**UNIVERSIDADE FEDERAL DE SANTA CATARINA**

# Seleção de práticas ágeis para o desenvolvimento de linhas de produto

**Diego Spillere de Souza**

**Florianópolis - SC**
**2012/1**

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

# Seleção de práticas ágeis para o desenvolvimento de linhas de produto

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação

Florianópolis – SC
2012/1

Diego Spillere de Souza

# Seleção de práticas ágeis para o desenvolvimento de linhas de produto

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação

Orientadora: Patrícia Vilain

Banca examinadora

Raul Sidnei Wazlawick

Ricardo Pereira e Silva

# Índice

## Lista de figuras

## Lista de tabelas

# 1 Introdução

Os métodos ágeis têm despertado grande interesse por parte dos desenvolvedores por serem rápidos, eficientes e atenderem as necessidades dos clientes. Mesmo apresentando características comuns, vários métodos têm sido criados [FAGUNDES, 2005].

Outra definição que desperta interesses é a que diz: "Linha de produto de software é um novo paradigma voltado à extrema utilização do reuso para construção de famílias de produtos. [...] Permitem que se estabeleça a infra-estrutura necessária para que uma família de produtos, aplicações que fornecem serviços parecidos, contudo, diferenciados, seja constituída para a construção rápida das aplicações." [ESM15, 2009]

Este trabalho prevê o estudo de práticas ágeis, selecionando aquelas que se aplicam no desenvolvimento de uma linha de produtos. Para tanto, um estudo aprofundado destes dois grandes objetos da Engenharia de Softwares será realizado, caracterizando e comparando cada um deles. Padrões e práticas de desenvolvimento serão alvo desse estudo.

A pesquisa desenvolvida caracteriza a fundamentação para o desenvolvimento de um framework que inclui práticas para o desenvolvimento de linha de produto com métodos ágeis. A metodologia proposta será aplicada em um exemplo para que possa ser avaliada a sua utilização.

## *1.1 Objetivos*

### 1.1.1 Objetivo Geral

O objetivo geral deste trabalho é apresentar uma proposta de adaptação do Framework de Métodos Ágeis[1] para dar suporte ao desenvolvimento de linhas de produto de software.

### 1.1.2 Objetivos Específicos

- Identificação das etapas de desenvolvimento de uma linha de produto de software;

[1] Neste trabalho, o termo "Framework de Métodos Ágeis" referencia o framework para seleção de práticas ágeis proposto em [Fagundes, 2005].

- Inclusão das etapas essenciais de desenvolvimento de uma linha de produtos no Framework de Métodos Ágeis, resultando em um framework estendido.
- Validação das etapas do framework estendido através de um exemplo de uso.

Os próximos capítulos descrevem o estudo que está sendo realizado. O capítulo 2 trata-se de uma abordagem teórica dos conceitos e tópicos relacionados. Cada assunto recebe uma breve descrição para contextualização e compreensão que servirá de base para o entendimento da proposta. Além disso, demonstra como são feitos ao passos para o desenvolvimento de uma linha de produtos. Todas as atividades e práticas desse assunto serão abordadas apresentando seus conceitos e características.

Também recebem destaque os trabalhos que apresentam um foco semelhante a este. No capítulo 3 serão descritos os trabalhos relacionados onde algumas publicações que apresentam objetivos semelhantes aos desse trabalho serão retratadas.

A proposta deste documento será apontada e descrita no capítulo 4. Todo esse embasamento realizado irá proporcionar uma metodologia que permita o desenvolvimento de forma ágil de uma linha de produtos.

Por fim, a prática do framework estendido será avaliada através de um exemplo de sua aplicação. O capítulo 5 mostrará toda essa etapa de forma clara e simples. As conclusões desse relatório são apresentadas logo após.

# 2 Fundamentação teórica

Neste capítulo é feito o detalhamento dos assuntos relacionados com o trabalho desenvolvido. Serão abordados cada um dos temas que serviram de base durante o estudo realizado. Cada um deles será detalhado de maneira que permita a compreensão do assunto e exposição dos seus conceitos e ideias principais.

## *2.1 Métodos Ágeis*

Os métodos ágeis tiveram origem a partir de 1990 e surgiram em oposição aos métodos tradicionais de desenvolvimento de software que se praticava na época. A comunidade de software começava a questionar a efetividade dos métodos tradicionais no desenvolvimento dos projetos de software e a dificuldade de serem praticados.

Na indústria de software, os problemas ganham ainda mais proporção pelo fato de ocorrerem muitas modificações e com certa frequência. Quando se trabalha diretamente com o cliente, é comum que nas fases iniciais ele tenha dificuldade para definir suas reais necessidades e, portanto, os requisitos do software, o que acaba comprometendo todo o processo de desenvolvimento tradicional [HIGHSMITH, 2002].

Os métodos ágeis tiveram reforço com a publicação do manifesto ágil. Um grupo compreendido de 17 analistas e consultores de software resolveram por em prática e divulgar suas ideias em Utah em fevereiro de 2001. O manifesto de desenvolvimento ágil, em suas práticas e proposições, procura trabalhar em torno de quatro grandes valores que estão descritos a seguir [MANIFESTO, 2001]:

- Indivíduos e iteração entre eles - mais que processos e ferramentas;
- Software em funcionamento - mais que documentação abrangente;
- Colaboração com o cliente - mais que negociação de contratos;
- Responder a mudanças - mais que seguir um plano.

Como uma alternativa para divulgar e facilitar a compreensão dessas ideias, os membros desse grupo, elaboraram 12 princípios que estão presentes em sua filosofia [MANIFESTO, 2001]. São eles:

- Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor.
- Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.
- Entregar software funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos.
- Pessoas relacionadas a negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.
- Construir projetos ao redor de indivíduos motivados, dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.
- O método mais eficiente e eficaz de transmitir informações para um time de desenvolvimento, e por dentro dele, é através de uma conversa cara a cara.
- Software funcional é a medida primária de progresso.
- Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter, indefinidamente, passos constantes.
- Contínua atenção à excelência técnica e bom design, aumenta a agilidade.
- Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.
- As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis.
- Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.

Como principais diferenças entre os métodos ágeis e os métodos tradicionais de desenvolvimento de software podem ser citadas [FOWLER, 2005; CHIN, 2004]:

- Os métodos ágeis são adaptativos e não preditivos: diferentemente dos enfoques tradicionais que defendem o planejamento integral do escopo no início do projeto e um controle rígido de mudanças, os planos dos métodos

ágeis são elaborados e adaptados ao longo do projeto, permitindo, e algumas vezes estimulado, a incorporação das mudanças requeridas pelo cliente;

- Os métodos ágeis são orientados a pessoas e não a processos: os métodos tradicionais de desenvolvimento de software têm, em geral, a pretensão de funcionar independentemente de quem os executa. Já os métodos ágeis levam em consideração as pessoas, sendo elaborados para auxiliá-las.

Nas próximas duas seções, estão exemplificados o Scrum e o Extreme Programming, que são métodos ágeis com destaque no cenário atual de desenvolvimento de software.

### 2.1.1 Scrum

Desenvolvido e mantido por Ken Schwaber e Jeff Sutherland, o Scrum é um framework de desenvolvimento ágil bastante conhecido [SCHWABER e SUTHERLAND, 2011]. Segundo eles, a utilização desse método permite a solução de problemas adaptativos garantindo a produtividade e o valor do produto. Considerado leve e de fácil entendimento, o Scrum é usado há vinte anos no gerenciamento e desenvolvimento de projetos incluindo práticas simples de gerenciamento e desenvolvimento de software. Partindo de teorias empíricas, o Scrum se utiliza da experiência como base do conhecimento para a tomada de decisões. Previsibilidade e controle dos riscos serão alcançados de forma incremental [SCHWABER e SUTHERLAND, 2011].

Uma definição da Aliança Scrum [SCRUM ALLIANCE, 2011] determina que o Scrum é um framework ágil para a execução de projetos complexos. Diz ainda que funciona para qualquer escopo, apesar de ter sido desenvolvido para projetos de software. São inúmeras a sua utilização em trabalhos inovadores apesar do framework ser enganosamente simples.

O controle do processo do Scrum ocorre em torno de três aspectos [SCRUM ALLIANCE, 2011]:

- Transparência – os responsáveis pelo projeto devem não somente conhecer os aspectos significativos do processo como também compartilharem um padrão comum de entendimento. Ou seja, os aspectos importantes devem ser de conhecimento comum e de mesma interpretação por todos os responsáveis da equipe.

- Inspeção – os elementos do processo e o andamento das atividades devem ser inspecionados frequentemente por alguém capacitado. O objetivo dessa medida é a detecção de variações inesperadas.
- Adaptação – caso seja percebido algum tipo de desvio no processo de desenvolvimento, o mesmo deve ser corrigido o mais breve possível.

Schwaber e Sutherland [SCHWABER e SUTHERLAND, 2011] mostram de que maneira essas inspeções e adaptações ocorrem. Serão descritas na sequência e são conhecidas como:

- Reunião de planejamento do Sprint;
- Reunião diária do Scrum;
- Reunião de revisão do Sprint;
- Retrospectiva do Sprint.

Antes de caracterizar cada uma delas, é importante esclarecer que o termo sprint se refere a um período de tempo necessário para a realização de uma versão funcional do produto. Normalmente sua duração não ultrapassa um mês, pois em períodos maiores que este podem ocorrer mudanças de planejamento, ocasionando o aumento do risco e da complexidade. Um sprint pode ser visto como uma iteração do desenvolvimento de software.

#### 2.1.1.1 Reunião de planejamento do Sprint

É o momento em que toda a equipe se reúne para definir o trabalho a ser executado. Deve ser discutido o que será entregue no próximo sprint e como deve ser realizado.

#### 2.1.1.2 Reunião Diária do Scrum

São reuniões curtas, normalmente de 15 minutos, feitas diariamente pela equipe de desenvolvimento. Durante esse tempo, é possível sincronizar o que já foi feito e planejar o dia seguinte: O que foi feito? O que será feito até o próximo encontro? E quais as dificuldades? São questões que devem ser respondidas para melhorar a comunicação, identificar e resolver problemas no desenvolvimento, e permitir uma tomada rápida de decisões, sempre com o intuito de garantir que o objetivo da tarefa seja alcançado.

#### 2.1.1.3 Reunião de revisão do Sprint

Realizada no final do sprint para verificar o que foi adicionado ao produto, esta reunião indica ao cliente o que está concluído. Podem ser discutidos os problemas superados pela equipe além de dúvidas que surgem ao longo do encontro. Com esta reunião, o dono do produto tem a possibilidade de acompanhar o andamento do desenvolvimento e rever o término das atividades. Ao final dessa etapa surgem as primeiras ideias para a próxima reunião de planejamento.

#### 2.1.1.4 Retrospectiva do Sprint

Deve ser feito para que a equipe possa propor melhorias para o próximo sprint baseado no que já foi feito. Podem ser revisadas as relações entre as pessoas, processos e ainda as ferramentas utilizadas. Garantindo assim a visualização do que já foi bem feito e o que pode ser reaplicado. Por consequência, um plano de melhorias é alcançado aumentando a qualidade do produto e proporcionando a adaptação da equipe.

A Figura 1, apresentada a seguir, mostra uma visão geral do funcionamento do Scrum, onde o projeto é dividido em partes menores (funcionalidades) incluídas no product backlog (repositório com todas as funcionalidades, características, tecnologias e bugs pendentes). Cada uma dessas funcionalidades farão parte de um sprint que entra em produção. O sprint é destacado com um ciclo de 2 a 4 semanas e no decorrer destes ciclos o produto final é alcançado.

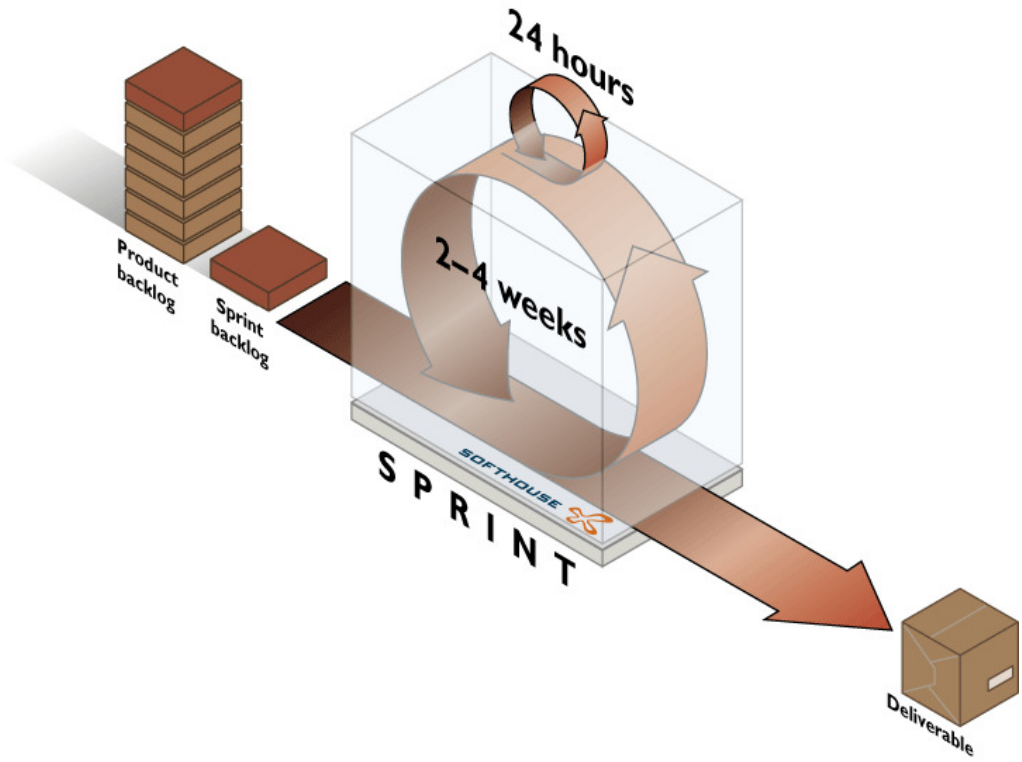



**Figura 1 - Processo do Scrum [SANCHEZ, 2007]**

### 2.1.2 Extreme Programming (XP)

O Extreme Programming (XP) também é um processo ágil de desenvolvimento. Teve início em 1996 e desde então vem sendo utilizado em diferentes organizações pelo mundo. Esse método busca principalmente a satisfação do cliente. O XP realiza as entregas ao longo do processo de desenvolvimento, assim o cliente vai conhecendo e utilizando o software, ao invés de deixar tudo para o final.

O XP é voltado para projetos com requisitos vagos e sujeitos a constantes mudanças, pode ser utilizado em sistemas orientados a objetos, em equipes de até 12 desenvolvedores e com desenvolvimento incremental, onde as funcionalidades vão sendo agregadas ao longo do tempo [TELES, 2004].

O trabalho em equipe é outro ponto importante do processo. Gerentes, desenvolvedores e cliente fazem parte da mesma equipe colaborativa. Durante o processo de desenvolvimento, a cada entrega o cliente pode fornecer uma opinião para a equipe de desenvolvimento que realiza as alterações de acordo com a demanda. Essa troca de informações gera um mecanismo onde o cliente apresenta as suas prioridades e, em resposta, a equipe de desenvolvimento dá mais atenção àquilo que ele precisa, contemplando um conjunto de funcionalidades de cada vez, conforme a demanda. Essas manutenções contínuas podem ocasionar falhas àquilo

que já existe, portanto a equipe deve estar atenta e disposta a enfrentar esse tipo de risco para que o projeto evolua com agilidade e segurança [TELES, 2004].

Um projeto XP reúne características como: simplicidade, comunicação, feedback, respeito e coragem. Pequenos objetivos são estabelecidos entre os membros da equipe até que seja alcançado o objetivo final. Sendo assim é possível manter a equipe produtiva mesmo que haja necessidade de alterações. A Figura 2 apresenta um fluxograma que demonstra o trabalho entre desenvolvedores e clientes atuando juntos. Nele o gerente acaba acompanhando a comunicação e relacionamentos durante o trabalho [EXTREME PROGRAMING, 2011]. No desenvolvimento de um projeto descrito pela figura, o XP prioriza os requisitos a serem implementados dando origem a um plano de iteração. Este plano entra num ciclo de desenvolvimento da equipe onde será desenvolvido o software.

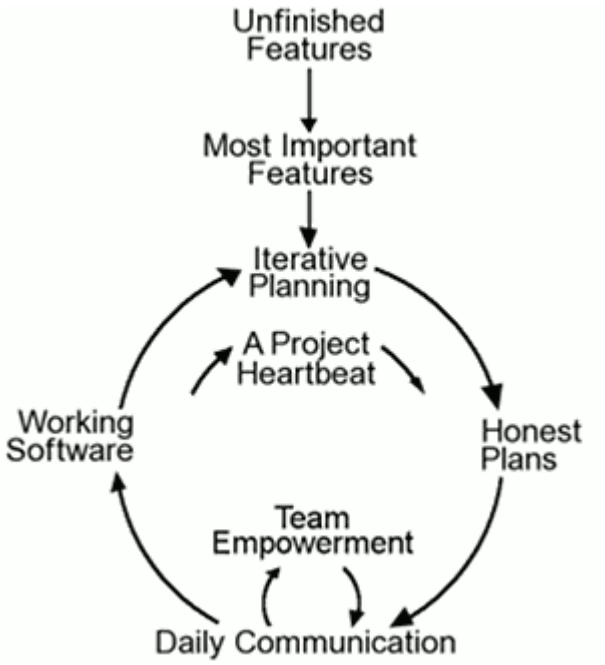



**Figura 2 - Processo XP [EXTREME PROGRAMING, 2011]**

A seguir estão descritas as práticas do XP [EXTREME PROGRAMING, 2011].

#### 2.1.2.1 Planejamento

**Histórias de usuários**

É uma descrição feita pelo cliente das coisas que o sistema precisa fazer. A partir dela os desenvolvedores definem o tempo para desenvolver cada uma dessas histórias.

**Planejamento e cronograma**

Deve ser feita uma reunião de planejamento para criar um plano global do projeto. Na reunião de planejamento será estimada a duração, em semanas, de cada história de usuário sem incluir dependências, apenas testes. A ordem das tarefas é determinada pela prioridade que o cliente decide para cada história. Além disso, podem ser determinadas quais histórias de usuários vão ser implementadas em cada versão do sistema e as datas para os lançamentos.

**Lançamentos pequenos**

Devem ser liberadas pequenas versões para o cliente a cada iteração sempre que possível.

**Projeto iterativo**

O desenvolvimento iterativo traz agilidade ao processo de desenvolvimento. Recomenda-se que o projeto seja dividido em pequenas iterações com duração de uma a três semanas.

**Planejamento a cada iteração**

Uma reunião de planejamento deve ocorrer a cada iteração produzido o plano de tarefas. As histórias de usuário que farão parte da iteração, são escolhidas pelo cliente. Além disso, os testes de aceitação já podem ser definidos.

#### 2.1.2.2 Gerência

**Espaço de trabalho aberto**

A comunicação é muito importante para uma equipe XP. Colocar os computadores em uma área central para a programação incentiva as pessoas a trabalhar em conjunto. A opção de mesas em torno do perímetro proporciona um lugar para trabalhar sozinho sem se desligar do resto da equipe.

**Definir um ritmo sustentável**

Para definir o ritmo, o final da iteração deve ser levado a sério. Caso não haja possibilidade de terminar dentro do prazo, uma reunião de planejamento da iteração deve redefinir o escopo.

**Reunião em pé**

O objetivo dessa reunião a cada dia é a comunicação entre toda a equipe onde são discutidos os problemas, soluções e promove foco para a equipe. Os participantes devem ficar de pé em círculo para evitar longas discussões.

**Medida da velocidade do projeto**

A velocidade de projeto é uma medida de quanto trabalho está sendo feito. Para medir a velocidade de projeto são somadas as estimativas de tempo das histórias de usuários da iteração, e também as estimativas para as tarefas concluídas durante a iteração. Esta medição é utilizada para o planejamento iteração.

**Mover as pessoas ao redor**

Mover as pessoas dentro da equipe evita perda de conhecimento e gargalos de codificação.

**Fixar o processo quando ele quebra**

Não se deve hesitar em mudar caso o processo não funcione bem. De qualquer forma as regras devem ser seguidas até que a equipe mude. Todos os desenvolvedores devem saber exatamente o que esperar do outro, tendo um conjunto de regras é a única maneira de definir essas expectativas.

#### 2.1.2.3 Projeto

**Simplicidade**

Um projeto simples sempre leva menos tempo para terminar do que um problema complexo. Sempre que possível, um problema complexo deve ser substituído por algo simples.

**Metáfora do sistema**

Uma metáfora do sistema serve explicar o projeto para as novas pessoas, sem recorrer a documentos enormes.

**Cartões CRC**

O uso de cartões de Classe, Responsabilidades e Colaboração (CRC) para projetar o sistema permite que as equipes inteira contribua para o projeto. Quanto

mais as pessoas podem ajudar a projetar o sistema maior o número de boas idéias incorporadas.

**Reduzir o risco**

Criar soluções para difíceis problemas técnicos ou de projeto sempre que uma dificuldade técnica ameaça o desenvolvimento do sistema. Um par de desenvolvedores deve trabalhar o problema por uma semana ou duas e reduzir o risco potencial.

**Não adicionar funcionalidades antecipadamente**

Deve-se tomar cuidado para não adicionar funcionalidades mesmo sabendo exatamente como adicioná-la ou porque tornaria o sistema melhor. Isto pode atrasar e desperdiçar recursos.

**Refatoração**

A refatoração deve ser feita sempre que necessário para manter o código limpo e conciso, para fácil compreensão, modificação e extensão.

#### 2.1.2.4 Codificação

**Cliente sempre disponível**

Um dos poucos requisitos do XP é ter o cliente disponível. Todas as fases de um projeto XP requerem comunicação com o cliente, de preferência no local.

**Normas de código**

O código deve ser formatado seguindo padrões de codificação que mantenha o código consistente e fácil para toda a equipe de ler e refatorar.

**Codificação de testes de unidade**

Criação de uma unidade de teste ajuda o desenvolvedor considerar realmente o que precisa ser feito.

**Programação em par**

Todo o código é criado por duas pessoas trabalhando juntas em um único computador. A programação em pares aumenta a qualidade do software, sem afetar o tempo de entrega.

**Apenas um par integra código de cada vez**

Integração estritamente seqüencial pelos próprios desenvolvedores, em combinação com a propriedade coletiva de código é uma solução simples para detectar problemas. Apenas um par de desenvolvedores integra, testa e confirma as alterações a qualquer momento.

**Integrar muitas vezes**

Os desenvolvedores devem estar integrando e compartilhando o código sempre que possível.

**Configurar um computador para integração**

Um único computador deve ser dedicado para sincronizar as alterações liberadas e assegurar a estabilidade entre a equipe.

**Propriedade coletiva**

A propriedade coletiva incentiva todos a contribuir com ideias novas para os segmentos do projeto. Qualquer desenvolvedor pode alterar o código para adicionar funcionalidades, corrigir problemas, melhorar o projeto ou refatorar.

#### 2.1.2.5 Testes

**Testes de unidade**

Os testes de unidade são de grande importância para o XP. Deve ser criado um quadro de teste de unidade automatizado. Posteriormente todas as classes do sistema devem ser testadas. Código sem testes não podem ser liberados.

**Correção de problemas**

Quando um problema é encontrado testes são criados para evitar que ele volte a ocorrer.

**Testes de aceitação**

São criados a partir de histórias de usuário selecionadas na iteração. Testes de aceitação são testes caixa preta e representam o resultado esperado do sistema.

## 2.2 Framework de Métodos Ágeis

O trabalho apresentado em [FAGUNDES, 2005] consiste em uma análise comparativa de diferentes métodos de desenvolvimento ágil e o desenvolvimento de um framework que reúne a práticas ágeis desses métodos comparados. Este framework permite a especificação de um processo ágil para quem pretende executar um projeto desse tipo, ressaltando as características comuns de diferentes métodos e uma coleção com as melhores práticas a serem implementadas. A partir deste framework, um subconjunto de práticas será selecionado para definir o novo processo ágil.

A Figura 3 abaixo apresenta uma visão geral do framework com o fluxo das atividades definidas e suas as iterações:

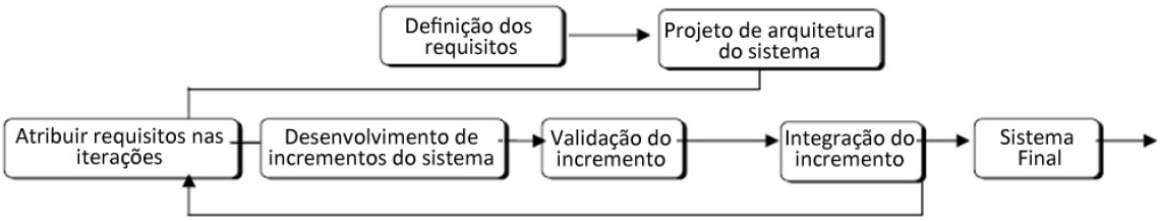



**Figura 3 - Fluxo de atividades do Framework de Métodos Ágeis [FAGUNDES, 2005]**

A proposta do framework reúne atividade que são sugeridas por diferentes métodos ágeis. São eles:

- XP,
- Scrum,
- Feature Driven Development,
- Adaptive Software Development,
- Agile Modeling,
- Dynamic Systems Development Method,
- Lean Software Development,
- Crystal Clear.

A seguir estão descritas as práticas do framework agrupadas em diferentes atividades.

### 2.2.1 Atividade: Definição dos requisitos.

**Lista de requisitos (Obrigatório)**

É o documento contendo os requisitos do sistema e suas prioridades. Cada requisito deve conter sua descrição, duração e os responsáveis pelo desenvolvimento. O cliente e programador estão envolvidos nessa tarefa.

**Modelo geral**

Em casos onde houver necessidade, o diagrama de classes pode ser agregado ao projeto para facilitar o seu entendimento. É responsabilidade do especialista de domínio.

**Documentação inicial**

Apresenta uma visão geral do sistema com a estimativa de custos atual, os benefícios esperados, riscos e recursos humanos. Além disso, pode reunir informações como os contatos dos principais envolvidos, tecnologias e ferramentas necessárias durante o desenvolvimento.

### 2.2.2 Atividade: Projeto de arquitetura do sistema

**Projeto de arquitetura do sistema**

Uma visão técnica da arquitetura do projeto. Os recursos utilizados nessa prática vão de acordo com a necessidade do projeto. Podem ser compreendidos por digramas de classe e diagramas de sequência. Os envolvidos para execução dessa prática são programadores e gerente de projeto.

### 2.2.3 Atividade: Atribuir requisitos nas iterações

**Planejamento da iteração (Obrigatório)**

É feita a distribuição dos requisitos dentro de cada iteração considerando suas prioridades, riscos e dependências. A cada iteração deve ser feita uma reunião para planejar o desenvolvimento. Havendo necessidade, o cliente pode estar presente nessa reunião. A duração de cada iteração apresentada nos métodos ágeis que originaram o framework sugere o tempo de uma a oito semanas, cabendo à equipe definir o período mais adequado a cada iteração de desenvolvimento. Ainda deve ser feita a distribuição de cada requisito para um desenvolvedor da equipe. É indicado o

tamanho máximo de seis membros e um responsável por equipe. Os envolvidos são programadores, programador chefe e o gerente de projetos.

### 2.2.4 Atividade: Desenvolvimento de incrementos do sistema

**Reuniões diárias**

Estas reuniões rápidas, realizadas a cada dia de trabalho, tem duração de 15 a 30 minutos. Nelas os participantes informam ao grupo o progresso de suas atividades e dificuldades encontradas. Os papéis relacionados são programadores e gerente de projetos.

**Histórias do usuário**

O cliente descreve as histórias dos usuários para cada funcionalidade do sistema detalhando a prioridade de cada uma delas. Os papéis envolvidos são clientes e programadores.

**Casos de uso**

Cada requisito do sistema que será desenvolvido na iteração, deve corresponder a um caso de uso detalhando a funcionalidade. Os envolvidos também são clientes e programadores.

**Projeto da iteração**

Consiste em um projeto de sistema baseado nos requisitos conhecidos da iteração corrente. Podem-se utilizar diagramas UML, como por exemplo, o diagrama de classes, para representar o projeto de toda a iteração, ou até mesmo algum outro tipo de diagrama.

**Desenvolvimento da iteração (Obrigatório)**

Consiste na codificação dos requisitos que fazem parte da iteração. Devem ser utilizados padrões para estruturação de código e um sistema de versões. Os envolvidos são os programadores.

**Escrita dos testes de unidade**

Sugere o desenvolvimento dos testes de unidade antes da codificação. Os papéis relacionados são cliente e programadores.

**Escrita dos testes de aceitação**

Também sugere o desenvolvimento de testes de aceitação anteriores à codificação. São envolvidos o cliente e os programadores.

**Desenvolvimento de código coletivo**

Toda a equipe fica responsável pelo desenvolvimento, portanto é necessário que todos entendam e conheçam o que está sendo feito. As partes envolvidas são os programadores.

**Programação em pares**

Sugere que o desenvolvimento seja feito por um par de programadores. Nesse modelo, um desenvolvedor pensa em como implementar e outro analisa o funcionamento do que está sendo feito, sempre pensando na simplificação do código. Os programadores são envolvidos.

**Refatoração**

Sempre que necessário o código deve ser refeito. É responsabilidade dos programadores.

**Programação lado-a-lado**

Esta prática indica uma configuração do ambiente de trabalho onde seja possível a visualização do monitor da dupla de desenvolvedores que está ao lado. Os programadores são envolvidos.

### 2.2.5 Atividade: Validação do incremento

**Inspeção de código**

Serve para a detecção de defeitos e revisão do código escrito. Os programadores devem inspecionar um o código do outro.

