

Aula 15

Professores:

Cláudia Maria Lima Werner
Leonardo Gresta Paulino Murta

Componentes

Conteúdo:

- Ativos reutilizáveis x Componentes
- Interfaces
- DBC x OO
- Reutilização no DBC
- Customização de Componentes
- Restrições no DBC

Ativos Reutilizáveis x Componentes

→ Em 1969

- McIlRoy já vislumbra uma indústria de componentes de software reutilizáveis para domínios específicos
- Componentes = rotinas de código
- Domínios = infra-estrutura
 - rotinas de aproximação numérica
 - conversão de entrada/saída
 - geometria 2D e 3D
 - processamento de texto
 - persistência

Ativos Reutilizáveis x Componentes



Hoje

- Ativos reutilizáveis = quaisquer artefatos passíveis de reutilização
- Componentes
 - Unidade de composição auto-contida (encapsulamento)
 - Interfaces bem definidas (contrato)
 - Grau de maturidade e documentação (qualidade)
 - Especificado em diferentes níveis (abstração)
 - Obedece a restrições (arquitetura/plataforma)
- Domínios
 - Infra-estrutura (reutilização horizontal)
 - Negócio (reutilização vertical)

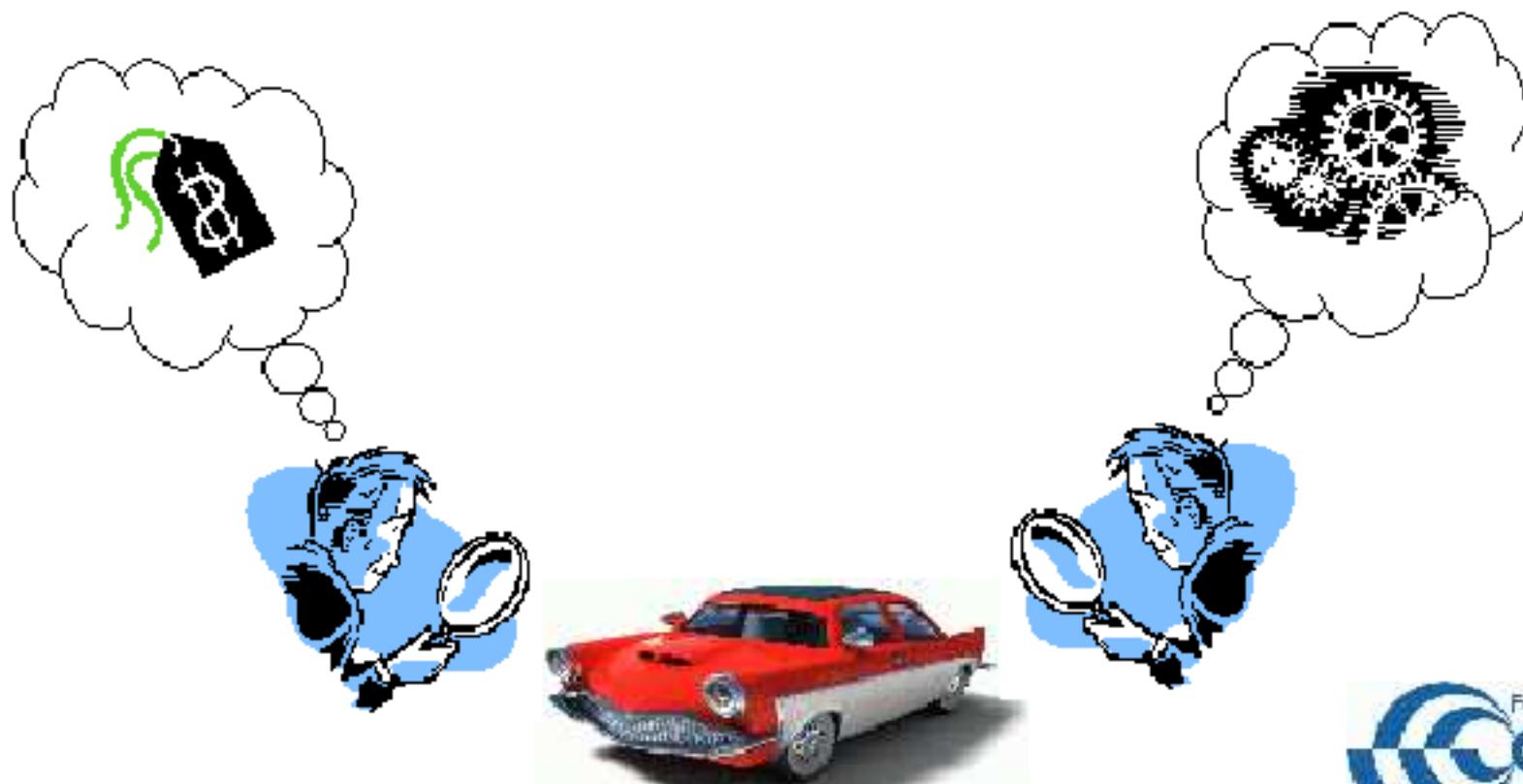
Interfaces

- ➔ Um conjunto de assinaturas de operações que podem ser invocadas por um cliente
 - ➡ Cada operação deve ter sua semântica especificada (Szyperski, 1998)
- ➔ Determinam como o componente pode ser reutilizado e interconectado com outros componentes
- ➔ Um componente exporta (provê) serviços que outro componente importa (requer)
- ➔ A conexão entre os componentes (em especial os distribuídos/heterogêneos) é usualmente intermediada pela infra-estrutura

