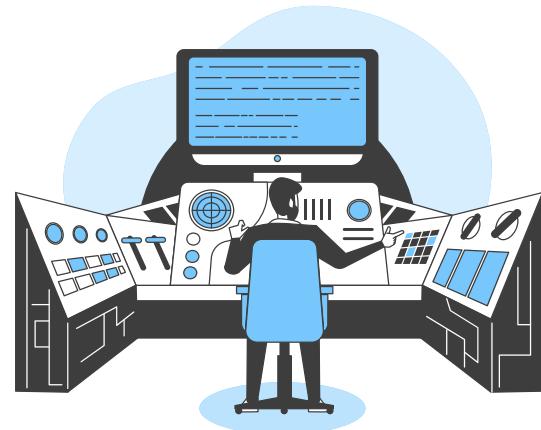
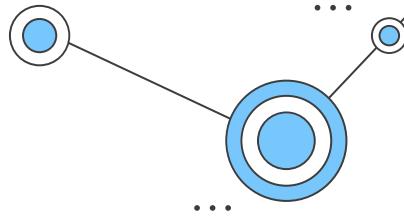
- Interface fornecida
 - ✓ Uma interface fornecida por um componente, significando uma interface em que o componente fornece um serviço para outros componentes
- Interface requerida
 - ✓ Uma interface utilizada pelo componente, significando uma interface à qual o componente se adapta quando solicita serviços de outros componentes



Arquitetura de Software - Modelagem

Camadas

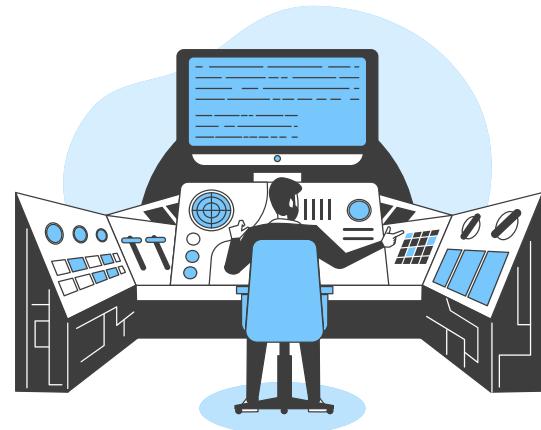
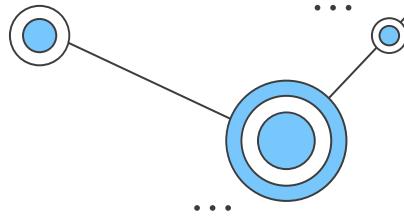
- Uma outra forma de organizar a arquitetura de um sistema complexo em partes menores é através de camadas de software.
- Uma camada de software, é uma coleção de subsistemas.
 - Cada camada corresponde a um conjunto de funcionalidades de um sistema de software.
 - Funcionalidades de alto nível dependem de funcionalidades de baixo nível.
- Camadas fornecem um nível de abstração através do agrupamento lógico de subsistemas relacionados.



Arquitetura de Software - Modelagem

Camadas

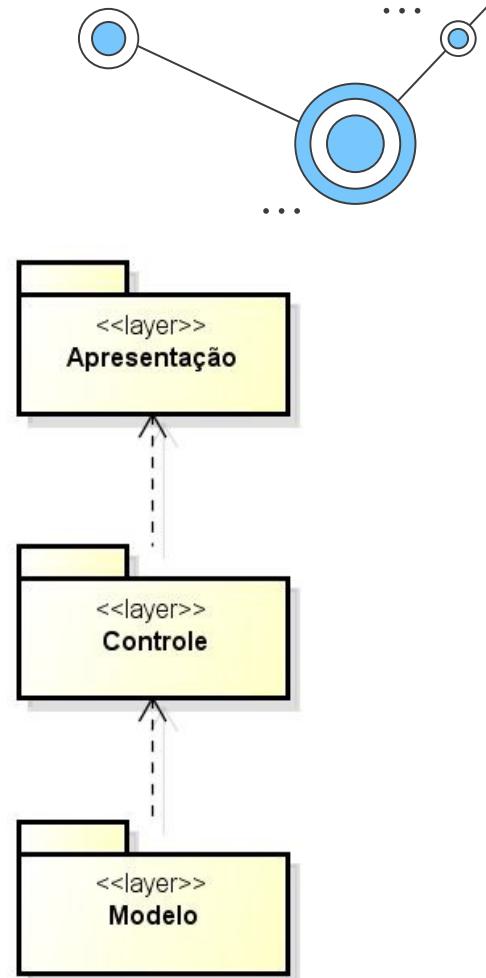
- Princípio geral: camadas de abstração mais alta devem depender das camadas de abstração mais baixa, e não o contrário.
- Permite que o sistema de software seja mais portável e modificável.
 - Uma mudança em uma camada mais baixa que não afete a sua interface não implicará em mudanças nas camadas mais altas.
 - E vice-versa, uma mudança em uma camada mais alta que não implica na criação de um novo serviço em uma camada mais baixa não irá afetar estas últimas.



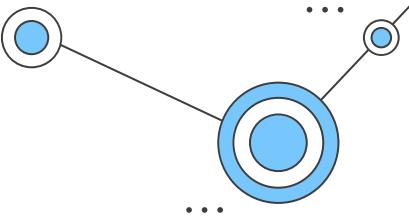
Arquitetura de Software - Modelagem

Camadas

- A UML não tem nenhum elemento gráfico pré-específico para representar camadas de um sistema.
- No entanto, pacotes também podem ser utilizados para representar camadas.
 - O estereótipo <<camada>> pode ser utilizado no pacote que represente uma camada.
 - O fato de uma camada utilizar outra pode ser representado por um relacionamento de dependência.

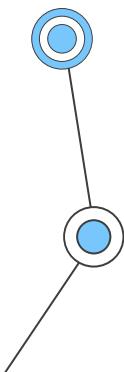
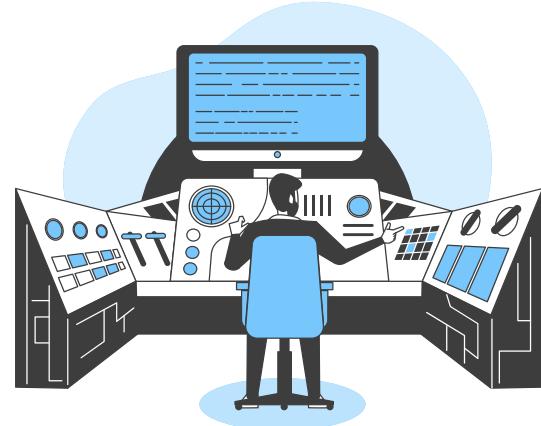


Arquitetura de Software - Modelagem



Modelagem Visão implementação

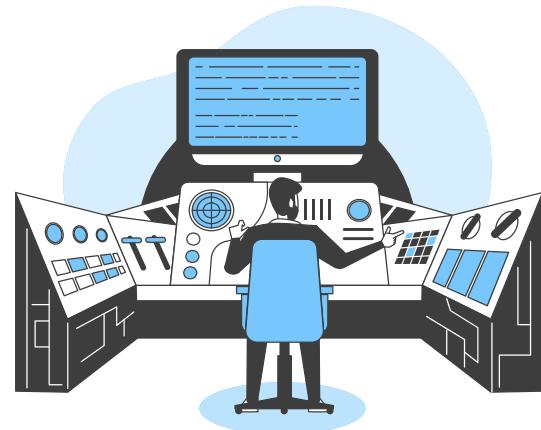
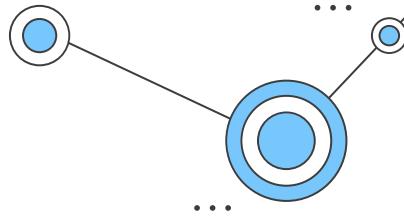
- Visualização de implementação expressa os Componentes de uma aplicação.
- Capturada através de Diagramas de Componentes UML.
 - diagrama de componentes
 - ✓ Identifica os componentes que fazem parte de um sistema, um subsistema ou até mesmo os componentes ou classes internas de um componente individual
 - ✓ Pode ser utilizado como uma forma de documentar como estão estruturados os arquivos físicos de um sistema, permitindo assim uma melhor compreensão do mesmo, além de facilitar a reutilização do mesmo



Arquitetura de Software - Modelagem

Componentes

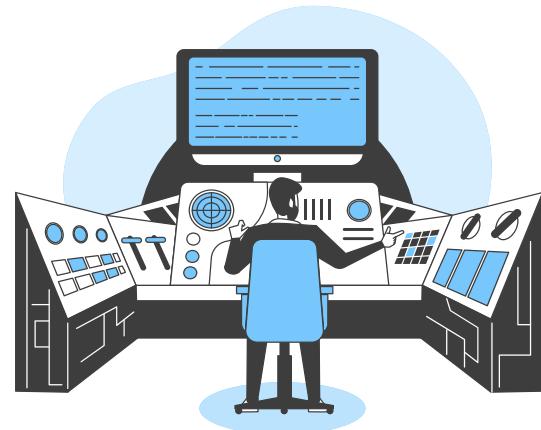
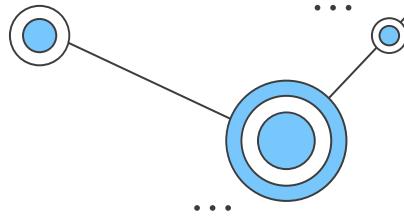
- Além dos subsistemas e das classes, um outro tipo de classificador é o componente.
- Analogia: chip de memória.
 - Um componente eletrônico que pode ser utilizada para construir um computador.
 - Pode ser substituído por outro chip mais poderoso que possua a mesma especificação.
- Da mesma forma, pode-se pensar em um sistema de software constituído de diversos componentes, elementos que existem a tempo de execução do sistema.



Arquitetura de Software - Modelagem

Componentes

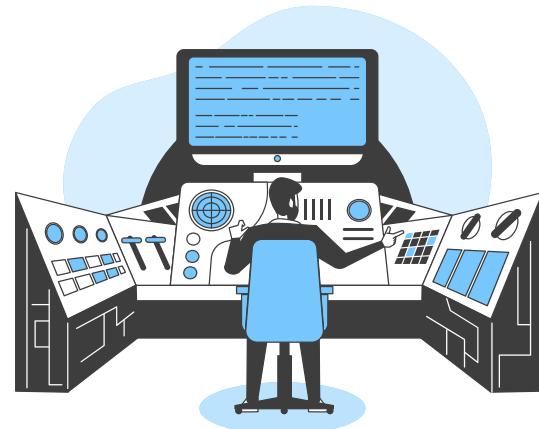
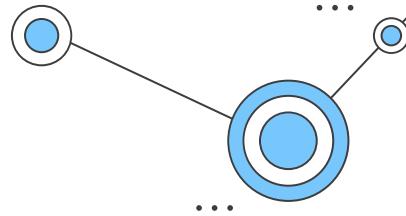
- Um componente é uma unidade configurável de software que pode ser utilizada na construção de vários sistemas e que pode ser substituída por outra unidade que forneça a mesma funcionalidade.
- Sendo um classificador, um componente pode prover acesso aos seus serviços através de uma interface.
- Adicionalmente, um componente fornece serviços a outros componentes do sistema, mas pode também requisitar serviços.



Arquitetura de Software - Modelagem

Componentes x Classes

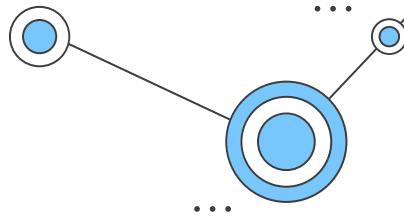
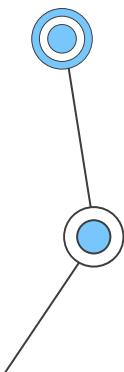
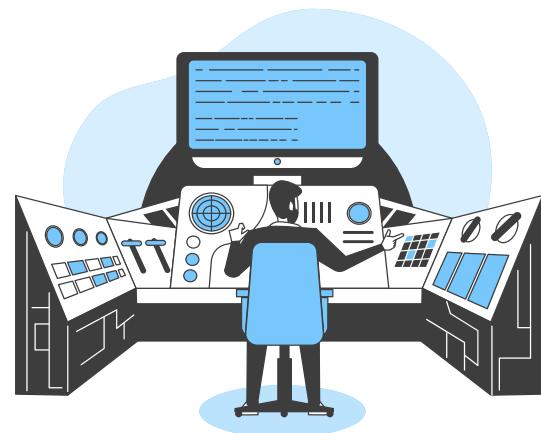
- As classes representam abstrações lógicas;
 - os componentes representam coisas que vivem no mundo dos bits;
- As classes podem em ter atributos e operações diretamente.
 - Em geral, os componentes somente têm operações que alcançadas por meio de suas interfaces;
- Os componentes representam o pacote físico de componentes lógicos e se encontram em um nível de abstração diferente;



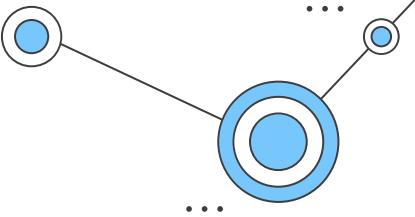
Arquitetura de Software - Modelagem

Componentes x Classes

- Outra diferença é que componentes representam conceitos físicos.
 - ao invés de conceitos lógicos representados por classes e subsistemas.
- Componentes são o empacotamento físico do sistema: uma única classe pode fornecer objetos para vários componentes.
- Quando construído segundo o paradigma da OO, um componente é composto de diversos objetos.