### 2.2.6 Atividade: Integração do incremento

**Reunião de revisão da integração**

Ao final de cada iteração, deve ser feita uma reunião para que a equipe possa avaliar a iteração que foi concluída. As partes envolvidas são os programadores e gerente de projetos.

### 2.2.7 Atividade: Sistema final

**Geração de uma breve documentação**

Caso a equipe não tenha feito isto ao longo do projeto, é importante que um breve documento seja feito. Uma documentação do usuário também é importante. As partes envolvidas são os programadores e gerente de projetos.

**Entrega do sistema ao cliente**

Caso não exista mais alguma exigência após a implantação, uma nova entrega deve ser feita ao cliente. De qualquer forma, uma reunião pode ser feita para o reconhecimento do final do projeto. Essa etapa diz respeito a todos os interessados.

A tabela apresentada a seguir retrata as dependências de cada prática do framework. Para aquelas práticas que são obrigatórias, não é apresentada na tabela esta dependência.

A letra "E" representa as práticas excludentes. Quando há a ocorrência de uma letra indica que a prática apresentada na linha está relacionada com a prática apresentada na coluna.

**Tabela 1 - Práticas ágeis excludentes.**

| | Lista de requisitos (O) | Modelo geral | Documentação inicial | Projeto da arquitetura do sistema | Planejamento da iteração (O) | Reuniões diárias | Histórias de usuários | Casos de uso | Projeto da iteração | Desenvolvimento da iteração (O) | Escrita dos testes de unidade | Escrita dos testes de aceitação | Desenvolvimento de código coletivo | Programação em pares | Refatoração | Programação lado-a-lado | Inspeção de código | Reunião de revisão da iteração | Breve documentação | Entrega do sistema |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Lista de requisitos (O) | | | | | | | | | | | | | | | | | | | | |
| Modelo geral | | | | | | | | | | | | | | | | | | | | |
| Documentação inicial | | | | | | | | | | | | | | | | | | | | |
| Projeto da arquitetura do sistema | | | | | | | | | | | | | | | | | | | | |
| Planejamento da iteração (O) | | | | | | | | | | | | | | | | | | | | |
| Reuniões diárias | | | | | | | | | | | | | | | | | | | | |
| Histórias de usuários | | | | | | | | **E** | | | | | | | | | | | | |
| Casos de uso | | | | | | | **E** | | | | | | | | | | | | | |
| Projeto da iteração | | | | | | | | | | | | | | | | | | | | |
| Desenvolvimento da iteração (O) | | | | | | | | | | | | | | | | | | | | |
| Escrita dos testes de unidade | | | | | | | | | | | | | | | | | | | | |
| Escrita dos testes de aceitação | | | | | | | | | | | | | | | | | | | | |

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Desenvolvimento de código coletivo | | | | | | | | | | | | | | | | | | | |
| Programação em pares | | | | | | | | | | | | | | | E | | | | |
| Refatoração | | | | | | | | | | | | | | | | | | | |
| Programação lado-a-lado | | | | | | | | | | | | | **E** | | | | | | |
| Inspeção de código | | | | | | | | | | | | | | | | | | | |
| Reunião de revisão da iteração | | | | | | | | | | | | | | | | | | | |
| Breve documentação | | | | **E** | | | | | **E** | | | | | | | | | | |
| Entrega do sistema | | | | | | | | | | | | | | | | | | | |

## 2.3 Linhas de Produtos

Linha de produto de software (LP) é uma técnica de engenharia de software para a criação de um portfólio de sistemas de software similares a partir de uma base compartilhada de componentes e modo produção potencialmente semelhante [KRUEGER, 2011]. A ideia consiste na utilização de artefatos genéricos que se aplicam a diferentes produtos que contemplam um determinado escopo. Dessa forma é possível alcançar uma série de melhorias durante o seu desenvolvimento como o tempo, custos e qualidade do que está sendo feito.

As LPs de Software são um paradigma de desenvolvimento que cresce a cada dia em empresas que buscam uma melhoria no tempo de desenvolvimento, redução de custos, aumento na produtividade e a qualidade de seus produtos. Com esta técnica é possível ainda o ingresso no mercado de uma forma rápida e flexível com a característica de customização em massa [SEI, 2011].

O que difere uma LP de um reuso realizado de maneira oportunista, é o fato de que as LPs de software se beneficiam da reutilização de componentes de uma forma planejada e direcionada para uma parcela que contemple a maioria dos produtos do escopo. Uma produção em massa representa o grande avanço para a indústria de desenvolvimento de software que essa abordagem proporciona. Apesar disso, a customização dos produtos também é levada em consideração representando um diferencial nesse tipo de produção. Isso é garantido pela seleção e gerência das características em comum dos produtos no momento do seu planejamento.

Segundo o Software Engineering Institute (SEI), uma LP pode resolver os problemas de escassez de recursos em uma organização. O tamanho de uma organização não é impedimento para a implantação de uma LP. Eles estimam que a execução de uma LP melhora a produtividade e a qualidade em até 10 vezes [SEI, 2011].

As vantagens da implementação de uma LP de software podem ser observadas em diversas áreas do desenvolvimento [KRUEGER, 2011]:

- Redução do tempo de implantação;
- Redução dos problemas em cada produto;
- Redução do esforço de implantação de um novo produto;
- Aumento da área de atuação com a criação de novos produtos;
- Competitividade de diferentes produtos;
- Aumento na qualidade dos produtos desenvolvidos;
- Melhora no modelo de negócios e de mercado;
- Redução do risco ao atingir novos mercados.

Uma abordagem mostrando detalhes de como isso pode ser feito e implementado dentro de uma organização pode ser vistos na sequência com a apresentação de um framework que retrata os passos e modelos que um projeto de software deve seguir para o desenvolvimento de uma arquitetura de LP.

## 2.4 Framework de Linhas de Produtos

Esta seção irá descrever os conceitos e sugestões do Framework para Linha de Produtos de Software [NORTHROP e CLEMENTS, 2007], proposto por Linda M. Northrop e Paul Clements, pesquisadores na área integrantes do SEI.

O framework reúne informações para auxiliar nos esforços de desenvolvimento de software em uma LP, com o objetivo de identificar os conceitos fundamentais e premissas para o desenvolvimento. Para isto, é preciso conhecer a área de atuação para o qual o software está sendo desenvolvido e só então é possível determinar as práticas a serem utilizadas. Cada LP exigirá diferentes práticas que variam de acordo com o tipo de sistema, profundidade do domínio, dos bens legados, objetivos da organização, maturidade do processo existentes, habilidade dos envolvidos, estratégia de produção, além de muitos outros fatores.

Consequentemente, não existe um conjunto de práticas corretas ou específicas. Portanto, o framework não deve ser tratado como um guia de

programação, mas sim como um conjunto de práticas que devem ser utilizadas de acordo com a necessidade e adequação.

Uma característica essencial de atuação em LPs é o desenvolvimento de um núcleo de componentes e o desenvolvimento de produtos a partir desse núcleo. Estas duas atividades podem ocorrem em ordens distintas ou até mesmo em paralelo. Além disso, todo o processo deve ser gerenciado, para isto existem práticas específicas. De maneira geral, a Figura 4 caracteriza cada uma das atividades essenciais que o framework propõe:

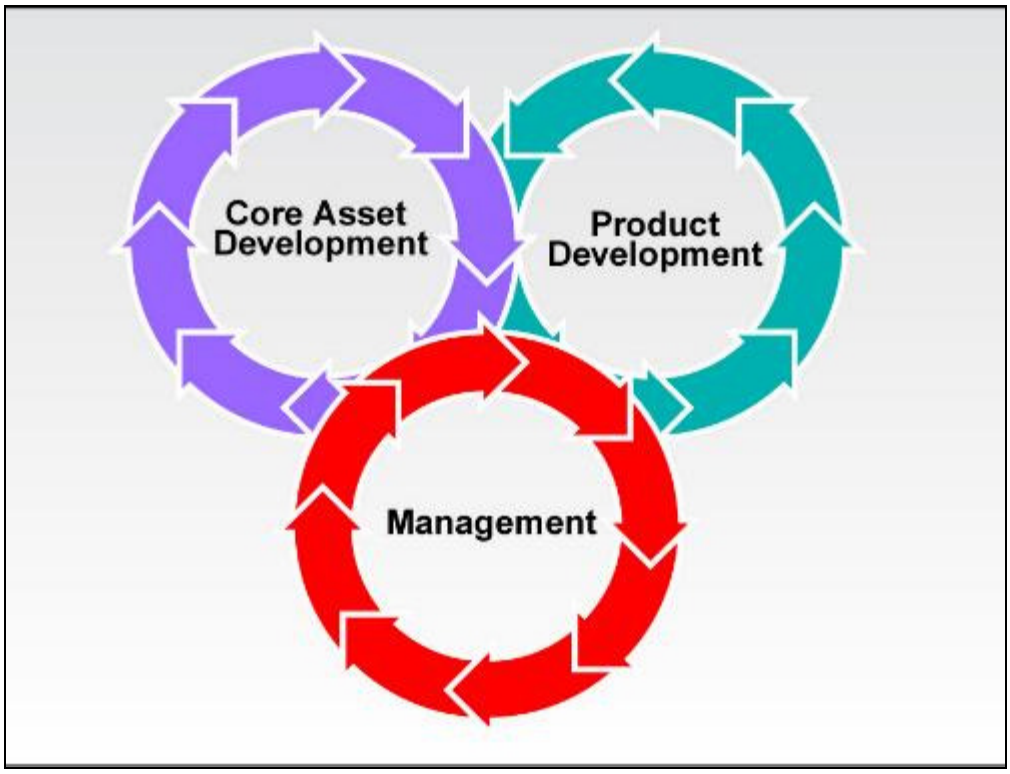



**Figura 4 - Atividades essenciais [NORTHROP e CLEMENTS, 2007]**

É possível observar que as atividades, mesmo estando relacionadas entre si, permanecem em constante desenvolvimento. Cada uma delas é imprescindível e altamente iterativa, porém a relação entre si permite seu refinamento constante e em ordem qualquer. Também pode ser destacado o fato de não existir uma ordem de execução nas atividades de desenvolvimento dos produtos. Nada impede que a partir de produtos existentes sejam extraídas as informações de requisitos genéricos e de arquitetura para o desenvolvimento do núcleo de uma LP. Dessa maneira, a realização dos produtos pode ser monitorada e servir como entrada para realimentar o núcleo ativo da LP.

O chamado núcleo de componentes ativos no desenvolvimento da LP tem a função de reunir as informações necessárias para a produção. O contexto em que está inserida essa produção acaba variando de acordo com alguns fatores:

- Restrições do produto – Como o nome sugere, devem ser levantadas informações de cada um dos produtos que serão desenvolvidos. Características em comum, dados comportamentais, tecnologias, informações de mercado são exemplos de conceitos que precisam ser consideradas.
- Restrições de produção – de maneira similar, a produção também precisa ser inspecionada. Destacam-se características como o tempo para entrar no mercado, normas que devem ser seguidas durante o processo e expectativas de mercado por exemplo.
- Estratégia de produção – deve ser abordado, nesse ponto, qual abordagem se adéqua à produção. Essa pode ocorrer de maneira proativa, reunindo as características do núcleo da LP partindo para os produtos; ou de forma reativa, tendo um conjunto de produtos e a partir deles generalizando uma linha de produção.
- Componentes pré existentes – a utilização de sistemas legados ou arquiteturas existentes acabam agilizando e até contribuindo com a qualidade do produto. Nessa etapa, é feita uma previsão dos componentes que podem ser reutilizados por completo, partes deles ou ainda uma adequação para a necessidade existente.

Tendo em mãos o resultado dessas etapas e definida a área de atuação do produto, ou seja, tendo conhecido o domínio que atuam as organizações em que a LP está inserida, está feita a análise inicial do desenvolvimento. A partir daí, utiliza-se as práticas sugeridas pelo framework. A seguir, um resumo de cada uma delas é apresentando, destacando seus conceitos e facilitando a sua compreensão. Para cada uma destas práticas ou, de acordo com o projeto, para o conjunto de algumas delas pode ser feita uma iteração dentro do projeto.

### 2.4.1 Definição da arquitetura

É o momento em que deve ser definida a estrutura do sistema através dos componentes necessários e suas relações. As características de qualidade de software, relações com outros sistemas, os objetivos do software a ser desenvolvido e

as bibliotecas de componentes que estarão disponíveis, fazem parte dos requisitos que devem ser analisados e definidos nesta etapa.

A definição da arquitetura inclui a interface dos componentes. Em seguida, a ligação dos componentes é definida. Pressupondo as facilidades que a tecnologia atual permite, é natural que seja feita uma análise de todo esse funcionamento. Esta análise pode indicar as chamadas de procedimentos, protocolos de comunicação, persistência de objetos e padronização dos métodos.

A documentação ainda faz uma projeção da arquitetura e suas particularidades como uma visão geral. Deve apresentar a divisão da hierarquia do software, os processos e sua comunicação, a divisão de camadas desses processos e também projeção de hardware desse sistema.

Sem dúvida, essa arquitetura corresponde a uma abordagem conceitual com uma série de abstrações que, por consequência, permite variações em sua implementação. O objetivo dessa tarefa numa LP é justamente o fornecimento dessa grande quantidade de variações que, ao serem aplicadas, tornam-se novos produtos. Essa flexibilidade é desejável e fundamental para a atuação do arquiteto dessa LP. O mecanismo irá suportar a variabilidade dos produtos se manifestando de maneira que produtos diferentes tenham atributos de qualidade diferentes, por exemplo, a segurança e o desempenho de cada um. No nível de produção, também deve permitir diferentes abordagens de desenvolvimento de acordo com cada organização. A integração dos componentes de forma flexível e confiável é outra condição necessária.

### 2.4.2 Desenvolvimento de componentes

Os componentes de uma arquitetura de software são uma generalização em partes menores das peças que compreendem o sistema. O arquiteto de software deve definir essa lista de componentes para que as equipes possam desenvolver a partir deles os produtos da LP. Estes componentes são unidades que os programadores já utilizam em suas técnicas de reuso. Ao realizar esse tipo de prática, programadores ganham tempo de programação, além de ter uma garantia de funcionamento ao utilizar algo que normalmente já foi testado anteriormente.

O desenvolvimento de componentes pela própria equipe ao invés da utilização daqueles encontrados externamente, muitas vezes deve ser priorizado para manter as características que foram projetadas na arquitetura do projeto. Os

componentes que serão disponibilizados para o desenvolvimento de produtos devem sustentar a flexibilidade que uma LP exige na implantação de suas variantes.

### 2.4.3 Mineração de projetos existentes

Consiste na utilização de um sistema antigo ou parte dele no sistema em desenvolvimento mesmo que isto não tenha sido previsto. Isso justifica a utilização de modelos de negócios, regras de base, especificação dos requisitos, cronogramas, orçamentos e planos de desempenho anteriores a esse projeto. Todos estes também serão alvo de uma avaliação para que haja uma seleção daqueles que possam ser aplicados nessa nova LP.

A documentação pode passar despercebida, porém é viável a sua reutilização. Um vasto conhecimento em nível de corporação é disponibilizado nas antigas documentações, e no caso de sistemas com algum tipo de relação, isto pode ser aproveitado.

É preciso compreender o material que se encontra disponível. Muitas vezes o apoio de analistas que conheçam o antigo e o novo sistema é fundamental. O arquiteto do novo sistema é responsável pela seleção da base antiga e inserção na nova arquitetura. Para fazer parte do novo projeto, o material disponível precisa contemplar os novos requisitos de variabilidade, desempenho e confiabilidade.

Rastreadas as partes de projetos que interessam, a inclusão destes novos itens na LP pode se assemelhar a uma iteração de reengenharia ou apenas o desenvolvimento de uma nova parte da arquitetura. Isto varia de acordo com o projeto, o planejamento e coordenação adotados pela equipe envolvida nessa tarefa.

### 2.4.4 Engenharia de requisitos

O conjunto de requisitos determina as características do sistema, ou seja, determina o que o sistema deve fazer, seu comportamento, as propriedades que ele apresenta, suas qualidades e as restrições que deve satisfazer. A engenharia de requisitos é uma atividade que executa os seguintes passos:

- Elicitação de requisitos – é o processo onde as necessidades do usuário e do sistema são conhecidas, analisadas e documentadas.
- Análise de requisitos – é feito um estudo dessas necessidades a fim de refinar essas características e restrições.

- Especificação de requisitos – corresponde a uma documentação clara e precisa das necessidades do usuário e restrições do sistema.
- Verificação de requisitos – deve-se garantir que todos os requisitos do sistema foram listados corretamente e de forma clara e consistente.
- Gerenciamento de requisitos – as atividades definidas na iteração precisam ser agendadas, acompanhadas e também documentadas.

Essa prática envolve três partes da equipe de desenvolvimento: o cliente que expõe suas necessidades, porém não entende o sistema que está sendo desenvolvido; o desenvolvedor que tem o conhecimento técnico, mas pode não entender o que o cliente precisa; e, por último, o responsável da equipe por escrever a documentação que tem que relatar o desejo do cliente de acordo com a especificação técnica que está sendo formulada.

Certamente que requisitos conflitantes fazem parte dessa seleção de requisitos. Inevitavelmente um acordo entre as partes conflitantes deve sugerir mudanças ou acordos a fim de conciliar os requisitos do sistema. Tudo isso precisa estar muito bem documentado explicitando as decisões tomadas até o acordo final.

Os requisitos podem surgir ao longo da execução de todo o projeto e esse estudo inicial precisa ser bastante sólido alcançando o maior número de necessidades possíveis para que os esforços de desenvolvimento não sofram com mudanças bruscas no decorrer das atividades. Algumas tarefas, como a parte de testes, por exemplo, ficam diretamente ligadas com as diretrizes de qualidade e requisitos especificados nesse momento.

### 2.4.5 Integração do sistema de software

É a combinação dos componentes de software para serem testados individualmente e integrados ao sistema. Isto ocorre de acordo com o surgimento de partes intermediárias ou subsistemas do projeto entre o final da etapa de desenvolvimento dos componentes e os testes propriamente ditos. Essa integração contínua acaba sendo menos arriscada alcançando partes que vão sendo incrementadas até alcançar as versões maiores do projeto.

A integração deve ir além da compatibilidade entre os componentes e suas assinaturas, ela precisa garantir o seu bom funcionamento em conjunto com as demais partes do sistema. Essa ideia constitui a interface entre os componentes do

sistema. Quando bem definidos e documentados esses parâmetros do funcionamento dos componentes uns com os outros, a integração contínua acontece de maneira simples.

### 2.4.6 Testes

O objetivo dessa iteração é de identificar possíveis falhas que o sistema apresenta e que podem ser recuperadas, e verificar se o funcionamento está de acordo com a especificação. Em alguns casos é possível estimar a confiabilidade do software.

Os testes devem ser executados com as principais entradas do programa, aquelas que têm maior probabilidade de ocasionarem falhas e os erros que o usuário pode provocar. Diferentes tipos de testes podem ser aplicados no sistema e todos devem respeitar as etapas a seguir:

- Análise: o que está sendo testado precisa passar por uma análise prévia para identificar o que é mais apropriado e o que pode ser feito;
- Construção: cenários e artefatos precisam ser feitos para permitir a execução da cada teste;
- Execução e avaliação: feito o teste, o resultado obtido passa por uma análise e determina a tomada de decisão dependendo do julgamento.

O resultado dessa iteração pode ser mostrado em quatro diferentes tipos de elementos. São eles:

- Casos de teste: seleção dos testes a serem executados. De acordo com a cobertura e garantia que se deseja atingir, diferentes contextos devem ser analisados;
- Documento de testes: o plano de testes e o relatório com os resultados são os principais documentos obtidos nessa etapa;
- Conjunto de dados de teste: são os dados necessários para simulação e execução do teste para estabelecer a condição necessária para o seu exercício;
- Software de teste: algumas verificações precisão ser feitas de forma automatizada com auxílio de um software específico para aquela avaliação.

De acordo com o que está sendo desenvolvido, inúmeras são as verificações possíveis e testes que podem ser aplicados para a validação da tarefa. Entre as possíveis verificações podemos destacar: validação do projeto modelo, teste de unidade, teste de subsistemas de integração, testes de integração do sistema, testes de regressão, testes de conformidade, testes de aceitação, testes de implantação e modelos de confiabilidade.

Os testes compreendem uma atividade que é reutilizada por várias vezes ao longo dos produtos que são desenvolvidos. Um bom planejamento irá facilitar essa característica.

### 2.4.7 Compreensão dos domínios relevantes

Esta seção descreve os esforços envolvidos para o conhecimento do domínio em que o software a ser desenvolvido está inserido. Os domínios correspondem à especialidade da organização para qual o sistema está sendo desenvolvido. O conhecimento do domínio inclui uma série de conceitos e terminologias utilizados pelos profissionais da área. É comum a necessidade da utilização de vários domínios para um único sistema. A prática dessa iteração envolve itens como:

- Identificação das áreas de conhecimento úteis para o desenvolvimento;
- Identificar problemas e soluções conhecidas nesse domínio;
- Representar essas informações para que possam ser repassadas aos interessados no decorrer das tarefas.

Esse objetivo pode ser alcançado compartilhando experiências anteriores dos envolvidos ou, se preciso for, com a contratação de um perito para fazer esse papel.

Qualquer nível de informações alcançadas por essa iteração deve ser documentado para a tomada de decisões futuras. O desejável é que possam ser compreendidas as questões como as implicações técnicas e de negócios associadas ao projeto; recursos, capacidades e tecnologias que serão oferecidas nos produtos e disponibilização dos artefatos que exploram o domínio.

### 2.4.8 Softwares externos

A existência de softwares externos que já implementam a função desejada, é outra facilidade que pode ser aplicada no desenvolvimento da arquitetura. O que se está procurando pode surgir de diversas formas como a aquisição de um software comercial, utilização de um código aberto ou algo que esteja disponível na web em

arquiteturas orientadas a serviço. A utilização de códigos abertos é provavelmente o mais indicado pela sua flexibilidade. Deve ser avaliado o custo envolvido nessa aquisição e licenças necessárias.

O uso de softwares externos tem crescido constantemente no desenvolvimento de sistemas. Infraestrutura de comunicação, gerenciamento de redes, banco de dados são exemplos de domínios que se utilizam essa ideia. Apesar das diferentes fontes possíveis para essa prática, existem algumas características comuns a elas e que devem ser seguidas:

- Analisar a compatibilidade de arquitetura – devem ser avaliadas a flexibilidade e variabilidade do sistema para que nada seja comprometido;
- Entender os requisitos da organização – rever as restrições e normas que devem ser seguidos e se o software externo atende a essas necessidades;
- Estudar o mercado em detalhes – estar atento aos novos surgimentos e tecnologias que vão sendo disponibilizados e podem ser aplicados na LP;
- Desenvolver requisitos de forma flexível – muitas vezes os softwares não atendem exatamente os requisitos do projeto e a flexibilização destes facilita a tarefa de encontrar o que atenda a necessidade;
- Desenvolver a avaliação dos produtos e tecnologias – os critérios de avaliação destes produtos externos devem estar disponíveis para a sua aquisição;
- Seleção de produtos viáveis e tecnologias – corresponde a uma qualificação dos candidatos que possam cumprir a tarefa desejada;
- Compra dos produtos – políticas de compra devem ser definidas para evitar decisões precipitadas;
- Integração com a arquitetura – devem ser avaliadas as mudanças necessárias para a integração de um software com a arquitetura já existente;
- Teste de configuração – essa inclusão exige que novos testes de interface sejam elaborados;
- Gerenciamento do sistema de forma contínua – significa monitorar a estrutura atual e definir diretrizes que auxiliam na substituição destes componentes.

Cada uma das etapas descritas acima pelo framework recebe uma série de exemplificações e sugestões práticas que devido ao seu tamanho foram omitidas neste documento. Cabe ao projetista ou engenheiro de software responsável pelo projeto ir mais além na especificação de cada iteração utilizando ou refinando as técnicas que o framework tem de acordo com a sua necessidade e experiência nesse tipo de projeto.

Outras características relacionadas a um projeto de software, como gestão técnica e organizacional fazem parte da especificação do framework. Novamente não foram abordadas estas características, pois não são foco da proposta de framework e do exemplo de uso que serão descritos ao longo desse trabalho.

## *2.5 Desenvolvimento de uma LP*

Neste capítulo, serão descritos os passos práticos para o desenvolvimento de uma LP segundo a Tese de Doutorado de Eduardo Santana de Almeida [ALMEIDA, 2007]. Estes passos servirão de base para o desenvolvimento do framework proposto.

### 2.5.1 Engenharia de Domínio

A etapa selecionada para fazer parte deste capítulo trata-se da engenharia de domínio que é similar a etapa de desenvolvimento do núcleo de componentes descrita na seção anterior. A engenharia de domínio é definida por [ALMEIDA, 2007] como sendo "a atividade de coletar, organizar e armazenar a experiência passada em sistemas de construção ou peças de sistemas em um domínio particular sob a forma de artefatos reutilizáveis, bem como proporcionar um meio adequado para a reutilização desses artefatos na construção de novos sistemas"

Para realização dessa tarefa, três etapas serão seguidas:

- Análise de Domínio – definição dos requisitos reutilizáveis para o sistema em questão.
- Projeto de Domínio – desenvolvimento de uma arquitetura comum para o sistema e um plano de produtos.
- Implementação de Domínio – implementação do núcleo que será reutilizado.

Cada uma dessas três etapas será descrita a seguir em forma de tópicos que definem as partes necessárias para o seu desenvolvimento.

### 2.5.1.1 Análise de Domínio

A primeira atividade que caracteriza a análise de domínio, e será detalhada a seguir, é o plano de domínio.

#### *2.5.1.1.1 Plano de domínio*

Trata-se de uma fase de preparação e análise de uma possível infra-estrutura de reuso para certo domínio. Devem ser executadas as atividades:

- Análise das partes envolvidas – identificação das partes interessadas e o papel de cada um deles.
- Definir os objetivos – definição do desejo de cada um dos interessados com relação ao projeto.
- Definir restrições – definição das restrições da organização e restrições de mercado mostrando o que está fora do escopo.
- Análise de mercado – havendo a possibilidade e necessidade, é realizada uma pesquisa dos fatores externos que determinam o sucesso do domínio no mercado.
- Coleta de dados – reunir e analisar as informações do domínio em questão, como a documentação disponível e conhecimento de especialistas.

Depois de realizadas estas atividades, devemos identificar as características e definir o escopo de domínio da seguinte maneira:

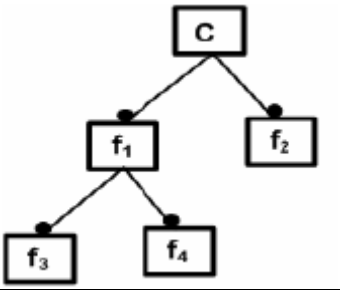
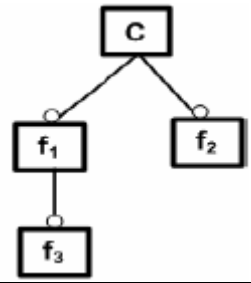
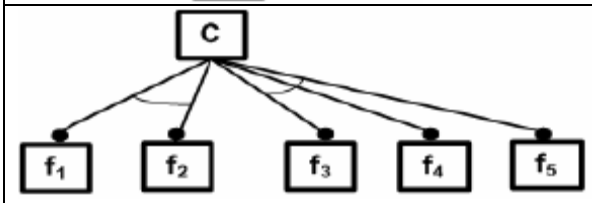
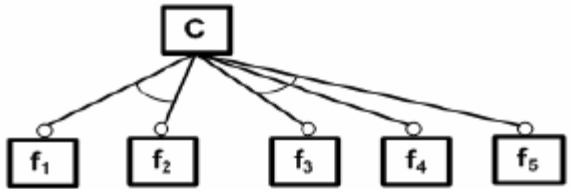
- Mapear as aplicações candidatas – identificar as características que serão definidas pela arquitetura de domínio que será desenvolvida posteriormente. Estas aplicações podem ser caracterizadas de três tipos: aplicações existentes são aquelas que foram desenvolvidas antes desse processo; aplicações futuras são aquelas que os requisitos são claros, porém ainda não foram desenvolvidas; e possíveis aplicações onde os requisitos não são claros, mas são consideradas relevantes.
- Desenvolver funções de avaliação – refinamento dos objetivos de negócio relevantes para o domínio levando em consideração as informações produzidas nas atividades anteriores.
- Caracterização das aplicações – aplicação de funções de caracterização para cada aplicação mapeada.

- Análise de benefícios – Com base nas informações que foram adquiridas acima é feita a definição do escopo.

#### *2.5.1.1.2 Modelo de domínio*

A sequência das atividades anteriores se dá com o desenvolvimento de um modelo para o domínio em análise. Significa uma mudança na direção das atividades para questões estruturais e conceituais. O modelo resultante desta etapa descreve a uniformização e a variabilidade do domínio. Cada recurso apontado na etapa anterior fará parte desse modelo como um nó em uma estrutura que mostra de forma gráfica a hierarquia do domínio. Na tabela a seguir (Tabela 2) serão apresentas algumas figuras que exemplificam esse modelo.

