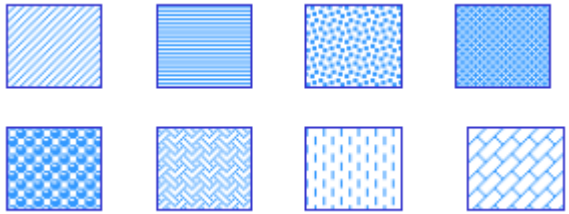
Integração de Unidade e Integração

Paulyne Juca (paulyne@ufc.br)

Adaptado de “Testes de Unidades e de Integração” da Prof. Eliane Martins -UNICAMP

Teste de unidade

- Visam exercitar detalhadamente uma unidade do sistema.
  - Sem perda de generalidade, consideraremos aqui uma unidade como sendo um componente, onde por componente entenda-se:
    - Entidade executável independente.
    - Seu código fonte pode ou não ser conhecido: depende se os testes estão sendo feitos pelo fornecedor ou pelo usuário do componente.
    - Pode representar:
      - Uma função.
      - Uma classe ou um tipo abstrato de dados.
      - Um grupo pequeno de classes.
      - Um framework.
      - Um sistema, cujo acesso se dá através de sua interface: gráfica ou API (Application Programming Interface).



Teste de Unidade

- Tipos de testes
  - Caixa branca:
    - Mais comuns
      - Visam exercitar código (ou partes dele)
  - Caixa preta:
    - Mais comuns: testes de interfaces
      - Visam determinar se implementação aceita dados válidos e rejeita dados inválidos
    - Menos comuns: testes baseados em modelos
      - Visam determinar se a unidade apresenta o comportamento especificado

- Preciso de 11 voluntários. ☺

O QUE APRENDEMOS?

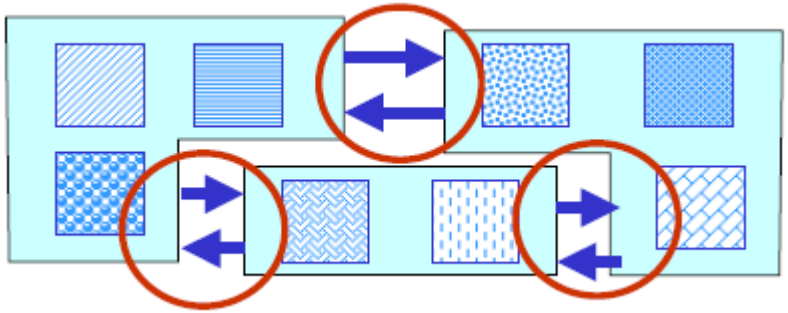
- Antes de mais nada, precisamos definir características comuns e dependências
  - Caso contrário:
    - Partes do corpo desproporcionais
      - Mãos e pés de tamanhos diferentes
      - Braços finos ou grossos demais
- Precisamos também definir como as partes serão interligadas (interface)
  - Mensagens esperadas
  - Formato de entradas e saídas
  - Nome dos métodos e classes

E se a gente fosse fazer o sistema da biblioteca

- Quais os módulos que podemos identificar?
- Quais podem ser feitos em paralelo?
- Quais não podem?

Teste de integração

- Integram unidades já testadas
- Visam descobrir problemas de interação e de
- compatibilidade entre as unidades testadas



Falhas de Integração

- Falhas de interpretação: ocorrem quando a funcionalidade implementada por uma unidade difere do que é esperado.
  - B implementa incorretamente um serviço requerido por A.
  - B não implementa um serviço requerido por A.
  - B implementa um serviço não requerido por A e que interfere com seu funcionamento.
- Falhas devido a chamadas incorretas:
  - B é chamado por A quando não deveria (chamada extra).
  - B é chamado em momento da execução indevido (chamada incorreta).
  - B não é chamado por A quando deveria (chamada ausente).

Falhas de Integração

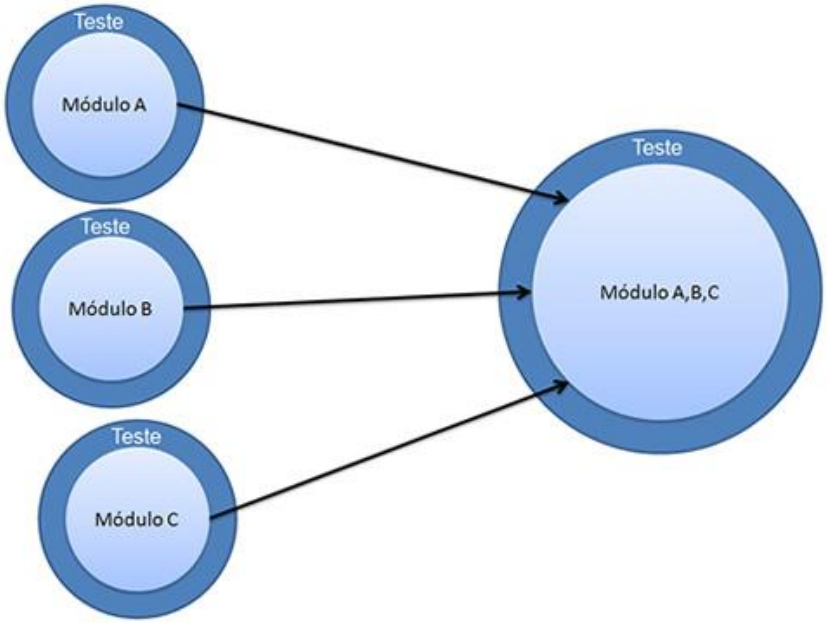
- Falhas de interface: ocorrem quando o padrão de interação (protocolo) entre duas unidades é violado.
  - violação da integridade de arquivos e estruturas de dados globais
  - tratamento de erros (exceções) incorreto
  - problema de configuração / versões
  - falta de recursos para atender a demanda das unidades
  - objeto incorreto é associado a mensagem (polimorfismo)
- Problemas não funcionais: ocorre quando requisitos não funcionais são violados
  - Módulo B não tem o tempo de resposta esperado por A
  - B lança exceções não esperadas por A

