SOFTWARE REUSE

Ian Sommerville, 8° edição – Capítulo 18

Aula de Luiz Eduardo Guarino de Vasconcelos

Objetivos

- Explicar os benefícios e alguns problemas do reuso de software
- Descrever diferentes tipos de componentes reutilizáveis e processos de reuso
- Introduzir famílias de aplicações como um caminho para o reuso
- Descrever os padrões do projeto como abstrações de alto nível que promovem o reuso

Tópicos abordados

- Desenvolvimento baseado no componente
- Famílias de aplicações
- Padrões de projeto

Reuso de Software

- Na maioria das disciplinas de engenharia, sistemas são projetados com base na composição de componentes existentes que foram utilizados em outros sistemas.
- A engenharia de software, até então, tinha como base o desenvolvimento tradicional. Porém tem sido reconhecido que, para alcançar software com mais qualidade, de forma mais rápida e com baixo custo, é necessário adotar um processo de desenvolvimento baseado na reutilização generalizada e sistemática de software.

Engenharia de Software baseada no reuso

- Reuso de sistemas de aplicações
  - Todo o sistema pode ser reutilizado pela sua incorporação, sem mudança, em outros sistemas (sistemas de prateleira) ou pelo desenvolvimento de famílias e aplicações.
- Reuso de Componentes
  - Componentes de uma aplicação que variam desde subsistemas até objetos isolados podem ser reutilizados.
- Reuso de funções
  - Componentes de software que

Prática do Reuso

- Reuso de sistemas de aplicações
  - Amplamente praticado quando sistemas de software são implementados como famíliasde aplicações. Reuso COTS está se tornando gradativamente mais comum
- Reuso de Componentes
  - Agora visto como elemento-chave para o reuso efetivo e amplo através da engenharia de software baseada em componentes. Entretanto, é ainda relativamente imaturo
- Reuso de funções
  - Comum em alguns domínios de aplicações (ex. engenharia) onde as bibliotecas especificas do domínio de funções reutilizáveis foram estabelecidas

Benefícios do Reuso

- Maior confiabilidade
  - Os componentes já foram experimentados e testados em sistemas que já estão funcionando
- Redução dos riscos de processo
  - Menos incertezas sobre os custos de desenvolvimento.
  - Uso efetivo de especialistas
  - Reuso de componentes ao invés de pessoas.
- Conformidade com padrões
  - Os padrões são embutidos ao se reutilizar componentes.
- Desenvolvimento acelerado
  - Evita o desenvolvimento e validação, acelerando a produção

Requisitos para o projeto com reuso

- Deve ser possível encontrar componentes reutilizáveis apropriados.
- O responsável pelo reuso dos componentes precisa ter certeza de que os componentes se comportarão como especificado e de que serão confiáveis.
- Os componentes devem ser bem documentados para ajudar o usuário a compreendê-los e adaptá-los a uma nova aplicação.

Problemas com reuso

- Aumento nos custos de manutenção
- Falta de ferramentas de apoio
- Síndrome do “não foi inventado aqui”
- Manutenção de uma biblioteca de componentes
- Encontrar e adaptar componentes reutilizáveis

Reuso baseado em geradores

- Geradores de programas envolve o reuso de padrões e algoritmos.
- Reuso baseado em geradores só é possível quando as abstrações de domínio e seu mapeamento em código executável podem ser definidos.
- Uma linguagem de domínio específico é usada para compor e controlar essas abstrações.

Tipos de geradores de programa

- Tipos de geradores de programas
  - Geradores de aplicações para processamento de dados corporativos.
  - Geradores de analisador gramatical para processamento de linguagem.
  - Geradores de código em ferramentas CASE
- O reuso com base em geradores tem uma boa relação custo-benefício, mas sua aplicabilidade está limitada a um número de domínios de aplicações relativamente pequeno.
- É mais fácil para os usuários finais desenvolver programas usando geradores, em comparação com outras abordagens de reuso.