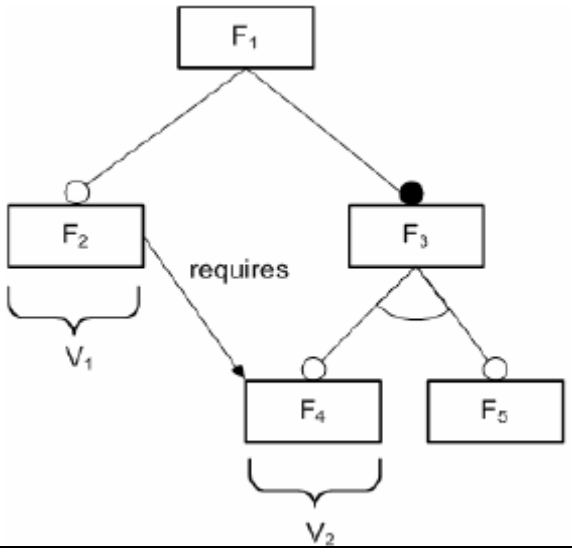
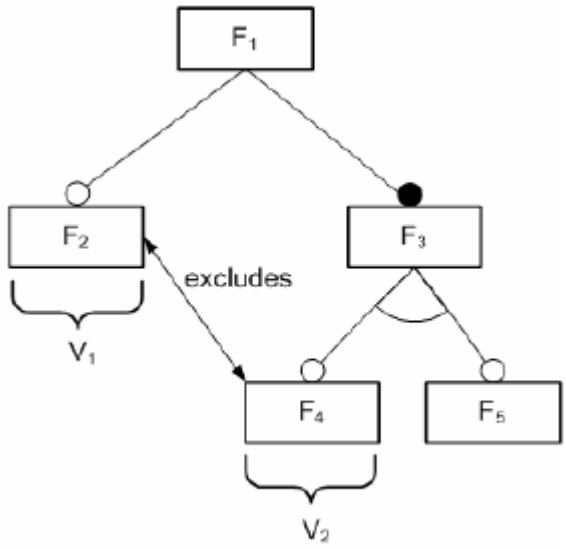
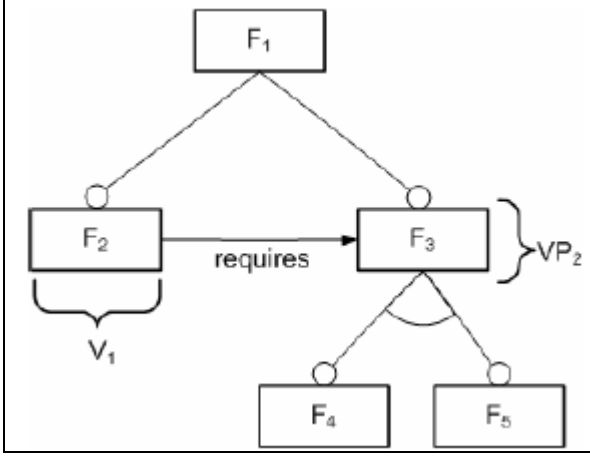
**Tabela 2 - Representação das características do modelo de domínio**

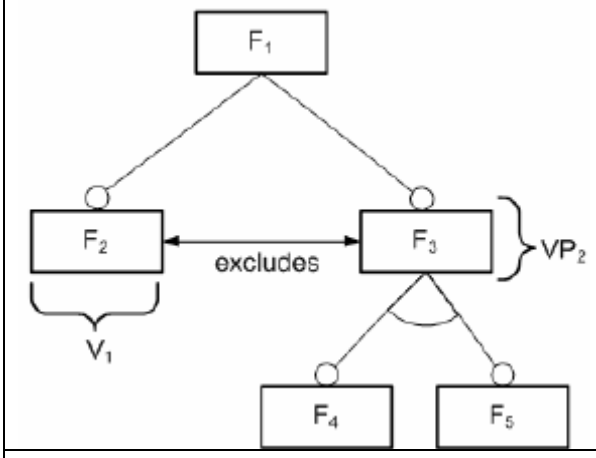
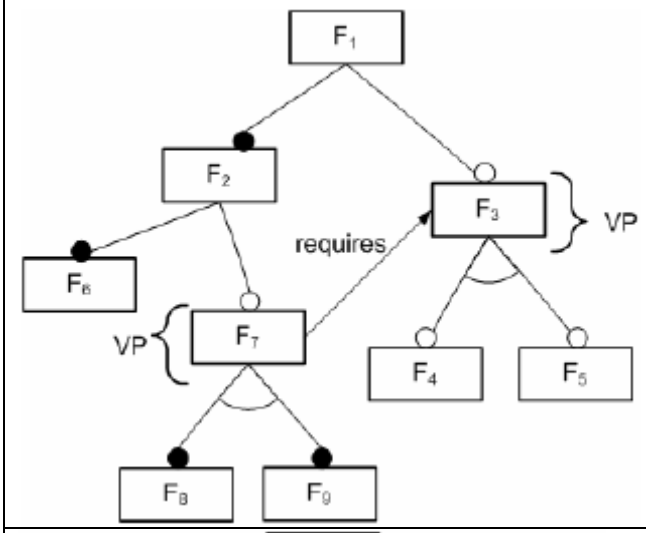
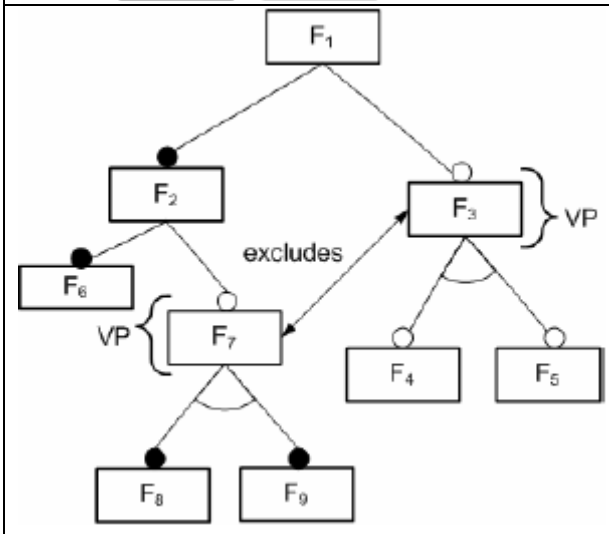
| Figura | Descrição |
|---|---|
|  | Característica obrigatória – Representa a ligação das características que compõe o nó acima. A figura indica que o nó C é composto pelo conjunto {C, f1, f2, f3, f4} e f1 é composto por {f1, f3, f4}. |
|  | Característica opcional - Representa a possibilidade de incluir uma característica apenas se o nó pai também for incluído. De acordo com a figura, uma instância de C pode ser: {C}, {C, f1}, {C, f1, f3}, {C, f2}, {C, f1, f2} ou {C, f1, f2, f3} |
|  | Característica alternativa - Representa um conjunto de características onde apenas uma pode ser incluída na descrição. A instância de C ao lado pode derivar: {C, f1, f3}, {C, f1, f4}, {C, f1, f5}, {C, f2, f3}, {C, f2, f4} ou {C, f2, f5} |
|  | Característica ou - Representa um conjunto de características onde um nó pode ter um ou mais recursos desse conjunto. Qualquer combinação sem repetição com pelo menos um elemento de cada conjunto: {f1, f2} + {f3, f4, f5} é uma representação válida de C. |

Além disso, algumas regras complementam o modelo com as informações de dependência mútua e exclusão mútua entre as características opcionais do modelo. Estas características opcionais são chamadas de variantes quando são nós folhas do

modelo e pontos de variação quando são nós pai e/ou avós. Abaixo, a Tabela 3 mostra um exemplo de cada uma dessas representações:

**Tabela 3 - Representação das regras do modelo de domínio**

| Figura | Descrição |
|---|---|
|  | Variante requer variante – quando a seleção de V1 exige a seleção de V2 |
|  | Variante exclui variante – quando a seleção de V1 exclui a seleção de V2 |
|  | Variante requer ponto de variação – quando a seleção de V1 exige a seleção de um ponto de variação VP2 |

| | |
|---|---|
|  | Variante exclui ponto de variação – quando a seleção de V1 exclui a seleção de um ponto de variação VP2 |
|  | Ponto de variação requer ponto de variação – quando a seleção do VP F7 exige a seleção de F3 |
|  | Ponto de variação exclui ponto de variação – quando a seleção de VP exclui a seleção de outro VP, neste caso, F3 e F7. |

Executada essa etapa do modelo de domínio, é necessário validá-la conforme a descrição apresentada na sequência.

#### *2.5.1.1.3 Validação do domínio*

As seguintes atividades devem ser realizadas para que a etapa de validação seja alcançada:

**Recursos para documentos**

Consiste nas características descritas abaixo:

- Descrição semântica – cada recurso deve ter uma descrição breve da sua semântica.
- Fundamentação – citar o motivo pelo qual a característica foi incluída no modelo.
- Partes interessadas e programa do cliente – anotar as partes interessadas e os programas de cliente de cada recurso.
- Exemplos de aplicações – se já existe aplicação que utiliza certo recurso, esta deve ser citada na documentação.
- Restrições – verificar as dependências criadas entre as características opcionais do sistema.
- Atributo aberto/fechado – deve-se selecionar um ponto de variação como aberto ou fechado. Aberto significa que as características que estão abaixo dele são esperadas, caso contrário deve ser marcado como fechado.
- Prioridades – para definir a relevância de cada recurso, podem ser atribuídas prioridades a eles.

**Verificar sinônimos**

Rever a ocorrência de sinônimos na descrição semântica. Ou seja, a existência de termos diferentes com mesmo significado para o domínio.

**Verificar homônimos**

Da mesma forma, devem ser revistos a ocorrência de homônimos onde um mesmo termo literal é usado com significados diferentes.

**Validação do modelo**

Após a documentação de cada recurso, eles dever ser revisados no modelo para validar a sua integridade.

**Documentar o domínio**

Para realizar a documentação do domínio deve ser seguidos os passos abaixo:

- Descrição do domínio – define as responsabilidades do domínio
- Definição das regras do domínio – critérios para a inclusão dos membros do domínio.
- Seleção de sistemas exemplares – descreve um conjunto de sistemas onde ocorre a funcionalidade do domínio.
- Documentação – descreve a documentação dos sistemas exemplares.
- Contexto de domínio – relaciona o domínio em foco com os demais.
- Genealogia de domínio – descreve a evolução e dependência entre os sistemas que pertencem ao domínio.
- Recursos – conjunto de características descritas no domínio.

As tarefas citadas até este ponto fazem parte da primeira grande etapa do desenvolvimento da Engenharia de Domínio, ou seja, sua análise. O próximo passo se dá através do Projeto de Domínio que será detalhado a seguir.

#### 2.5.1.2 Projeto de domínio

Esta etapa irá alcançar uma arquitetura comum para um conjunto de aplicações. Além disso, toda saída especificada do processo de análise deve ser mapeada na especificação do projeto. Deve-se determinar como os requisitos, incluindo a variabilidade se refletem na arquitetura. Os passos para a realização dessa tarefa serão descritos ao longo desta unidade.

##### *2.5.1.2.1 Decompor um módulo*

Corresponde a uma abstração e decomposição dos módulos produzidos na análise de domínio escolhendo aqueles que farão parte da arquitetura. Devem ser levantadas as informações detalhadas sobre bens do domínio, os objetivos de negócio e restrições que podem influenciar o projeto de arquitetura. A abordagem da tese não descreve essa prática com um conjunto rigoroso de critérios a serem seguidos, porém devem ser consideradas e haver um equilíbrio de algumas questões como: disponibilidade, acoplamento, extensibilidade, flexibilidade, funcionalidade, ocultação de informações, manutenção, modificabilidade, execução, separação de interesses, escalabilidade, segurança e usabilidade.

##### *2.5.1.2.2 Refinar um módulo*

Trata-se de um processo iterativo dividido em três partes:

- Escolha dos drivers da arquitetura – seleção dos drivers da arquitetura. Nesse caso, os drivers corresponde a combinação de requisitos funcionais e de qualidade. São expressos pelo modelo de domínio, pelos atributos de qualidade e os cenários, sempre que aplicável.
- Escolha dos padrões da arquitetura – definição dos padrões que serão aplicados nos drivers selecionados.
- Alocar funcionalidades usando aspectos – definir como os módulos podem ser instanciados.

#### *2.5.1.2.3 Representar a variabilidade*

Nesta etapa o autor seguiu alguns padrões para representar a capacidade de alterar ou personalizar o sistema, ou seja, a variabilidade. De acordo com o tipo de requisito, existe um padrão diferente para a representação.

Para os recursos alternativos, podem ser usados os padrões:

- Abstract Factory – fornece uma interface que define as famílias de objetos. Estas famílias podem ser relacionadas ou dependentes sem especificar suas classes concretas.
- Método Factory – define a interface de um objeto e a subclasse decide qual instanciar.
- Prototype – especifica os tipos de objetos que serão criados utilizando um protótipo e criando objetos a partir dele.
- Strategy – define uma família de algoritmos, os encapsula e tornam intercambiáveis.
- Método Template – define o esqueleto de um algoritmo e operações deixando alguns passos para as subclasses.

Para os recursos ou, podem ser usados os padrões:

- Builder – separa a construção de um objeto complexo da sua representação de maneira que uma mesma construção pode resultar em diferentes implementações.

- Observer – define uma dependência de um-para-muitos de maneira que uma mudança no estado de um objeto implica uma mudança em seus dependentes automaticamente.

Para os recursos opcionais, podem ser usados os padrões:

- Builder – este padrão decide se um recurso está ou não presente em uma aplicação.
- Decorator – pode ser utilizado principalmente em características adicionais onde é responsável por gerenciar e chamar a execução.
- Observer – pode ser utilizado da mesma maneira que nos recursos ou.

#### *2.5.1.2.4 Definir um componente*

Depois de realizada a decomposição e refinamento dos módulos, devem ser definidos os componentes dessa arquitetura. Para isto serão descritas três tarefas.

**Grupo de componente**

Nesta etapa devem ser agrupados os componentes seguindo os passos a seguir:

- Medida de dependência funcional – devem-se avaliar as dependências entre os casos de uso em clusters, relacionando com um componente.
- Clusters de casos de uso – avaliado a dependência funcional entre eles, devem ser agrupados os casos de uso relacionados.
- Alocar classes para os componentes – é feita a reserva de classes para cada componente usando um modelo dinâmico de acordo com um diagrama de sequência.
- Selecionar os componentes candidatos – escolher a melhor configuração de agrupamentos de componentes considerando o número de componentes e sua granularidade.

O autor apresenta funções matemáticas que realizam as medidas e definem as duas primeiras atividades. Neste trabalho não foram abordados estes tópicos com este nível de detalhamento.

**Identificar um componente**

É uma revisão dos agrupamentos gerados considerando os modelos iniciais analisando se os componentes respeitam a definição do domínio.

**Especificar um componente**

Nesta atividade devem ser feitas a interfaces e especificações dos componentes. Isto pode ser realizado seguindo os passos:

- Identificar as interfaces – para cada caso de uso onde há responsabilidades do sistema que precisam ser modelados deve ser criada uma interface. Isso resulta num conjunto inicial de interfaces.
- Identificar classes e refinamento da especificação – definir quais classes farão parte do núcleo. Para isto, a classe deve fazer parte do sistema de maneira independente das demais.

#### *2.5.1.2.5 Representar a arquitetura de domínio*

Com a especificação realizada nas etapas anteriores, é possível representar essa arquitetura baseada em componentes.

### 2.5.1.3 Implementação do domínio

Para realização desta terceira e última etapa, que tem como principal objetivo implementar e documentar o software, o autor utilizou o Open Service Gateway Interface (OSGI) para gerenciar a iteração e o ciclo de vida da aplicação.

O OSGI é uma especificação de interface Java que define um padrão, no ambiente orientado a componente, de computação para serviços de rede.

Por não se tratar da especificação que será abordada no decorrer deste trabalho, levaremos em consideração a realização de duas tarefas para a realização desta etapa:

- Implementação de componentes – que é feita baseada nos requisitos.
- Documentação de componentes – com objetivo de facilitar o desenvolvimento de novos softwares a partir destes e ainda ajudar em futuras tarefas como a portabilidade.

Assim é concluída a descrição do trabalho realizado por Eduardo Almeida que serve de base para os passos de desenvolvimento de uma Engenharia de Domínio para uma LP.

### 2.5.2 Engenharia de aplicação

Com objetivo de descrever o desenvolvimento da engenharia de aplicação de um LP, foi utilizado o padrão Feature-Oriented Reuse Method (FORM) como referência para este estudo.

O FORM é definido como um método que busca e identifica características comuns e a variabilidade entre aplicações baseado em características para o desenvolvimento de uma arquitetura de componentes. Com a descrição de um domínio considerando suas unidades comuns e variáveis de computação, é possível a construção de diferentes arquiteturas a partir dele [FORM, 1998].

Engenharia de aplicação é o processo de desenvolvimento de uma aplicação específica reutilizando os conhecimentos obtidos na engenharia de domínio [FORM, 1998].

Duas etapas são caracterizadas pelo FORM para a realização de engenharia de aplicação. A descrição de cada uma delas é feito a seguir:

#### 2.5.2.1 Análise de requisitos e Seleção de recursos

A engenharia de aplicação inicia selecionando as características definidas na engenharia de domínio que farão parte do produto. Isto começa analisando os requisitos do aplicativo que será desenvolvido.

A seleção das características é feita a partir do modelo de domínio, que apresenta os recursos que são selecionáveis e suas relações, portanto o mais importante dessa etapa é considerar um conjunto válido de características que mais se assemelhem e atendam a necessidade do produto.

#### 2.5.2.2 Seleção da arquitetura e o desenvolvimento de aplicativos.

Com a seleção dos recursos que foi realizada anteriormente, um modelo de referência da arquitetura é gerado imediatamente de forma automática, apenas espelhando o que foi definido.

A partir da arquitetura, os componentes reutilizáveis são facilmente encontrados apenas seguindo essa especificação. Os módulos específicos do produto devem ser codificados de forma que complemente essa arquitetura reutilizada. Essa abordagem proporciona o desenvolvimento da cada produto a partir do núcleo de

componentes minimizando os possíveis problemas que ocorrem no desenvolvimento de software.

# 3 Trabalhos relacionados

Neste capítulo é feita uma abordagem mostrando os pontos em comum do trabalho que está sendo realizado e descrito nesse documento com outros que abordam os mesmos temas e foco semelhante.

Um trabalho que faz um comparativo de tecnologias para melhorar a produtividade de desenvolvimento de software é apresentado em [BALBINO, ALMEIDA e MEIRA, 2011]. Ele reúne dois processos de software: métodos ágeis e LPs. Sua proposta é o desenvolvimento de um processo para LPs baseada em princípios ágeis e será explicada a seguir.

## *3.1 RiPLE-SC*

O desenvolvimento de uma LP exige o gerenciamento da variabilidade dos produtos durante o desenvolvimento da arquitetura de software o que acaba dificultando a sua concepção comparada a um projeto normal de software. Neste sentido, a introdução de características ágeis no desenvolvimento de uma LP pode melhorar ainda mais o seu processo de desenvolvimento [BALBINO, ALMEIDA e MEIRA, 2011]. É baseado nesta ideia que eles apresentam o RiPLE-SC: um processo ágil para o escopo de uma LP. Ao longo desta seção serão descritas a práticas sugeridas pelo RiPLE-SC.

O RiPLE-SC é dividido em quatro fases: Pré-escopo, Escopo de domínio, Escopo de produto e Escopo de componentes.

### 3.1.1 Pré-escopo

É feito um levantamento das características de visão geral do projeto, o contexto operacional e organizacional, os papéis dos envolvidos, a meta de negócios e potencial de mercado. Isto é feito seguindo as seguintes práticas:

- Reunião pré-escopo – Identifica informações da equipe do cliente e sua organização. É o primeiro contato entre o cliente e a equipe de desenvolvimento.
- Análise de mercado – faz a coleta das características de negócio, avaliação da competitividade, segmento do mercado, planos do cliente e a integração destes fatos em um plano de negócios. O objetivo é de obter informações do mercado e o domínio em que o projeto está inserido.

### 3.1.2 Escopo de Domínio

É a determinação do conjunto de domínios e subdomínios, considerando a maturidade, volatilidade do mercado, potencialidade, riscos, experiência e o acoplamento do código já existente. Os membros da equipe devem realizar uma análise e discutir os possíveis domínios. Essa comunicação entre a equipe reduz a necessidade de documentação. Para alcançar este objetivo, é sugerido a seguinte tarefa:

- Análise de domínios – é um processo iterativo e incremental feita ao longo das etapas definindo os domínios pertencentes à aplicação.

### 3.1.3 Escopo de Produto

Para oferecer ao cliente exatamente o que ele precisa, deve ser definido um portfólio de produtos para satisfazer a demanda dos clientes. Para completar essa etapa, algumas tarefas serão executadas:

- Construção das histórias de usuários – as histórias são utilizadas para entender a necessidade do cliente e, portanto, as características que os produtos precisam ter.
- Identificação das características – é a identificação das características que precisam ser abordadas pelos produtos da LP.
- Reunião de análise de recursos – é uma reunião para análise das características feita entre o cliente e a equipe de desenvolvimento.
- Identificação dos produtos – é a definição de uma lista de produtos que fazem parte da LP.
- Mapeamento dos produtos – tendo a lista de produtos da LP, podem ser listados numa tabela os requisitos de cada um deles.
- Validação do mapeamento – consiste em uma reunião entre o cliente e especialistas de domínio para a análise dos produtos definidos e seus requisitos.

### 3.1.4 Escopo de Componentes

Determina as características relevantes e os componentes que precisam ser construídos para serem reutilizados nos produtos. Deve ser analisada, de forma quantitativa, a necessidade de cada recurso. Fazem parte dessa tarefa as seguintes práticas:

- Criação de métricas – com base nos objetivos de negócio, são definidas as métricas que identificam a necessidade de um componente.
- Aplicação das métricas – uma análise do esforço e a necessidade de cada componente são feitas.

Prioridade no mapeamento dos produtos – é uma seleção dos componentes mais importantes para a LP.

É dessa maneira que o RiPLE-SC contempla o desenvolvimento de uma LP de software para que possa ser implementada de forma ágil atendendo as necessidades dos clientes e garantindo o funcionamento do processo.

Outro trabalho que será descrito é o de [GHANAM e MAURER, 2008] que apresenta um modelo iterativo de engenharia ágil de LP e está compreendido na próxima seção.

## 3.2 Modelo Iterativo de Engenharia Ágil de LP

O modelo proposto tem como alvo organizações que já utilizam métodos ágeis e desejam criar sua LP. Dessa forma, foi feita uma abordagem "bottom-up" onde a LP é construída a partir dos produtos existentes, ou seja, essa plataforma é criada iterativamente evoluindo ao longo do projeto [GHANAM e MAURER, 2008]. O modelo leva em consideração quatro pontos:

- Análise de requisitos – para o desenvolvimento de uma LP, a análise do domínio exige um esforço bastante alto já que é feito um planejamento para projetos futuros. Para os métodos ágeis, as incertezas iniciais de um projeto impedem esse tipo de investimento levando o foco para as necessidades imediatas. Algo semelhante ocorre com a documentação dessa análise onde as LP precisam de um nível de detalhamento que não são aprovados pelos métodos ágeis;
- Planejamento do reuso – os métodos ágeis consideram as necessidades atuais por ser algo concreto e vão acrescentando funcionalidades ao longo do desenvolvimento. Em LP, deve existir um conjunto de artefatos que será reutilizado no desenvolvimento exigindo que a arquitetura seja definida antecipadamente ao contrário dos métodos ágeis, que desenvolvem a arquitetura de forma incremental;

- Gerência do núcleo de artefatos – o sucesso de uma LP depende de um núcleo de artefatos com uma boa gestão com mecanismos que permitam sua reutilização e manutenção. Novamente os métodos ágeis dispensam esse tipo de gerenciamento em sua produção, gerando dúvidas de como deve ser feito o controle de bens essenciais e sua gerência;
- Arquitetura flexível – a arquitetura de uma LP deve permitir pontos de variação em seus artefatos que permitam a conexão de suas interfaces entre os diferentes produtos. Já nos métodos ágeis a arquitetura vai surgindo de baixo para cima ao longo do desenvolvimento dificultando o planejamento desses pontos de variação.

A base do modelo foi inspirada na técnica Test Driven Development, onde todo o desenvolvimento é direcionado por testes. Os testes de aceitação fazem uma ponte entre o desenvolvimento ágil e as LPs e irão conduzir a reutilização de artefatos de desenvolvimento.

Para detalhar o as atividades do modelo, é considerado que a organização que pretende desenvolver a LP já possui dois sistemas A e B em desenvolvimento, e decide criar um terceiro sistema C que pertence ao mesmo domínio. A partir daí inicia-se as seguintes tarefas:

### 3.2.1 Equipe do núcleo de artefatos

Antes do desenvolvimento do sistema C, uma equipe deve ficar responsável pela criação e manutenção do núcleo da LP. Isso ocorre através da mineração dos sistemas A e B extraindo módulos que possam ser reutilizados nos próximos produtos, extraindo uma camada genérica para os módulos e definindo as variações dos artefatos. O processo de mineração fica condicionado aos testes de aceitação que estão associados a cada módulo.

### 3.2.2 Avaliação e Extração

Antes do desenvolvimento do sistema C, o conjunto de histórias de usuário deve ser transformado em testes de aceitação. O processo de avaliação sugere a comparação dos testes de aceitação de C com os que já existem dos sistemas anteriores. Quando é encontrada uma grande semelhança entre eles, um processo de adaptação dos artefatos é feito para que este passe a abranger o novo sistema. Para extrair o teste de aceitação genérico, três etapas podem ser realizadas:

- Inserir todos os testes de aceitação em uma camada genérica;

- Especificar os pontos de variação;
- Especificar a relação entre os pontos de variação e os testes de aceitação.

### 3.2.3 Refatoração

O desenvolvimento de um modelo dos testes de aceitação permite a condução do processo de refatoração. As características que foram classificadas anteriormente como genéricas serão incluídas por essa refatoração ao núcleo, e aquelas que pertencem à camada de variabilidade irão conduzir o desenvolvimento de módulos específicos.

### 3.2.4 Gerenciamento do núcleo de componentes

Com a refatoração foram criados um novo modulo dos componentes genéricos e os módulos específicos, consequentemente devem ser incluídas essas mudanças ao repositório principal e ainda a correspondência destes módulos com os testes de aceitação. Inicialmente, estes testes podem ser executados manualmente, porém a sua automatização auxilia na documentação, rastreabilidade e manutenção dos módulos.

### 3.2.5 Incorporação do núcleo de componentes

Os esforços para o desenvolvimento do núcleo de componentes pode ser utilizado para fazer parte dos sistemas anteriores ao desenvolvimento da LP. Para isto, o modulo resultante deve ser repassado de volta para que as equipes que desenvolveram o sistema A e B, por exemplo, possam evoluir dentro desse processo.

### 3.2.6 Evolução da arquitetura

Com a execução dos passos anteriores no nível de projeto, o número de artefatos reutilizáveis vai crescendo e resultando em um nível maior de reutilização. Quanto maior o número de componentes, mais estável vai se tornando a arquitetura aumentando a camada genérica e separando a camada de variabilidade. Essa abordagem ascendente evita o custo inicial de uma abordagem tradicional de desenvolvimento de uma LP, e garante uma arquitetura estável com redução de custos conforme o número de produtos vai crescendo.

Assim finaliza o modelo que propõe de integração do desenvolvimento ágil de software com a engenharia de uma LP através de uma abordagem “bottom-up” em que o núcleo de componentes é extraído conforme a demanda com a inclusão de testes de aceitação para substituir a documentação de uma LP.

## *3.3 Integrando LP e desenvolvimento ágil*

O terceiro e último trabalho a ser descrito nesse capítulo foi proposto por [MOHAM, RAMESH e SUGUMARAN, 2010]: Integrando engenharia de LP de software e desenvolvimento ágil. Nas próximas seções, é apresentada a sua proposta.

Pesquisas anteriores já utilizaram os sistemas adaptativos [MESO e JAIN, 2006] complexos para sistemas que resultam em padrões de comportamento como na variabilidade de domínios e análise de métodos de desenvolvimento de software. Os princípios básicos dos sistemas adaptativos complexos ajudam a entender como os métodos ágeis podem se adaptar às condições dinâmicas do ambiente. São eles: tensão adaptativa, complexidade de requisito, taxa de alteração, arquitetura modular, qualificações positivas, complexidade causal e ritmos de coordenação.

### 3.3.1 Integrando o desenvolvimento ágil e de famílias de produtos

O modelo proposto pelo estudo tomou como base uma empresa de desenvolvimento de software para apoio de rendimentos já conhecida pela entrega rápida de seus softwares e resolveu criar sua LP.

Para a definição do processo, os sete princípios dos sistemas adaptativos complexos foram integrados em adaptações coevolutivas aplicados em conjunto. As condições para o desenvolvimento de uma LP são determinadas pela natureza de mercado que podem ser destacados três fatores:

- Extensão de variação necessária no portfólio de produtos – dependendo da diversidade dos produtos, uma organização precisa adaptar seus processos para lidar com pontos em comuns e a variabilidade.
- Restrições de tempo – a limitação de tempo na entrega de um produto customizado exige a escolha de uma abordagem apropriada.
- Expectativas de capacidade - um portfólio de produtos com as funcionalidades adequadas deve ser estabelecido antes do seu processo e definição de recursos.

### 3.3.2 Princípios dos Sistemas adaptativos complexos e do processo integrado

Os processos foram especializados da seguinte maneira:

#### 3.3.2.1 Tensão adaptativa

Três praticas são utilizadas para alinhar os processos de evolução com as necessidades organizacionais:

- Entrelace progressivo da engenharia de domínio e aplicação – uma abordagem reativa para a LP para equilibrar a extensão dos pontos em comum e a variabilidade. Isso é feito pois eles consideram que grande parte da variabilidade não pode ser prevista.
- Envolver os clientes continuamente para determinar os requisitos de forma adaptativa – o desenvolvimento de uma plataforma que atenda os clientes e as variações são tratadas como refatoração.
- Controlar o escopo – a criação de um mecanismo para equilibrar o conflito entre cumprir os requisitos e limitando personalizações com o escopo das variações.

#### 3.3.2.2 Complexidade de requisito

A complexidade de cada necessidade deve ser avaliada de acordo com os seguintes pontos:

- Estabelecer linhas de base de processo padrão – proporciona flexibilidade para lidar com a complexidade ambiental.
- Estabelecer um repositório de conhecimento – devem ser documentadas as decisões de design, fundamentos, e experiências em um repositório de conhecimento estruturado.
- Um projeto simples, mas equilibrado – pode ser mais vantajoso utilizar um código personalizado ao invés de componentes genéricos que levam a uma maior complexidade.
- Refatorar seletivamente – ao invés de um quadro de refatoração, devem ser selecionados os artefatos que produzem os benefícios mais significativos.