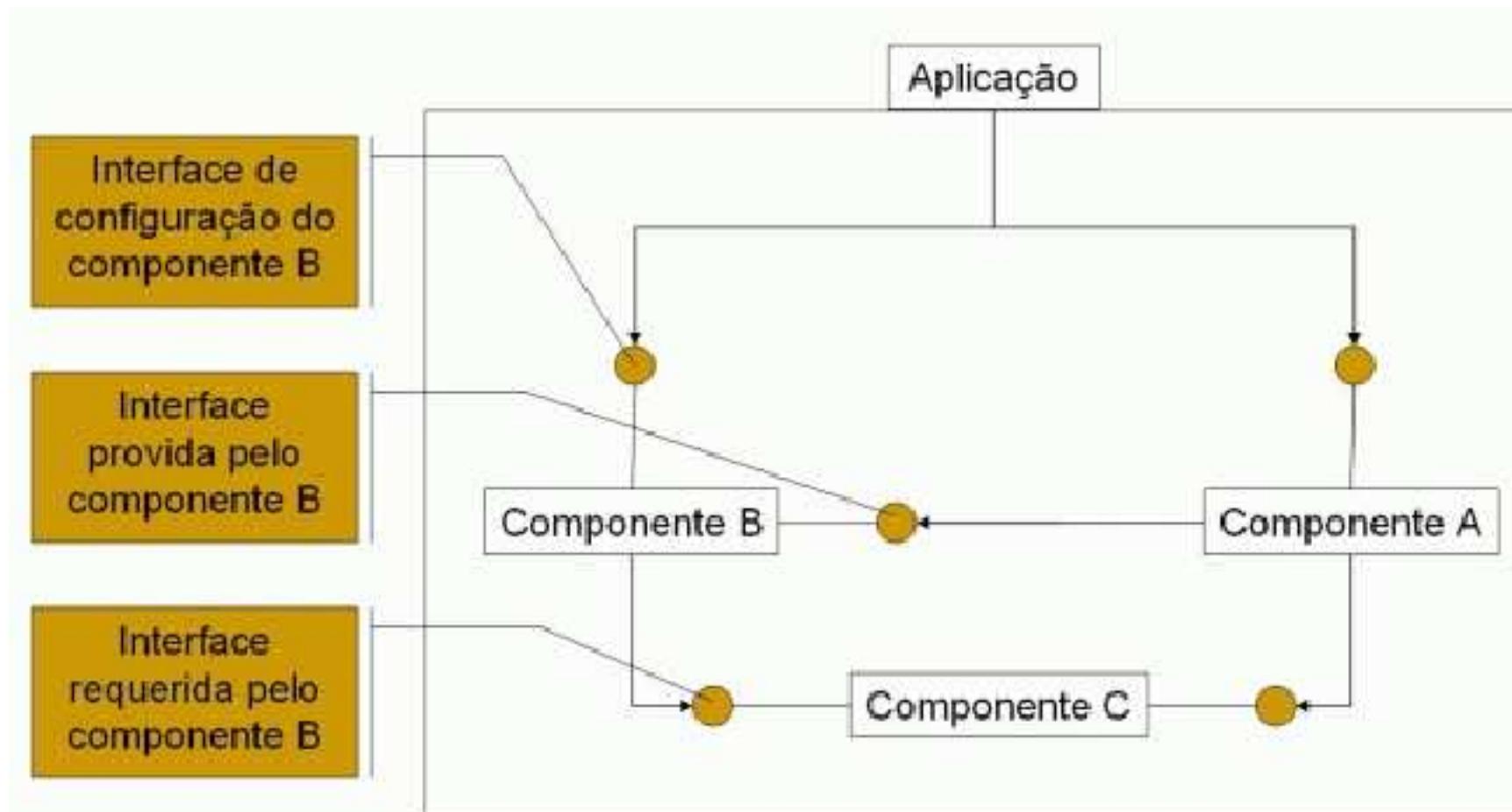
Interfaces

→ Diferentes interfaces podem ser utilizadas para diferentes propósitos

- Interfaces de negócio (providas e requeridas)
- Interfaces de configuração



Interfaces



Interfaces



Interfaces muito específicas

- Podem dificultar a reutilização
- Poucos cenários se adequarão à interface
- Ex.: Aplicação se conecta ao Google via Web Services
- Ex.: Conexão via RMI

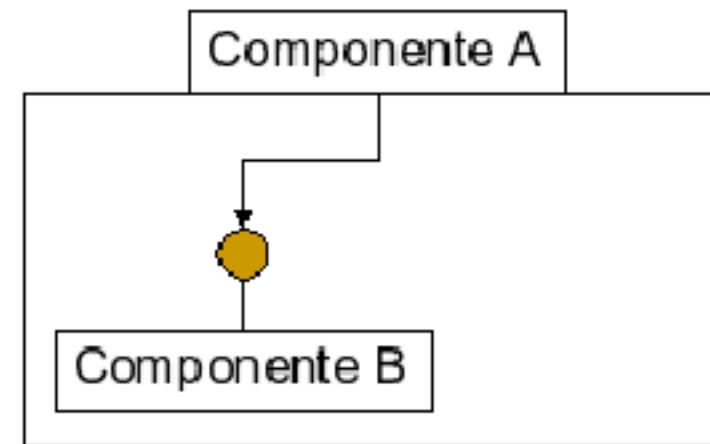
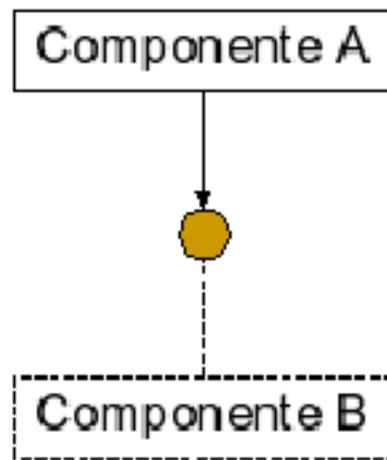


Interfaces muito genéricas

- Componentes se conectam com facilidade
- Podem sobrecarregar o protocolo de comunicação entre os componentes
- Ex.: Aplicação se conecta a BD via interface ODBC (protocolo SQL)
- Ex.: Conexão via Sockets
- Analogia: interfaces de hardware quase sempre são genéricas (ex.: PCI, AGP, USB, etc.)

Interfaces

- O relacionamento entre componentes pode se dar via
- Interfaces requeridas
 - Sub-componentes



Interfaces



Relacionamento via interface requerida

- Componente A precisa de algum componente com determinadas funcionalidades
- A implantação do Componente A só terá sucesso caso algum componente forneça essas funcionalidades
- O usuário do Componente A pode escolher qual é a implementação desejada para prover essas funcionalidades
- Ex.: Aplicações que dependem de um BD
- Analogia: Placa mãe sem vídeo on-board

Interfaces



Relacionamento via sub-componente

- Componente A engloba o Componente B
- A implantação do Componente A é auto-contida (inclui o componente B)
- Somente o desenvolvedor do Componente A poderá substituir o Componente B por outros componentes
- Ex.: Aplicações que incluem um BD (JBoss, Subversion, etc.)
- Analogia: Placa mãe com vídeo on-board

DBC x OO

→ Reutilização em OO

- Padrões
- Frameworks
- Objetos(?)

