Departamento de Sistemas Computacionais

Desenvolvimento de Software Orientado a Aspectos utilizando RUP e AspectJ

Sérgio Soares
sergio@dsc.upe.br

UPE UNIVERSIDADE DE PERNAMBUCO

POLI ESCOLA POLITÉCNICA DE PERNAMBUCO

Programação Orientada a Objetos

- Lida com conceitos mais intuitivos
- Permite ganhos
  - Reuso
  - Manutenção
  - Adaptação
- Padrões de projetos
  - Auxiliam a POO

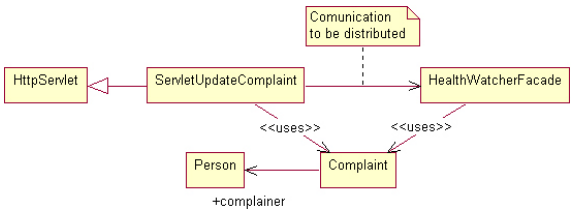
Exemplo: Sistema Disque Saúde

- Um sistema de informação
  - Registra e encaminha queixas para o sistema de saúde
  - Exibe informações sobre unidades de saúde e suas especialidades
- Requisitos não-funcionais
  - Distribuído
  - Acesso concorrente
  - Extensível
    - Armazenamento de dados
    - Tecnologia de distribuição

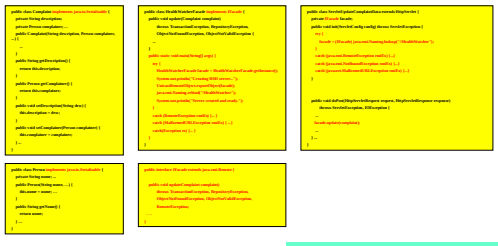
Exemplo: Sistema Disque Saúde

- Implementado em Java
  - Servlets implementam a GUI
  - Utiliza vários padrões de projetos para contemplar requisitos não-funcionais
- Como implementar a distribuição no sistema?

Disque Saúde local



Disque Saúde distribuído com RMI



Código RMI é vermelho...

Implementação OO: problemas!

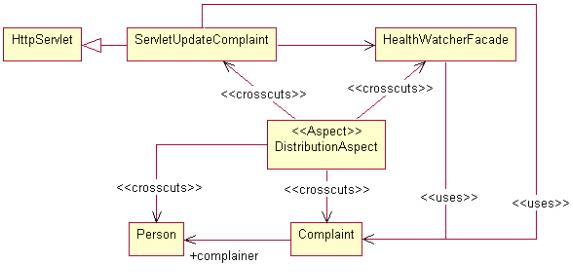
- *Tangled code* (código entrelaçado)
  - código de distribuição misturado com código de negócio e de GUI
- *Spread code* (código espalhado)
  - código de distribuição em várias classes
- Distribuição é um *crosscutting concern* (interesse transversal)
- Difícil de manter e reusar
  - mudanças no protocolo de distribuirão (RMI, CORBA, EJB) são invasivas

AOP — Aspect-oriented programming

- Melhora a modularidade de interesses transversais
  - distribuição, gerenciamento de dados, controle de concorrência, tratamento de exceções, logging, debugging, …
- Auxilia separação de interesses (*separation of concerns*)
  - aumenta extensibilidade e reuso

Disque Saúde distribuído com AOP (usando Java RMI)

Disque Saúde distribuído com AOP

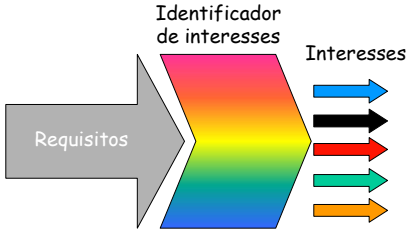


Implementação com AOP

- Aumento em modularidade, reuso e extensibidade
  - Mais unidades de código
  - Mudanças no sistema local podem causar impacto nos aspectos de distribuição
- *Separation of concerns* (separação de interesses)
  - Relação entre os aspectos e o resto do sistema nem sempre é clara
- Normalmente menos linhas de código