#### 3.3.2.3 Taxa de alteração

Três práticas devem ser feitas para uma adaptação contínua e incremental para as mudanças e necessidades do sistema.

- Derivação incremental de produtos – primeiro é criado uma plataforma padrão e funcional para as funcionalidades exigidas que proporciona um quadro de referência para definir as necessidades variáveis.
- Adaptar os mecanismos de variação – para lidar com as variações de maneira rápida, podem ser criados componentes genéricos configurados em função da necessidade.
- Adaptar uma estratégia de escopo da LP – conforme a variabilidade necessária vai evoluindo e estabilizando a plataforma, é possível acomodar novos requisitos ao invés de modificar os existentes para acomodar as variações.

#### 3.3.2.4 Projeto modular

É dividido em quatro práticas:

- Estabelecer um limite equilibrado entre as equipes de engenharia de domínio e de aplicação – os desenvolvedores podem trabalhar em qualquer aspecto do desenvolvimento. Apenas em grandes personalizações para um produto é definido uma equipe fixa.
- Desenvolver uma arquitetura flexível – estabelecer uma arquitetura em camadas suficiente para trabalhar com as mudanças.
- Uso de COTS quando viável – utilizar componentes COTS (Commercial Off-the-Shelf) sempre que a funcionalidade necessária tiver menor importância.
- Refatorar para reutilização – o uso de ferramentas para extrair código que pode ser refatorado para a reutilização.

#### 3.3.2.5 Qualificações positivas

São integradas três práticas:

- Proporcionar a autonomia adequada – é um processo de realimentação onde a variabilidade é movida para uma estrutura mais complexa.
- Uso de práticas de gestão de risco – incorporação de práticas que gerenciem os riscos.

- Estender a plataforma para acomodar a variabilidade – como a plataforma inicial não tem grande capacidade de lidar com variações, os módulos podem ser refatorados para se tornarem mais passíveis de personalização.

#### 3.3.2.6 Complexidade causal

Resume-se à seguinte prática:

- Gerenciar os relacionamentos complexos – é a gerência das relações entre estilo de gestão organizacional, processos de desenvolvimento, gestão da tecnologia, reconhecimento da complexidade, múltiplas direções e não linearidade.

#### 3.3.2.7 Ritmos de coordenação

Para esta etapa, três práticas foram integradas:

- Facilitar a dualidade das estruturas top-down e bottom-up – ampliação da complexidade das ligações entre as estruturas top-down e bottom-up, resultando em um limite dinâmico entre o núcleo ativo e as equipes de desenvolvimento de produtos.
- Alinhar a infra-estrutura com as necessidades do processo – implantação de uma infra-estrutura necessária para gerir pontos em comum e variações.
- Incorporar uma estrutura organizacional mista – uma equipe desenvolve a plataforma inicial coletivamente, em vez de dividir em engenharia de domínio separado e as equipes de engenharia de aplicação. Após alguns ciclos são feitas as personalizações importantes para equipes separadas.
- Estas foram as práticas propostas por [MOHAM, RAMESH e SUGUMARAN, 2010] para o integrar o desenvolvimento ágil com uma LP.

# 4 Framework Estendido

O trabalho desenvolvido busca a aplicação de práticas ágeis para o desenvolvimento de uma LP. Se analisados algumas definições e princípios destas tecnologias, é possível identificar pontos conflitantes principalmente entre o conceito de desenvolvimento ágil comparado com LPs. A agilidade no processo de desenvolvimento requer a redução de etapas de definição de diagramas e documentação presentes no desenvolvimento tradicional de um projeto de software. Em contrapartida, para que a criação de uma LP de software possa ser executada com a qualidade e eficiência desejada, são exigidas, justamente, esse tipo de prática que desenvolve diagramas e documenta o que está sendo feito.

O estudo realizado permitiu a especificação de um framework que mostra as etapas e práticas para o desenvolvimento de toda uma linha de produtos de software.

Para simplificar o uso das práticas que foram descritas na seção 2.5, foi criada uma tabela que faz a comparação entre os frameworks. Essa tabela apresenta nas duas primeiras colunas as práticas que foram estudadas e na terceira coluna a pratica resultante que foi incluída no framework estendido. As práticas que foram abandonadas ou não tiveram um mapeamento direto de sua funcionalidade, estão grifadas em cinza na tabela. Primeiramente é mostrado abaixo a comparação das práticas ágeis com o framework proposto por [ALMEIDA, 2007] para a Engenharia de domínio.

**Tabela 4 - Comparação entre as práticas de origem da Engenharia de Domínio do framework**

| | **Framework de métodos ágeis** | **Framework [Almeida, 2007]** | **Framework Estendido** |
|---|---|---|---|
| **Engenharia de domínio** | Lista de requisitos (Obrigatório) | Plano de domínio | Lista de requisitos de cada aplicação (obrigatório) |
| | Modelo geral | Modelo de domínio | Modelo de domínio (obrigatório) |
| | Documentação inicial | | Documentação |
| | | Validação do domínio | |
| | Projeto de arquitetura do sistema | Decompor um módulo | Projeto da arquitetura do sistema (obrigatório) |
| | | Refinar um módulo | |
| | | Representar a variabilidade | |
| | | Definir um componente | |
| | | Representar a arquitetura de domínio | Representação da arquitetura de domínio (obrigatório) |
| | Planejamento da iteração (Obrigatório) | | Planejamento da iteração (obrigatório) |
| | Reuniões diárias | | Reunião diária |
| | Histórias do usuário | | - |
| | Casos de uso | | |
| | Projeto da iteração | | Projeto da iteração |
| | Desenvolvimento da iteração | | Desenvolvimento da iteração |

| | | | |
|---|---|---|---|
| | (Obrigatório) | | (obrigatório) |
| | Escrita dos testes de unidade | | Escrita dos testes de unidades |
| | Escrita dos testes de aceitação | | - |
| | Desenvolvimento de código coletivo | | Desenvolvimento coletivo do código |
| | Programação em pares | | Programação em pares |
| | Refatoração | | Refatoração |
| | Programação lado-a-lado | | Programação lado-a-lado |
| | Inspeção de código | | Inspeção do código |
| | Reunião de revisão da integração | | - |
| | Geração de uma breve documentação | | |
| | Entrega do sistema ao cliente | | |

Abaixo, temos a comparação das práticas ágeis com o padrão [FORM, 1998] para a Engenharia de aplicação:

**Tabela 5 - Comparação entre as práticas de origem da Engenharia de Aplicação do framework**

| | Framework de métodos ágeis | Padrão [FORM, 1998] | Framework Estendido |
|---|---|---|---|
| **Engenharia de Aplicação** | Lista de requisitos (Obrigatório) | | Lista de requisitos (Obrigatório) |
| | | Análise de requisitos e Seleção de recursos | - |
| | | Seleção da arquitetura e o desenvolvimento de aplicativos. | Seleção dos componentes (Obrigatório) |
| | Modelo geral | | Modelo geral |
| | Documentação inicial | | Documentação inicial |
| | Projeto de arquitetura do sistema | | - |
| | Planejamento da iteração (Obrigatório) | | Planejamento da iteração (Obrigatório) |
| | Reuniões diárias | | Reuniões diárias |
| | Histórias do usuário | | Histórias do usuário |
| | Casos de uso | | Casos de uso |
| | Projeto da iteração | | Projeto da iteração |
| | Desenvolvimento da iteração (Obrigatório) | | Desenvolvimento da iteração (Obrigatório) |
| | Escrita dos testes de unidade | | Escrita dos testes de unidade |
| | Escrita dos testes de aceitação | | Escrita dos testes de aceitação |
| | Desenvolvimento de código coletivo | | Desenvolvimento coletivo do código |
| | Programação em pares | | Programação em pares |
| | Refatoração | | Refatoração |
| | Programação lado-a-lado | | Programação lado-a-lado |
| | Inspeção de código | | Inspeção de código |
| | Reunião de revisão da integração | | Reunião de revisão da integração |
| | Geração de uma breve documentação | | Geração de uma breve documentação |
| | Entrega do sistema ao cliente | | Entrega do sistema ao cliente |

A seguir a caracterização de cada uma delas será apresentada:

## 4.1 Visão geral do desenvolvimento

Durante o desenvolvimento das etapas de uma LP, algumas tarefas são realizadas através de iterações. Para facilitar o entendimento, a figura abaixo mostra uma visão geral desse fluxo:

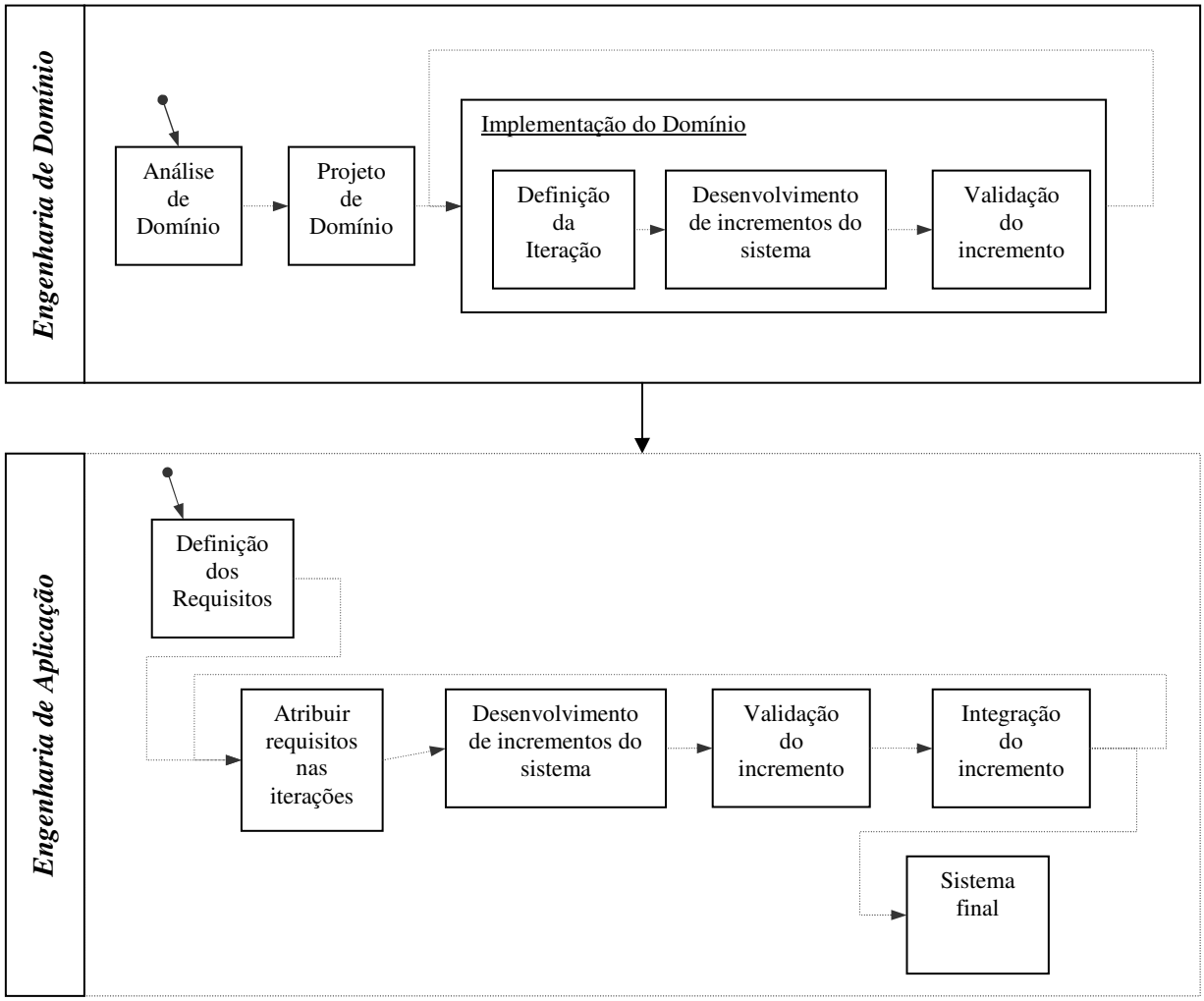



**Figura 5 - Visão geral do desenvolvimento proposto.**

Pode-se observar que durante a Engenharia de domínio, inicialmente é feita uma análise de domínio, um modelo de domínio e o projeto da arquitetura do núcleo de LP. Posteriormente, o desenvolvimento de cada componente é feito de maneira iterativa.

Na realização de Engenharia de aplicação, o processo é bastante semelhante, inicialmente ocorre uma análise dos requisitos do produto seguida de uma especialização do modelo de domínio e da arquitetura para o produto que está sendo produzido. O desenvolvimento também ocorre de forma iterativa. Posteriormente ocorre a entrega do produto ao cliente com a validação do sistema e o sistema final.

## *4.2 Engenharia de domínio*

Conforme descrito anteriormente, trata-se da etapa que irá reunir as informações e características referentes a certo domínio descrevendo na forma de unidades que serão reutilizadas. Para realizar a engenharia de domínio, os passos a seguir devem ser seguidos.

### 4.2.1 Análise de domínio

A análise de domínio compreende a etapa onde é realizada uma análise geral das aplicações pertencentes a LP e um estudo das características de cada uma delas.

#### 4.2.1.1 Lista de requisitos de cada aplicação (obrigatório)

Nesta etapa devem ser mapeadas as aplicações que farão parte da LP. Devem ser avaliadas as aplicações que já foram ou estão em processo de desenvolvimento e também as aplicações que serão desenvolvidas posteriormente.

Para cada aplicação pertencente à LP, devem ser listados os requisitos de cada uma delas.

#### 4.2.1.2 Modelo de domínio (obrigatório)

É a descrição de um modelo inicial que descreva as características do sistema. Normalmente isto é feito destacando as unidades levantadas como requisito das aplicações apresentando as relações de composição, as opções de inclusão de cada característica e ainda, a hierarquia de cada uma delas. O detalhamento deste modelo é apresentado na seção 4.5.

#### 4.2.1.3 Documentação

Esta documentação deve conter uma breve descrição das características e do modelo gerado. Além disso, pode ser apresentada uma visão geral do sistema apontando uma estimativa de custos, as expectativas de cada aplicação e risco.

### 4.2.2 Projeto de domínio

Nesta parte é feita a especificação da arquitetura reutilizável do sistema da seguinte maneira:

#### 4.2.2.1 Projeto da arquitetura do sistema (obrigatório)

Compreende uma visão técnica da arquitetura do domínio em questão.

- Decompor e refinar um módulo - Devem ser feitas as abstrações para as características levantadas na etapa de análise de domínio. Caso haja a

necessidade, uma característica pode ser quebrada em mais unidades ou até mesmo pode ocorrer o agrupamento de algumas delas.

- Definir um componente - Deve ser revisado o refinamento das características identificadas anteriormente e verificado se ele respeita as especificações iniciais do modelo. E para cada módulo é necessário especificar as interfaces e quais classes farão parte do núcleo de componentes.

#### 4.2.2.2 Representação da arquitetura de domínio (obrigatório)

A arquitetura desenvolvida deve ser representada baseada nos componentes especificados. Podem ser utilizados diferentes recursos para descrever essa arquitetura como diagrama de camadas ou diagrama de pacotes.

### 4.2.3 Implementação de domínio

Nesta etapa de implementação do núcleo da LP, fazem parte as seguintes tarefas:

#### 4.2.3.1 Definição da iteração

Cada iteração será definida podendo utilizar a seguinte prática:

##### *4.2.3.1.1 Planejamento da iteração (obrigatório)*

Deve ser definida a melhor maneira para o desenvolvimento de cada módulo descrito na arquitetura. Devem ser distribuídos os módulos ou requisitos de cada um deles dentro da iteração.

Para a iteração, deve-se estipular o tempo necessário para o seu desenvolvimento. De acordo com o framework de desenvolvimento ágil que foi citado anteriormente, este período não deve exceder oito semanas.

Dependendo da equipe de desenvolvimento, é possível que haja a distribuição das tarefas entre seus membros.

#### 4.2.3.2 Desenvolvimento de incrementos do sistema

Sempre que houver necessidade ou, normalmente, ao final de cada iteração, o sistema pode ser revisado e adicionadas novas funcionalidades a ele seguindo os passos a seguir:

##### *4.2.3.2.1 Reunião diária*

São reuniões curtas que ocorrem a cada dia de trabalho entre os membros da equipe que informam seus progressos ou dificuldades.

#### *4.2.3.2.2 Projeto da iteração*

Consiste em um projeto para descrever a nova funcionalidade. Pode-se incrementar a documentação existente ou desenvolver um diagrama simples para este propósito.

#### *4.2.3.2.3 Desenvolvimento da iteração (obrigatório)*

É a codificação dos componentes que pertencem à iteração.

#### *4.2.3.2.4 Escrita dos testes de unidades*

Recomenda-se a escrita de testes de unidade antes da codificação, baseado nas interfaces pré-definidas.

#### *4.2.3.2.5 Desenvolvimento coletivo do código*

O desenvolvimento é responsabilidade de todos que fazem parte da equipe. Portanto, o conhecimento e entendimento de todos se faz necessário.

#### *4.2.3.2.6 Programação em pares*

O desenvolvimento da codificação pode ser feito por um par de programadores, onde enquanto um deles implementa o código, o outro analisa e sugere simplificações sobre o que está sendo feito.

#### *4.2.3.2.7 Refatoração*

A qualquer momento que a equipe julgar necessário deve ser feito uma refatoração do código.

#### *4.2.3.2.8 Programação lado-a-lado*

Para melhor relacionamento entre os membros da equipe, uma configuração que aponta uma divisão do ambiente de trabalho onde os programadores ficam dispostos lado-a-lado é bastante importante.

### 4.2.3.3 Validação do incremento

Sempre que uma modificação considerável for feita existe a necessidade de ser executada a seguinte validação:

#### *4.2.3.3.1 Inspeção do código*

É a procura por defeitos que possam ter sido introduzidos no código gerado.

## *4.3 Engenharia de aplicação*

Trata-se da especificação do núcleo de artefatos criado na engenharia de domínio para o desenvolvimento de um produto em particular. Para realizar a engenharia de aplicação, os passos a seguir devem ser seguidos.

### 4.3.1 Atividade: Definição dos requisitos.

#### 4.3.1.1 Lista de requisitos (Obrigatório)

É a definição dos requisitos do produto. A lista de requisitos pode conter uma descrição de cada um dos requisitos e sua prioridade.

#### 4.3.1.2 Seleção dos componentes (Obrigatório)

A partir da lista de requisitos, devem ser selecionados os componentes que fazem parte do núcleo de artefatos, que farão parte desse produto. O modelo de domínio desenvolvido na engenharia de domínio pode ser utilizado para realização desta tarefa.

#### 4.3.1.3 Modelo geral

Caso haja algum requisito novo, ou seja, que não está presente no núcleo, pode ser feita uma especialização do modelo que já existe incluindo este requisito.

#### 4.3.1.4 Documentação inicial

É um documento que apresenta uma visão geral do sistema considerando seus custos, benefícios e riscos. Pode também reunir informações como o contato do cliente e principais envolvidos, tecnologias e ferramentas que esse produto necessite.

### 4.3.2 Atividade: Atribuir requisitos nas iterações

#### 4.3.2.1 Planejamento da iteração (Obrigatório)

Deve ser definido o que será desenvolvido dentro de cada iteração levando em conta o que será reutilizado do núcleo de artefatos, a prioridade dos novos requisitos, riscos e dependências.

Como já foi dito anteriormente, uma iteração baseada em métodos ágeis deve compreender um período de uma a oito semanas. A partir disto deve ser estipulado o período necessário para o desenvolvimento da iteração.

Por fim, deve haver a distribuição dos requisitos entre os desenvolvedores que fazem parte da equipe.

### 4.3.3 Atividade: Desenvolvimento de incrementos do sistema

#### 4.3.3.1 Reuniões diárias

São reuniões realizadas entre os membros da equipe para que sejam discutidos os avanços e dificuldades do desenvolvimento da iteração.

#### 4.3.3.2 Histórias do usuário

É a descrição feita pelo cliente das histórias de usuário para cada funcionalidade do produto.

#### 4.3.3.3 Casos de uso

Cada requisito do sistema deve ter um caso de uso correspondente. Nesta etapa, é bastante importante considerar os novos requisitos identificados para o produto.

#### 4.3.3.4 Projeto da iteração

Consiste em um projeto de sistemas contendo diagramas UML baseado nos requisitos da iteração em desenvolvimento.

#### 4.3.3.5 Desenvolvimento da iteração (Obrigatório)

Compreende o desenvolvimento do código dos requisitos produto. Esse desenvolvimento deve seguir os padrões e estruturação de código que são utilizados na LP.

#### 4.3.3.6 Escrita dos testes de unidade

Sugere o desenvolvimento dos testes de unidade antes da codificação.

#### 4.3.3.7 Escrita dos testes de aceitação

Os testes de aceitação também devem ser desenvolvidos antes da codificação.

#### 4.3.3.8 Desenvolvimento coletivo do código

É de responsabilidade de toda equipe o desenvolvimento do código. Portanto, o seu entendimento e conhecimento ficam condicionados a todos os membros.

#### 4.3.3.9 Programação em pares

Propõe que o desenvolvimento do código seja feito por duplas de programadores onde um deles executa a codificação permitindo que o outro analise o que está sendo feito e sugira simplificações.

#### 4.3.3.10 Refatoração

Sempre que a equipe julgar necessário, pode ser feita a refatoração do código.

#### 4.3.3.11 Programação lado-a-lado

Consiste em uma configuração do ambiente de trabalho onde os programadores são dispostos lado-a-lado.

### 4.3.4 Atividade: Validação do incremento

#### 4.3.4.1 Inspeção de código

É a revisão de todo o código desenvolvido na iteração. Serve para identificação de possíveis erros. Os programadores devem inspecionar o código do outro.

### 4.3.5 Atividade: integração do incremento

#### 4.3.5.1 Reunião de revisão da integração

São reuniões que devem ocorrer ao final de cada iteração para discutir se foi concluída toda etapa de integração do código.

### 4.3.6 Atividade: sistema final

#### 4.3.6.1 Geração de uma breve documentação

Caso a documentação necessária do produto não tenha sido gerada ao longo do desenvolvimento, ao final da execução, uma breve descrição do produto deve ser feita e até entregue juntamente para o cliente.

#### 4.3.6.2 Entrega do sistema ao cliente

Concluídas todas as exigências de desenvolvimento, o produto deve ser entregue ao cliente. Nesta esta etapa uma reunião deve ser feita para reconhecimento do final do projeto.

## *4.4 Práticas descartadas*

Para conciliar a tarefa de integração dos Frameworks no desenvolvimento da Engenharia de domínio e de aplicação de uma LP, algumas práticas deixaram de fazer parte da proposta de metodologia desenvolvida neste capítulo.

Para justificar essa abordagem, as práticas que foram abolidas são citadas e detalhados os motivos pelos quais foi tomada essa decisão.

**Práticas ágeis excluídas da Engenharia de domínio:**

Na atividade: Desenvolvimento de incrementos do sistema:

- Histórias do usuário – foram removidas, pois no desenvolvimento do núcleo de uma LP não existe o usuário interagindo diretamente com o sistema para gerar esse fluxo.
- Casos de uso – também foi removido pelo fato de não existir um usuário interagindo diretamente com o sistema para gerar o detalhamento correspondente a um caso de uso.
- Escrita dos testes de aceitação – Como não haverá uma entrega ao cliente ao final da engenharia de domínio, não devem ser feitos estes testes.

Na atividade: Validação do incremento: nenhuma prática foi excluída.

Na atividade: Integração do incremento:

- Reunião de revisão da integração – com a integração ocorrendo ao longo do processo não há a necessidade de uma reunião para revisar a integração das várias partes do sistema.

Na atividade: Sistema final

- Geração de uma breve descrição – para cada requisito pode ser gerada uma documentação, portanto esse relatório geral do que foi produzido não foi considerado.
- Entrega do sistema cliente – novamente, essa etapa de engenharia de domínio não terá um cliente envolvido, não resultando em uma entrega.

**Práticas do framework [ALMEIDA, 2007] excluídas da Engenharia de domínio:**

Na análise de domínio:

- Validação do domínio – foi retirada para simplificar a análise de domínio e proporcionar agilidade ao processo.

No projeto de domínio:

- Representar a variabilidade – foram retirados os padrões de gerência de projeto dessa etapa também para garantir a agilidade do processo.

**Práticas ágeis excluídas da Engenharia de aplicação:**

Na atividade: Projeto da arquitetura do sistema

- Projeto da arquitetura do sistema – no processo de engenharia de aplicação, deve-se reutilizar a arquitetura desenvolvida na engenharia de domínio.

**Práticas do [FORM, 1998] da Engenharia de aplicação:**

- Seleção da arquitetura e o desenvolvimento de aplicativos – como foi dito anteriormente, a arquitetura deve ser reutilizada. Quanto ao desenvolvimento de aplicativos, foi feito no framework com a atividade: desenvolvimento de incrementos do sistema, porém não foi definida a partir dessa prática do form.

## *4.5 Especificação do Modelo de Domínio*

A representação de um modelo de domínio em uma LP pode ser feita de diferentes maneiras. Normalmente é utilizado o chamado Modelo de Features para desenvolver essa importante etapa da engenharia de domínio.

A modelagem das características é uma atividade para identificar as características externamente visíveis dos produtos. A classificação destas características é baseada em termos de recursos, o domínio de tecnologias, técnicas de implementação e ambientes operacionais [KANG e LEE, 2002].

Para a definição desta tarefa foi considerado o padrão Feature-Oriented Domain Analysis (FODA) que apresenta as diretrizes para realização de toda a parte de engenharia de domínio, no entanto, iremos tomar como base as definições do modelo. A documentação técnica do FODA define o modelo como uma representação das características padrão de uma LP e suas relações. Essas relações representam uma estrutura lógica agrupando as características. Além disso, devem ser apontados quais recursos são opcionais e alternativos dentro dessa estrutura [FODA, 1990].

Quando falamos das características dos produtos, elas podem ter uma variação de significado conforme o método que é utilizado. O FODA considera como característica, qualquer conceito relevante ou aparente para vários interessados. Nem sempre é claro definir essa unidade do modelo, portanto é válido destacar que o foco

do modelo é expor estas características exaltando seus pontos em comum e sua variabilidade dentro da LP, ao invés de seus detalhes específicos [KANG e LEE, 2002].

A partir destas características, a seguir será descrita a representação do modelo definido para este trabalho. Ele foi definido respeitando os parâmetros apontados no relatório técnico do FODA [FODA, 1990] e no artigo Conceitos e Diretrizes de Modelagem de Características para LP que mostra uma abordagem mais prática [KANG e LEE, 2002].

### 4.5.1 Representação do modelo

A abordagem utilizada na elaboração deste trabalho considera que cada característica recebe uma classificação de acordo como estará condicionada no sistema:

- Obrigatória
- Opcional
- Alternativa
- Ou

Também se utilizou dos seguintes relacionamentos para representação do modelo de domínio:

- Composição
- Generalização/Especificação
- Implementação

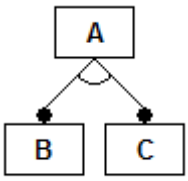
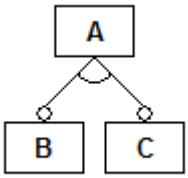
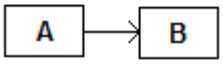
Além disso, as dependências entre as características são:

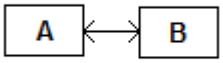
- Requer
- Exclui

Todas estas informações não representadas graficamente conforme a notação descrita na tabela a seguir:

**Tabela 6 - Notação do modelo de domínio proposto**

| Recurso | Representação gráfica | Descrição |
|---|---|---|

| | | |
|---|---|---|
| Obrigatória |  | São as características comuns para os diferentes produtos. |
| Opcional |  | São características selecionáveis dentro do modelo de acordo com cada produto. |
| Alternativa |  | É um conjunto de características onde apenas uma pode ser selecionada. De acordo com a imagem somente que o nó A for selecionado o produto deve conter apenas uma característica do conjunto {B, C}. Esse conjunto pode ser pensado como especializações de A. |
| Ou |  | É um conjunto de características onde uma ou mais delas podem ser selecionada. De acordo com a imagem somente que o nó A for selecionado o produto deve conter qualquer combinação de característica do conjunto {B, C}. Esse conjunto pode ser pensado como especializações de A. |
| Composição |  | Quando uma característica é compreendida por todo o nó inferior |
| Generalização/ Especificação |  | Dependendo do sentido da hierarquia, há generalização quando um nó superior apresentar essa relação para “n” filhos e o inverso para especialização. |
| Implementação |  | Indica que uma característica implementa um recurso de outra. |
| Requer |  | É uma dependência entre características onde A requer a seleção de B. |

| | | |
|---|---|---|
| Exclui |  | É uma dependência de exclusão mutua onde a seleção de uma característica exclui a seleção da outra. |

# 5 Exemplo de Uso da Proposta

Neste capítulo será descrita parte do desenvolvimento de uma LP seguindo os passos propostos nesse trabalho. A LP que será apresentada possuirá três produtos, são eles:

- Gerenciador eletrônicos de documentos (GED) – é uma ferramenta on-line que permite o acesso a documentos através de um navegador web. Permite o gerenciamento, controle, armazenamento, compartilhamento e visualização das informações dos documentos. Deve garantir o acesso dos documentos de forma ágil e segura. Além disso, armazena os documentos em um formato interno que permite a pesquisa indexada e ainda é possível assinar digitalmente um documento.
- Gerenciador de arquivos (GA) – é um serviço para organizar o armazenamento de arquivos numa estrutura de diretórios. Esse aplicativo web deve permitir o envio de arquivos em formato binário, controle dos diretórios, grupos de usuários, manipulação de arquivos, gerenciamento do espaço livre. Esse aplicativo terá uma interface web.
- Agenda de compromissos – é uma ferramenta on-line acessada por navegador que oferece uma agenda para cadastrar compromissos de um usuário individualmente ou de um grupo de usuários pertencentes a uma organização. Deve permitir a criação de compromissos, atribuição de compromissos para um grupo de usuários, alerta via e-mail de compromissos, confirmação de presença no compromisso em grupo, edição de um compromisso e anexar um arquivo aos compromissos.