→ Reutilização de objetos é difícil

- Granularidade fina dos objetos
- Objeto colaboram (não são auto-contidos)
- Herança (ex. em Java, todos herdam de Object)

→ Métodos em OO não focam na reutilização de
objetos

- Os modelos são criados do zero
- Não existem atividades explícitas para a busca por
elementos reutilizáveis

DBC x OO

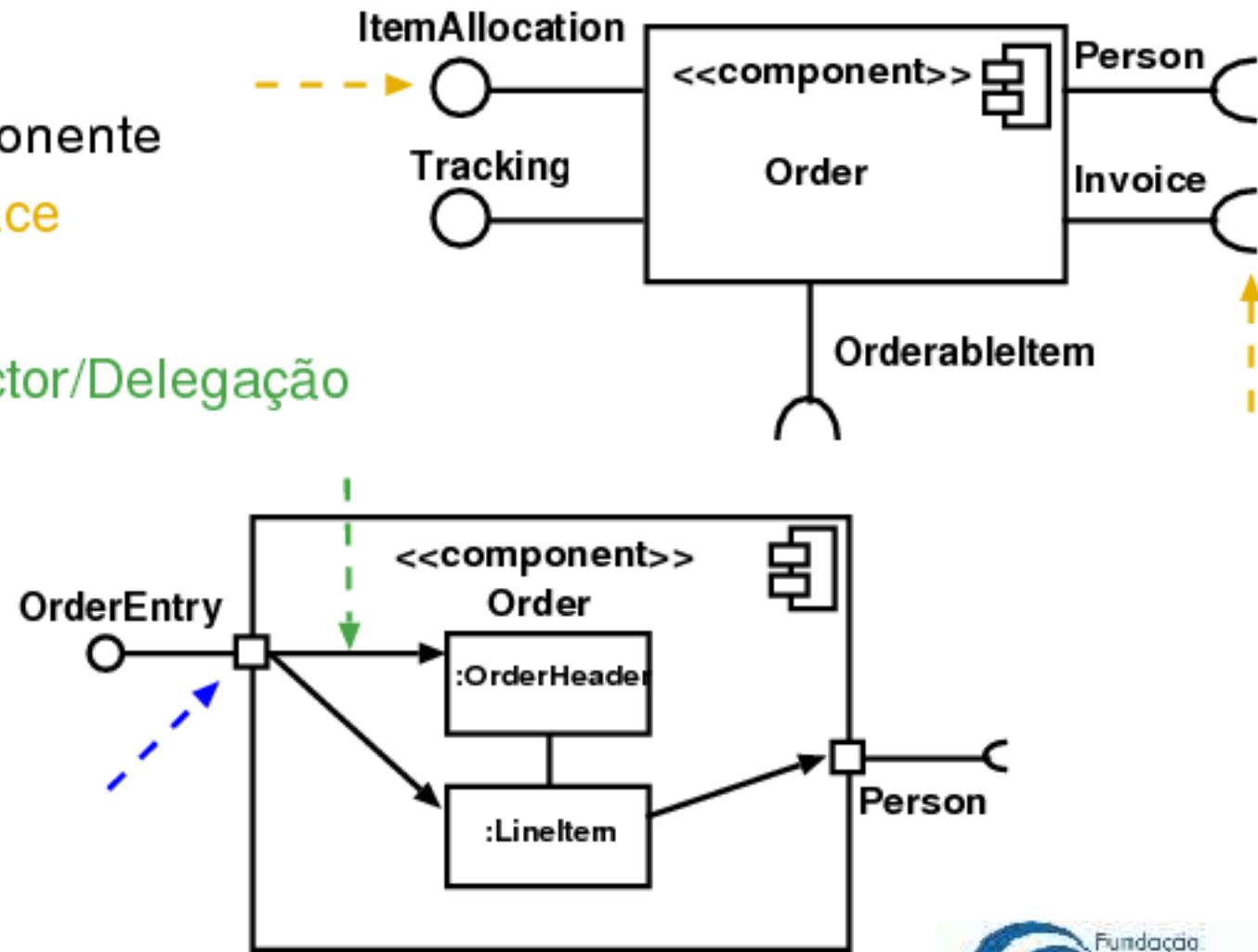


- Algumas das vantagens do DBC envolvem
 - Controle sobre a complexidade
 - Capacidade de substituição de partes do sistema
 - Capacidade de configuração do sistema
 - Métodos com ênfase na reutilização
 - UML Components
 - Catalysis
 - Kobra

