JULIANA PINHEIRO CAMPOS

EXTRAÇÃO DE CANDIDATOS A ASPECTOS A PARTIR DE DESCRIÇÕES DE FLUXO DE CASOS DE USO

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

VIÇOSA
MINAS GERAIS - BRASIL
2009

JULIANA PINHEIRO CAMPOS

EXTRAÇÃO DE CANDIDATOS A ASPECTOS A PARTIR DE DESCRIÇÕES DE FLUXO DE CASOS DE USO

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

APROVADA: 28 de agosto de 2009.

Antônio Maria Pereira de Resende
(Co-Orientador)

Alcione de Paiva Oliveira

Marco Túlio de Oliveira

Marcus Vinícius Alivim Andrade

José Luis Braga
(Orientador)

Dedico essa dissertação aos meus pais
Francisco e Selma

E ao meu amado namorado
Victor.

AGRADECIMENTOS

Agradeço a Deus, que guia todos os meus passos. Agradeço por cuidar da minha vida, pela coragem e força para vencer desafios e por confortar meu coração nos momentos difíceis.

Agradeço aos meus pais, Francisco e Selma, pelo amor e por todas as lições de vida ensinadas. Obrigada por me ensinar que a educação é o bem mais precioso, por confiarem em mim e me incentivarem a prosseguir sempre.

Às minhas irmãs Anna Luiza, Anna Carina e Tatiana, pelo carinho e por me apoiarem sempre. Carina e Tati, obrigada pela companhia em Viçosa, pelos ótimos momentos de convivência. Ao meu sobrinho João Pedro, cunhados e toda a minha família pelo grande incentivo.

Em especial, agradeço ao meu namorado Victor que sempre esteve ao meu lado. Obrigada meu amor, por tudo que faz por mim. O seu amor, dedicação, paciência e companheirismo foram fundamentais. Agradeço também ao José Antônio e à Aulenir, por me receberem como filha em sua casa.

Agradeço ao meu orientador, Professor José Luis Braga, pelos ensinamentos, atenção, amizade e paciência durante a graduação e o mestrado. Agradeço também aos meus co-orientadores, Professores Antônio Maria Pereira de Resende e Vladimir Oliveira Di Iorio, pelos conselhos e questionamentos sobre o trabalho.

A Universidade Federal de Viçosa, especialmente aos professores e funcionários do Departamento de Informática, pela formação acadêmica recebida e por estarem sempre dispostos a ajudar.

A Capes, pelo apoio financeiro.

Aos colegas do mestrado e da graduação, pelo apoio. Um agradecimento especial à Lissandra e ao Alexandre, amigos e companheiros de estudo.

Agradeço ao Professor Frederico José Vieira Passos e todos os funcionários da CEAD, pelas oportunidades e pelo apoio. Em especial, à Professora Silvane Guimarães Silva Gomes, pela grande amizade.

Aos meus amigos e a todos que, de algum modo, contribuíram direta ou indiretamente para conclusão dessa etapa da minha vida. Cheguei até aqui porque tive a oportunidade de conhecer, conviver e aprender com pessoas muito especiais.

BIOGRAFIA

JULIANA PINHEIRO CAMPOS, filha de Francisco Pereira Campos e Selma Regina Pinheiro Campos, nascida em 26 de abril de 1984 na cidade de Ponte Nova, Minas Gerais.

No ano de 2003, após concluir o ensino médio na cidade de Ponte Nova, iniciou o curso de Ciência da Computação na Universidade Federal de Viçosa - UFV, concluído no ano de 2007.

Também em 2007, ingressou no Curso de Mestrado do Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Viçosa, defendendo sua dissertação em agosto de 2009.

Atualmente, é professora substituta do Departamento de Engenharia de Controle e Automação do Centro Federal de Educação Tecnológica de Minas Gerais – CEFET, atuando no campus III em Leopoldina, desde abril de 2009.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

RESUMO

CAMPOS, Juliana Pinheiro, M.Sc., Universidade Federal de Viçosa, agosto de 2009. **Extração de candidatos a aspectos a partir de descrições de fluxo de casos de uso.** Orientador: José Luis Braga. Co-Orientadores: Antônio Maria Pereira de Resende e Vladimir Oliveira Di Iorio.

A identificação de candidatos a aspectos nas fases iniciais do processo de desenvolvimento de *software* permite melhorar a modularização dos requisitos, detectar conflitos mais cedo e manter a separação de interesses nas fases posteriores do processo. Porém, a identificação de aspectos em fases iniciais não é trivial já que os requisitos transversais se encontram espalhados nos diversos documentos elaborados nessas fases. Essa tarefa pode ser muito cara e consumir muito tempo. Dessa forma, o objetivo deste trabalho é obter uma técnica que auxilie a identificação de candidatos a aspectos na fase de requisitos, por meio de inspeção em descrições de fluxo de casos de uso. Para alcançar esse objetivo, foi projetado um *template* para descrição de casos de uso de forma estruturada, com seções pré-definidas, nas quais podem ser identificados RNFs e RFs candidatos a aspectos. Também foi projetado um *checklist* para verificação das descrições de casos de uso. Foram realizados dois estudos de caso: o primeiro com o objetivo de avaliar a facilidade de compreensão e utilização do *template* e *checklist;* o segundo com o objetivo de avaliar a utilização da técnica proposta. Os estudos de caso realizados permitiram identificar os principais RNFs incluídos nas descrições realizadas pelos participantes e em quais seções da descrição eles aparecem. A análise dos resultados obtidos com os estudos de caso são evidências de que a técnica proposta facilita a inspeção em busca de candidatos a aspectos.

ABSTRACT

CAMPOS, Juliana Pinheiro, M.Sc., Universidade Federal de Viçosa, August, 2009. **Identification of aspect candidates in use cases descriptions**. Adviser: José Luis Braga. Co-Advisers: Antônio Maria Pereira de Resende and Vladimir Oliveira Di Iorio.

The identification of aspect candidates in the early stages of software development process allows improving the modularization of the requirements, to detect conflicts early and to maintain the separation of concerns in the later stages of the process. However, the identification of aspects in initial phases is a non-trivial task because the crosscutting concerns can often be scattered across the several requirements documents. This task can be costly and too time-consuming. The goal of this research is to presents a technique to assist in the identification of aspect candidates in the requirements, through inspection. To achieve this objective, a template for use cases description structured was designed, with pre-defined sections, in which non-functional requirements (NFRs) and functional requirements (FRs) aspect candidates can be more readily identified. A checklist for check the use cases description was designed too. Two case studies were carried out: the first to evaluate the ease of understanding and use of template and checklist; the second to evaluate the use of proposed technique. These case studies allow identifying the mains NFRs included in the descriptions made by the participants and in which sections of description they appear. The analysis of results obtained from the case studies let us conclude that the proposed technique facilitates the identification of aspect candidates.

1 INTRODUÇÃO

A utilização de *software* trouxe para a vida moderna novas oportunidades de educação, entretenimento e desenvolvimento de negócios. São muitas as facilidades oferecidas pelos sistemas computacionais incorporados às diversas atividades, tanto pessoais como profissionais. Hoje em dia, o *software* está presente, explicitamente ou mesmo sem se fazer notar, em todos os aspectos de nossa vida (PFLEEGER, 2004). Os computadores se tornaram comuns no dia-a-dia das pessoas que buscam solucionar problemas com requisitos de complexidade cada vez maiores e difíceis de compreender. Para garantir que o *software* obtenha êxito, ou seja, atenda de forma eficiente às necessidades e expectativas dos usuários facilitando o seu dia-a-dia, é necessário utilizar um método disciplinado no seu desenvolvimento.

A disciplina que estuda os processos de desenvolvimento de *software* e o gerenciamento destes processos para que os sistemas sejam desenvolvidos com qualidade e atendam as necessidades de seus usuários é a Engenharia de *Software*. Segundo SOMMERVILLE (2004), a Engenharia de *Software* é uma disciplina de engenharia que se ocupa de todos os aspectos da produção de *software*, desde os estágios iniciais de especificação do sistema até a manutenção depois que ele entrou em operação. Os engenheiros de *software* adotam práticas sistemáticas no desenvolvimento buscando alto índice de acerto e satisfação dos usuários.

Para PRESSMAN (2005), a Engenharia de *Software* abrange 3 elementos fundamentais que fornecem a base para construção de *software* de alta qualidade: métodos, ferramentas e processos. Os métodos proporcionam os detalhes de "como fazer" para construir o *software*, incluindo atividades de modelagem e outras técnicas descritivas. As ferramentas fornecem suporte automatizado ou semi-automatizado aos métodos e processos de desenvolvimento. E o processo define a base para o controle de gestão dos projetos de *software* e estabelece o contexto no qual os métodos são aplicados, as metas são estabelecidas, a qualidade é garantida e as alterações são adequadamente gerenciadas.

Um processo de desenvolvimento de *software* é um conjunto de atividades que levam à produção de um *software*. As atividades envolvidas em um processo devem ser bem definidas e documentadas. Elas permitem especificar, projetar, implementar, validar e manter um sistema. Uma descrição simplificada de todas as atividades envolvidas no processo de desenvolvimento, uma abstração do processo real, constitui um modelo de processo de *software*.

No processo de desenvolvimento de um *software*, o primeiro passo é analisar o problema e identificar as características necessárias ao sistema para atingir seus objetivos. Nesta fase, chamada de *identificação(extração) de requisitos* e incluída em todos os modelos de processo, os engenheiros de requisitos se utilizam de entrevistas ou outras técnicas para identificar as características necessárias ao sistema. Um documento de requisitos de *software* é elaborado com as características identificadas. Portanto, os clientes e usuários devem concordar com as características descritas neste documento. Uma etapa de validação assegura que estes requisitos estão corretos.

Várias técnicas são utilizadas para desenvolver a compreensão dos requisitos de um sistema. Os casos de uso da UML (*Unified Modeling Language*) (UML, 2009) são uma das técnicas mais utilizadas. A UML é um padrão aberto, controlado pelo OMG (*Object Management Group*) (OMG, 2009) e é utilizada pelos desenvolvedores para ajudar a transmitir algumas características de um sistema. Os casos de uso descrevem possíveis interações entre os usuários e os sistemas que serão representadas nos requisitos do *software*, recorrendo a diagramas de atividades e descrições textuais. A lógica dos casos de uso é descrita sob o ponto de vista do ator externo, deixando claros seus requisitos funcionais relativos ao sistema a ser desenvolvido.

Recentemente, pesquisas apontam a importância e o impacto de incluir entre os requisitos de um problema, os relacionados com os `aspectos` (ARAÚJO; COUTINHO, 2003; RASHID *et al*., 2003; ROSENHAINER, 2004; ARAÚJO *et al*., 2005; SAMPAIO *et al*., 2005). Um `aspecto` é uma nova unidade de modularização utilizada na Programação Orientada a Aspectos (POA). A POA propõe a utilização de `aspectos` para modularização dos interesses transversais (*crosscutting concerns*) do sistema (KICZALES *et al*., 1997). Um interesse (*concern*) é uma característica ou um requisito necessário ao sistema que possui responsabilidade bem-definida. LADDAD (2003) classifica os interesses do sistema

em interesses de negócio (*core concerns*) e interesses transversais (*crosscutting concerns*). Os interesses de negócio capturam a funcionalidade principal de um módulo. Já os interesses transversais capturam os requisitos do sistema que atravessam vários módulos, dificultando a modularização. Os `aspectos` possibilitam a separação de interesses que mais tarde serão compostos em um sistema consistente. Essa modularização contribui para construção de sistemas de *software*, melhorando a forma como os sistemas complexos são compreendidos e auxiliando na manutenção dos vários interesses e na adição de novas características ao sistema.

A POA é uma nova maneira revolucionária de pensar sobre Engenharia de *Software* (JACOBSON; NG, 2004). Com a POA, surgiu a necessidade de tratar os `aspectos` em todas as fases do desenvolvimento de *software*. Na próxima seção será explicado porque grande atenção tem sido dada a identificação de aspectos na fase de requisitos.

1.1 O problema e sua importância

No desenvolvimento de *software*, é importante separar os diferentes interesses em módulos e desenvolver cada um desses módulos separadamente. Um módulo é uma unidade compilável do sistema responsável por uma tarefa bem definida (DIJKSTRA, 1976). A integração de vários módulos compõe o sistema.

A divisão de um problema complexo em problemas menores (módulos) que podem ser tratados como uma unidade isolada é chamada de separação de interesses (*separation of concerns*). O princípio de separação de interesses, bastante difundido na Engenharia de *Software*, é apresentado por DIJKSTRA (1976). A separação de interesses é importante porque facilita o entendimento do problema, permitindo ao desenvolvedor concentrar esforços no desenvolvimento de uma funcionalidade do sistema por vez. Além disso, facilita a manutenção do *software*, a adição de novas funcionalidades e a reutilização de módulos.

A Programação Orientada a Objetos (POO), paradigma de desenvolvimento de *software* muito utilizado atualmente, busca obter a modularização encapsulando um conjunto de requisitos ou funcionalidades em `classes`. Cada `classe` implementada representa um conjunto de `objetos`. Ela contém as características (atributos) e comportamentos (métodos) comuns aos `objetos` da `classe`. Um

`objeto` é qualquer coisa concreta ou abstrata que exista no mundo real. Um *sofware* desenvolvido utilizando POO pode ser visto como uma associação de `objetos` que colaboram entre si (LADDAD, 2003).

A POO permite a modularização adequada dos interesses de negócio do sistema, mas não é capaz de resolver adequadamente a modularização de interesses transversais que tendem a se espalhar por todo o código do sistema. Segundo JACOBSON & NG (2004), a incapacidade de manter esses interesses separados durante o projeto e implementação torna um sistema difícil para entender e manter.

A POA surgiu com o objetivo de fornecer uma melhor separação de interesses transversais. É uma alternativa de evolução, prometendo a capacidade de reduzir a complexidade de desenvolvimento de sistemas e manter os benefícios obtidos com a POO (RESENDE *et al.*, 2005). Quando não é possível obter a modularização completa do sistema utilizando a POO, os aspectos são utilizados para obter essa modularização.

A POA ajuda a modularizar os interesses transversais durante a implementação, mas é necessário modularizar interesses transversais nas fases iniciais do processo de desenvolvimento de *software*, durante a identificação de requisitos, para que essa separação seja mantida nas fases posteriores do processo. Além disso, a separação dos interesses transversais nessas fases iniciais evita custos relacionados à refatoração de código (*refactoring*) em fases posteriores do processo e permite detectar conflitos mais cedo. Porém, a identificação de interesses transversais nos requisitos que podem ser capturados como aspectos na implementação (candidatos a aspectos), ou seja, requisitos implementáveis por orientação a aspectos, ainda é um problema.

A identificação de candidatos a aspectos na fase de requisitos é um grande desafio pelo fato dos interesses transversais se encontrarem espalhados nos diversos documentos, elaborados nessa fase, sujeitos a erros, informações incompletas, inconsistentes, imprecisas e ambíguas. Como em qualquer técnica de Engenharia de Requisitos (*Requirements Engineering* - RE), alguém tem que buscar por aspectos em um grande volume de documentos (CHITCHYAN *et al.*, 2006a). Buscar por candidatos a aspectos nesses documentos seria muito caro e consumiria muito tempo. Reconhecendo essa dificuldade, é importante analisar documentos elaborados nessa fase, identificar e apresentar novas formas de auxiliar a busca por candidatos a aspectos nesses documentos.

1.2 Objetivos

O objetivo geral deste trabalho é a obtenção de uma técnica que auxilie a identificação de candidatos a aspectos na fase de requisitos, utilizando análise de descrições de fluxo de casos de uso.

Os objetivos específicos foram:

- Estudar os casos de uso da UML e sua especificação utilizando descrições textuais, conhecendo vários estilos de descrição de casos de uso e boas práticas utilizadas nessas descrições;
- Aumentar a compreensão acerca da POA, do processo de desenvolvimento de *software* orientado a aspectos e dos recursos utilizados em linguagens orientadas a aspectos;
- Investigar o que é um aspecto no nível de requisitos e como fazer a inspeção nos requisitos em busca de candidatos a aspectos;
- Investigar metodologias e ferramentas que permitam a identificação de candidatos a aspectos nas fases iniciais do processo de desenvolvimento de *software*;
- Descrever padrões identificados na análise de descrições de casos de uso em busca de candidatos a aspectos que possam auxiliar a inspeção posterior;
- Apresentar situações que comprovem que a utilização da técnica obtida auxilia a inspeção em busca de candidatos a aspectos nas descrições de casos de uso;
- Especificar e construir um *plugin* para um ambiente integrado de desenvolvimento (*Integrated Development Environment* - IDE) que suporte arquitetura de *plugins*, que auxilie a identificação de candidatos a aspectos nas descrições de fluxo de casos de uso.

1.3 O fluxo de trabalho utilizado no desenvolvimento dessa pesquisa

Buscando alcançar os objetivos propostos neste trabalho, foram realizadas as seguintes atividades:

1. **Revisão bibliográfica:** No início do trabalho, foi realizada uma extensa revisão bibliográfica para aprofundar os conhecimentos acerca do contexto em que o

trabalho está inserido. Foram exploradas as áreas: extração de requisitos, particularmente os casos de uso da UML e sua especificação utilizando descrições textuais; POA e *early aspects*. Também foram investigados os trabalhos relacionados. Inicialmente foi realizada a leitura do livro "*Aspect-Oriented Software Development with Use Cases*" (JACOBSON; NG, 2004) que apresenta uma solução para manter a separação de interesses transversais usando aspectos na implementação. Vários artigos científicos que abordam assuntos relacionados ao presente trabalho também foram lidos. Esses trabalhos serão apresentados ao longo do texto.

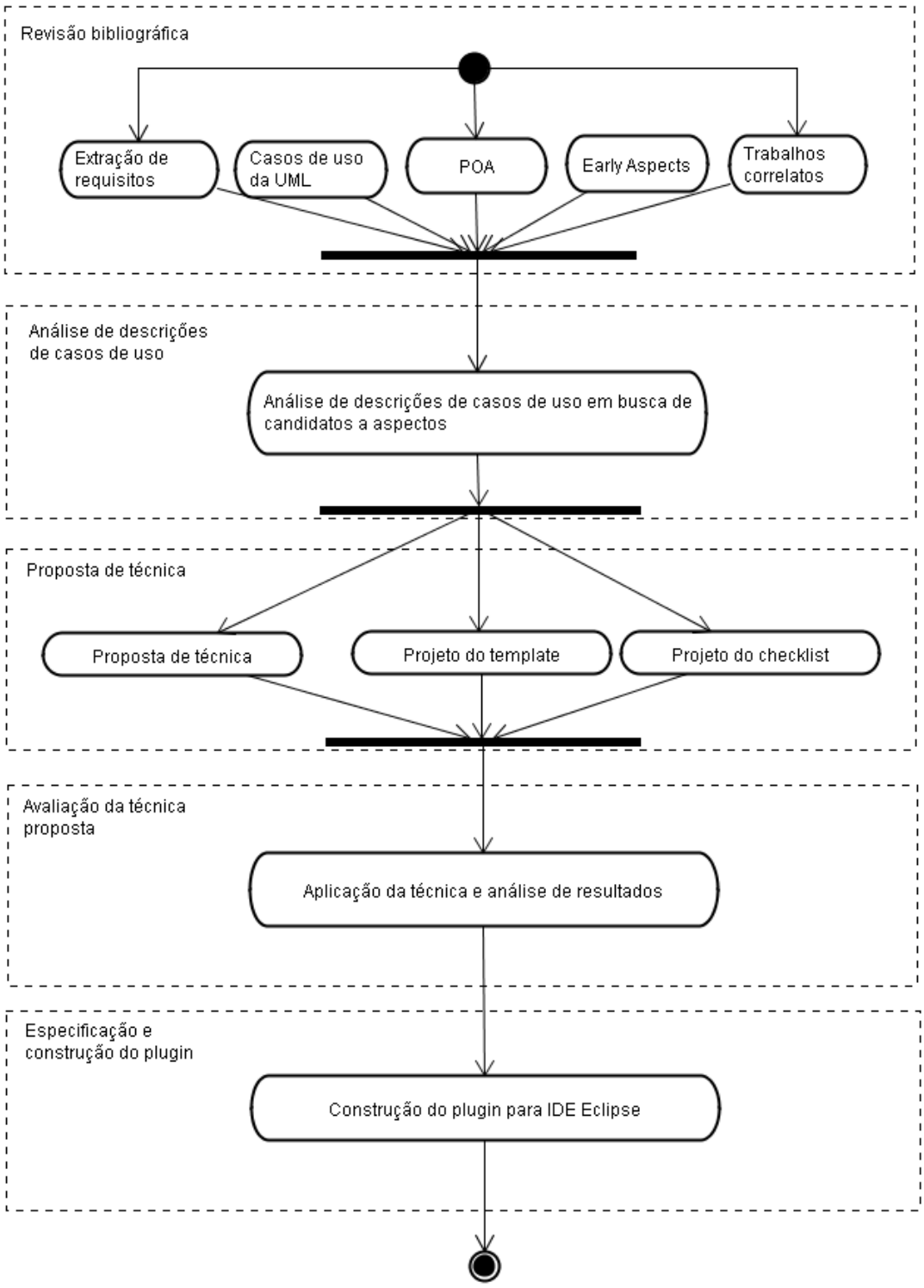
2. **Análise de descrições de casos de uso:** Foram analisadas diversas descrições de casos de uso presentes na literatura em busca de candidatos a aspectos. Durante essa análise, foram observados alguns padrões relacionados à seção da descrição em que candidatos a aspectos foram encontrados e tipo de requisitos candidatos a aspectos identificados em cada seção.

3. **Proposta de técnica:** Foi proposta uma técnica para auxiliar a identificação de candidatos a aspectos por meio de inspeção na fase de requisitos. Para utilização da técnica, foram projetados um *template* para descrição de casos de uso e um *checklist* para verificação da descrição. O *template* desenvolvido contém as principais seções nas quais foram identificados candidatos a aspectos durante a análise de descrições de casos de uso encontradas na literatura. O *checklist* foi desenvolvido para melhorar a descrição realizada utilizando o *template*.

4. **Avaliação da técnica proposta:** Foram realizados dois estudos de caso. Inicialmente o *template* e *checklist* foram utilizados com o objetivo de avaliar a facilidade de compreensão e utilização desses documentos e identificar candidatos a aspectos incluídos nas descrições realizadas pelos participantes. Após essa avaliação, foi realizada a aplicação da técnica com o objetivo de mostrar que a sua utilização facilita a inspeção em busca de candidatos a aspectos.

5. **Especificação e construção do *plugin*:** Como subproduto deste trabalho, foi especificado um *plugin* para o ambiente integrado de desenvolvimento (*Integrated Development Environment - IDE*) Eclipse que implementa a técnica proposta para auxiliar a identificação de candidatos a aspectos utilizando descrições de casos de uso. A implementação desse *plugin* foi iniciada. Ele permite ao usuário descrever o fluxo de casos de uso do sistema e deve auxiliá-lo com a estruturação do texto de forma a facilitar a análise posterior. Técnicas de análise de texto serão utilizadas na

análise dessas descrições para sugerir candidatos a aspectos.



Fonte: Elaborada pelo autor.

Figura 1.1. Fluxograma das etapas de desenvolvimento do trabalho

1.4 Estrutura da dissertação

No Capítulo 2 é apresentado o referencial teórico, resultado da revisão bibliográfica realizada. Nesse capítulo são discutidos os principais conceitos relacionados ao trabalho, incluindo os conceitos que buscamos nas descrições de casos de uso para identificar candidatos a aspectos. Também são apresentados os trabalhos correlatos que tratam a identificação de aspectos em documentos de requisitos e que relacionam casos de uso e aspectos.

No Capítulo 3 são descritos os principais padrões identificados durante a análise de descrições de casos de uso e também a técnica proposta para auxiliar a inspeção em busca de candidatos a aspectos em descrições de casos de uso. Também são descritos o *template* e *checklist* desenvolvidos.

No Capítulo 4 são apresentados os estudos de caso realizados durante o trabalho. São apresentados os objetivos das avaliações, a metodologia utilizada, os questionários respondidos pelos participantes das avaliações, as análises desses questionários e os resultados desses experimentos.

No Capítulo 5 são apresentadas as conclusões, incluindo as principais contribuições disponibilizadas por esse trabalho. Também são apresentadas sugestões de trabalhos futuros que poderão contribuir para melhoria do trabalho desenvolvido.

Ao final do documento é apresentado um apêndice contendo: o *template* e o *checklist* que foram descritos no Capítulo 3; e os questionários respondidos pelos participantes descritos no Capítulo 4.

2 REFERENCIAL TEÓRICO

Este capítulo tem por objetivo apresentar os principais conceitos relacionados a este trabalho. Inicialmente, será apresentado o conceito de *Extração de Requisitos* para compreender o contexto onde os casos de uso da UML são utilizados e também a dificuldade inerente a esse processo. Em seguida, serão apresentados os temas centrais deste trabalho: *Casos de Uso* e *POA*. O estudo de *Casos de Uso* foi importante para compreender sua descrição textual, artefato utilizado na busca de candidatos a aspectos. Com o estudo de POA, foi possível familiarizar-se com seus conceitos principais, compreender o processo de *Desenvolvimento de Software Orientado a Aspectos* e recursos de *Linguagens Orientadas a Aspectos* utilizados em exemplos apresentados posteriormente. Na seqüência, a pesquisa sobre *Early Aspects* busca apresentar em que consiste o trabalho e as motivações que impulsionaram seu desenvolvimento, além de apresentar a técnica utilizada para auxiliar a identificação de aspectos: a identificação através de inspeção. Por último, foram descritos os *trabalhos correlatos* que contribuíram para o desenvolvimento deste trabalho.