Em direção ao Desenvolvimento de Software Orientado a Aspectos (DSOA)

Passo 1: Identificando interesses (decomposição)



Engenharia de requisitos

Interesses do Disque Saúde

Interface com o usuário

Gerenciamento de dados

Distribuição

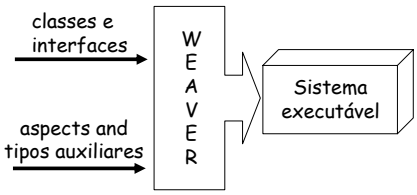
Regras de negócio

Controle de concorrência

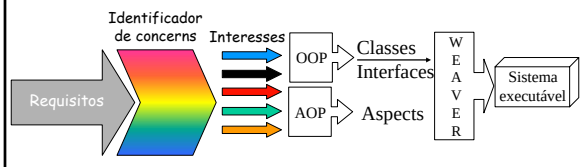
Passo 2: Implementar o sistema, separando os interesses

- Alguns *concerns* são bem modelados como objetos (núcleo do sistema)
  - interface com o usuário
  - regras de negócio
- Outros (*crosscutting concerns*) como aspectos
  - distribuição, controle de concorrência, armazenamento de dados

Passo 3: Recompor o sistema

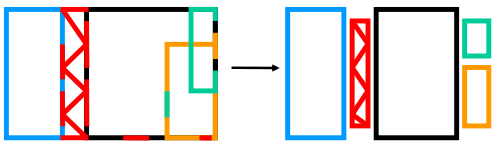


Desenvolvimento de Software Orientado a Aspectos



Requisitos funcionais
Controle de concorrência
Gerenciamento de dados
Interface com o usuário
Distribuição

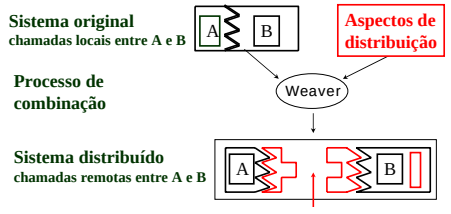
OOP vs AOP



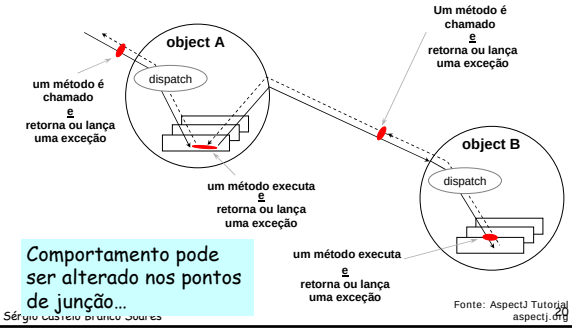
Requisitos funcionais
Controle de concorrência
Gerenciamento de dados
Interface com o usuário
Distribuição

Combinação (*weaving*) é usada para …

- Compor o "núcleo" do sistema com os aspectos



Composição nos pontos de junção (*join points*)



Comportamento pode ser alterado nos pontos de junção…

Fonte: AspectJ Tutorial aspectj.org

Conjunto de pontos de junção (`pointcut`)

- Identifica pontos de junção de um sistema
  - chamadas e execuções de métodos (e construtores)
  - acessos a atributos
  - tratamento de exceções
  - inicialização estática e dinâmica
- Expõe o contexto nos *join points*
  - —argumentos de métodos, objetos alvo, atributos
- Composição de pontos de junção
  - `&&, || e !`

AspectJ: identificando chamadas de métodos da fachada (servidor)

identifica código dentro da classe ...

nome do conjunto de pontos de junção

```
pointcut facadeMethodsCall():
        within(HttpServlet+) &&
        call(* IFacade+.*(..));
```

identifica chamadas de …

qualquer método

com quaisquer argumentos

Comportamento transversal (`advice`)