Teste de Integração

- Importância
  - Determinar se os diversos componentes de um sistema podem interoperar
    - Sistemas podem ser constituídos de componentes próprios e de terceiros (COTS)
  - Desenvolvimento incremental
    - A cada novo incremento, testes de integração precisam ser realizados

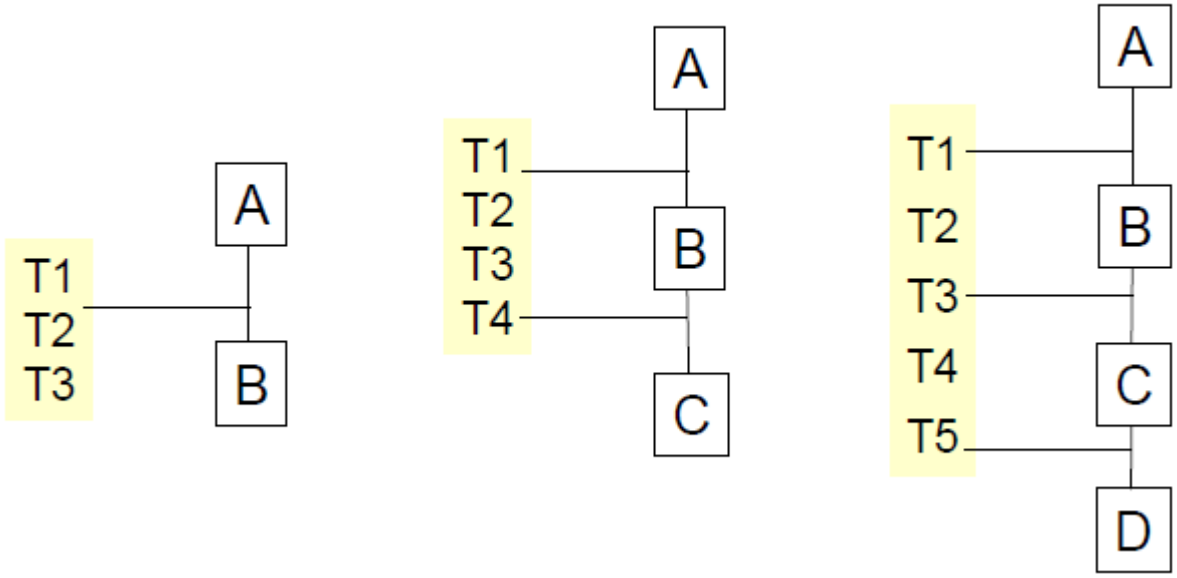
Abordagens de Teste de Integração

- Não-incremental (big-bang)
  - Módulos testados individualmente primeiro
  - Módulos integrados de uma vez só
    - Esforço de preparação é menor
    - Esforço para observação, diagnóstico e correção de falhas é maior



Abordagens de Teste de Integração

- Incremental
  - Módulos testados individualmente primeiro
  - Módulos integrados ao poucos
    - Estratégias mais comuns: top-down ou bottom-up



Incremental

- Segundo Pezzè e Young, as estratégias de escolha de módulos a serem integrados podem ser:
  - Baseadas na estrutura do sistema
    - Uso do grafo de dependências
    - Baseadas na arquitetura
  - Orientadas por funcionalidades

Análise de Dependências

- Componentes podem depender uns dos outros de várias formas:
  - Composição e agregação (use de classes para definir atributos)
  - Herança
  - Variáveis globais
  - Chamadas a interfaces (API)
  - Objetos servidores
  - Objetos usados como parâmetros de mensagens
  - Ponteiros para objetos passados como parâmetros
  - Tipos de parâmetros usados na definição de classes genéricas