2.1 Extração de requisitos

Os requisitos de *software* representam todas as características e funcionalidades que um *software* deve disponibilizar aos seus usuários. São as características necessárias para que o sistema realize as tarefas solicitadas pelos usuários. Segundo (SOMMERVILLE, 2004), um requisito pode ser visto como uma declaração abstrata, de alto nível, de uma função que o sistema deve fornecer ou de uma restrição do sistema, ou pode ser visto como uma função detalhada, matematicamente formal, de uma função do sistema.

Os sistemas são projetados e implementados a partir dos requisitos. Esses podem ser classificados como requisitos funcionais (RFs) ou requisitos não-funcionais (RNFs). Os RFs são os que constituem o objetivo final do sistema e os RNFs compreendem elementos específicos de projeto, muitas vezes sem relação direta com o problema em questão (TIRELO *et al.*, 2004).

A Engenharia de Requisitos é uma sub-área da Engenharia de *Software* responsável por desenvolver uma especificação do *software* que servirá como base para a construção do sistema. Ela engloba: o estudo da viabilidade, ou seja, verifica a possibilidade de se desenvolver o sistema; a elicitação e análise de requisitos; a especificação de requisitos, a validação dos requisitos e o gerenciamento dos requisitos (SOMMERVILLE, 2004). O objetivo é identificar todos os requisitos do *software*, analisar, documentar e manter esses requisitos de forma sistemática. A Engenharia de Requisitos é muito importante, pois permite a interação entre as pessoas que solicitaram o *software* e as pessoas que irão desenvolvê-lo. É necessário desenvolver a compreensão dos requisitos o suficiente para lhe permitir prosseguir com o projeto e construção em um nível de risco aceitável (WIEGERS, 2003).

Uma etapa essencial da Engenharia de Requisitos para que o desenvolvimento seja bem sucedido é a extração de requisitos. Nessa fase, os engenheiros de requisitos trabalham com os clientes e usuários finais do sistema a fim de explorar o domínio do problema, descobrir quais as funções o *software* deve fornecer e quais as restrições do sistema. O engenheiro de requisitos atua como um investigador, induzindo o usuário a falar sobre suas expectativas com relação ao produto.

Para auxiliar a extração de requisitos, podem ser utilizadas técnicas para obtenção de informações como: entrevistas, aplicação de questionários, análise de documentação existente, acompanhamento do trabalho dos clientes e usuários, desenvolvimento de protótipos de telas do sistema e esquemas que permitam a compreensão dos clientes, usuários e de todos os profissionais envolvidos no processo. A escolha da técnica de elicitação de requisitos depende do tempo e recursos disponíveis para o engenheiro de requisitos e do tipo de informação que necessita ser extraída (NUSEIBEH; EASTERBROOK, 2000).

Identificar os requisitos do sistema é um dos maiores desafios encontrados na produção de *software*. A elicitação de requisitos é talvez a tarefa de desenvolvimento de *software* mais difícil, mais crítica e propensa a erros (WIEGERS, 2003). O levantamento de requisitos é um processo difícil porque os clientes muitas vezes não sabem o que querem ou o que é necessário e expressam suas necessidades sem levar em consideração que os engenheiros de requisitos não conhecem o seu contexto de trabalho. De acordo com (BROOKS, 1995), é impossível para os clientes especificar completamente, de forma precisa e corretamente os requisitos de um produto de

software moderno antes de ter uma primeira versão do produto construída. Portanto, não é fácil para o engenheiro de requisitos compreender a natureza dos problemas.

O trabalho de (CHRISTEL; KANG, 1992) identifica três categorias de problemas que ajudam a compreender porque a extração de requisitos é difícil:

- **problemas de escopo:** confusões podem ser geradas porque os clientes e usuários expressam os requisitos em detalhes desnecessários ou não conseguem expressar o mínimo de detalhes.
- **problemas de compreensão:** podem surgir dentro dos grupos ou entre os grupos de clientes, usuários e desenvolvedores. Os clientes e usuários não sabem exatamente o que querem ou o que é necessário, têm pouco conhecimento das capacidades e limitações de sistemas, ou especificam requisitos ambíguos.
- **problemas de volatilidade:** os requisitos mudam ao longo do tempo.

A dificuldade inerente ao processo de extração de requisitos faz com que muitos erros sejam cometidos durante esse processo, prejudicando a especificação de requisitos. Se essas falhas não são detectadas durante a Engenharia de Requisitos, elas produzem problemas no projeto do sistema e na implementação, que podem levar a alterações posteriores no sistema e custos relacionados ao retrabalho.

O retrabalho é a principal conseqüência de problemas nos requisitos (WIEGERS, 2003). Durante o processo de desenvolvimento, quanto mais tarde um problema resultante de erro na fase de extração de requisitos for descoberto, maior será o custo para remover esse erro. Isso acontece porque uma mudança nos requisitos implica em mudanças no projeto e na implementação do sistema. De acordo com (PRESSMAN, 2005), o custo de correção de erros na fase de projeto é cerca de três a seis vezes mais alto do que na fase de definição de requisitos. BOEHM; BASILI (2001), dizem que é 100 vezes mais caro reparar erros na produção do que durante as fases anteriores do desenvolvimento.

As principais causas de alterações em projetos de *software* devido à ineficiência na fase de extração de requisitos são: requisitos incompletos ou inconsistentes, requisitos que foram especificados erradamente, requisitos que foram mal interpretados pelo engenheiro de requisitos, requisitos que o usuário não conseguiu expressar corretamente, requisitos que deveriam ter sido especificados e passaram despercebidos e falta de controle na alteração de requisitos. GLASS

(2003), afirma que a instabilidade de requisitos é uma das causas mais comuns de problemas em projetos.

O levantamento e a gestão dos requisitos representam um papel chave, determinando o sucesso de projetos de *software* e a qualidade dos sistemas entregues ao cliente (NUSEIBEH; EASTERBROOK, 2000). Uma extração de requisitos bem organizada e executada permite o desenvolvimento de uma boa especificação de requisitos. O guia IEEE para Especificação de Requisitos de *Software* (IEEE, 1984) define uma boa especificação de requisitos como sendo: não ambígua, completa, verificável, consistente, modificável, rastreável e usável durante as fases de operações e manutenção. Uma boa gestão dos requisitos permite descobrir erros com antecedência e minimizar os impactos de modificações posteriores. O processo "Gestão de Requisitos" faz parte dos modelos de maturidade do processo de desenvolvimento de *software* como CMMI (*Capability Maturity Model Integration*) (SEI, 2009) e mpsBR (Melhoria do Processo do *Software* Brasileiro) (SOFTEX, 2007), o que mostra sua importância na Engenharia de Requisitos.

2.2 Casos de uso

Os casos de uso da UML são uma técnica para auxiliar na obtenção e compreensão dos requisitos, criada por Ivar Jacobson (JACOBSON *et al.*, 1993) na década de 1970. A UML é uma família de notações gráficas, que ajuda na descrição e no projeto de sistemas de *software*, particularmente daqueles construídos utilizando o estilo orientado a objetos (FOWLER, 2005). Um caso de uso descreve uma interação entre o usuário e o sistema, definindo uma funcionalidade pedida pelos clientes e usuários que deve ser fornecida pelo sistema final. Como sua lógica é descrita sob o ponto de vista do ator externo, o caso de uso deve ser simples e fácil para os clientes e usuários compreenderem. Hoje, os casos de uso têm sido utilizados por grande parte da comunidade de adeptos da orientação a objetos como suplemento dos meios mais formais de modelagem de objetos, durante a análise do sistema (PFLEEGER, 2004).

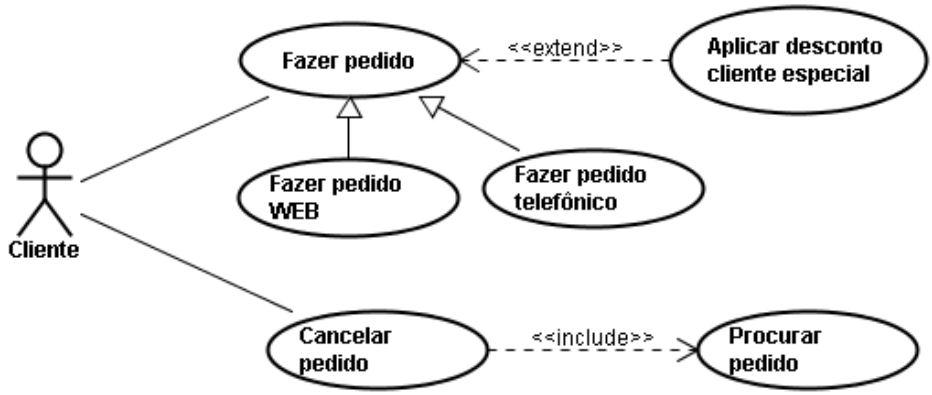
O conjunto de casos de uso de um sistema representa todas as possíveis interações, ou seja, todos os requisitos do sistema (SOMMERVILLE, 2004). A UML

descreve um diagrama de casos de uso que representa os atores do sistema, os casos de uso e os relacionamentos entre eles (ERIKSSON *et al.*, 2004). Os atores representam os papéis desempenhados pelos diversos usuários que poderão interagir com o sistema. Pode ser um usuário ou outro sistema e é quem inicia o caso de uso. Os relacionamentos possíveis entre os casos de uso são:

- **inclusão (*include*):** indica que um caso de uso (base) contém o comportamento definido em outro caso de uso. É utilizado quando existem comportamentos comuns a vários casos de uso. Esses comportamentos são descritos em um único caso de uso que é incluído em todos os outros que possuem o mesmo comportamento. O caso de uso incluído é obrigatoriamente inserido no caso de uso base sempre que este é executado. Um ponto de inclusão (*inclusion point*) indica o local no caso de uso base ao qual o caso de uso incluído está associado.
- **extensão *(extend)*:** especifica que o comportamento de um caso de uso incorpora implicitamente o comportamento de outro caso de uso em um ou mais locais específicos chamados de pontos de extensão (*extension points*). Esses pontos são definidos no caso de uso base e a extensão será adicionada a ele sob uma condição. É utilizado para adicionar condicionalmente comportamentos para um caso de uso existente. Esse relacionamento modela um comportamento opcional do sistema, ou seja, a execução do caso de uso estendido não é obrigatória ao executar o caso de uso base.
- **generalização (*generalization)*:** permite que um caso de uso herde as características de outro, seguindo a mesma semântica da herança entre classes. Representa um caso de uso generalizado (pai) que descreve as características compartilhadas por todos os casos de uso especializados (filhos).

Um exemplo de diagrama de casos de uso UML para um sistema de vendas é apresentado na figura 2.1. Existe somente um ator, Cliente, representado por um ícone padrão, 웃, no diagrama. Os casos de uso são representados por elipses. O relacionamento *include* foi utilizado para incluir no caso de uso `Cancelar pedido` o subfluxo descrito no caso de uso `Procurar pedido`. Sempre que um cliente executar o caso de uso `Cancelar pedido`, ele vai procurar o pedido no sistema. A funcionalidade `Procurar pedido` pode ser incluída em outros casos de uso. Basta acrescentar pontos de inclusão nos casos de uso que possuem esse

comportamento. O caso de uso `Fazer pedido` estende o seu comportamento com o comportamento do caso de uso `Aplicar desconto cliente especial`. Isso ocorre quando o sistema calcula os valores totais para cada produto fornecido no pedido, se o cliente que faz o pedido for um cliente especial. O caso de uso `Fazer pedido` serve como um *template* para todos os casos de uso que tratam o pedido de um cliente. Ele generaliza os casos de uso `Fazer pedido WEB` e `Fazer pedido telefônico`.



Fonte: Adaptado de (SCHNEIDER; WINTERS, 2001)

Figura 2.1. Diagrama de casos de uso

Os diagramas de casos de uso da UML focam na coleta rápida de informações do sistema e têm como propósitos (ERIKSSON *et al.*, 2004):

- Decidir e descrever os requisitos funcionais de um sistema, resultando em um acordo entre clientes, usuários e desenvolvedores.
- Dar uma descrição clara e consistente do que o sistema deve fazer. Essa descrição será utilizada para comunicar todos os requisitos para os desenvolvedores e fornecer base para o projeto do sistema.
- Fornecer uma base para executar testes de sistema, verificar se o sistema funciona corretamente e validar os requisitos (verificar se o sistema inclui todas as funções desejadas).

Várias são as vantagens de se utilizar um diagrama de casos de uso para visualizar um sistema. Entre elas, podemos destacar: a possibilidade de examinar cada caso de uso separadamente e entendê-lo sem precisar conhecer todos os detalhes do sistema; a possibilidade de utilizar os casos de uso como base para estimar quanto tempo e esforço serão necessários para o projeto e a codificação do

sistema; e controlar o desenvolvimento em termos dos casos de uso. Os gerentes podem acompanhar o progresso de cada caso de uso, à medida que o sistema é projetado, codificado e testado (ROBERTSON; ROBERTSON, 2006)

É importante descrever o fluxo de execução de um caso de uso detalhadamente para mostrar como os atores e os sistemas interagem passo a passo. Nessa descrição, são incluídos todos os eventos que podem ocorrer. Não há um padrão para fazer essa descrição. Ela é feita normalmente utilizando o diagrama de atividades da UML, ou uma descrição textual que pode ser mais conveniente em algumas situações. Segundo (JACOBSON; NG, 2004), a forma textual é mais freqüentemente utilizada porque é mais facilmente compreendida pelos usuários finais.

Um caso de uso pode ser visto como um conjunto de cenários ligados por um objetivo em comum de usuário, onde um cenário é uma seqüência de passos que descreve uma interação entre um usuário e um sistema (FOWLER, 2005). Cada passo é uma declaração simples que descreve um elemento da interação. Um estilo comum de descrição de casos de uso se inicia com a escolha de um cenário principal, que é o mais simples, conhecido como fluxo básico de execução. WAZLAWIC (2004) diz que o fluxo básico é também chamado de caminho feliz, pois é uma seqüência na qual se descreve o que acontece quando tudo dá certo. O quadro 2.1 mostra um exemplo de descrição de fluxo para o caso de uso `Fazer pedido`. O fluxo básico desse caso de uso ocorre quando o cliente faz o pedido com sucesso. Como explicado anteriormente, um ponto de extensão identifica o passo do fluxo básico no qual o comportamento adicional, definido no caso de uso de extensão, será inserido. No exemplo do quadro 2.1, o comportamento do caso de uso `Aplicar desconto cliente especial` será inserido, condicionalmente, no passo 5.

Após a descrição do fluxo básico, é necessário descrever comportamentos opcionais ou excepcionais em relação ao comportamento normal do caso de uso descrito no fluxo básico. A descrição da seqüência de passos realizados para tratar uma variação ou exceção é um fluxo alternativo do caso de uso. Todo fluxo alternativo possui uma condição inicial que deve ser atendida para que ele inicie. No exemplo do quadro 2.1 pode ocorrer uma exceção ao executar o passo 7, `se` o cliente preenche alguma informação errada. Os passos necessários para tratar essa exceção são descritos como um fluxo alternativo.

Nome: Fazer pedido.

Atores: Cliente

Pré-condição: Um usuário válido logou no sistema

Fluxo básico:

1. O cliente seleciona a opção fazer pedido.
2. O cliente fornece seu nome e endereço.
3. O cliente informa os códigos dos produtos desejados.
4. O sistema preenche a descrição e preço de cada produto informado.
5. O sistema calcula os valores totais para cada produto fornecido. (Ponto de extensão: Aplicar desconto cliente especial)
6. O cliente preenche informações relativas a pagamento.
7. O sistema verifica as informações, salva o pedido como pendente, e encaminha informações de pagamento para o sistema de cobrança.

Fluxos alternativos:

7a. Existe alguma informação incorreta.

7a.1 O sistema solicita ao usuário que corrija a informação.

Pós-condição: Se o pedido não foi cancelado, ele é salvo no sistema e marcado como confirmado.

Fonte: Adaptado de (SCHNEIDER; WINTERS, 2001)

Quadro 2.1.Descrição textual do caso de uso fazer pedido

Além do fluxo básico e fluxos alternativos, a descrição textual de um caso de uso pode incluir outras seções como: o ator que inicia o caso de uso, como ele é iniciado, pré-condições e pós-condições que descrevem respectivamente o que o sistema garante como verdadeiro antes que o caso de uso seja iniciado e o que o sistema assegura no final do caso de uso e requisitos suplementares aplicados ao caso de uso.

Alguns autores (SCHNEIDER; WINTERS, 2001; COCKBURN, 2001; AMBLER, 2004) descrevem questões de estilo e boas práticas para escrita de casos de uso. Um caso de uso bem escrito, segundo (COCKBURN, 2001), é fácil de ler e consiste de sentenças escritas em somente uma forma gramatical, um passo de ação simples, no qual um ator obtém um resultado ou passa informação para outro ator.

AMBLER (2004) diferencia dois tipos de casos de uso: casos de uso essenciais e casos de uso de sistema. Os casos de uso essenciais são simples e contêm apenas a essência da informação. Eles capturam as intenções do usuário de uma maneira independente de tecnologia e implementação. Já os casos de uso de sistema são uma expansão dos casos de uso essenciais, descrevendo em detalhes como os usuários interagem com o sistema. Estes podem levar em consideração questões tecnológicas e incluem decisões de implementação. Os casos de uso considerados neste trabalho são casos de uso de sistema, que podem conter detalhes de implementação e onde podem ser identificados candidatos a aspectos.

A UML é um verdadeiro padrão para representar modelos de sistemas orientados a objetos. Assim, os casos de uso e a obtenção de requisitos com base em casos de uso são cada vez mais utilizados para diminuir os riscos associados com a extração de requisitos (SOMMERVILLE, 2004).

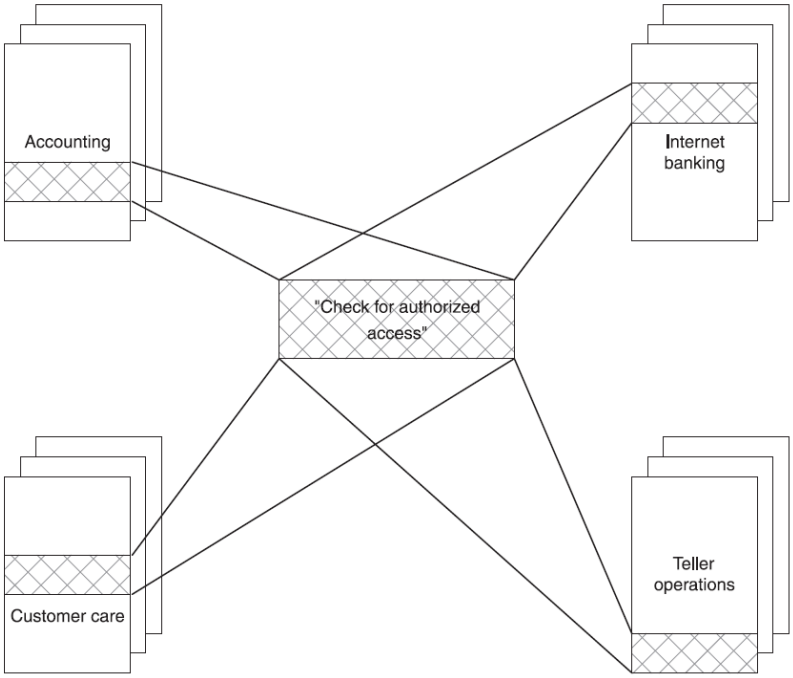
2.3 Programação Orientada a Aspectos

A Programação Orientada a Aspectos (POA) foi proposta em (KICZALES *et al.*, 1997) com o objetivo de melhorar a modularização dos interesses transversais, oferecendo uma nova forma, encapsulada, para implementação desses interesses. Segundo ELRAD *et al.* (2001), a POA é uma tecnologia utilizada para obter melhor separação de interesses, logo, tem como objetivo tornar o projeto e código de sistemas mais modulares, permitindo que seus vários interesses fiquem localizados, ao invés de entrelaçados e espalhados.

A POA surgiu para suprir algumas necessidades da Programação Orientada a Objetos (POO). Utilizando a POO é possível representar de forma encapsulada os interesses de negócio do sistema. Porém, os interesses transversais podem ter seu comportamento distribuído por vários módulos e por esse motivo, é difícil separar esses interesses em um único módulo utilizando as abstrações existentes nas linguagens orientadas a objetos. Exemplos de interesses transversais são: segurança, desempenho, *logging*, integridade de transações, autenticação, persistência, distribuição e tratamento de erros.

A baixa modularização dos interesses transversais acarreta problemas como o espalhamento de código e o entrelaçamento de código. De acordo com (LADDAD, 2003):

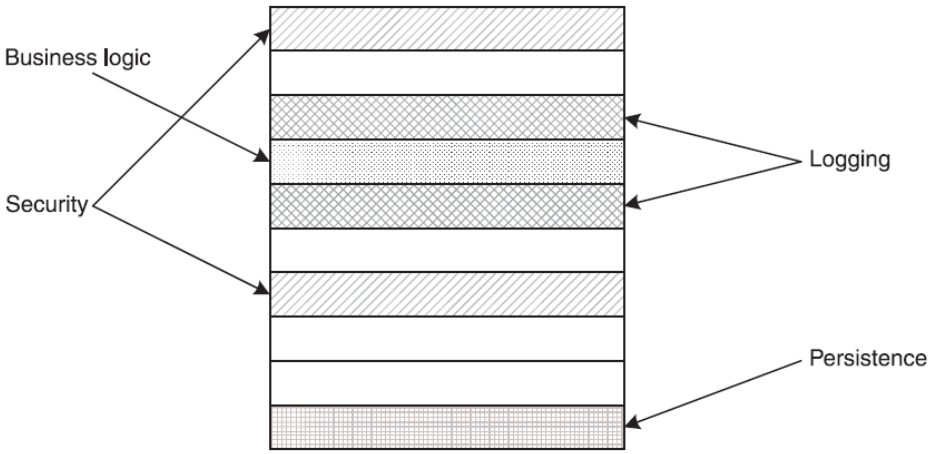
- O **espalhamento de código (*code scattering*)** acontece quando o código de um único interesse transversal está espalhado em diversos módulos do sistema. A figura 2.2 ilustra essa situação.



Fonte: (LADDAD, 2003)

Figura 2.2. Espalhamento de código

- O **entrelaçamento de código (*code tangling*)** acontece quando um único módulo do sistema possui códigos associados a múltiplos interesses simultaneamente. O entrelaçamento de código é apresentado na figura abaixo.



Fonte: (LADDAD, 2003)

Figura 2.3. Entrelaçamento de código

A POA fornece meios para evitar o espalhamento e o entrelaçamento de código. Ela se utiliza de uma nova unidade de abstração, o aspecto (*aspect*), que permite encapsular os interesses transversais, mantendo-os separados dos demais interesses do sistema. Dessa forma, é possível manter os benefícios obtidos com a POO e reduzir a complexidade de desenvolvimento de *software* (MILLER, 2001), utilizando a POA quando as abstrações existentes na POO não são suficientes para que a modularização do sistema seja completa.

Utilizando POA, os interesses do sistema são implementados de forma independente e não afetam uns aos outros. Os aspectos implementam os interesses transversais e os outros interesses do sistema são modelados por meio de classes, a unidade de modularização em POO (TIRELO *et al.*, 2004). As classes não precisam se preocupar com os interesses transversais. A POA fornece um mecanismo de composição desses interesses, encapsulados em aspectos, durante a compilação e execução do sistema (JACOBSON; NG, 2004).

Segundo (RESENDE; SILVA, 2005), a POA permite separar a atividade de desenvolver as funcionalidades requeridas da atividade de integrar o *software*. É possível reutilizar componentes de outros sistemas e alterar sua estrutura interna como atributos e métodos, utilizando a POA, no momento em que a programação da integração estiver sendo feita.

Dentre os vários benefícios obtidos utilizando a POA no desenvolvimento de sistemas, destacam-se (LADDAD, 2003):

- **módulos com responsabilidades mais claras:** a POA permite que um módulo seja responsável somente pelo seu interesse, sem se preocupar com interesses transversais. Isso resulta em melhoria na rastreabilidade.
- **modularização:** a POA fornece um mecanismo para abordar cada interesse separadamente com o mínimo de acoplamento. O resultado é uma implementação modular mesmo na presença de interesses transversais e um sistema com muito menos duplicação de código. Assim, obtém-se maior facilidade de entendimento e manutenção do sistema.
- **facilidade na evolução do sistema:** a POA modulariza os interesses transversais em aspectos e faz com que os módulos funcionais não tenham consciência de que esses interesses estão sendo interceptados. Para adicionar um novo interesse transversal, basta incluir um novo aspecto, sem alterar os outros módulos. Além disso, quando um novo módulo funcional é adicionado

ao sistema, os aspectos existentes podem modificar o mesmo, evitando retrabalho.