Reuso com base em geradores

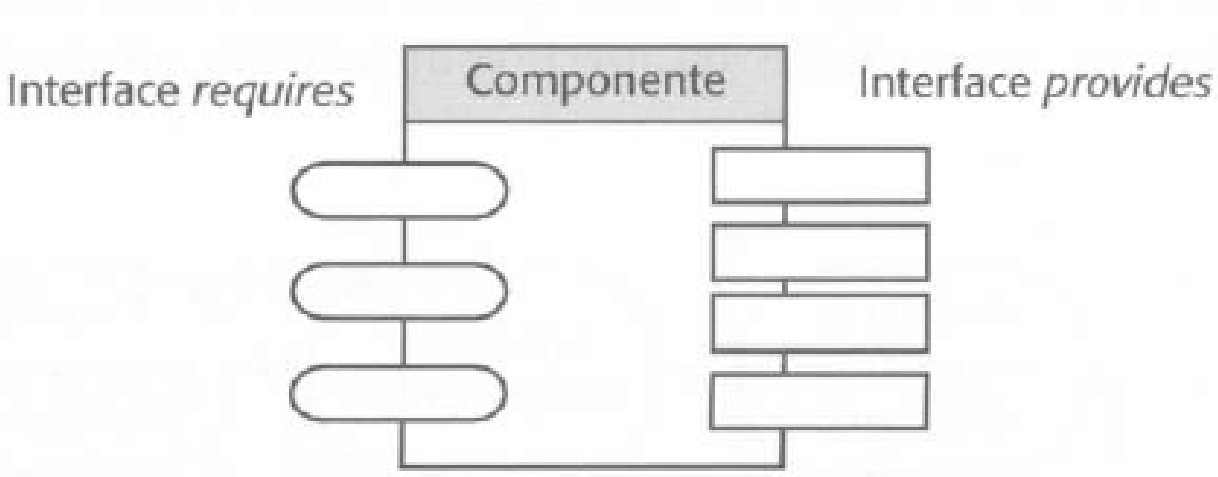
Desenvolvimento baseado em componentes

- A engenharia de software baseada em componentes (ESBC) é uma abordagem baseada no reuso para o desenvolvimento de software.
- Surgiu da frustração de que o desenvolvimento orientado a objetos não tinha conduzido ao um extensivo reuso. As classes de objetos individuais eram muito detalhadas e específicas.
- Componentes são mais abstratos do que classes de objetos e podem ser considerados provedores de serviços *stand alone*.

Componentes

- Componente é um provedor de serviços, sem se preocupar a respeito de onde esse componente está sendo executado ou com sua linguagem de programação
- Um componente é uma entidade executável independente que pode ser feito de um ou mais objetos executáveis.
- Os componentes publicam sua interface e todas as interações são feitas por meio dessa interface.
- Componentes podem ser simples como, por exemplo, uma função matemática ou um sistema maior tal como um componente de planilha de cálculo.

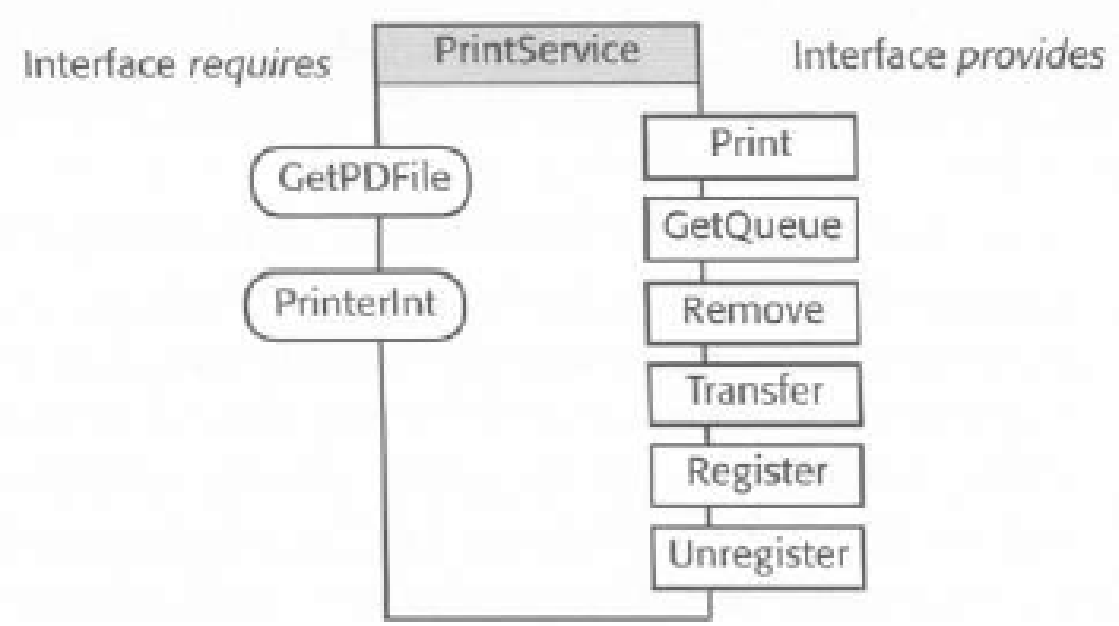
Interface de componentes



Interfaces de componentes

- Interface *Provides* (“*fornece*”)
  - Define os serviços fornecidos pelo componente a outros componentes
- Interface *Requires* (“*requer*”)
  - Define os serviços que especificam que serviços devem estar disponíveis para o componente funcionar como especificado