Análise de Dependências

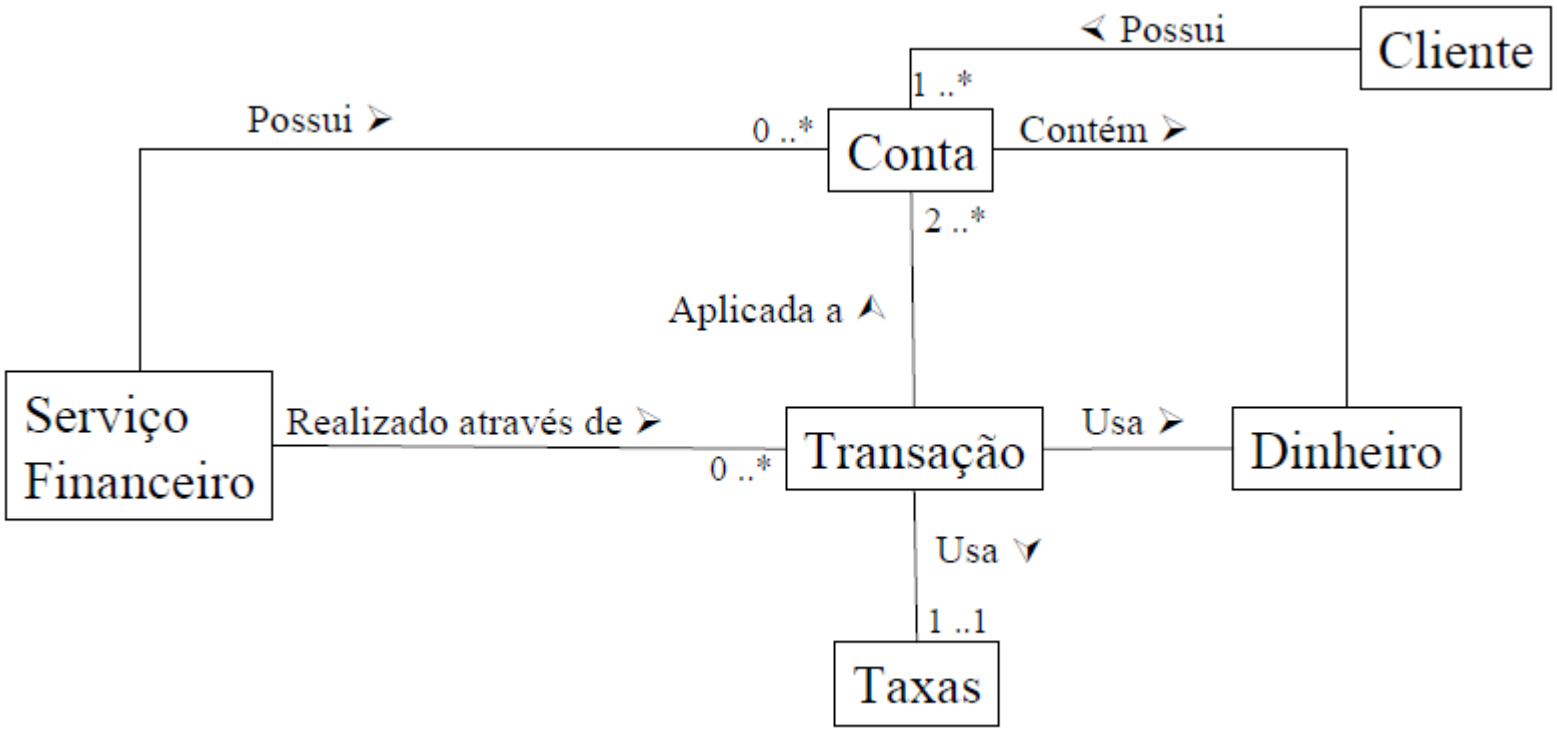
- Análise de dependências pode ser útil
  - Para determinar a ordem de testes
  - Para determinar impacto de modificações

- Tipos de dependências
  - Explícitas:
    - Troca de mensagens
    - Chamada de procedimentos
    - Uso de comunicação entre processos
  - Implícitas:
    - Comunicação através de dados persistentes
    - Restrições de sequencia
    - Restrições de tempo

Grafo de Dependência

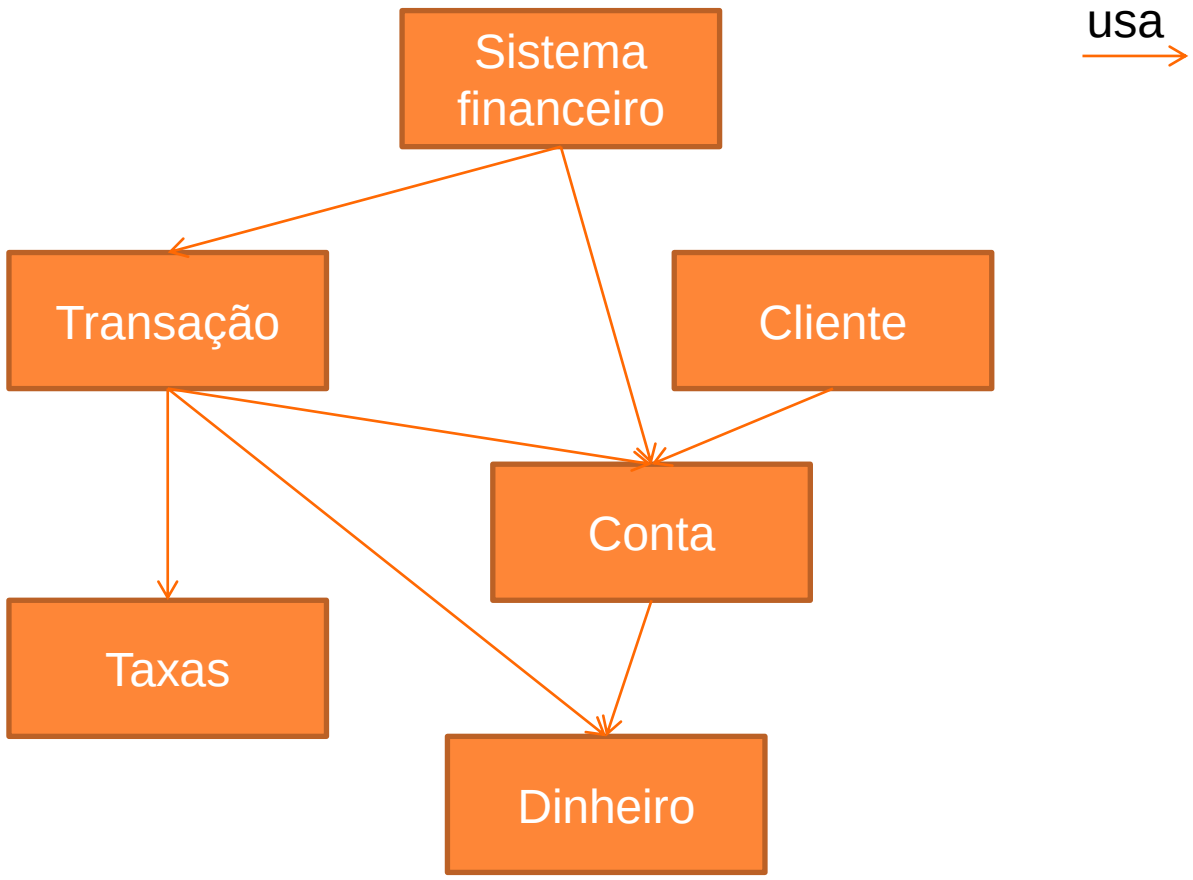
- Um modelo muito comum nos testes de integração
- Aplicável em paradigma estruturado ou OO
- Nós: unidades
  - Funções ou métodos
  - Objetos
  - Componentes
- Arcos: dependências entre unidades:
  - A chama B
  - B é parte de A
  - A envia mensagem para B