- **implementação tardia de requisitos:** com a POA, pode-se adiar a implementação de alguns requisitos, uma vez que é possível implementá-los posteriormente em aspectos separados. Novos requisitos de natureza transversal podem ser tratados criando novos aspectos.
- **reusabilidade de código:** a chave para uma maior reusabilidade de código está em uma implementação com alto grau de coesão. A POA implementa cada aspecto como um módulo separado com uma única responsabilidade, permitindo a implementação com alta coesão.
- **redução do custo da implementação**: a POA evita o custo de modificação de muitos módulos que implementam interesses transversais, permitindo a redução do custo da implementação. Além disso, a produtividade aumenta porque um desenvolvedor mantém o foco na implementação do interesse de responsabilidade do módulo, sem se preocupar com os outros, o que também possibilita a redução do custo de implementação.

RESENDE & SILVA (2005) citam possíveis desvantagens da POA. Dentre elas, a dificuldade de saber exatamente o que acontece em uma classe ao ler o seu código fonte, já que o código de interesses transversais está encapsulado em aspectos fora da classe. Esta característica é conhecida como *obliviousness*. Para amenizar esse problema, existem ferramentas que apresentam o entrelaçamento e espalhamento de código em sua interface, no editor de código fonte, como o plugin de *AspectJ* para o ambiente de programação Eclipse. Outro problema apresentado é a possibilidade de se alterar a estrutura de uma classe. A transversatilidade estática permite inserir novos atributos e operações em classes e também modificar a hierarquia de tipos. Porém, problemas devidos a essa característica podem ocorrer se a POA for utilizada incorretamente, ou sem um planejamento adequado.

2.3.1 Desenvolvimento de *Software* Orientado a aspectos

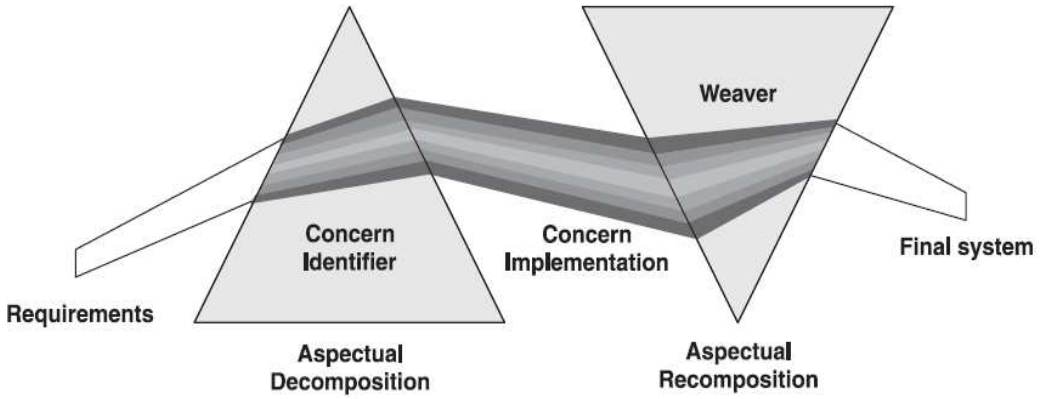
O Desenvolvimento de *Software* Orientado a Aspectos (DSOA) engloba as fases de análise, projeto, implementação, testes e manutenção no processo de desenvolvimento de *software*. Segundo (JACOBSON; NG, 2004), DSOA engloba um conjunto de técnicas para ajudar a conseguir uma melhor modularidade. Essas

técnicas buscam identificar, modularizar, representar e compor interesses transversais (EARLY ASPECTS, 2009).

O DSOA é realizado em três etapas distintas (LADDAD, 2003):

1. **Decomposição aspectual (*aspectual decomposition***): nessa fase, os interesses do sistema são decompostos em interesses de negócio (RFs) e interesses transversais (RNFs).
2. **Implementação dos interesses (*concern implementation*):** após a decomposição, utiliza-se uma linguagem de programação qualquer para implementar cada interesse separadamente, de forma independente. Os interesses de negócio e os interesses transversais são implementados em módulos distintos.
3. **Recomposição aspectual (*aspectual recomposition***): nessa fase, são definidas as regras de recomposição do sistema. Essas regras indicam em que pontos do código os interesses transversais serão inseridos. Elas são especificadas nos aspectos. O processo chamado de recomposição ou costura (*weaving*) se utiliza dessas regras para compor o sistema final.

A figura 2.4 ilustra as etapas do processo de DSOA descritas anteriormente: decomposição e implementação de interesses e, por último, a recomposição que combina as implementações dos interesses utilizando aspectos para obter o sistema completo.



Fonte: (LADDAD, 2003)

Figura 2.4. Desenvolvimento de *Software* Orientado a Aspectos

Portanto, para implementar um sistema orientado a aspectos é necessário uma linguagem de programação qualquer para programar os diversos interesses do sistema, uma linguagem orientada a aspectos para definir as regras de recomposição

do sistema e um *weaver* que é responsável por integrar todos os interesses implementados. *Weaver* é o nome dado aos compiladores de linguagens orientadas a aspectos (RESENDE; SILVA, 2005).

2.3.2 Linguagens Orientadas a Aspectos

As linguagens de POA permitem a composição de interesses independentes do sistema. Para isso, essas linguagens introduziram novos recursos, descritos a seguir (GRADECKI; LESIECKI, 2003):

- **pontos de junção (*joinpoints*):** são pontos bem definidos no fluxo de execução de um programa onde um interesse transversal pode ser inserido. Exemplos de *joinpoints* são as chamadas e execução de métodos, chamada e execução de construtores e a execução de exceções.
- **conjuntos de junção (*pointcuts*):** são pontos de junção agrupados e as informações de contexto desses pontos. Operadores lógicos como && (e), || (ou) e ! (negação) podem ser utilizados na definição de *pointcuts*.
- **regras de junção (*advices*):** são os códigos de interesses transversais que devem ser executados em pontos de junção definidos nos conjuntos de junção. As regras de junção podem ser executadas antes (*before*) ou depois (*after*) de um ponto de junção ou podem alterar o fluxo de execução do programa (*around*) permitindo que o ponto de junção interceptado não seja executado e outro código seja executado em seu lugar.
- **aspectos (*aspects*):** um aspecto é a unidade de implementação de interesses transversais que encapsula pontos de junção, conjuntos de junção e regras de junção. Aspectos são criados da mesma forma que as classes e permitem o encapsulamento completo do código relacionado a um interesse particular.

Assim, uma linguagem de POA permite capturar pontos específicos (*joinpoints*) definindo *pointcuts.* Quando esses pontos são alcançados pelo processador durante a execução do programa, ocorre uma parada e um *advice* pode ser executado (RESENDE; SILVA, 2005). O compilador de aspectos (*weaver*) tem como função inserir os códigos das regras de junção nos pontos de junção definidos nos aspectos.

AspecJ (KICZALES *et al.*, 2001; LADDAD, 2003), *HyperJ* (OSSHER; TARR, 2000), *AspectC* (COADY *et al*., 2001) e *AspectC++* (AspectC++, 2009) são exemplos de linguagens orientadas a aspectos. Existe também uma proposta para

POA independente de linguagem, um compilador chamado *Weave.Net* que permite que aspectos e componentes sejam escritos em várias linguagens (LAERTY; CAHILL, 2003).

Dentre as linguagens orientadas a aspectos, a linguagem *AspectJ* é uma das mais citadas na literatura (ASPECTJ, 2009; TIRELO *et al.*, 2004; GRADECKI; LESIECKI, 2003; LADDAD, 2003; KICZALES, 2001; KICZALES *et al.*, 2001). *AspectJ* é uma extensão da linguagem Java (JAVA, 2009) e utiliza essa linguagem para a implementação dos interesses em módulos separados. Para definir as regras de *weaving*, *AspectJ* utiliza pontos de junção, conjuntos de junção e regras de junção.

AspectJ suporta dois tipos de implementação de interesses transversais: a transversalidade dinâmica (*dynamic crosscutting*), que permite definir implementação adicional em pontos bem definidos na execução do programa, e a transversalidade estática (*static crosscutting*), que permite definir novas operações e membros em assinaturas estáticas de classes e interfaces de um programa Java (TIRELO *et al*., 2004). A transversatilidade dinâmica consiste em inserir um novo comportamento em tempo de execução. Um exemplo seria realizar uma ação antes ou após a chamada de um método. A transversatilidade estática consiste em inserir alterações na estrutura estática do programa. *AspectJ* permite os seguintes tipos de transversatilidade estática: inserir campos e métodos em classes e interfaces, modificar a hierarquia de tipos, declarar erros e advertências de compilação e enfraquecer exceções.

Para ilustrar a utilização dos elementos de *AspectJ,* é apresentado um exemplo. O quadro 2.2 mostra uma classe chamada `Conta` que inclui além dos métodos de acesso (*get)* e modificadores (*set)* para seus atributos, os métodos para realização de transações como `depositar` e `verificar saldo`. O quadro 2.3 ilustra uma classe chamada `Autorizacao` responsável por verificar se um usuário tem a permissão necessária para realizar uma transação. O aspecto `ContaAspect`, apresentado no quadro 2.4, registra a operação `depositar` da classe `Conta` e verifica se um usuário possui autorização para realizar as transações definidas na classe `Conta`.

```
1    public class Conta {
2
3    private int numero;
4    private float saldo;
5    private String nome;
6
7    public Conta(int numero, float saldo, String nome){…}
8
9    public int getNumero(){…}
10   public void setNumero(int numConta){…}
11   public float getSaldo(){…}
12   public void setSaldo(float saldoConta){…}
13   public String getNome(){…}
14   public void setNome(String nomeTitularConta){…}
15
16   public void depositar(float valor) {…}
17
18   public void verificarSaldo(){…}
19   }
20   …
```

Fonte: Elaborada pelo autor.

Quadro 2.2. Conta.java

```
1    public class Autorizacao {
2    …
3
4    public static boolean estaAutorizado(){…}
5
6    …
7    }
```

Fonte: Elaborada pelo autor.

Quadro 2.3. Autorizacao.java

```
1    public aspect ContaAspect {
2
3    pointcut RealizaDeposito(): execution(public void
     Conta.depositar(float));
4    pointcut VerificaAutorizacao(): call(public void Conta.*(..));
5
6    after(): RealizaDeposito(){
7      System.out.println("Transação efetuada: Realização de depósito.");
8
9    }
10
11   void around(): VerificaAutorizacao(){
12     if(Autorizacao.estaAutorizado()){
13       System.out.println("Usuário autorizado a realizar a transação: " +
     thisJoinPoint.getSignature().getName());
14       proceed();
15   }
16
17   else
18     System.out.println("Você não possui permissão para realizar essa
     transação");
19    }
}
```

Fonte: Elaborada pelo autor.

Quadro 2.4. ContaAspect.aj

O aspecto `ContaAspect` define um conjunto de junção chamado `RealizaDeposito` (linha 3) que captura a execução do método `depositar` da classe `Conta`. Foi definida uma regra de junção *`before`* (linha 6) que contém código para ser executado antes da execução do método `depositar` capturado pelo conjunto de junção `RealizaDeposito`. Nesse caso, uma mensagem será impressa antes da execução do método `depositar`. Outro conjunto de junção é definido na linha 4, chamado `VerificaAutorizacao`. Ele captura a chamada de todos os métodos públicos da classe `Conta`. A regra de junção `around` (linha 11) contém o código para verificar a autorização do usuário. Ela permite que os pontos de junção definidos no conjunto de junção `VerificaAutorizacao` sejam interceptados e no seu lugar seja executado o método `estaAutorizado` da classe `Autorizacao.` Se o usuário possuir autorização para executar a operação, o código da chamada interceptada é executado. Isso é feito utilizando `proceed` (linha 14). Caso contrário, uma mensagem é impressa.

No quadro 2.5 é apresentada uma classe `Teste`, responsável por executar operações sobre um objeto do tipo `Conta`. O resultado da execução da classe `Teste` está ilustrado no quadro 2.6.

```
1    public class Teste {
2
3    public static void main(String[] args) {
4    Conta conta = new Conta(1, 0, "João");
5    conta.depositar(200);
6    conta.verificarSaldo();
7     }
8    }
```

Fonte: Elaborada pelo autor.

Quadro 2.5. Teste.java

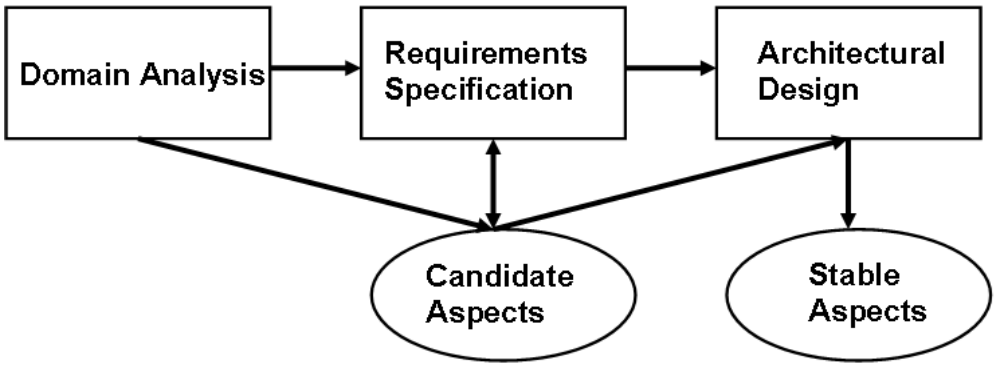
```
Usuário autorizado a realizar a transação: depositar
Transação efetuada: Realização de depósito.
Usuário autorizado a realizar a transação: verificarSaldo
O saldo é R$200.0
```

Fonte: Elaborada pelo autor.

Quadro 2.6. Resultado da execução da classe Teste.java

2.4 *Early Aspects*

Early Aspects (EA) representa o subconjunto de atividades pertencentes ao DSOA que tem como objetivo identificar aspectos nas fases iniciais do desenvolvimento que antecedem a implementação, como análise de domínio, especificação de requisitos e projeto arquitetural (RESENDE *et al*., 2005). A figura 2.5 ilustra essas fases e pode-se observar que os interesses transversais identificados são chamados de candidatos a aspectos até que seja tomada a decisão de implementá-los como aspecto. Essa é uma decisão da fase de projeto. A expressão EA também é utilizada para se referir aos candidatos a aspectos identificados nas fases iniciais do ciclo de vida do *software*. Este trabalho preocupa-se com a identificação de aspectos na fase de especificação requisitos.



Fonte: (RESENDE *et al*., 2005)

Figura 2.5. Escopo de *Early Aspects*

A identificação de aspectos na fase de requisitos permite melhorar a modularização dos requisitos, detectar conflitos mais cedo e manter a separação de interesses nas fases posteriores do processo. Isso evita custos relacionados a refatoração de código (*refactoring).* RASHID *et al*. (2002), demonstra que a separação inicial de interesses transversais suporta a determinação efetiva de seu mapeamento e influência em artefatos nas fases de desenvolvimento posteriores.

Para (ARAÚJO *et al*., 2005), um aspecto no nível de requisitos é uma propriedade de escopo geral representada por um requisito ou um conjunto coerente de requisitos como, por exemplo, os requisitos de segurança, que afetam muitos outros requisitos do sistema de forma a poder restringir e/ou alterar o comportamento específico dos requisitos afetados. Para (BANIASSAD *et al*., 2006), um aspecto nos

requisitos é um interesse que afeta múltiplos artefatos produzidos na fase de requisitos.

Algumas técnicas podem ser utilizadas para auxiliar a identificação de aspectos nos requisitos. ROSENHAINER (2004) destaca duas delas: a identificação através de inspeção, em que uma especificação de requisitos ou parte dela é inspecionada em busca de candidatos a aspectos; e a identificação suportada por técnicas de recuperação de informação, em que um programa busca por declarações que parecem ser afetadas por um interesse que o analista imagina ser transversal. Segundo (BANIASSAD *et al*., 2006), ao fazer a inspeção nos requisitos, deve-se procurar por:

- Atributos de qualidade (tais como segurança, consistência de dados ou confiabilidade) do sistema. Engenheiros de requisitos podem identificar palavras relacionadas a esses atributos de qualidade usando técnicas de busca simples ou ferramentas feitas especialmente para análise de requisitos orientada a aspectos, como a ferramenta *EA-Miner* (CHITCHYAN *et al*., 2006a; CHITCHYAN *et al*., 2006b).
- Requisitos que descrevem a influência de um interesse sobre outro. É necessário identificar quais interesses são afetados e como os interesses transversais afetam esses interesses. Técnicas de visualização como *Theme/ Doc* (BANIASSAD; CLARKE, 2004a; BANIASSAD; CLARKE, 2004b) e *EA-Miner* (CHITCHYAN *et al*., 2006a; CHITCHYAN *et al*., 2006b) podem apontar tais requisitos porque elas mostram quais requisitos descrevem quais interesses revelando sobreposição de interesses.
- Interesses espalhados. Termos, conceitos, ou comportamentos que aparecem em múltiplos requisitos. É possível que nem todos os interesses espalhados sejam capturados como aspectos.

2.5 Trabalhos correlatos

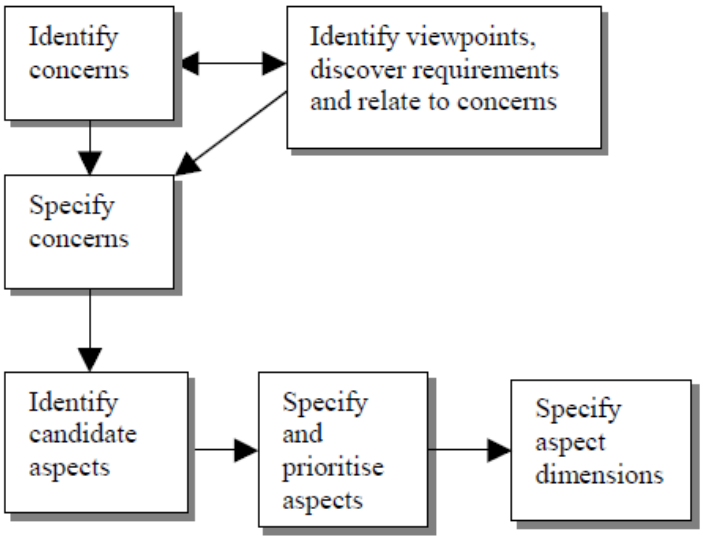
Nesta subseção, serão apresentados alguns trabalhos encontrados na literatura relacionados ao trabalho desenvolvido na presente dissertação. Os trabalhos apresentados podem ser agrupados em: trabalhos que tratam a identificação de aspectos em documentos de requisitos (BRITO *et al*.,2002; MOREIRA *et al*., 2002; RASHID *et al.*, 2002; BANIASSAD; CLARKE, 2004a; BANIASSAD; CLARKE, 2004b; CHITCHYAN *et al*., 2006a; CHITCHYAN *et al*., 2006b; BANIASSAD *et*

al., 2006; RESENDE, 2007); e trabalhos que relacionam casos de uso e aspectos (ARAÚJO; COUTINHO, 2003; JACOBSON, 2003; JACOBSON; NG, 2004). O objetivo é situar o trabalho realizado na área de pesquisa em EA, mostrar como estes trabalhos influenciaram a abordagem proposta e mostrar como o trabalho apresentado pode complementar os trabalhos correlacionados.

A extração de aspectos, foco desta dissertação, é uma das atividades realizadas durante a Engenharia de Requisitos Orientada a Aspectos (EROA). A EROA engloba as atividades realizadas durante a Engenharia de Requisitos responsáveis por tratar os interesses transversais. Segundo (ARAÚJO *et al*., 2005), a EROA tem como objetivo fornecer uma forma sistemática para a identificação, modularização, representação e composição de propriedades transversais, funcionais e não funcionais, durante a Engenharia de Requisitos.

Em 2002, vários trabalhos se destacaram na área de EROA. BRITO *et al*. (2002) e MOREIRA *et al*. (2002) apresentam um modelo para identificar e especificar atributos de qualidade, RNFs, que atravessam vários requisitos e integrar esses atributos com os RFs. Os RFs são especificados usando casos de uso e um *template* foi proposto para especificar RNFs no estágio de requisitos. Diagramas de casos de uso e diagramas de seqüência UML foram estendidos para representar a composição de RNFs e RFs. Os RNFs são identificados ao analisar o conjunto inicial de requisitos.

Em (RASHID *et al.*,2002), um modelo para EROA é proposto. Esse modelo também suporta a separação de RFs e RNFs no nível de requisitos, mas utiliza *viewpoints* como técnica para obter a separação de interesses do sistema, ao invés de casos de uso como no modelo proposto em (BRITO *et al*., 2002) e (MOREIRA *et al*., 2002). De acordo com (FINKELSTEIN; SOMMERVILLE, 1996), um *viewpoint* é a combinação de um agente e a visão desse agente com relação ao sistema que está sendo descrito ou modelado, sendo que um agente é um participante ou ator na construção dessa descrição. As atividades realizadas no modelo proposto por (RASHID *et al*., 2002) são apresentadas na figura 2.6: identificar interesses; identificar *viewpoints*, descobrir requisitos e relacioná-los aos interesses; especificar interesses; identificar candidatos a aspectos; especificar e priorizar aspectos; e especificar as dimensões dos aspectos. Técnicas para auxiliar a identificação de candidatos a aspectos são necessárias ao utilizar esse modelo.



Fonte: (RASHID *et al*., 2002)

Figura 2.6. Modelo para EROA

Pode-se observar que os modelos descritos acima utilizam a técnica de inspeção para identificar candidatos a aspectos: RNFs nos requisitos do sistema (BRITO *et al*., 2002; MOREIRA *et al*., 2002) ou interesses que afetam requisitos de dois diferentes *viewpoints* (RASHID *et al*., 2002). O presente trabalho visa contribuir com a atividade de identificação de candidatos a aspectos na EROA, apresentando uma nova forma de auxiliar a inspeção utilizando a descrição de casos de uso como documento de análise.

Outro trabalho que trata a identificação de aspectos em requisitos é proposto em (BANIASSAD; CLARKE, 2004a) e (BANIASSAD; CLARKE, 2004b). A abordagem conhecida como *Theme/Doc* permite visualizar os relacionamentos entre comportamentos em um documento de requisitos para identificar e isolar aspectos. Os temas (*themes*) representam as características do sistema. Aqueles temas que possuem comportamentos associados a vários outros temas, são chamados de temas transversais (*crosscutting themes*) e são identificados como aspectos. Uma ferramenta que suporta a abordagem *Theme/Doc* identifica interesses transversais de forma semi-automática nas especificações de requisitos, detectando ocorrências repetidas de palavras de ação que sugerem RFs candidatos a aspecto. O usuário deve fornecer manualmente a lista com as palavras de ação como documento de entrada. Os resultados são apresentados em uma visão gráfica que mapeia os relacionamentos entre os interesses. A ferramenta *Theme/Doc* fornece suporte semi-automático para identificação de aspectos, porém depende de palavras chaves que podem ser

fornecidas erroneamente pelo usuário do sistema. Além disso, essa abordagem não identifica RNFs candidatos a aspectos.

Outra ferramenta que fornece suporte automatizado para identificação de candidatos a aspectos no nível de requisitos é chamada *EA-Miner* (CHITCHYAN *et al*., 2006a; CHITCHYAN *et al*., 2006b). A ferramenta *EA-Miner* usa texto em linguagem natural como entrada e auxilia a geração de documentos de especificação de requisitos, identificando interesses base (*viewpoints*, casos de uso e cenários). Ela utiliza técnicas de processamento de linguagem natural para identificar a dispersão de interesses nos documentos de entrada e buscar candidatos a aspectos conhecidos nesses documentos, porém descarta a possibilidade de utilizar a estrutura de um documento de requisitos que poderia facilitar a identificação de candidatos a aspectos.

A principal diferença entre as abordagens utilizadas nas ferramentas *Theme/Doc* (BANIASSAD; CLARKE, 2004a; BANIASSAD; CLARKE, 2004b) e *EA-Miner* (CHITCHYAN *et al*., 2006a; CHITCHYAN *et al*., 2006b) e a apresentada nesta dissertação, é que esta propõe a utilização de um documento de especificação de requisitos estruturado, a descrição de casos de uso, e utiliza essa estrutura para identificar os candidatos a aspectos.

O trabalho de (BANIASSAD *et al*., 2006) descreve como identificar e capturar candidatos a aspectos nos requisitos e no projeto arquitetural, e como os candidatos a aspectos são mapeados de uma fase para outra durante o desenvolvimento de *software*. A abordagem apresentada nesse trabalho oferece uma visão integrada que utiliza práticas de várias abordagens existentes para trabalhar com EA. Para identificar aspectos nos requisitos, BANIASSAD *et al*. (2006) sugere procurar por atributos de qualidade, requisitos que descrevem a influência de um interesse sobre outro e interesses espalhados como apresentado na seção 2.4. Esses são os conceitos que buscamos nas descrições de casos de uso para identificar candidatos a aspectos neste trabalho.

RESENDE (2007) apresenta um método para identificação e definição de aspectos iniciais (MIDAI) com o objetivo de aumentar a eficiência e a eficácia e reduzir os riscos e recursos envolvidos na utilização da POA. O MIDAI é dividido em duas atividades principais. A primeira atividade, identificação de aspectos iniciais candidatos, utiliza um conjunto de heurísticas para identificar aspectos iniciais em um documento de especificação de requisitos. A segunda atividade é a definição de

quais aspectos iniciais identificados na primeira atividade apresentam mais vantagens, caso sua implementação seja realizada utilizando OA. Essa atividade é realizada utilizando uma equação de decisão que possibilita atribuir pesos a cada aspecto inicial candidato.