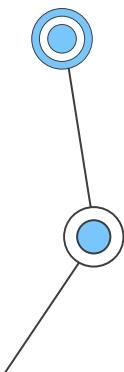
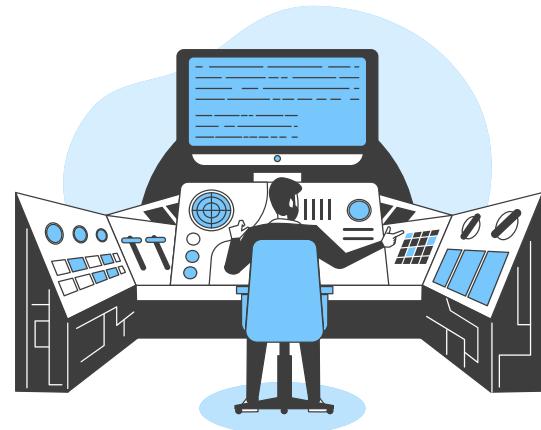
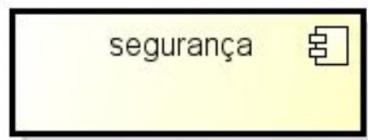


Arquitetura de Software - Modelagem



Componentes na UML

- UML: "um componente de software é um módulo, pacote ou subsistema que tem uma função e uma interface claramente definidas e pode ser integrado em um ou mais sistemas".
- A UML define prevê diversos estereótipos prontos para representar componentes, como:
 - Executável, biblioteca, tabela, documento, arquivo
- UML fornece uma notação específica para componentes, detalhada mais adiante.

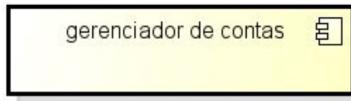


Arquitetura de Software - Modelagem

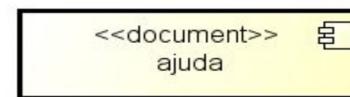
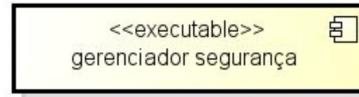
Diagrama de componentes

- Notação

Nome simples



Nome simples com estereótipos



Com interfaces

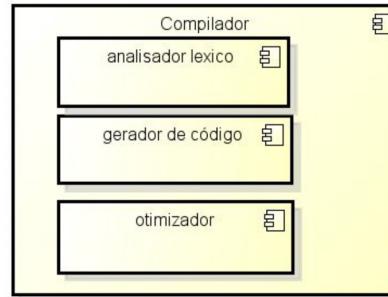


Arquitetura de Software - Modelagem

Diagrama de componentes

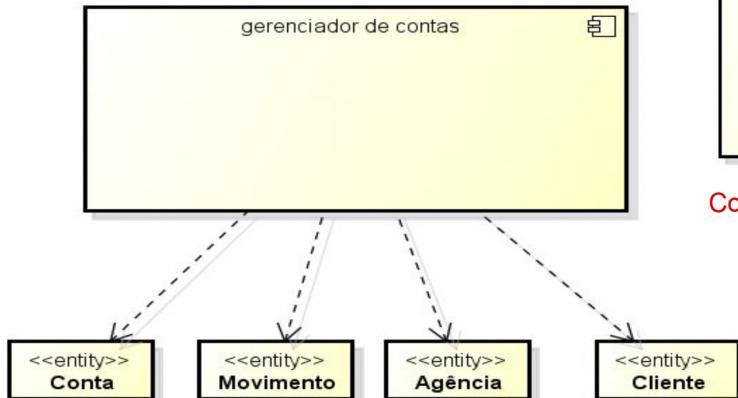


Componentes com classes internas



Compilar

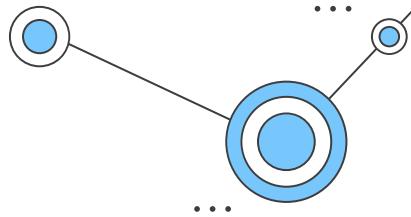
Componentes com componentes internos



Classes internas relacionadas com dependência

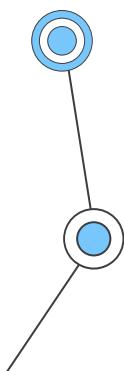
Arquitetura de Software - Modelagem

Diagrama de componentes



Portas

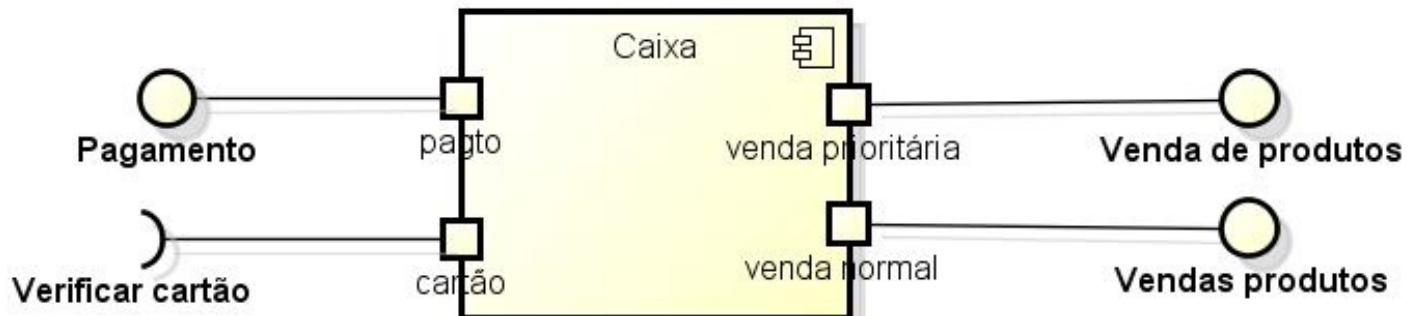
- As interfaces são úteis para declarar o comportamento geral de um componente, mas elas não têm identidade individual.
- A implementação do componente deve garantir meramente que todas as operações em todas as interfaces fornecidas sejam implementadas.
- As portas podem ser usadas para ter um controle maior sobre a implementação.
- Uma PORTA é uma janela explícita em um componente encapsulado.
 - É uma característica estrutural que encapsula a interação entre o conteúdo de uma classe e seu ambiente.
 - O comportamento da porta é especificado por suas interfaces fornecidas e requeridas
 - Elas permitem que a estrutura interna possa ser modificada sem afetar os clientes externos
 - clientes externos não têm visibilidade para internos
 - Uma classe ou componente pode ter várias portas
 - Cada porta tem um conjunto de interfaces fornecidas e requeridas



Arquitetura de Software - Modelagem

Diagrama de componentes

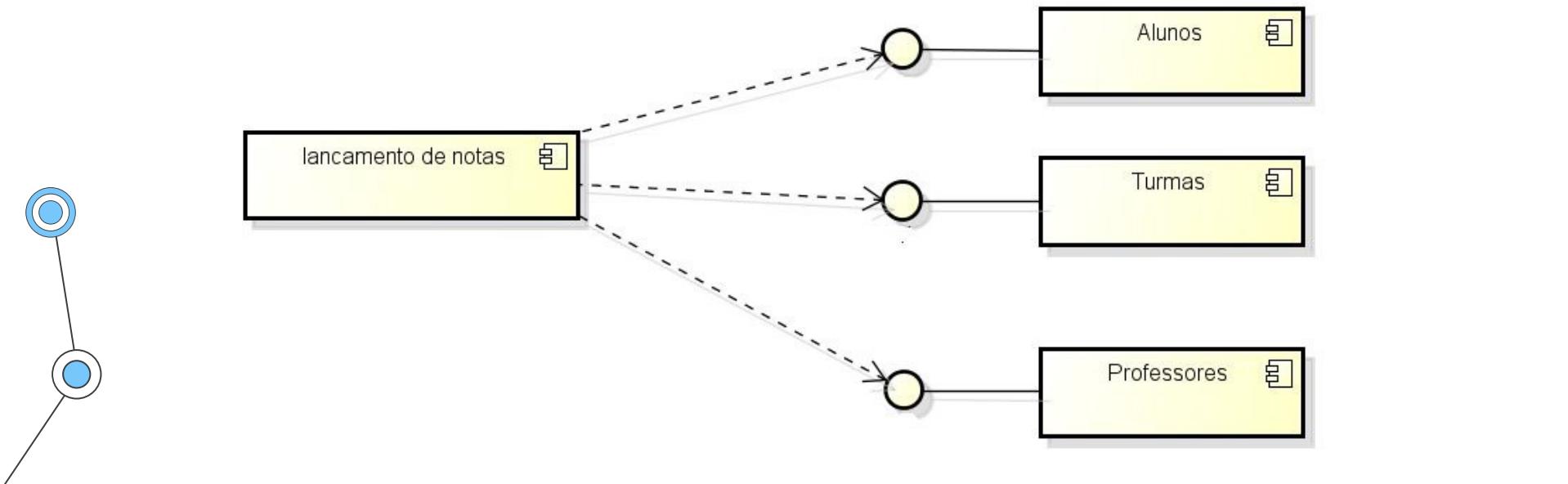
Portas



Arquitetura de Software - Modelagem

Diagrama de componentes

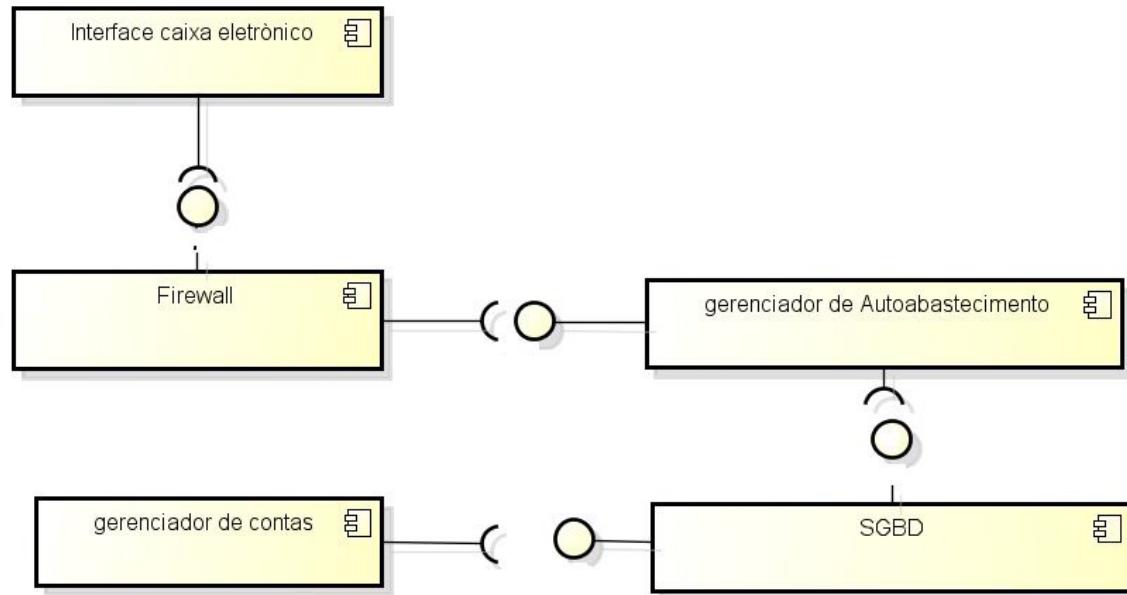
Exemplo de diagrama de componentes.