Exemplo: diagrama de classes



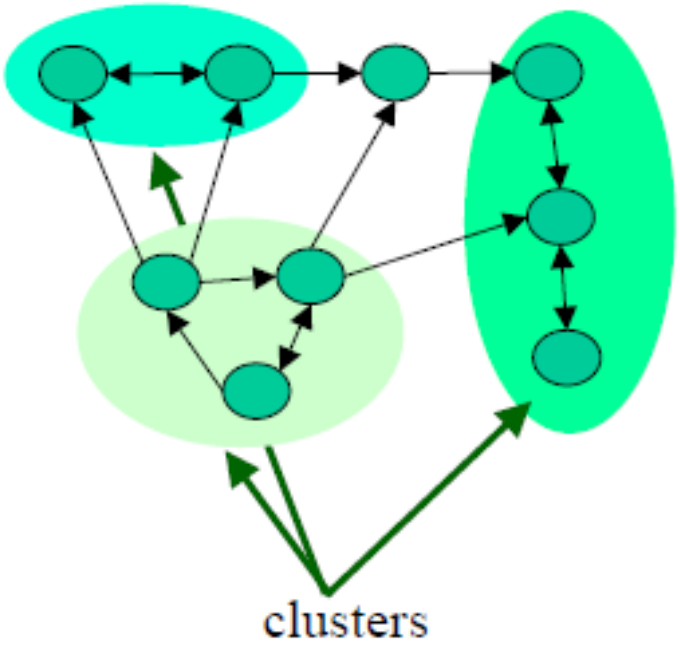
[inspirado em Binder00, 13.1.3]

Exemplo de dependência: X usa Y



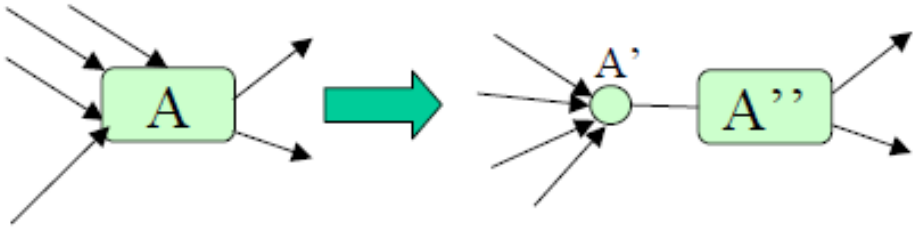
Detecção de Ciclos

- Um aspecto importante do grafo de dependências é a possibilidade de detectar ciclos, isto é, elementos que estão fortemente acoplados
- Ciclos são elementos que precisam ser refatorados

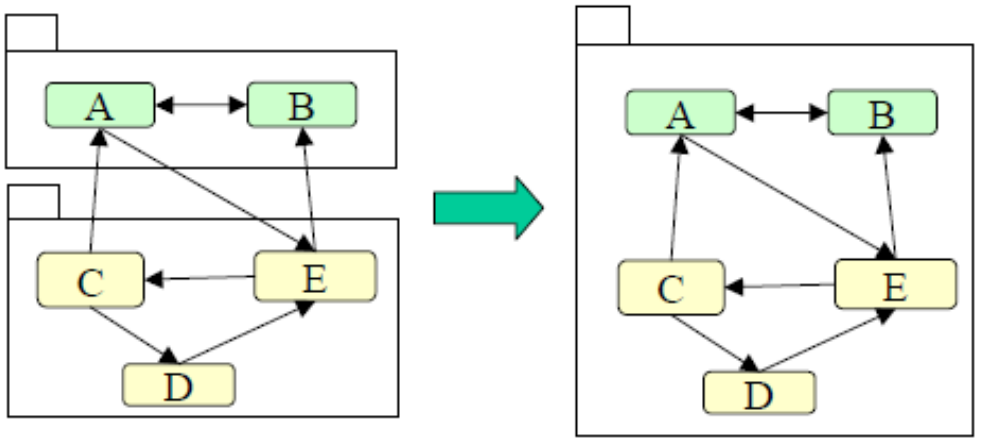


Eliminando Ciclos

- Dependências entre classes:
  - Dividir a classe, criando uma interface



- Dependências entre pacotes:
  - Juntar pacotes
  - Dividir as classes fortemente acopladas em outro pacote