Dentre os trabalhos que relacionam casos de uso e aspectos, um trabalho que se destacou é apresentado em (ARAÚJO; COUTINHO, 2003). Esse trabalho descreve como aspectos poderiam ser integrados a um método de requisitos orientado a *viewpoints* que integra modelos UML. São identificados os atores e casos de uso para cada *viewpoint*. *Templates* são utilizados para descrever RNFs, *viewpoints* e casos de uso. Os RNFs que atravessam mais de um caso de uso ou *viewpoint* são candidatos a aspectos. Casos de uso incluídos por mais de um caso de uso ou que estendem mais de um caso de uso são chamados de casos de uso aspectuais e também são candidatos a aspectos. Casos de uso que atravessam vários *viewpoints* também são casos de uso aspectuais. Essa abordagem sugere que RNFs, casos de uso incluídos e extensões de casos de uso são candidatos a aspectos, porém não busca identificar os RNFs que podem surgir no fluxo de descrição de casos de uso. Além disso, essa abordagem não se preocupa em identificar os pontos em que os RNFs, casos de uso incluídos ou extensões de casos de uso são inseridos na descrição de casos de uso.

Em (JACOBSON, 2003), é apresentada outra abordagem relacionando casos de uso e aspectos. JACOBSON (2003) afirma que a POA fornece o *link* que faltava a fim de manter a modularização de casos de uso no projeto e implementação, e apresenta uma abordagem de desenvolvimento de *software* em que o sistema completo é desenvolvido a partir de um caso de uso base e extensões a esse caso de uso base. Todas as extensões seriam compostas ao caso de uso base utilizando orientação a aspectos. A solução proposta considera uma transição direta do modelo de casos de uso para o projeto orientado a aspectos através do mapeamento entre as construções de sua técnica de extensão e as construções de POA. Esse mapeamento é apresentado na tabela 2.1.

Como ilustrado na tabela 2.1, pontos de extensão no caso de uso base podem ser implementados como pontos de junção no fluxo de execução do programa base utilizando POA e as extensões de casos de uso são como os *advices* que modificam o comportamento do programa quando alcançam aqueles pontos de junção. Esse mapeamento influenciou a realização da pesquisa apresentada nesta dissertação, pois

pode ser utilizado nas atividades de EROA. O mapeamento entre outras construções do modelo de casos de uso e POA pode ser identificado e a tabela 2.1 pode ser complementada.

Tabela 2.1. Mapeamento entre modelo de casos de uso e POA

Modelo de casos de uso	Equivalente na POA
Caso de uso base	Programa base
Extensão	*Advice*
Pontos de extensão	*Join points*
Lista de pontos de extensão	*Pointcuts*

Fonte: Traduzido de (JACOBSON, 2003)

A abordagem apresentada em (JACOBSON, 2003) é complementada em (JACOBSON; NG, 2004). Além da solução para manter extensões separadas, é apresentada uma solução para manter interesses *peers* separados. Interesses *peers* são interesses distintos, que não necessitam um do outro para existir. Porém, quando esses interesses são implementados, observa-se espalhamento e entrelaçamento de código (JACOBSON; NG, 2004). Essa abordagem não busca identificar interesses transversais, mas modularizar esses interesses em casos de uso e compor esses interesses ao sistema usando POA. Alguns conceitos de POA, como os *pointcuts*, são adicionados à descrição de casos de uso. Porém, a descrição de casos de uso apresentada é complexa, pode ter mais de um fluxo básico e requer conhecimento dos recursos da POA. Para descrever uma extensão de caso de uso, por exemplo, deve-se incluir o *extension pointcut*, o equivalente ao ponto de extensão no caso de uso base.

3 EXTRAÇÃO DE CANDIDATOS A ASPECTOS EM DESCRIÇÕES DE CASOS DE USO

Após o estudo e a compreensão dos conceitos relacionados ao trabalho, foram analisadas diversas descrições de casos de uso encontradas na literatura (AMBLER, 2004; COCKBURN, 2001; FOWLER, 2005; JACOBSON; NG, 2004; SCHNEIDER; WINTERS, 2001; WAZLAWIC, 2004). Essas descrições de casos de uso foram analisadas em busca de candidatos a aspectos utilizando a abordagem proposta por (BANIASSAD *et al.*, 2006).

3.1 Requisitos candidatos a aspectos

Um interesse que afeta múltiplos artefatos produzidos na fase de requisitos é um candidato a aspecto (BANIASSAD *et al.*, 2006). Neste trabalho, os artefatos considerados são as descrições de casos de uso. Logo, os interesses que aparecem em múltiplas descrições de casos de uso são, provavelmente, candidatos a aspectos. Durante a análise das descrições de casos de uso, foi observado que é possível identificar RNFs e RFs candidatos a aspectos e que eles costumam aparecer em seções específicas da descrição de casos de uso de sistema. Nos próximos tópicos são apresentadas discussões sobre a identificação de RNFs e RFs candidatos a aspectos nas descrições de casos de uso analisadas.

3.1.1 Não-funcionais

BANIASSAD *et al*. (2006) sugere localizar os atributos de qualidade do sistema durante a inspeção nos requisitos em busca de aspectos. Os atributos de qualidade, ou RNFs, são restrições impostas ao funcionamento do sistema. São interesses transversais que afetam naturalmente outros requisitos, portanto, são fortes candidatos a aspectos.

Os casos de uso têm sido utilizados para especificar RFs, que constituem o objetivo final do sistema (TIRELO *et al.*, 2004), ao contrário dos RNFs. Os RNFs

normalmente aparecem em seções separadas como requisitos especiais ou suplementares à descrição de casos de uso na especificação de requisitos. Porém, ao buscar RNFs em casos de uso, foi observado que alguns deles podem ser capturados no fluxo de eventos de casos de uso em uma descrição mais completa, como em casos de uso de sistema.

Nas seções *pré-condições* da descrição de casos de uso, por exemplo, foram identificadas palavras relacionadas a um RNF `segurança` do sistema. Na descrição textual do caso de uso `fazer pedido,` apresentado no quadro 2.1, a pré-condição "Um usuário válido logou no sistema" se refere à permissão para executar o caso de uso `fazer pedido.` O mesmo acontece no exemplo do quadro 3.1, em que a pré-condição "O cliente foi autenticado pelo sistema" se refere à permissão para executar o caso de uso `reservar quarto`. A autorização necessária para `fazer pedido` ou `reservar quarto` está relacionada à `confiabilidade` do sistema, que é um subtipo do RNF `segurança`, segundo classificação de tipos e subtipos de RNFs de CHUNG *et al*. (2000). Outros exemplos identificados na seção *pré-condições* são apresentados na tabela 3.1. Segundo COCKBURN (2001), as pré-condições mais comuns dizem que o usuário está logado ou que o usuário foi validado.

Como as pré-condições descrevem o que o sistema garante como verdadeiro antes que o caso de uso seja iniciado, elas não serão verificadas novamente durante a execução do caso de uso, mas podem ser verificadas novamente para que outros casos de uso possam ser iniciados. Elas podem aparecer em múltiplas descrições de casos de uso. A verificação dessas pré-condições pode ser implementada utilizando POA. Um exemplo de solução orientada a aspectos para verificação de uma pré-condição é apresentada no quadro 2.4. O aspecto `ContaAspect` verifica se um usuário possui autorização para realizar as transações definidas na classe `Conta.` A regra de junção `around` altera o fluxo de execução do programa e só permite a execução dos métodos da classe `Conta` se o usuário estiver autorizado.

O RNF de `persistência` foi identificado nas seções *pós-condições* das descrições de casos de uso analisadas. As pós-condições do sistema descrevem o que o sistema assegura após a execução bem-sucedida do caso de uso (independente do fluxo percorrido), ou satisfazem as pré-condições dos próximos casos de uso (WIEGERS, 2003). O RNF de `persistência` está associado ao armazenamento

Descrição: Esse caso de uso descreve como um cliente reserva um quarto.

Pré-condição: O cliente foi autenticado pelo sistema.

Fluxo básico:

O caso de uso começa quando um cliente deseja reservar um quarto.

1. O cliente seleciona a opção reservar um quarto.
2. O sistema mostra os tipos de quartos existentes no hotel.
3. O cliente Verifica despesa total da estadia.
4. O cliente faz a reserva para o quarto escolhido.
5. O sistema deduz do banco de dados o número de quartos do tipo escolhido disponíveis para reserva.
6. O sistema cria uma nova reserva.
7. O sistema mostra o número de confirmação da reserva e instruções para realizar check in.
8. O caso de uso termina.

Fluxos alternativos:

5a. Existe uma reserva idêntica no sistema.

5a.1 O sistema mostra a reserva existente e pergunta ao cliente se ele deseja continuar com a reserva.

5a.2 Se o cliente deseja continuar, o sistema procede a nova reserva.

5a.3 Se é uma duplicata, o caso de uso termina.

Subfluxos:

S1. Verifica despesa total da estadia.

1. O cliente seleciona seu tipo de quarto desejado e indica seu período de estadia.
2. O sistema calcula o custo para o período especificado.

Pós-condição: Um novo registro de reserva é criado e o número de quartos disponíveis para os dados especificados é decrementado. Se a reserva não foi bem sucedida, não há alteração no banco de dados.

Requisitos especiais

O sistema deve tratar cinco reservas concorrentes.

Cada reserva não deve demorar mais do que 20 segundos.

Fonte: Traduzido de (JACOBSON; NG, 2004)

Quadro 3.1.Descrição do caso de uso Reservar Quarto

permanente de informações no sistema. Observe os exemplos de pós-condições apresentadas na tabela 3.1. Elas indicam que alguma informação foi armazenada durante o fluxo de execução do caso de uso (nos 2 primeiros exemplos da tabela) ou deve ser armazenada permanentemente, deve ser realizada após a execução do caso de uso (nos 2 últimos exemplos da tabela).

Tabela 3.1. Exemplos de RNFs identificados nas descrições de casos de uso analisadas

Seção da descrição de casos de uso	Exemplos de RNFs identificados
Pré-condição	• Autenticação do funcionário junto ao sistema. • O usuário deve ter feito "log-in" e obtido autorização do sistema. • O usuário está registrado na empresa. • O cliente foi autenticado pelo sistema.
Pós-condição	• O estudante foi registrado no curso. • Um novo registro de reserva foi criado. • O pedido é gravado no sistema e marcado como confirmado. • A situação do pedido é marcada como cancelada.
Fluxos alternativos	• O pedido não foi encontrado. • Dados submetidos incompletos. • O estudante não atende os pré-requisitos necessários. • As informações não conferem.
Requisitos especiais	• O sistema deve responder a qualquer comando do usuário em 1 segundo. • O sistema deve tratar 5 reservas concorrentes. • Cada reserva não pode demorar mais que 20 segundos.

Fonte: Elaborada pelo autor.

Outro exemplo interessante do RNF `persistência` identificado em pós-condições é apresentado no exemplo do quadro 3.1. O registro de reserva de quarto deve ser criado e o número de quartos disponíveis para reserva deve ser decrementado no banco de dados imediatamente após a execução bem-sucedida do caso de uso, mantendo as informações armazenadas atualizadas para os próximos casos de uso. Isso poderia ser realizado utilizando POA: os métodos responsáveis por armazenar essas informações no banco de dados do sistema (regras de junção)

podem ser inseridos no fluxo de execução do programa após (*after*) a execução do(s) método (s) implementado(s) para realizar os casos de uso (pontos de junção).

A seção *fluxos alternativos* em uma descrição de casos de uso descreve uma variação no comportamento normal desse caso de uso ou um comportamento excepcional, onde uma exceção está sendo tratada. `Tratamento de exceção` é um subtipo do RNF `segurança.` Logo, um fluxo alternativo tratando uma exceção pode ser incluído no fluxo principal de um caso de uso usando POA. Um exemplo para o fluxo alternativo apresentado na descrição do caso de uso do quadro 3.1 é apresentado nos quadros 3.2 e 3.3.

A classe `TratarReservaQuarto` (quadro 3.2) tem uma operação chamada `fazerReserva` (linha 3) que lança uma exceção `ExcecaoReservaIdentica` se for encontrada uma reserva idêntica no sistema. O aspecto `TrataReservaIdentica` (quadro 3.3) define um conjunto de junção chamado `fazendoReserva` (linha 3) que captura a execução do método `fazerReserva` da classe `TratarReservaQuarto.` Foi definida uma regra de junção ***`after throwing`*** (linha 6) que contém código para ser executado após ocorrer uma exceção `ExcecaoReservaIdentica` capturado pelo conjunto de junção `fazendoReserva.` O caso de uso `Emprestar Fitas`, apresentado no quadro 3.4, inclui outros exemplos de fluxos alternativos tratando exceções.

```
1     class TratarReservaQuarto {
2     ...
3     public void fazerReserva() throws ExcecaoReservaIdentica {
4
5         if(reserva.verificaReservaIdentica()) {
6               throw new ExcecaoReservaIdentica();
7         }
8
9         criarReserva();
10
11       }
12    ...
13
14    }
```

Fonte: Traduzido de (JACOBSON; NG, 2004)

Quadro 3.2. TratarReservaQuarto.java

```
1     aspect TrataReservaIdentica {
2       ...
3     pointcut fazendoReserva():
4      execution(void TratarReservaQuarto.fazerReserva());
5     ...
6     after throwing (ExcecaoReservaIdentica e) : fazendoReserva(){
7          // Código para tratar reservas idênticas no sistema
8       }
9
10    }
```

Fonte: Traduzido de (JACOBSON; NG, 2004)

Quadro 3.3. TrataReservaIdentica.aj

Nome: Emprestar fitas

Fluxo básico:

1. O cliente chega ao balcão com as fitas que deseja locar.
2. O cliente informa o seu nome e entrega as fitas ao funcionário.
3. O funcionário registra o nome do cliente e inicia a locação.
4. O funcionário registra cada uma das fitas.
5. O funcionário finaliza a locação, devolve as fitas ao cliente e lhe informa a data de devolução e o valor total da locação.
6. O cliente deixa a locadora com as fitas.

Fluxos alternativos:

3a. O cliente não possui cadastro.

3a.1 O cliente deve informar seus dados para cadastro.

3a.2 O funcionário registra o cadastro.

3a.3 Retorna ao fluxo principal no passo 3.

3b. O cliente possui pendências no cadastro (locação anterior não foi paga)

3b.1 O cliente paga seu débito.

3b.2 O funcionário registra a quitação do débito, eliminando assim a pendência.

3b.3 Retorna ao passo 3.

4a. Uma fita está reservada para outro cliente

4a.1 O funcionário informa que a fita não está disponível para locação.

4a.2 Prossegue a locação do passo 4 sem incluir a fita reservada.

Fonte: (WAZLAWIC, 2004)

Quadro 3.4. Parte da descrição textual do caso de uso essencial `Emprestar Fitas`

Se o fluxo alternativo descreve uma variação no comportamento normal do caso de uso, o comportamento descrito no fluxo alternativo também pode ser incluído no fluxo principal usando POA. Nesse caso, um *advice* ***around*** pode ser usado para substituir o comportamento do método executado se a condição inicial do fluxo alternativo for satisfeita, ou prosseguir se a condição não for satisfeita.

Nas seções *requisitos especiais* das descrições de casos de uso analisadas, foram identificados os RNFs: `desempenho` (subtipo tempo de resposta) e concorrência. Os exemplos são apresentados na tabela 3.1. A restrição de tempo imposta para `reservar quarto` no exemplo do quadro 3.1 se relaciona a um RNF `desempenho` do sistema.

A tabela 3.1 apresenta alguns dos RNFs identificados nas descrições de casos de uso analisadas neste trabalho. Outros RNFs podem ser identificados em outras descrições de casos de uso. Todos os RNFs identificados são candidatos a aspectos. As palavras-chave que indicam relação com algum RNF, ou seja, que representam operacionalizações desses RNFs encontram-se sublinhadas nessa tabela.

3.1.2 Funcionais

Os RFs descrevem as funcionalidades esperadas em um sistema e de interesse dos usuários. RFs podem afetar outros requisitos do sistema, acarretando espalhamento de código na implementação, e, conseqüentemente, dificultando a modularização do sistema.

Segundo BANIASSAD *et. al* (2006), ao buscar aspectos em requisitos é necessário, além de identificar os RNFs, identificar também os termos, conceitos, ou comportamentos que aparecem em múltiplos requisitos e identificar os requisitos afetados por eles. Nas descrições de casos de uso analisadas, foi observado que os comportamentos repetidos que aparecem em múltiplos casos de uso já se encontram separados em casos de uso de inclusão se a descrição foi realizada corretamente. O relacionamento *include* entre casos de uso é utilizado para evitar a descrição repetida de uma seqüência de passos em mais de um caso de uso, ou seja, descreve um requisito que afeta uma lista de casos de uso. Os requisitos afetados nesse caso são os casos de uso base que contém o comportamento definido no caso de uso incluído. Logo, os RFs descritos em casos de uso de inclusão são considerados candidatos a aspectos.

O comportamento descrito em um caso de uso de inclusão pode ser incluído no fluxo de execução de todos os casos de uso base que possuem esse comportamento usando POA. Os pontos de inclusão indicam os locais nos casos de uso base onde o caso de uso será incluído. No projeto do sistema orientado a aspectos, esses pontos de inclusão podem ser implementados como pontos de junção. Um conjunto de junção agrupa todos esses pontos, onde o comportamento descrito no caso de uso de inclusão (*advice*) será inserido.

Um exemplo de caso de uso de inclusão é apresentado no quadro 3.1. A descrição do caso de uso `Reservar quarto` possui um subfluxo (relacionamento *include*) chamado `Verifica despesa total da estadia`, incluído no passo 3 desse caso de uso. O subfluxo pode ser incluído em outros casos de uso do sistema que necessitem calcular o custo total da estadia como, por exemplo, um caso de uso `Realizar check out`. O método que implementa o subfluxo `Verifica despesa total da estadia` (regra de junção) pode ser inserido no fluxo de execução do programa após (*after*) a execução dos métodos implementados para realizar o passo anterior aos pontos de inclusão (pontos de junção) nos casos de uso base.

Os casos de uso de extensão não costumam estender o comportamento de mais de um caso de uso, ou seja, não descrevem comportamentos repetidos. A descrição de comportamento repetido também pode acontecer se um ou mais casos de uso forem incluídos no fluxo de um caso de uso de extensão. Apesar disso, eles são considerados candidatos a aspectos porque podem ser compostos ao sistema utilizando POA. Como apresentado na seção 2.3, JACOBSON (2003) descreve um mapeamento entre as construções do modelo de casos de uso para o projeto orientado a aspectos, mostrando como implementar extensões de casos de uso como aspectos. Apesar de não ser uma prática comum, o fluxo básico de um caso de uso de extensão pode ter um ou mais casos de uso incluídos.

Considerando que os casos de uso incluídos também são candidatos a aspectos, a transição para as construções de POA também é direta da mesma forma que JACOBSON (2003) apresenta o mapeamento para o relacionamento de extensão. É proposta então, neste trabalho, a modificação da tabela 2.1 para apresentar também o mapeamento para o relacionamento de inclusão. A tabela 3.2 apresenta a modificação proposta, adicionando as inclusões (casos de uso de inclusão), os pontos

de inclusão e a lista de pontos de inclusão. Os casos de uso de inclusão são como os *advices* que modificam o comportamento do programa quando alcançam pontos de junção. Os pontos de inclusão no caso de uso base podem ser implementados como pontos de junção no fluxo de execução do programa base utilizando POA e a lista de pontos de junção corresponde aos conjuntos de junção.

Tabela 3.2. Modificação da tabela 2.1 incluindo casos de uso de inclusão e pontos de inclusão

Modelo de casos de uso	**Equivalente na POA**
Caso de uso base	Programa base
Extensão, **inclusão**	*Advice*
Pontos de extensão, **pontos de inclusão**	*Join points*
Lista de pontos de extensão, **lista de pontos de inclusão**	*Pointcuts*

Fonte: Adaptado de (JACOBSON, 2003).

3.2 Descrição da técnica

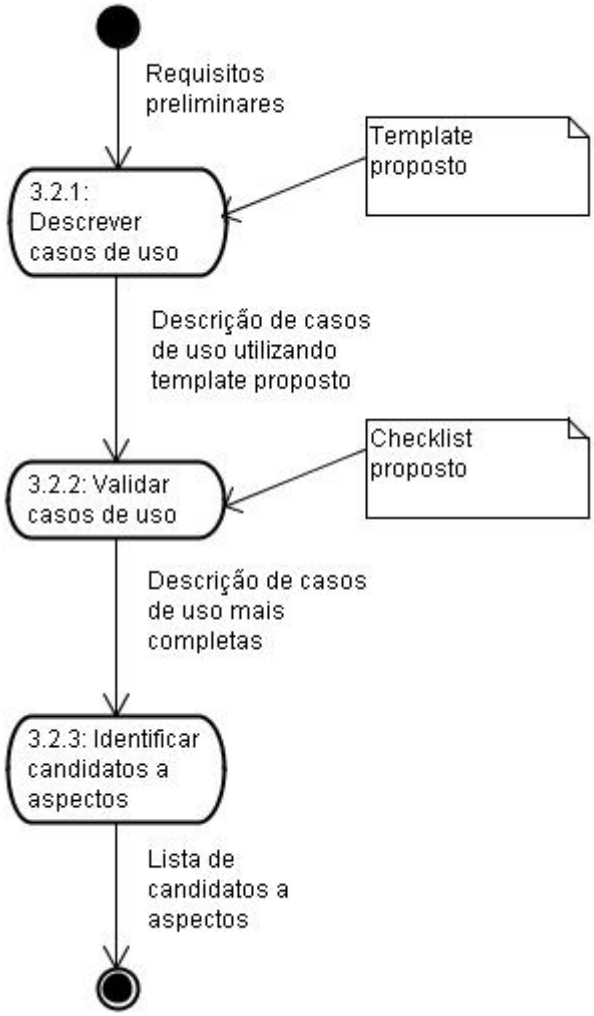
Como apresentado na seção 3.1, durante a análise de casos de uso foram observados alguns padrões relacionados à seção da descrição em que candidatos a aspectos foram encontrados e tipo de requisitos candidatos a aspectos identificados em cada seção. A identificação desses padrões foi essencial para obter uma técnica para auxiliar a inspeção em busca de candidatos a aspectos em descrições de casos de uso.

A técnica proposta neste trabalho busca identificar candidatos a aspectos analisando um conjunto de casos de uso descritos em um formato (*template)* específico. Esse *template* foi projetado para descrever casos de uso de forma estruturada, com seções pré-definidas, nas quais podem ser identificados RNFs e RFs candidatos a aspectos.

A técnica é constituída pelas seguintes atividades, apresentadas na figura 3.1:

- **Descrever casos de uso:** A partir dos requisitos iniciais do problema, os casos de uso são descritos utilizando o *template* proposto. Esse *template* contém as seções nas quais candidatos a aspectos foram identificados durante a análise de descrições de casos de uso encontradas na literatura.

- **Validar casos de uso:** Os casos de uso descritos utilizando o *template* podem ser melhorados, modificados, utilizando um *checklist* para verificação da descrição.
- **Identificar candidatos a aspectos:** A relação de candidatos a aspectos é obtida por inspeção nas descrições de casos de uso. Durante essa atividade, os casos de uso são analisados em busca de RNFs e RFs candidatos a aspectos.



Fonte: Elaborada pelo autor.

Figura 3.1. Diagrama de atividades para a técnica proposta.

Informações adicionais sobre essas atividades são apresentadas nos próximos tópicos, incluindo como o *template* e *checklist* foram projetados.

3.2.1 Descrever casos de uso

Os usuários podem descrever casos de uso das mais variadas formas, desde casos de uso simples e informais até casos de uso formais, com vários detalhes. Segundo COCKBURN (2001), a escolha do formato para descrever casos de uso e o grau de detalhamento de uma descrição depende da sua finalidade. Como neste trabalho a descrição será utilizada para análise em busca de candidatos a aspectos, é interessante utilizar um *template* mais completo, incluindo todas as seções em que candidatos a aspectos foram identificados para facilitar uma análise posterior.

O *template* proposto, *EAI Template (Template for **E**arly **A**spects **I**dentification)*, é baseado nos t*emplates* de COCKBURN e do RUP descritos em (COCKBURN, 2001). Os *templates* de COCKBURN e do RUP (COCKBURN, 2001) e outros encontrados na literatura, ou não incluem todas as seções em que candidatos a aspectos foram identificados, ou não utilizam a numeração de passos, essenciais para identificação de candidatos a aspectos.

O *EAI template* utiliza a numeração de passos nas seções *fluxo básico* e *fluxos alternativos*, a combinação de dígitos e letras para numerar passos, e os casos de uso incluídos sublinhados do *template* de COCKBURN (COCKBURN, 2001). Como descrito na seção anterior, os casos de uso incluídos e as extensões são candidatos a aspectos. Por esse motivo, é importante destacar esses casos de uso na descrição para facilitar a identificação durante a inspeção. O *EAI template* apresenta tanto os casos de uso incluídos, como as extensões sublinhados e a numeração de passos indica exatamente onde esses casos de uso são inseridos, onde o impacto ocorre. A combinação de dígitos e letras é utilizada para indicar o passo onde a condição dos fluxos alternativos e pontos de extensão se aplicam.