Arquitetura de Software - Modelagem

Diagrama de componentes

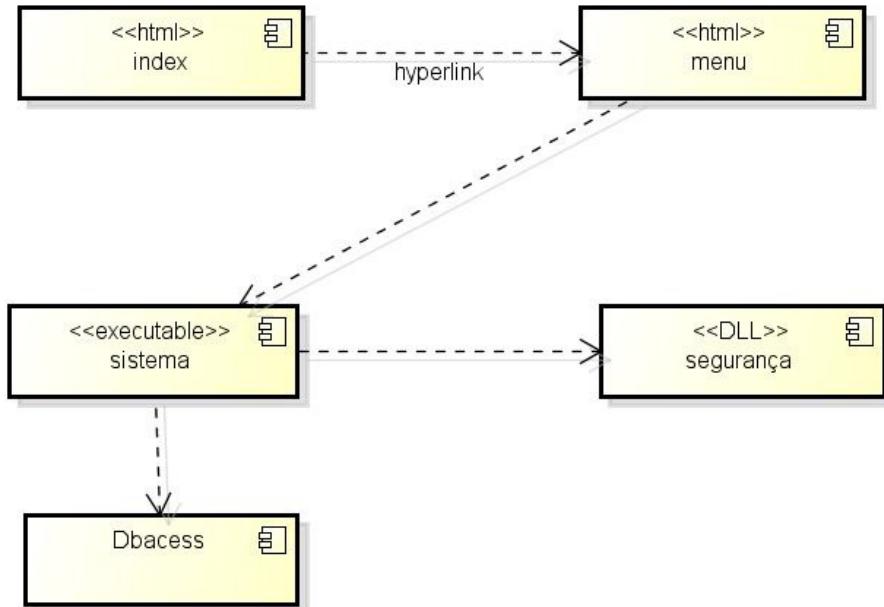
Exemplo de diagrama de componentes.



Arquitetura de Software - Modelagem

Diagrama de componentes

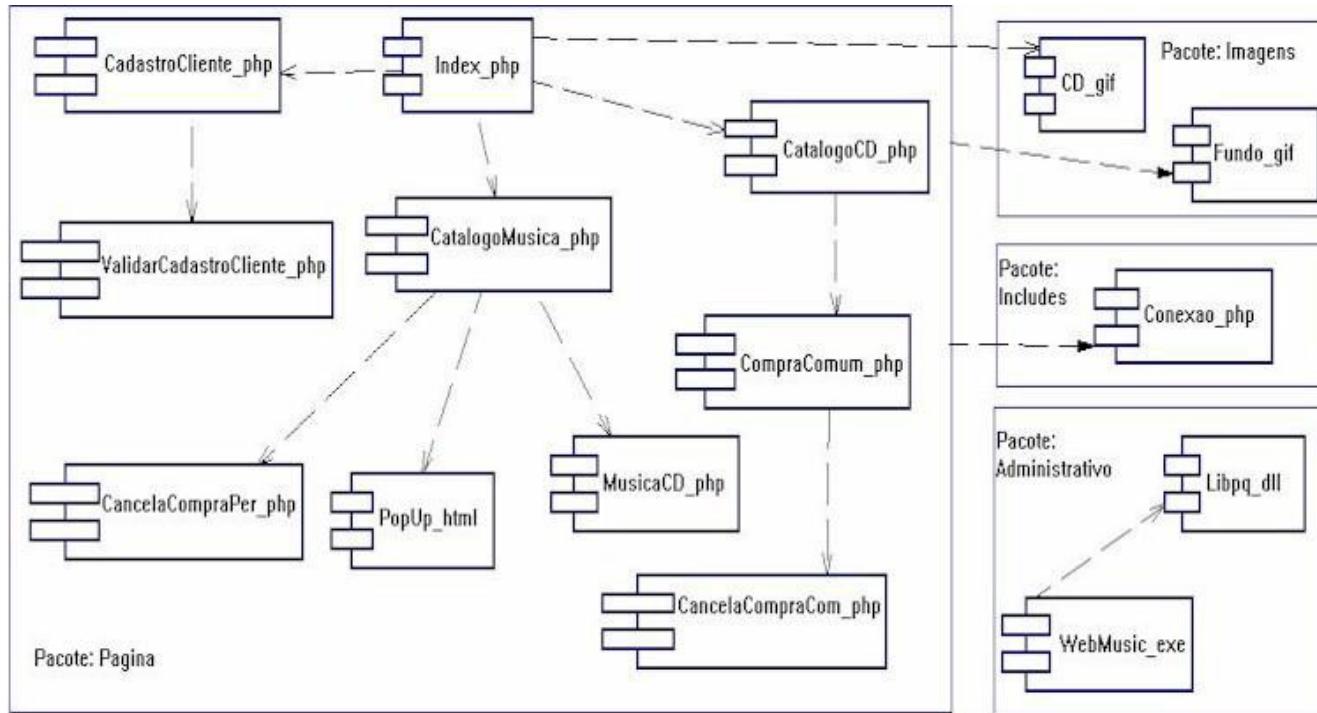
Exemplo de diagrama de componentes.



Arquitetura de Software - Modelagem

Diagrama de componentes

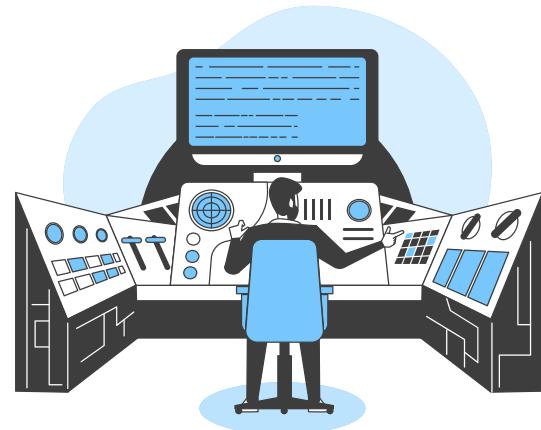
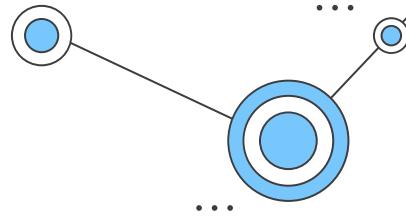
Exemplo de diagrama de componentes. (Pacotes - diretório [UML 1.5])



Arquitetura de Software - Modelagem

Diagrama de componentes - dicas

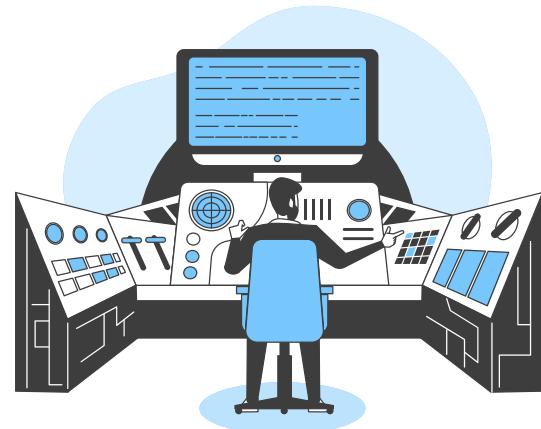
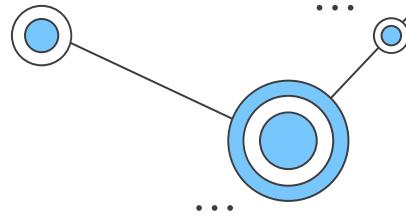
- Tem como foco comunicar um aspecto da visão estática de implementação sistema.
- Contém somente os elementos essenciais à compreensão desse aspecto.
- Fornece detalhes consistentes com o seu nível de abstração.
- Deve ter o suficiente para entender o seu propósito.
- Use elementos estereotipados para dar mais clareza.
- Use notas e cores para facilitar o entendimento.



Arquitetura de Software - Modelagem

Modelagem Visão implantação

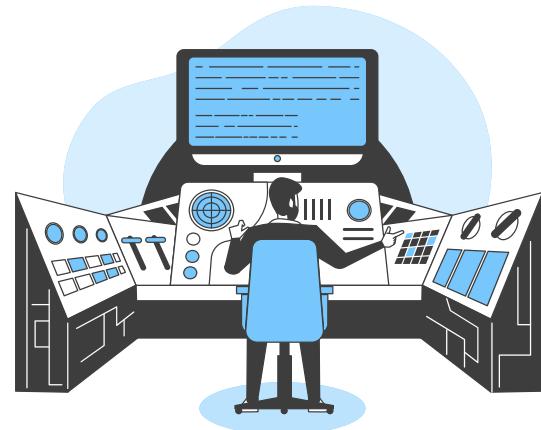
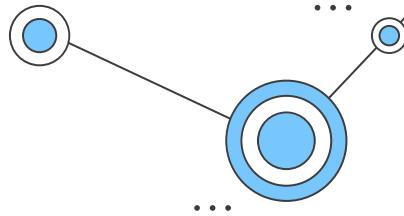
- Objetivo:
 - Expressar – a configuração dos processadores (nodos) em tempo de execução,
 - ✓ Os links de comunicação entre eles e
 - ✓ As instâncias dos componentes e objetos que residem neles.
- A visualização de implantação é capturada através de diagramas de implantação UML.
 - diagrama de implantação mostra a topologia física da rede



Arquitetura de Software - Modelagem

Diagrama de implantação

- Os nós são conectados uns aos outros através das conexões.
 - Conexões mostram mecanismos de comunicação entre os nós.
 - Meios físicos (cabo coaxial, fibra ótica, etc.) ou protocolos de comunicação (TCP/IP, HTTP, etc.).
- Um nó é representado através de um cubo.
 - Nome e tipo do nó definidos no interior do cubo.
 - A sintaxe para o nome e o tipo do nó é similar à utilizada para os diagramas de objetos.
 - Tanto o nome quanto o tipo são opcionais.
- Uma conexão é representada graficamente por uma linha (estereotipada) ligando dois nós.



Arquitetura de Software - Modelagem

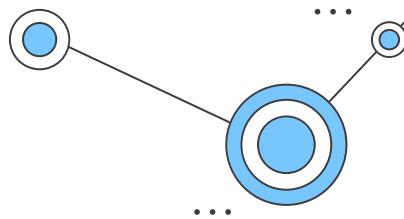
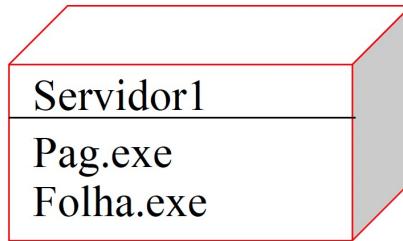
Diagrama de implantação

- Notação

Nome simples



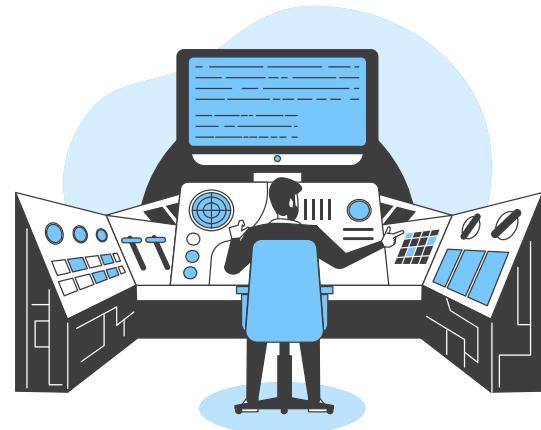
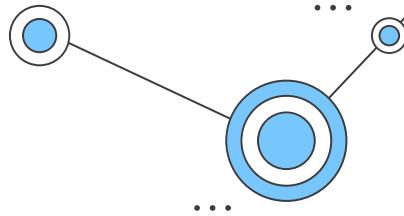
Nós estendidos



Arquitetura de Software - Modelagem

Diagrama de implantação

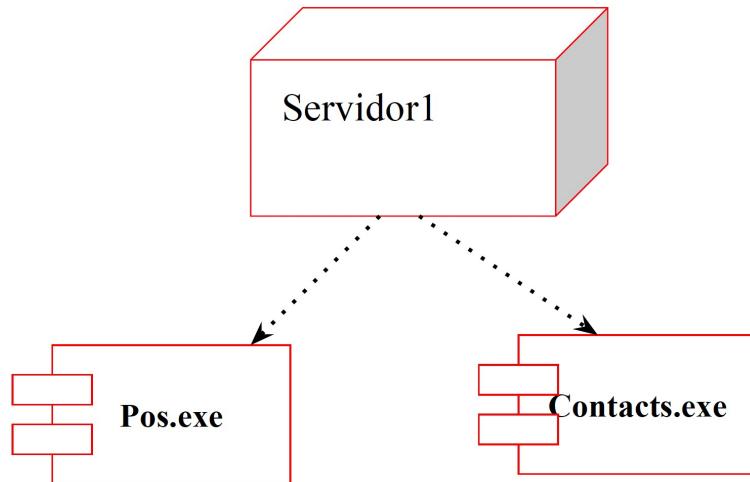
- Nó x Componente
- Os componentes são itens que participam da execução de um sistema;
 - os nós são itens que executam os componentes;
- Os componentes representam os pacotes físicos de elementos lógicos;
 - os nós representam o funcionamento físico dos componentes;



