UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

COORDENAÇÃO DO CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

LEANDRO SIQUEIRA DA SILVA

UM MÉTODO PARA IDENTIFICAÇÃO DE ASPECTOS EM NÍVEL DE ANÁLISE BASEADO EM ATRIBUTOS DE REQUISITOS NÃO-FUNCIONAIS

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2012

LEANDRO SIQUEIRA DA SILVA

UM MÉTODO PARA IDENTIFICAÇÃO DE ASPECTOS EM NÍVEL DE ANÁLISE BASEADO EM ATRIBUTOS DE REQUISITOS NÃO-FUNCIONAIS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas da Coordenação de Análise e Desenvolvimento de Sistemas – COADS - da Universidade Tecnológica Federal do Paraná.

Orientadora: Profª. Drª. Simone Nasser Matos

PONTA GROSSA

2012

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa

Diretoria de Graduação e Educação Profissional

TERMO DE APROVAÇÃO

UM MÉTODO PARA IDENTIFICAÇÃO DE ASPECTOS EM NÍVEL DE ANÁLISE BASEADO EM ATRIBUTOS DE REQUISITOS NÃO-FUNCIONAIS

por

LEANDRO SIQUEIRA DA SILVA

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 14 de maio de 2012 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Profª. Drª. Simone Nasser Matos
Orientadora

_____
Profª. Msc. Simone de Almeida
Membro titular

_____
Profª. Msc. Helyane B. Borges
Responsável pelos Trabalhos de Conclusão de Curso

_____
Profª. Msc. Eliana C. M. Ishikawa
Membro titular

_____
Profª. Msc. Simone de Almeida
Coordenadora do Curso
UTFPR – Câmpus Ponta Grossa

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

AGRADECIMENTOS

Agradeço a Deus pela vida, saúde e força para superar as dificuldades enfrentadas ao longo dessa caminhada.

Agradeço a professora Simone Nasser Matos que, mesmo em situações difíceis, dedicou toda a atenção necessária para a realização desse trabalho. Graças a seu conhecimento, dedicação e profissionalismo foi possível transformar uma simples ideia em um trabalho de conclusão de curso.

Agradeço a minha namorada pelo amor e carinho. A minha família pelo apoio recebido ao longo dessa jornada.

Agradeço aos professores que contribuíram, tanto com conhecimento técnico quanto com sabedoria, para minha formação acadêmica.

Agradeço aos meus colegas de turma pela diversão proporcionada durante as aulas.

Enfim, agradeço a todos aqueles que contribuíram, de forma consciente ou não, para a realização do meu objetivo.

RESUMO

SIQUEIRA DA SILVA, Leandro. **Um método para identificação de aspectos em nível de análise baseado em atributos de requisitos não-funcionais**. 2012. 93f. Trabalho de Conclusão de Curso - Curso Superior em Análise e Desenvolvimento de Sistemas - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2012.

A Orientação a Aspectos surgiu para melhorar algumas limitações encontradas na Orientação a Objetos tal como o espalhamento de preocupações. A refatoração de código é a técnica utilizada como processo de identificação de aspectos durante a fase de programação. Porém, esse processo é semelhante ao da identificação de requisitos que já deveriam ter sido identificados e aplicados dentro de um sistema. Este trabalho apresenta um método para realizar a identificação de aspectos durante a fase de análise baseando-se em matriz de adjacência, descrição de caso de uso e atributos de requisitos não-funcionais. O método foi aplicado em um estudo de caso, além de ter sido comparado com os outros métodos da literatura que focam no levantamento dos aspectos na fase inicial de desenvolvimento. Os resultados obtidos por meio da aplicação do método e a conclusão acerca do mesmo também são apresentados.

Palavras-chave: Aspectos. Métodos de Análise. AspectJ.

ABSTRACT

SIQUEIRA DA SILVA, Leandro. **A method for aspects identification at analysis level based in non-functional requirements attributes**. 2012. 92f. 2012. 93f. Trabalho de Conclusão de Curso - Curso Superior em Análise e Desenvolvimento de Sistemas - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2012.

The Aspect Orientation appeared to improve some limitations found in the Object Orientation, like crosscutting concerns. The code refactoring is the technique used to identify aspects during the programming phase. However, this process is like identifying requirements that already should have be identified and applied into a system. This work shows a method to identifying aspects during the analysis phase. This method is based in adjacency matrix, use case description and non-functional requirements attributes. It was applied in a case study, besides have been compared to other methods found in the literature. The results obtained using the method and conclusions about the same are shown.

Keywords: Aspects. Analysis Methods. AspectJ.

LISTA DE ILUSTRAÇÕES

Figura 1 – Fases de desenvolvimento de caso de uso 17
Figura 2 – Atividades realizadas nos estágios iterativos 18
Figura 3 – Descrição de caso de uso 19
Figura 4 – *Use Case Slice* Reservar Quarto 22
Figura 5 – Tipos de caso de uso 23
Figura 6 - *Use Case Slice* com relacionamento do tipo *include* 23
Figura 7 - *Use Case Slice* com relacionamento do tipo *extend* 24
Figura 8 - *Use Case Slice* com relacionamento do tipo generalização 25
Figura 9 - Processo Geral do Método 26
Figura 10 - Descrição Textual “Processar Opção” 26
Figura 11 - Responsabilidades identificadas 27
Figura 12 - Exemplo de *join point* utilizando AspectJ 31
Figura 13 - Exemplo de aspecto utilizando *AspectJ* 31
Figura 14 - Modelo de *weaving* 33
Figura 15 - Processo geral do método proposto 38
Figura 16 - *Garbage Collector* implementado em AspectC++ 42
Figura 17 - Espalhamento de aspectos em matrizes adjacentes 46
Figura 18 – Subprocesso para identificar e tratar espalhamento de aspectos em matriz de adjacência 48
Figura 19 – Espalhamento de atributos de aspectos 58
Figura 20 – Codificação de aspecto utilizando Javascript 64
Figura 21 – Codificação de aspecto utilizando AspectJ 65

LISTA DE QUADROS

Quadro 1 – Princípios de Engenharia de Software 28
Quadro 2 - Semelhanças entre aspectos e requisitos não-funcionais.....37
Quadro 3 - Requisitos Não-Funcionais e seus atributos 40
Quadro 4 - Requisito Não-Funcional Desempenho.....41
Quadro 5 – Modelo de matriz de adjacência.....43
Quadro 6 – Descrição textual Realizar Cadastro de Obra 44
Quadro 7 – Matriz de adjacência para a descrição textual Realizar Cadastro de Obra 44
Quadro 8 – Descrição textual Consultar Cadastro de Obra 45
Quadro 9 – Matriz de adjacência para a descrição textual Consultar Cadastro de Obra 45
Quadro 10 – Matriz auxiliar utilizada para tratar espalhamento de aspectos 47
Quadro 11 – Matriz auxiliar preenchida com aspectos do tipo desempenho 49
Quadro 12 – Descrição textual utilizada para representar os aspectos identificados50
Quadro 13 – *Poincut* representando um tipo de aspecto 51
Quadro 14 – Representação de aspectos dentro de descrição textual.....52
Quadro 15 – Detalhes sobre os aspectos identificados 52
Quadro 16 – Descrição textual Logar no Sistema.....56
Quadro 17 – Descrição textual Conferir Login.....57
Quadro 18 – Matriz de adjacência Segurança para a descrição Logar no Sistema..58
Quadro 19 - Matriz auxiliar para atributos de aspecto do tipo segurança 59
Quadro 20 – Identificação de aspectos dentro da descrição textual Logar no Sistema 60
Quadro 21 – Identificação de aspectos dentro da descrição textual Conferir Login..61
Quadro 22 – Detalhes sobre os aspectos identificados do tipo usabilidade.....61
Quadro 23 – Atributos de aspecto identificados na descrição textual Cadastrar Usuário.....62
Quadro 24 – Atributos de aspecto para a descrição textual Enviar Mensagem 62
Quadro 25 – Atributos de aspecto implementados para o sistema Cadastro de Usuário.....64
Quadro 26 – Atributos de aspectos implementados para o sistema de Login.....66
Quadro 27 – Comparativo entre o método proposto e a técnica proposta por Kong e Yuan (2009).....67
Quadro 28 - Comparativo entre o método proposto e o método apresentado por Jacobson e NG (2004) 68
Quadro 29 – Comparativo entre o método proposto e a abordagem apresentada por Hornung 68
Quadro 30 – Requisito Não Funcional Manutenibilidade 77
Quadro 31 – Requisito Não Funcional Confiabilidade.....77
Quadro 32 – Requisito Não Funcional Usabilidade.....78
Quadro 33 – Requisito Não Funcional Segurança 79

Quadro 34 – Matriz de adjacência Usabilidade para a descrição Logar no Sistema.81
Quadro 35 – Matriz de adjacência Confiabilidade para a descrição Logar no Sistema 81
Quadro 36 – Matriz de adjacência Manutenibilidade para a descrição Logar no Sistema 82
Quadro 37 – Matriz de adjacência Desempenho para a descrição Logar no Sistema 82
Quadro 38 – Matriz de adjacência Segurança para a descrição Conferir Login.....83
Quadro 39 – Matriz de adjacência Usabilidade para a descrição Conferir Login 83
Quadro 40 – Matriz de adjacência Confiabilidade para a descrição Conferir Login ..83
Quadro 41 – Matriz de adjacência Manutenibilidade para a descrição Conferir Login 84
Quadro 42 – Matriz de adjacência Desempenho para a descrição Conferir Login....84
Quadro 43 – Matriz auxiliar para o aspecto do tipo usabilidade 86
Quadro 44 – Matriz auxiliar para o aspecto do tipo confiabilidade 86
Quadro 45 - Matriz auxiliar para o aspecto do tipo manutenibilidade.....87
Quadro 46 – Matriz auxiliar para o aspecto do tipo desempenho 87
Quadro 47 – Detalhes sobre os aspectos identificados do tipo segurança 89
Quadro 48 – Detalhes sobre os aspectos identificados do tipo desempenho 89
Quadro 49 – Detalhes sobre os aspectos identificados do tipo confiabilidade.....90
Quadro 50 – Detalhes sobre os aspectos identificados do tipo manutenibilidade.....90
Quadro 51 – Atributos de aspectos identificados em Preencher Campos Obrigatórios, sistema Cadastro de Usuário.....92
Quadro 52 – Atributos de aspectos identificados em Validar Campos Obrigatórios, sistema Cadastro de Usuário 92
Quadro 53 – Atributos de aspectos identificados em Validar Dados Específicos, sistema Cadastro de Usuário 92
Quadro 54 – Atributos de aspectos identificados em Preencher Campos Obrigatórios, sistema Fale Conosco 93
Quadro 55 – Atributos de aspectos identificados em Validar Campos Obrigatórios, sistema Fale Conosco.....93
Quadro 56 – Atributos de aspectos identificados em Validar Dados Específicos, sistema Fale Conosco.....93