- Define código adicional que deve ser executado…
  - before
  - after
    - —after returning
    - —after throwing
  - ou around

pontos de junção

AspectJ: antes (`before`) de chamar métodos da fachada

```
private IFacade remoteFacade;
before(): facadeMethodsCall() {
  getRemoteInstance();
}
synchronized void getRemoteInstance() {...
  remoteFacade =
    (IFacade) java.rmi.Naming.lookup(...);
  ...}
```

AspectJ: transformando chamadas locais em remotas

```
void around(Complaint c) throws Ex1,…:
  facadeMethodsCall() && args(c) &&
  call(void update(Complaint))
{
  try { remoteFacade.update(c);
  } catch (RemoteException rmiEx) {...}
}
```

obtendo e utilizando argumento de método em um ponto de junção

Além de mudanças (dinâmicas) com comportamento transversal…

- AspectJ suporta mudanças estáticas
  - alterar relação de subtipo
  - adicionar membros a classes

AspectJ: mudanças estáticas

```
declare parents:
  HealthWatcherFacade implements IFacade;
declare parents: Complaint || Person
  implements java.io.Serializable;
public static void
  HealthWatcherFacade.main(String[] args){
  try {...
    java.rmi.Naming.rebind("/HW");
  } catch  ...
}
```

Adicionando o método main na classe fachada

Alterando a hierarquia de tipos

Mais construtores de AspectJ

- **target**(<nome do tipo>)
  - Identifica pontos de junção onde o objeto alvo é uma instância de <nome do tipo>
- **this**(<nome do tipo>)
  - Identifica pontos de junção cujo objeto em execução é uma instância de <nome do tipo>
- **withincode**(<assinatura de método>)
  - Identifica pontos de junção cujo código em execução pertence ao corpo do método ou construtor especificado por <assinatura de método>

Mais construtores de AspectJ

- **cflow**(<conjunto de junção>)
  - Identifica pontos de junção que estejam no fluxo de execução identificado por <conjunto de junção>
- **get**(<assinatura>) / **set**(<assinatura>)
  - Leitura/atribuição a atributos
- **declare soft**: <nome do tipo> : <conjunto de junção>;
  - A exceção <nome do tipo> será encapsulada em uma exceção não checada em tempo de compilação (runtime) em qualquer ponto de junção definido por <conjunto de junção>

Aspecto de distribuição em AspectJ

```
public aspect DistributionAspect {
  declare parents: ...
  private IFacade remoteFacade;
  public static void
    HealthWatcherFacade.main(String[] as)...
  pointcut facadeMethodsCall(): ...
  before(): facadeMethodsCall() ...
  private synchronized void
    getRemoteInstance() ...
  void around(Complaint complaint) ...
}
```

Aspectos de desenvolvimento: *debugging* simples com AspectJ

```
public aspect DatabaseDebugging {
    pointcut queryExecution(String sql):
        call(* Statement.*(String)) &&
        args(sql);
    before(String sql): queryExecution(sql) {
        System.out.println(sql);
    }
}
```


AspectJ: pontos positivos

- Modularidade, reuso, e extensibilidade de software
- Inconsciência (o*bliviousness*)
- Suporte a desenvolvimento com IDEs
  - debugging
- Produtividade
- Permite implementação e testes progressivos

AspectJ: pontos negativos

- Novo paradigma
  - relação entre classes e aspectos deve ser minimizada
  - inconsciência (o*bliviousness*)
- Projeto da linguagem
  - tratamento de exceções
  - conflitos entre aspectos
- Requer suporte de ferramentas
- Combinação (apenas) estática

Referências

- Gregor Kiczales et. al. Aspect-Oriented Programming. European Conference on Object-Oriented Programming, ECOOP'97
- http://groups.yahoo.com/group/asoc-br/
- http://www.aosd.net/
- http://www.eclipse.org/aspectj

Desenvolvimento de Software Orientado a Aspectos utilizando RUP e AspectJ

Parte 2

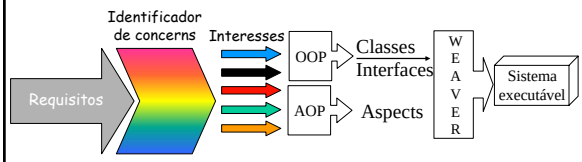
DSOA – Desenvolvimento de Software Orientado a Aspectos

- AspectJ implementa aspectos em Java . . .
- . . . mas como desenvolver sistemas orientados a aspectos?