Arquitetura de Software - Modelagem

Diagrama de implantação

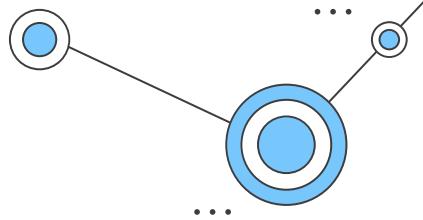
- Nós x componentes



- Os nós podem ser organizados em pacotes, da mesma maneira que se organiza classes e componentes;

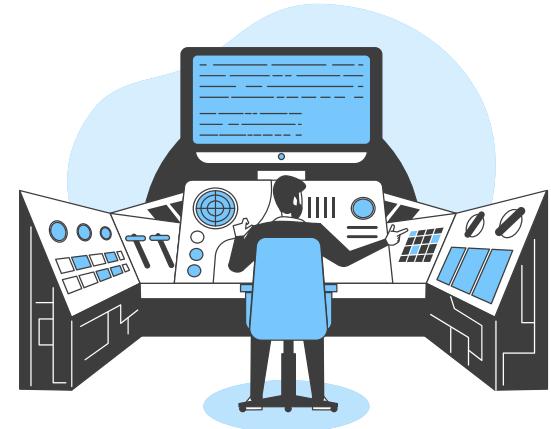
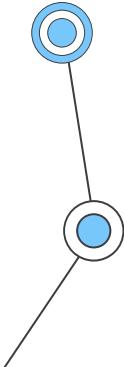
Arquitetura de Software - Modelagem

Diagrama de implantação



Usos comuns

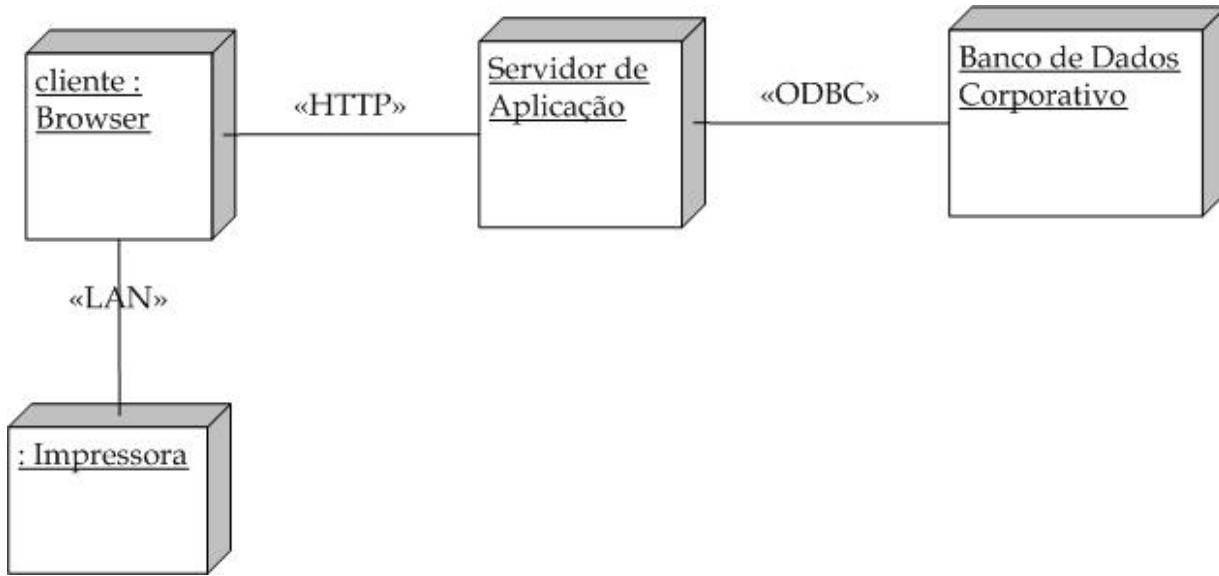
- Para fazer modelagem de sistemas embutidos.
- Para fazer modelagem de sistema cliente/servidor.
- Para fazer modelagem de sistema totalmente distribuídos.



Arquitetura de Software - Modelagem

Diagrama de implantação

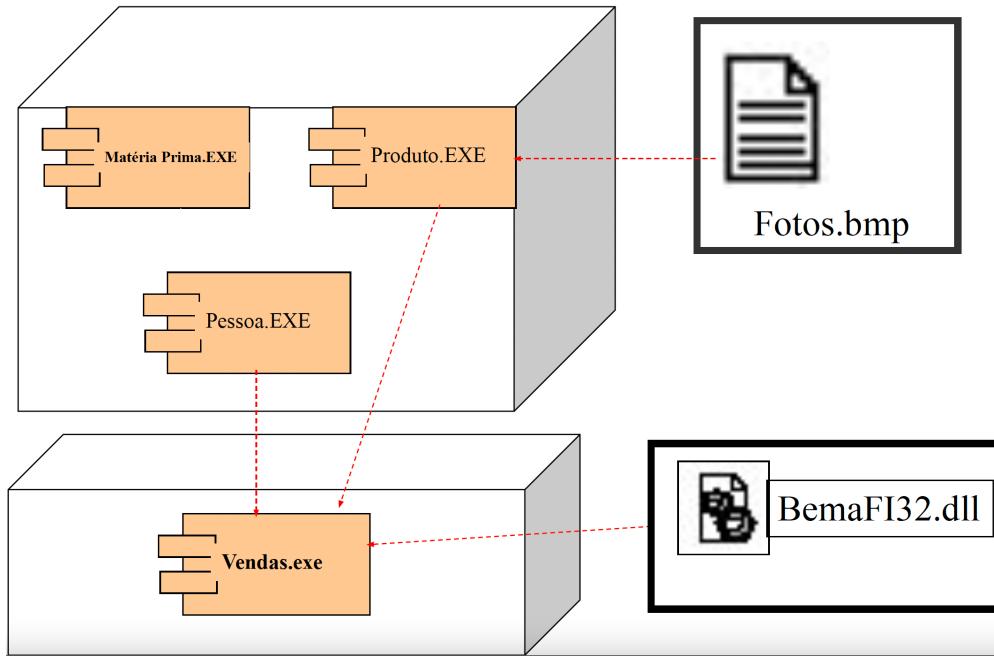
Exemplo de diagrama de implantação.



Arquitetura de Software - Modelagem

Diagrama de implantação

Exemplo de diagrama de componente.



Arquitetura de Software - Modelagem

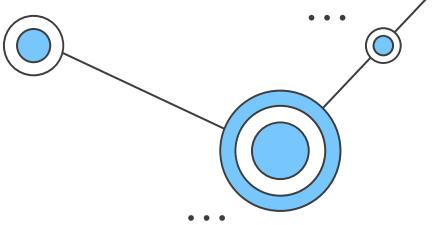
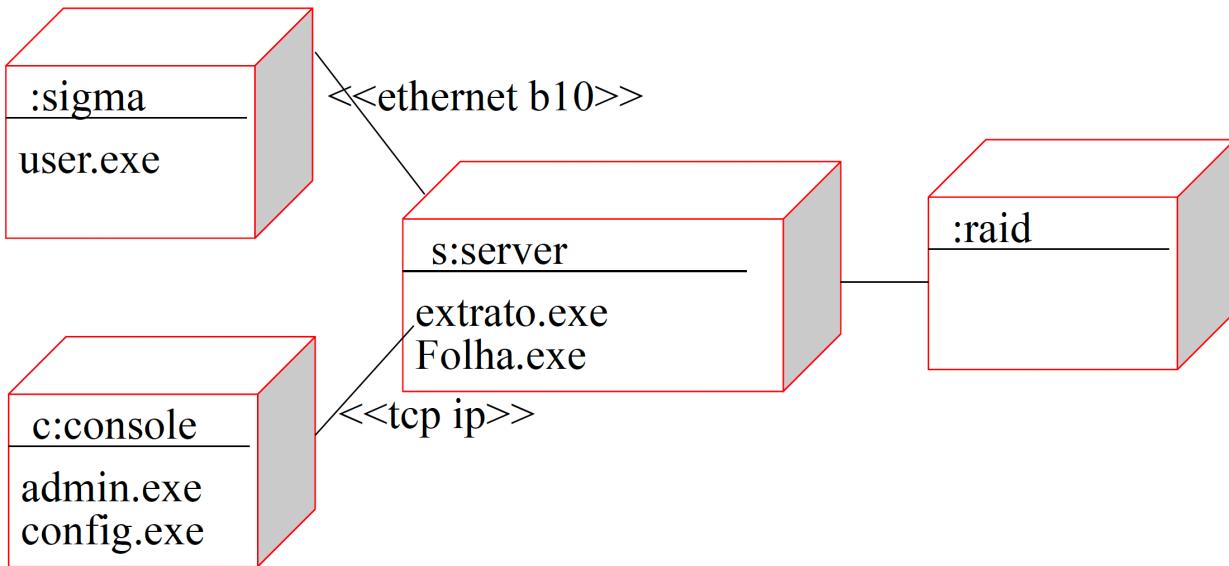


Diagrama de implantação

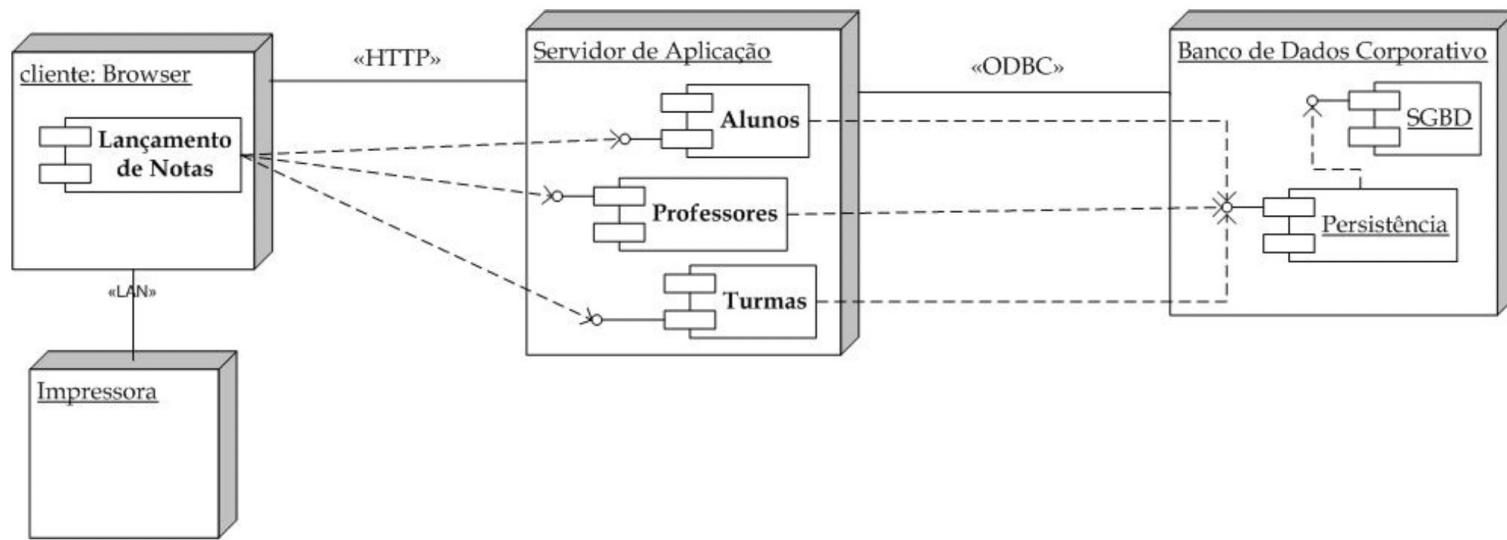
Modelagem da distribuição de componentes