LISTA DE SIGLAS

DT	Descrição Textual
EROA	Engenharia de Requisitos Orientada a Aspectos
FA	Fluxo Alternativo
FB	Fluxo Básico ou Principal
GPES	Grupo de Pesquisa em Engenharia de Software
HTML	*HyperText Markup Language*
MRF	Modelo de Requisitos do *Framework*
OA	Orientação a Aspectos
OO	Orientação a Objetos
POO	Projeto Orientado a Objetos
RF	Requisito Funcional
RNF	Requisito Não-Funcional
UCS	*Use Case Slice*
UML	*Unified Modeling Language*

LISTA DE ACRÔNIMOS

CASE	*Computer-Aided* Software *Engineering*
POA	Programação Orientada a Aspectos
RUP	*Rational Unified Process*

SUMÁRIO

1 INTRODUÇÃO 13
1.1 OBJETIVO 14
1.1.1 Objetivo Geral 14
1.1.2 Objetivos Específicos 14
1.2 PROBLEMA 14
1.3 ORGANIZAÇÃO DO TRABALHO 15
2 ANÁLISE, PROJETO E PROGRAMAÇÃO ORIENTADA A ASPECTOS 16
2.1 ANÁLISE ORIENTADA A ASPECTOS 16
2.1.1 Abordagem de Modelagem de Caso de Uso para Aquisição de *Early Aspects* 17
2.1.2 Identificação de aspectos baseada em *Use Case Slice* 21
2.1.3 Identificação de candidatos a Aspectos Utilizando Responsabilidades 25
2.2 PROJETO ORIENTADO A ASPECTOS 28
2.2.1 Programação orientada a aspectos 29
2.2.1.1 Preocupação Transversal 30
2.2.1.2 *Join Point* 30
2.2.1.3 Aspecto 31
2.2.1.4 *Advice* 32
2.2.1.5 *Pointcut* 32
2.2.1.6 *Weaving* 32
2.3 ANÁLISE DOS MÉTODOS DE IDENTIFICAÇÃO DE ASPECTOS NA FASE DE ANÁLISE 34
3 MÉTODO PROPOSTO PARA IDENTIFICAÇÃO DE ASPECTOS NA FASE DE ANÁLISE 36
3.1 JUSTIFICATIVA DA CRIAÇÃO DO MÉTODO 36
3.2 PROCESSO GERAL DO MÉTODO PROPOSTO 37
3.2.1 Identificar requisitos não funcionais 39
3.2.1.1 Identificação de Atributos dos Requisitos Não Funcionais 40
3.2.2 Montar matriz de adjacência 42
3.2.3 Identificar aspectos em matriz de adjacência 43
3.2.4 Verificar espalhamento de aspectos em matriz de adjacência e tratar espalhamento 46
3.3 TRATAR ASPECTOS IDENTIFICADOS 49
3.4 VANTAGENS DO MÉTODO PROPOSTO 52
4 APLICAÇÃO DO MÉTODO PROPOSTO 54
4.1 JUSTIFICATIVA PARA A ESCOLHA DAS DESCRIÇÕES TEXTUAIS 54
4.2 IDENTIFICAÇÃO DE ASPECTOS UTILIZANDO O MÉTODO PROPOSTO 55
4.2.1 Identificar requisitos não funcionais 55
4.2.2 Montar matriz de adjacência 55

4.2.3 Identificar aspectos em matriz de adjacência 57
4.2.4 Verificar espalhamento de aspectos em matriz de adjacência 58
4.2.5 Tratar espalhamento em matriz de adjacência 59
4.2.6 Tratar aspectos identificados 59
4.3 ATRIBUTOS DE ASPECTOS IDENTIFICADOS PARA OS SISTEMAS DE LOGIN, CADASTRO DE USUÁRIO E FALE CONOSCO 62
5 RESULTADOS 63
5.1 CODIFICAÇÃO DE ASPECTOS.....63
5.2 COMPARATIVO ENTRE OS MÉTODOS 66
5.2.1 Abordagem de modelagem de caso de uso para aquisição de *early aspects* 66
5.2.2 Identificação de aspectos baseada em *Use Case Slice* 67
5.2.3 Identificação de candidatos a aspectos utilizando responsabilidades 68
5.2.4 resumo do comparativo entre métodos.....69
6 CONCLUSÃO.....70
6.1 TRABALHOS FUTUROS 70
APÊNDICE A - Quadro de Requisitos Não Funcionais 76
APÊNDICE B - Matrizes de adjacência resultantes da aplicação do método ..80
APÊNDICE C - Matrizes auxiliares desenvolvidas durante a aplicação do método 85
APÊNDICE D - Atributos de aspectos identificados em Logar no Sistema e Conferir Login 88

1 INTRODUÇÃO

A necessidade em desenvolver aplicações suscetíveis à mudança exige um paradigma de desenvolvimento que facilite a sua criação, manutenção e evolução.

A Orientação a Objetos (OO) é o principal paradigma utilizado para o desenvolvimento de tais aplicações. Características como: herança, reuso de código e polimorfismo são algumas das vantagens proporcionadas por esse paradigma. Contudo, as vantagens oferecidas pelo paradigma OO não são suficientes para tratar as Preocupações Transversais (*Crosscuting Concerns*) de uma aplicação.

Preocupações Transversais são características que estão espalhadas pelo sistema. Esse espalhamento dificulta sua modularização e compromete o reuso de código e a manutenibilidade do software. Consequentemente, o princípio de que cada classe deve possuir apenas uma preocupação é comprometido.

A Orientação a Aspectos (OA) é um paradigma criado para suprir as necessidades do paradigma OO, sendo que sua base é a orientação a objetos. Por meio do uso da OA é possível estender as vantagens proporcionadas pela Orientação a Objetos e aumentar o nível de modularidade das aplicações.

Este trabalho apresenta os métodos de Jacobson e NG (2004), Kong e Yuan (2009) e Hornung (2010) para identificação de aspectos em nível de análise. Propõe também um método para identificação de aspectos construído a partir da análise de um estudo sobre Requisitos Não-Funcionais (RNF) (MAIRIZA et al., 2010), aplicação de descrições de caso de uso e matriz de adjacência.

A aplicação do método foi realizada em um estudo de caso de um sistema de *Login*, *Fale Conosco* e *Cadastro de Usuário* que foi desenvolvido pelo Grupo de Pesquisa em Engenharia de Software (TORRENS; MATOS, 2010; KOSSOSKI, 2010). Após sua aplicação, o método foi comparado com os outros da literatura mostrando as suas diferenças em relação à base teórica.

Os aspectos identificados no estudo de caso foram codificados, usando *JavaScript* (*JavaScript,* 2012), *AspectJ* (*AspectJ*, 2012) e *jQuery AOP* (*jQuery AOP*, 2012), com o objetivo de mostrar a ligação entre teoria e prática, mas ressalta-se que o método proposto é usado somente para identificação dos aspectos nas fases iniciais do processo de desenvolvimento de sistemas.

1.1 OBJETIVO

A seguir serão descritos os objetivos gerais e específicos deste trabalho.

1.1.1 Objetivo Geral

Criar e aplicar um método para a identificação de aspectos nas atividades iniciais do desenvolvimento de software, usando como referência atributos de requisitos não-funcionais, matriz de adjacência e descrições de casos de uso.

1.1.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Descrever as vantagens e limitações dos métodos de identificação de aspectos descritos por Jacobson e NG (2004), Kong e Yuan (2009) e Hornung (2010).
- Desenvolver um método para identificação de aspectos em nível de análise.
- Aplicar o método proposto em um estudo de caso.
- Comparar o método proposto em relação aos outros de identificação de aspectos da literatura que atuem também em fase inicial da criação de sistemas.

1.2 PROBLEMA

A Orientação a Aspectos (AO) possui várias implementações para a etapa de programação. Contudo, as etapas anteriores do desenvolvimento de software necessitam de métodos para a identificação de aspectos. Boa parte dos desenvolvedores que utilizam o paradigma OA também identifica aspectos a partir do código fonte.

Um dos problemas enfrentados por desenvolvedores que utilizam a OA é que os aspectos são geralmente identificados em atividades de refatoração de código. Ou seja, o desenvolvedor não consegue identificar aspectos sem um código

fonte para refatorar. Outro problema decorrente dessa abordagem é que identificar aspectos dentro de um software já aplicado em ambiente de produção é o mesmo que identificar requisitos que já deveriam ter sido identificados e aplicados no sistema.

Um método para identificação de aspectos durante a fase de análise é uma alternativa mais viável para identificação de aspectos se comparado a refatoração de código fonte. O uso de um método para identificação de aspectos nas etapas iniciais do desenvolvimento de software permite diminuir os custos de desenvolvimento e construir software mais robusto e consistente.

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho está dividido em seis capítulos. O capítulo 2 descreve a Análise e Projeto Orientado a Aspectos, apresentando os três métodos, Jacobson e NG (2004), Kong e Yuan (2009) e Hornung (2010), para identificação de aspectos e descreve uma análise sobre eles.

O capítulo 3 relata o método proposto para identificação de aspectos. O capítulo 4 exibe um caso de uso onde o método foi aplicado. O capítulo 5 apresenta a implementação de alguns atributos de aspectos identificados pelo método, além de realizar uma breve comparação entre os métodos da literatura e o proposto.

Por fim, o capítulo 6 traz as conclusões acerca do trabalho apresentado e descrevendo as propostas para trabalhos futuros.

2 ANÁLISE, PROJETO E PROGRAMAÇÃO ORIENTADA A ASPECTOS

Este capítulo apresenta alguns conceitos sobre Análise e Projeto Orientado a Aspectos. A Seção 2.1 descreve a fase de Análise Orientada a Aspectos e apresenta os métodos para identificação de aspectos. A seção 2.2 discorre sobre a fase de Projeto Orientado a Aspectos e apresenta a Programação Orientada a Aspectos. Por fim, a seção 2.3 relata uma análise dos métodos detalhados na seção 2.1.

2.1 ANÁLISE ORIENTADA A ASPECTOS

A coleta de requisitos é a principal tarefa dentro do desenvolvimento de software. Essa tarefa ocorre, principalmente, durante a fase de análise.

Os requisitos coletados podem ser divididos em Requisitos Funcionais (RF) e Requisitos Não-Funcionais (RNF). Os RFs dizem respeito à regra de negócio do sistema, enquanto os RNFs estão relacionados com padrões de qualidade tais como: segurança, confiabilidade e desempenho (XAVIER, 2009).

Os aspectos podem apresentar características similares aos Requisitos Funcionais e Requisitos Não-Funcionais, mas eles são geralmente identificados como Requisitos Não-Funcionais.

A identificação de aspectos durante a fase de análise pode tornar-se uma tarefa complicada, uma vez que os Requisitos Não-Funcionais não recebem a mesma atenção que os funcionais. Consequentemente, eles não recebem a devida importância.

É importante identificar aspectos durante a fase de análise, porque eles têm impacto direto na qualidade do software. Caso eles não sejam identificados nesta fase, o processo de levantamento em etapas posteriores tornar-se-á mais difícil, uma vez que a identificação e a implementação desses aspectos dependerá do funcionamento do sistema. Isso significa que, quando o ciclo de desenvolvimento de um software não possui atividades voltadas para a identificação de aspectos na fase de análise, esses são levantados e implementados por meio de refatoração de código (FOWLER et al., 1999).