Um componente de serviços de impressão



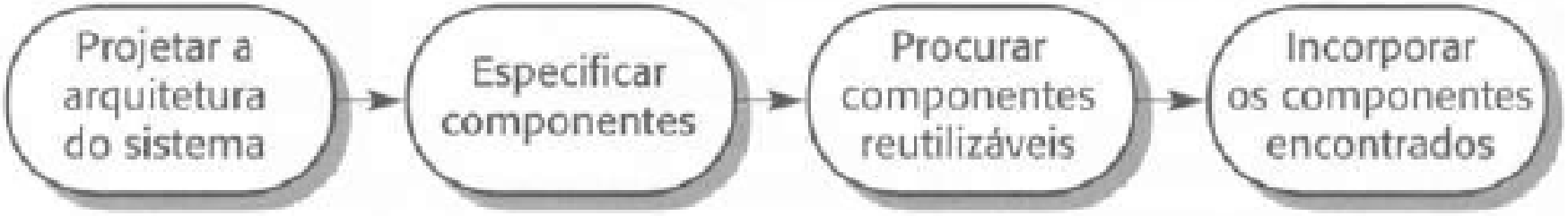
Abstrações de componentes

- *Abstração funcional*
  - O componente implementa uma única função, como uma função matemática.
- *Agrupamentos Casuais*
  - O componente é uma coleção de entidades fracamente relacionadas que podem ser declarações de dados, funções, etc.
- *Abstrações de dados*
  - O componente representa uma abstração de dados ou classe em uma linguagem orientada a objetos
- *Abstrações em Clusters*
  - O componente é um grupo de classes relacionadas que trabalham em conjunto
- *Abstração de sistema*
  - O componente é um sistema inteiramente auto contido.

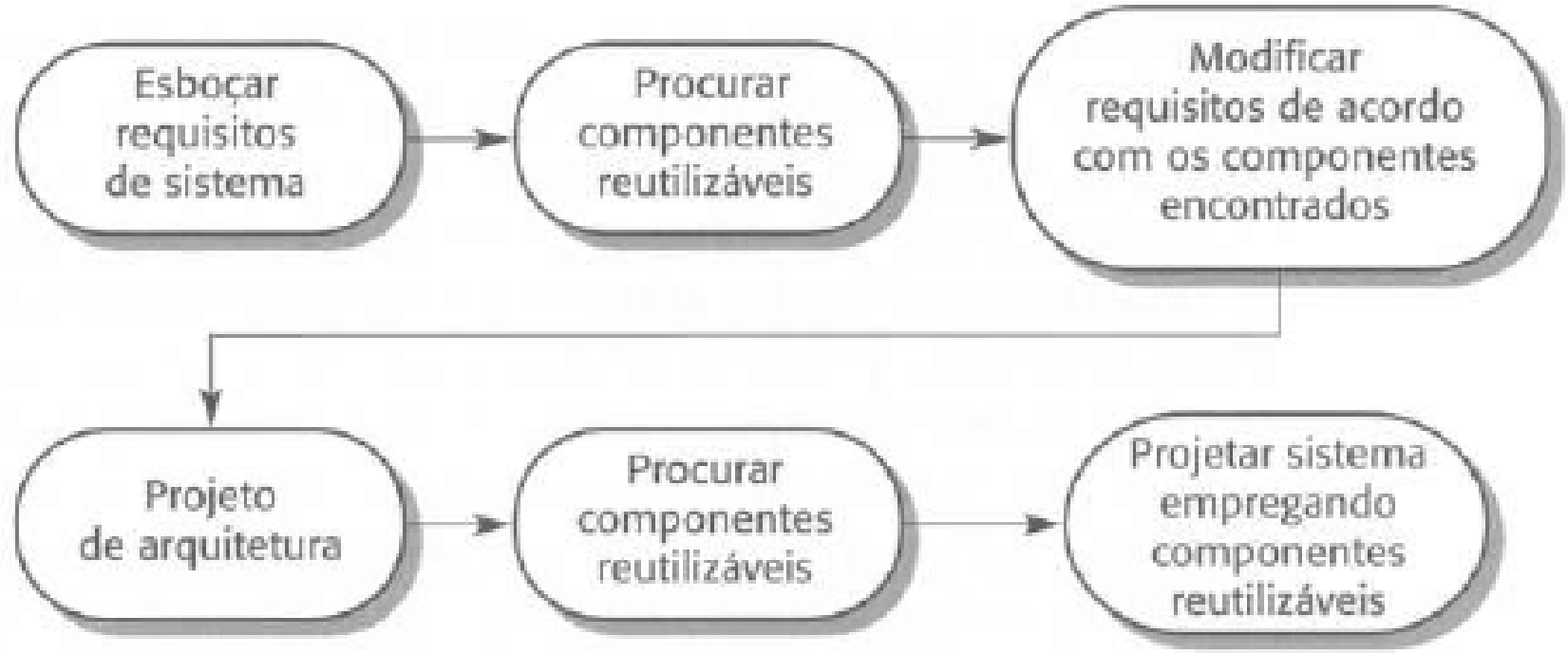
Processos ESBC

- **Desenvolvimento baseado em componentes pode** ser integrado em um **processo de software padrão,** incorporando atividades de **reuso no processo.**
- Contudo, em **desenvolvimento dirigido pelo reuso, os requisitos do sistema são modificados para refletir os** componentes que estão disponíveis.
- ESBC usualmente envolve um processo de prototipação ou um processo de desenvolvimento incremental e uma linguagem de *scripting é utilizada para* ***integrar*** componentes reutilizáveis.