Arquitetura de Software - Modelagem

Diagrama de implantação

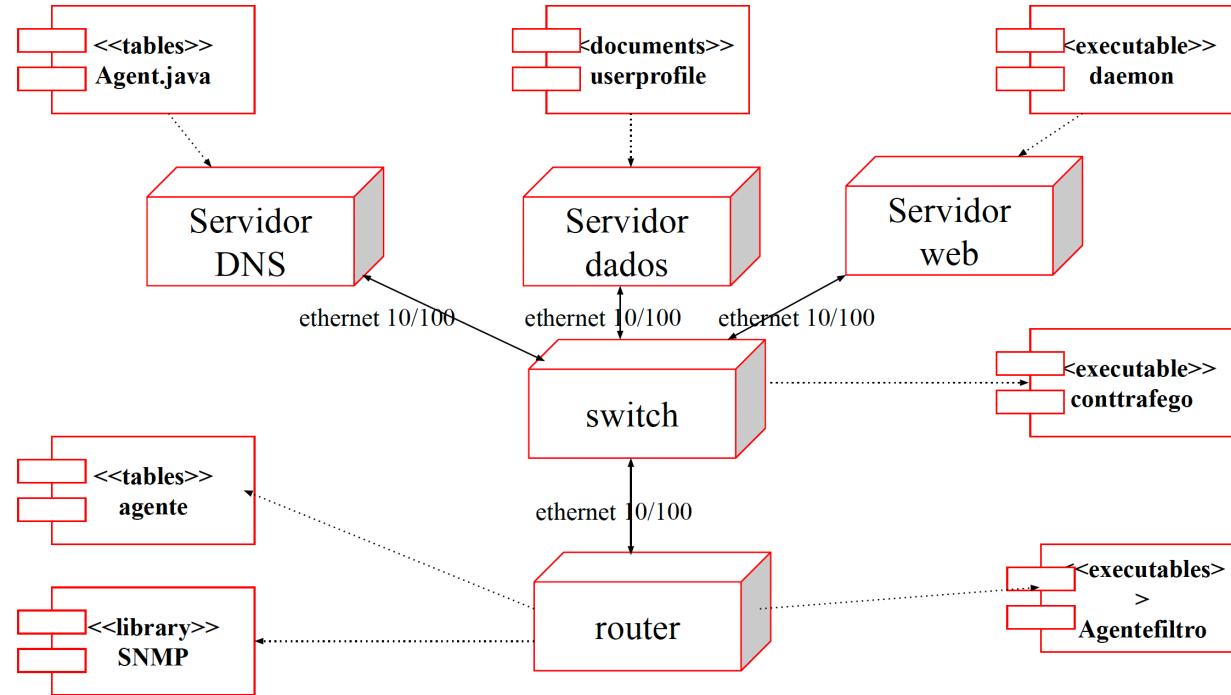
Exemplo de diagrama de implantação exibindo componentes.



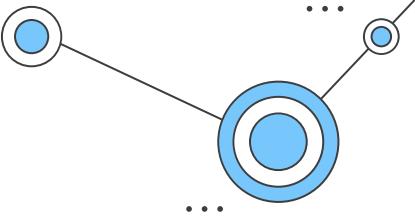
Arquitetura de Software - Modelagem

Diagrama de implantação

Exemplo de diagrama de implantação [UML 1.5]



Arquitetura de Software - Modelagem



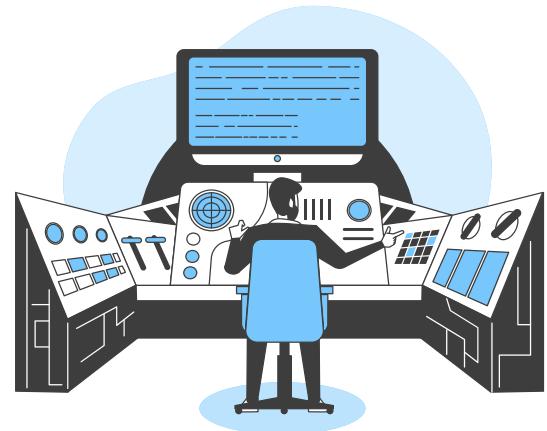
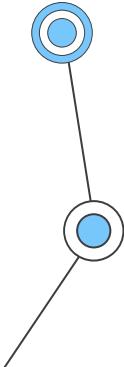
Alocando componentes aos nós

A atividade de alocação de componentes aos nós físicos só tem sentido para sistemas distribuídos.

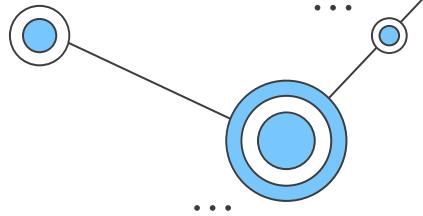
- Para sistemas que utilizam um único processador, não há necessidade desta atividade.

Um dos principais objetivos: distribuir a carga de processamento do sistema para aumentar o desempenho.

- No entanto, nem sempre isso aumenta o desempenho.
- Isso porque a sobrecarga de comunicação entre os nós pode anular os ganhos obtidos com a distribuição do processamento.



Arquitetura de Software - Modelagem



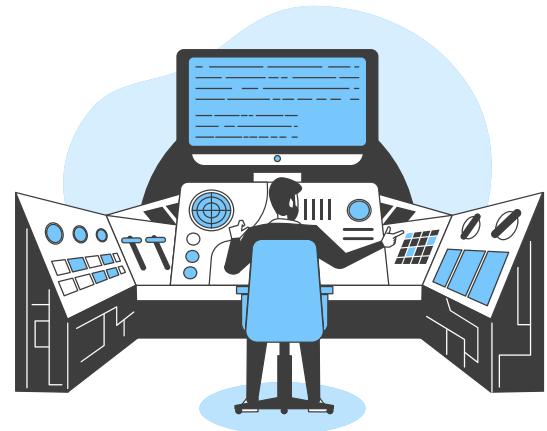
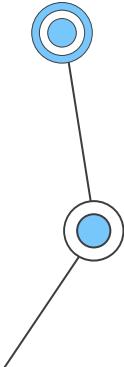
Alocando componentes aos nós

Envio de mensagem versus “distância”

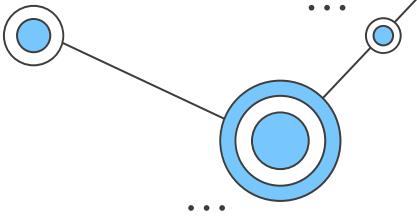
- dentro de um processo executando em um nó.
- entre processos no mesmo nó.
- ultrapassa as fronteiras de um nó para ser executada em outra máquina.

Portanto, durante a alocação de componentes, o arquiteto de software deve considerar diversos fatores.

- muitos desses fatores se sobrepõem ou são incompatíveis.



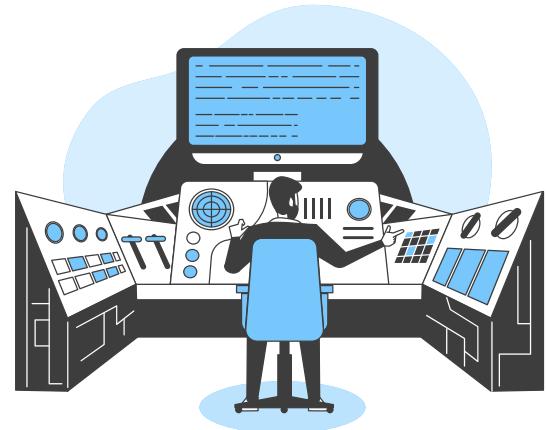
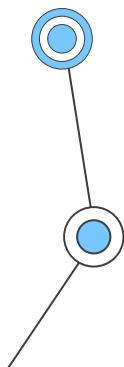
Arquitetura de Software - Modelagem



Alocando componentes aos nós

Fatores relacionados ao desempenho:

- Utilização de dispositivos
- Carga computacional
- Capacidade de processamento dos nós
- Realização de tarefas
- Tempo de resposta

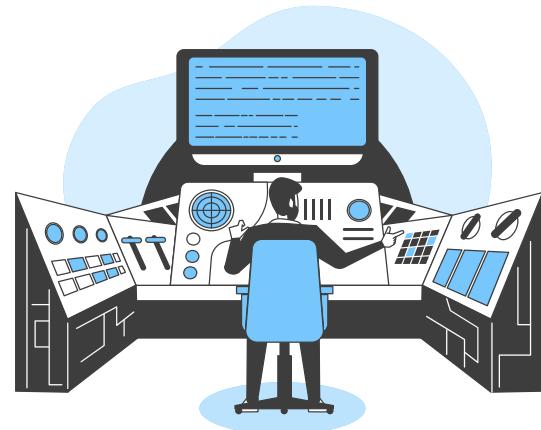
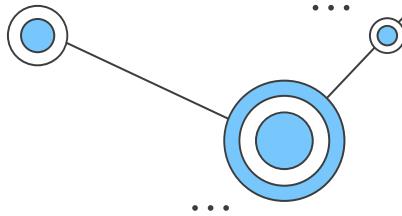


Arquitetura de Software - Modelagem

Alocando componentes aos nós

Fatores não relacionados ao desempenho:

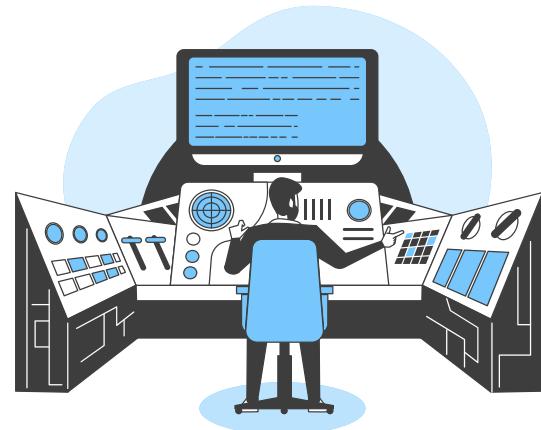
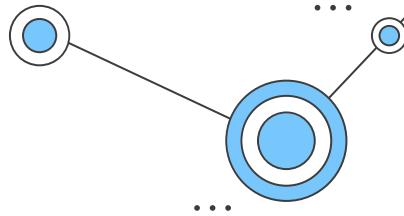
- Requisitos não funcionais do sistema.
- Segurança.
- Diferenças de plataformas (de hardware ou de sistema operacional) dos componentes do sistema.
- Características dos usuários do sistema: leva em conta a localização física dos usuários, que máquinas eles utilizam, como eles estão conectados, etc.
- Necessidade ou benefícios da distribuição das camadas lógicas do sistema.
- Redundância: o mesmo componente pode ter que ser alocado a mais de um nó.



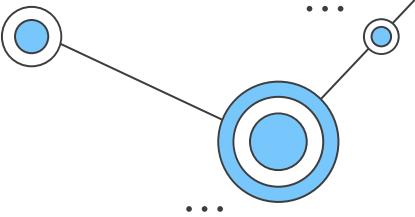
Arquitetura de Software - Modelagem

Arquitetura física no processo de desenvolvimento

- A construção dos diagramas de componentes é iniciada na fase de elaboração (projeto da arquitetura) e refinada na fase de construção (projeto detalhado).
- A construção de diagramas de componentes se justifica para componentes de execução.
 - permite visualizar as dependências entre os componentes e a utilização de interfaces a tempo de execução do sistema.
 - sistema distribuído: representar seus componentes utilizando diagramas de implantação.

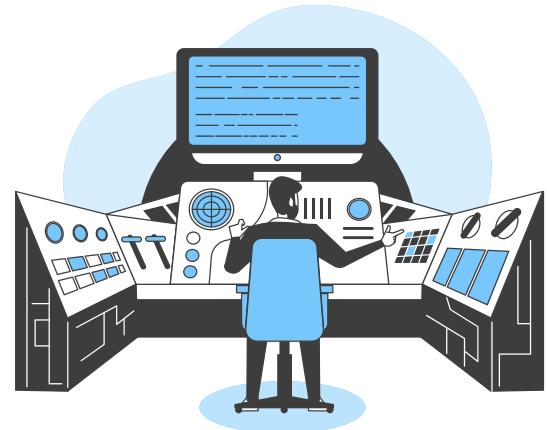
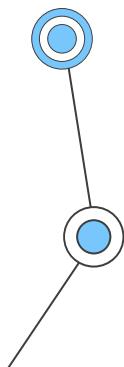


Arquitetura de Software - Modelagem

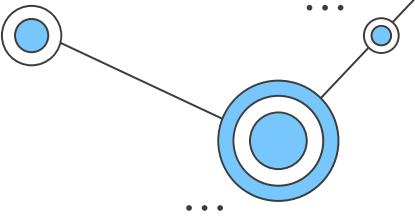


Arquitetura física no processo de desenvolvimento

- Não é recomendável usar diagramas de componentes para representar dependências de compilação entre os elementos do código fonte do sistema.
 - A maioria dos ambientes de desenvolvimento tem capacidade de manter as dependências entre códigos fonte, códigos objeto, códigos executáveis e páginas de script.
 - Só adiciona mais diagramas que terão que ser mantidos e que não terão uma real utilidade.
 - Há também vários sistemas de gerenciamento de configurações e de versões de código (e.g, CVS).