DBC x OO

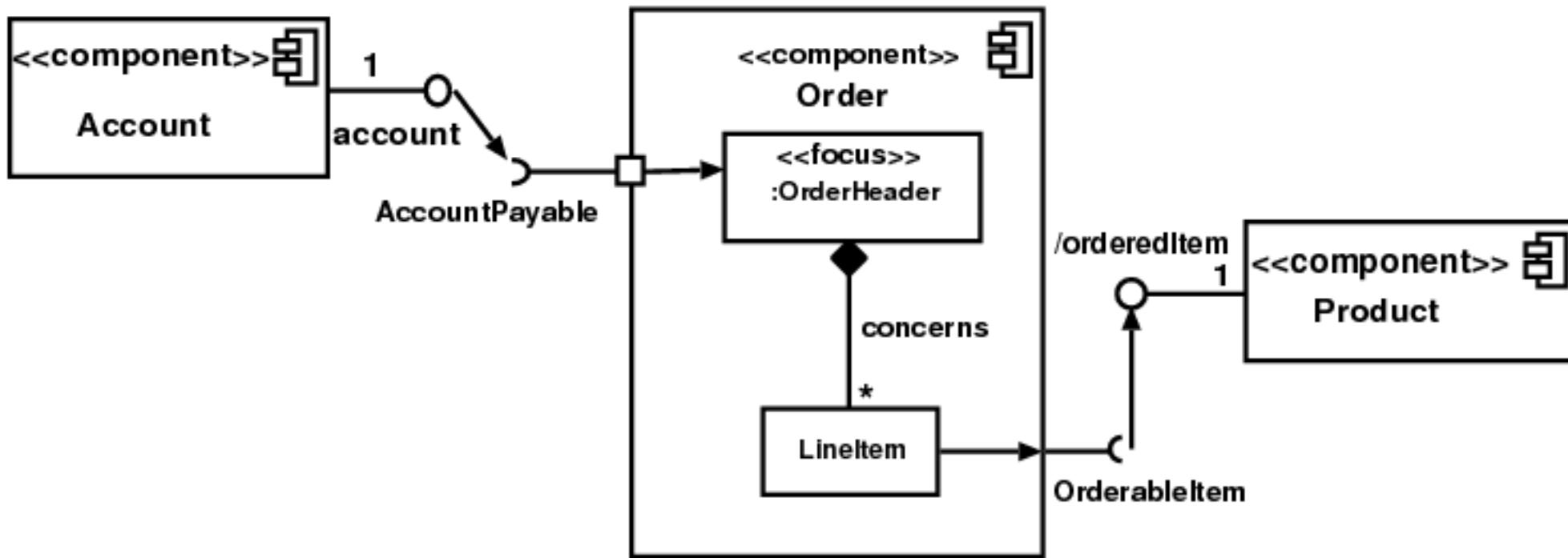
→ UML como Notação para DBC

- Componente
- Interface
- Porta
- Conector/Delegação



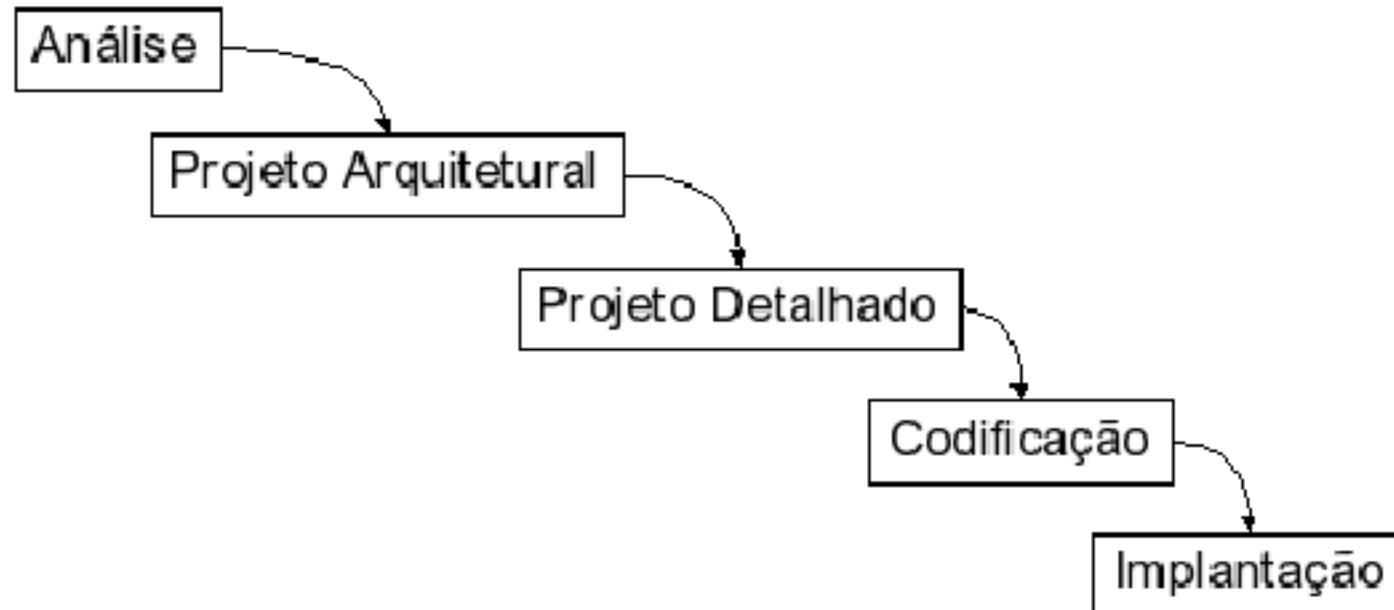
DBC x OO

→ Projeto interno do componente
— OO



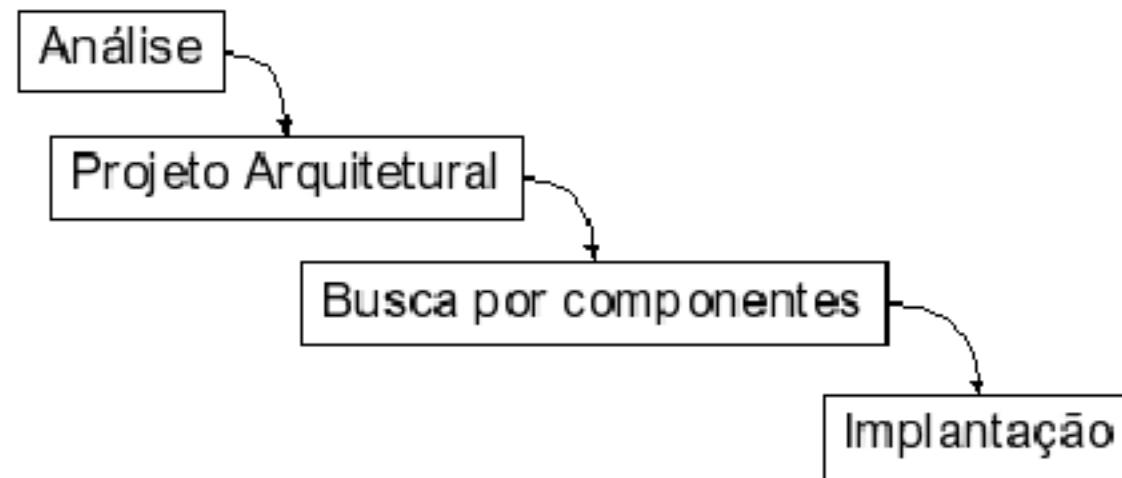
Reutilização no DBC

→ Processo convencional



Reutilização no DBC

→ Processo ideal com reutilização



Reutilização no DBC

→ Usualmente

- Nem todos os componentes desejados são encontrados
- Os componentes encontrados não são exatamente o que se esperava