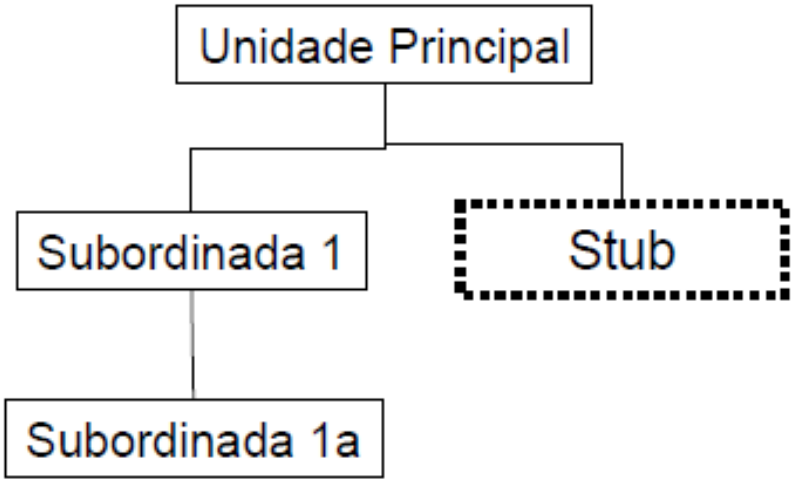
Teste Apoio

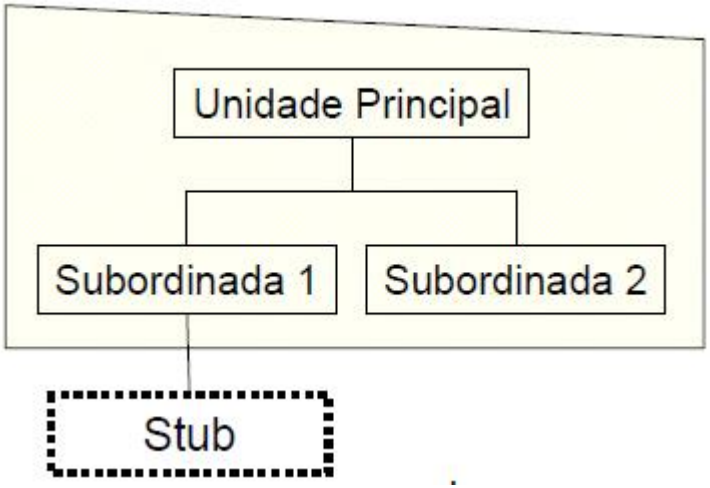
- Código desenvolvido para facilitar o teste
  - Usado para permitir a execução de parte do código durante o desenvolvimento do sistema completo
  - Analogia: andaimes de construção: estrutura construída ao redor de um prédio durante sua construção.

- Tipos:
  - Drivers: programas ativadores substitutos.
  - Harness: substituem partes do ambiente de desenvolvimento
  - Stubs: substituem funcionalidades que geram dados para o componente
  - Mocks: emulam comportamento de outros componentes

Integração Top-Down

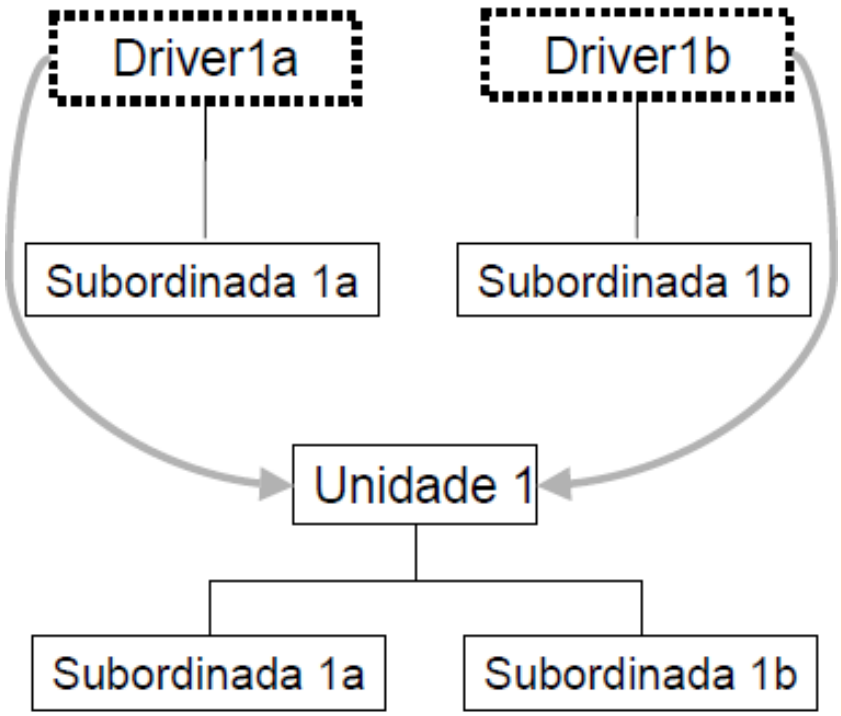
- Começa com a unidade principal e vai aos poucos integrando as unidades subordinadas
- Em OO: classes de controle primeiro
- Utiliza stubs em lugar das unidades subordinadas
- Pode ser feita em:
  - Profundidade
  - Largura