Método de implementação OA

- Guias para o DSOA
- Complementar a técnicas de programação OO e padrões de projeto

DSOA

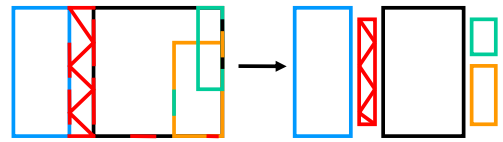


- UI e negócio usam OO
- Distribuição, gerenciamento de dados, e controle de concorrência usam OA

Derivando o método de implementação

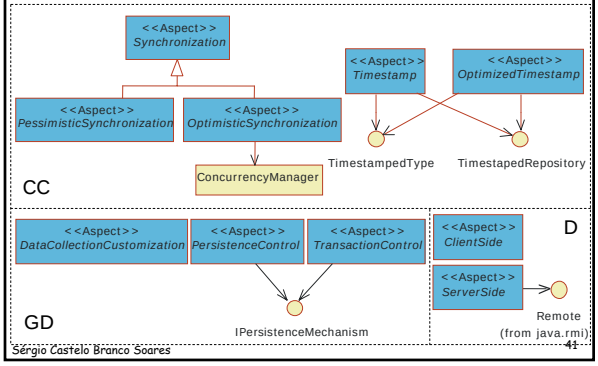
- Estudo de caso para reestruturar um programa OO em um programa OA
  - POA é útil
  - sugestão de melhorias em AspectJ
  - dependências e impactos entre os aspectos
    - distribuição vs. gerenciamento de dados
  - framework e padrões de aspectos para implementar os interesses

OOP vs AOP



Requisitos funcionais
Gerenciamento de dados
Distribuição
Controle de concorrência
Interface com o usuário

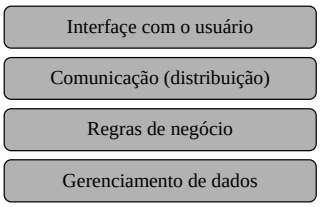
Framework de aspectos (AspectJ)

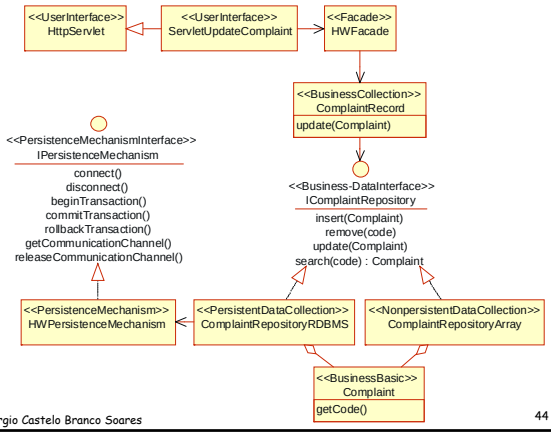


Método de implementação

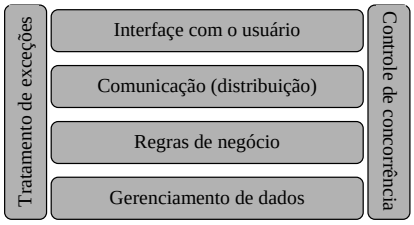
- Definido para uma arquitetura de software específica
  - utilizada em vários sistemas de informação OO reais!
  - permire a definição de guias mais precisos
  - alguns tipos podem ser gerados automaticamente
    - ferramentas de suporte