Tendo em vista esse domínio de aplicações a execução do framework é apresentada a seguir.

## *5.1 Tecnologias previstas*

Para o desenvolvimento deste núcleo de componentes de produtos, é previsto a utilização de tecnologias de implementação que não serão detalhadas ao longo deste relatório. De qualquer forma, vale destacar o uso das seguintes tecnologias:

- Java 1.6;
- JSF 2.0;

- Tomcat 6.0;
- Richfaces 4.0;
- Ferramenta Eclipse Índigo.

## 5.2 Processo definido na Engenharia de Domínio

Cada uma das atividades e práticas propostas pelo framework estendido é apresentada na tabela abaixo indicando, na segunda coluna, quais delas foram incluídas no processo de desenvolvimento e execução da Engenharia de domínio desse exemplo de uso.

**Tabela 7 - Processo da Engenharia de domínio**

| | Framework Estendido | |
|---|---|---|
| Engenharia de Domínio | Lista de requisitos de cada aplicação (obrigatório) | ☑ |
| | Modelo de domínio (obrigatório) | ☑ |
| | Documentação | |
| | Projeto da arquitetura do sistema (obrigatório) | ☑ |
| | Representação da arquitetura de domínio (obrigatório) | ☑ |
| | Planejamento da iteração (obrigatório) | ☑ |
| | Duração da iteração (obrigatório) | ☑ |
| | Distribuição dos módulos para os responsáveis | |
| | Reunião diária | ☑ |
| | Projeto da iteração | |
| | Desenvolvimento da iteração (obrigatório) | ☑ |
| | Escrita dos testes de unidades | |
| | Desenvolvimento coletivo do código | |
| | Programação em pares | |
| | Refatoração | |
| | Programação lado-a-lado | |
| | Inspeção do código | |

## 5.3 Passos da Engenharia de Domínio

### 5.3.1 Lista de requisitos

Abaixo estão descritos os requisitos que farão parte da LP. Aqueles que serão reutilizados em todas as três aplicações serão destacados como requisito “comum”.

- Suporte e edição de documentos – um documento deve ser enviado, removido e editado pelo sistema.
- Suporte e edição de arquivos – um arquivo deve ser enviado, movido e removido do sistema. Os arquivos devem ter um dono e permitir o seu acesso a um grupo de usuários.

- Suporte e edição de compromissos – um compromisso deve ser criado no sistema, ter um dono e poder ser associado a um grupo de usuários.
- Visualização de documentos – deve existir um visualizador e editor para os principais formatos de documentos (jpg, gif, doc, odt, pdf, html, XML, etc).
- Conversão de formato dos documentos – o sistema deve converter o formato dos documentos para qualquer formato interno.
- Suporte a arquivos binários – deve manter os arquivos em seu formato original.
- Recuperação de arquivos – deve existir um local que mantenha os últimos arquivos removidos.
- Download e upload (comum) – deve enviar e receber os arquivos e documentos.
- Pesquisa indexada – dever realizar a pesquisa de documentos de maneira indexada.
- Controle de espaço – o espaço para armazenamento de cada usuário deve ser gerenciado.
- Gerência de usuários (comum) – é a gerência de acesso dos usuários ao sistema.
- Controle de acesso aos arquivos – o arquivo pode permitir sua leitura e edição para um grupo de usuários.
- Registro de usuários (comum) – os usuários do sistema devem estar cadastrados.
- E-mail – o sistema deve enviar e-mail para os usuários cadastrados.
- Cobrança – o acesso ao sistema pode ser cobrado mensalmente.
- Assinatura digital – deve permitir que os usuários assinem digitalmente seus arquivos e documentos.

### 5.3.2 Modelo de Domínio

As características da LP são descritas através do seguinte modelo (Figura 6):

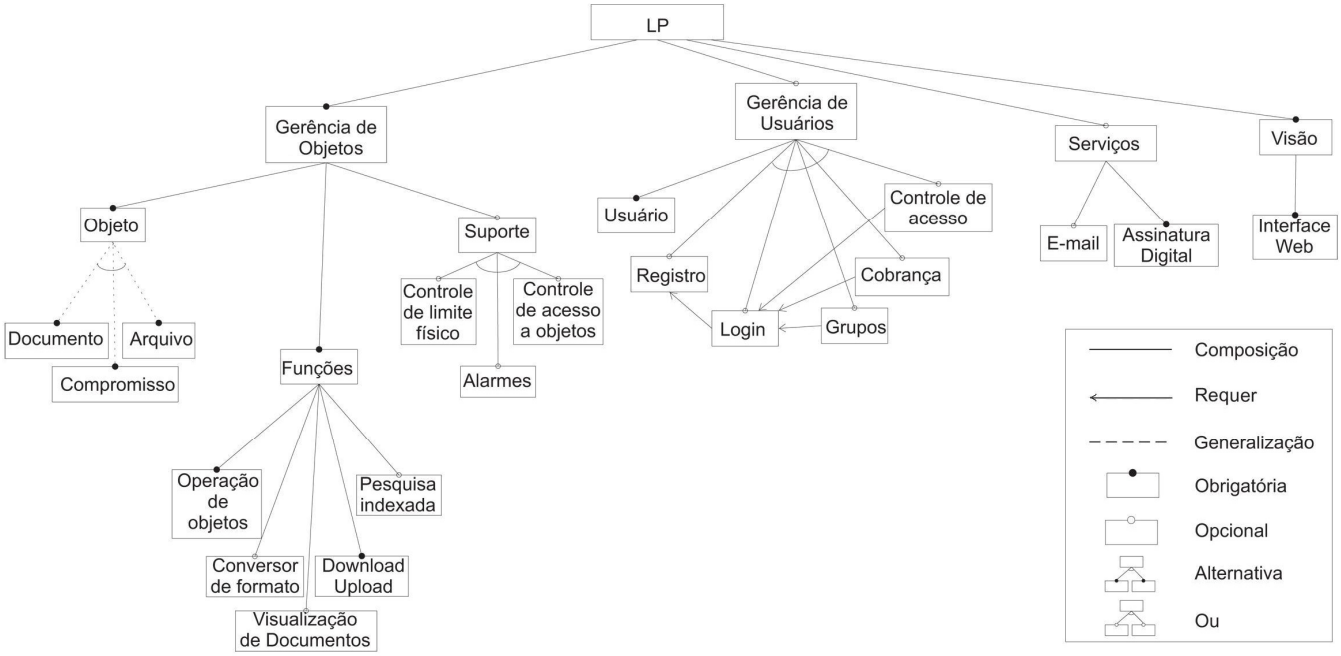



**Figura 6 - Modelo de dominio desenvolvido da LP**

### 5.3.3 Projeto da arquitetura do sistema

**Definir um componente**

Os componentes que farão parte do núcleo de artefatos com os pontos comuns para esta LP são:

- Objeto
- Operação de objetos
- Download/Upload
- Interface Web

Os componentes que farão parte dos artefatos com os pontos de variação para esta LP são:

- Conversor de formatos
- Visualização de documentos
- Pesquisa indexada
- Controles
- Alarmes
- Gerencia de usuário

- E-mail
- Assinatura digital

### 5.3.4 Representação da arquitetura de domínio

A arquitetura da LP é descrita através do seguinte diagrama de pacotes:

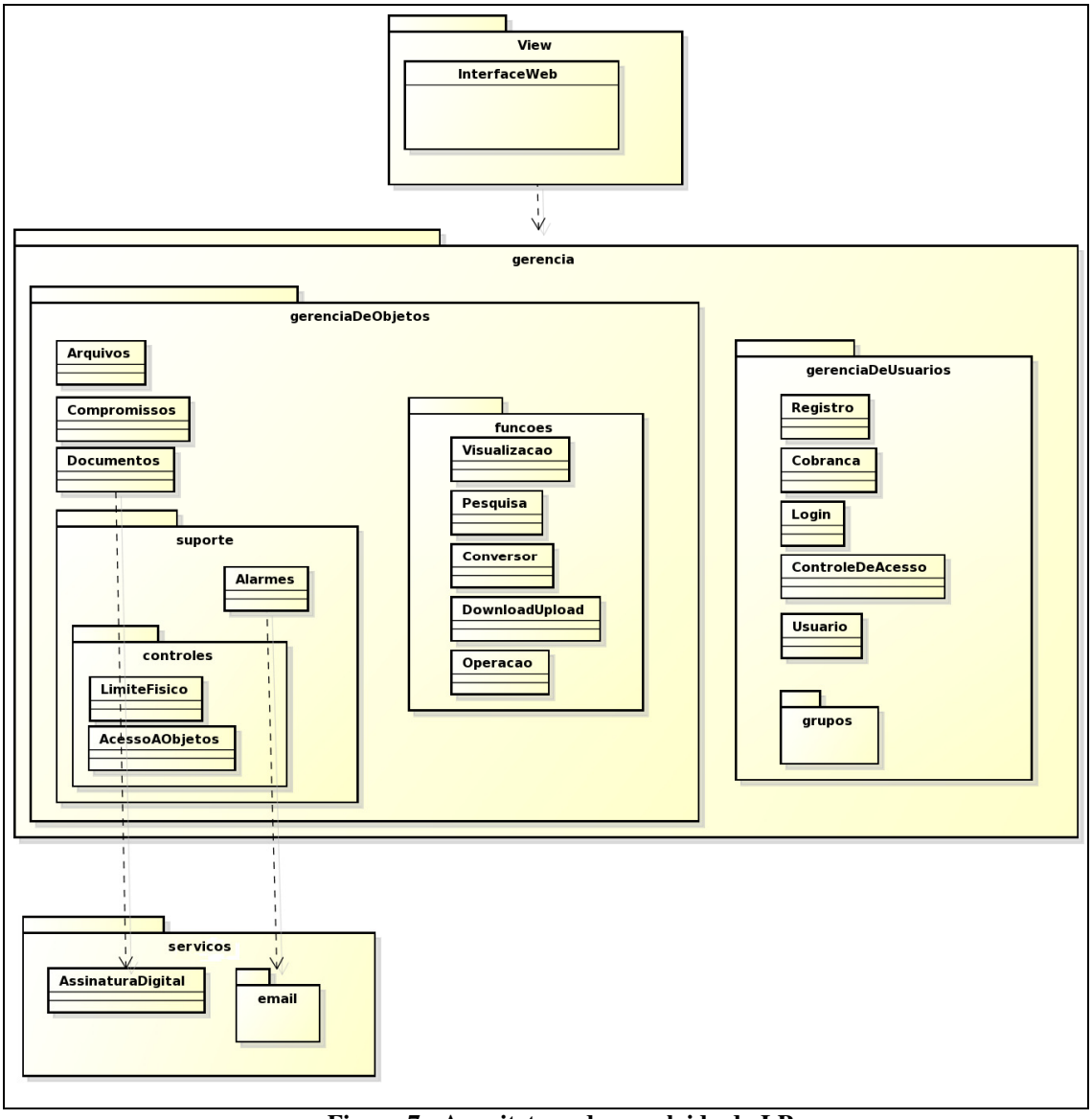



**Figura 7 - Arquitetura desenvolvida da LP**

Para o desenvolvimento das iterações são definidos dois colaboradores, o autor deste documento atuando como programador e sua orientadora como gerente de projeto.

### 5.3.5 Iteração 1

#### 5.3.5.1 Planejamento da iteração

Desenvolvimento do componente de Gerência de Objetos: inclui a codificação dos objetos Documento, Arquivos e Compromissos.

#### 5.3.5.2 Duração da iteração

A iteração deve durar uma semana.

#### 5.3.5.3 Distribuição dos módulos para os responsáveis

O responsável pelo desenvolvimento é o programador da equipe.

#### 5.3.5.4 Reunião diária

As reuniões ocorrem entre o programador e a gerente de projetos para avaliar o andamento da iteração.

#### 5.3.5.5 Desenvolvimento da iteração

A codificação dos componentes resultou nas classes destacadas na figura abaixo:

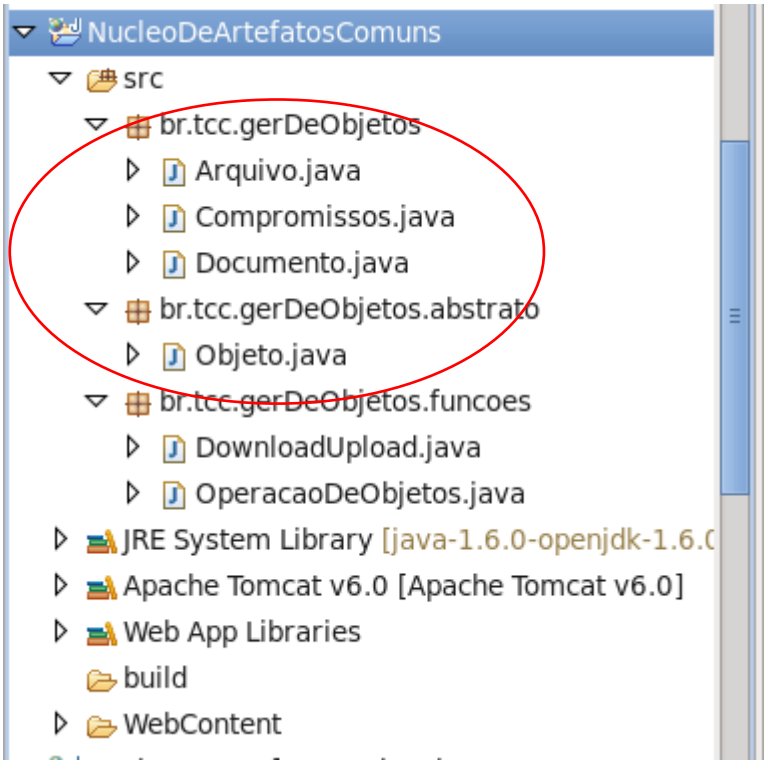



**Figura 8 - Codificação de gerência de objetos**

### 5.3.6 Iteração 2

#### 5.3.6.1 Planejamento da iteração

Desenvolvimento do componente que realizam as seguintes funções: Download/Upload e Operação de objetos.

#### 5.3.6.2 Duração da iteração

A iteração deve durar uma semana.

#### 5.3.6.3 Distribuição dos módulos para os responsáveis

O responsável pelo desenvolvimento é o programador da equipe.

#### 5.3.6.4 Reunião diária

As reuniões ocorrem entre o programador e a gerente de projetos para avaliar o andamento da iteração.

#### 5.3.6.5 Desenvolvimento da iteração

A codificação dos componentes resultou nas classes destacadas na figura abaixo:

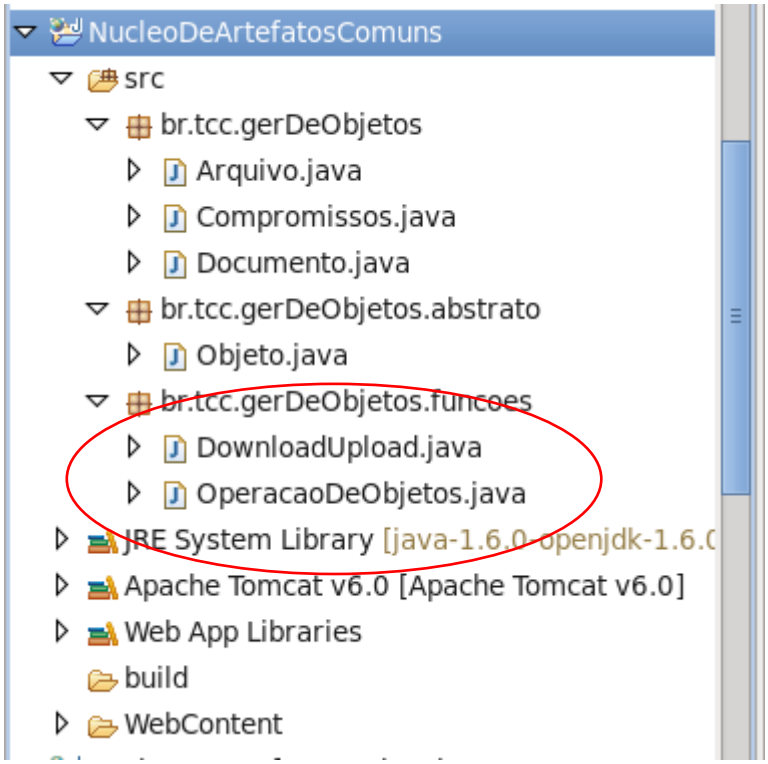



**Figura 9 - Codificação das funções de objetos**

### 5.3.7 Iteração 3

#### 5.3.7.1 Planejamento da iteração

Essa terceira iteração foi destinada ao desenvolvimento dos pontos de variação do sistema. Inclui um visualizador para os documentos e o módulo de Gerência de usuários.

#### 5.3.7.2 Duração da iteração

A iteração deve durar uma semana.

#### 5.3.7.3 Distribuição dos módulos para os responsáveis

O responsável pelo desenvolvimento é o programador da equipe.

#### 5.3.7.4 Reunião diária

As reuniões ocorrem entre o programador e a gerente de projetos para avaliar o andamento da iteração.

#### 5.3.7.5 Desenvolvimento da iteração

O desenvolvimento resultou nas seguintes classes abaixo:

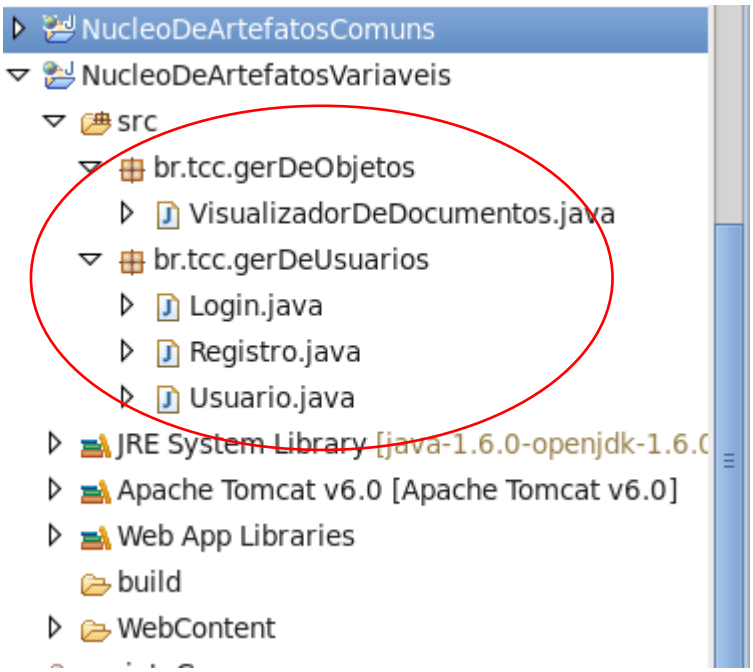



**Figura 10 - Codificação dos pontos de variação**

## 5.4 Processo definido na Engenharia de Aplicação

Cada uma das atividades e práticas propostas pelo framework estendido é apresentada na tabela abaixo indicando, na segunda coluna, quais delas foram incluídas no processo de desenvolvimento da engenharia de aplicação e implementação de um produto desse exemplo de uso.

**Tabela 8 - Processo da Engenharia de aplicação**

| | Framework Estendido | |
|---|---|---|
| Aplicação | Lista de requisitos (Obrigatório) | ☑ |
| | Seleção dos componentes (Obrigatório) | ☑ |
| | Modelo geral | ☑ |
| | Visão executiva – Documentação | |
| | Visão do projeto – Documentação | |
| | Planejamento da iteração (Obrigatório) | ☑ |
| | Duração de iterações (Obrigatório) | ☑ |
| | Distribuição de requisitos para os responsáveis (Obrigatório) | ☑ |
| | Reuniões diárias | |
| | Histórias do usuário | |

| | | |
|---|---|---|
| **Engenharia de** | Casos de uso | |
| | Projeto da iteração | |
| | Desenvolvimento da iteração (Obrigatório) | ☑ |
| | Escrita dos testes de unidade | |
| | Escrita dos testes de aceitação | |
| | Desenvolvimento coletivo do código | |
| | Programação em pares | |
| | Refatoração | |
| | Programação lado-a-lado | |
| | Inspeção de código | |
| | Reunião de revisão da integração | |
| | Geração de uma breve documentação | |
| | Entrega do sistema ao cliente | |

## *5.5 Passos da Engenharia de Aplicação – Primeiro produto*

A primeira aplicação selecionada para ser desenvolvida durante essa etapa foi o GED, que realiza o gerenciamento, controle, armazenamento, compartilhamento e visualização das informações dos documentos. Além disso, o GED armazena os documentos em um formato interno que permite a pesquisa indexada pelo nome e seu conteúdo. É possível ainda assinar digitalmente um documento.

### 5.5.1 Lista de requisitos

- Suporte e edição de documentos – um documento deve ser enviado, removido e editado pelo sistema.
- Visualização de documentos – deve existir um visualizador e editor para os principais formatos de documentos (jpg, gif, txt, doc, odt, pdf, html, xml, etc).
- Conversão de formato dos documentos – o sistema deve converter o formato dos documentos para qualquer formato interno que permita a indexação do seu conteúdo.
- Download e upload – deve enviar e receber os documentos.
- Pesquisa indexada – deve realizar a pesquisa de documentos de maneira indexada.
- Gerência de usuários – é a gerência de acesso dos usuários ao sistema.
- Registro de usuários – os usuários do sistema devem estar cadastrados.
- Assinatura digital – deve permitir que os usuários assinem digitalmente seus documentos.

### 5.5.2 Seleção dos componentes

- Interface Web
- Gerência de objetos
- Suporte
- Controle de acesso
- Funções
- Gerência de Usuários
- Grupos
- Serviços

### 5.5.3 Modelo geral

A especificação do modelo desenvolvido na LP é o seguinte:

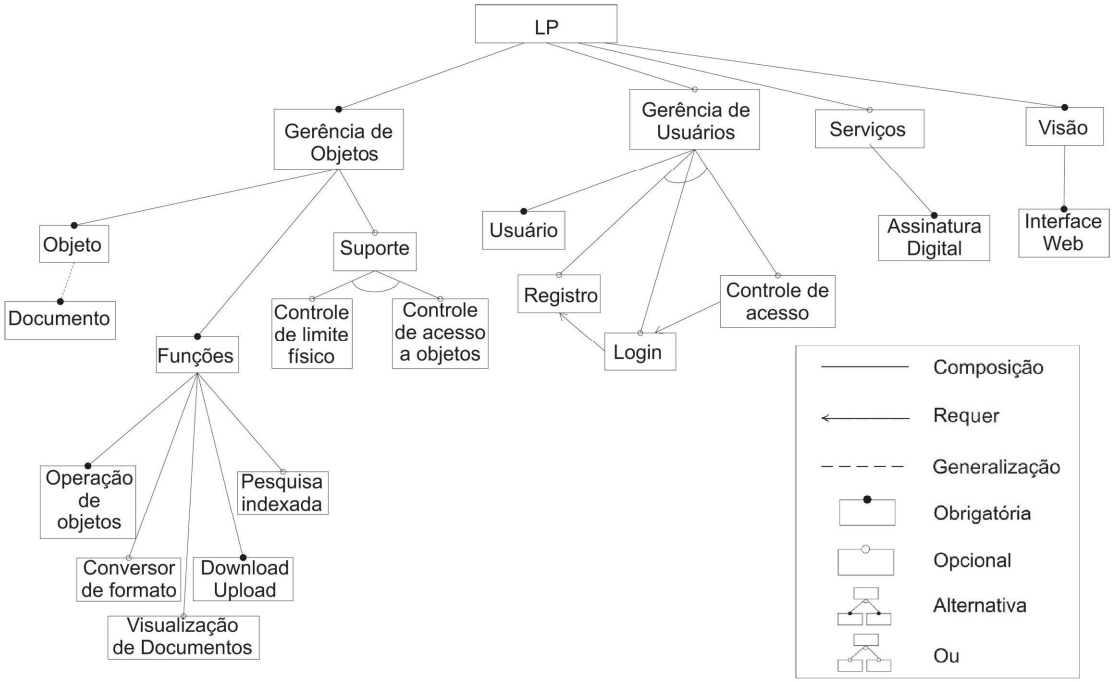



**Figura 11 - Modelo de domínio especializado do GED**

Para o desenvolvimento das iterações são definidos dois colaboradores, o autor deste documento atuando como programador e sua orientadora como gerente de projeto. Uma parte do aplicativo será implementada conforme a descrição das próximas seções. Inicialmente o GED permite o registro do usuário, seu acesso no sistema, o envio de um documento, remover o documento do sistema e edição de documento de texto.

### 5.5.4 Iteração 1

#### 5.5.4.1 Planejamento da iteração

Na primeira iteração foi feita a camada de visão com a interface com suas páginas web e seus respectivos ManagedBeans.

#### 5.5.4.2 Duração da iteração

A iteração deve durar uma semana.

#### 5.5.4.3 Distribuição de requisitos para os responsáveis

O responsável pelo desenvolvimento é o programador da equipe.

#### 5.5.4.4 Desenvolvimento da iteração

O desenvolvimento resultou nas páginas e classes abaixo:

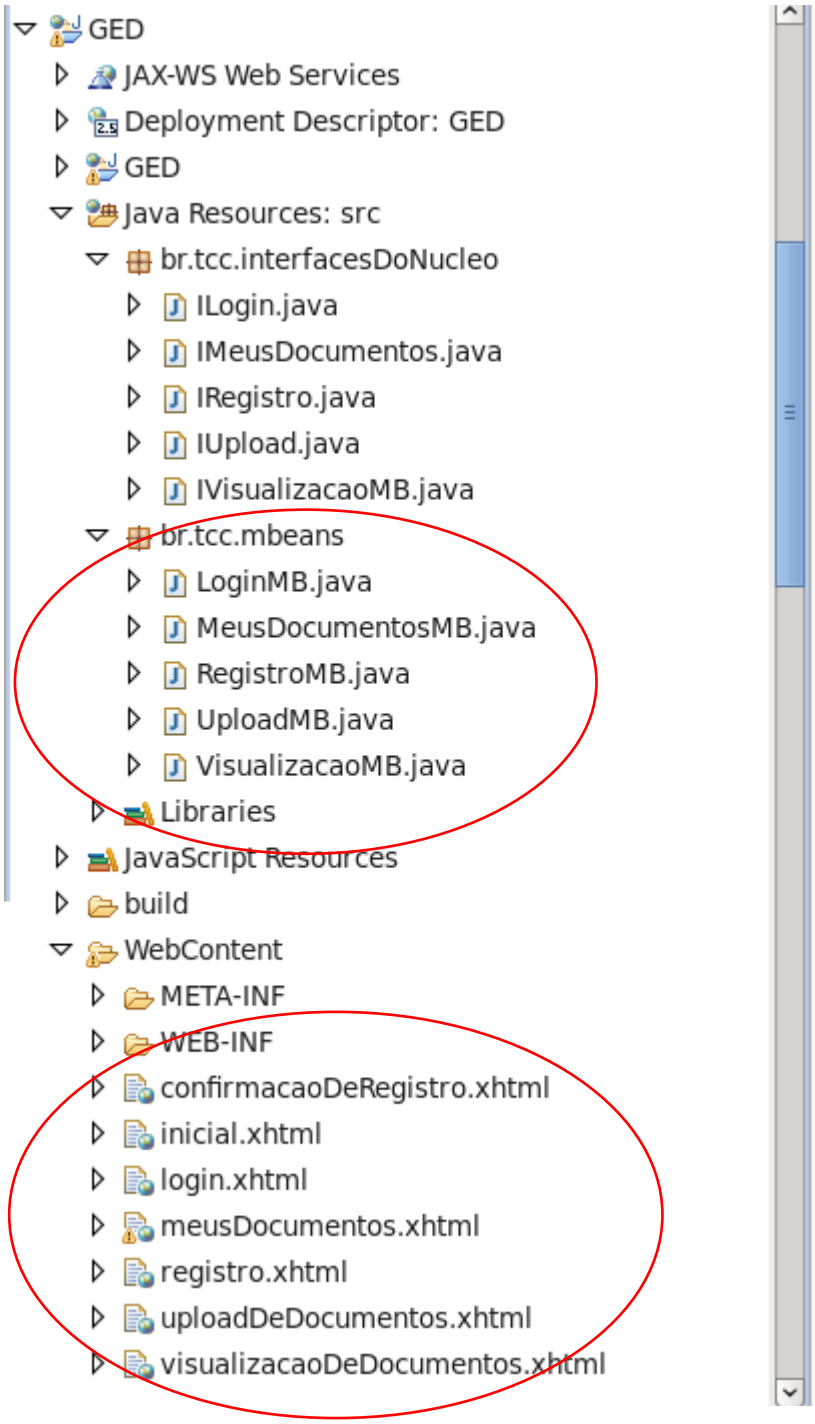



**Figura 12 - Codificação da visão**

### 5.5.5 Iteração 2

#### 5.5.5.1 Planejamento da iteração

Nesta iteração foi desenvolvida uma camada que faz a interface entre os ManagedBeans e o núcleo de componentes.

#### 5.5.5.2 Duração da iteração

A iteração deve durar uma semana.

#### 5.5.5.3 Distribuição de requisitos para os responsáveis

O responsável pelo desenvolvimento é o programador da equipe.