Além disso, o *EAI template* inclui as seções *pré-condições, pós-condições e requisitos especiais* do *template* do RUP (COCKBURN, 2001), pois nessas seções podem ser identificados RNFs candidatos a aspectos. Uma adaptação apresentada no *EAI template* é a numeração utilizada nessas seções. As pré-condições, pós-condições e os requisitos especiais são listados um a um com o objetivo de manter uma ordem dentro da lista, evitando possíveis conflitos quanto à ordem de execução. Outra adaptação é a descrição da lista de pontos de extensão. A estrutura de cada ponto de extensão é a referência ao passo alterado, seguida do nome do caso de uso, seguida do `se` e a descrição da condição.

A tabela 3.3 mostra uma comparação dos *templates* citados, indicando com um 'X' as seções e características incluídas em cada um deles. A numeração de passos nas seções *fluxo básico e fluxos alternativos* no *template* do RUP é opcional.

Tabela 3.3. Comparação dos *templates* de *Cockburn*, do RUP e do *EAI Template*

Seções/ Características	***Template* de Cockburn**	***Template* do RUP**	***EAI Template***
Nome	X	X	X
Contexto	X	-	-
Escopo	X	-	-
Descrição	X	X	X
Atores	X	X	X
Stakeholders	X	-	-
Pré-condições	X	X	X
Garantias mínimas	X	-	-
Fluxo principal	X	X	X
Fluxos alternativos	-	X	X
Pontos de extensão	X	X	X
Pós-condições	-	X	X
Requisitos especiais	-	X	X
Outras Informações	X	-	-
Numeração de passos nas seções *fluxo básico* e *fluxos alternativos*	X	opcional	X
Numeração de passos nas seções *pré-condições*, *pós-condições* e *requisitos especiais.*	-	-	X
Casos de uso de inclusão sublinhados.	X	-	X
Casos de uso de extensão sublinhados	-	-	X

Fonte: Elaborada pelo autor.

A descrição de todas as seções incluídas no *template* segue abaixo:

- **Nome:** nome do caso de uso;
- **Descrição:** descreve a finalidade do caso de uso;
- **Atores:** lista os atores envolvidos no caso de uso;
- **Pré-condições:** descrevem o que o sistema deve garantir como verdadeiro antes que o caso de uso seja iniciado;
- **Fluxo principal:** descreve as interações entre os atores e o sistema necessárias para alcançar o objetivo como uma seqüência de passos.
- **Fluxos alternativos:** descreve as interações entre os atores e os sistemas para cada fluxo alternativo. Possui uma condição inicial e deve indicar o passo onde essa condição se aplica.
- **Pontos de extensão:** descrevem todos os pontos do fluxo principal onde o comportamento de outro caso de uso (de extensão) pode ser adicionado sob uma condição;
- **Pós-condições:** descrevem o que o sistema deve assegurar no final do caso de uso;
- **Requisitos especiais:** descrevem os requisitos do caso de uso que não são abordados no fluxo principal. Geralmente são RNFs relacionados ao caso de uso.

No quadro 3.5 é apresentado um exemplo de utilização do *EAI template*. O caso de uso Reservar quarto, apresentado no quadro 3.1, foi adequado ao formato proposto no *EAI template*.

Além de auxiliar a busca por candidatos a aspectos, a utilização desse *template* facilita a legibilidade e descrição de casos de uso pelo usuário. Utilizando o *EAI template*, o usuário não tem que se preocupar com as seções que devem ser obrigatoriamente descritas e está, inconscientemente, separando os candidatos a aspectos. O *template* serve como um guia, assegurando a estruturação correta do documento. A vantagem de utilizar esse *template* em relação a outros da literatura é que ele inclui de fato todas as seções em que candidatos a aspectos já foram identificados em descrições de casos de uso e utiliza a numeração de passos. A numeração de passos indica pontos no fluxo de execução em que os candidatos a aspectos serão inseridos, ou seja, pontos que podem ser implementados como pontos de junção (*joinpoints*) no fluxo de execução do programa base utilizando POA.

Nome: Reservar quarto.

Descrição: Esse caso de uso descreve como um cliente reserva um quarto.

Atores: Cliente

Pré-condições

1. O cliente foi autenticado pelo sistema.

Fluxo básico

1. O cliente seleciona reservar quarto.
2. O sistema mostra os tipos de quartos existentes no hotel.
3. O cliente Verifica despesa total da estadia.
4. O cliente faz a reserva para o quarto escolhido.
5. O sistema deduz do banco de dados o número de quartos do tipo escolhido disponíveis para reserva.
6. O sistema verifica que não existe uma reserva idêntica.
7. O sistema cria uma nova reserva.
8. O sistema mostra o número de confirmação da reserva e instruções para realizar *check in* e o caso de uso termina.

Fluxos alternativos

6a. Existe uma reserva idêntica no sistema.

6a.1 O sistema mostra a reserva existente e o caso de uso termina.

Pontos de extensão

5a. Tratar lista de espera se não existem quartos disponíveis do tipo escolhido.

Pós-condições

1. Um novo registro de reserva foi criado.
2. O número de quartos disponíveis para os dados especificados é decrementado.

Requisitos especiais

1. O sistema deve tratar 5 reservas concorrentes.
2. Cada reserva não deve demorar mais do que 20 segundos.

Fonte: Elaborada pelo autor.

Quadro 3.5. Caso de uso `Reservar quarto` utilizando o *EAI template*.

3.2.2 Validar casos de uso

A utilização do *EAI template* para descrever casos de uso não garante que ele será utilizado da forma correta. Mesmo utilizando esse *template* para descrever casos de uso, os usuários podem deixar de preencher alguma(as) seções da descrição, podem omitir alguma informação, ou preencher alguma informação errada.

Para melhorar as descrições de casos de uso do usuário e tentar obter completude e correção, foi projetado um *checklist*. O *checklist* contém questões que buscam guiar o usuário de forma que tanto os RNFs quanto os RFs sejam descritos corretamente. Esse *checklist* possibilita identificar erros, omissões e oportunidades de acrescentar informações na descrição. Possibilita também identificar oportunidades de reúso através de relacionamentos *include* e *extend* nas descrições. Como explicado anteriormente casos de uso incluídos e extensões de casos de uso são candidatos a aspectos. Logo, essas informações são importantes.

O *checklist* foi projetado tendo como base as diretrizes encontradas na literatura para descrição de casos de uso (COCKBURN, 2001; AMBLER, 2004). Por exemplo, há uma questão no *checklist* que verifica se sentenças “se... então” foram utilizadas no fluxo principal da descrição, detectando a ocorrência de um fluxo alternativo (COCKBURN, 2001). Se sentenças “se...então” aparecem no fluxo principal é necessário reescrevê-las. Freqüentemente, na sentença anterior a ela, existe uma verificação. Essa verificação deve ser seguida da condição sob a qual o fluxo básico continua sem interrupção, descrevendo o cenário bem sucedido. O cenário de falha deve ser descrito como fluxo alternativo.

A estrutura da descrição também foi levada em consideração no projeto do *checklist* para facilitar a busca posterior de candidatos a aspectos. Por exemplo, é verificado se todas as condições necessárias para realização do caso de uso são descritas como pré-condições, onde RNFs de segurança (candidatos a aspectos) costumam ser identificados.

3.2.3 Identificar candidatos a aspectos

As descrições de casos de uso devem ser analisadas em busca de RNFs e RFs candidatos a aspectos. A utilização do *EAI template* proposto para descrição de casos de uso auxilia uma inspeção posterior em busca de candidatos a aspectos. Isso ocorre

porque as seções da descrição nas quais candidatos a aspectos costumam ser encontrados são conhecidas.

RNFs candidatos a aspectos poderão ser encontrados nas seções *pré-condições, pós-condições e requisitos especiais*, buscando nessas seções palavras ou expressões relacionadas a algum RNF, ou seja, palavras que podem ser encontradas em operacionalizações de RNFs como apresentado na subseção 3.1.1.

Para facilitar a identificação dessas palavras foi criado neste trabalho um catálogo de indicativos de RNFs. Esse catálogo foi gerado a partir do conhecimento dos tipos de RNFs disponíveis em CHUNG *et al.* (2000) e do conhecimento obtido durante a análise de descrições de casos de uso encontradas na literatura. Esse catálogo inclui tipos e subtipos de RNFs associados com as palavras relacionadas a eles, conforme apresentado na tabela 3.4. Foram criadas duas versões desse catálogo, em português e em inglês. Se uma das palavras ou expressões da coluna "Palavras relacionadas", ou uma de suas variações ou sinônimos aparecem no texto da descrição do caso de uso, e seu significado no contexto da frase está relacionada ao tipo de RNF que ele representa, esse RNF é um candidato a aspecto.

Tabela 3.4. Parte do catálogo de indicativos de RNFs gerado.

Tipos de RNFs	**Subtipos de RNFs**	**Palavras relacionadas**
Desempenho	Tempo de resposta	tempo de resposta, a tempo, dentro do prazo, em tempo real, atraso, demora, segundo, minuto.
	Rendimento/Transferência	atraso, demora, sobrecarga.
Persistência	-	registrar, gravar, marcar, criar, gerar, adicionar, armazenar, atualizar, status, consultar, recuperar, retornar, salvar, deletar, apagar, backup, cópia de segurança, banco de dados, base de dados, dados.
Segurança	Tratamento de exceções	disponível, falha, defeito, erro, exceção, recuperação.
	Confidencialidade	autorizar, não autorizar, autenticar, logar, permitir, validar, verificar, permissão, restringir, limitar, senha, ameaça, vulnerabilidade, ataque, invasão,...

Fonte: Elaborada pelo autor.

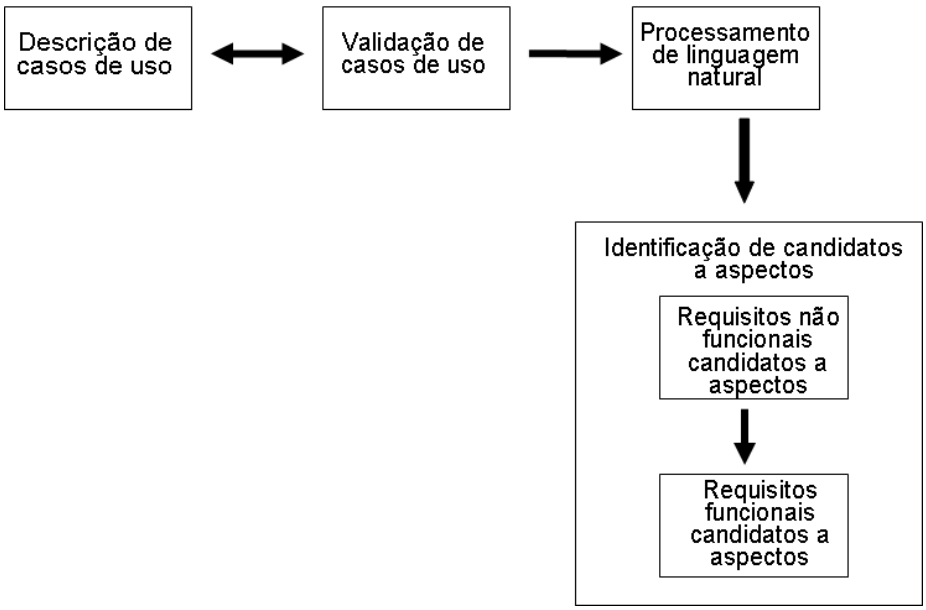
Mesmo que os RNFs identificados utilizando os indicativos da Tabela 3.4, não apareçam em múltiplas descrições de casos de uso, eles podem ser implementados utilizando POA, pensando na facilidade de evolução do sistema. A facilidade na evolução do sistema é um dos benefícios obtidos com a POA. Quando um novo módulo incluir o mesmo RNF, o aspecto existente pode modificar o mesmo, evitando retrabalho.

Além disso, cada *fluxo alternativo* descrevendo uma variação ou exceção indica um RNF candidato a aspecto.

Os RFs candidatos a aspectos são encontrados de forma explícita em casos de uso incluídos e extensões de casos de uso.

3.3 Ferramenta de apoio

Como subproduto deste trabalho, está em desenvolvimento uma ferramenta para automatizar a identificação de requisitos implementáveis por orientação a aspectos em descrições de casos de uso. A ferramenta é implementada como um *plugin* para o ambiente integrado de desenvolvimento (*Integrated Development Environment - IDE*) Eclipse. A plataforma Eclipse foi escolhida por ser uma IDE popular e para aproveitar as funcionalidades oferecidas por ela. A figura 3.2 ilustra um modelo composto por módulos inter-relacionados necessários ao *plugin*. Os módulos são descritos nos próximos tópicos.



Fonte: Elaborada pelo autor.

Figura 3.2. Diagrama de blocos do *plugin*

3.3.1 Módulo de descrição de casos de uso

Um *plugin* para automatizar a extração de candidatos a aspectos em descrições de casos de uso deve possuir um módulo para que os usuários possam descrever todos os casos de uso do sistema. O módulo de descrição de casos de uso deve permitir ao usuário descrever os casos de uso do sistema no formato do *template* proposto para identificação de candidatos, EAI *template*, evitando que o usuário tenha que se preocupar com as seções que devem ser descritas e assegurando que as principais seções onde candidatos a aspectos costumam ser identificados sejam descritas.

Esse módulo foi desenvolvido estendendo o editor de textos padrão da plataforma de desenvolvimento *Eclipse.* Os *plugins* responsáveis pela edição de texto no *Eclipse* são baseados no *framework JFaces*. *JFace* é um conjunto de ferramentas padrão para desenvolvimento de componentes para interface com o usuário em aplicações (JFACE, 2009).

Por enquanto, a descrição de casos de uso deve ser realizada em inglês, pois a ferramenta *Lingpipe* utilizada para processamento de linguagem natural (LINGPIPE, 2009) ainda não foi estendida para identificar palavras em português.

3.3.2 Módulo de validação de casos de uso

O objetivo desse módulo é auxiliar o usuário com a estruturação do texto, tentando manter a estrutura correta para facilitar a análise posterior. Ele deve possibilitar a identificação de erros, omissões e oportunidades de acrescentar informações na descrição e induzir o usuário a tentar reparar esses erros.

Esse módulo ainda não foi implementado. Ele será implementado como uma interface cooperativa para verificação dos casos de uso durante a descrição, a partir do *checklist* proposto apresentado na seção 3.2.2. O usuário receberá instruções sobre como o *template* deverá ser preenchido e se alguma informação preenchida deve ser alterada. Por exemplo, será informado ao usuário durante a descrição se ele deixar de preencher alguma seção importante do *template,* ou se ele inserir na descrição alguma expressão não desejada, como a expressão “se... então”, como explicado na seção 3.2.2.

3.3.3 Módulo de processamento de linguagem natural

O módulo de processamento de linguagem natural tem como objetivo analisar cada sentença da descrição de casos de uso atribuindo a cada palavra sua função gramatical (artigo, substantivo, adjetivo, verbo, etc). Essa técnica de análise de texto, que determina a função gramatical para cada palavra é conhecida como *part-of-speech (POS) tagging* (LINGPIPE, 2009). O resultado da técnica de POS é utilizado posteriormente no módulo de identificação de candidatos a aspectos.

Para realizar o mecanismo de POS foi utilizado o *framework LingPipe*, um conjunto de bibliotecas Java para apoio ao processamento de linguagem natural (LINGPIPE, 2009). *LingPipe* possui várias ferramentas de extração de informação e mineração de dados, dentre elas: identificação de relações entre entidades e ações; classificação de textos por idioma, gênero, tema, etc; identificação da função gramatical de palavras.

Esse módulo produz um arquivo de texto com a descrição de casos de uso e as *tags* resultantes da classificação das palavras. Por exemplo, cada unidade léxica da sentença "*The customer is logged*", recebe uma *tag* que define o seu valor sintático (Ex: "*The/at*", indicando que "*The*" é um artigo, cuja *tag* correspondente é "*at*"; "*customer/nn*", indicando que "*customer"* é um substantivo no singular, cuja *tag* correspondente é "*nn*"; e assim sucessivamente).

3.3.4 Módulo de identificação de candidatos a aspectos

O módulo de identificação de candidatos a aspectos tem como objetivo identificar os RFs e os RNFs candidatos a aspectos. Os *includes* e *extends* são classificados como RFs candidatos a aspectos. Os RNFs poderão ser encontrados buscando palavras relacionadas a algum RNF, ou seja, palavras que estão listadas no catálogo de indicativos de RNFs.

Esse módulo utiliza uma ontologia para identificar essas palavras relacionadas a algum RNF nas descrições de casos de uso. Uma ontologia representa um conjunto de conceitos e os relacionamentos entre eles dentro de um domínio. A ferramenta *LingPipe* (LINGPIPE, 2009) possui uma ontologia chamada *Brown Corpus* (BROWN CORPUS, 2009) para processamento de linguagem natural em inglês. Essa ontologia foi estendida com as palavras listadas na versão em inglês do catálogo de indicativos de RNFs para que elas possam ser identificadas como RNFs

candidatos a aspectos. Futuramente, serão incluídas na ontologia as palavras listadas na versão em português do catálogo.

O arquivo texto produzido pelo módulo de processamento de linguagem natural é analisado novamente nesse módulo. Os candidatos a aspectos identificados são marcados com a *tag* “as”. Algumas palavras classificadas no módulo de processamento de linguagem natural como artigo e verbos auxiliares (como ser e estar) não precisam ser comparados com as palavras do catálogo de indicativos de RNFs. Por enquanto, o resultado apresentado ao usuário é o arquivo texto produzido com as *tags* resultantes da classificação das palavras.

Novas palavras identificadas relacionadas a RNFs candidatos a aspectos podem ser incluídas na ontologia . O arquivo final produzido é utilizado no aperfeiçoamento e treinamento de ontologias baseado na localização da *tag* “as” e de derivações da palavra marcada com essa *tag*, através da avaliação de N-Gramas (LINGPIPE, 2009; SILVA; CHAPETTA, 2003).

4 ESTUDOS DE CASO

Foram realizados dois estudos de caso. Inicialmente foi realizado um estudo de caso de utilização do *template* e *checklist* propostos. O objetivo desse primeiro estudo foi avaliar a facilidade de compreensão e utilização desses documentos e identificar candidatos a aspectos incluídos nas descrições de casos de uso realizadas pelos participantes. No segundo estudo de caso, foi realizada a aplicação da técnica com o objetivo de mostrar que a sua utilização auxilia a inspeção em busca de candidatos a aspectos nas descrições de casos de uso.

Os estudos de caso foram realizados em um laboratório onde cada computador possuía uma pasta com os documentos necessários. Além do *EAI template* e *checklist* proposto, um documento contendo as instruções para realização do estudo de caso foi disponibilizado aos participantes (documento disponível no Apêndice A). Para cada estudo de caso foi elaborado um “plano de avaliação", um documento contendo detalhes de como a avaliação seria realizada.

Os estudos de caso são detalhados nas próximas seções. Os questionários aplicados aos participantes dos estudos estão disponíveis no Apêndice B.

4.1 Estudo de caso 1

A primeira experiência acadêmica foi realizada com 19 alunos voluntários que possuíam conhecimentos básicos sobre a técnica de casos de uso da UML. Esses alunos cursavam no semestre 2008/II a disciplina Engenharia de *Software* II, oferecida no sexto período do Bacharelado em Ciência da Computação da Universidade Federal de Viçosa.

4.1.1 Objetivos

A avaliação foi realizada para verificar se o *template* para descrição de casos de uso (*EAI template*) e o *checklist* proposto satisfazem seus objetivos, ou seja, se o

template facilita a legibilidade e descrição de casos de uso pelo usuário e se o *checklist* auxiliou o usuário a identificar erros, omissões e oportunidades de acrescentar informações na descrição. Com essa avaliação pretendia-se responder as seguintes questões:

1. Os usuários compreendem esses documentos igualmente bem?
2. É fácil para o usuário aprender a usar os documentos?
3. Os usuários se sentem satisfeitos com o nível de detalhamento exigido no *EAI template* para descrição de casos de uso?
4. A utilização do *EAI template* facilita a descrição e leitura dos casos de uso?
5. O *checklist* ajudou a identificar erros e omissões, acrescentar informações e identificar oportunidades de reúso através de relacionamentos *include* e *extend* nas descrições de casos de uso?

Além disso, as descrições de casos de uso feitas pelos participantes da avaliação foram utilizadas para analisar quais RNFs foram incluídos nas descrições de casos de uso e em quais seções da descrição.

4.1.2 Metodologia

A avaliação foi realizada da seguinte forma:

1. **Contato inicial:** Através de uma conversa informal com os participantes antes da avaliação, foram esclarecidos os objetivos e os detalhes de como a avaliação seria realizada. Foram transmitidas explicações gerais sobre utilização do *template* e *checklist*. Houve a preocupação de esclarecer aos participantes que os documentos é que seriam avaliados e não eles e que as tarefas deveriam ser executadas de forma bastante confortável.
2. **Descrição do problema:** Os participantes receberam a seguinte descrição de sistema: "*Sistema de controle de videolocadora para informatizar as funções de empréstimo e devolução de fitas. O objetivo do sistema é agilizar o processo de empréstimo e garantir maior segurança, ao mesmo tempo possibilitar maior controle das informações por parte da gerência. O sistema deverá calcular automaticamente o valor dos pagamentos a serem efetuados em cada empréstimo, inclusive multas e descontos devidos. A cada devolução de fitas corresponderá um pagamento, não sendo possível trabalhar com sistema de créditos. A impossibilidade de efetuar um pagamento deve deixar*

o cliente suspenso, ou seja, impossibilitado de tomar emprestadas novas fitas até saldar a dívida." (WAZLAWIC, 2004).

Em seguida, esclareceram dúvidas relacionadas a esse sistema.

3. **Descrição de casos de uso utilizando o *EAI template*:** A partir da descrição recebida, foi solicitada aos participantes a descrição dos casos de uso `Emprestar Fitas` e `Devolver Fitas` utilizando o *EAI template*. Esses casos de uso foram escolhidos porque são os dois casos de uso principais para o desenvolvimento da aplicação proposta. A partir deles, outros casos de uso poderão ser incluídos ou o comportamento deles poderá ser estendido com o comportamento de outros casos de uso.
4. **Verificação dos casos de uso utilizando o *checklist*:** Foi solicitado aos participantes a utilização do *checklist* para fazer as modificações necessárias nas descrições de casos de uso realizadas no passo anterior.
5. **Aplicação do questionário:** Depois de completadas essas tarefas, foi solicitado aos participantes o preenchimento de um questionário para avaliar o *template* e *checklist*.

4.1.3 Questionário

O questionário foi elaborado com o objetivo de coletar informações para avaliar a utilização do *template* e *checklist*. A partir da definição dos objetivos da avaliação, foram formuladas perguntas que buscam obter todas as informações desejadas. A análise das respostas ao questionário deve responder às questões levantadas na seção 4.1.1. O questionário aplicado aos participantes do estudo de caso 1 é apresentado no Apêndice B.1.

4.1.4 Análise das descrições de casos de uso realizadas pelos participantes

Os casos de uso descritos pelos participantes foram analisados em busca de RFs e RNFs candidatos a aspectos. Dos 52 casos de uso descritos, 14 eram casos de uso de inclusão ou extensão, representando RFs candidatos a aspectos.

Para identificar RNFs foram localizadas palavras que representam operacionalizações de RNFs. A tabela 4.1 apresenta os tipos de RNFs identificados nas descrições realizadas pelos participantes e as seções das descrições em que esses RNFs foram identificados. Apresenta também algumas palavras que identificaram o

RNF. Entre parênteses, após a palavra, aparece o número de casos de uso em que ela foi identificada como operacionalização do RNF indicado.

Somente em uma descrição de caso de uso apareceu uma variação no *fluxo alternativo*. Todos os outros *fluxos alterna*tivos descreviam tratamento de exceções. Após a análise das descrições realizadas pelos participantes, foi feita a atualização do catálogo de indicativos de RNFs com novas palavras identificadas que podem aparecer em operacionalizações de RNFs.

Tabela 4.1 . RNFs identificados nas descrições de casos de uso do Estudo de Caso 1

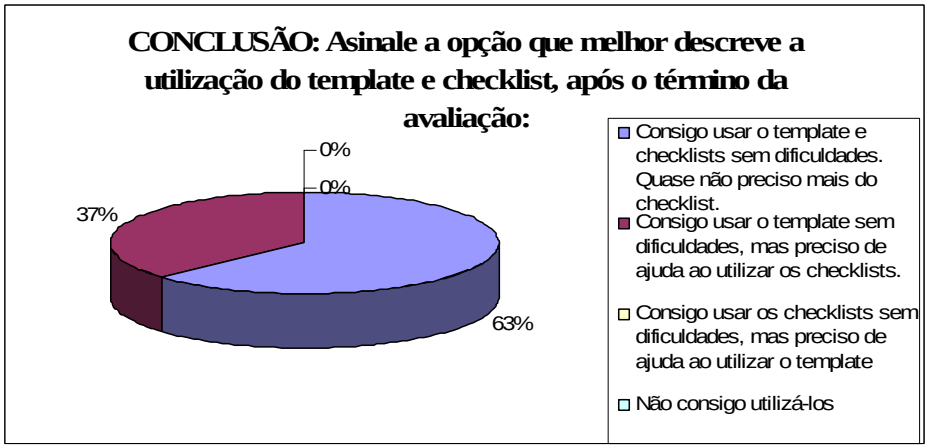
Seção da descrição	Tipo de RNF	Subtipo	Palavras
Pré-condições	segurança	confidencialidade	Autenticado (20), *login* (3), verificar (3)
Fluxo principal	persistência	-	banco de dados (4), cria (3), atualiza (2), registra (5), marca (2), adiciona (2), *status* (1)
Fluxo alternativo	segurança	tratamento de exceção	-
Pós-condições	persistência	-	Criado (15), associa (1), atualizada (2), marcada (8), gerado (2), adicionado (2), status (3)
Requisitos especiais	concorrência	-	Concorrentes (4)
Requisitos especiais	desempenho	tempo de resposta	Segundos (28), minutos (5)

Fonte: Elaborada pelo autor.