A seguir serão apresentados os métodos de Kong e Yuan (2009), Jacobson e NG (2004) e Hornung (2010). Esses métodos são utilizados para identificação de aspectos dentro da fase de análise, propondo a identificação de *early aspects*. Atualmente, existem os aspectos que são identificados durante as atividades de programação e *early aspects*, que são aspectos identificados durante as atividades de elicitação de requisitos (JACOBSON; NG, 2004).

2.1.1 Abordagem de Modelagem de Caso de Uso para Aquisição de *Early Aspects*

A técnica proposta por Kong e Yuan (2009) faz uso da metodologia de Engenharia de Software *Rational Unified Process* (RUP) para identificação de aspectos. Essa identificação geralmente ocorre na Análise de Requisitos, Análise de Domínio e Projeto de Arquitetura, mas também pode ser utilizada durante a fase de projeto de software. A Figura 1 ilustra os passos dessa abordagem.

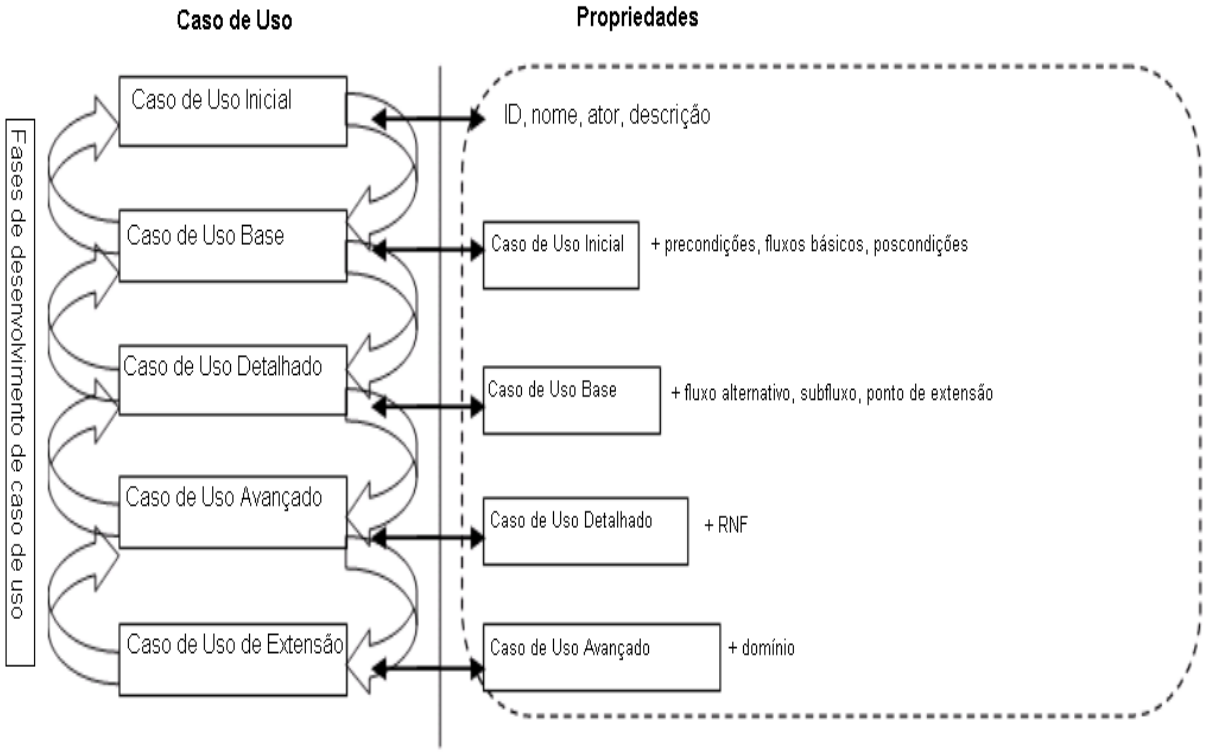


Figura 1 – Fases de desenvolvimento de caso de uso
Fonte: Adaptado de Kong; Yuan (2009)

Esta figura ilustra os passos percorridos durante a aplicação dessa abordagem. Nela a modelagem de caso de uso é dividida em cinco estágios iterativos, citados a seguir:

1) Modelagem de Caso de Uso Inicial: identificação dos atores, casos de uso e limites do sistema.
2) Modelagem de Caso de Uso Base: define o relacionamento entre atores e casos de uso.
3) Modelagem de Caso de Uso Detalhado: define os fluxos alternativos, subfluxos e pontos de extensão estruturados.
4) Modelagem de Caso de Uso Avançado: identificação de requisitos não funcionais.
5) Modelagem de Caso de Uso de Extensão: adiciona uma propriedade chamada domínio. Essa propriedade refere-se às classes conceituais que participam da realização do caso de uso.

A identificação de aspectos ocorre por meio da análise das propriedades de caso de uso. Essas propriedades são adicionadas ao modelo de caso de uso por meio da execução dos cinco estágios iterativos. A Figura 2 ilustra as atividades realizadas nesses cinco estágios. Essas atividades foram definidas com base no modelo de Engenharia de Requisitos Orientada a Aspectos (EROA) de Rashid et al. (2002).

Estágios de modelagem de caso de uso / Atividades de modelagem de caso de uso	Caso de Uso Inicial	Caso de Uso Básico	Caso de Uso Detalhado	Caso de Uso Avançado	Extender Caso de Uso
Identificar atores e casos de uso	✓	✓	✓		
Descrever caso de uso		✓	✓	✓	
Capturar e descrever RNF		✓	✓	✓	
Estruturar Modelo de caso de uso			✓	✓	✓
Identificar domínio			✓	✓	✓
Identificar RF candidatos a aspectos			✓		✓
Identificar RNF candidatos a aspectos				✓	
Identificar e revolver conflitos				✓	✓
Estruturar caso de uso e aspectos de RNF				✓	
Estruturar caso de uso e aspectos de RF					✓

Figura 2 – Atividades realizadas nos estágios iterativos
Fonte: Adaptado de Kong; Yuan (2009)

Esta figura ilustra as atividades realizadas em cada estágio da técnica proposta. A identificação de aspectos candidatos terá início a partir da atividade

Caso de Uso Detalhado. O processo para identificar se um aspecto candidato pode se tornar um aspecto será realizado por meio da análise de propriedades de caso de uso.

A Figura 3 ilustra um exemplo de Descrição Textual (DT) de caso de uso que utiliza os cinco estágios iterativos. Ela também apresenta as propriedades: Fluxo Alternativo, Caso de Uso de Inclusão, RNF e Domínio. Essas propriedades devem ser analisadas para a identificação de aspectos.

Nome do Caso de Uso: Preparar Formulário de Agendamento de Produção.

Ator: Planejador.

Fluxo Básico

1. Planejador seleciona criação de formulário de plano de produção.
2. Sistema oferece um novo formulário de plano de produção.
3. Planejador seleciona o ID do formulário de pedido.
4. Sistema exibe nome e quantidade do produto no formulário de pedido.
5. Planejador seleciona no formulário de pedido os produtos que planeja produzir.
6. Sistema Analisa Inventário Disponível (Caso de Uso de Inclusão).
7. Planejador preenche os dados de entrega de acordo com a quantidade do inventário.
8. Sistema salva formulário de plano de produção e sinaliza a condição do armazém como ocupada.

Fluxo Alternativo

7a. se quantidade disponível no inventário < estoque mínimo, então:

1. Sistema gera aviso de compra, Criar Pedido de Compra (Caso de Uso de Extensão).
2. Planejador preenche os dados de entrega de acordo com o contrato de compra.

7b. se quantidade disponível no inventário > estoque mínimo, então:

1. Avança para o passo 7 do Fluxo Básico.

Requisitos Não Funcionais

Segurança, Auditoria.

Domínio

Pedido, produto, contrato de compra, inventário, formulário de agendamento de produção.

Figura 3 – Descrição de caso de uso
Fonte: Adaptado de Kong; Yuan (2009)

As propriedades ilustradas pela Figura 3 são descritas a seguir:

- Fluxo Alternativo: é uma das propriedades que aparece em destaque na Descrição Textual de caso de uso. Ela indica um fluxo de evento opcional ou excepcional. Esse fluxo costuma ter apenas uma atividade

encapsulada dentro dele. Essa propriedade torna-se um candidato a aspecto porque ela pode ser utilizada por vários casos de uso e, ainda assim, não pertencer ao domínio do problema desses casos de uso.

- Caso de Uso de Extensão: a ideia por trás dessa propriedade é adicionar comportamento ao sistema sem modificá-lo, tal como aspectos. Por esse motivo, essa propriedade torna-se um candidato a aspectos.
- Requisitos Não Funcionais: é comum os Requisitos Não-Funcionais serem mapeados para aspectos. Por esse motivo, eles se tornam candidatos a aspectos.
- Domínio: essa propriedade é adicionada após a realização do estágio Caso de Uso Avançado. Os domínios de cada caso de uso foram identificados sem levar em consideração as preocupações transversais. Deste modo, nomes de domínios iguais, mas com preocupações diferentes, podem aparecer nos casos de uso. O estudo apresentado por Kong e Yuan (2009) traz uma Matriz de Preocupação de Domínio de Caso de Uso para auxiliar o analista a identificar aspectos.

As propriedades utilizadas para a identificação de candidatos a aspectos são coletadas a partir do processo de criação de casos de uso ilustrado pela Figura 1. Cada caso de uso apresentado nessa figura representará uma parte da descrição textual de caso de uso.

Há duas opções que podem ser tomadas para a identificação de aspectos candidatos:

- Identificação durante a criação da descrição textual de casos de uso: utilizar essa opção significa que, durante cada iteração, os desenvolvedores irão procurar por aspectos dentro da descrição de caso de uso. Por exemplo, após a criação do caso de uso base, aspectos serão procurados dentro das informações obtidas a partir desse caso de uso. Esse processo se repete para todos os casos de uso.

- Identificação após a criação da descrição textual de casos de uso: a identificação de aspectos só ocorre quando a descrição textual estiver completamente finalizada. Isso significa que somente após os cinco estágios iterativos serem realizados é que a identificação de aspectos será iniciada.

A primeira opção permite procurar novos aspectos a cada iteração. Já a segunda permite que a única preocupação, durante todas as iterações, seja com a regra de negócio. Após a conclusão da descrição textual é que os aspectos serão identificados.

2.1.2 Identificação de aspectos baseada em *Use Case Slice*

A técnica de caso de uso é muito utilizada para a identificação das preocupações de um sistema (JACOBSON; NG, 2004). Ela explora essas preocupações a partir da perspectiva do usuário do sistema.

A grande limitação atual dessa técnica é a lacuna entre ela e a orientação a aspectos. A técnica de caso de uso não possui uma unidade para encapsular os aspectos identificados, enquanto que a programação orientada a aspectos, que será apresentada na seção 2.3, utiliza uma unidade de encapsulamento chamada aspecto para modularizar preocupações transversais de um sistema. A unidade de encapsulamento *Use Case Slice (USC)* foi desenvolvida com o intuito de preencher essa lacuna entre caso de uso e programação orientada a aspectos.