#### 5.5.5.4 Desenvolvimento da iteração

Resultou nas seguintes interfaces:

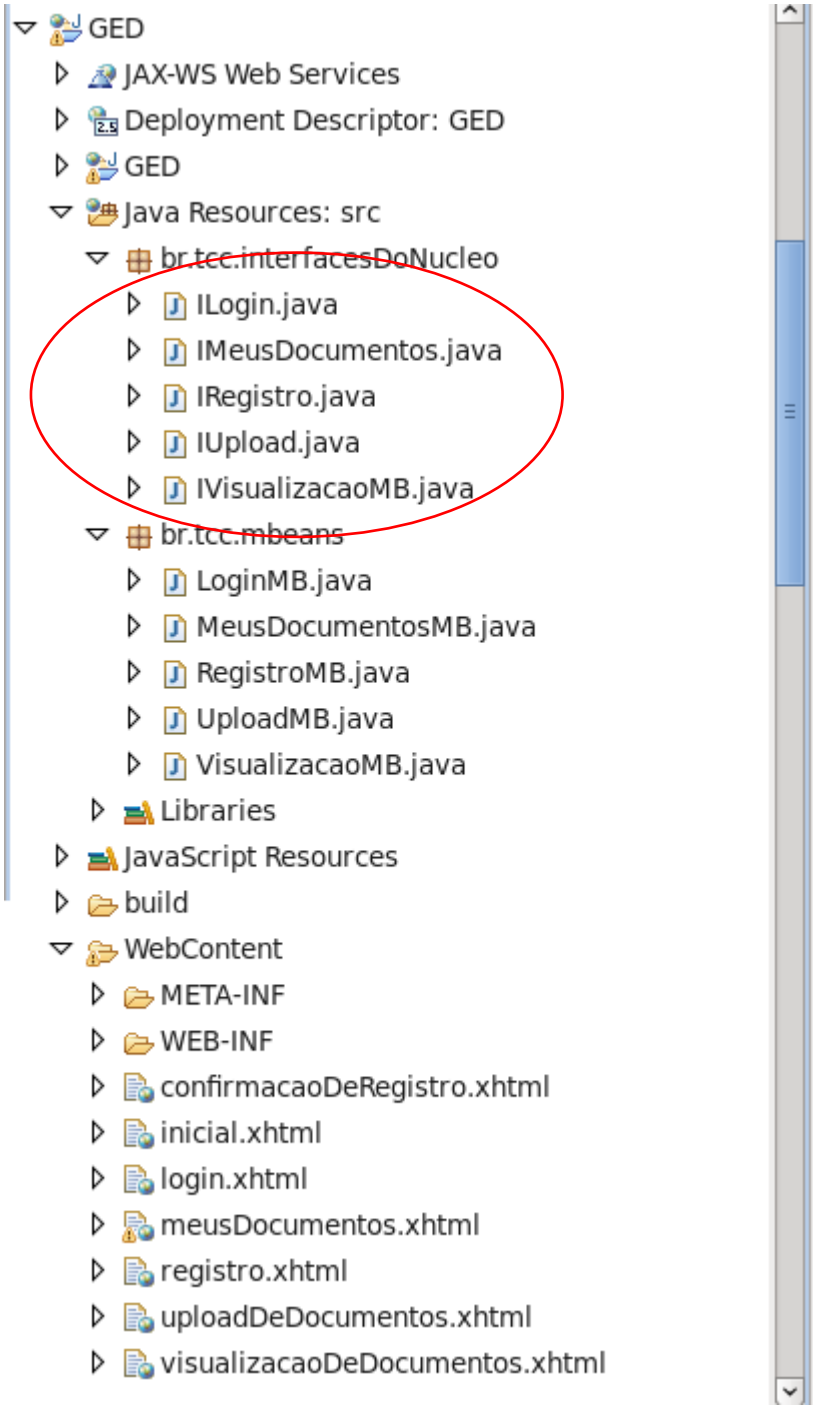



**Figura 13 - Codificação da camada de interface**

Dessa maneira concluímos a implementação de parte do exemplo de uso do framework onde avaliamos as suas etapas e utilização.

### 5.5.6 Interface com o usuário do GED

A implementação do GED apresenta as telas descritas a seguir.

- Uma tela para o registro de usuário:

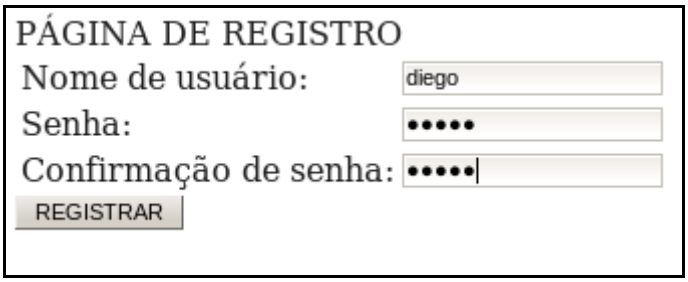



**Figura 14 - Tela de registro do GED**

- Uma tela para o usuário se identificar no sistema informando o nome de usuário e senha:


**Figura 15 - Tela de acesso do GED**

- Tela de navegação do sistema:

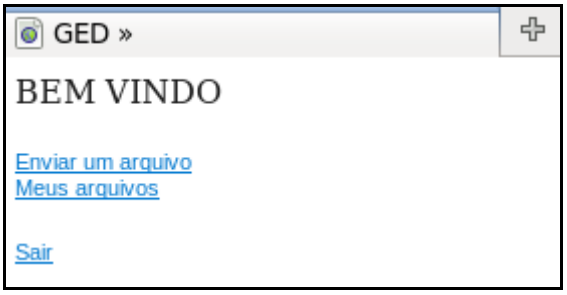



**Figura 16 - Tela de navegação do GED**

- Tela para visualizar os documentos inseridos no sistema:

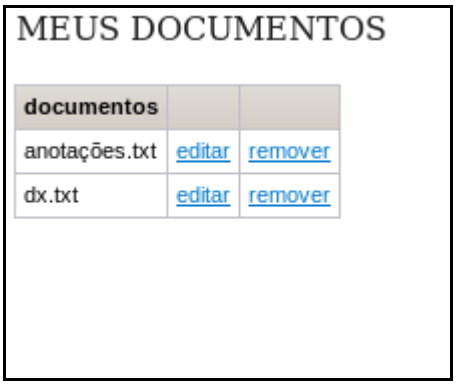



**Figura 17 - Tela de documentos do GED**

- Tela para enviar os documentos:

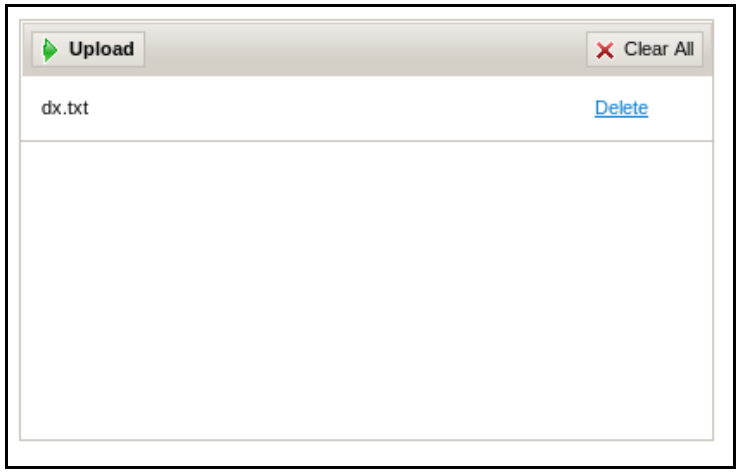



**Figura 18 - Tela de envio de documentos do GED**

- Tela de edição dos documentos:

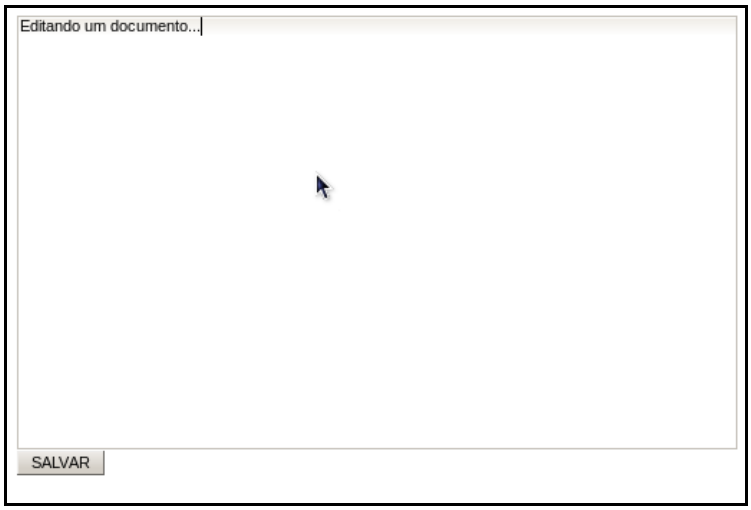



**Figura 19 - Tela de edição do GED**

## *5.6 Passos da Engenharia de Aplicação – Segundo produto*

A segunda aplicação desenvolvida foi a Agenda de Compromissos, que permite a criação de compromissos, atribuição de compromissos para um grupo de usuários, alerta via e-mail de compromissos, confirmação de presença no compromisso em grupo, edição de um compromisso e anexar um arquivo aos compromissos.

### 5.6.1 Lista de requisitos

- Suporte e edição de compromissos – um compromisso deve ser criado no sistema, ter um dono e poder ser associado a um grupo de usuários.
- Download e upload – deve enviar e receber os compromissos, alem de arquivos como anexos.
- Gerencia de usuários – é a gerência de acesso dos usuários ao sistema.
- Registro de usuários – os usuários do sistema devem estar cadastrados.
- E-mail – o sistema deve enviar e-mail para os usuários cadastrados.

### 5.6.2 Seleção dos componentes

- Compromissos
- Operação de objetos
- Download/Upload
- Controle de acesso a objetos
- Alarmes
- Gerencia de usuários
- E-mail
- Interface Web

### 5.6.3 Modelo geral

A especificação do modelo desenvolvido na LP é o seguinte:

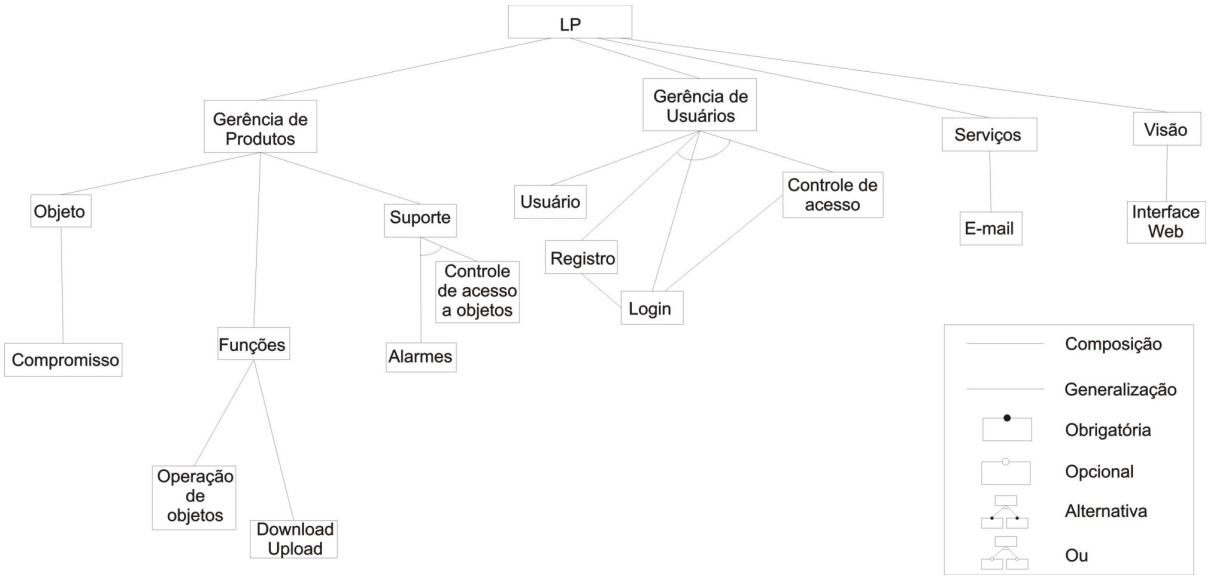



**Figura 20 - Modelo de domínio especializado da agenda**

Para o desenvolvimento das iterações são definidos dois colaboradores, o autor deste documento atuando como programador e sua orientadora como gerente de projeto. Uma parte do aplicativo será implementada conforme a descrição das próximas seções. Inicialmente a agenda permite o registro do usuário, seu acesso no sistema, criação de um compromisso e visualização de um calendário com os compromissos do dia.

## 5.6.4 Iteração 1

### 5.6.4.1 Planejamento da iteração

Na primeira iteração foi feita a camada de visão com a interface com suas páginas web e seus respectivos ManagedBeans.

### 5.6.4.2 Duração da iteração

A iteração deve durar uma semana.

### 5.6.4.3 Distribuição de requisitos para os responsáveis

O responsável pelo desenvolvimento é o programador da equipe.

### 5.6.4.4 Desenvolvimento da iteração

O desenvolvimento resultou nas páginas e classes abaixo:

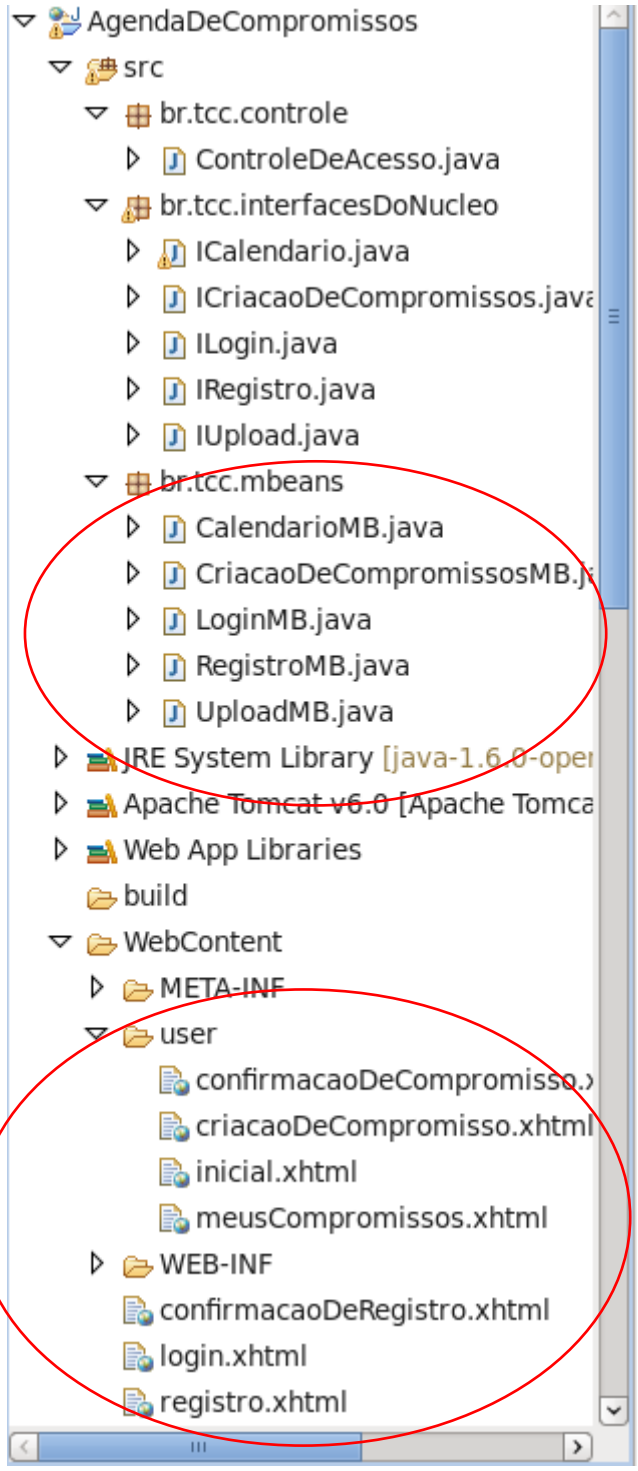



**Figura 21 - Codificação da visão**

### 5.6.5 Iteração 2

#### 5.6.5.1 Planejamento da iteração

Nesta iteração foi desenvolvida uma camada que faz a interface entre os ManagedBeans e o núcleo de componentes.

#### 5.6.5.2 Duração da iteração

A iteração deve durar uma semana.

#### 5.6.5.3 Distribuição de requisitos para os responsáveis

O responsável pelo desenvolvimento é o programador da equipe.

#### 5.6.5.4 Desenvolvimento da iteração

Resultou nas seguintes interfaces:

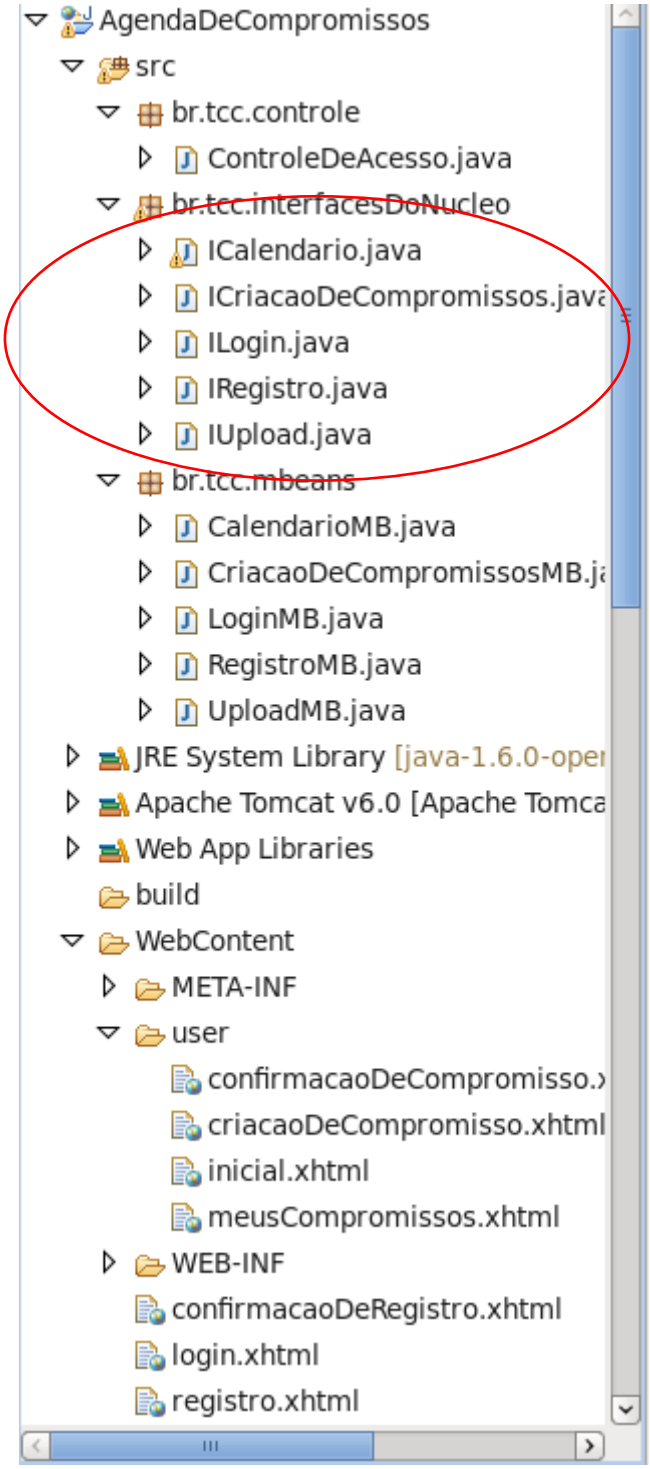



**Figura 22 - Codificação da camada de interface**

Dessa maneira concluímos a implementação de parte do exemplo de uso do framework onde avaliamos as suas etapas e utilização.

### 5.6.6 Interface com o usuário da Agenda de Compromissos

A implementação da agenda apresenta as telas descritas a seguir.

- Uma tela para o registro de usuário:

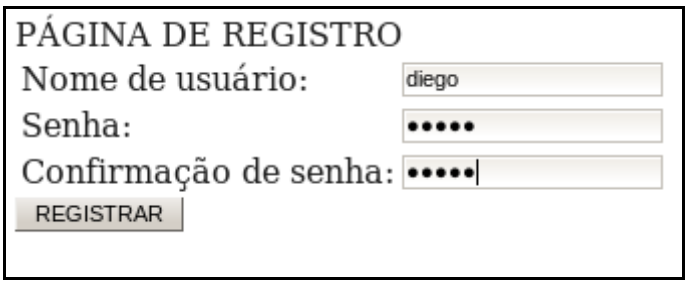



**Figura 23 - Tela de registro da agenda**

- Uma tela para o usuário se identificar no sistema informando o nome de usuário e senha:


**Figura 24 - Tela de acesso da agenda**

- Tela de navegação do sistema:

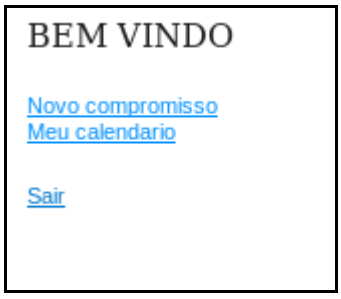



**Figura 25 - Tela de navegação da agenda**

- Tela para criar os compromissos:


**Figura 26 - Tela de criação de compromissos da agenda**

- Tela do calendário para apresentar os compromissos:


**Figura 27 - Tela do calendário da agenda**

## *5.7 Passos da Engenharia de Aplicação – Terceiro produto*

A aplicação selecionada para ser desenvolvida durante essa etapa foi o Gerenciador de arquivos que armazena arquivos numa estrutura de diretórios

permitindo o envio de arquivos em formato binário, controle dos diretórios, grupos de usuários, manipulação de arquivos, gerenciamento do espaço livre.

### 5.7.1 Lista de requisitos

- Suporte e edição de arquivos – um arquivo deve ser enviado, movido e removido do sistema. Os arquivos devem ter um dono e permitir o seu acesso a um grupo de usuários.
- Recuperação de arquivos – deve existir um local que mantenha os últimos arquivos removidos.
- Download e upload – deve enviar e receber os arquivos.
- Controle de espaço – o espaço para armazenamento de cada usuário deve ser gerenciado.
- Controle de acesso aos arquivos – o arquivo pode permitir sua leitura e edição para um grupo de usuários.
- Registro de usuários – os usuários do sistema devem estar cadastrados.

### 5.7.2 Seleção dos componentes

- Arquivos
- Operação de objetos
- Download/Upload
- Controle de limite físico
- Controle de acesso a objetos
- Gerencia de usuários
- Interface Web

### 5.7.3 Modelo geral

A especificação do modelo desenvolvido na LP é o seguinte:

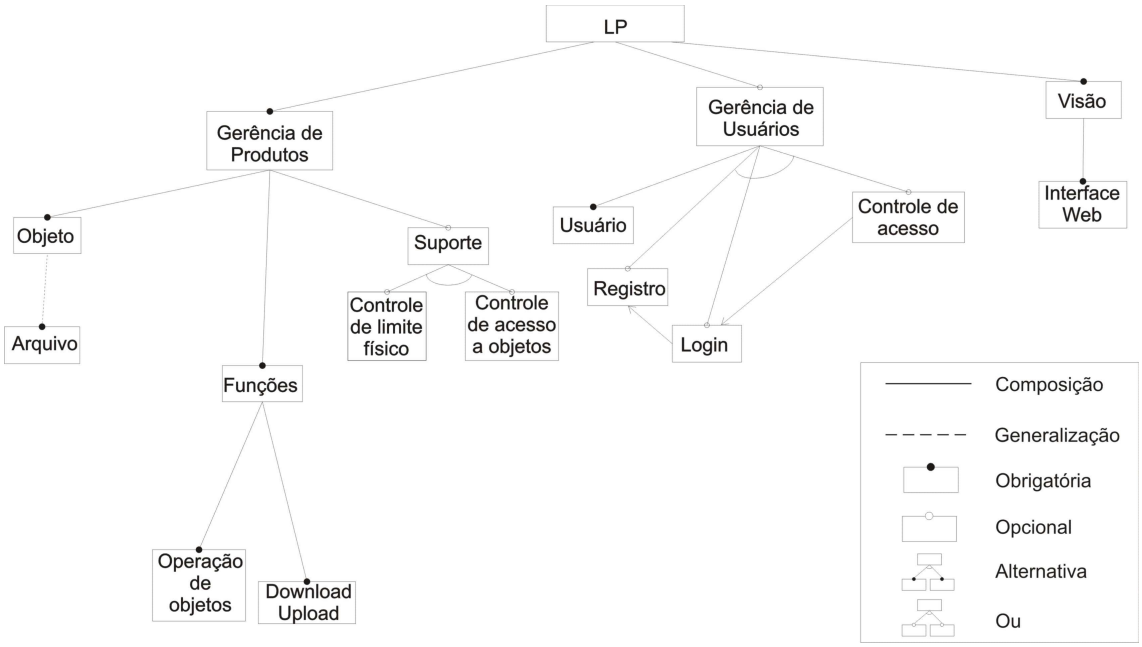



**Figura 28 - Modelo de domínio especializado do gerenciador de arquivos**

Para o desenvolvimento das iterações são definidos dois colaboradores, o autor deste documento atuando como programador e sua orientadora como gerente de projeto. Uma parte do aplicativo será implementada conforme a descrição das próximas seções. Inicialmente o gerenciador de arquivos permite o envio de arquivos criando seus diretórios e a visualização dos arquivos em uma estrutura de árvore.

## 5.7.4 Iteração 1

### 5.7.4.1 Planejamento da iteração

Na primeira iteração foi feita a camada de visão com a interface com suas páginas web e seus respectivos ManagedBeans.

### 5.7.4.2 Duração da iteração

A iteração deve durar uma semana.

### 5.7.4.3 Distribuição de requisitos para os responsáveis

O responsável pelo desenvolvimento é o programador da equipe.

### 5.7.4.4 Desenvolvimento da iteração

O desenvolvimento resultou nas páginas e classes abaixo:

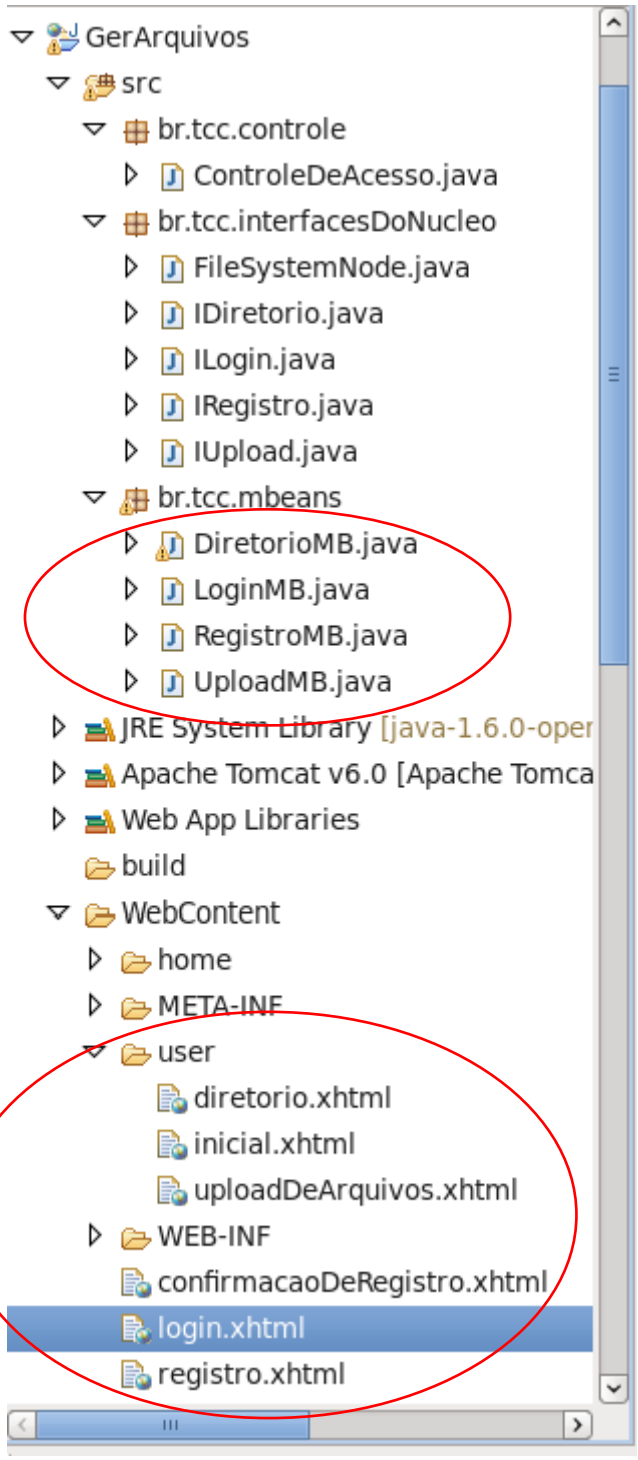



**Figura 29 - Codificação da visão**

### 5.7.5 Iteração 2

#### 5.7.5.1 Planejamento da iteração

Nesta iteração foi desenvolvida uma camada que faz a interface entre os ManagedBeans e o núcleo de componentes.

#### 5.7.5.2 Duração da iteração

A iteração deve durar uma semana.

#### 5.7.5.3 Distribuição de requisitos para os responsáveis

O responsável pelo desenvolvimento é o programador da equipe.