4.1.5 Análise dos questionários

Analisando os questionários respondidos pelos participantes, foi observado que eles compreenderam a utilização do *template*, mas sentiram dificuldades ao utilizar o *checklist*. 37% dos participantes disseram que precisam de ajuda ao utilizar

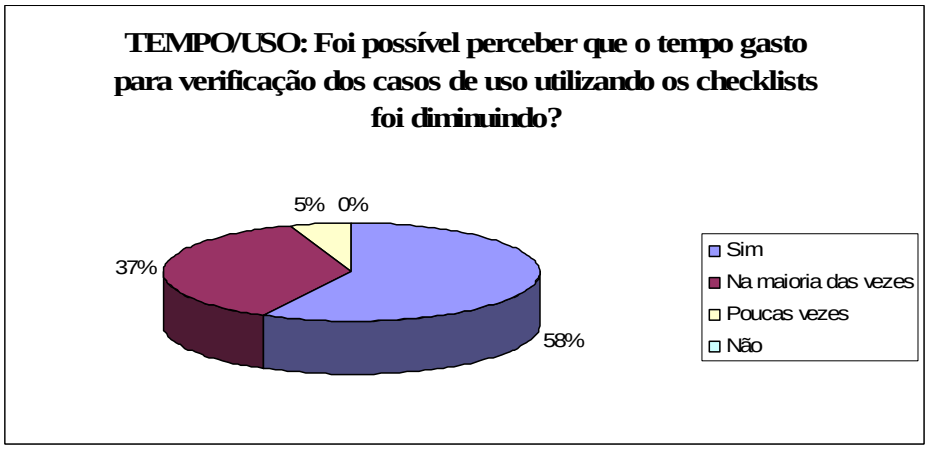
o *checklist* após a avaliação como apresentado no gráfico da figura 4.1. Isso indica a necessidade de revisar o *checklist*, tentando melhorar as perguntas e incluir mais exemplos.



Fonte: Elaborada pelo autor.

Figura 4.1. Resultados sobre a utilização dos documentos após a avaliação.

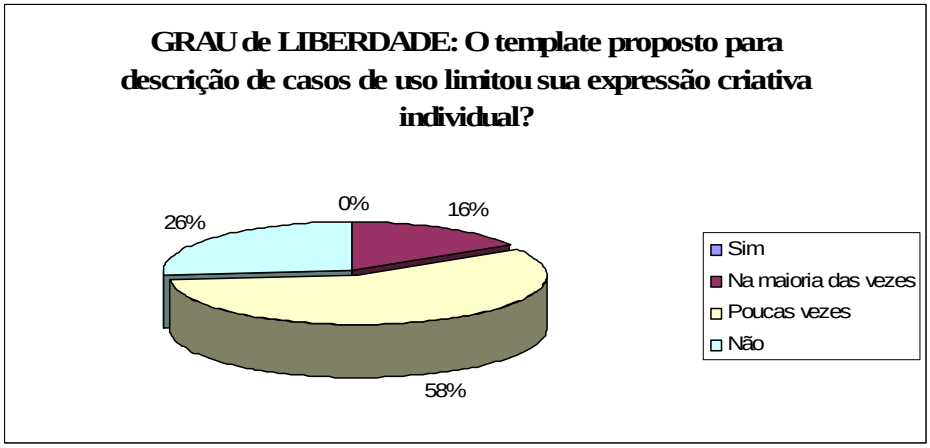
Apesar disso, é possível concluir que é fácil para o usuário aprender a usar o *template*, pois 68% dos participantes não precisaram de ajuda na primeira vez que o utilizaram e após a avaliação, 63% dos participantes disseram que conseguem utilizar *template* e *checklist* sem dificuldades e que quase não sentem necessidade do *checklist* para verificar os casos de uso. Dos 19 participantes, 58% perceberam que o tempo gasto para verificação dos casos de uso foi diminuindo, ou seja, que com o tempo eles aprendem as boas práticas para descrição de casos de uso e não será mais necessário utilizar o *checklist* para verificação. 37% perceberam isso na maioria das vezes e 5% disseram que poucas vezes foi possível perceber.



Fonte: Elaborada pelo autor.

Figura 4.2. Resultados sobre o tempo gasto para verificação dos casos de uso.

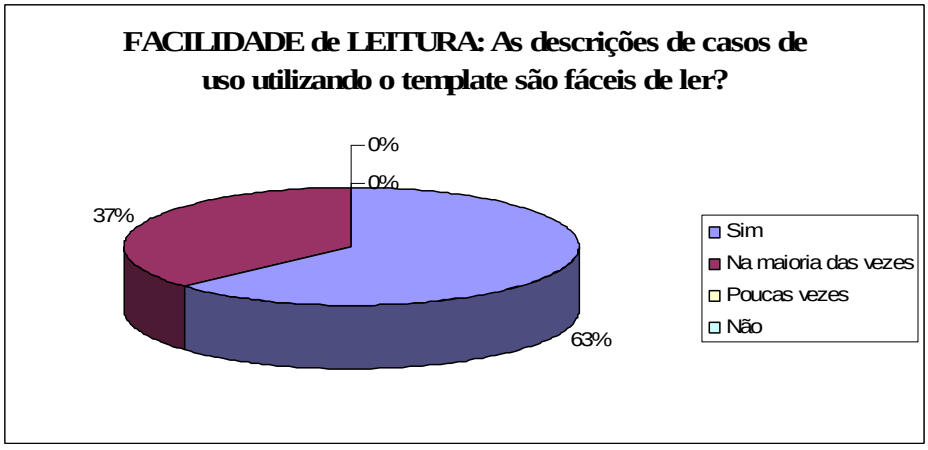
Dos 19 participantes, 26% afirmaram que o *template* proposto não limitou sua expressão criativa individual e 58% afirmaram que poucas vezes isso ocorreu. 16% (3 participantes) afirmaram que na maioria das vezes o *template* limitou sua expressão criativa individual.



Fonte: Elaborada pelo autor.

Figura 4.3. Resultados sobre o nível de detalhes exigido no template.

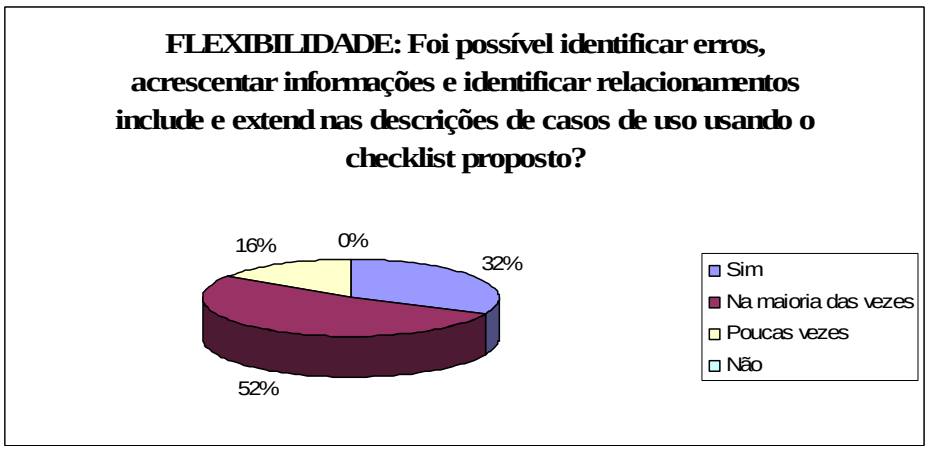
Pode-se concluir que a utilização do *template* facilita a descrição e leitura dos casos de uso. Dos alunos que participaram da experiência utilizando o *template*, 53% consideraram fácil e os outros 47% disseram que na maioria das vezes foi fácil escrever casos de uso utilizando o *template*; 63% dos participantes consideraram fácil a leitura dos casos de uso utilizando o *template* e 37% disseram que na maioria das vezes foi fácil.



Fonte: Elaborada pelo autor.

Figura 4.4. Resultados sobre a facilidade de leitura das descrições de casos de uso utilizando o *template*.

A utilização do *checklist* auxiliou os participantes a identificar erros e acrescentar informações nas descrições como apresentam os dados da figura 4.5.



Fonte: Elaborada pelo autor.

Figura 4.5. Resultados sobre a possibilidade de melhorar a descrição utilizando o *checklist.*

Nenhum participante demonstrou insatisfação ao utilizar os documentos. Todos os participantes disseram estar satisfeitos (74%) ou muito satisfeitos (26%) com a utilização dos documentos.

Após a análise desse estudo de caso, foram realizadas algumas modificações no *template* e *checklist*, atendendo sugestões dos participantes. São descritas abaixo algumas das alterações realizadas:

- Modificação das instruções de preenchimento da seção *requisitos especiais* no *template*, incluindo exemplos de RNFs.
- Inclusão de exemplos em algumas questões do *checklist*.
- Modificação das instruções de preenchimento de algumas questões do *checklist*, incluindo mais detalhes.

No Apêndice A são apresentadas as versões finais dos documentos (*template* e *checklist)*, incluindo todas as alterações realizadas durante a pesquisa.

4.2 Estudo de caso 2

O segundo estudo de caso foi realizado com 7 alunos voluntários que cursavam no semestre 2009/I a disciplina Engenharia de *Software* oferecida no Programa de Pós-Graduação (Mestrado) em Ciência da Computação da Universidade Federal de Viçosa. Esse foi um grupo de formação heterogênea, com formação e origens distintas, que entraram no mestrado em Ciência da Computação do DPI no

ano de 2008. Esses alunos possuíam conhecimentos básicos sobre a técnica de casos de uso da UML.

Esse estudo de caso foi realizado no mesmo ambiente de avaliação do estudo de caso 1, no mesmo laboratório. Os documentos disponibilizados aos participantes, o problema apresentado e os casos de uso descritos também foram os mesmos do estudo de caso 1. As diferenças existentes no procedimento seguido neste estudo de caso são apresentadas na subseção 4.2.2.

4.2.1 Objetivos

Essa avaliação foi realizada para verificar se a utilização da técnica apresentada neste trabalho auxilia a inspeção em busca de candidatos a aspectos em descrições de casos de uso. Como na primeira avaliação, também foram coletadas informações para avaliar se o *template* e *checklist* satisfazem seus objetivos. Com essa avaliação pretendia-se responder, além das questões apresentadas na seção 4.1.1, as seguintes questões:

1. Utilizando a técnica proposta, os candidatos a aspectos foram encontrados mais facilmente?
2. Qual a quantidade de candidatos a aspectos foram identificados pelos participantes antes e após conhecer os resultados da análise de descrições de casos de uso em busca de candidatos a aspectos (padrões identificados), apresentados no capítulo 3 deste trabalho?

4.2.2 Metodologia

A avaliação foi realizada em duas etapas. Os passos realizados na primeira etapa correspondem aos passos apresentados na seção 4.1.2 do estudo de caso 1. Porém, o questionário aplicado ao final dessa etapa foi modificado para coletar informações a respeito do conhecimento dos participantes sobre POA e sobre os candidatos a aspectos que puderam identificar antes de conhecer os resultados da pesquisa.

A segunda etapa foi realizada da seguinte forma:

1. **Apresentação de resultados deste trabalho:** Os participantes receberam um treinamento com o objetivo de apresentar os resultados desta dissertação, os padrões identificados na análise de descrições de casos de uso em busca de candidatos a aspectos.

2. **Aplicação do questionário 2:** Após o treinamento, foi solicitado aos participantes o preenchimento de outro questionário com o objetivo de coletar informações sobre os candidatos a aspectos identificados pelos participantes após o conhecimento dos resultados da dissertação e sobre a facilidade de identificação dos candidatos a aspectos.

4.2.3 Questionários

Os questionários aplicados aos participantes desse estudo de caso também foram elaborados a partir da definição dos objetivos da avaliação. O questionário aplicado aos participantes na primeira etapa da avaliação é apresentado no Apêndice B.2 e o questionário aplicado na segunda etapa é apresentado no Apêndice B.3.

4.2.4 Análise das descrições de casos de uso realizadas pelos participantes

Dos 19 casos de uso descritos, 5 eram casos de uso de inclusão ou extensão, representando RFs candidatos a aspectos.

Para identificar RNFs foram localizadas palavras que representam operacionalizações de RNFs. A tabela 4.2 apresenta os tipos de RNFs identificados, as seções das descrições em que esses RNFs foram identificados e as palavras que identificaram o RNF. Entre parênteses, após a palavra, aparece o número de casos de uso em que ela foi identificada como operacionalização do RNF indicado.

É interessante observar que não foi identificada nenhuma palavra relacionada ao RNF de `persistência` no fluxo principal como aconteceu no primeiro estudo de caso. Também foi realizada a atualização do catálogo de indicativos de RNFs com novas palavras identificadas que podem aparecer em operacionalizações de RNFs.

4.2.5 Análise dos questionários

Analisando os questionários respondidos pelos participantes, foi observado que eles também sentiram dificuldades ao utilizar o *checklist.* A maioria dos participantes, equivalente a 57%, disseram que precisam de ajuda ao utilizar o *checklist* após a avaliação.

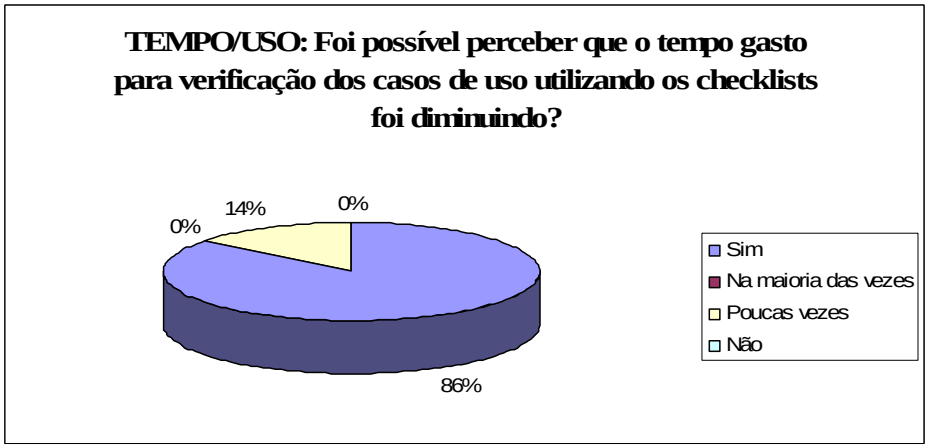
Como no primeiro estudo de caso, é possível concluir que é fácil para o usuário aprender a usar o *template*, pois 71% dos participantes não precisaram de ajuda na primeira vez que o utilizaram. Dos participantes, 86% perceberam que o

tempo gasto para verificação dos casos de uso foi diminuindo. Somente 1 participante disse que poucas vezes foi possível perceber.

Tabela 4.2 . RNFs identificados nas descrições de casos de uso do Estudo de Caso 2

Seção da descrição	Tipo de RNF	Subtipo	Palavras
Pré-condições	segurança	confidencialidade	Autenticado (6), *login* (2), cadastrado (3)
Fluxo alternativo	segurança	tratamento de exceção	-
Pós-condições	persistência	-	Criado (2), associa (1), atualizada (4), adicionado (2),
Requisitos especiais	concorrência	-	Concorrentes (2)
Requisitos especiais	desempenho	tempo de resposta	Segundos (2), minutos (3)

Fonte: Elaborada pelo autor.

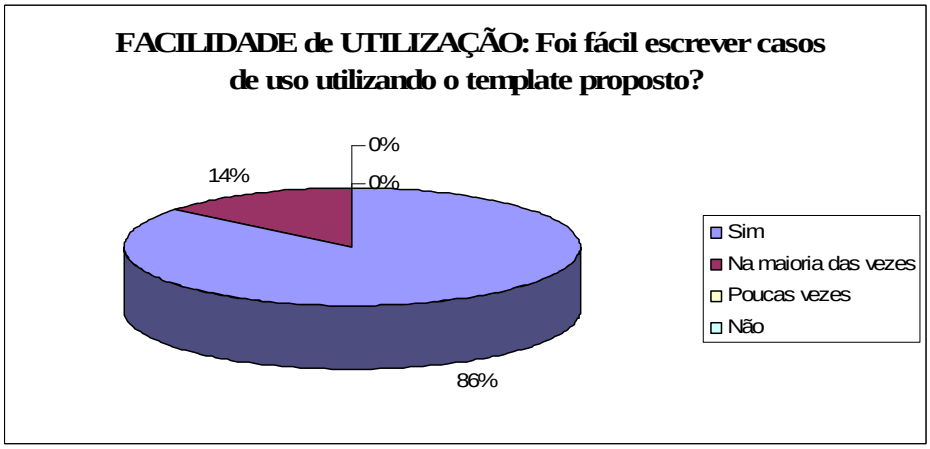


Fonte: Elaborada pelo autor.

Figura 4.6. Resultados sobre o tempo gasto para verificação dos casos de uso no segundo estudo de caso.

Dos 7 participantes, 43% afirmaram que o *template* proposto não limitou sua expressão criativa individual e 43% afirmaram que poucas vezes isso ocorreu. Somente 1 participante afirmou que na maioria das vezes o *template* limitou sua expressão criativa individual.

Pode-se concluir que a utilização do *template* facilita a descrição e leitura dos casos de uso. Dos alunos que participaram do estudo de caso, 86% consideraram fácil escrever casos de uso utilizando o *EAI template*; 71% dos participantes consideraram fácil a leitura dos casos de uso utilizando o *template* e 29% disseram que na maioria das vezes foi fácil.



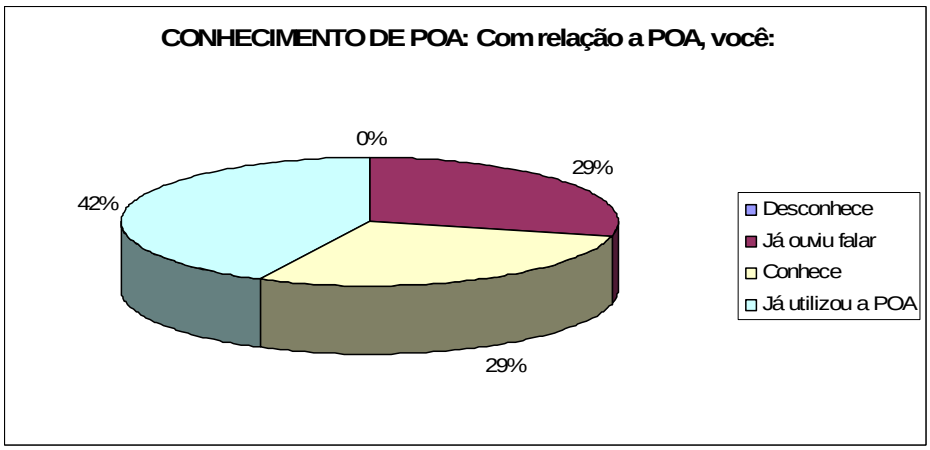
Fonte: Elaborada pelo autor.

Figura 4.7. Resultados sobre a facilidade de utilização do *template*.

O *checklist* auxiliou os participantes na verificação da descrição de casos de uso. Somente 1 participante disse que poucas vezes foi possível identificar erros, omissões e acrescentar informações utilizando o *checklist*.

Assim como no primeiro estudo de caso, nenhum participante demonstrou insatisfação. Todos os participantes disseram estar satisfeitos (57%) ou muito satisfeitos (43%) com a utilização dos documentos.

Como apresentado na figura 4.8, nenhum participante desconhecia completamente a POA. Ao contrário, a maioria conhecia ou até mesmo já havia utilizado a POA. Os alunos que já haviam utilizado a POA identificaram 1 ou 2 candidatos a aspectos antes do treinamento realizado na segunda etapa da avaliação. Um dos alunos que afirmou conhecer a POA identificou 2 candidatos a aspecto. Os alunos que somente ouviram falar sobre POA não responderam a questão 10 do questionário, ou seja, não identificaram nenhum candidato a aspecto.



Fonte: Elaborada pelo autor.

Figura 4.8. Resultados sobre o conhecimento de POA.

Após o treinamento realizado na segunda etapa da avaliação, todos os participantes identificaram pelo menos mais um candidato a aspecto, como apresentado na tabela 4.3. Antes de conhecer os resultados da dissertação, foram identificados no total 5 candidatos a aspectos diferentes pelos participantes. Após conhecer os resultados da dissertação, foram identificados no total 10 candidatos a aspectos diferentes pelos participantes.

A tabela 4.4 apresenta os tipos de candidatos a aspectos identificados pelos participantes antes e depois do treinamento para identificar candidatos a aspectos em descrições de casos de uso.

Tabela 4.3 . Resultados sobre a quantidade de candidatos a aspectos identificados antes e após o treinamento da segunda etapa da avaliação.

Conhecimento de POA dos participantes	Candidatos a aspectos identificados antes do treinamento	Candidatos a aspectos identificados após o treinamento
Já utilizou a POA	1	5
	1	2
	2	3
Conhece POA	2	4
	0	3
Já ouviu falar de POA	0	4
	0	1

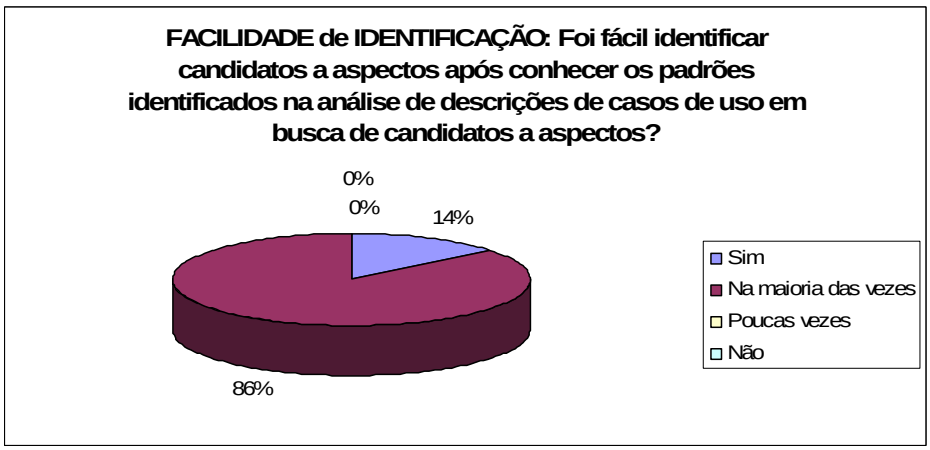
Fonte: Elaborada pelo autor.

Tabela 4.4. Resultados sobre os tipos de candidatos a aspectos identificados antes e após o treinamento da segunda etapa da avaliação.

Conhecimento de POA	Tipos de candidatos a aspectos identificados antes do treinamento	Tipos de candidatos a aspectos identificados após o treinamento
Já utilizou a POA	Segurança (confidencialidade, tratamento de exceção).	Segurança (confidencialidade, tratamento de exceção), desempenho (tempo de resposta), casos de uso de inclusão e extensão.
Conhece POA	Persistência, caso de uso de inclusão.	Segurança (confidencialidade, tratamento de exceção), concorrência, persistência, caso de uso de inclusão.
Já ouviu falar de POA	-	Segurança (confidencialidade, tratamento de exceção), persistência, concorrência, caso de uso de inclusão.

Fonte: Elaborada pelo autor.

Pode-se concluir que os candidatos a aspectos foram encontrados mais facilmente utilizando a técnica proposta. 14% dos participantes disseram que foi fácil e os outros 86% disseram que na maioria das vezes foi fácil identificar candidatos a aspectos após conhecer os padrões identificados na análise de descrições de casos de uso em busca de candidatos a aspectos.

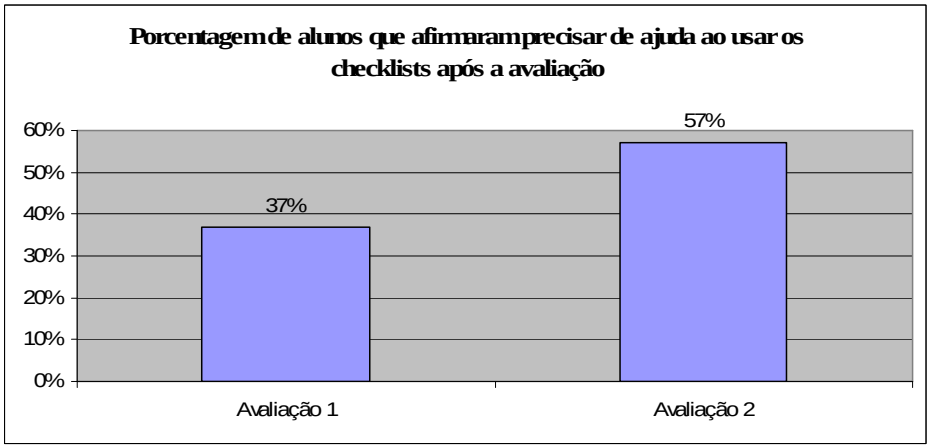


Fonte: Elaborada pelo autor.