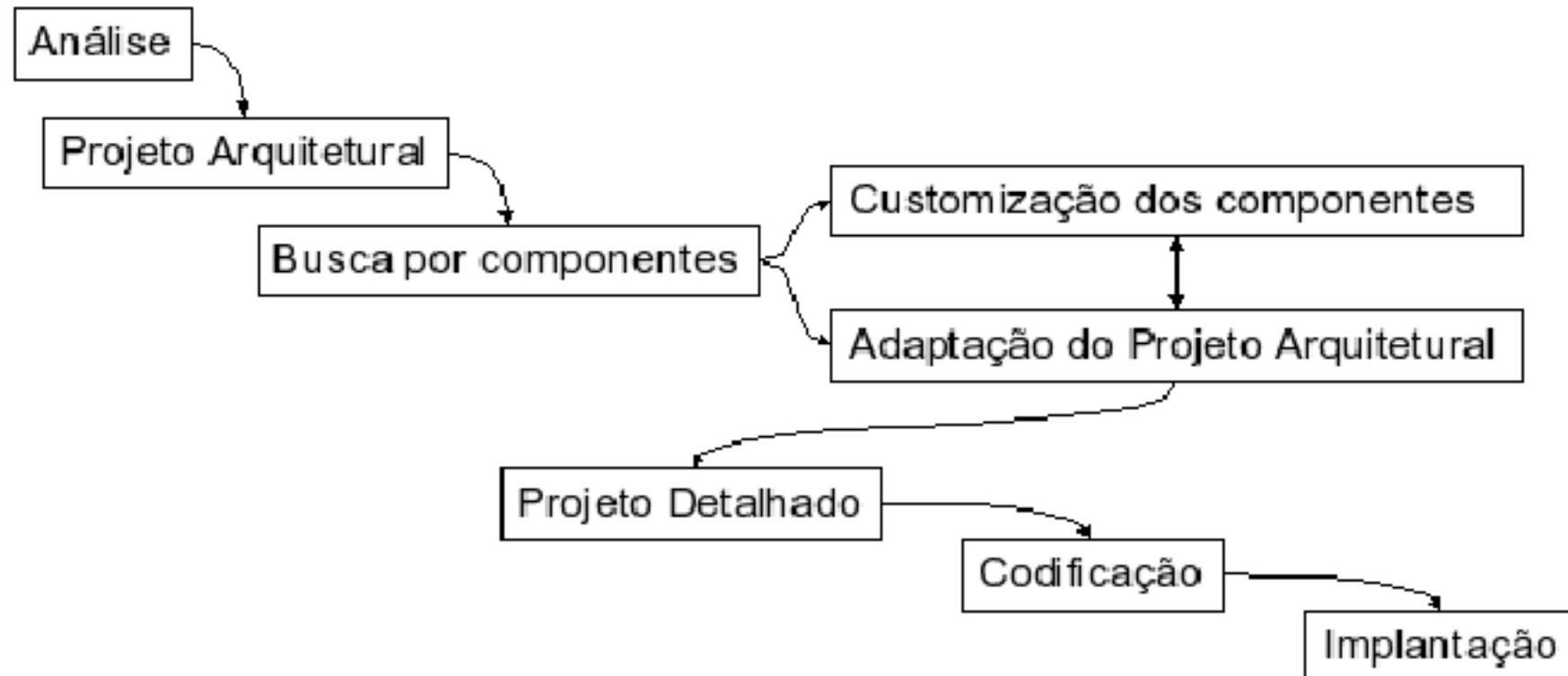
→ Possíveis soluções

- Adaptar a arquitetura
- Customizar o componente

→ Normalmente, ambos são necessários em diferentes graus

Reutilização no DBC

→ Processo com reutilização



Customização de Componentes

- Existem duas formas principais para a customização dos componentes
 - Variabilidade
 - Adaptação
- Idealmente, a customização via variabilidade deve ser preferida, pois usa mecanismos predefinidos para esse fim
- Customização via variabilidade
 - Geração (tempo de desenvolvimento)
 - Parametrização (tempo de implantação)
 - Interface de configuração (tempo de execução)

Customização de Componentes

- ➡ Nem sempre a customização via variabilidade é possível
 - ➡ Nem todas as situações são previstas
 - ➡ Neste caso, customização via adaptação pode ser a única solução