#### 5.7.5.4 Desenvolvimento da iteração

Resultou nas seguintes interfaces:

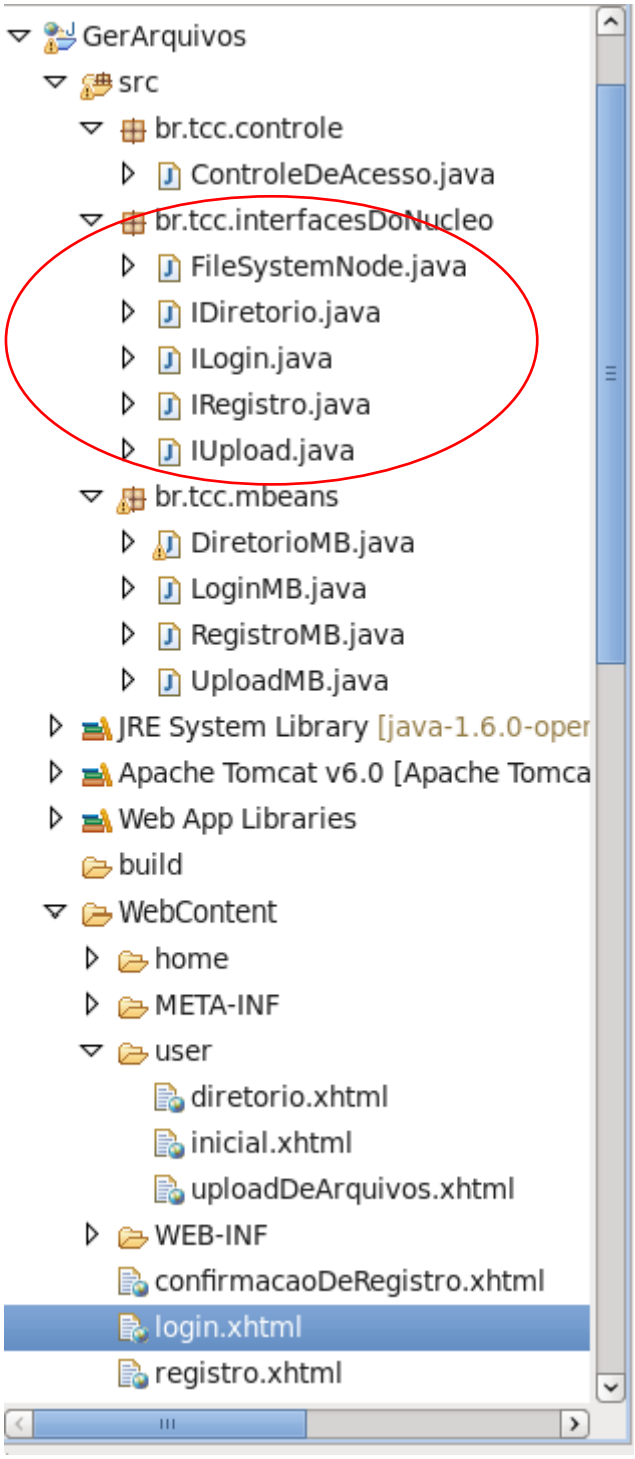



**Figura 30 - Codificação da camada de interface**

Dessa maneira concluímos a implementação de parte do exemplo de uso do framework onde avaliamos as suas etapas e utilização.

### 5.7.6 Interface com o usuário do Gerenciador de Arquivos

A implementação do gerenciador apresenta as telas descritas a seguir.

- Uma tela para o registro de usuário:

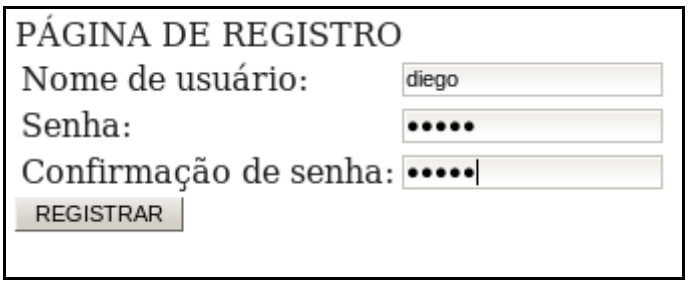



**Figura 31 - Tela de registro do gerenciador de arquivos**

- Uma tela para o usuário se identificar no sistema informando o nome de usuário e senha:


**Figura 32 - Tela de acesso do gerenciador de arquivos**

- Tela de navegação do sistema:

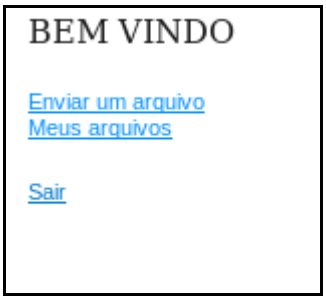



**Figura 33 - Tela de navegação do sistema**

- Tela para enviar os arquivos

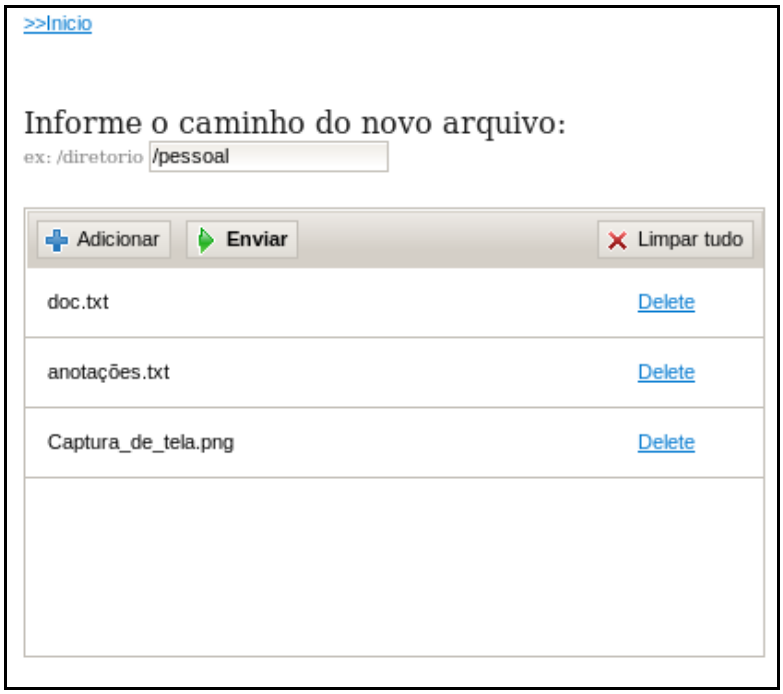



**Figura 34 - Tela de envio de arquivos do gerenciador**

- Tela visualizar os arquivos

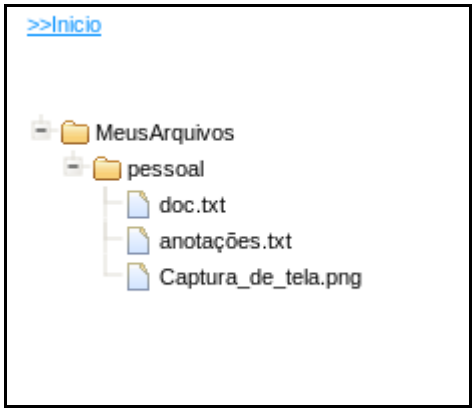



**Figura 35 - Tela de visualização de arquivos do gerenciador**

## 5.8 Componentes reutilizados nas aplicações

As aplicações implementadas na engenharia de aplicação, apesar de serem reduzidas, já puderam utilizar os componentes do núcleo de artefatos, ou seja, foi possível aproveitar os componentes reutilizáveis do sistema. A tabela abaixo apresenta as características que foram implementadas em cada aplicação, destacando a reutilização de alguns artefatos com total compatibilidade:

**Tabela 9 - Componentes reutilizados**

| | GED | Agenda de compromissos | Gerenciador de arquivos |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Objeto | ☑ | ☑ | ☑ |
| Operação de objetos | ☑ | ☑ | ☑ |
| Conversor de formato | | | |
| Visualização de documentos | ☑ | | |
| Download Upload | ☑ | ☑ | ☑ |
| Pesquisa indexada | | | |
| Controle de limite físico | | | |
| Alarmes | | ☑ | |
| Controle de acesso a objetos | | | |
| Usuário | ☑ | ☑ | ☑ |
| Login | ☑ | ☑ | ☑ |
| Registro | ☑ | ☑ | ☑ |
| Grupos | | | |
| Cobrança | | | |
| Controle de acesso | ☑ | ☑ | ☑ |
| E-mail | | ☑ | |
| Assinatura digital | | | |
| Interface web | ☑ | ☑ | ☑ |

# 6 Conclusão

O estudo inicial e de fundamentação para este trabalho procurou reunir as definições e principalmente os pontos conflitantes entre métodos ágeis e LP de software. Com base nas características de cada um destes temas, foi analisada a possibilidade de execução da proposta, tendo como resposta um panorama positivo.

Foram analisados métodos de desenvolvimento ágil de software apresentando alguns deles e principalmente o framework que reúne as características de cada um deles apresentando uma abordagem bastante completa dos passos e práticas que essa técnica de engenharia de software proporciona.

O aprofundamento desse estudo se deu com a análise de como o desenvolvimento de uma LP pode ser executado. As etapas de desenvolvimento de um núcleo de artefatos com a engenharia de domínio e a reutilização desse núcleo no desenvolvimento de cada produto com a engenharia de aplicação foram os principais objetos dessa abordagem.

Consequentemente, estes dois grandes eixos estudados foram combinados para dar origem às definições do framework composto nesse trabalho. Neste ponto foram comparados os temas que originaram o grande confronto de ideias desse tipo de abordagem onde o desenvolvimento de uma LP exige uma documentação completa e estudo detalhado do domínio e a metodologia ágil simplifica ou até mesmo exclui grande parte da documentação e etapas do desenvolvimento tradicional de software. O alvo desta etapa permitiu identificar alguns pontos que são essenciais para que esse estudo alcançasse o seu objetivo: o confronto dessas idéias deve ocorrer de forma ponderada e iterativa. Ponderada, pois os dois lados dessa balança devem estar equilibrados para que o desenvolvimento de uma LP de forma ágil ocorra. E iterativa, pois para alcançar esse objetivo, ele deve ser conquistado em pequenas tarefas que se repetem.

Se comparado o framework desenvolvido com os trabalhos relacionados, esta proposta ressalta um processo que prioriza a agilidade do sistema e ainda assim permite o desenvolvimento de uma LP. Ao invés de introduzir características ágeis a um processo de desenvolvimento de uma LP, foi feito o processo inverso, incluindo as tarefas necessárias para o desenvolvimento de uma LP ao processo ágil proposto por [FAGUNDES, 2005]. Além disso, não há a necessidade de partir de produtos

existentes para realizar a engenharia de domínio e extrair o núcleo de artefatos. Com isto foi proposto o framework que admite a seleção de suas práticas, removendo aquelas que não se aplicam no processo da LP que será desenvolvida. Dessa forma, proporciona ainda mais agilidade ao processo.

O desenvolvimento do aplicativo como exemplo de utilização do framework, apesar de se tratar de um projeto bastante reduzido, proporcionou a avaliação das funcionalidades do framework e acima de tudo garantiu que as etapas para o desenvolvimento de uma LP podem ser cumpridas seguindo essa metodologia, sem que haja um esforço além do que é tradicionalmente proposto pelos métodos ágeis.

A realização deste estudo apontou que esta proposta de desenvolvimento é válida e possível dentro da engenharia de software e acima de tudo, que pode ser feito de uma forma simples e eficiente.

Como trabalho futuros relacionados com essa pesquisa fica a possibilidade de:

- Aprofundamento do estudo comparando mais padrões de desenvolvimento para LP e métodos ágeis.
- O desenvolvimento de uma LP e suas aplicações utilizando este framework para diferentes domínios.
- A comparação de tempo de desenvolvimento e custos entre o desenvolvimento ágil uma LP a partir do framework estendido, e o desenvolvimento tradicional de uma LP.

# 7 Referências

[ALMEIDA, 2007] ALMEIDA, Eduardo Santana. RiDE: The RiSE Process for Domain Engineering. Disponível em <http://www.ivanmachado.com.br/research/rise/thesis/> acessado em maio de 2012.

[BALBINO, ALMEIDA e MEIRA, 2011] BALBINO, Marcela; ALMEIDA, Eduardo S.; MEIRA, Silvio. An Agile Scoping Process for Software Product Lines. The 23rd International Conference on Software Engineering & Knowledge Engineering. Miami, 2011.

[CHIN, 2004] CHIN, Gary. Agile project management: how to succeed in the face of changing project requirements. NY: Amacon, 2004.

[CLEMENTS, 2002] CLEMENTS, Paul; NORTHROP, Linda. Software Product Lines: Practices and Patterns. Boston: Addison-Wesley, 2002, 563 p.

[CONALLEN, 1999] CONALLEN, Jim, Modeling Web Application Architetures with UML, Communications of ACM, v. 42, n. 10, October 1999. (1999a).

[ESM15, 2009] SCAICO, Pasqueline. SCAICO, Alexandre. LIMA, Filipe L C. Linhas de produto de software: Introdução, conceitos e desafios para sua adotação. Engenharia de Software Magazine, Rio de Janeiro, 15ª ed, 48-53.2009

[EXTREME PROGRAMING, 2011] Extreme Programming: A gentle introduction. Disponível em <http://www.extremeprogramming.org/> acessado em dezembro de 2011.

[FAGUNDES, 2005] FAGUNDES, P. B. Framework Para Comparação e Análise de Métodos Ágeis. 2005. 134f. Dissertação (Mestrado em Ciência da Computação) – Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis, 2005.

[FODA, 1990] KANG, Kyo C. , COHEM, Sholom G. , HESS, James A. , NOVAK, William E. e PETERSON, A. Spencer. Disponível em <http://www.sei.cmu.edu/reports/90tr021.pdf> acessado em maio de 2012.

[FORM, 1998] KANG Kyo, C. , KIM, Sajoong, LEE, Jaejoon,KIM, Kijoo, KIM, Gerard Jounghyun e SHIN, Euiseob. FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. Disponível em <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.95.7568> acessado em maio de 2012.

[FOWLER, 2005] FOWLER, Martin. The new methodology. Dezembro de 2005. Disponível em <http://www.martinfowler.com/articles/newMethodology.html> acesso em novembro de 2011.

[GHANAM e MAURER, 2008] GHANAM, Yaser e MAURER, Frank. An Iterative Model for Agile Product Line Engineering. Disponível em <http://ebe.cpsc.ucalgary.ca/ebe/uploads/APLE/splc2008.pdf> acessado em maio de 2012.

[HIGHSMITH, 2002] HIGHSMITH, Jim. Agile software development ecosystems. Boston: Addison-Wesley, 2002.

[KANG e LEE, 2002] LEE, Kwanwoo, KANG, Kyo C. e LEE, Jaejoon. Disponível em <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.98.8783/> acessado em maio de 2012.

[KRUEGER, 2011] KRUEGER, Charles W. Introduction to Software Product Lines. Disponível em <http://www.softwareproductlines.com/> acessado em dezembro de 2011.

[LEE, 2002] Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. Disponível em <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.98.8783> acessado em maio de 2012.

[MANIFESTO, 2001] Manifesto Ágil. Disponível em <http://manifestoagil.com.br/>, acessado em dezembro de 2011.

[MESO e JAIN, 2006] MESO Peter, JAIN Radhika. Agile Software Development: Adaptive Systems Principles and Best Practices. Disponível em <http://www.ism-journal.com/ITToday/93704.pdf>, acessado em maio de 2012.

[MOHAM, RAMESH e SUGUMARAN, 2010] MOHAM, Kannan; RAMESH Baruch C. B. e SUGUMARAN, Vijayan. Integrating Software Product Line Engineering and Agile Development. . IEEE Software, vol27 nº3, 48-55.2010

[NORTHROP e CLEMENTS, 2007] NORTHROP, Linda; CLEMENTS, Paul. A Framework for Software Product Line Practice, Version 5.0. Disponível em <http://www.sei.cmu.edu/productlines/frame_report/index.html> acessado em dezembro de 2011.

[SANCHEZ, 2007] SANCHEZ, Ivan. Scrum em 2 minutos. Disponível em <http://dojofloripa.wordpress.com/2007/02/07/scrum-em-2-minutos/> acessado em dezembro de 2011.

[SCHWABER e SUTHERLAND, 2011] SCHWABER, K. e SUTHERLAND, J. The Scrum Guide the official rulebook. 2011. Disponível em: <http://www.scrum.org/> acessado em novembro de 2011.

[SEI, 2011] Software Product Lines - Overview. Disponível em <http://www.sei.cmu.edu/productlines/> acessado em dezembro de 2011.

[TELES, 2004] TELES, Vinicius M. Extreme Programming. Disponível em <http://novatec.com.br/livros/extreme/> acessado em dezembro de 2011.

# 8 Anexos

## 8.1 Código fonte

### Núcleo de Artefatos Comuns

Objeto:

```
package br.tcc.gerDeObjetos.abstrato;

import java.io.InputStream;

public class Objeto {

	protected String nome;
	protected InputStream dado;

	public Objeto(String nome, InputStream dado){
		this.nome = nome;
		this.dado = dado;
	}

	public void setNome(String nome){
		this.nome = nome;
	}

	public void setDado(InputStream dado){
		this.dado = dado;
	}

	public String getNome() {
		return nome;
	}

	public InputStream getDado() {
		return dado;
	}

}
```

DownloadUpload:

```
package br.tcc.gerDeObjetos.funcoes;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;

import br.tcc.gerDeObjetos.abstrato.Objeto;

public class DownloadUpload {

	private static final String HOME = "/home/diego/tcc/";
```

```
	public void enviarObjeto(String usuario, InputStream is, String
nome){
		try {
			String caminho = "";
			for(String s: usuario.split("/")){
				caminho += s+"/";
				File dir = new File(HOME+caminho);
				if(!dir.isDirectory()){
					dir.mkdir();
				}
			}

			FileOutputStream out = new
FileOutputStream(HOME+usuario+"/"+nome);
			BufferedReader reader = new BufferedReader( new
InputStreamReader(is));
			String linha = "";
			while( ( linha = reader.readLine() ) != null ){
				out.write((linha+"\n").getBytes());
			}
			out.close();
		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	public InputStream retornaObjeto(String nomeDoDocumento, String
usuario) {
		InputStream is = null;
		try {
			is = new
FileInputStream(HOME+usuario+"/"+nomeDoDocumento);
		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		return is;
	}

	public ArrayList<Objeto> retornaListaDeObjeto(String usuario) {
		File dir = new File(HOME+usuario+"/");
		if(!dir.isDirectory()){
			dir.mkdir();
			return new ArrayList<Objeto>();
		}
		ArrayList<Objeto> listaDeObjetos = new ArrayList<Objeto>();
		for(File f: dir.listFiles()){
			try {
				listaDeObjetos.add(new Objeto(f.getName(), new
FileInputStream(f)) );
			} catch (FileNotFoundException e) {
			}
		}

		return listaDeObjetos;
	}

	public String getPath(String usuario){
		return HOME+usuario+"/";
```

```
	}
}
```

OperacaoDeObjetos:

```
package br.tcc.gerDeObjetos.funcoes;

import java.io.File;

public class OperacaoDeObjetos {

	public boolean removeObjeto(String nomeDoDocumento, String
usuario) {
		File f =  new
File("/home/diego/tcc/"+usuario+"/"+nomeDoDocumento);
		if(f.isFile()){
			f.delete();
			return true;
		}
		return false;
	}
}
```

Arquivo:

```
package br.tcc.gerDeObjetos;

import java.io.InputStream;

import br.tcc.gerDeObjetos.abstrato.Objeto;

public class Arquivo extends Objeto{

	private String diretorio;

	public Arquivo(String nome, InputStream dado) {
		super(nome, dado);
	}

	public void setDiretorio(String diretorio) {
		this.diretorio = diretorio;
	}

	public String getDiretorio() {
		return diretorio;
	}

}
```

Compromissos:

```
package br.tcc.gerDeObjetos;

import java.io.InputStream;
import java.util.Date;

import br.tcc.gerDeObjetos.abstrato.Objeto;

public class Compromissos extends Objeto {

	private Date dataDoCompromisso;
	private String conteudo;
```

```
	public Compromissos(String nome, InputStream dado) {
		super(nome, dado);
	}

	public Date getDataDoCompromisso() {
		return dataDoCompromisso;
	}

	public void setDataDoCompromisso(Date dataDoCompromisso) {
		this.dataDoCompromisso = dataDoCompromisso;
	}

	public String getConteudo() {
		return conteudo;
	}

	public void setConteudo(String conteudo) {
		this.conteudo = conteudo;
	}

}
```

Documento:

```
package br.tcc.gerDeObjetos;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;

import br.tcc.gerDeObjetos.abstrato.Objeto;

public class Documento extends Objeto {

	private ArrayList<String> indiceDePesquisa = new
ArrayList<String>();

	public Documento(String nome, InputStream dado) {
		super(nome, dado);

		BufferedReader reader = new BufferedReader(new
InputStreamReader(dado));
		String leitura;
		try {
			while((leitura = reader.readLine())!=null){
				for(String i:leitura.split(" ")){
					indiceDePesquisa.add(i);
				}
			}
		} catch (IOException e) {
		}

	}

	public boolean find(String chave){
		return indiceDePesquisa.contains(chave);
	}

	public ArrayList<String> getIndiceDePesquisa() {
		return indiceDePesquisa;
```

```
	}


}
```

## Núcleo de Artefatos Variáveis

Alarmes:

```
package br.tcc.gerDeObjetos.suporte;

import java.util.Date;
import java.util.Timer;

import br.tcc.servicos.Email;

public class Alarmes {

	public void novoEmail(String email, Date data) {
		Timer timer = new Timer();

		Email tarefa = new Email(email);

		timer.schedule(tarefa, data);
	}

}
```

VisualizadorDeDocumentos:

```
package br.tcc.gerDeObjetos;

import java.io.InputStream;
import java.util.ArrayList;

import br.tcc.gerDeObjetos.abstrato.Objeto;
import br.tcc.gerDeObjetos.funcoes.DownloadUpload;


public class VisualizadorDeDocumentos {
	public Documento retornaDocumento(String usuario, String
nomeDoDocumento){
		DownloadUpload download = new DownloadUpload();
		InputStream is = download.retornaObjeto(nomeDoDocumento,
usuario);
		if (is == null){
			return null;
		}
		return new Documento(usuario, is);
	}

	public ArrayList<Documento> retornaListaDeDocumento(String
usuario){
		DownloadUpload download = new DownloadUpload();
		ArrayList<Objeto> lista =
download.retornaListaDeObjeto(usuario);
		ArrayList<Documento> listaDeDocumentos = new
ArrayList<Documento>();
		for(Objeto is : lista){
			listaDeDocumentos.add(new Documento(is.getNome(),
is.getDado()));
		}
```

```
            return listaDeDocumentos;
      }
}
```

Login:

```
package br.tcc.gerDeUsuarios;

import java.util.ArrayList;

public class Login {

      public boolean conectarUsuario(String nome, String senha){
            ArrayList<Usuario> listaDeUsuarios =
Registro.getListaDeUsuarios();
            for(Usuario u : listaDeUsuarios){
                  if(u.getNome().equals(nome)){
                        if(u.getSenha().equals(senha)){
                              u.setLogado(true);
                              return true;
                        }
                  }
            }

            return false;
      }

      public void desconectarUsuario(String nome){
            ArrayList<Usuario> listaDeUsuarios =
Registro.getListaDeUsuarios();
            for(Usuario u : listaDeUsuarios){
                  if(u.getNome().equals(nome)){
                        u.setLogado(false);
                  }
            }
      }

      public boolean usuarioLogado(String nome){
            ArrayList<Usuario> listaDeUsuarios =
Registro.getListaDeUsuarios();
            for(Usuario u : listaDeUsuarios){
                  if(u.getNome().equals(nome)){
                        return u.isLogado();
                  }
            }

            return false;
      }
}
```

Registro:

```
package br.tcc.gerDeUsuarios;

import java.util.ArrayList;

public class Registro {
      private static ArrayList<Usuario> listaDeUsuarios = new
ArrayList<Usuario>();

      public Usuario cadastrarUsuario(String nome,String senha){
```

```
            for(Usuario u : listaDeUsuarios){
                  if(u.getNome().equals(nome)){
                        return null;
                  }
            }

            Usuario novoUsuario = new Usuario(nome, senha);
            listaDeUsuarios.add(novoUsuario);
            return novoUsuario;
      }

      public static void setListaDeUsuarios(ArrayList<Usuario>
listaDeUsuarios) {
            Registro.listaDeUsuarios = listaDeUsuarios;
      }

      public static ArrayList<Usuario> getListaDeUsuarios() {
            return listaDeUsuarios;
      }
}
```

Usuario:

```
package br.tcc.gerDeUsuarios;

public class Usuario {
      private String nome;
      private boolean logado = false;
      private String senha;

      public Usuario(String nome, String senha) {
            this.nome = nome;
            this.senha = senha;
      }

      public String getNome(){
            return nome;
      }

      public String getSenha(){
            return senha;
      }

      public void setLogado(boolean logado) {
            this.logado = logado;
      }

      public boolean isLogado() {
            return logado;
      }
}
```

Email:

```
package br.tcc.servicos;

import java.util.Date;
import java.util.TimerTask;

import org.apache.commons.mail.HtmlEmail;

public class Email  extends TimerTask {
```

```
	String email;

	public Email(String endereco) {
		this.email = endereco;
	}

	@Override
	public void run() {
		String msg = "VocÃª tem um compromisso hoje.";
		String senha = "senha";
		String origem = "email";
		String host = "host.com";
		int porta = 465;

		HtmlEmail hMail = new HtmlEmail();
		hMail.setHostName(host);
		hMail.setAuthentication(origem, senha);
		hMail.setSmtpPort(porta);
		hMail.setSSL(true);
		hMail.setTLS(true);
		hMail.setSentDate(new Date());

		try {
			hMail.addTo(email);
			hMail.setFrom(origem, "Alerta");
			hMail.setSubject("ALERTA DE COMPROMISSO");
			hMail.setTextMsg(msg);
			hMail.send();
		} catch (Exception e) {
			e.printStackTrace();
		}

	}
}
```

**Agenda de Compromissos**

ControleDeAcesso:

```
package br.tcc.controle;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import br.tcc.gerDeUsuarios.Login;

public class ControleDeAcesso implements Filter {
	private static final String SIGNON_PAGE_URI =
"/faces/login.xhtml";

	public void init(FilterConfig filterConfig) throws
ServletException {
	}
```

```
	public void doFilter(ServletRequest req, ServletResponse res,
			FilterChain chain) throws IOException, ServletException
{
		HttpServletResponse response = (HttpServletResponse) res;
		HttpServletRequest request = (HttpServletRequest) req;

		if (!this.authorize((HttpServletRequest) req)) {

	request.getRequestDispatcher(SIGNON_PAGE_URI).forward(req, res);
		} else {
			response.setHeader("Cache-Control", "no-store");
			response.setHeader("Pragma", "no-cache");
			response.setDateHeader("Expires", 0);
			chain.doFilter(req, res);
		}
	}

	public void destroy() {
	}

	private boolean authorize(HttpServletRequest req) {
		HttpSession session = req.getSession(false);
		if (session != null) {
			String nomeDoUsuario = (String)
session.getAttribute("usuario");
			Login login = new Login();
			if (login != null && login.usuarioLogado(nomeDoUsuario)
== true) {
				return true;
			}
		}
		return false;
	}
}
```

ICalendario:

```
package br.tcc.interfacesDoNucleo;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;

import br.tcc.gerDeObjetos.Compromissos;
import br.tcc.gerDeObjetos.abstrato.Objeto;
import br.tcc.gerDeObjetos.funcoes.DownloadUpload;

public class ICalendario {

	public ArrayList<Compromissos> consultaCompromissos(String
usuario, Date data) {
		ArrayList<Compromissos> compromissosDoDia = new
ArrayList<Compromissos>();
		DownloadUpload download = new DownloadUpload();
		BufferedReader br;
		SimpleDateFormat sdf = new SimpleDateFormat("ddMMyyyy");
		String dataBase =  sdf.format(data);
		String conteudo, tmp;
```

```
            Compromissos c;
            for(Objeto o: download.retornaListaDeObjeto(usuario)){
                  conteudo = "";
                  br = new BufferedReader(new
InputStreamReader(o.getDado()));
                  try {
                        if (dataBase.equals(br.readLine())){
                              c = new Compromissos(o.getNome(),
o.getDado());
                              c.setDataDoCompromisso(data);
                              while((tmp = br.readLine())!=null){
                                    conteudo+=tmp+"\n";
                              }
                              c.setConteudo(conteudo);
                              compromissosDoDia.add(c);
                        }
                  } catch (IOException e) {
                        e.printStackTrace();
                  }
            }

            return compromissosDoDia;
      }

}
```

ICriacaoDeCompromisso:

```
package br.tcc.interfacesDoNucleo;

import java.io.ByteArrayInputStream;
import java.io.InputStream;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

import br.tcc.gerDeObjetos.funcoes.DownloadUpload;
import br.tcc.gerDeObjetos.suporte.Alarmes;

public class ICriacaoDeCompromissos {
      private DownloadUpload upload = new DownloadUpload();
      SimpleDateFormat dateFormat = null;

      public void novoCompromisso(String dia, String mes, String ano,
                  String descricao, String usuario) throws ParseException
{
            dateFormat = new SimpleDateFormat("ddMMyyyy");
            dateFormat.setLenient(false);
            dateFormat.parse(dia+mes+ano);
            InputStream inputStream = new
ByteArrayInputStream((dia+mes+ano+"\n"+descricao).getBytes());
            upload.enviarObjeto(usuario, inputStream,
"compromisso_"+System.currentTimeMillis());

      }

      public void novoCompromissoComAlarme(String dia, String mes,
String ano,
                  String descricao, String diaAlarme, String mesAlarme,
                  String anoAlarme, String hora, String email, String
usuario) throws ParseException {
            dateFormat = new SimpleDateFormat("ddMMyyyyHH:mm");
            dateFormat.setLenient(false);
```

```
            dateFormat.parse(dia+mes+ano+hora);

            Alarmes alarmes = new Alarmes();
            Date dataAlarme =
dateFormat.parse(diaAlarme+mesAlarme+anoAlarme+hora);
            alarmes.novoEmail(email, dataAlarme);

            InputStream inputStream = new
ByteArrayInputStream((dia+mes+ano+"\n"+descricao).getBytes());
            upload.enviarObjeto(usuario, inputStream,
"compromisso_"+System.currentTimeMillis());
      }
}
```