Use Case Slice é uma unidade utilizada para preservar a modularidade dos casos de uso durante sua realização e conter os elementos específicos necessários para a realização de um Caso de Uso. Eles são apresentados em pacotes que contêm o estereótipo <<*use case slice*>>, seguido pelo nome do caso de uso (JACOBSON; NG, 2004). A Figura 4 ilustra um exemplo dessa unidade.

A realização de Caso de Uso irá gerar classes e recursos (atributos, operações e relacionamentos) específicos para cada Caso de Uso. Algumas classes e recursos podem ser utilizados para a realização de um ou mais Casos de Uso.

Um *Use Case Slice* é composto por:

- Uma colaboração que descreve a realização do caso de uso.

- Classes específicas para a realização do caso de uso.
- Extensões de classes existentes específicas para a realização do caso de uso.

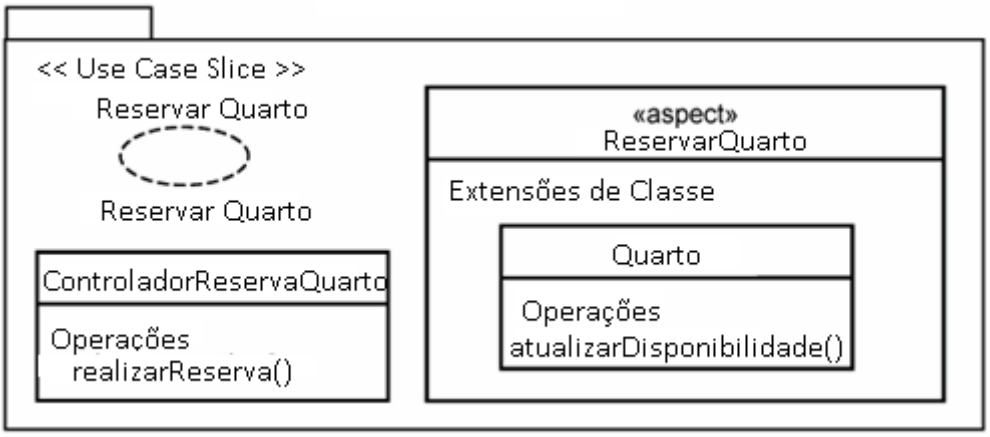


Figura 4 – *Use Case Slice* Reservar Quarto
Fonte: Adaptado de Jacobson; NG (2004)

A Figura 4 ilustra um *Use Case Slice* utilizado para a modelagem de um sistema de gerenciamento de hotel fictício. A colaboração “Reservar Quarto” possui um conjunto de diagramas (diagrama de interação, diagrama de classe, entre outros) necessários para a sua realização.

A classe “ControladorReservaQuarto” é específica para a realização desse Caso de Uso. Isso significa que a realização de qualquer caso de uso desse sistema não depende da classe “ControladorReservaQuarto”, exceto “Reservar Quarto”.

O aspecto “ReservarQuarto” tem uma classe de extensão chamada “Quarto”. Essa classe é utilizada para a realização de mais de um caso de uso. Ela possui um método chamado “atualizarDisponibilidade”. Porém, esse método é utilizado apenas pelo caso de uso “Reservar Quarto”. Se o uso desse método fosse necessário em mais de um caso de uso ele seria declarado em outro *Use Case Slice*.

É importante ressaltar que os casos de uso possuem vários tipos de relacionamento, dentre os quais podem ser citados: *include*, *extend* e generalização. Estes podem ser utilizados com *Use Case Slice*. A Figura 5 apresenta esses tipos de relacionamento.

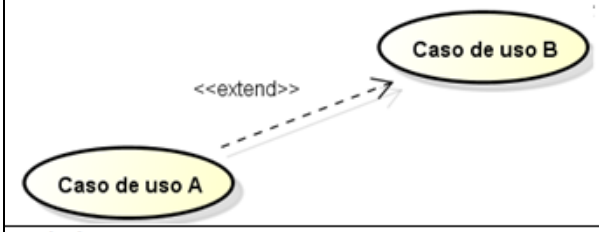
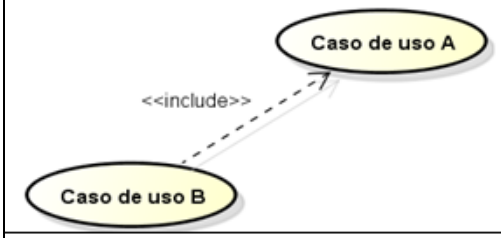
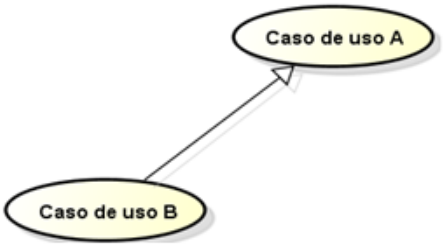
Extend 	Um relacionamento do tipo *extend* do caso de uso A para o caso de uso B indica que o caso de uso A (opcional) pode ser acrescentado ao comportamento do caso de uso B. O estereótipo <<extend>> é utilizado para representar esse relacionamento.
Include 	Um relacionamento do tipo *include* do caso de uso B para o caso de uso A significa que, o caso de uso A é essencial para o funcionamento correto do caso de uso B. O estereótipo <<include>> é utilizado para representar esse relacionamento.
Generalização 	Um relacionamento entre um caso de uso mais geral (caso de uso A) com um caso de uso mais especializado (caso de uso B). O caso de uso A possui informações que podem ser comuns entre vários casos de uso, enquanto o caso de uso B possui informações específicas para seu comportamento, além daquelas herdadas do caso de uso A.

Figura 5 – Tipos de caso de uso
Fonte: Autoria Própria

Os relacionamentos ilustrados pela Figura 5 são aplicados a partir do *Use Case Slice* ilustrado pela Figura 4 e seguem o mesmo raciocínio relacionado ao sistema de gerenciamento de hotel.

A Figura 6 mostra o relacionamento do tipo *include* entre os *Use Case Slices* “Reservar Quarto” e ”Verificar Detalhes Quarto”. Esse comportamento é modelado como dependência entre *Use Case Slices*.

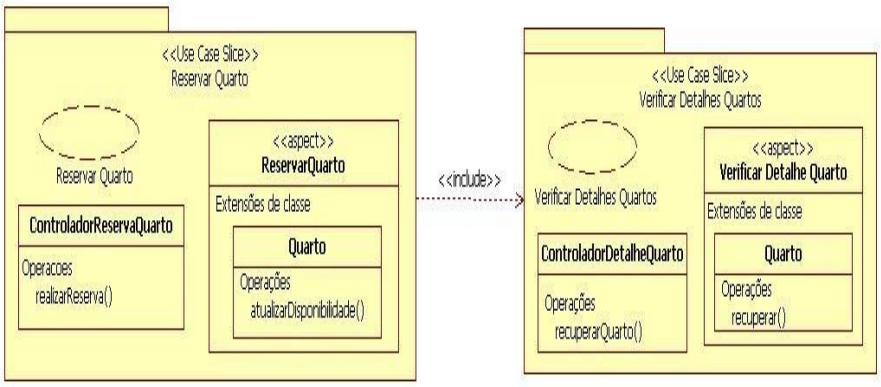


Figura 6 - *Use Case Slice* com relacionamento do tipo *include*
Fonte: Adaptado de Jacobson; NG (2004)

Esta figura ilustra parte do processo para fazer uma reserva em um hotel. O UCS “Reservar Quarto” realiza a reserva de quarto. Para que esse comportamento seja realizado de forma correta, ele deve obrigatoriamente verificar os detalhes do quarto. A verificação se torna obrigatória, uma vez que ela está representada com um relacionamento do tipo <<*include*>>. A verificação de detalhes de quartos está modularizada dentro do UCS “Verificar Detalhes Quartos”. Isso evita o espalhamento dessa funcionalidade por vários UCSs. Ainda, a classe “ControladorReservaQuarto” pode ter uma associação ou dependência com esta classe. Os *Use Case Slices* ilustrados pela Figura 6, “Verificar Quartos” e “Verificar Detalhes do Quarto”, não apresentam dependências entre aspectos, apenas com as classes.

A Figura 7 apresenta o relacionamento do tipo <<*extend*>>. Dentro de um sistema de hotel é indispensável a existência de quartos. Dessa forma, a classe Quarto não é específica de um caso de uso e também não deve ser definida dentro de um *Use Case Slice*.

Desse modo, ela é definida dentro de uma unidade representada pelo estereótipo <<*non-uc specific slices*>>, porque ela faz parte do domínio da regra de negócio. Dessa forma, ela estará disponível para que outras classes possam estendê-la.

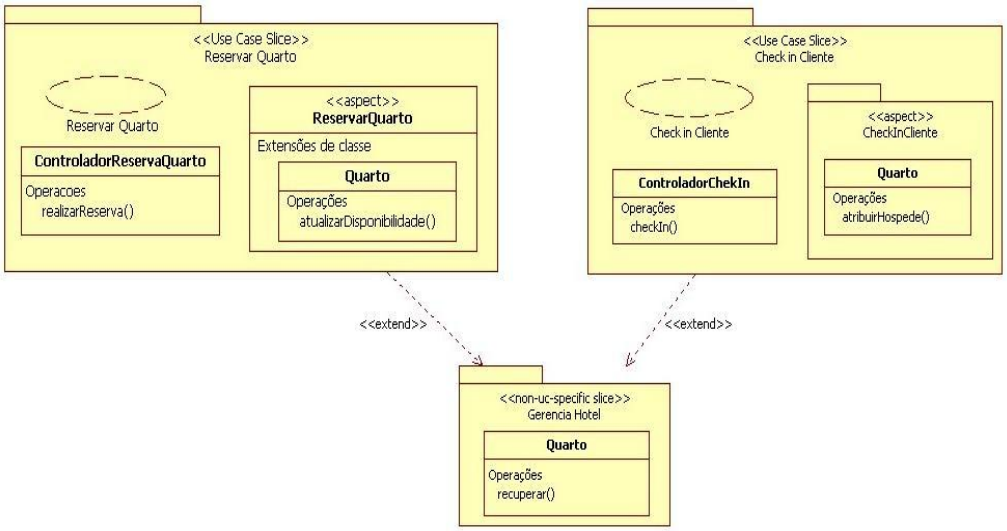


Figura 7 - *Use Case Slice* com relacionamento do tipo *extend*
Fonte: Adaptado de Jacobson; NG (2004)

Esta figura apresenta uma unidade com o estereótipo <<*non-uc specific slices*>>. Já que esse tipo de unidade representa algum tipo de domínio de negócio, ele não possui aspectos.

A Figura 8 ilustra o relacionamento do tipo generalização. Nela é possível observar que a classe “ControladorReservaInstalação” apresenta um método chamado “realizarReserva()”, que pode ser abstrato ou não. A classe “ControladorReservaQuarto” especializa a classe “ControladorReservaInstalação”. Em seguida, é possível visualizar que a classe “Quarto”, dentro do aspecto “ReservarQuarto”, especializa a classe “Instalação”.