- ➡ Customização via adaptação
 - ➡ Caixa branca
 - ➡ Caixa preta

Customização de Componentes

→ Requisitos para customização via adaptação

- Separação
- Transparência
- Combinação
- Reutilização
- Configuração

→ Técnicas de adaptação

- Copiar e colar
- Herança
- Embrulho

Customização de Componentes

→ Requisito "Separação"

- Componente e adaptação como elementos distintos
- Facilita a manutenção
- Não é preciso entender o componente a fundo

→ Requisito "Transparência"

- Outros componentes vêem o componente de forma equivalente antes e depois da adaptação

→ Requisito "Combinação"

- A adaptação pode ser combinada com outras adaptações (adaptação da adaptação)

Customização de Componentes

→ Requisito "Reutilização"

- Parte genérica da adaptação (reutilizável)
- Parte específica da adaptação

→ Requisito "Configuração"

- Parte específica da adaptação parametrizável

Customização de Componentes

→ Técnica "Copiar e colar"

- Necessita um grande entendimento da implementação do componente
- Difícil de manter as múltiplas cópias
- Gera resultados rapidamente

→ Técnica "Herança"

- O componente precisa ter sido implementado com linguagem OO
- Código da adaptação fica separado do componente
- Necessita entendimento das partes do componente

Customização de Componentes

→ Técnica "Embrulho"

- Não é necessário um entendimento profundo do componente
- Grande quantidade de código "desnecessário" (elementos da interface que não mudaram)
- Possibilidades limitadas de adaptação
- Redução de desempenho

→ Não existe uma técnica perfeita

- Algumas apóiam mais requisitos que outras
- Algumas somente estão disponíveis em situações específicas

Customização de Componentes

Requisito	Copiar e Colar	Herança	Embrulho
Separação	--	-	+
Transparência	+	+	-
Combinação	-	-	+
Reutilização	-	-	+/-
Configuração	-	-	-

Restrições no DBC

→ Restrições sobre componentes

- Requisitos
- Legados
- Em nível conceitual: arquitetura
- Em nível implementacional: plataforma

→ Arquitetura

- Estabelece estilos
- Define conexões
- Usa protocolos

Restrições no DBC

→ Plataforma

- Fornece funcionalidades básicas
- Tenta garantir alguns atributos de qualidade

→ Ex.: O sistema operacional pode ser visto como plataforma para execução de programas convencionais

→ Principais plataformas de DBC

- J2EE
- .NET
- CORBA

Restrições no DBC



Funcionalidades básicas de uma plataforma

- Localização de componentes
- Entrega de mensagens
- Persistência de dados
- Transação
- Autenticação
- Autorização
- Distribuição
- Etc.

Restrições no DBC

- ➡ Cada plataforma define os tipos possíveis de componentes
- ➡ Genericamente, tipos comuns são
 - ➡ Conceitos do negócio, que demandam um mecanismo de persistência
 - ➡ Funcionalidades do negócio, que demandam um mecanismo de processamento
 - ➡ Síncrono
 - ➡ Assíncrono

Restrições no DBC

- ➡ A plataforma monitora os componentes
 - ➡ Componentes podem ser eliminados (ex.: consumo excessivo de processamento/memória)
- ➡ Os componentes devem estar aderentes à arquitetura/plataforma
 - ➡ Linguagens permitidas
 - ➡ Mecanismos de persistência utilizados
 - ➡ Formas de distribuição e conexão (com outros componentes e com a plataforma)
 - ➡ Mecanismos de controle de concorrência
- ➡ Componentes devem seguir procedimentos definidos pela arquitetura/plataforma
 - ➡ Ex.: Notificar o término de uso de uma conexão com BD

Restrições no DBC

- ➡ Já que o componente é tão dependente da plataforma...
 - Como reutilizar em outras plataformas?
- ➡ Solução: Separar tipos diferentes de restrições
 - Restrições impostas pela plataforma (não reutilizáveis fora da plataforma)
 - Restrições independentes de plataforma (reutilizáveis)

Bibliografia

→ Básica

- Craig Larman, 2007, "Utilizando UML e Padrões", 3^a ed., Capítulos 37.2.

→ Extra

- Szyperski, C., 2002, Component Software: Beyond object-oriented programming, Addison-Wesley.
- Bosch, J., 2000, Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach, Addison Wesley
- Page-Jones, M., 1999, Fundamentals of Object-Oriented Design in UML, Addison-Wesley.