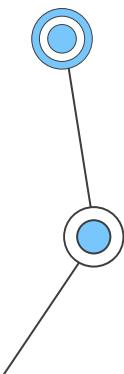
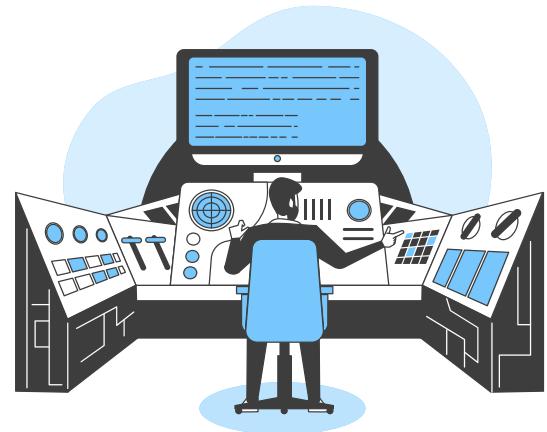


Arquitetura de Software - Modelagem

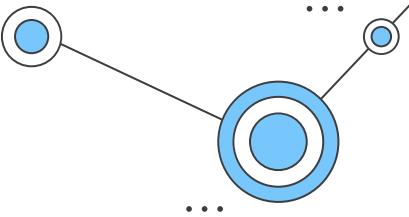


Arquitetura física no processo de desenvolvimento

- Em relação ao diagrama de implantação, sua construção tem início na fase de elaboração. Na fase de construção, os componentes são adicionados aos diversos nós.
 - Cada versão do sistema corresponde a uma versão do DI, que exibe os componentes utilizados na construção daquela versão.
 - O DI deve fazer parte dos manuais para instalação e operacionalização do sistema.
- Nem todo sistema necessita de diagramas de componentes ou diagramas de implantação.
 - Sistemas simples versus sistemas complexos.

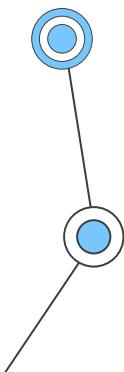
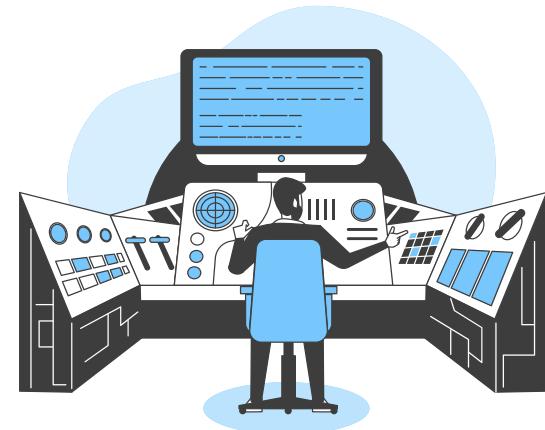


Arquitetura de Software - Modelagem



Arquitetura física

- Representa a disposição física do sistema de software pelo hardware disponível.
- A divisão de um sistema em camadas é independente da sua disposição física.
 - As camadas de software podem estar fisicamente localizadas em uma única máquina, ou podem estar distribuídas por diversos processadores.
 - Alternativamente, essas camadas podem estar distribuídas fisicamente em vários processadores. (Por exemplo, quando a camada da lógica do negócio é dividida em duas ou mais máquinas.)