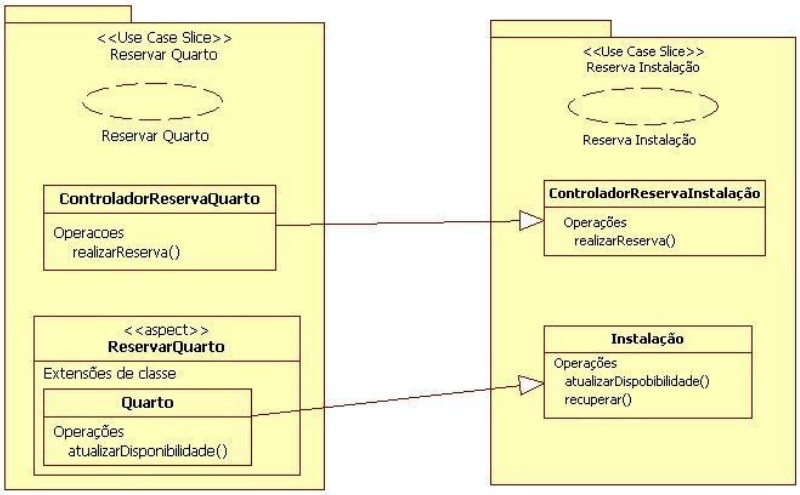


Figura 8 - *Use Case Slice* com relacionamento do tipo generalização
Fonte: Adaptado de Jacobson; NG (2004)

O uso de *Use Case Slice* é uma forma de evitar que tanto a identificação quanto a implementação de aspectos seja realizada somente com a refatoração de código.

2.1.3 Identificação de candidatos a Aspectos Utilizando Responsabilidades

A abordagem para identificação de aspectos baseada em responsabilidades (HORNUNG, 2010) apresenta uma maneira de identificar candidatos a aspectos durante a Análise de Requisitos. Com isso, pretende-se aproveitar as vantagens do Desenvolvimento de Software Orientado a Aspectos, tais como:

- Índices menores de entrelaçamento e espalhamento de código.
- Níveis maiores de modularidade e manutenibilidade do software.

Esse método foi proposto para funcionar como parte da abordagem descrita por Matos e Fernandes (2007). A Figura 9 ilustra o processo do método para identificação de candidatos a aspectos utilizando responsabilidades.

Figura 9 - Processo Geral do Método
Fonte: Hornung (2010, p. 51)

O Modelo de Requisitos do *Framework* (MRF) é utilizado como entrada para o processo ilustrado pela Figura 9. Ele é constituído por diagramas de Caso de Uso representados graficamente na notação UML-F (FONTOURA et al., 2000) e Descrição Textual.

O primeiro subprocesso, PASSOS 1-3, é composto por três passos e é responsável pela identificação e representação das responsabilidades espalhadas pelos casos de uso (HORNUNG, 2010). O resultado deste subprocesso é um documento MRF com as responsabilidades espalhadas.

O Passo 1 é realizado para identificar as responsabilidades de cada caso de uso. Para identificar as responsabilidades, o seguinte critério é utilizado:

Responsabilidade = Verbo + Complemento

A identificação das responsabilidades de caso de uso é a principal etapa desse método. A Figura 10 ilustra uma Descrição Textual de caso do uso “Processar Opção” (HORNUNG, 2010) e a Figura 11 apresenta as responsabilidades identificadas em “Processar Opção” a partir da aplicação do critério citado anteriormente.

Nome do Caso de Uso	Processar Opção
Estabilidade [X]	Flexibilidade [ ]
Ações do Usuário	Responsabilidades do Sistema
1. Selecionar opção	2. Verificar a opção selecionada
	3. Executar uma ação a partir da opção escolhida
4. Apresentar resultado sobre a opção escolhida	

Figura 10 - Descrição Textual “Processar Opção”
Fonte: Hornung (2010, p. 53)

Verbo	Pergunta ao verbo	Complemento	Responsabilidade
Verificar...	O quê?	Opção	Verificar opção
Executar...	O quê?	Ação	Executar ação

Figura 11 - Responsabilidades identificadas
Fonte: Hornung (2010, p. 53)

O Passo 2, definir o Conjunto de Responsabilidades Espalhadas, consiste em identificar as responsabilidades que estão espalhadas por várias descrições textuais. Em seguida, é necessário analisá-las em busca de erros, como por exemplo, uma responsabilidade que não pertence a uma determinada descrição textual.

O Passo 3 consiste em representar as responsabilidades espalhadas. Para isso é necessário criar um caso de uso para cada responsabilidade. A associação entre esses casos de uso e aqueles já representados no MRF será realizada utilizando o estereótipo <<*crosscuts*>>.

O resultado desses três passos constitui o MRF contendo as responsabilidades espalhadas dos casos de uso.

O segundo subprocesso, PASSOS 4-6, contém os três últimos passos do processo. Por meio desses passos é possível identificar casos de uso candidato a aspectos.

O Passo 4 consiste em identificar os casos de uso colaboradores. Casos de uso colaboradores são aqueles que, de alguma forma, contribuem para a realização de um caso de uso “base”. Dentro da *Unified Modeling Language* (UML), essas associações podem ser do tipo <<*include*>> e <<*extend*>>.

O Passo 5 tem o objetivo de identificar os casos de uso candidatos a aspecto. Para isso, os seguintes critérios podem ser utilizados:

- Estar associado a mais de um caso de uso.
- Ser um caso de uso do tipo não funcional.
- Estar associado a um caso de uso por meio do estereótipo <<*extend*>>.
- Permitir alterar o fluxo de execução do caso de uso base sem impedir a sua realização.

Para que um caso de uso seja classificado como candidato a aspecto, basta que ele atenda qualquer um desses critérios.

Por fim, o passo 6, representar os casos de uso candidatos a aspectos, é realizado para representar o Diagrama de Caso de Uso e Descrição Textual dos casos de uso candidatos a aspecto identificados no Passo 5.

O método para identificação de casos de uso candidatos a aspectos deve ser utilizado a partir de um Diagrama de Caso de Uso e Descrição Textual. Torna-se inviável a utilização desse método se esses artefatos não estiverem disponíveis.

2.2 PROJETO ORIENTADO A ASPECTOS

O Projeto Orientado a Aspectos é uma área que possui poucos métodos aplicáveis se comparado ao Projeto Orientado a Objetos (POO). Apesar disso, existem alguns princípios da Engenharia de Software utilizados na Orientação a Objetos que podem ser aplicados na Orientação a Aspectos para melhorar as técnicas de Projeto Orientado a Aspectos (CHAVEZ, 2004). O Quadro 1 ilustra esses princípios.

Princípio	**Definição**
Separação de *Concerns*	Os requisitos do sistema (*concerns*) devem ser isoladamente considerados.
Acoplamento baixo	Um módulo deve comunicar-se com o menor número possível de módulos para realizar suas responsabilidades.
Acoplamento fraco	A comunicação entre dois módulos deve ser realizada utilizando somente as informações necessárias para realizar suas responsabilidades.
Ocultamento de informação	Encapsulamento. As informações de um componente devem ser acessadas por meio de *interfaces*.
Coesão	Componentes que possuem ligação lógica devem ser agrupados.

Quadro 1 – Princípios de Engenharia de Software
Fonte: Adaptado de Chavez (2004)

A Orientação a Aspectos aplica todos os princípios ilustrados no quadro anterior, mas tem o intuito de tratar adequadamente as Preocupações Transversais de um sistema.

A Separação de Preocupações é um dos princípios mais conhecidos dentro da Engenharia de Software. A aplicação correta desse princípio resulta em classes com responsabilidades bem definidas e facilita a evolução dos sistemas.

O uso de aspectos pode minimizar as dependências entre componentes. Essa característica permite que o projeto de software possua baixo acoplamento (CHAVEZ, 2004).

Ocultamento de informação, também conhecido como encapsulamento, recebeu um grande interesse no contexto da POO. Por meio da aplicação de algumas abordagens orientadas a aspectos é possível quebrar o encapsulamento apresentado pela POO. Por outro lado, essas abordagens conseguem tratar comportamentos que, de outra forma, ficariam entrelaçados e espalhados pelo sistema (CHAVEZ, 2004).

A coesão é uma medida da ligação entre os elementos localizados dentro de um mesmo módulo. Quando os aspectos de um módulo estão relacionados à mesma preocupação, isso significa que eles possuem alta coesão. Contudo, quando eles são pouco relacionados, significa que eles possuem baixa coesão (CHAVEZ, 2004).

2.2.1 Programação orientada a aspectos

Dentro do desenvolvimento de software cada projeto apresenta um conjunto de preocupações principais relacionadas ao domínio de negócio dessa aplicação. Da mesma forma, cada aplicação possui seu conjunto de preocupações secundárias. Essas preocupações podem não estar ligadas diretamente ao domínio de negócio, mas são essenciais para garantir a qualidade do software. Dentro da Programação Orientada a Aspectos (POA), essas preocupações secundárias são conhecidas como Preocupações Transversais (*Crosscutting Concerns*).

O conceito de POA foi apresentado pela primeira vez na Conferência Europeia de Programação Orientada a Objetos de 1997, por Gregor Kiczales e outros pesquisadores do Xerox Palo Alto Research Lab, Vale do Silício, Califórnia.

A POA é uma técnica de programação que estende as vantagens oferecidas pela POO.

Enquanto a POO consegue tratar adequadamente as preocupações principais de um software, ela não apresenta a mesma eficiência no tratamento das Preocupações Transversais. Por meio da introdução de uma unidade de encapsulamento chamada aspecto é possível tratar adequadamente esse tipo de preocupação (LADDAD, 2003).

Um simples aspecto pode contribuir para a implementação de vários procedimentos e classes (ELRAD et al., 2001). A POA não surgiu para substituir a POO, mas sim para aperfeiçoá-la e completá-la (ZHENGYAN, 2011).

Há várias linguagens que aplicam a orientação a aspectos, dentre elas estão: *AspectJ* (KICZALES et al., 2001), *AspectC*++ (SPINCZYK et al., 2002), desenvolvida para atender sistemas embarcados, *AspectAda* (PEDERSON; CONSTANTINIDES, 2005), *AspectScript* (TOLEDO et al., 2010), uma extensão orientada a aspectos para *Javascript*, AspectMatLab (ASLAM et al., 2010) e *Arachne* (DOUENCE et al., 2005), utilizada para modularizar preocupações identificadas em serviços de sistemas operacionais.