Arquitetura OO em camadas





Arquitetura OA em camadas



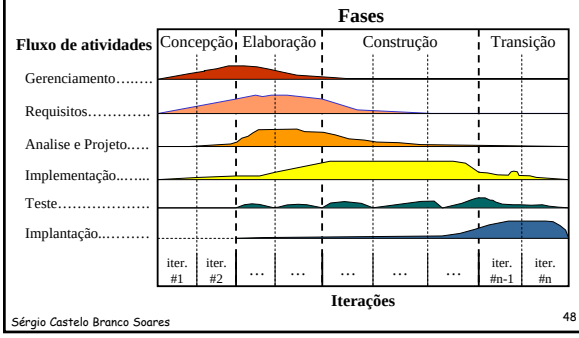
Método de implementação

- Como integrar com desenvolvimento guiado por casos de uso e com o RUP
  - impacto na dinâmica do processo
    - uso de uma abordagem de implementação alternativa
  - impacto nas atividades do "processo"
    - gerenciamento, requisitos, análise e projeto, implementação, e teste
    - mudanças e definição de novas atividades

RUP

- Não é um processo!
  - mudanças no framework do processo permitindo refletir as mesmas nas suas instâncias

RUP



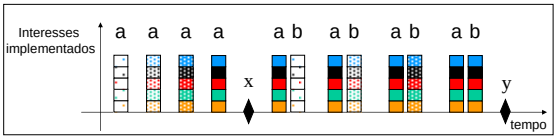
Uma abordagem alternativa de implementação

Implementação Progressiva

Implementação progressiva

- Abordagem alternativa
  - abstrai inicialmente parte dos requisitos não-funcionais
    - ex.: distribuição, persistência, e controle de concorrência
  - permite validação antecipada de requisitos funcionais
    - aumento em produtividade
  - diminui a complexidade de testes
    - testes progressivos

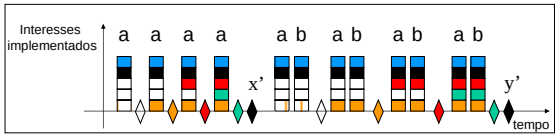
Abordagem regular



Interface com o usuário
Distribuição
Requisitos funcionais
Gerenciamento de dados persistente
Controle de concorrência
Milestone (fim de iteração)

a e b são casos de uso, conjuntos de casos de uso ou cenários

Abordagem progressiva



Interface com o usuário
Distribuição
Requisitos funcionais
Gerenciamento de dados persistente
Gerenciamento de dados não-persistente
Controle de concorrência
Milestone (fim de iteração)
Iteração functional

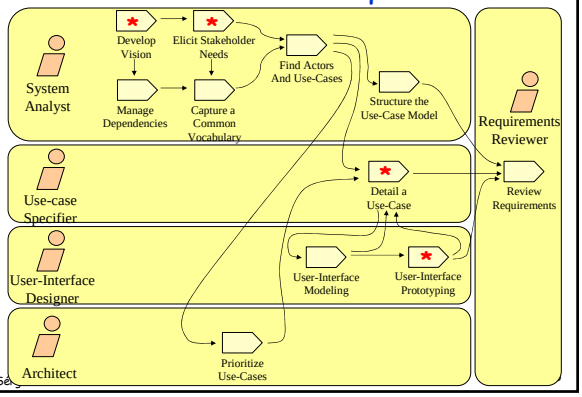
a e b são casos de uso, conjuntos de casos de uso ou cenários

Mudanças nas atividade do RUP

Gerenciamento

- No planejamento das iterações
  - considerar a abordagem progressiva
  - iterações funcionais e não-funcionais
- Análise de riscos
  - uso de um paradigma novo (DSOA)
  - uso de uma arquitetura de software específica

Atividades de requisito



Requisitos

- Desenvolver a visão / Elicitar necessidades dos stakeholders
  - a arquitetura de software a ser utilizada visa extensibilidade
    - —suporta separação de vários interesses
      - distribuição, controle de concorrência, gerênciamento de dados e tratamento de exceções
    - —possui suporte a geração automática de tipos

generalizar a arquitetura ...