Processo de reuso 'oportunista'



Desenvolvimento com reuso



Problemas com ESBC

- Incompatibilidades de componentes podem significar que as economias de custos e o calendário são menores que o esperado
- Encontrar e entender componentes.
- Gerenciar a evolução a medida que os requisitos se modificam em situações nas quais é impossível alterar os componentes do sistema.

Frameworks de aplicações

- *Frameworks são um projeto de subsistema* constituído de um conjunto de classes abstratas e concretas e da interface entre elas.
- Os subsistemas são implementados com o acréscimo de componentes e o fornecimento da implementação concreta das classes abstratas nos *frameworks*.
- *Frameworks são entidades moderadamente* grandes que podem ser reutilizadas.

Classes de Frameworks

- Frameworks de infra-estrutura de sistema
  - Permite o desenvolvimento das infra-estruturas de sistema, tais como comunicações, interfaces com o usuário e
  - Compiladores.
- *Frameworks de integração com middleware*
  - Padrões e classes que aceitam a comunicação de componentes e a troca de informações. (CORBA, COM, DCOM, Java Beans)
- *Frameworks de aplicações corporativos*
  - Suportam o desenvolvimento de tipos específicos de aplicações tais como telecomunicações e sistemas financeiros.

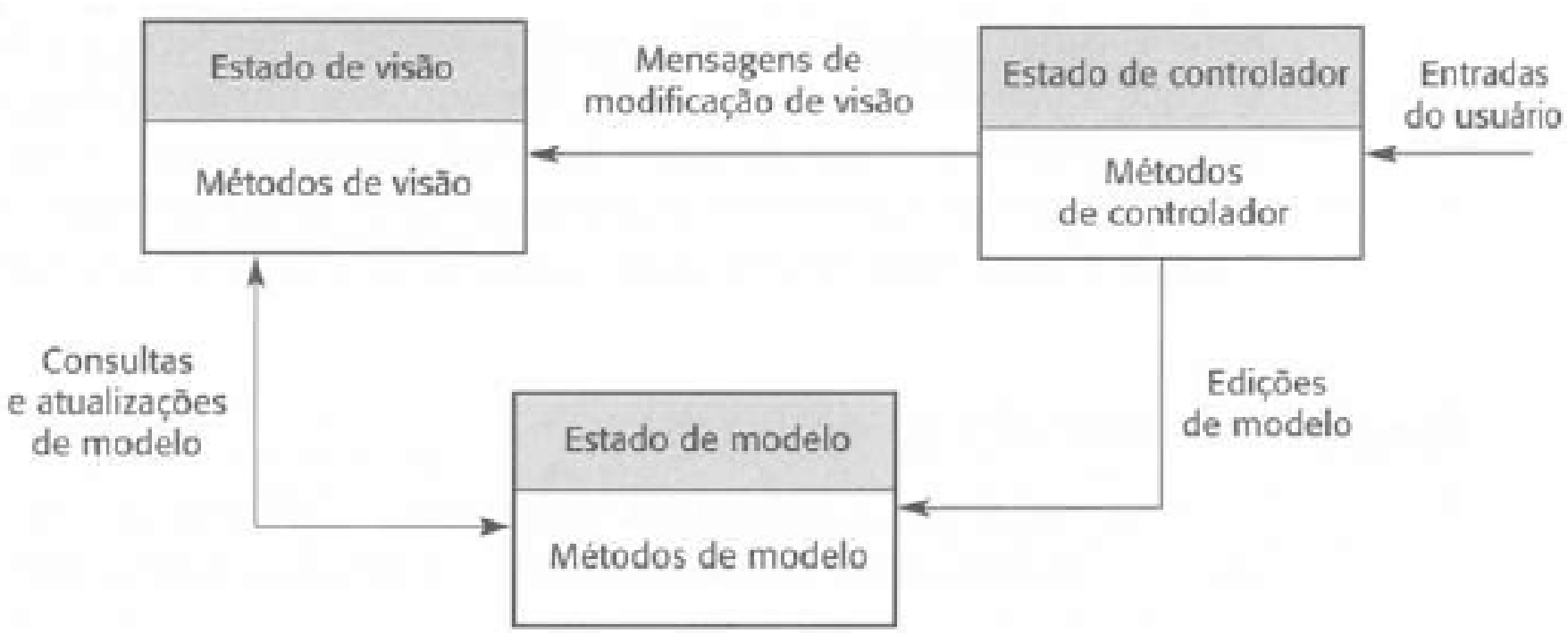
Ampliando frameworks

- *Frameworks são genéricos e podem ser ampliados para* criar um subsistema ou uma aplicação mais específica.
- Ampliação de um *framework envolve:*
  - O acréscimo de classes concretas que herdam operações de classes abstratas no framework.
  - O acréscimo de métodos que são chamados em resposta aos eventos que são reconhecidos pelo *framework*.
- O principal problema com frameworks é a sua complexidade e o tempo que leva para aprender a utilizá-los.