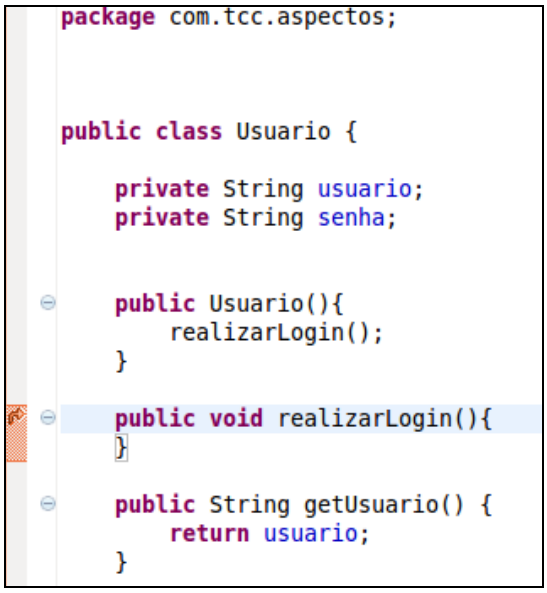
Em seu estudo, Zhengyan (2011) apresenta os conceitos mais importantes da tecnologia POA. Esses conceitos são: Preocupação Transversal, *Join Point,* Aspecto*, Advice, Pointcut* e *Weaving.*

2.2.1.1Preocupação Transversal

A POA surgiu para tentar resolver o problema das preocupações transversais. Esse tipo de preocupação pode ser identificado quando a sua implementação está espalhada por várias classes ou métodos.

2.2.1.2*Join Point*

É uma posição bem definida dentro de um programa, tal como a chamada a de um método ou o lançamento de exceção. A Figura 12 ilustra um exemplo de *join point*.



```
package com.tcc.aspectos;



public class Usuario {

    private String usuario;
    private String senha;


    public Usuario(){
        realizarLogin();
    }

    public void realizarLogin(){
    }

    public String getUsuario() {
        return usuario;
    }
```

Figura 12 - Exemplo de *join point* utilizando AspectJ
Fonte: Autoria Própria

Esta figura mostra um trecho de uma classe chamada *Usuario*. Ela possui dois atributos, um método construtor, métodos *getters* e *setters* e um método chamado realizarLogin(). A seta apontando para esse método indica que um *join point* foi definido nesse método. Essa definição pode ser encontrada no aspecto ilustrado pela Figura 13.

2.2.1.3Aspecto

Um aspecto é semelhante a uma classe. A principal diferença entre eles é que um aspecto tem o objetivo de encapsular as preocupações transversais de um sistema. A Figura 13 ilustra um exemplo de aspecto implementado por meio da linguagem *AspectJ*.

```
package com.tcc.aspectos;

public aspect Logging {

    pointcut log() : execution(* Usuario.realizarLogin(..));

    before() : log(){
        //advice responsável por logging
    }
}
```

Figura 13 - Exemplo de aspecto utilizando *AspectJ*
Fonte: Autoria Própria

Esta figura ilustra um aspecto implementado por meio da linguagem *AspectJ*. Nela é possível observar o *pointcut* 'log()' apontando para *join points* onde ocorre a execução do método 'realizarLogin()' da classe 'Usuario', O sinal de '*' indica que tanto os métodos que retornam algum valor quanto aqueles que não tem retorno serão referenciados por esse *poincut*. A representação '(..)' indica que o método pode ter zero, um ou mais parâmetros. Toda vez que esse método entrar em execução, a implementação do *pointcut* 'log', o *advice* 'log()', será executado.

2.2.1.4*Advice*

Advice é a implementação de uma preocupação transversal. A Figura 13 ilustra um *advice* que realiza *logging*. O *poincut log()* define as condições para a execução desse *advice*. Toda vez que ocorrer a execução do método *realizarLogin(),* o *advice* será executado antes desse método.

2.2.1.5*Pointcut*

Utilizado para apontar onde o *advice* será utilizado. O *pointcut* pode apontar para um ou mais *join points*.

A Figura 13 ilustra um exemplo de *pointcut*. Um *pointcut* pode definir vários *join points*. O aspecto ilustrado nesta figura define um *join point* para toda a execução do método *realizarLogin()* que está dentro da classe *Usuario.*

2.2.1.6*Weaving*

O nome do processo no qual um aspecto é adicionado a um objeto chama-se *weaving* e pode ocorrer em tempo de compilação ou em tempo de execução. A Figura 14 ilustra um modelo desse processo.

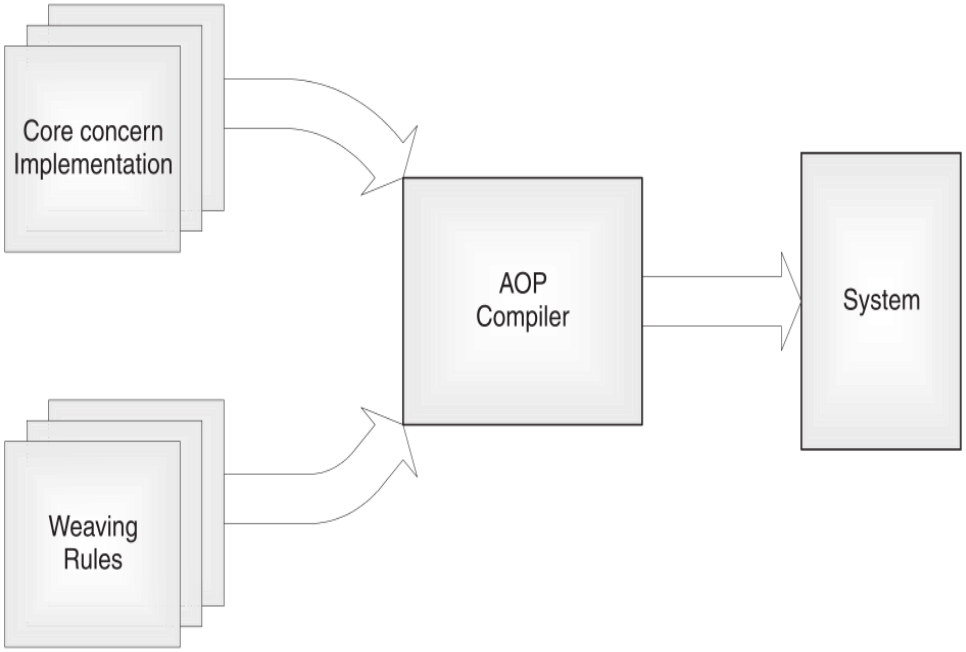


Figura 14 - Modelo de *weaving*
Fonte: Laddad (2003, p. 25)

A figura anterior ilustra um modelo de *weaving* na forma de um compilador. A implementação das preocupações envolve tanto as preocupações principais quanto as transversais.

As regras de *weaving* especificam como integrar as preocupações para formar o sistema final. Por exemplo, uma dessas regras pode informar onde o registro de *log* deve ser executado no sistema. O compilador POA produz um código 'costurado'. Esse código é uma versão executável do sistema.

A Programação Orientada a Aspectos tem sido utilizada em grandes projetos industriais, tais como: *IBM Websphere Application Server*, *JBoss Application Server* e *Siemens' Soarian* (RASHID et al., 2010). Segundo Laddad (2003) algumas das vantagens da POA são:

- Cada módulo possui responsabilidade mais clara: cada módulo é encarregado de apenas uma responsabilidade. Por exemplo, um módulo acessando uma base de dados não é responsável pelo *pool* de conexões do banco de dados.
- Nível maior de modularização: POA fornece mecanismos para tratar cada preocupação separadamente e com o mínimo de acoplamento. Isso resulta em menor nível de código duplicado.

- Evolução do sistema torna-se mais fácil: POA encapsula as preocupações transversais do sistema dentro de aspectos. A partir disso, adicionar uma nova funcionalidade torna-se mais fácil porque não exige que outros módulos sejam alterados. Se a nova funcionalidade for uma preocupação transversal, ela é encapsulada dentro de um aspecto. Caso contrário, ela é encapsulada dentro de uma classe, por exemplo.
- Nível de reuso de código é maior: ao utilizar POA, a tendência é que cada classe possua apenas uma responsabilidade. Consequentemente, o reuso de código torna-se mais fácil, porque cada classe é responsável apenas por uma tarefa.

2.3 ANÁLISE DOS MÉTODOS DE IDENTIFICAÇÃO DE ASPECTOS NA FASE DE ANÁLISE

Os métodos apresentados na Seção 2.2 utilizam diagrama de caso de uso e Descrição Textual para identificação dos possíveis aspectos. Lembrando que isso ocorre na fase de Análise.

O método proposto por Kong e Yuan (2009) propõe a aplicação da metodologia RUP para ser utilizada com caso de uso e descrição textual. O processo desse método é dividido em cinco etapas. Apenas a primeira etapa apresenta possibilidade quase nula de identificar candidatos a aspectos. A identificação de candidatos a aspectos pode ocorrer após a realização de cada etapa ou ao fim das cinco etapas. No primeiro caso, identificar candidatos a aspectos após a realização de cada etapa pode desviar a atenção do responsável pela tarefa de identificação de requisitos. No segundo caso, identificar candidatos a aspectos apenas no fim desse processo irá permitir que essa identificação ocorra com maior conhecimento sobre o domínio do sistema.

O método apresentado por Jacobson e NG (2004) propõe o uso de *Use Case Slice* para encapsular aspectos durante a fase de análise. Os relacionamentos utilizados por casos de uso, tais como <<*extend*>> e <<*include*>>, também podem ser utilizados pelos *Use Case Slices* para modelar os candidatos a aspectos. As dificuldades no uso desse método são:

- Não consegue identificar todos os aspectos do sistema (KOSSOSKI, 2010).
- Exige que o pessoal responsável pela análise tenha conhecimento sobre aspectos.

A proposta de Hornung (2010) propõe a identificação de casos de uso candidatos a aspectos por meio de responsabilidades. O método apresentado realiza as seguintes atividades:

- Obter as responsabilidades a partir da análise das Descrições Textuais.
- Identificar as responsabilidades espalhadas entre os casos de uso.
- Adaptar o Modelo de Caso de Uso para representar, da melhor forma possível, as responsabilidades identificadas.
- Identificar casos de uso colaboradores.
- Definir a relação entre casos de uso base e colaborador.
- Representar o Modelo de Caso de Uso os casos de uso candidatos a aspectos.

Durante a análise deste método identificou-se como pontos de dificuldades o seguinte:

- Cada passo do processo consome muito tempo;
- Difícil de ser aplicada dentro de metodologias Ágeis.

As dificuldades identificadas são fatores determinantes no momento da escolha de um método para identificação de aspectos. O método apresentado nesse trabalho foi desenvolvido com o objetivo de remover essas dificuldades do processo de identificação de aspectos e será descrito no próximo capítulo.

3 MÉTODO PROPOSTO PARA IDENTIFICAÇÃO DE ASPECTOS NA FASE DE ANÁLISE

Este capítulo apresenta o método proposto para identificação de aspectos. A seção 3.1 justifica a criação do método. A Seção 3.2 descreve o processo geral do método proposto explicando sobre: a identificação de requisitos não-funcionais, a criação da matriz de adjacência e como utilizá-la para identificar aspectos, as instruções necessárias para verificar e tratar o espalhamento de aspectos dentro das matrizes adjacentes. Por fim, apresenta as suas vantagens.

3.1 JUSTIFICATIVA DA CRIAÇÃO DO MÉTODO

A Orientação a Aspectos tem sido utilizada para a construção de vários tipos de sistema (MUNNELLY;CLARKE, 2009, TAEHO;HONGCHUL, 2008). Além disso, permite tratar adequadamente as preocupações transversais identificadas dentro de qualquer sistema, diminuindo o entrelaçamento de código e facilitando a evolução do software.