Figura 4.9. Resultados sobre a facilidade de identificação de candidatos a aspectos utilizando a técnica proposta.

4.3 Conclusões e comparações entre os estudos de casos

Como apresentado nas seções anteriores, os participantes das avaliações tiveram dificuldades para aprender a usar o *checklist*. A porcentagem de alunos que afirmaram precisar de ajuda ao usar o *checklist* em cada avaliação é apresentada na figura 4.10. Pode-se observar, analisando essa figura, que o nível de maturidade dos participantes não influenciou na dificuldade de aprendizado do *checklist.* Os participantes do segundo estudo de caso, alunos da pós-graduação (com maior nível de maturidade), apresentaram uma dificuldade maior do que os alunos da graduação (participantes do primeiro estudo de caso). Como explicado na seção 4.2, os alunos da pós-graduação possuem uma formação mais heterogênea. Esses alunos possuem conhecimentos sobre a técnica de casos de uso da UML que foram obtidos a partir de livros e técnicas distintas. Os alunos da graduação possuem conhecimento mais homogêneo e mais recente em sala de aula, do que os alunos da pós-graduação.



Fonte: Elaborada pelo autor.

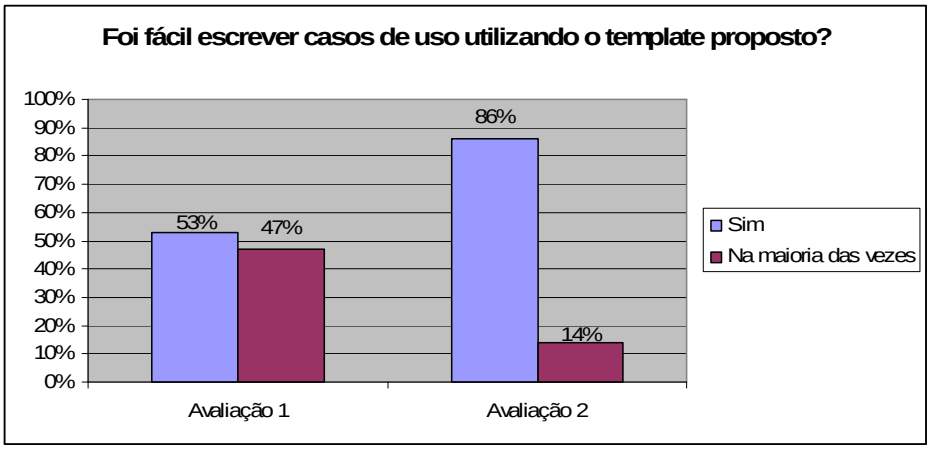
Figura 4.10. Porcentagem de participantes com dificuldades para aprender a usar o *checklist*.

Isso indica que deve ser feita uma avaliação mais detalhada para identificar quais as dificuldades dos participantes na utilização do *checklist*. Apesar disso, na ferramenta implementada para identificação de candidatos a aspectos, a verificação da descrição de caso de uso será feita durante a descrição através de uma interface cooperativa para auxiliar o usuário com a estruturação do texto. O *checklist* não será utilizado diretamente pelos usuários.

O nível de detalhes que devem ser incluídos na descrição de casos de uso utilizando o *template* não foi um problema para a maioria dos participantes. No

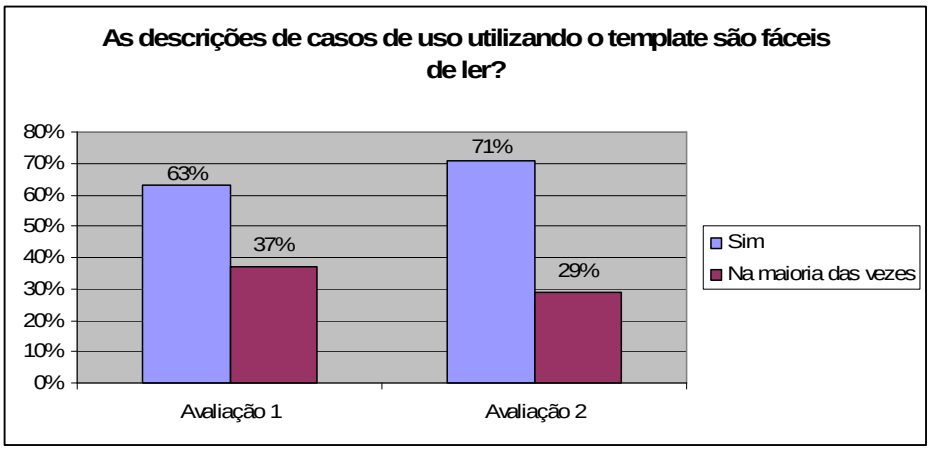
primeiro estudo de caso, 3 participantes afirmaram que na maioria das vezes o *template* limitou sua expressão criativa individual. Somente 1 participante afirmou o mesmo no segundo estudo de caso. É interessante observar que 26% dos participantes do primeiro estudo de caso afirmaram que o *template* proposto não limitou sua expressão criativa individual, contra 43% dos participantes do segundo estudo de caso. Para os participantes do primeiro grupo, alunos da graduação (menor nível de maturidade), foi mais difícil seguir as instruções do *template* e *checklist*. É mais difícil para eles seguir regras ou qualquer processo disciplinado do que para os alunos da pós-graduação, participantes do segundo grupo. O mesmo pode ser observado ao comparar a porcentagem de participantes do grupo 1 e do grupo 2 que afirmaram ser fácil escrever casos de uso utilizando o *template* proposto (figura 4.11). Os participantes do grupo 2 acharam mais fácil escrever casos de uso utilizando o *template* do que os participantes do primeiro grupo.

O *template* proposto facilita a descrição e leitura de casos de uso, como apresentado nas figuras 4.11 e 4.12.



Fonte: Elaborada pelo autor.

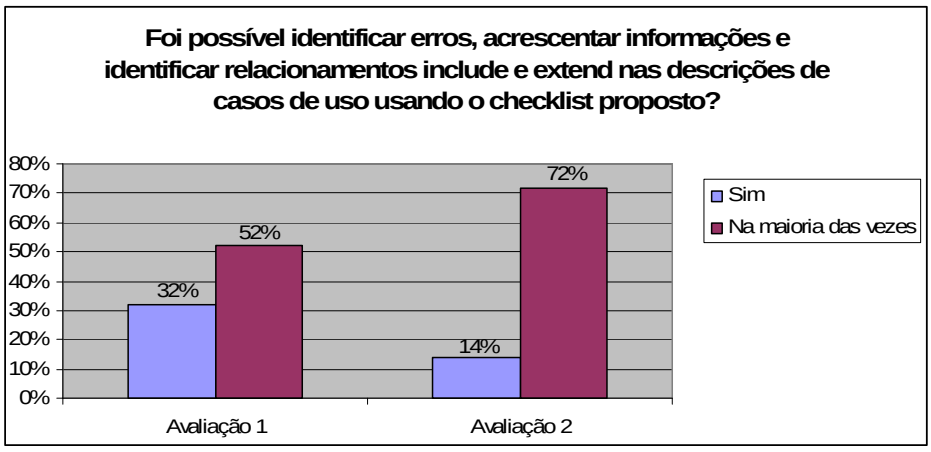
Figura 4.11. Porcentagem de participantes que consideram fácil escrever casos de uso utilizando o *template*.



Fonte: Elaborada pelo autor.

Figura 4.12. Porcentagem de participantes que consideram fácil ler casos de uso utilizando *o template.*

É possível concluir que a utilização do *checklist* tornou a descrição de casos de uso menos sujeita a erros. Observe a porcentagem de participantes que identificaram erros, omissões e oportunidades de acrescentar informações nas descrições de caso de uso utilizando o *checklist* na figura 4.13.



Fonte: Elaborada pelo autor.

Figura 4.13. Porcentagem de participantes que identificaram possibilidades de melhorar as descrições usando o *checklist*.

Como apresentado na figura 4.9, pode-se concluir que os candidatos a aspectos foram encontrados mais facilmente utilizando a técnica proposta. O número de candidatos a aspectos diferentes identificados pelos participantes dobrou após o conhecimento dos resultados desta pesquisa. Analisando a tabela 4.3, pode-se dizer que após conhecer os resultados da dissertação, o conhecimento de POA não influenciou a quantidade de aspectos identificados em cada grupo. Por exemplo, um aluno que já havia utilizado POA identificou 5 candidatos a aspectos após o

treinamento, sendo que havia identificado 1 antes do treinamento; e um aluno que somente ouviu falar sobre POA identificou 4 candidatos a aspectos, sendo que não havia identificado candidatos a aspectos antes do treinamento. Outros 2 alunos, um que já havia utilizado e outro que somente ouviu falar sobre POA, identificaram somente mais um candidato a aspecto.

Os dados da tabela 4.3 foram analisados utilizando o Teste T para dados pareados. O Teste T é um teste estatístico simples que pode ser utilizado quando existem duas amostras pequenas e iguais (STEEL *et al*., 1996) como a obtida quando os mesmos indivíduos buscaram candidatos a aspectos utilizando a técnica proposta e o método intuitivo (tabela 4.3). Esse teste confirmou que as diferenças apresentadas na tabela 4.3 não foram ao acaso, que o número de candidatos a aspectos identificados após o treinamento foi maior. A probabilidade destas diferenças terem sido ao acaso é de no máximo 0,0047, ou seja, pelo Teste T para dados pareados declaramos a 0,47% (p-value = 0,0047) que as diferenças observadas são significativas. Pode-se afirmar ainda que o grau de confiabilidade da hipótese de que a técnica proposta possibilitou encontrar mais aspectos é de 95%.

Além disso, analisando a tabela 4.4, pode-se concluir que o conhecimento de POA influenciou no tipo de candidatos a aspectos identificados. Alunos que já haviam utilizado POA identificaram RNFs relacionados ao `desempenho` do sistema, enquanto alunos que afirmaram conhecer e alunos que somente ouviram falar sobre a POA identificaram RNFs relacionados à `concorrência` e `persistência` do sistema.

5 CONCLUSÕES E TRABALHOS FUTUROS

A partir da pesquisa realizada, foi proposta uma técnica para auxiliar a identificação de candidatos a aspectos na fase de requisitos. Para isso, foi proposto um *template* para descrição de casos de uso e um *checklist* para verificação da descrição. O *template* foi projetado para incluir as seções nas quais candidatos a aspectos foram identificados, utilizar a numeração de passos que indica exatamente onde os candidatos a aspectos devem ser inseridos e dar ênfase aos casos de uso incluídos e extensões na descrição, pois esses são candidatos a aspectos. Dentre os trabalhos investigados, não foi encontrado um *template* para descrição de casos de uso com essas características em conjunto. A utilização da estrutura da descrição para identificar candidatos a aspectos é a principal diferença entre a abordagem apresentada neste trabalho e outras encontradas na literatura (BANIASSAD; CLARKE, 2004a; BANIASSAD; CLARKE, 2004b; CHITCHYAN *et al*., 2006a; CHITCHYAN *et al*., 2006b). Assim, o objetivo geral proposto na seção 1.2 foi alcançado.

A maior parte dos objetivos específicos foi alcançada. Através de revisão bibliográfica foi possível aprofundar os conhecimentos sobre os casos de uso da UML e POA, investigar como identificar candidatos a aspectos nos requisitos e metodologias e ferramentas já utilizadas para identificação de aspectos nos requisitos. O estudo de casos de uso da UML possibilitou conhecer as principais diferenças entre as descrições encontradas na literatura e identificar as alterações necessárias nas descrições para facilitar a identificação de candidatos a aspectos. O conhecimento de boas práticas para descrição de casos de uso facilitou a elaboração do *checklist* proposto. Aprofundar os conhecimentos acerca da POA foi essencial para verificar a possibilidade de implementar os candidatos a aspectos identificados. Conhecer metodologias e ferramentas utilizadas para identificação de aspectos nos requisitos contribuiu para obtenção da técnica proposta. Tudo isso foi discutido no referencial teórico, apresentado no capítulo 2.

Os padrões relacionados aos candidatos a aspectos identificados e às seções da descrição em que eles foram identificados são descritos no capítulo 3, conforme objetivo específico proposto. Esses padrões foram identificados durante a análise de descrições de casos de uso presentes na literatura.

Dois estudos de caso foram realizados com o objetivo de avaliar os documentos propostos (*EAI template e checklist*) e validar a técnica obtida. Os participantes dessas experiências afirmaram que a utilização do *template* tornou mais fácil a descrição e a leitura dos casos de uso. Conforme esperado, foram identificados RFs e RNFs candidatos a aspectos nessas descrições apresentados no capítulo 4 desta dissertação. A utilização do *EAI template* para descrição de casos de uso e o conhecimento das seções das descrições em que candidatos a aspectos costumam ser identificados reduz o esforço necessário na busca de candidatos a aspectos nos requisitos. Os resultados dos estudos de caso mostraram que a técnica obtida facilita a inspeção em busca de candidatos a aspectos e que essa técnica pode ser utilizada independente de uma ferramenta que a suporte.

Foi iniciado o desenvolvimento da ferramenta proposta na seção 3.3 para automatizar a identificação de candidatos a aspectos. A ferramenta está sendo implementada como um *plugin* para o ambiente integrado de desenvolvimento Eclipse. Dos módulos propostos, somente o módulo de validação de casos de uso ainda não foi implementado. Os outros foram implementados como projeto final de curso de um aluno de graduação em Ciência da Computação da Universidade Federal de Viçosa.

5.1 Principais contribuições

As principais contribuições disponibilizadas por este trabalho foram:

a) Desenvolvimento de um *template* para descrição de casos de uso e *checklist* para verificação de casos de uso que podem ser utilizados com outras finalidades, diferentes da apresentada neste trabalho.
b) Técnica proposta para identificar candidatos a aspectos em descrições de casos de uso que pode ser utilizada por qualquer usuário, conforme apresentado nos estudos de caso.

c) Catálogo de indicativos de RNFs que inclui tipos e subtipos de RNFs e palavras relacionadas a eles, palavras que podem identificar esse RNF.
d) Identificação das seções na descrição de casos de uso onde candidatos a aspectos costumam ser identificados e dos tipos de RNFs que costumam ser identificados em cada seção.

Acredita-se que os resultados são relevantes, visto que apresentam uma solução diferenciada para auxiliar a identificação de candidatos a aspectos em descrições de casos de uso, independente de uma ferramenta, como apresentado nos estudos de caso do capítulo 4.

5.2 Trabalhos futuros

A seguir, são apresentadas algumas sugestões de trabalhos futuros:

a) **Realização de outros estudos de caso**. Podem ser realizadas avaliações para verificar quais as dificuldades encontradas pelos participantes ao utilizar o *checklist* proposto, avaliações com problemas mais complexos e menos conhecidos do que o sistema de videolocadora e avaliações com alunos que não possuem conhecimento algum sobre POA, alunos que nunca ouviram falar sobre a POA. Estudos de caso também podem ser realizados para comparar a utilização da técnica proposta com a utilização de outras metodologias existentes para identificar candidatos a aspectos. Além disso, pode ser realizado um estudo de caso com dois grupos diferentes (um utilizando a técnica proposta e o outro não) analisando as mesmas descrições de casos de uso em busca de candidatos a aspectos.
b) **Finalização do *plugin* proposto.** Como apresentado anteriormente, a implementação da ferramenta proposta não foi concluída. Todos os módulos especificados devem ser implementados para facilitar a aplicação da técnica. Futuramente, os recursos do *plugin* também podem ser expandidos. A ferramenta necessita de uma interface gráfica amigável, pois ainda está sendo executada dentro do ambiente eclipse. É necessário implementar também um módulo para facilitar a visualização de candidatos a aspectos.
c) **Extensão da técnica proposta**. A técnica proposta pode ser estendida para incluir, por exemplo, indicações de como implementar os candidatos a aspectos identificados utilizando OA. Podem ser elaborados guias para

auxiliar o usuário a mapear o tipo de candidato a aspecto identificado na descrição de casos de uso para o projeto OA.

d) **Avaliação de utilização do *template* proposto para análise de pontos de casos de uso.** Acredita-se que o *template* proposto neste trabalho pode facilitar a análise de pontos de casos de uso. Pontos de Caso de Uso é uma técnica para estimar esforço e prazo associados ao desenvolvimento de *software* no início do projeto com base no modelo de casos de uso (ANDRADE, 2004) . Nesta técnica, o nível de detalhes incluídos nas descrições de casos de uso influencia na qualidade final da medição. Uma das desvantagens da utilização de pontos por casos de uso são as diferenças de estimativa realizadas por analistas diferentes devido à variação na escrita de casos de uso. O *EAI template* pode ser utilizado para aplicação da técnica, garantindo que as regras utilizadas para descrição e validação dos casos de uso seriam as mesmas.

APÊNDICE A

A.1 *EAI Template* para descrição de casos de uso

O *EAI Template (Template for Early Aspects Identification*) é baseado nos *templates* de COCKBURN e do RUP descritos em (COCKBURN, 2001).

Orientação para uso do *template*:
Em cada seção da descrição de casos de uso existem instruções de preenchimento em cor azul e entre tags ("<" e ">"). Após ler a instrução de preenchimento e compreendê-la, substitua-a por texto de sua autoria, relacionado à seção.

Nome: <Dê um nome para o caso de uso. Use uma frase verbal curta (verbo + substantivo) expressando a intenção do usuário, o objetivo do caso de uso. >

Descrição: <Descreva a finalidade do caso de uso.>

Atores: <Liste os atores envolvidos no caso de uso. Um ator é uma pessoa, organização ou sistema externo que representa um papel em uma ou mais interações com o sistema. O ator é quem inicia o caso de uso.>

Pré-condições: <Liste cada pré-condição do caso de uso. Uma pré-condição é uma declaração simples que descreve o que o sistema deve garantir como verdadeiro antes que o caso de uso seja iniciado. Indica o estado da aplicação antes do início do caso de uso. A descrição da pré-condição deve ser uma sentença completa preferivelmente no tempo passado. Cada pré-condição possui a estrutura:

<nº. da pré-condição><descrição da pré-condição>

Fluxo principal: <Descreva de forma clara as interações entre os atores e o sistema necessárias para alcançar o objetivo. As interações devem ser descritas como uma seqüência de passos, desde o passo que inicia o caso de uso até o passo em que o objetivo é alcançado. Certifique-se de usar os mesmos termos que foram utilizados nos outros casos de uso se o significado pretendido é o mesmo. Quando este caso de uso incluir um outro, escreva o ator que iniciará o caso de uso seguido do nome do caso de uso incluído. Sublinhe o nome do caso de uso incluído. Na descrição da ação, use sentenças completas no tempo presente com um verbo de ação na voz ativa. Cada passo possui a estrutura:

<nº. do passo><descrição da ação>

A estrutura da sentença que representa a descrição da ação deve ser simples:
<sujeito>...<verbo>...<objeto direto>...<frase preposicional>.

Quando alguns passos do fluxo principal podem ser repetidos, escreva os passos que serão repetidos após a condição para que a repetição ocorra. A repetição possui a estrutura:

Enquanto <descrição da condição> faça
<nº. do passo><descrição da ação>

Fluxos alternativos: <Descreva as interações entre os atores e o sistema como uma seqüência de passos para cada fluxo alternativo deste caso de uso. Um fluxo alternativo descreve um comportamento opcional ou excepcional em relação ao comportamento normal do caso de uso. Descrevem variações, erros potenciais, erros na lógica de negócios e exceções. Um fluxo alternativo possui uma condição inicial que precisa ser atendida para que esse fluxo inicie e deve indicar o passo onde essa condição se aplica. A referência a esse passo usa a combinação de dígitos e letras. O dígito é uma referência ao passo alterado no fluxo principal e a letra é usada para distinguir fluxos alternativos que alteram o mesmo passo no fluxo principal (a letra 'a' representa o primeiro fluxo alternativo para um determinado passo, a letra 'b' o segundo e assim sucessivamente). A condição inicial para cada fluxo alternativo possui a estrutura:

<referência ao passo alterado> <descrição da condição>

A seqüência de passos do fluxo alternativo também usa a combinação de dígitos e letras. Cada passo possui a estrutura:

<referência ao passo alterado>.<nº. do passo><descrição da ação>

Pontos de extensão: <Liste todos os pontos de extensão deste caso de uso. Um ponto de extensão corresponde a um passo do fluxo principal onde o comportamento de outro caso de uso pode ser adicionado. A seqüência de passos do caso de uso de extensão é inserida no caso de uso base sob uma condição. A referência ao passo alterado usa combinação de dígitos e letras como no fluxo principal. O nome do caso de uso de extensão deve ser sublinhado. A estrutura de cada ponto de extensão é:

<referência ao passo alterado><nome do caso de uso de extensão> se
<descrição da condição>

Pós-condições: <Liste todas as pós-condições do caso de uso. Uma pós-condição é uma declaração simples que descreve o que o sistema deve assegurar imediatamente após o término da execução bem-sucedida do caso de uso. Na escrita de pós-condições, use uma sentença completa preferivelmente no tempo passado. Cada pós-condição possui a estrutura:

<nº. da pós-condição><descrição da pós-condição>

Requisitos especiais: <Liste todos os requisitos do caso de uso que não são abordados pelo fluxo principal. Geralmente são requisitos não- funcionais (desempenho, concorrência, confiabilidade, usabilidade, segurança, etc) relacionados ao caso de uso. >

<nº. do requisito especial><descrição do requisito especial>

A.2 *CheckList* para melhoria da descrição de casos de uso

Orientação para uso do *template*:

As questões de 1 a 12 devem ser utilizadas para verificar cada caso de uso. As questões 13 e 14 devem ser utilizadas após a descrição de todos os casos de uso do sistema. Todas as perguntas devem gerar uma resposta sim. Para cada resposta negativa, siga as instruções em azul que seguem a questão para alterar a descrição de seu caso de uso.

1. As seções nome, descrição, atores e fluxo principal do caso de uso foram preenchidas?

   Essas seções não podem ficar em branco. Não deixe de preenchê-las adequadamente.

2. Login, autorização, permissão e verificações necessárias para realização do caso de uso são descritas como pré-condições?

   Condições necessárias para realização do caso de uso devem ser descritas como pré-condições.

   Exemplo:

   Antes:

   Fluxo principal:

   1. O estudante entra com nome e número de matrícula.
   2. O sistema verifica as informações do estudante e se ele possui os pré-requisitos para realizar a matrícula na disciplina.

   Depois:

   Pré-Condições:

   1. O estudante foi autenticado pelo sistema.
   2. Foi verificado que o estudante possui os pré-requisitos necessários.

3. Casos de uso que serão executados para estabelecer uma pré-condição foram descritos?

   Se para estabelecer uma pré-condição for necessário outro caso de uso, certifique-se de descrevê-lo.

   Exemplo:

   Pré-condição:

   1. O estudante foi autenticado pelo sistema.

   É necessário descrever o caso de uso: Autenticar estudante.

4. O ator está visível em cada passo do fluxo principal?

Em cada passo do fluxo principal, deixe explícito quem é o ator que realiza o passo, quem é o sujeito da sentença.

5. Existem ações executadas pelo ator e ações executadas pelo sistema na descrição do fluxo principal?

   Não descreva o caso de uso do ponto de vista do ator ou do ponto de vista do sistema. Tanto as ações executadas pelo ator, quanto as ações executadas pelo sistema, devem estar claramente descritas.

6. Os dados que passam em uma única direção foram coletados em apenas um passo no fluxo principal?

   Não devemos usar passos separados para coletar cada item de dado de um mesmo ator no fluxo principal.

   Exemplo:

   Antes:

   1. O sistema solicita o nome do usuário.
   2. O usuário entra com o nome.
   3. O sistema solicita o endereço.
   4. O usuário entra com o endereço...

   Depois:

   1. O sistema solicita o nome e endereço do usuário.
   2. O usuário entra com o nome e endereço.

7. Sentenças Se...Então (If... Then) não foram utilizadas no fluxo principal?

   Se sentenças Se... Então aparecem no fluxo principal é necessário reescrevê-las. Freqüentemente, as pessoas escrevem que o sistema verifica uma condição. Quando encontrar a expressão Se... Então, olhe a sentença anterior a ela. Provavelmente ela usa o termo “verifica” ou então existe uma verificação implícita. Substitua essa sentença (explícita ou implícita) utilizando o termo “valida” seguida da condição sob a qual o fluxo básico continua sem interrupção e faça a segunda sem o Se...Então. Dessa forma é descrito o cenário bem sucedido. Descreva o cenário de falha como fluxo alternativo.

   Exemplo:

   Antes:

   2. O sistema verifica se a senha está correta.
   3. Se sim, então o sistema apresenta as ações disponíveis para o usuário.

   Depois:

2. O sistema valida que a senha está correta.

3. O sistema apresenta as ações disponíveis para o usuário.

Fluxos alternativos:

2a. Senha incorreta.

2a.1 O sistema informa ao usuário que a senha está incorreta.

2a.2 O sistema retorna ao fluxo principal no passo 1.