Controlador da visualização do modelo (MVC)

- Framework de infraestrutura do sistema para projeto GUI
- Permitir múltiplas presentações de um objeto e separar interações de suas representações
- Framework MVC envolve a instantização de um número de padrões

MVC (Modelo-Visão-Controlador)



Reuso de produtos COTS

- COTS - Commercial Off-The-Shelf systems (produtos de prateleira produzido por terceiros)
- Sistemas COTS são aplicações completas que oferecem uma API (Application Programming Interface)
- A construção de sistemas, pela integração de sistemas COTS, é uma estratégia de desenvolvimento viável para alguns tipos de sistemas, tais como sistemas de comércio eletrônico, telecomunicações, governo.

Problemas com integração de sistemas COTS

- Falta de controle sobre a funcionalidade e o desempenho
  - Sistemas COTS podem ser menos efetivo do que parecem.
- Problemas com a interoperabilidade de sistemas COTS
  - Cada produto inclui diferentes suposições de como será utilizado, tornando a integração difícil.
- Nenhum controle sobre a evolução de sistema
  - O suporte COTS, não os usuários do sistema, controla a evolução.
- Suporte técnico dos fabricantes de COTS
  - O suporte pode variar muito e é dependente do fabricante.

Desenvolvimento de componentes para reuso

- Características de componentes que permitem a reutilização:
  - Deve refletir abstrações estáveis do domínio do problema.
  - Deve ocultar a maneira como seu estado é representado.
  - Deve ser tão independente quanto possível,
  - Todas as exceções devem ser parte da interface do componente.
- Deve haver uma conciliação entre a facilidade de reutilização e a facilidade de uso de um componente.
  - Interface mais genérica aumenta a possibilidade de reutilização de um componente, porém resulta em uma interface mais complexa e,

Componentes reutilizáveis

- O custo de desenvolvimento de componentes para a reutilização é maior que o custo de equivalentes específicos. Esse custo extra para a reutilização não deve ser considerado como um custo de projeto e sim um custo organizacional.
- Componentes genéricos podem ser menos eficientes, com relação ao tamanho e tempo de execução, do que os seus correspondentes específicos.

Aumento do reuso

- Generalização do nome
  - Nomes em um componente podem ser modificados para que não reflitam diretamente uma entidade específica de uma aplicação
- Generalização da operação
  - Operações podem ser adicionadas para dar funcionalidade extra. Operações específicas da aplicação podem ser removidas
- Generalização da exceção
  - As exceções específicas da aplicação são removidas, e gerenciadores de exceções são incluídos para aumentar a robustez do componente
- Certificação do componente
  - Componene é certificado como reutilizável