Porém, a carência de métodos para identificação de aspectos na fase de análise de sistemas é uma grande dificuldade enfrentada pelos desenvolvedores que utilizam esse paradigma.

A principal atividade realizada na etapa de análise é a elicitação dos requisitos funcionais e não funcionais do sistema.

Os Requisitos Funcionais compreendem o negócio do cliente, enquanto os não funcionais estão relacionados a padrões de qualidade, tais como: desempenho, confiabilidade, robusteza e outros.

Dentre os métodos de desenvolvimento de software contemplados na literatura atual, os requisitos funcionais do sistema são traduzidos para um conjunto de objetos e suas interações. Enquanto isso, os requisitos não-funcionais podem ser traduzidos para aspectos.

Para justificar essa afirmação, o Quadro 2 apresenta as semelhanças entre aspectos e requisitos não funcionais. O método apresentado nesse trabalho foi desenvolvido a partir dessas similaridades.

Aspecto	Requisito Não Funcional
Aspectos são comportamentos que, quando identificados, estão entrelaçados e espalhados pelo domínio de aplicação.	Estão espalhados pelos requisitos funcionais do sistema.
Encapsula vários tipos de atributos, dentre os quais estão os de qualidade do sistema.	Estão relacionados a padrões de qualidade do sistema.
Aumenta o nível de satisfação do desenvolvedor, pois propicia um sistema com maior nível de manutenibilidade e propício a evolução.	Aumenta o nível de satisfação do cliente.
Vários estudos destacam a importância dos aspectos para o desenvolvimento de software.	Requisitos não-funcionais são requisitos mais críticos que requisitos funcionais.
Não é possível desenvolver um sistema apenas com aspectos.	Requisitos não funcionais existem apenas na presença de requisitos funcionais.
Costumam ser identificados sobre um sistema que já foi aplicado em um ambiente de produção.	Costumam ser identificados sobre um sistema que já foi aplicado em um ambiente de produção.

Quadro 2 - Semelhanças entre aspectos e requisitos não-funcionais
Fonte: Autoria Própria

A próxima seção apresenta uma visão geral do método proposto.

3.2 PROCESSO GERAL DO MÉTODO PROPOSTO

Esta seção apresenta o funcionamento do método proposto. A Figura 15 ilustra um diagrama de atividades UML mostrando as atividades realizadas pelo método proposto.

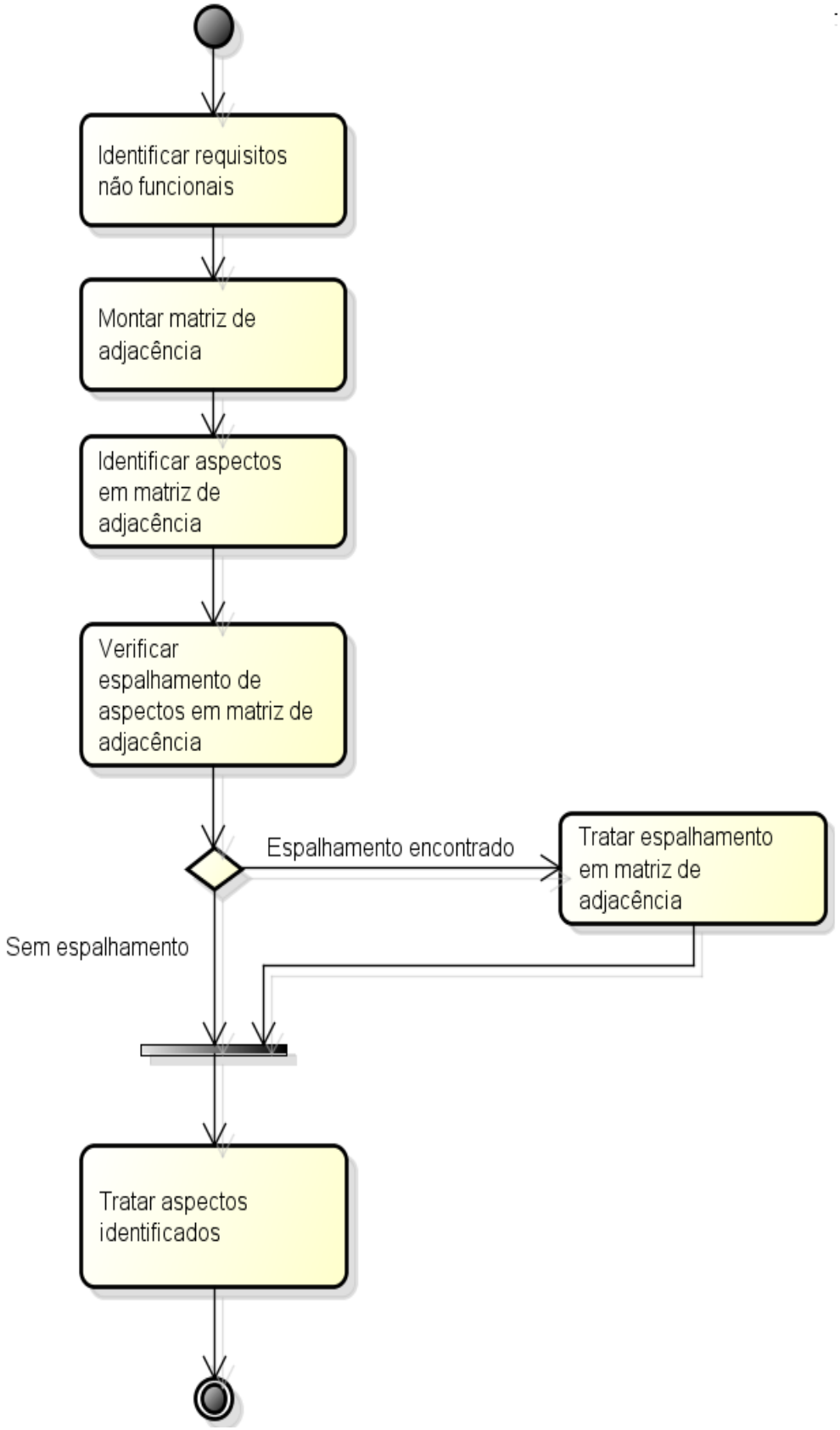


Figura 15 - Processo geral do método proposto
Fonte: Autoria Própria

A atividade *Identificar requisitos não funcionais* é responsável pela identificação dos requisitos não-funcionais que serão utilizados para o levantamento dos aspectos.

A atividade *Montar matriz de adjacência* é realizada para preencher as matrizes de adjacência com os passos das descrições de caso de uso.

A atividade *Identificar aspectos em matriz de adjacência* é responsável pelo preenchimento da matriz de adjacência com os passos das descrições de caso de uso.

A próxima atividade, *Verificar espalhamento de aspectos em matriz de adjacência,* é realizada para examinar se um mesmo aspecto foi identificado em mais de uma matriz de adjacência. Caso algum espalhamento seja identificado, a atividade *Tratar espalhamento em matriz de adjacência* será requisitada e executada.

Por fim, a atividade *Tratar aspectos identificados* é realizada para representar os aspectos dentro das descrições textuais de caso de uso.

A seguir se detalha cada uma das atividades do processo geral do método proposto.

3.2.1 Identificar requisitos não funcionais

O conceito sobre requisitos não-funcionais ainda não possui um consenso, mas sabe-se que esse tipo de requisito é tão importante quanto requisitos funcionais. Se os requisitos não-funcionais de um sistema não são atendidos, a qualidade do produto final estará comprometida. Caso o cliente exija explicitamente que um requisito não-funcional esteja presente no sistema, esse requisito passa a ser um requisito funcional (XAVIER; 2009).

Os requisitos funcionais representam o que o cliente deseja que o software faça. Por sua vez, os requisitos não-funcionais permitem ao software realizar, de forma satisfatória, o que deve ser feito. Apesar disso, os requisitos não-funcionais costumam ser negligenciados. Isso ocorre porque eles têm natureza mais abstrata e são descritos de forma breve e vaga (CYSNEIROS; 2001, CHUNG et al.; 1999).

Apesar de sua importância, o número de métodos para identificação de requisitos não funcionais contemplados na literatura atual é muito inferior ao número de métodos para identificação de requisitos funcionais.

As técnicas apresentados por Gastaldo e Midorikawa (2003) e Xavier (2009) podem ser utilizados para identificar os requisitos não funcionais utilizados pelo método proposto. Porém, o presente trabalho utiliza os atributos relacionados aos requisitos não funcionais identificados no estudo de Mairizia et al. (2010), descritos a seguir.

3.2.1.1Identificação de Atributos dos Requisitos Não Funcionais

O estudo desenvolvido por Mairizia et al.(2010) identifica cinco requisitos não funcionais mais frequentemente usados no desenvolvimento de software: Desempenho, Confiabilidade, Segurança, Usabilidade e Manutenibilidade. Esses requisitos são compostos por vários atributos, os quais estão exibidos no Quadro 3.

RNF	Definição	Atributos
Desempenho	Especifica a capacidade do software para fornecer desempenho apropriado, em relação à quantidade de recursos necessária para realizar todas as funcionalidades sob determinadas condições.	Tempo de resposta, capacidade, latência, *throughput,* processamento, velocidade de execução, *transit delay,* carga de trabalho, utilização de recursos, uso de memória, precisão, eficiência, *compliance, modes, delay, miss rates,* perda de dados, processamento de transação concorrente.
Confiabilidade	Especifica a capacidade do software para operar sem falhas e manter um nível especificado de desempenho quando utilizado sob condições normais durante um determinado período de tempo.	Completude, precisão, consistência, disponibilidade, integridade, corretitude, maturidade, tolerância a falhas, *recoverability,* confiabilidade, *compliance, failure rate/ critical failure.*
Segurança	Requisitos sobre a prevenção de acesso não autorizado ao sistema, programas e dados.	Confidencialidade, integridade, disponibilidade, controle de acesso, autenticação.
Usabilidade	Especifica as interações do usuário final com o sistema e o esforço necessário para aprender, operar, preparar entrada de dados e interpretar a resposta do sistema.	Capacidade de ser aprendido, compreensibilidade, *operabilidade, attractiveness,* capacidade de ser usável, *compliance,* facilidade de uso, engenharia humana, amigável ao usuário, fácil memorização, eficiência, *user productivity, usefulness, likeability,* tempo de reação do usuário.
Manutenibilidade	Descreve a capacidade do software para ser modificado que incluam: corrigir um defeito ou realizar uma melhora no software.	Capacidade de ser testado, compreensibilidade, capacidade de ser modificável, capacidade de ser analisado, *stability, maintainability compliance*.

Quadro 3 - Requisitos Não-Funcionais e seus atributos
Fonte: Adaptado de Mairizia et al. (2010)

Este quadro apresenta os RNFs e seus atributos contemplados no estudo de Mairizia et al. (2010), os quais foram obtidos por meio de pesquisa em diversas fontes, tais como: artigos, revistas e livros. O método proposto sugere o uso destes requisitos não funcionais e seus atributos para realizar a identificação de aspectos.