ILogin:

```
package br.tcc.interfacesDoNucleo;

import br.tcc.gerDeUsuarios.Login;

public class ILogin {

      public boolean logarUsuario(String nomeDeUsuario, String
senhaDeUsuario) {
            return new Login().conectarUsuario(nomeDeUsuario,
senhaDeUsuario);
      }

      public void deslogarUsuario(String nomeDeUsuario) {
            new Login().desconectarUsuario(nomeDeUsuario);
      }

}
```

IRegistro:

```
package br.tcc.interfacesDoNucleo;

import br.tcc.gerDeUsuarios.Registro;

public class IRegistro {

      public boolean registrarUsuario(String nome, String senha) {
            Registro registro = new Registro();

            if (registro.cadastrarUsuario(nome, senha) != null){
                  return true;
            }
            return false;
      }
}
```

IUpload:

```
package br.tcc.interfacesDoNucleo;

import java.io.InputStream;

import br.tcc.gerDeObjetos.funcoes.DownloadUpload;


public class IUpload {
```

```
	public void enviar(InputStream inputStream, String nome, String
usuario) {
		DownloadUpload upload = new DownloadUpload();
		upload.enviarObjeto(usuario, inputStream, nome);
	}

}
```

CalendarioMB:

```
package br.tcc.mbeans;

import java.util.ArrayList;
import java.util.Date;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.context.FacesContext;
import javax.faces.event.ActionEvent;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import br.tcc.gerDeObjetos.Compromissos;
import br.tcc.interfacesDoNucleo.ICalendario;


@ManagedBean(name ="calendarioMB")
@RequestScoped
public class CalendarioMB {
	private Date data;
	private ICalendario calendario = new ICalendario();
	private ArrayList<Compromissos> compromissosDoDia = new
ArrayList<Compromissos>();

	public void consultaData(ActionEvent e){
		FacesContext facesContext =
FacesContext.getCurrentInstance();
		HttpServletRequest request = (HttpServletRequest)
facesContext
				.getExternalContext().getRequest();
		HttpSession session = request.getSession();
		String usuario = (String) session.getAttribute("usuario");

	setCompromissosDoDia(calendario.consultaCompromissos(usuario,
data));
	}

	public void setData(Date data) {
		this.data = data;
	}

	public Date getData() {
		return data;
	}

	public void setCompromissosDoDia(ArrayList<Compromissos>
compromissosDoDia) {
		this.compromissosDoDia = compromissosDoDia;
	}

	public ArrayList<Compromissos> getCompromissosDoDia() {
		return compromissosDoDia;
```

```
	}

}
```

CriacaoDeCompromissoMB:

```
package br.tcc.mbeans;

import java.text.ParseException;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import br.tcc.interfacesDoNucleo.ICriacaoDeCompromissos;

@ManagedBean(name ="criacaoDeCompromissosMB")
@RequestScoped
public class CriacaoDeCompromissosMB {
	private boolean dataIncorreta = false;
	private String dia;
	private String mes;
	private String ano;
	private String diaAlarme;
	private String mesAlarme;
	private String anoAlarme;
	private String horaAlarme;
	private String email;
	private String descricao;
	private boolean definirAlarme = true;
	private ICriacaoDeCompromissos compromissos = new
ICriacaoDeCompromissos();

	public String novoCompromisso(){
		FacesContext facesContext =
FacesContext.getCurrentInstance();
		HttpServletRequest request = (HttpServletRequest)
facesContext
				.getExternalContext().getRequest();
		HttpSession session = request.getSession();
		String usuario = (String) session.getAttribute("usuario");

		if(definirAlarme){
			try {
				compromissos.novoCompromissoComAlarme(dia, mes,
ano, descricao, diaAlarme, mesAlarme, anoAlarme, horaAlarme, email,
usuario);
			} catch (ParseException e) {
				setDataIncorreta(true);
				return null;
			}
		}else {
			try {
				compromissos.novoCompromisso(dia, mes, ano,
descricao, usuario);
			} catch (ParseException e) {
				setDataIncorreta(true);
				return null;
			}
		}
		return "ok";
```

```
}

public void setDataIncorreta(boolean dataIncorreta) {
      this.dataIncorreta = dataIncorreta;
}

public boolean isDataIncorreta() {
      return dataIncorreta;
}

public String getDia() {
      return dia;
}

public void setDia(String dia) {
      this.dia = dia;
}

public String getMes() {
      return mes;
}

public void setMes(String mes) {
      this.mes = mes;
}

public String getAno() {
      return ano;
}

public void setAno(String ano) {
      this.ano = ano;
}

public String getDiaAlarme() {
      return diaAlarme;
}

public void setDiaAlarme(String diaAlarme) {
      this.diaAlarme = diaAlarme;
}

public String getMesAlarme() {
      return mesAlarme;
}

public void setMesAlarme(String mesAlarme) {
      this.mesAlarme = mesAlarme;
}

public String getAnoAlarme() {
      return anoAlarme;
}

public void setAnoAlarme(String anoAlarme) {
      this.anoAlarme = anoAlarme;
}

public String getHoraAlarme() {
      return horaAlarme;
}

public void setHoraAlarme(String horaAlarme) {
```

```
			this.horaAlarme = horaAlarme;
		}

		public String getDescricao() {
			return descricao;
		}

		public void setDescricao(String descricao) {
			this.descricao = descricao;
		}

		public boolean isDefinirAlarme() {
			return definirAlarme;
		}

		public void setDefinirAlarme(boolean definirAlarme) {
			this.definirAlarme = definirAlarme;
		}

		public void setEmail(String email) {
			this.email = email;
		}

		public String getEmail() {
			return email;
		}
}
```

LoginMB:

```
package br.tcc.mbeans;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import br.tcc.interfacesDoNucleo.ILogin;

@ManagedBean(name ="loginMB")
@RequestScoped
public class LoginMB {
		private String nomeDeUsuario;
		private String senhaDeUsuario;
		private boolean senhaIncorreta = false;
		private ILogin login = new ILogin();

		public String getSenhaDeUsuario() {
			return senhaDeUsuario;
		}

		public void setSenhaDeUsuario(String senhaDeUsuario) {
			this.senhaDeUsuario = senhaDeUsuario;
		}

		public void setNomeDeUsuario(String nomeDeUsuario) {
			this.nomeDeUsuario = nomeDeUsuario;
		}

		public String getNomeDeUsuario() {
			return nomeDeUsuario;
		}
```

```
	public String login(){

		if(login.logarUsuario(nomeDeUsuario, senhaDeUsuario)){
			FacesContext facesContext =
FacesContext.getCurrentInstance();
			HttpServletRequest request = (HttpServletRequest)
facesContext
					.getExternalContext().getRequest();
			HttpSession session = request.getSession();
			session.setAttribute("usuario", nomeDeUsuario);
			return "logado";
		}

		senhaIncorreta = true;
		return null;
	}

	public boolean isSenhaIncorreta() {
		return senhaIncorreta;
	}

	public String logout(){
		FacesContext facesContext =
FacesContext.getCurrentInstance();
		HttpServletRequest request = (HttpServletRequest)
facesContext
				.getExternalContext().getRequest();
		HttpSession session = request.getSession();
		String usuario = (String) session.getAttribute("usuario");
		session.removeAttribute("usuario");

		login.deslogarUsuario(usuario);
		return "sair";
	}
}
```

RegistroMB:

```
package br.tcc.mbeans;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;

import br.tcc.interfacesDoNucleo.IRegistro;

@ManagedBean(name = "registroMB")
@RequestScoped
public class RegistroMB {
	private String nome;
	private String senha;
	private String confirmacaoDeSenha;
	private boolean confirmacaoIncorreta;
	private IRegistro registro = new IRegistro();

	public String registrar(){

		if(senha.equals(confirmacaoDeSenha)){
			if(registro.registrarUsuario(nome, senha)){
				return "ok";
			}
		}
```

```
            confirmacaoIncorreta=true;
            return null;
      }

      public String getNome() {
            return nome;
      }

      public void setNome(String nome) {
            this.nome = nome;
      }

      public String getSenha() {
            return senha;
      }

      public void setSenha(String senha) {
            this.senha = senha;
      }

      public String getConfirmacaoDeSenha() {
            return confirmacaoDeSenha;
      }

      public void setConfirmacaoDeSenha(String confirmacaoDeSenha) {
            this.confirmacaoDeSenha = confirmacaoDeSenha;
      }

      public void setConfirmacaoIncorreta(boolean confirmacaoIncorreta)
{
            this.confirmacaoIncorreta = confirmacaoIncorreta;
      }

      public boolean isConfirmacaoIncorreta() {
            return confirmacaoIncorreta;
      }

}
```

UploadMB:

```
package br.tcc.mbeans;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import org.richfaces.event.FileUploadEvent;
import org.richfaces.model.UploadedFile;

import br.tcc.interfacesDoNucleo.IUpload;


@ManagedBean(name = "uploadMB")
@RequestScoped
public class UploadMB {

    public void listener(FileUploadEvent event) throws Exception {
        UploadedFile item = event.getUploadedFile();
        IUpload upload = new IUpload();
        FacesContext facesContext = FacesContext.getCurrentInstance();
```

```
            HttpServletRequest request = (HttpServletRequest)
facesContext
                        .getExternalContext().getRequest();
            HttpSession session = request.getSession();
            String usuario = (String) session.getAttribute("usuario");
          upload.enviar(item.getInputStream(), item.getName(), usuario);
    }

}
```

**Gerenciador Eletrônico de Documentos**

ControleDeAcesso:

```
package br.tcc.controle;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import br.tcc.gerDeUsuarios.Login;

public class ControleDeAcesso implements Filter {
      private static final String SIGNON_PAGE_URI =
"/faces/login.xhtml";

      public void init(FilterConfig filterConfig) throws
ServletException {
      }

      public void doFilter(ServletRequest req, ServletResponse res,
                  FilterChain chain) throws IOException, ServletException
{
            HttpServletResponse response = (HttpServletResponse) res;
            HttpServletRequest request = (HttpServletRequest) req;

            if (!this.authorize((HttpServletRequest) req)) {

      request.getRequestDispatcher(SIGNON_PAGE_URI).forward(req, res);
            } else {
                  response.setHeader("Cache-Control", "no-store");
                  response.setHeader("Pragma", "no-cache");
                  response.setDateHeader("Expires", 0);
                  chain.doFilter(req, res);
            }
      }

      public void destroy() {
      }

      private boolean authorize(HttpServletRequest req) {
            HttpSession session = req.getSession(false);
            if (session != null) {
```

```
                  String nomeDoUsuario = (String)
session.getAttribute("usuario");
                  Login login = new Login();
                  if (login != null && login.usuarioLogado(nomeDoUsuario)
== true) {
                        return true;
                  }
            }
            return false;
      }
}
```

ILogin:

```
package br.tcc.interfacesDoNucleo;

import br.tcc.gerDeUsuarios.Login;

public class ILogin {

      public boolean logarUsuario(String nomeDeUsuario, String
senhaDeUsuario) {
            return new Login().conectarUsuario(nomeDeUsuario,
senhaDeUsuario);
      }

      public void deslogarUsuario(String nomeDeUsuario) {
            new Login().desconectarUsuario(nomeDeUsuario);
      }

}
```

IMeusDocumentos:

```
package br.tcc.interfacesDoNucleo;

import java.util.ArrayList;

import br.tcc.gerDeObjetos.Documento;
import br.tcc.gerDeObjetos.VisualizadorDeDocumentos;
import br.tcc.gerDeObjetos.funcoes.OperacaoDeObjetos;

public class IMeusDocumentos {

      public ArrayList<Documento> retornalistaDeDocumentos(String
usuario) {
            return new
VisualizadorDeDocumentos().retornaListaDeDocumento(usuario);
      }

      public void remover(String nome, String usuario) {
            OperacaoDeObjetos oper = new OperacaoDeObjetos();
            oper.removeObjeto(nome, usuario);
      }

}
```

IRegistro:

```
package br.tcc.interfacesDoNucleo;

import br.tcc.gerDeUsuarios.Registro;
```

```
public class IRegistro {

	public boolean registrarUsuario(String nome, String senha) {
		Registro registro = new Registro();

		if (registro.cadastrarUsuario(nome, senha) != null){
			return true;
		}
		return false;
	}
}
```

IUpload:

```
package br.tcc.interfacesDoNucleo;

import java.io.InputStream;

import br.tcc.gerDeObjetos.funcoes.DownloadUpload;


public class IUpload {

	public void enviar(InputStream inputStream, String nome, String
usuario) {
		DownloadUpload upload = new DownloadUpload();
		upload.enviarObjeto(usuario, inputStream, nome);
	}

}
```

IVisualizacao:

```
package br.tcc.interfacesDoNucleo;

import br.tcc.gerDeObjetos.Documento;
import br.tcc.gerDeObjetos.VisualizadorDeDocumentos;

public class IVisualizacaoMB {

	public Documento retornaDocumento(String nome, String usuario) {
		VisualizadorDeDocumentos visualizacao = new
VisualizadorDeDocumentos();
		return visualizacao.retornaDocumento(usuario, nome);
	}

}
```

LoginMB:

```
package br.tcc.mbeans;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import br.tcc.interfacesDoNucleo.ILogin;

@ManagedBean(name ="loginMB")
```

```
@RequestScoped
public class LoginMB {
	private String nomeDeUsuario;
	private String senhaDeUsuario;
	private boolean senhaIncorreta = false;
	private ILogin login = new ILogin();

	public String getSenhaDeUsuario() {
		return senhaDeUsuario;
	}

	public void setSenhaDeUsuario(String senhaDeUsuario) {
		this.senhaDeUsuario = senhaDeUsuario;
	}

	public void setNomeDeUsuario(String nomeDeUsuario) {
		this.nomeDeUsuario = nomeDeUsuario;
	}

	public String getNomeDeUsuario() {
		return nomeDeUsuario;
	}

	public String login(){

		if(login.logarUsuario(nomeDeUsuario, senhaDeUsuario)){
			FacesContext facesContext =
FacesContext.getCurrentInstance();
			HttpServletRequest request = (HttpServletRequest)
facesContext
					.getExternalContext().getRequest();
			HttpSession session = request.getSession();
			session.setAttribute("usuario", nomeDeUsuario);
			return "logado";
		}

		senhaIncorreta = true;
		return null;
	}

	public boolean isSenhaIncorreta() {
		return senhaIncorreta;
	}

	public String logout(){
		FacesContext facesContext =
FacesContext.getCurrentInstance();
		HttpServletRequest request = (HttpServletRequest)
facesContext
				.getExternalContext().getRequest();
		HttpSession session = request.getSession();
		String usuario = (String) session.getAttribute("usuario");
		session.removeAttribute("usuario");

		login.deslogarUsuario(usuario);
		return "sair";
	}
}
```

MeusDocumentosMB:

```
package br.tcc.mbeans;
```

```
import java.util.ArrayList;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import br.tcc.gerDeObjetos.Documento;
import br.tcc.interfacesDoNucleo.IMeusDocumentos;

@ManagedBean(name="meusDocumentosMB")
@RequestScoped
public class MeusDocumentosMB {
      private ArrayList<Documento> lista = new ArrayList<Documento>();
      private IMeusDocumentos documentos = new IMeusDocumentos();

      public MeusDocumentosMB() {
        FacesContext facesContext = FacesContext.getCurrentInstance();
            HttpServletRequest request = (HttpServletRequest)
facesContext
                        .getExternalContext().getRequest();
            HttpSession session = request.getSession();
            String usuario = (String) session.getAttribute("usuario");
            lista = documentos.retornalistaDeDocumentos(usuario);
      }

      public void setLista(ArrayList<Documento> lista) {
            this.lista = lista;
      }

      public ArrayList<Documento> getLista() {
            return lista;
      }

      public String remover(){
            FacesContext facesContext =
FacesContext.getCurrentInstance();
            HttpServletRequest request = (HttpServletRequest)
facesContext
                        .getExternalContext().getRequest();
            HttpSession session = request.getSession();
            String usuario = (String) session.getAttribute("usuario");
            String nome = facesContext.getExternalContext()
                        .getRequestParameterMap().get("removerDoc");
            documentos.remover(nome, usuario);
            return "remover";
      }
}
```

RegistroMB:

```
package br.tcc.mbeans;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;

import br.tcc.interfacesDoNucleo.IRegistro;

@ManagedBean(name = "registroMB")
@RequestScoped
public class RegistroMB {
      private String nome;
```

```
	private String senha;
	private String confirmacaoDeSenha;
	private boolean confirmacaoIncorreta;
	private IRegistro registro = new IRegistro();

	public String registrar(){

		if(senha.equals(confirmacaoDeSenha)){
			if(registro.registrarUsuario(nome, senha)){
				return "ok";
			}
		}

		confirmacaoIncorreta=true;
		return null;
	}

	public String getNome() {
		return nome;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public String getSenha() {
		return senha;
	}

	public void setSenha(String senha) {
		this.senha = senha;
	}

	public String getConfirmacaoDeSenha() {
		return confirmacaoDeSenha;
	}

	public void setConfirmacaoDeSenha(String confirmacaoDeSenha) {
		this.confirmacaoDeSenha = confirmacaoDeSenha;
	}

	public void setConfirmacaoIncorreta(boolean confirmacaoIncorreta)
{
		this.confirmacaoIncorreta = confirmacaoIncorreta;
	}

	public boolean isConfirmacaoIncorreta() {
		return confirmacaoIncorreta;
	}

}
```

UploadMB:

```
package br.tcc.mbeans;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import org.richfaces.event.FileUploadEvent;
```

```
import org.richfaces.model.UploadedFile;

import br.tcc.interfacesDoNucleo.IUpload;


@ManagedBean(name = "uploadMB")
@RequestScoped
public class UploadMB {

    public void listener(FileUploadEvent event) throws Exception {
        UploadedFile item = event.getUploadedFile();
        IUpload upload = new IUpload();
        FacesContext facesContext = FacesContext.getCurrentInstance();
            HttpServletRequest request = (HttpServletRequest)
facesContext
                        .getExternalContext().getRequest();
            HttpSession session = request.getSession();
            String usuario = (String) session.getAttribute("usuario");
        upload.enviar(item.getInputStream(), item.getName(), usuario);
    }

}
```

VisualizacaoMB:

```
package br.tcc.mbeans;

import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStreamReader;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import br.tcc.gerDeObjetos.funcoes.DownloadUpload;
import br.tcc.interfacesDoNucleo.IVisualizacaoMB;

@ManagedBean(name = "visualizacaoMB")
@ViewScoped
public class VisualizacaoMB {

      private String documento = "default";
      private IVisualizacaoMB visualizacao = new IVisualizacaoMB();
      private String nome;
      private String usuario;

      public VisualizacaoMB() {
            FacesContext facesContext =
FacesContext.getCurrentInstance();
            HttpServletRequest request = (HttpServletRequest)
facesContext
                        .getExternalContext().getRequest();
            HttpSession session = request.getSession();
            usuario = (String) session.getAttribute("usuario");
            nome = facesContext.getExternalContext()
                        .getRequestParameterMap().get("abrirDoc");
```

```
            BufferedReader reader = new BufferedReader( new
InputStreamReader(visualizacao.retornaDocumento(nome,
usuario).getDado()));
            String linha = "";
            try {
                  while( ( linha = reader.readLine() ) != null ){
                        documento += linha+"\n";
                  }
            } catch (IOException e) {
            }
      }

      public String getDocumento() {
            return documento;
      }

      public void setDocumento(String documento) {
            this.documento = documento;
      }

      public String salvarDocumento() {
            new DownloadUpload().enviarObjeto(usuario, new
ByteArrayInputStream(documento.getBytes()) , nome);
            return null;
      }
}
```

### Gerenciador De Arquivos

ControleDeAcesso:

```
package br.tcc.controle;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import br.tcc.gerDeUsuarios.Login;

public class ControleDeAcesso implements Filter {
      private static final String SIGNON_PAGE_URI =
"/faces/login.xhtml";

      public void init(FilterConfig filterConfig) throws
ServletException {
      }

      public void doFilter(ServletRequest req, ServletResponse res,
                  FilterChain chain) throws IOException, ServletException
{
            HttpServletResponse response = (HttpServletResponse) res;
            HttpServletRequest request = (HttpServletRequest) req;

            if (!this.authorize((HttpServletRequest) req)) {
```

```
      request.getRequestDispatcher(SIGNON_PAGE_URI).forward(req, res);
            } else {
                  response.setHeader("Cache-Control", "no-store");
                  response.setHeader("Pragma", "no-cache");
                  response.setDateHeader("Expires", 0);
                  chain.doFilter(req, res);
            }
      }

      public void destroy() {
      }

      private boolean authorize(HttpServletRequest req) {
            HttpSession session = req.getSession(false);
            if (session != null) {
                  String nomeDoUsuario = (String)
session.getAttribute("usuario");
                  Login login = new Login();
                  if (login != null && login.usuarioLogado(nomeDoUsuario)
== true) {
                        return true;
                  }
            }
            return false;
      }
}
```

FileSystemNode:

```
package br.tcc.interfacesDoNucleo;

import static com.google.common.base.Predicates.containsPattern;
import static com.google.common.base.Predicates.not;
import static com.google.common.collect.Iterables.filter;
import static com.google.common.collect.Iterables.transform;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Set;

import javax.faces.context.ExternalContext;
import javax.faces.context.FacesContext;

import com.google.common.base.Function;
import com.google.common.collect.Iterables;
import com.google.common.collect.Lists;
public class FileSystemNode {

    private static final Function<String, FileSystemNode> FACTORY = new
Function<String, FileSystemNode>() {
        public FileSystemNode apply(String from) {
            return new FileSystemNode(from.substring(0, from.length() -
1));
        };
    };

    private static final Function<String, String> TO_SHORT_PATH = new
Function<String, String>() {
        public String apply(String from) {
            int idx = from.lastIndexOf('/');
```

```
            if (idx < 0) {
                return from;
            }

            return from.substring(idx + 1);
        };
    };

    private String path;

    private List<FileSystemNode> directories;

    private List<String> files;

    private String shortPath;

    public FileSystemNode(String path) {
        this.path = path;
        int idx = path.lastIndexOf('/');
        if (idx != -1) {
            shortPath = path.substring(idx + 1);
        } else {
            shortPath = path;
        }
    }

    public synchronized List<FileSystemNode> getDirectories() {
        if (directories == null) {
            directories = Lists.newArrayList();

            Iterables.addAll(directories,
transform(filter(getResourcePaths(), containsPattern("/$")), FACTORY));
        }

        return directories;
    }

    public synchronized List<String> getFiles() {
        if (files == null) {
            files = new ArrayList<String>();

            Iterables.addAll(files, transform(filter(getResourcePaths(),
not(containsPattern("/$"))), TO_SHORT_PATH));
        }

        return files;
    }

    private Iterable<String> getResourcePaths() {
        FacesContext facesContext = FacesContext.getCurrentInstance();
        ExternalContext externalContext =
facesContext.getExternalContext();
        Set<String> resourcePaths =
externalContext.getResourcePaths(this.path);

        if (resourcePaths == null) {
            resourcePaths = Collections.emptySet();
        }

        return resourcePaths;
    }

    public String getShortPath() {
```

```
        return shortPath;
    }

}
```

IDiretorio:

```
package br.tcc.interfacesDoNucleo;

import java.util.List;

import br.tcc.gerDeObjetos.funcoes.DownloadUpload;

public class IDiretorio {
      List<FileSystemNode> srcRoots;

      public List<FileSystemNode> retornaListaFileSystem(String usuario)
{
            DownloadUpload dowload = new DownloadUpload();

            String srcPath = dowload.getPath(usuario);

            if (srcRoots == null) {
                  srcRoots = new
FileSystemNode(srcPath).getDirectories();
            }
            return srcRoots;
      }

}
```

ILogin:

```
package br.tcc.interfacesDoNucleo;

import br.tcc.gerDeUsuarios.Login;

public class ILogin {

      public boolean logarUsuario(String nomeDeUsuario, String
senhaDeUsuario) {
            return new Login().conectarUsuario(nomeDeUsuario,
senhaDeUsuario);
      }

      public void deslogarUsuario(String nomeDeUsuario) {
            new Login().desconectarUsuario(nomeDeUsuario);
      }

}
```

IRegistro:

```
package br.tcc.interfacesDoNucleo;

import br.tcc.gerDeUsuarios.Registro;

public class IRegistro {

      public boolean registrarUsuario(String nome, String senha) {
            Registro registro = new Registro();
```

```
            if (registro.cadastrarUsuario(nome, senha) != null){
                  return true;
            }
            return false;
      }
}
```

IUpload:

```
package br.tcc.interfacesDoNucleo;

import java.io.InputStream;

import br.tcc.gerDeObjetos.funcoes.DownloadUpload;


public class IUpload {

      public void enviar(InputStream inputStream, String nome, String
usuario) {
            DownloadUpload upload = new DownloadUpload();
            upload.enviarObjeto(usuario, inputStream, nome);
      }

}
```

DiretorioMB:

```
package br.tcc.mbeans;

import java.util.List;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import org.richfaces.component.UITree;
import org.richfaces.event.TreeSelectionChangeEvent;

import br.tcc.interfacesDoNucleo.FileSystemNode;
import br.tcc.interfacesDoNucleo.IDiretorio;

@ManagedBean(name = "diretorioMB")
@RequestScoped
public class DiretorioMB {

      public synchronized List<FileSystemNode> getListaDeNodos() {
            FacesContext facesContext =
FacesContext.getCurrentInstance();
            HttpServletRequest request = (HttpServletRequest)
facesContext
                        .getExternalContext().getRequest();
            HttpSession session = request.getSession();
            String usuario = (String) session.getAttribute("usuario");

            IDiretorio dir = new IDiretorio();

            return dir.retornaListaFileSystem(usuario);
```

```
	}
}
```

LoginMB:

```
package br.tcc.mbeans;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import br.tcc.interfacesDoNucleo.ILogin;

@ManagedBean(name ="loginMB")
@RequestScoped
public class LoginMB {
	private String nomeDeUsuario;
	private String senhaDeUsuario;
	private boolean senhaIncorreta = false;
	private ILogin login = new ILogin();

	public String getSenhaDeUsuario() {
		return senhaDeUsuario;
	}

	public void setSenhaDeUsuario(String senhaDeUsuario) {
		this.senhaDeUsuario = senhaDeUsuario;
	}

	public void setNomeDeUsuario(String nomeDeUsuario) {
		this.nomeDeUsuario = nomeDeUsuario;
	}

	public String getNomeDeUsuario() {
		return nomeDeUsuario;
	}

	public String login(){

		if(login.logarUsuario(nomeDeUsuario, senhaDeUsuario)){
			FacesContext facesContext =
FacesContext.getCurrentInstance();
			HttpServletRequest request = (HttpServletRequest)
facesContext
					.getExternalContext().getRequest();
			HttpSession session = request.getSession();
			session.setAttribute("usuario", nomeDeUsuario);
			return "logado";
		}

		senhaIncorreta = true;
		return null;
	}

	public boolean isSenhaIncorreta() {
		return senhaIncorreta;
	}

	public String logout(){
		FacesContext facesContext =
FacesContext.getCurrentInstance();
```

```
            HttpServletRequest request = (HttpServletRequest)
facesContext
                        .getExternalContext().getRequest();
            HttpSession session = request.getSession();
            String usuario = (String) session.getAttribute("usuario");
            session.removeAttribute("usuario");

            login.deslogarUsuario(usuario);
            return "sair";
      }
}
```

RegistroMB:

```
package br.tcc.mbeans;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;

import br.tcc.interfacesDoNucleo.IRegistro;

@ManagedBean(name = "registroMB")
@RequestScoped
public class RegistroMB {
      private String nome;
      private String senha;
      private String confirmacaoDeSenha;
      private boolean confirmacaoIncorreta;
      private IRegistro registro = new IRegistro();

      public String registrar(){

            if(senha.equals(confirmacaoDeSenha)){
                  if(registro.registrarUsuario(nome, senha)){
                        return "ok";
                  }
            }

            confirmacaoIncorreta=true;
            return null;
      }

      public String getNome() {
            return nome;
      }

      public void setNome(String nome) {
            this.nome = nome;
      }

      public String getSenha() {
            return senha;
      }

      public void setSenha(String senha) {
            this.senha = senha;
      }

      public String getConfirmacaoDeSenha() {
            return confirmacaoDeSenha;
      }

      public void setConfirmacaoDeSenha(String confirmacaoDeSenha) {
```

```
            this.confirmacaoDeSenha = confirmacaoDeSenha;
      }

      public void setConfirmacaoIncorreta(boolean confirmacaoIncorreta)
{
            this.confirmacaoIncorreta = confirmacaoIncorreta;
      }

      public boolean isConfirmacaoIncorreta() {
            return confirmacaoIncorreta;
      }

}
```

UploadMB:

```
package br.tcc.mbeans;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import org.richfaces.event.FileUploadEvent;
import org.richfaces.model.UploadedFile;

import br.tcc.interfacesDoNucleo.IUpload;


@ManagedBean(name = "uploadMB")
@ViewScoped
public class UploadMB {
      private String caminho = "";

    public void listener(FileUploadEvent event) throws Exception {
        UploadedFile item = event.getUploadedFile();
        IUpload upload = new IUpload();
        FacesContext facesContext = FacesContext.getCurrentInstance();
            HttpServletRequest request = (HttpServletRequest)
facesContext
                        .getExternalContext().getRequest();
            HttpSession session = request.getSession();
            String usuario = (String) session.getAttribute("usuario");
        upload.enviar(item.getInputStream(), item.getName(),
(usuario+"/MeusArquivos"+getCaminho()));
    }

      public void setCaminho(String caminho) {
            this.caminho = caminho;
      }

      public String getCaminho() {
            return caminho;
      }

}
```