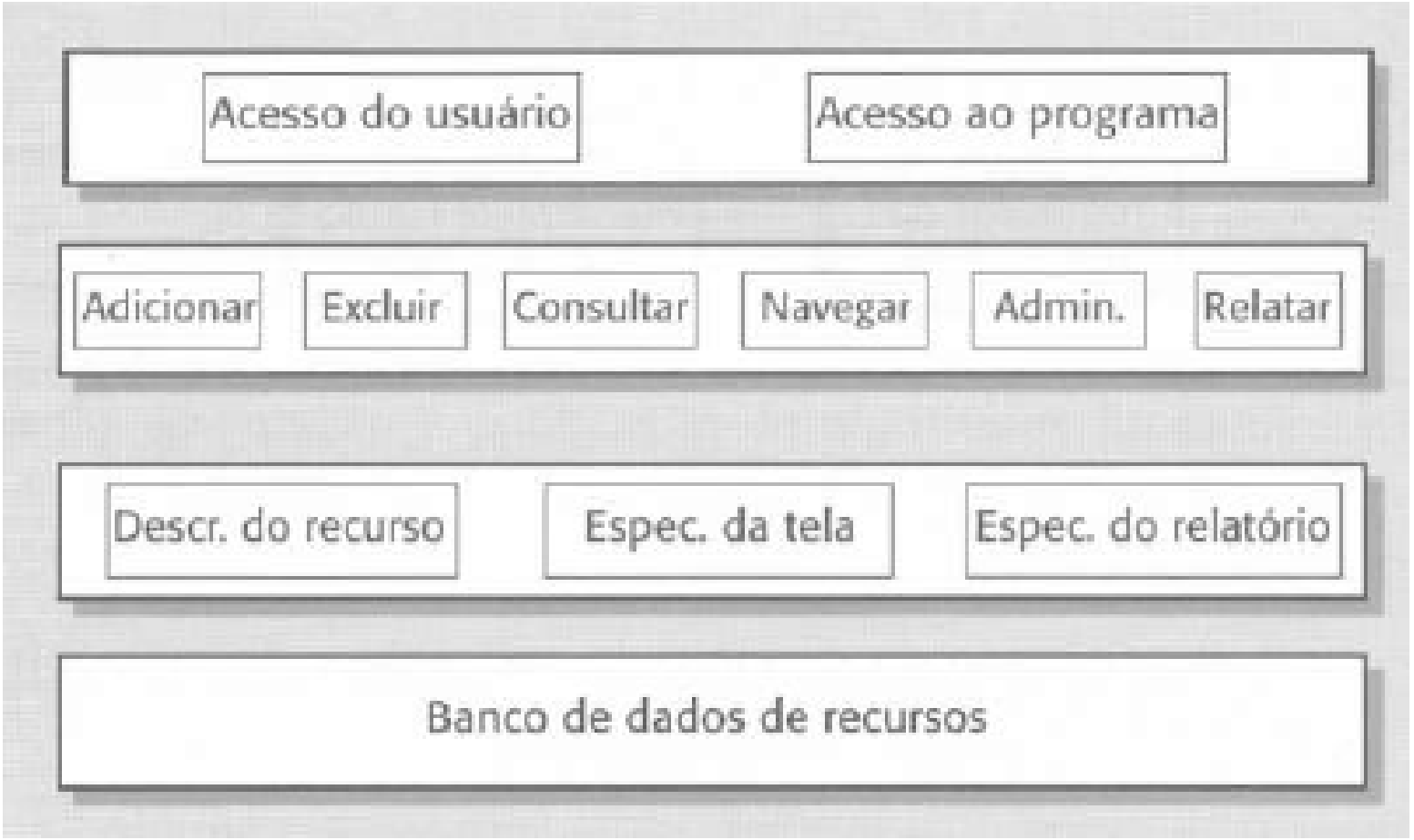
Famílias de aplicações

- Uma família de aplicações, ou linha de produto, é um conjunto de aplicações relacionadas que tem uma arquitetura de domínio específico em comum.
- O núcleo em comum da família de aplicações é reutilizado cada vez que uma nova aplicação é requerida.
- Cada aplicação específica é especializada de alguma maneira.

Especialização de uma família de aplicação

- Especialização de plataforma
  - Diferentes versões da aplicação são desenvolvidas para diferentes plataformas.
- Especialização da configuração
  - Diferentes versões da aplicação são criadas para lidar com diferentes dispositivos periféricos.
- Especialização funcional
  - Diferentes versões da aplicação são criadas para clientes com diferentes requisitos.