Integração bottom-up

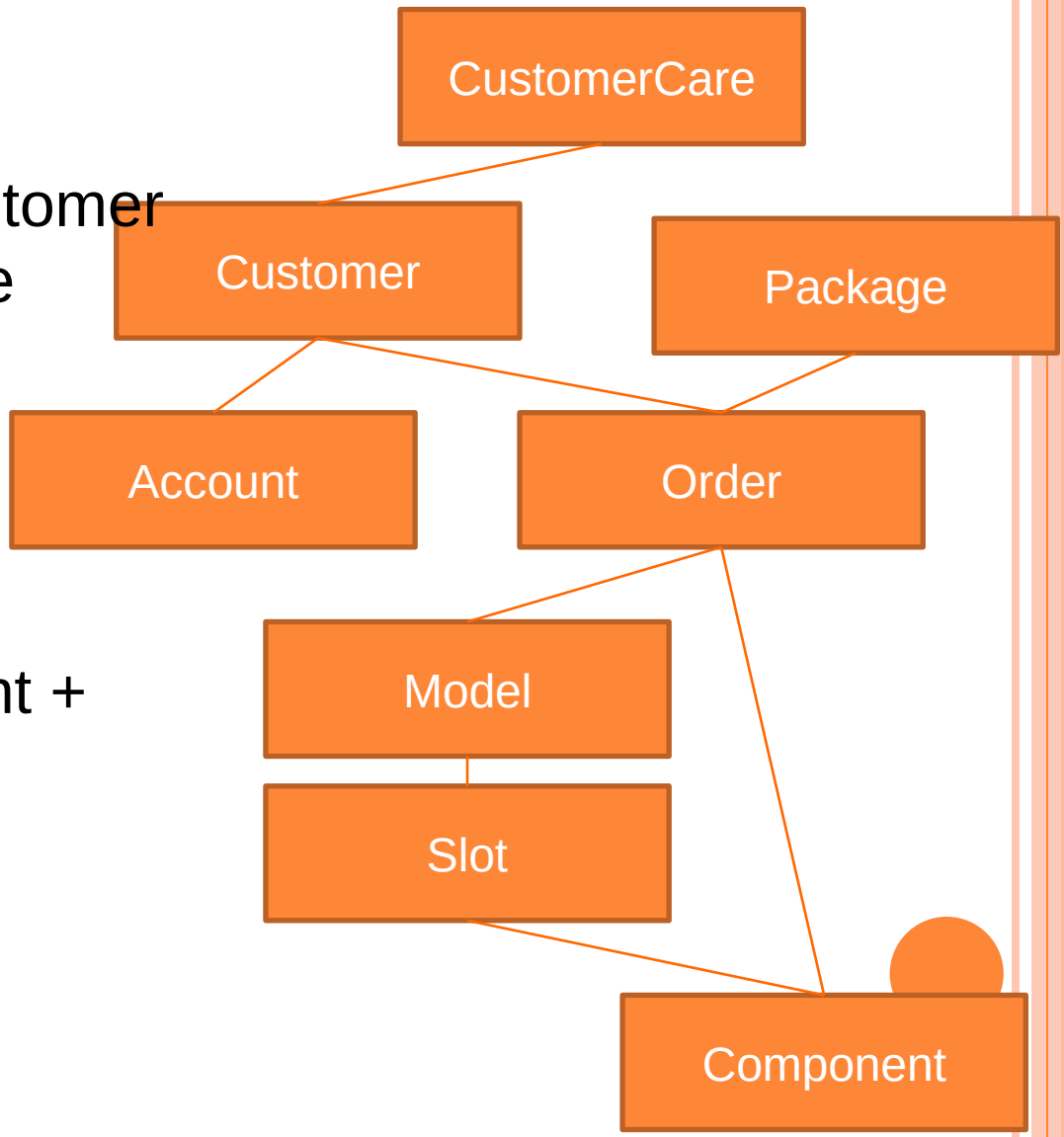
- Começa a integração pelas unidades subordinadas
- Em OO: começar pelas classes independentes ou que usam poucas servidoras
- Utiliza drivers em lugar das unidades de controle
- Os algoritmos de mais baixo nível são testados antes de serem integrados ao resto do sistema