Arquitetura de Software - Modelagem

Diagrama de Implantação e de Componente: Exemplo 1

- Definição das camadas em pacotes

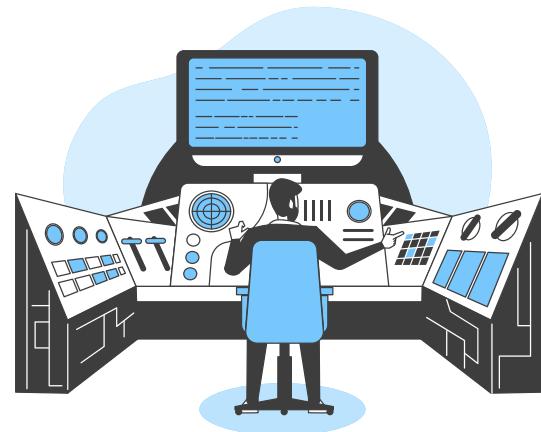


Diagrama de Implantação e de Componente: Exemplo 1

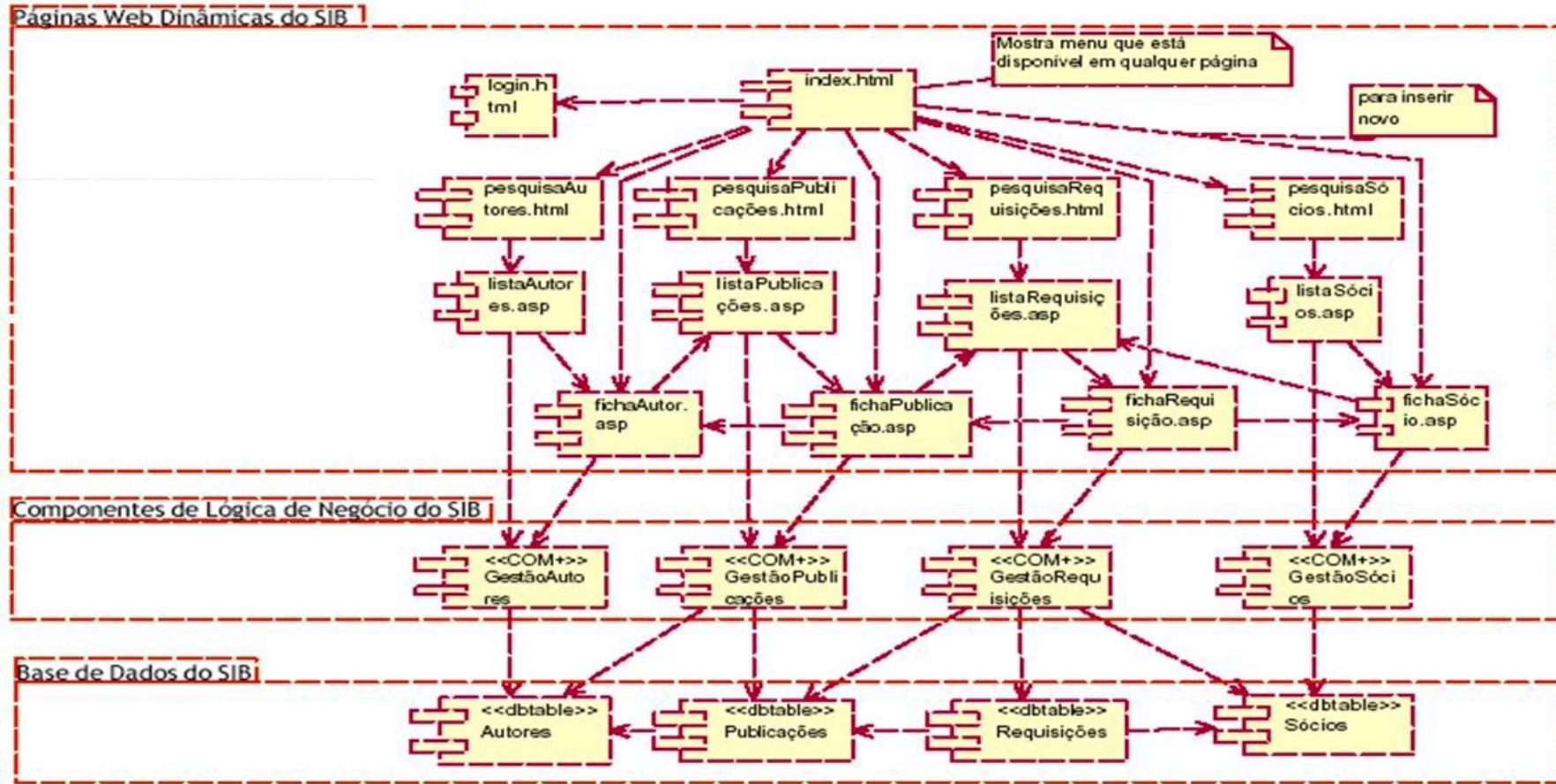


Diagrama de Implantação e de Componente: Exemplo 2

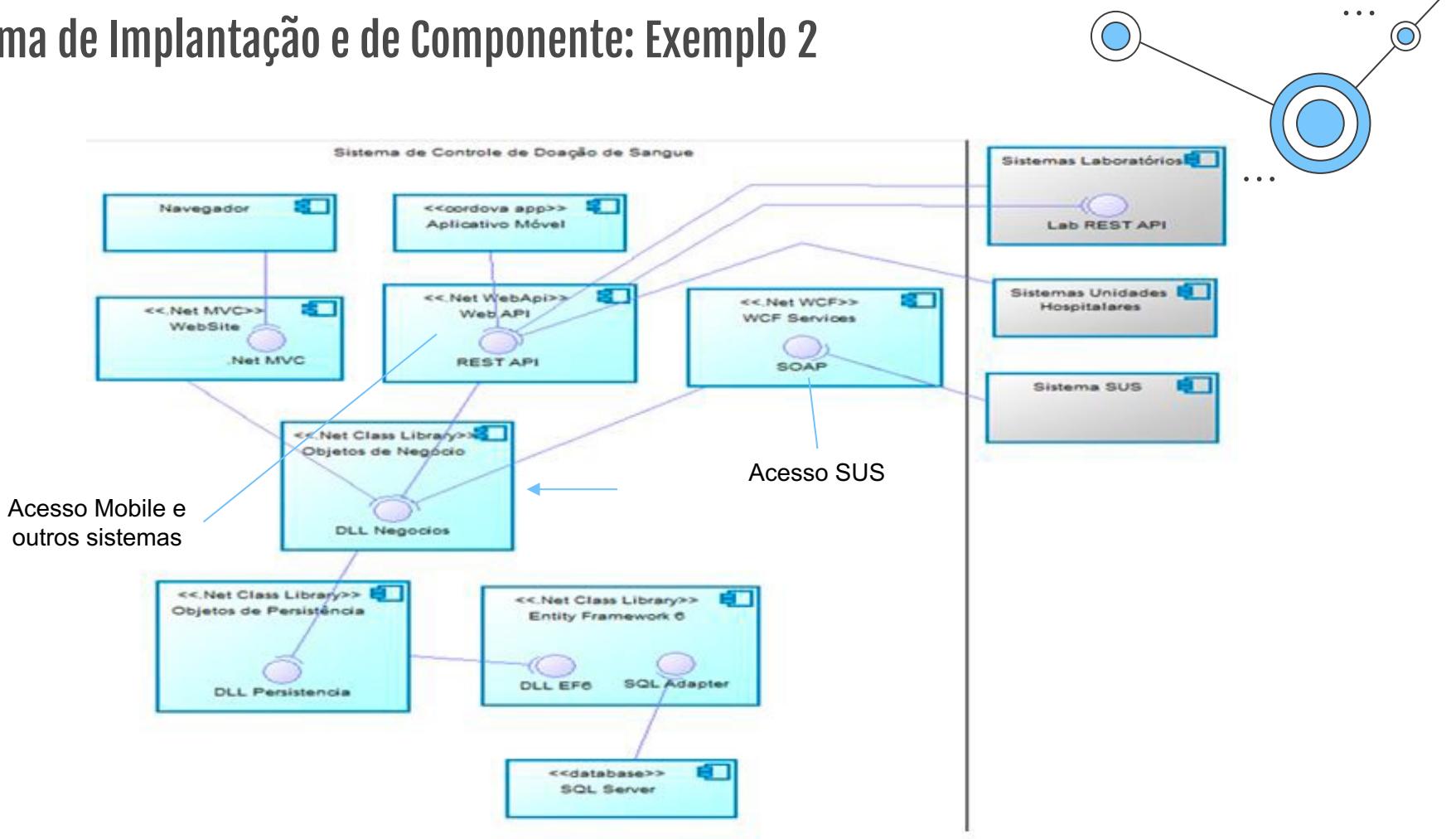


Diagrama de Implantação e de Componente: Exemplo 3

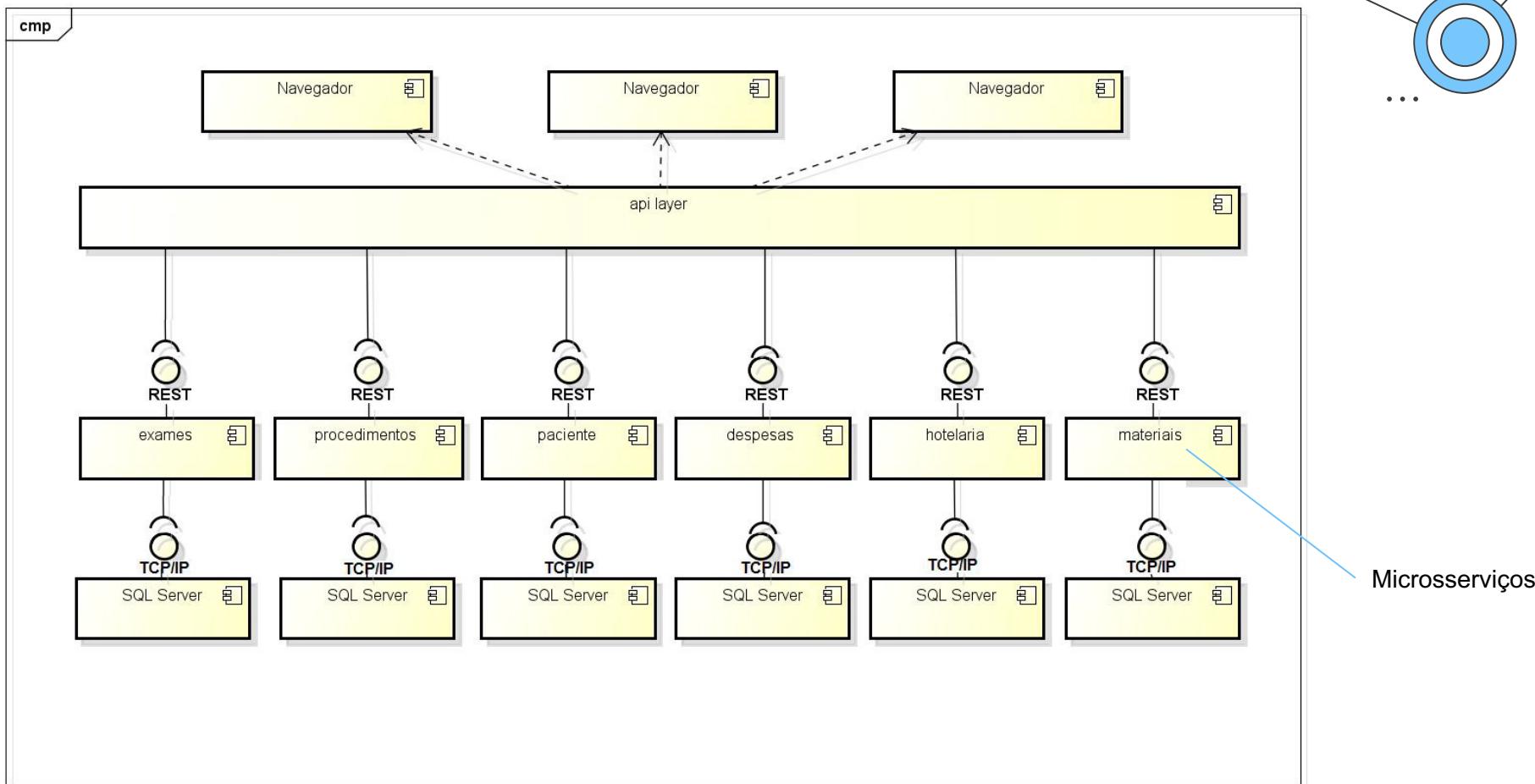


Diagrama de Implantação e de Componente: Exemplo 4

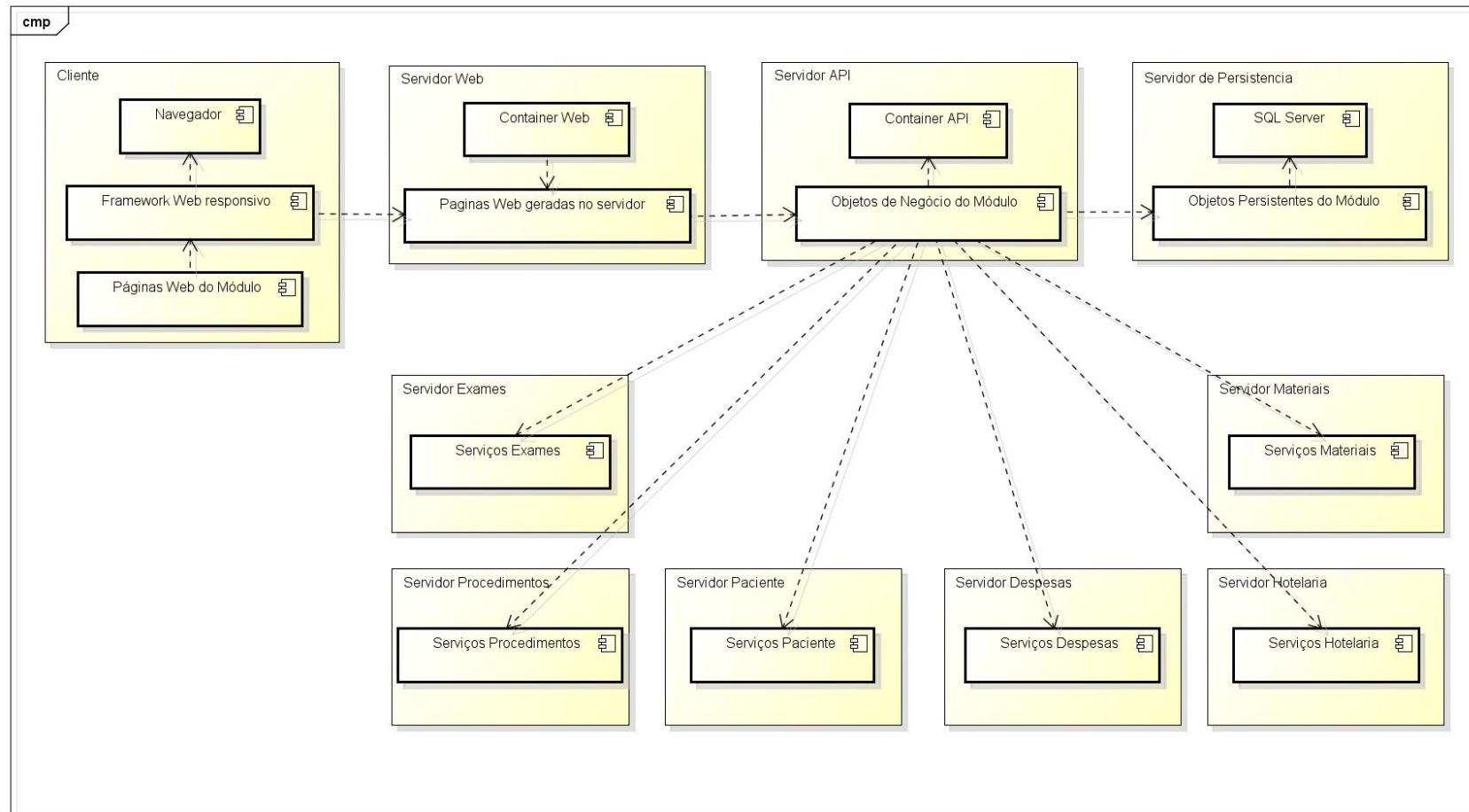


Diagrama de Implantação e de Componente: Exemplo 5

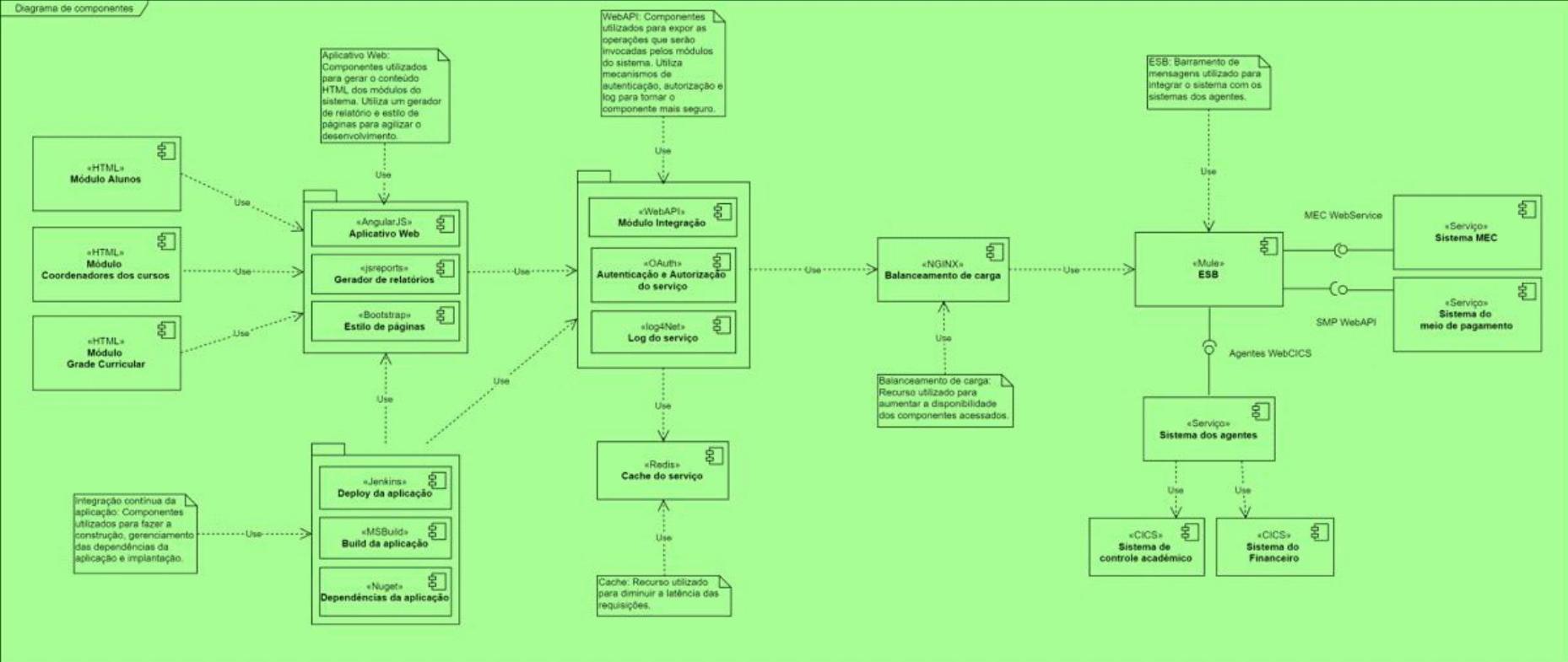


Diagrama de Implantação e de Componente: Exemplo 6

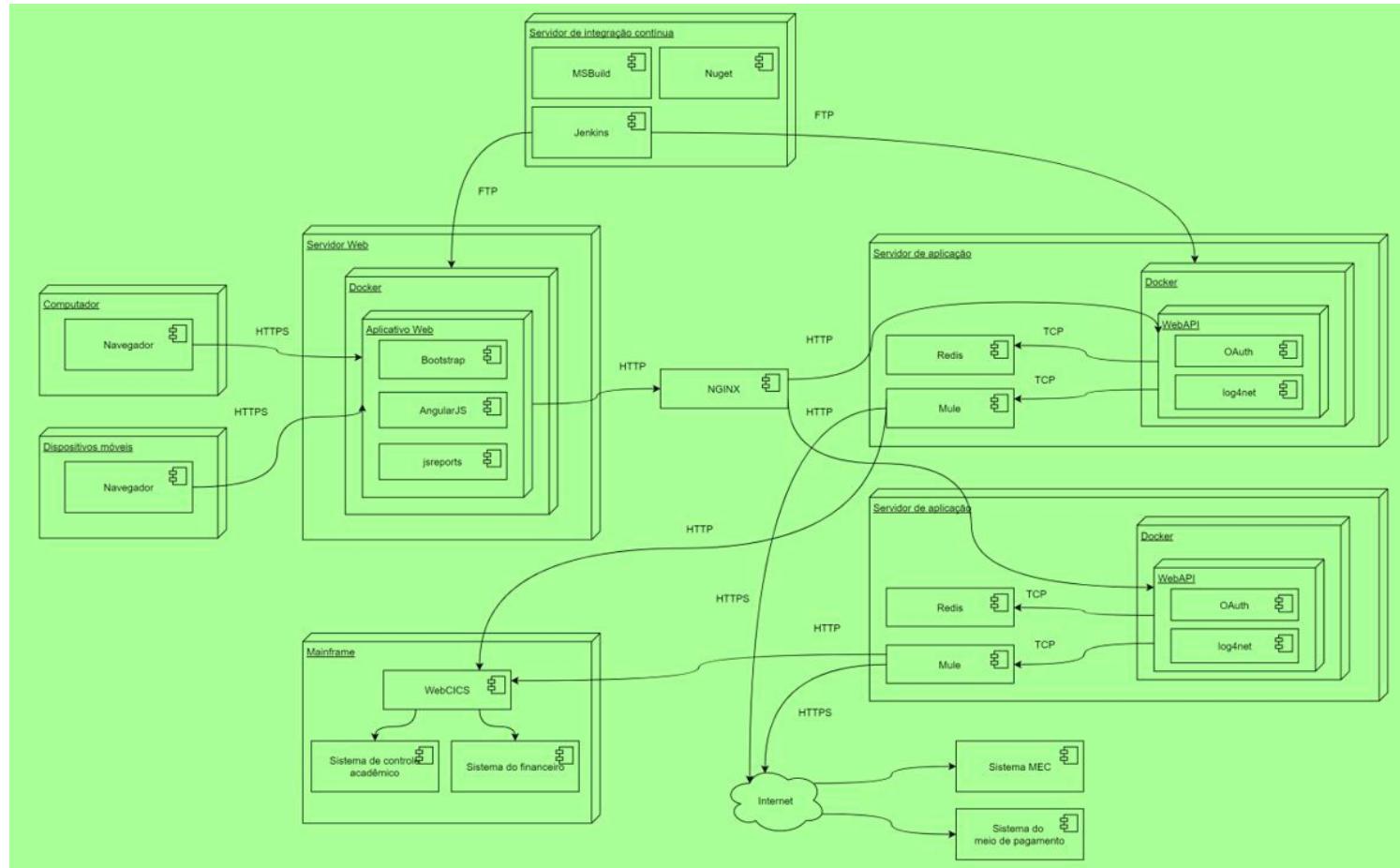


Diagrama de Implantação e de Componente: Exemplo 7

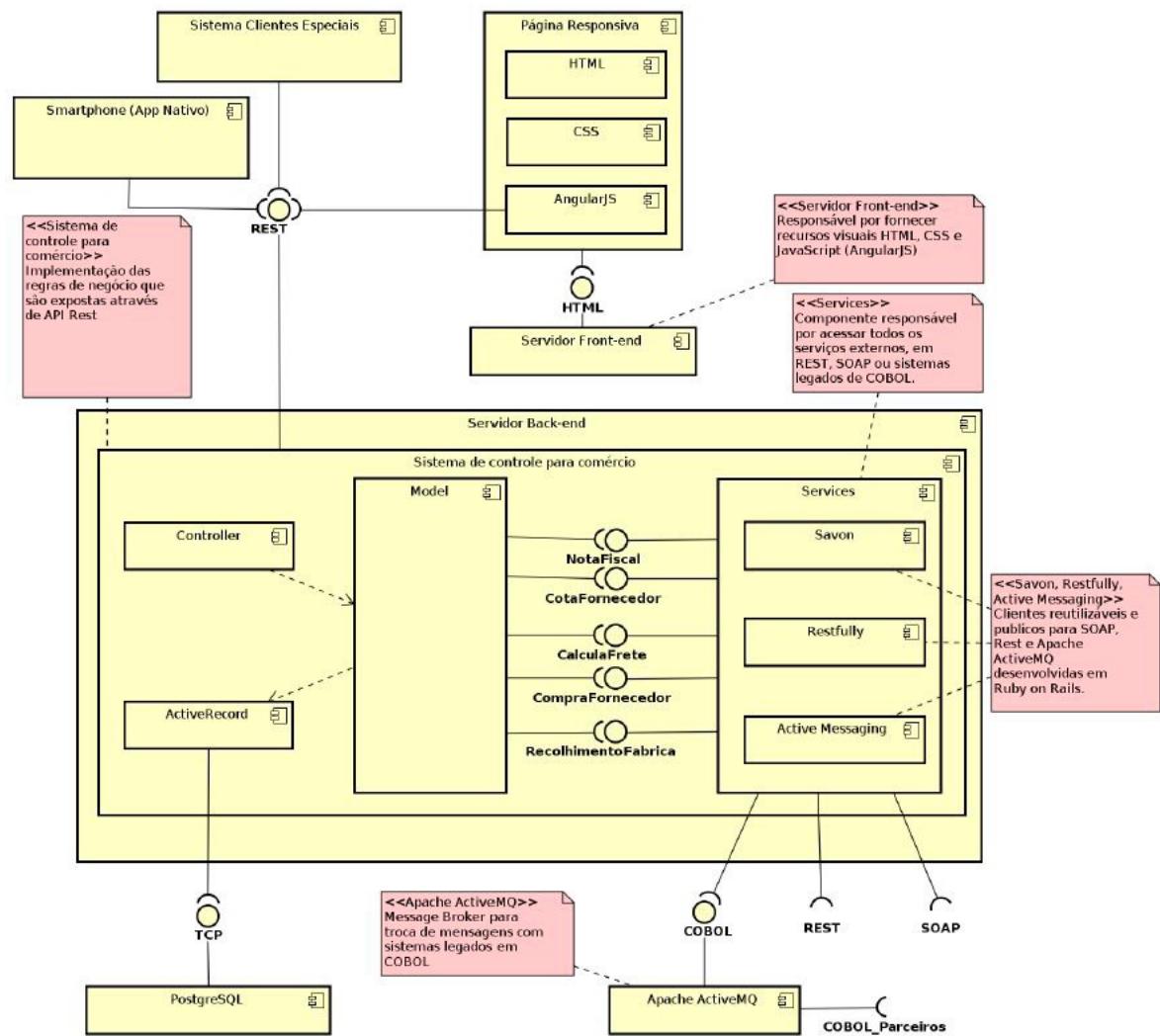