Um sistema genérico de gerenciamento de recursos



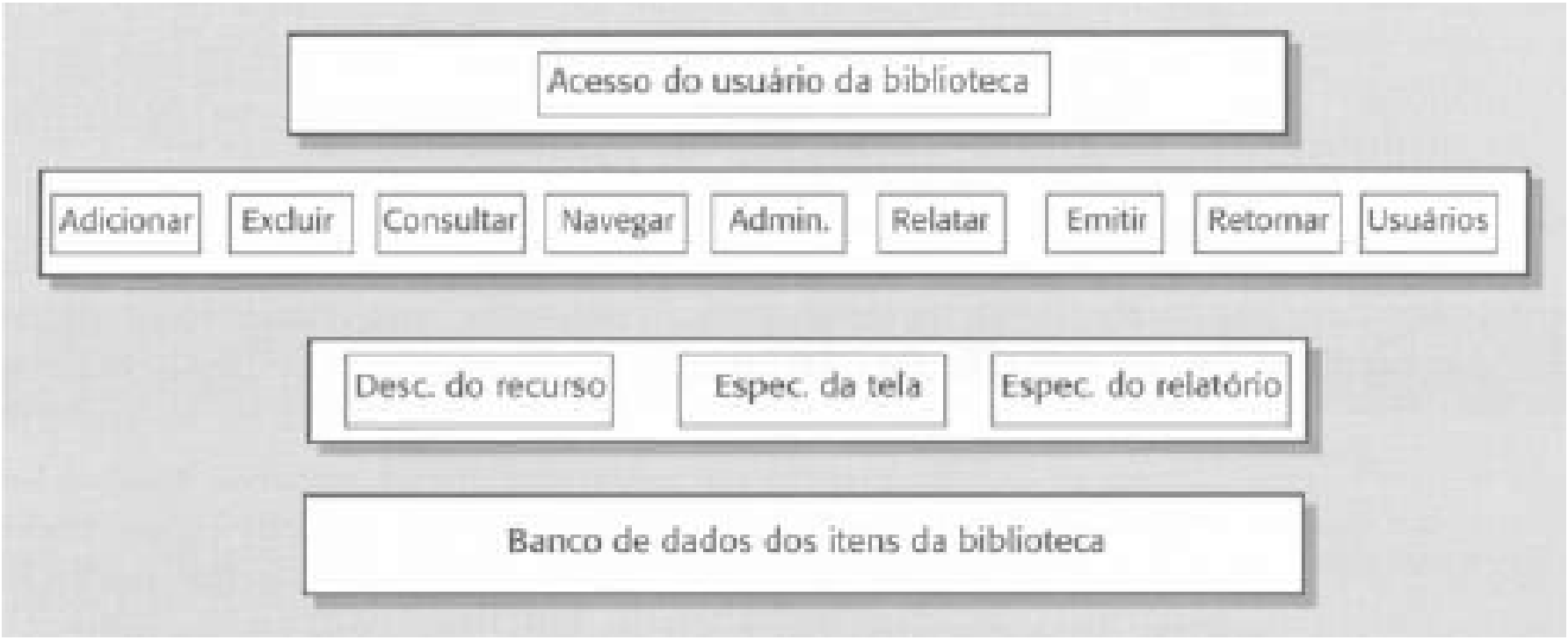
Sistema de gerenciamento de inventário

- Base de dados do recurso
  - Mantém os detalhes das coisas que estão sendo gerenciadas
- Descrições de E/S
  - Descreve as estruturas na base de dados do recurso e formatos de entrada e saída que são usados.
- Nível consulta
  - Fornece funções que implementam consultas sobre os recursos.
- Interfaces de Acesso
  - Uma interface do usuário e uma interface para programar a aplicação.

Arquiteturas de uma família de aplicações

- As arquiteturas devem ser estruturadas de forma a separar sub-sistemas diferentes e permitir sua modificação
- A arquitetura deve também separar entidades e suas descrições e os níveis mais altos das entidades de acesso do sistema através da descrição ao invés de diretamente

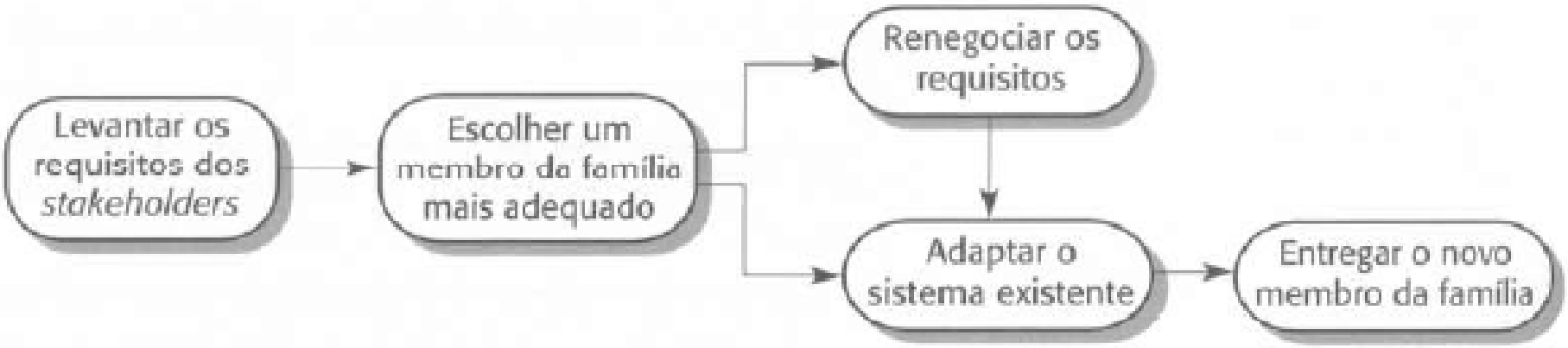
Um sistema de biblioteca



Sistema de biblioteca

- Os recursos gerenciados pelo sistema são os livros na biblioteca.
- Para essa aplicação, deve ser acrescentada novas funcionalidades (Emprestar e devolver recursos e permitir que os usuários sejam registrados no sistema).

Desenvolvimento de um membro da família



Desenvolvimento de um membro da família

- Elicitar os requisitos stakeholder
  - Usar membros existentes como protótipos
- Escolher o membro que melhor se encaixa
  - Encontrar o membro que preenchhe melhor os requisitos
- Renegociar requisitos
  - Adaptar os requisitos cmo for necessário para as capacidades do software
- Adaptar sistema existente
  - Desenvolver novos módulos e fazer mudança nos membros da família
- Entregar novo membro da família
  - Documentar as características principais para futuro desenvolvimento
- do membro