Requisitos

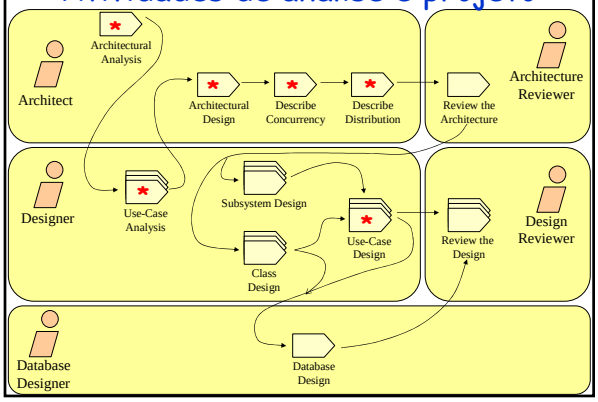
- Detalhar caso de uso
  - identificar aspectos candidatos
    - —requisitos não-funcionais
    - —e funcionais também?
    - —pesquisas na área
      - Early Aspects

http://www.early-aspects.net/

Requisitos

- Prototipar interface com o usuário
  - considerar o uso da abordagem de implementação progressiva
  - gerar um protótipo mais rapidamente e não o descartar

Atividades de análise e projeto



Análise e projeto

- Análise arquitetural
  - considerar o uso da arquitetura de software específica
- Análise de caso de uso
  - identificar tipos da arquitetura de software específica
  - classes de entidade -> classes básicas

Análise e projeto

- Projeto arquitetural
  - considerar o suporte oferecido pelo método
    - persistência -> JDBC
    - distribuição -> RMI
    - controle de concorrência -> várias opções
  - classes de controle -> atributos da classe fachada
  - classes de fronteira -> interfaces com o usuário ou com subsistemas

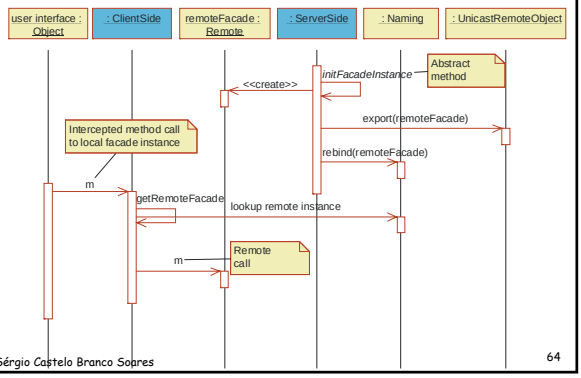
Análise e projeto

- Descrever concorrência
  - usar um método de controle de concorrência [Soares 2001]
- Descrever distribuição
  - o método suporta a distribuição entre a IU e a fachada
  - pequenas modificações no método para suportar a distribuição de outras partes

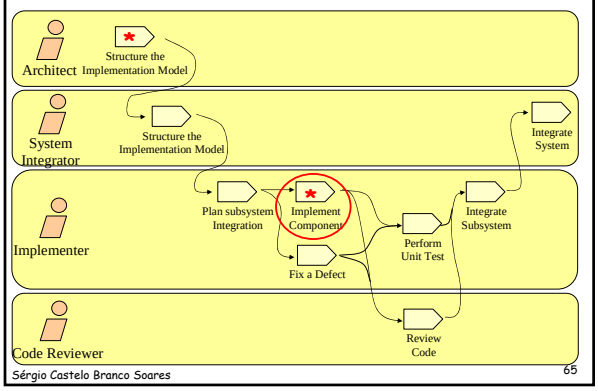
Análise e projeto

- Projetar caso de uso
  - diagramas de sequência com a iteração entre os objetos da arquitetura de software específica
  - o framework de aspectos já guia o impacto dos mesmos nos objetos do sistema
    - diagramas mostrando a iteração dos aspectos pre-definidos [Soares 2004]

Diagrama de sequência – distribuição



Atividades de implementação



Implementação

- Estruturar o modelo de implementação
  - gerar classes da arquitetura
    - ferramentas de modelagem geram as classes básicas
    - ferramenta de suporte ao método gera coleções de negócio e de dados, interfaces negócio-dados e a classe fachada
  - aspectos relacionados aos interesses transversais também podem ser gerados
    - progressiva vs. não-progressiva

Implementação

- Implementar componente
  - implementar a parte funcional e a interface com o usuário dos casos de uso selecionados
  - e . . .