Exemplo

- Top-down

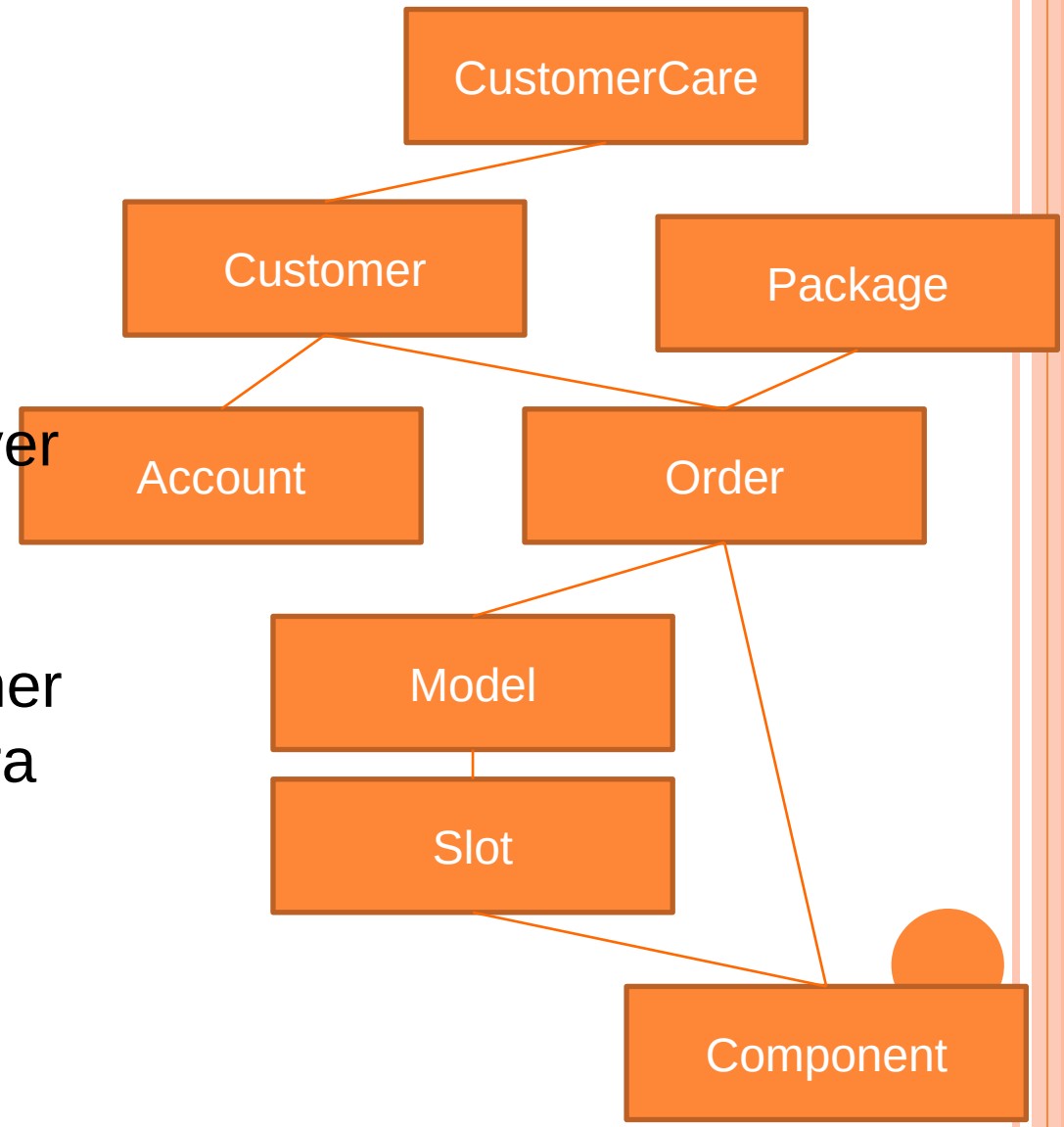
1. CustomerCare + Customer (stubs para account e order)
2. + Account + Order + Package (stubs para model e component)
3. + Model + Component + Slot



Exemplo

- Bottom-up

1. Slot + Component (driver para model e order)
2. + model + order (driver para package e customer)
3. + Package + Customer + Account (driver para customerCare)
4. + CustomerCare



Integração por colaboração

- Objetivo: cobertura de interações entre unidades
- Integrar unidades necessárias para realizar uma determinada colaboração
- As colaborações devem ser explicitamente especificadas
  - Diagramas de colaboração ou de atividades podem servir de modelo de base
- A ordem de integração das colaborações também pode ser obtida com o uso de um grafo de dependências

Integração Mista

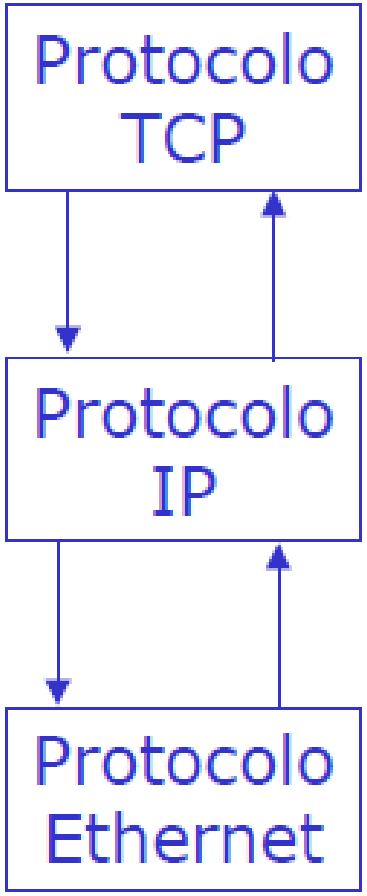
- Combina várias das técnicas anteriores
  - Usar as técnicas mais adequadas de acordo com a parte do sistema que se quer integrar
- Útil quando sistema a ser integrado depende de uma infraestrutura (backbone) que também está em desenvolvimento
  - Não vale a pena criar stubs para substituir a infra-estrutura de execução
  - Começar os testes integrando as unidades que compõem a infraestrutura
  - Depois da infra-estrutura ter sido devidamente testada, integrar as demais unidades do sistema a ela, da maneira mais conveniente