8. Para cada passo onde aparecem os termos “verifica” e “valida” foi descrito um fluxo alternativo?

Após a descrição do fluxo principal do caso de uso é necessário observar o que poderia dar errado em cada passo descrito e escrever esses outros cenários em que ocorre alguma exceção como fluxos alternativos. Verifique se para cada passo onde aparecem os termos “verifica” e “valida” foi descrito um fluxo alternativo que prevê o caso de falha, ou seja, a não verificação ou validação da condição.

9. Ao final do fluxo alternativo está explicitada a ação a ser adotada?

Após a descrição do fluxo alternativo é necessário explicitar se o caso de uso termina ou se o fluxo principal é retomado. Se o fluxo principal será retomado, escreva onde (em que passo) ele será retomado.

Exemplo:

Fluxos alternativos:

2a. Senha incorreta.

2a.1 O sistema informa ao usuário que a senha está incorreta.

2a.2 O sistema retorna ao fluxo principal no passo 1.

10. Fluxos alternativos com nível de complexidade semelhante ao seu fluxo principal foram descritos em um caso de uso de extensão?

O fluxo alternativo pode ser simplesmente o tratamento de um erro onde o usuário é informado que o erro acontece. Mas pode também ser uma seqüência de ações mais complexas semelhante à seqüência de ações do fluxo principal. Observe se essa seqüência de ações pode ser descrita em um outro caso de uso, estendendo o comportamento do caso de uso em questão. Se possível, faça isso.

11. Fluxos que adicionam um novo comportamento ou funcionalidade foram descritos em um caso de uso de extensão?

Uma funcionalidade adicional ou um novo comportamento deve ser descrito como uma extensão de caso de uso para os casos de uso existentes.

12. Os requisitos não-funcionais (RNFs) aplicáveis foram considerados na descrição do caso de uso?

    RNFs são elementos específicos de projeto, muitas vezes sem relação direta com o problema em questão. São conhecidos como restrições ou atributos de qualidade de um sistema como usabilidade, desempenho, confiabilidade, segurança, concorrência, entre outros. Pode ser necessário estabelecer condições especiais como pré e pós-condições para satisfazer algum RNF.

    Orientações: As próximas questões devem ser respondidas após a descrição de todos os casos de uso do sistema.

13. Comportamentos comuns a vários casos de uso foram descritos em um caso de uso separado e incluído nos outros casos de uso que repetiam esse comportamento?

    Se existem comportamentos idênticos, ou seja, passos repetidos no fluxo principal em mais de um caso de uso, descreva-os em um caso de uso separado. O novo caso de uso deve ser incluído em todos os outros casos de uso que repetiam esse comportamento. Na descrição do fluxo principal, escreva o ator que iniciará o caso de uso seguido do nome do caso de uso incluído sublinhado. Isso implica que toda vez que o caso de uso é executado, o caso de uso incluído também é executado.

14. Todos os casos de uso incluídos e casos de uso de extensão apontados na descrição dos casos de uso (ou seja, os que estão sublinhados) foram descritos?

    Todos os casos de uso incluídos e de extensão que foram sublinhados na descrição dos casos de uso do sistema devem ser descritos utilizando o template de casos de uso e as orientações disponíveis no documento t emplate_use_cases.doc.

A.3 Instruções para realização do estudo de caso

1) Descrição do problema:
É proposto o desenvolvimento de um sistema de controle de videolocadora, que vai informatizar as funções de empréstimo e devolução de fitas. O objetivo do sistema é agilizar o processo de empréstimo e garantir maior segurança, ao mesmo tempo possibilitar maior controle das informações por parte da gerência. O sistema deverá calcular automaticamente o valor dos pagamentos a serem efetuados em cada empréstimo, inclusive multas e descontos devidos. A cada devolução de fitas corresponderá um pagamento, não sendo possível trabalhar com sistema de créditos. A impossibilidade de efetuar um pagamento deve deixar o cliente suspenso, ou seja, impossibilitado de tomar emprestadas novas fitas até saldar a dívida.

2) Descrição de casos de uso: Descreva os casos de uso *emprestar fitas* e *devolver fitas* utilizando o *template* de casos de uso e orientações disponíveis no documento *template_use_cases.doc* (documento apresentado no Apêndice A.1).

3) Validação/ Modificação de casos de uso: Utilize o *checklist* disponível no documento *checklist_use_cases.doc* (documento apresentado no Apêndice A.2) para fazer as modificações necessárias nas descrições de casos de uso do sistema.

APÊNDICE B

B.1 Questionário preenchido pelos participantes do estudo de caso 1

Prezado usuário, por favor responda o questionário a seguir para nos auxiliar a avaliar o *template* e *checklist* propostos. Suas sugestões na questão 10 serão importantes para o nosso estudo. Agradecemos sua colaboração.

1. NECESSIDADE DE AUXÍLIO: Qual das opções abaixo melhor descreve sua primeira experiência na utilização do *template* e *checklist*?
( ) Consegui usar o *template e checklist* sem dificuldades.
( ) Consegui usar o *template* sem dificuldades, mas precisei de ajuda ao utilizar o *checklist*.
( ) Consegui usar o *checklist* sem dificuldades, mas precisei de ajuda ao utilizar o *template*.
( ) Precisei de ajuda ao utilizar o *template e checklist*.

2. SATISFAÇÃO: Assinale sua satisfação quanto ao uso do *template* e *checklist*:
( )Muito satisfeito.
( )Satisfeito.
( )Insatisfeito.
( )Muito insatisfeito.

3. FACILIDADE de UTILIZAÇÃO: Foi fácil escrever casos de uso utilizando o *template* proposto?
( ) Sim.
( ) Na maioria das vezes.
( ) Poucas vezes.
( ) Não.

4. FACILIDADE de LEITURA: As descrições de casos de uso utilizando o *template* são fáceis de ler?
( ) Sim.
( ) Na maioria das vezes.
( ) Poucas vezes.
( ) Não.

5. GRAU de LIBERDADE: O *template* proposto para descrição de casos de uso limitou sua expressão criativa individual?
( ) Sim.
( ) Na maioria das vezes.
( ) Poucas vezes.
( ) Não.

6. FLEXIBILIDADE: Foi possível identificar erros, acrescentar informações e identificar relacionamentos *include* e *extend* nas descrições de casos de uso usando o *checklist* proposto?
( ) Sim.
( ) Na maioria das vezes.

( ) Poucas vezes.
( ) Não.

7. HELP: As orientações e exemplos do *template* e *checklist* foram claros e o auxiliaram de forma adequada?
( ) Sim.
( ) Na maioria das vezes.
( ) Poucas vezes.
( ) Não.

8. TEMPO/USO: Foi possível perceber que o número de respostas negativas foi diminuindo à medida que utilizou o *checklist*? Ou seja, o tempo gasto para verificação dos casos de uso utilizando o *checklist* foi diminuindo?
( ) Sim.
( ) Na maioria das vezes.
( ) Poucas vezes.
( ) Não.

9. CONCLUSÃO: Assinale a opção que melhor descreve a utilização do *template* e *checklist*, após o término da avaliação:
( ) Consigo usar o *template* e *checklist* sem dificuldades. Quase não preciso mais do *checklist*.
( ) Consigo usar o *template* sem dificuldades, mas preciso de ajuda ao utilizar o *checklist*.
( ) Consigo usar o *checklist* sem dificuldades, mas preciso de ajuda ao utilizar o *template*.
( ) Não consigo utilizá-los.

10. Sugestões.
_____
_____
_____
_____
_____
_____
_____
_____

B.2 Questionário preenchido pelos participantes do estudo de caso 2

Prezado usuário, por favor responda o questionário a seguir para nos auxiliar a avaliar o *template e checklist* propostos. Suas sugestões na questão 10 serão importantes para o nosso estudo. Agradecemos sua colaboração.

1. NECESSIDADE DE AUXÍLIO: Qual das opções abaixo melhor descreve sua primeira experiência na utilização do *template* e *checklist*?
( ) Consegui usar o *template e checklist* sem dificuldades.
( ) Consegui usar o *template* sem dificuldades, mas precisei de ajuda ao utilizar o *checklist*.
( ) Consegui usar o *checklist* sem dificuldades, mas precisei de ajuda ao utilizar o *template*.
( ) Precisei de ajuda ao utilizar o *template* e *checklist*.

2. SATISFAÇÃO: Assinale sua satisfação quanto ao uso do *template* e *checklist*:
( )Muito satisfeito.
( )Satisfeito.
( )Insatisfeito.
( )Muito insatisfeito.

3. FACILIDADE de UTILIZAÇÃO: Foi fácil escrever casos de uso utilizando o *template* proposto?
( ) Sim.
( ) Na maioria das vezes.
( ) Poucas vezes.
( ) Não.

4. FACILIDADE de LEITURA: As descrições de casos de uso utilizando o *template* são fáceis de ler?
( ) Sim.
( ) Na maioria das vezes.
( ) Poucas vezes.
( ) Não.

5. GRAU de LIBERDADE: O *template* proposto para descrição de casos de uso limitou sua expressão criativa individual?
( ) Sim.
( ) Na maioria das vezes.
( ) Poucas vezes.
( ) Não.

6. FLEXIBILIDADE: Foi possível identificar erros, acrescentar informações e identificar relacionamentos *include* e *extend* nas descrições de casos de uso usando o *checklist* proposto?
( ) Sim.

( ) Na maioria das vezes.
( ) Poucas vezes.
( ) Não.

7. HELP: As orientações e exemplos do *template* e *checklist* foram claros e o auxiliaram de forma adequada?
( ) Sim.
( ) Na maioria das vezes.
( ) Poucas vezes.
( ) Não.

8. TEMPO/USO: Foi possível perceber que o número de respostas negativas foi diminuindo à medida que utilizou o *checklist*? Ou seja, o tempo gasto para verificação dos casos de uso utilizando o *checklist* foi diminuindo?
( ) Sim.
( ) Na maioria das vezes.
( ) Poucas vezes.
( ) Não.

9. CONCLUSÃO: Assinale a opção que melhor descreve a utilização do *template* e *checklist*, após o término da avaliação :
( ) Consigo usar o *template* e *checklist* sem dificuldades. Quase não preciso mais do *checklist*.
( ) Consigo usar o *template* sem dificuldades, mas preciso de ajuda ao utilizar o *checklist*.
( ) Consigo usar o *checklist* sem dificuldades, mas preciso de ajuda ao utilizar o *template*.
( ) Não consigo utilizá-los.

10. CONHECIMENTO DE POA: Com relação à POA (Programação Orientada a Aspectos), você (marque apenas uma opção)?
( ) Desconhece.
( ) Já ouviu falar.
( ) Conhece.
( ) Já utilizou a POA.

Se conhece ou já utilizou a POA, consegue identificar algum requisito implementável por orientação a aspectos (candidato a aspecto) nos casos de uso *Emprestar fitas* e *Devolver fitas* que descreveu utilizando o *template* proposto? Qual(is)?

_____

_____

_____

11. Sugestões.

_____

_____

_____

B.3 Questionário preenchido na segunda etapa pelos participantes do estudo de caso 2

Prezado usuário, por favor, responda o questionário a seguir para nos auxiliar a avaliar a técnica proposta. Suas sugestões na questão 3 serão importantes para o nosso estudo. Agradecemos sua colaboração.

1. Após conhecer como extrair aspectos em casos de uso, qual (is) requisitos implementáveis por orientação a aspectos (candidatos a aspecto) consegue identificar nos casos de uso *Emprestar fitas* e *Devolver fitas* que descreveu utilizando o *template* proposto?

_____

_____

_____

_____

2. FACILIDADE de IDENTIFICAÇÃO: Foi fácil identificar candidatos a aspectos após conhecer os resultados da dissertação, ou seja, após conhecer os padrões identificados na análise de descrições de casos de uso em busca de candidatos a aspectos?
( ) Sim.
( ) Na maioria das vezes.
( ) Poucas vezes.
( ) Não.

3. Sugestões.

_____

_____

_____

_____

_____

REFERÊNCIAS BIBLIOGRÁFICAS

AMBLER, S. W. **The Object Primer: Agile Model-Driven Development with UML 2.0.** 3.ed. New York: Cambridge University Press, 2004. 572p.

ANDRADE, E. L. P.; OLIVEIRA, K. M. Uso Combinado de Análise de Pontos de Função e Casos de Uso na Gestão de Estimativa de Tamanho de Projetos de Software Orientado a Objetos. In: SIMPÓSIO BRASILEIRO DE QUALIDADE DE SOFTWARE, 3, 2004, Brasília. **Anais...** Brasília: 2004.

ARAÚJO, J., COUTINHO, P. Identifying Aspectual Use Cases Using a Viewpoint-Oriented Requirements Method. In: EARLY ASPECTS, 2003: Aspect-Oriented Requirements Engineering and Architecture Design Workshop, 2nd International Conference on Aspect Oriented Software Development, 2, 2003, Boston. **Proceedings…** Boston: 2003.

ARAÚJO, J., BANIASSAD, E., CLEMENTS, P., MOREIRA, A., RASHID, A., TEKINERDOGAN, B. Early Aspects: The Current Landscape. Technical Report COMP-001-2005, Lancaster University, Pittsburg, PA, 2005.

ASPECTC++. Disponível em: <http://www.aspectc.org/>. Acesso em: 22 fev. 2009.

ASPECTJ. Disponível em: <http://www.eclipse.org/aspectj>. Acesso em: 22 fev. 2009.

BANIASSAD, E., CLARKE, S. Finding aspects in requirements with theme/doc. In: Workshop on Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design, 2004a, Lancaster, UK. **Proceedings…** Lancaster, 2004a.

BANIASSAD, E., CLARKE, S. Theme: An Approach for Aspect-Oriented Analysis and Design. In: International Conference on Software Engineering - ICSE, 26, 2004b, Scotland, UK. **Proceedings**... Washington: IEEE CS Press, 2004b. p. 158-167.

BANIASSAD, E., CLEMENTS, P., ARAÚJO, J., RASHID, A., TEKINERDOGAM, B. Discovering Early Aspects. **IEEE Software**, v.23, n.1, p. 61-70, 2006.

BOEHM. B.; BASILI, V. R. Software Defect Reduction Top 10 List, **IEEE Computer**, v. 34, n. 1, p. 135-137, 2001.

BRITO, I.; MOREIRA, A.; ARAÚJO, J. A requirements model for quality attributes. In: Early Aspects 2002: Aspect-Oriented Requirements Engineering and Architecture Design Workshop, 1, 2002, Holanda. **Proceedings…** Holanda, 2002.

BROOKS, F. P. **The Mythical Man-Month: Essays on Software Engineering**. 20th Anniversary Edition. Boston: Addison-Wesley, 1995. 322 p.

CHITCHYAN, R.; SAMPAIO, A.; RASHID, A.; RAYSON, P. A Tool Suit for Aspect-Oriented Requirements Engineering. In: Workshop on Early Aspects: International Conference on Software Engineering, 2006a, Shanghai, China. **Proceedings…** New York: ACM, 2006a. p. 19-26.

CHITCHYAN, R.; SAMPAIO, A.; RASHID, A.; RAYSON, P. Evaluating EA-Miner: Are Early Aspect Mining Techniques Effective? In: Towards Evaluation of Aspect Mining (TEAM), 20, 2006b, Nantes, France. **Proceedings…** Nantes: Software Engineering Research Group, 2006.

CHUNG, L., NIXON, B., YU, E. MYLOPOULOS, J. **Non-Functional Requirements in Software Engineering**. Boston: Kluwer Academic Publishers, 2000. p. 476.

CHRISTEL, M. G.; KANG, K.C. Issues in Requirements Elicitation. Technical Report CMU/SEI-92-TR-12, Carnegie Mellon University, Pittsburgh, 1992. p.80.

COADY, Y.; KICZALES, G.; FEELEY, M.; SMOLYN G. Using AspectC to Improve the Modularity of Path-specific Customization in Operating System Code. In: European Software Engineering Conference Held Jointly With 9th ACM Sigsoft International Symposium on Foundations of Software Engineering, 8, 2001, Austria. **Proceedings…** Austria: ACM Press, 2001, p. 88-98.

COCKBURN, A. **Writing Effective Use Cases**. Boston: Addison-Wesley Professional, 2001. 304p.

DIJKSTRA, E. W. **A Discipline of Programming.** Englewood Cliffs, NJ: Prentice-Hall, 1976. 217p.

EARLY ASPECTS. **Early Aspects Web Site**. Disponível em: www.early-aspects.net. Acesso em: 09 mar.2009.

ELRAD, T.; KICZALES, G.; AKSIT, M.; LIEBERHER, K.; OSSHER, H. Discussing Aspects of AOP. **Communications of the ACM**. New York, NY, v.44, n.10, p.33-38, 2001.

ERIKSSON, H.E.; PENKER, M.; LYONS, B.; FADO, D. **UML2 Toolkit**. New York: Wiley, 2004. 552p.

FINKELSTEIN, A.; SOMMERVILLE, I. The Viewpoints FAQ. **BCS/IEE Software Engineering Journal**, Vol. 11, No. 1, 1996.

FOWLER, M. **UML Essencial: Um breve guia para a linguagem padrão de modelagem de objetos**. 3.ed. São Paulo: Bookman, 2005. 160p.

GLASS, R. L. **Facts and Fallacies of Software Engineering**. Boston, MA: Addison Wesley, 2003. 213p.

GRADECKI, J. D.; LESIECKI, N. **Mastering AspectJ: Aspect-Oriented Programing in Java**. 1.ed. Indianopolis, Indiana: Wiley, 2003. 456p.

IEEE – Institute of Electrical and Electronics Engineers. IEEE Guide to Software Requirements Specifications. ANSI/IEEE Standard 830- 1984, Institute of Electrical and Electronics Engineers, New York, 1984.

JACOBSON, I. Use cases and aspects – working seamlessly together. **Journal of Object Technology**, july 2003. v. 2, n. 4. p. 7-28.

JACOBSON, I.; CHRISTERSON, M.; JONSSON, P.; OVERGAARD, G. **Object-Oriented Software Engineering: A Use Case Driven Approach.** Wokingham: Addison-Wesley, 1993. 552p.

JACOBSON, I.; NG, P. W. **Aspect-Oriented Software Development with Use Cases**. New York: Addison-Wesley Professional, 2004. 464p.

JFACE. Disponível em: <http://wiki.eclipse.org/index.php/JFace>. Acesso em: 09 set. 2009.

JAVA. Disponível em: <http://java.sun.com>. Acesso em: 06 mar. 2009.

KICZALES, G.; LAMPING, J.; MENHDHEKAR, A.; MAEDA, C.; LOPES, C., V.; LOINGTIER, J.; IRWIN, J. Aspect-Oriented Programming. In: European Conference on Object-Oriented Programming (Ecoop'97), 11, 1997, Finland. **Proceedings…** Finland: Springer-Verlag, 1997. v.1241, p.220-242.

KICZALES, G.; HILSDALE, E.; HUGUNIN, J.; KERSTEN, M.; PALM, J.; GRISWOLD, W. G. An overview of aspectj. In: European Conference on Object- Oriented Programming (Ecoop'01), 15, 2001, Berlin. **Proceedings…** Berlin: Springer-Verlag, v. 2072, p.327-353.

KICZALES, G. In: ELRAD, T.; KICZALES, G.; AKSIT, M.; LIEBERHER, K.; OSSHER, H. Discussing Aspects of AOP. **Communications of the ACM**, New York, NY, v.44, n.10, p.33-38, 2001.

LADDAD, R. **AspectJ in Action: Practical Aspect-Oriented Programming.** Greenwich, CT: Manning, 2003. 512p.

LAERTY, D.; CAHILL, V. Language-independent Aspect-Oriented Programming. In: Acm Sigplan Conference on Object-Oriented Programing, Systems, Languages, and Applications, 18, 2003, California. **Proceedings…** New York: ACM Press, p. 1-12.

LINGPIPE. Disponível em: http://alias-i.com/lingpipe/. Acesso em: 09 set.2009.

BROWN CORPUS. **Brown Corpus Manual**. Disponível em: http://icame.uib.no/brown/bcm.html. Acesso em: 09 set. 2009.

MILLER, S. K. Aspect Oriented Programming Takes Aim at Software Complexity. **Magazine IEEE's Computer**, vol.34, no.4, p.18-21, 2001.

MOREIRA, A., ARAÚJO, J. e BRITO, I. Crosscutting quality attributes for requirements engineering. In: International Conference on Software Engineering and Knowledge Engineering, 14, 2002, Ischia, Italy. **Proceedings…** New York: ACM Press, 2002, v. 27, p.167-174.

NUSEIBEH, B.; EASTERBROOK, S. Requirements engineering: a roadmap. In: International Conference on Software Engineering, 22, 2000, Limerick, Ireland. **Proceedings**… Limerick, Ireland: ACM, 2000. p. 35-46.

OMG – Object Management Group. Disponível em: <http://www.omg.org/>. Acesso em: 01 abr. 2009.

OSSHER, H.; TARR, P. Hyper/J: Multi-Dimensional Separation of Concerns for Java**.** In: International Conference on Software Engineering, 22, 2000, Limerick, Ireland. **Proceedings…** Limerick, Ireland: ACM Press, 2000. p. 734-737.

PFLEEGER, S. L. **Engenharia de software: teoria e prática.** 2.ed. São Paulo: Prentice Hall, 2004. 560p.

PRESSMAN, R. **Software Engineering: A Practitioner's Approach.** 6.ed. New York: McGraw Hill, 2005. 752p.

RASHID, A.; SAWYER, P.; MOREIRA, A.; ARAÚJO, J. Early Aspects: A Model for Aspect-Oriented Requirements Engineering. In: IEEE Joint International Conference on Requirements Engineering, 10, 2002, Germany. **Proceedings…** Washington: IEEE Computer Society, 2002. p. 199-202.

RASHID, A., MOREIRA, A., ARAÚJO, J.: Modularisation and Composition of Aspectual Requirements. In: International Conference on Aspect Oriented Software Development (AOSD), 2, 2003, Boston. **Proceedings…** New York: ACM Press, 2003. p. 11-20.

RESENDE, A. M. P. MIDAI: Um método para Identificação e Definição de Aspectos Iniciais. 209f. Tese de Doutorado - Instituto Tecnológico de Aeronáutica, São José dos Campos, 2007.

RESENDE, A. M. P.; SILVA, C. C. **Programação Orientada a Aspectos em Java**. Rio de Janeiro: Brasport, 2005. 208p.

RESENDE, A.M.P.; SILVEIRA, F.F.; CUNHA, A. M. Early Aspects: Some analysis, trends and perspectives. In: Early Aspects Workshop, Held in Conjunction with OOPSLA'05 – Object-Oriented Programming, Systems,

Languages And Applications, 2005, San Diego, California. **Proceedings…** San Diego, 2005.

ROBERTSON, J.; ROBERTSON, S. Mastering the requirements process. 2.ed. Boston: Addison-Wesley Professional, 2006. 416p.

ROSENHAINER, L. Identifying crosscutting concerns in requirements specification. In: Workshop on Early Aspects: Aspect-oriented Requirements Engineering and Architecture Design, 2004, Lancaster, UK. **Proceedings…** Lancaster, 2004.

SAMPAIO, A., LOUGHRAN, N. RASHID, A., RAYSON, P.: Mining Aspects in Requirements. In: Early Aspects 2005: Aspect-Oriented Requirements Engineering and Architecture Design Workshop (held with AOSD 2005), 2005, Chicago, Illinois. **Proceedings…** Chicago, 2005.

SCHNEIDER, G.; WINTERS, J. **Applying Use Cases: A Practical Guide**. 2.ed. Boston: Addison-Wesley, 2001. 240p.

SEI – Software Engineering Institute. **CMMI Web Site**. Disponível em: <http://www.sei.cmu.edu/cmmi/>. Acesso em: 20 fev. 2009.

SILVA, L. F. S.; CHAPETTA, W. A. Processamento de Linguagem Natural Aplicada à Inspeção de Documento de Especificação de Requisitos de Software. Rio de Janeiro, Junho, 2003.

SOFTEX. **MPS.BR - Melhoria de Processo do Software Brasileiro – Guia Geral Versão 1.2.** 2007. 52 f. Disponível em: <http://www.softex.br/mpsbr/%5Fguias/MPS.BR_Guia_Geral_V1.2.pdf>. Acesso em: 20 fev. 2009.

SOMMERVILLE, I. **Engenharia de Software**. 6.ed. São Paulo: Addison Wesley, 2004. 592p.

STEEL, R. G. D.; TORRIE, J. H.; DICKEY, D. **Principles and Procedures of Statistics: A Biometrical Approach**. 3.ed. New York: McGraw-Hill Companies, 1996. 672p.

TIRELO, F.; BIGONHA, R. S.; BIGONHA, M. A. S.; VALENTE, M. T. O. Desenvolvimento de Software Orientado por Aspectos. In: Jornada de Atualização em Informática (JAI'04) – Sociedade Brasileira De Computação, 13, 2004, Salvador, BA, v.2, p. 57-96.

UML – Unified Modeling Language. Disponível em: <http://www.uml.org/>. Acesso em: 01 abr. 2009.

WAZLAWIC, R**. Análise e Projeto de Sistemas de Informação Orientados a Objetos.** Rio de Janeiro: Elsevier, 2004. 253p.

WIEGERS, K. E. **Software Requirements**. 2.ed. Redmond, Washington: Microsoft Press, 2003. 516p.