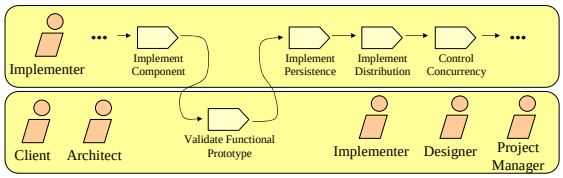
Implementação

- (versão não-progressiva)
  - gerar aspectos, específicos para o sistema em quetão, para lidar com os interesses transversais utilizando o framework de aspectos do método

Implementação

- (versão progressiva)
  - "desligar" persistência, distribuição e controle de concorrência
  - gerar coleções de dados não-persistentes e aspectos de não-persistência
  - e . . .

Novas atividades (abordagem progressiva)



Implementação

- Validar prototipo funcional
  - validar o protótipo com os stakeholders
  - anotar e executar mudanças de requisitos
  - validar os requisitos e suas eventuais mudanças

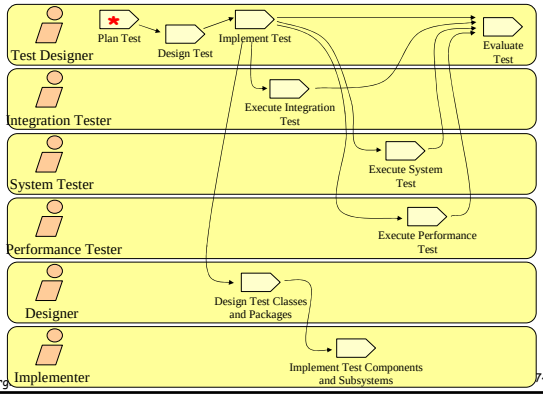
Implementação

- Implementar persistência
  - "desligar"
    - —aspectos de não-persistência
  - gerar / "ligar"
    - —aspectos de persistência
    - —coleções de dados persistentes
  - testar protótipo persistente

Implementação

- Implementar distribuição
  - gerar / "ligar" aspectos de distribuição
  - testar protótipo persistente e distribuído
- Controlar concorrência
  - gerar / "ligar" aspectos de controle de concorrência
    - —usando o método de controle de concorrência [Soares 2001]
  - testar protótipo com controle de concorrência

Atividades de teste



Teste

- Planejar teste
  - levar em conta a abordagem de implementação progressiva e o DSOA
  - testes progressivos
    - —"ligar" e "desligar" os interesses transversais
    - —testar várias combinações de interesses transversais

Resumo

- Método de implementação
  - actividades, abordagem alternativa de implementação, guias
  - relacionado a um processo de desenvolvimento real
- Aspectos podem afetar outros
  - aspectos de distribuição afetam aspectos de gerenciamento de dados
  - analise e projeto são essenciais para identificar tais diferenças

Resumo

- Padrões e framework de aspectos
  - reuso, geração de aspectos dependentes da aplicação
- Suporte de ferramenta
  - aumento de produtividade
  - refactoring de aspectos no futuro

Referências

- Ivar Jacobson, Grady Booch, and James Rumbaugh. The Unified Software Development Process. Addison Wesley, 1999.
- Grady Booch, Ivar Jacobson, e James Rumbaugh. Unified Modeling Language - User's Guide, Addison-Wesley, 1999.

Referências

[Soares 2001] Sérgio Soares. Desenvolvimento Progressivo de Programas Concorrentes Orientados a Objetos. Dissertação de mestrado. Centro de Informática-UFPE, Fevereiro 2001.

[Soares 2004] Sérgio Soares. An Aspect-Oriented Implementation Method. Tese de Doutorado. Centro de Informática-UFPE, Outubro 2004.

Software Productivity Group

http://www.cin.ufpe.br/spg

Desenvolvimento de Software Orientado a Aspectos utilizando RUP e AspectJ

Sérgio Soares

sergio@dsc.upe.br