Integração baseada na arquitetura

- Integração por camadas
  - Exercitar incrementalmente as interfaces e componentes em uma arquitetura em camadas
- Integração cliente/servidor
  - Exercitar redes de componentes fracamente acoplados que usam um servidor comum
- Integração de serviços distribuídos
  - Exercitar redes de componentes fracamente acoplados par a par

Integração em camadas

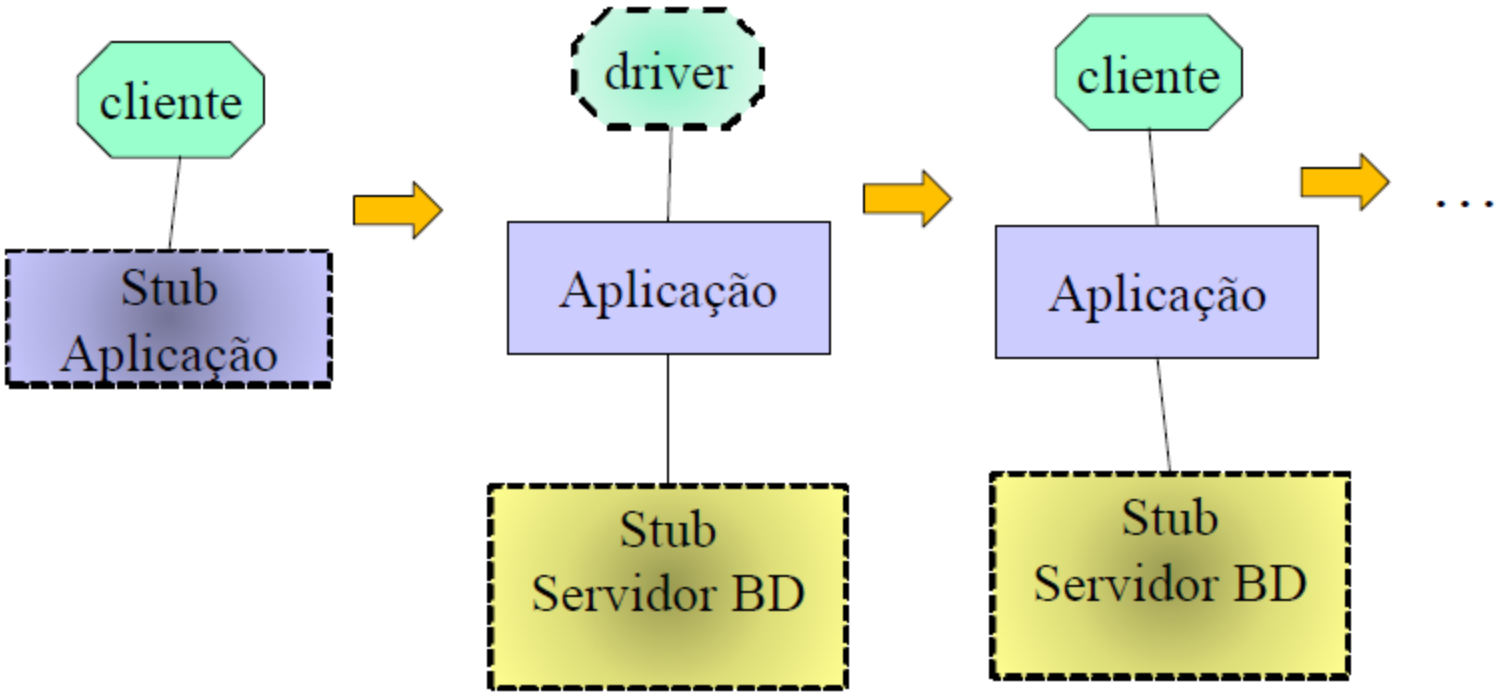
- Útil quando o sistema é modelado como uma hierarquia que permite interfaces somente entre camadas adjacentes
- Também combina diversas estratégias:
  - Quaisquer das estratégias já vistas para integrar unidades internas a cada camada
  - Cada camada é testada isoladamente
  - Usar estratégia descendente ou ascendente para integrar as camadas



Integração cliente-servidor

- Útil quando a arquitetura é distribuída, sem um ponto único de controle:
  - Servidores reagem a mensagens dos clientes
  - Clientes respondem a estímulos do ambiente
- Estratégia:
  1. Testar cada cliente com um stub do servidor
  2. Testar o servidor com stub do cliente
  3. Integrar clientes ao servidor

Integração cliente-servidor



Integração frequente

- Integração frequente
  - Executar testes de integração periodicamente (por hora, por dia, por semana)

- Processo Facebook
  - Toda sexta tem release.
  - Módulos estáveis até quarta entram na release de sexta
  - Quinta e sexta são reservados para testar a release

Referências

- http://qualidadebr.wordpress.com/tag/teste-de-integracao/