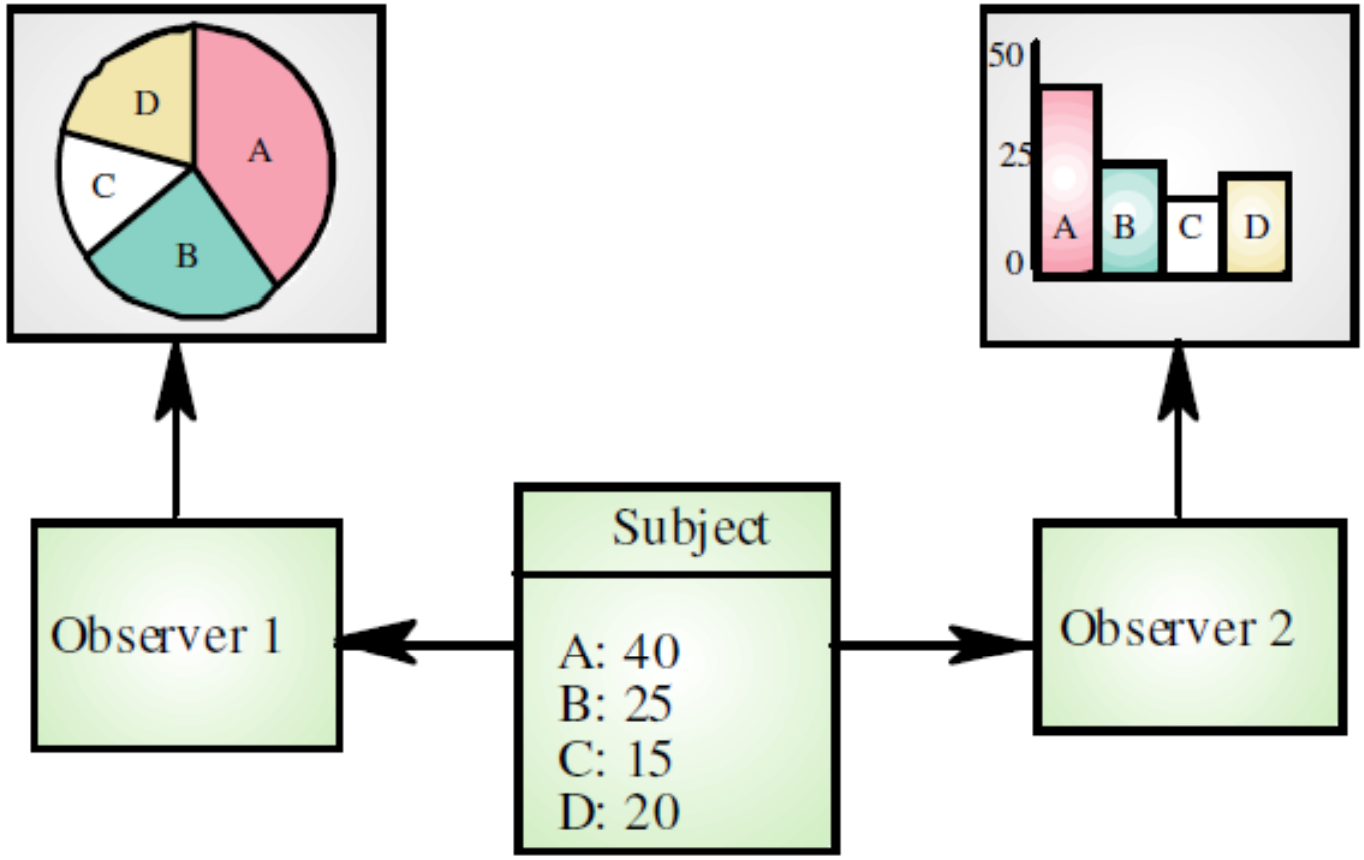
Padrões de Projeto (Design Pattern)

- Um padrão de projeto é uma forma de reutilizar o conhecimento abstrato sobre um problema e sua solução.
- O padrão é uma descrição do problema e a essência de sua solução.
- A descrição deve ser suficientemente abstrata de modo que possa ser reutilizada em diferentes casos.
- Padrões de projeto têm estado associados com o projeto orientado a objetos.

Elementos dos padrões

- Nome
  - Referência significativa ao padrão
- Descrição do problema
- Descrição da solução
  - Não é uma descrição de projeto concreta, é um *template para* uma solução que pode ser instanciada de diferentes maneiras.
- Conseqüências
  - Os resultados e as conciliações da aplicação do padrão.

Exemplo de Múltiplos Displays



Padrão Observer

- Nome
  - Observador
- Descrição
  - Separa o display do estado do objeto do objeto em si
- Descrição do problema
  - Usado quando vários displays de estado são necessários
- Descrição da solução
  - Ver slide com descrição UML
- Conseqüências
  - Optimizações para aumentar a performance do display não são práticas

Pontos-chave

- Projeto com reuso envolve projetar software com base em bons projetos e componentes existentes.
- Vantagens são custos mais baixos, desenvolvimento mais rápido de software e menores riscos.
- Engenharia de software baseada em componentes leva em conta componentes de *caixa preta, com requisitos e interfaces bem definidos.*
- O reuso de produtos COTS se ocupa do reuso de *sistemas de prateleira.*

Pontos-chave

- Os componentes de software para reuso devem ser independentes, refletir abstrações estáveis de domínio, fornecer acesso a estado por meio de operações de interface e não devem manipular exceções.
- Famílias de aplicações são aplicações relacionadas, que são desenvolvidas a partir de uma ou mais aplicações básicas.
- Padrões de projeto são abstrações de alto nível, que documentam soluções bem-sucedidas de projeto.