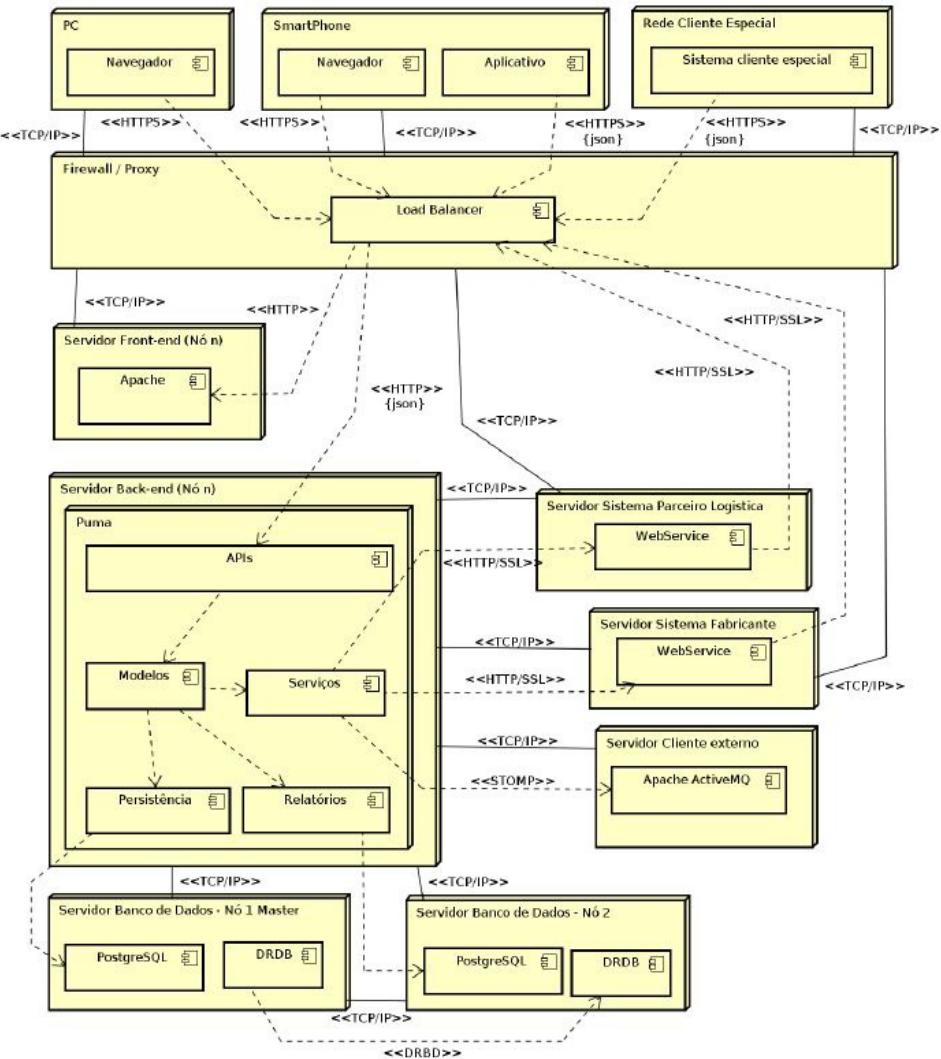


Diagrama de Implantação e de Componente: Exemplo 8

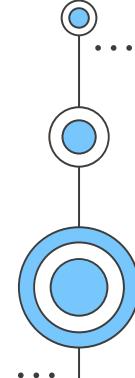




Projeto de Software

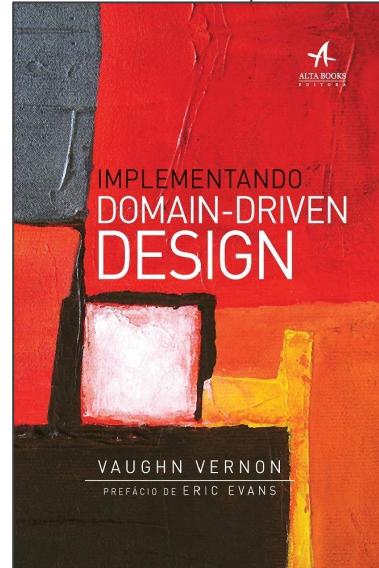
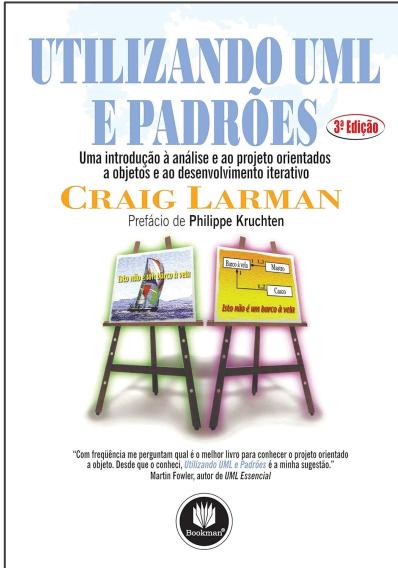
Referências básicas:

- **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:



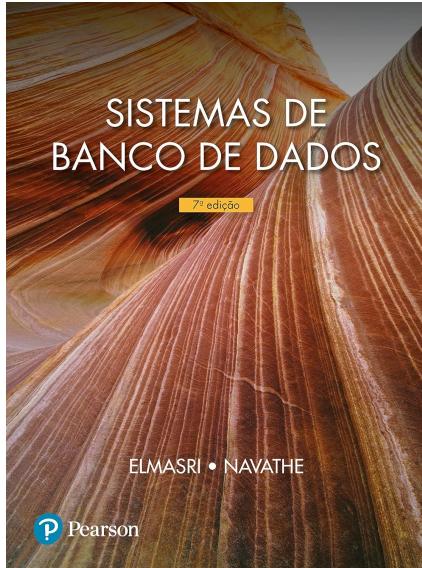
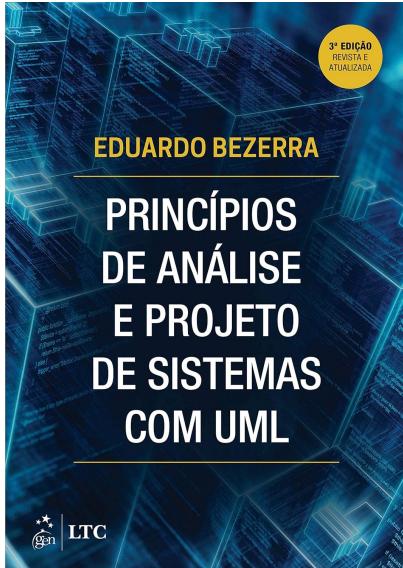
Projeto de Software

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**, 7^a 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).

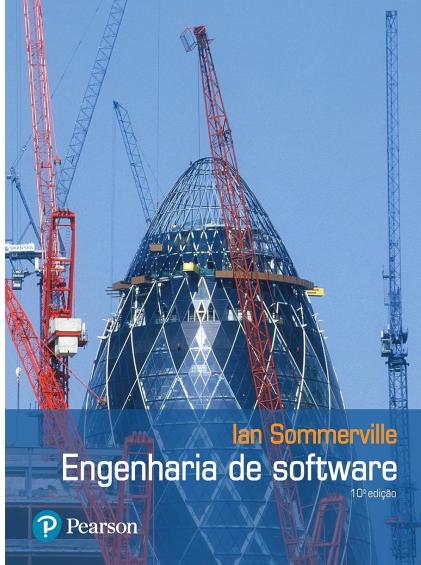
Projeto de Software

Referências complementares:



Projeto de Software

Referências complementares:



Obrigado!

Dúvidas?

joaopauloaramuni@gmail.com



[GitHub](#)



[LinkedIn](#)



[Lattes](#)

...