Com os RNFs identificados, o método proposto sugere que os seus respectivos atributos sejam distribuídos por linha e cada um receba uma sigla correspondente. Por exemplo, o Quadro 4 apresenta esta divisão para o RNF de desempenho. Para os outros RNFs descritos no Quadro 3 a divisão se encontra no Apêndice A.

Desempenho	
D_1	Tempo de resposta
D_2	Capacidade
D_3	Latência
D_4	*Throughput*
D_5	Processamento
D_6	Velocidade de execução
D_7	*Transit delay*
D_8	Carga de trabalho
D_9	Utilização de recursos
D_{10}	Uso de memória
D_{11}	Precisão
D_{12}	Eficiência
D_{13}	*Delay*
D_{14}	Perda de dados
D_{15}	Processamento de transação concorrente
D_{16}	*Garbage Collector*

Quadro 4 - Requisito Não-Funcional Desempenho
Fonte: Adaptado de Mairizia et al.(2010)

A identificação de RNF do tipo desempenho para desenvolver software orientado a aspectos pode parecer contraditório. Essa contradição existe porque software orientado a aspectos tende a apresentar maior troca de fluxo de controle durante sua execução. Quanto maior o número de *join points* identificados dentro do sistema, maior será essa troca de fluxo. Apesar disso, software orientado a aspectos apresenta um nível de desempenho semelhante ou, em determinados casos, maior do que aqueles que não são orientados a aspectos (WEN-LIN et al., 2011).

O encapsulamento desse tipo de requisito dentro de aspectos trará vantagens proporcionadas pela orientação a aspectos além de permitir o aumento de desempenho do software. A Figura 16 ilustra um aspecto *Garbage Collector* utilizando AspectC++, o qual melhora o desempenho de software.

```
aspect GCMarkSweep {

 advice (execution("% ...::operator
 new(...)")): after (){
   GCMarkSweep::aspectof()->
   addPointer(…);

  advice (execution("% ...::operator
  delete(void*)") || execution("%
  operator delete(void*)")): before(){
    GCMarkSweep::aspectof()->
    removePointer(…); }
}
```

Figura 16 - *Garbage Collector* implementado em AspectC++
Fonte: Mcheick; Sioud (2009)

O aspecto *GCMarkSweep* é responsável por remover da memória os objetos que não possuem referência para si mesmo.

A próxima seção descreve a atividade de *Montar matriz de adjacência* e como ela é utilizada dentro do método proposto.

3.2.2 Montar matriz de adjacência

A próxima atividade a ser realizada é a montagem de matriz de adjacência. Para isso é necessário que o analista possua tanto os requisitos não-funcionais quanto as descrições textuais de caso de uso do sistema. Esse trabalho utiliza os requisitos não-funcionais provenientes de Mairizia et al. (2010).

Matriz de adjacência é uma maneira de representar um grafo. Ela pode ser utilizada para identificar vulnerabilidades em redes de computadores (NOEL;JAJODIA, 2005, TAO LONG et al., 2010), mas para o método proposto essas matrizes serão usadas da seguinte forma:

- As linhas possuem os passos dos fluxos da descrição de caso de uso.
- As colunas contêm os atributos dos requisitos não funcionais.
- Quando uma intersecção for marcada, significa que o atributo daquela coluna é identificado no passo daquela linha. Isso também representa um aspecto.

O Quadro 5 ilustra um modelo dessa matriz de adjacência

Título da matriz					
	A_1	A_2	A_3	$A_{...}$	A_n
TF_1 P_1					
$TF_{...}$ $P_{...}$					
TF_m P_t					

Quadro 5 – Modelo de matriz de adjacência
Fonte: Autoria Própria

A variável ‘Título da matriz’ representa o nome da descrição textual. A variável ‘P’ refere-se a um passo da descrição textual de caso de uso, ‘TF’ representa um tipo de fluxo e a variável ‘A’ representa um atributo de requisito não funcional. Onde:

- TF_1 ... TF_m: representam os tipo de fluxo. Os valores possíveis são FB (fluxo básico ou principal), FA (fluxo alternativo) e S (subfluxo), sendo que um subfluxo pode conter um ou mais fluxos alternativos. Nesse caso, utiliza-se o número de tipos de fluxo necessários para representar cada passo. Por exemplo, se um subfluxo possuir dois fluxos alternativos, cada fluxo alternativo será representado por $S_1A_1P_1$ e $S_1A_2P_1$.
- P_1 ... P_t: representam os passos do fluxo, sendo *m* o total de passos.

Esta matriz é base para a próxima atividade de identificação de aspectos.

3.2.3 Identificar aspectos em matriz de adjacência

O próximo passo após a montagem das matrizes de adjacência com os requisito não-funcionais elicitados é a identificação de aspectos e possíveis espalhamento deles. Para demonstrar isso, serão utilizadas duas descrições de caso de uso sobre um sistema de gerenciamento de custos de obra civil. O Quadro 6 apresenta a primeira descrição textual.

UC001	Realizar Cadastro de Obra
Breve Descritivo:	Este Caso de Uso descreve o processo de cadastramento de obra.
Fluxo Principal	
P1- O usuário seleciona a opção de cadastrar obra.	
P2- O sistema disponibiliza um cadastro de obra para o usuário.	
P3- O usuário informa os dados.	
P4- O sistema valida os dados.	
P5- O usuário tenta salvar o cadastro.	
P6- O sistema salva o cadastro e exibe uma mensagem de sucesso.	

Quadro 6 – Descrição textual Realizar Cadastro de Obra
Fonte: Autoria Própria

Esse quadro descreve o procedimento realizado pelo usuário para cadastrar uma obra no sistema.

O analista deve verificar a presença de cada RNFs com seus respectivos atributos dentro dos passos da descrição textual. Se um atributo for identificado em algum passo, esse deve ser marcado e irá representar um aspecto dentro daquele passo. Este processo deve ser repetido para todos os RNFs levantados. O Quadro 7 apresenta o resultado desse processo considerando os atributos referentes ao RNF de Desempenho.

Cadastro de Obra								
	D_1	$D_{...}$	D_5	D_6	$D_{...}$	D_{11}	$D_{...}$	D_{15}
FB P_1								
FB P_2	X			X				
FB P_3				X				
FB P_4				X		X		
FB P_5	X		X	X		X		
FB P_6								X

Quadro 7 – Matriz de adjacência para a descrição textual Realizar Cadastro de Obra
Fonte: Autoria Própria

Este quadro ilustra os aspectos identificados na descrição textual 'Realizar Cadastro de Obra'. Os aspectos identificados no 'FB P_2' foram D_1(tempo de resposta) e D_6 (velocidade de execução). Dentro do 'FB P_3' o aspecto D_1 (Tempo de

resposta) foi identificado. Os aspectos $D_{1,}$ D_5 (Processamento), D_6 e D_{11} (Precisão) foram identificados no ’FB P_5’. Por fim, o aspecto D_{15} (Processamento de transação concorrente) foi identificado no ’FB P_6’.

O Quadro 8 ilustra o segundo exemplo de descrição textual apresentada nessa seção.

UC004	**Consultar Cadastro de Obra**
Breve Descritivo:	Este Caso de Uso descreve o processo da consulta de um cadastro de obra.
Cenários Principais	
P1 - O usuário insere algum dado sobre a obra para recuperar o cadastro da mesma.	
P2 - O usuário tenta consultar o cadastro da obra.	
P3 - O sistema recupera, a partir do dado inserido, o cadastro e disponibiliza-o para o usuário.	

Quadro 8 – Descrição textual Consultar Cadastro de Obra
Fonte: Autoria Própria

Este quadro descreve o procedimento realizado pelo usuário para ‘Consultar Cadastro de Obra’. O Quadro 9 apresenta a matriz de adjacência dessa descrição textual após o processo de identificação de aspectos.

Consulta de Cadastro de Obra								
	D_1	$D_{...}$	D_5	D_6	$D_{...}$	D_{11}	$D_{...}$	D_{15}
FB P_1								
FB P_2								
FB P_3	X			X				X

Quadro 9 – Matriz de adjacência para a descrição textual Consultar Cadastro de Obra
Fonte: Autoria Própria

Este quadro ilustra os aspectos identificados na descrição textual Consultar Cadastro de Obra. A próxima seção apresenta uma maneira de melhorar o uso dessas matrizes adjacentes.

3.2.4 Verificar espalhamento de aspectos em matriz de adjacência e tratar espalhamento

Um fenômeno que provavelmente irá ocorrer nas matrizes adjacentes é o espalhamento de aspectos. Por exemplo, dentro das matrizes ‘Cadastro de Obra’ e ‘Consulta de Cadastro de Obra’ ocorre o espalhamento de aspectos, porque ambas as matrizes possuem um ou mais aspectos em comum. A Figura 17 ilustra esse espalhamento.

Figura 17 - Espalhamento de aspectos em matrizes adjacentes
Fonte: Autoria Própria

Os aspectos espalhados são: D_1, D_6 e D_{15}. Neste caso, realizar-se-á essa atividade. Caso contrário, pode-se passar para a atividade “Tratar Aspectos Identificados”.

Um aspecto pode estar presente em mais de um passo da mesma descrição textual. Da mesma forma, um determinado aspecto pode ser identificado em várias descrições textuais.

O espalhamento de aspectos dentro de matriz de adjacência será considerado para facilitar o uso da matriz de adjacência nas atividades posteriores do desenvolvimento de software.

O espalhamento será tratado com o uso de uma matriz de adjacência auxiliar. O Quadro 10 ilustra esse tipo de matriz.

Título da matriz auxiliar				
	A_1	A_2	$A_{...}$	A_n
$UC_1 – TF_1\ P_1$				
$UC_{...} – TF_{...}\ P_{...}$				
UC_x - $TF_m\ P_t$				

Quadro 10 – Matriz auxiliar utilizada para tratar espalhamento de aspectos
Fonte: Autoria Própria

A matriz apresenta pequenas diferenças se comparada àquela ilustrada na seção 3.2.2. A variável ‘UC’ (*Use Case*) representa a identificação do caso de uso, enquanto a variável ‘P’ refere-se a um passo da descrição textual e TF representa o tipo de fluxo. Onde:

- A_1...A_n: representam os atributos identificados na primeira atividade, onde *n* é o total de atributos para cada tipo de RNFs;
- UC_1...$UC_{x:}$ representa para qual caso de uso (*Use Case*) o passo pertence, onde *x* é o total de casos de uso;
- P_1...$P_{t:}$ representam os passos do fluxo básico dos casos de uso, sendo *y* o total de passos;
- TF_1 ... TF_m: representam os tipos de fluxo. Os valores possíveis são FB (fluxo básico ou principal), FA (fluxo alternativo) e S (subfluxo), sendo que um subfluxo pode conter um ou mais fluxos alternativos.

O número de matrizes auxiliares será proporcional ao número de requisitos não-funcionais.

A verificação de espalhamento de aspectos será realizada após a atividade ilustrada pela seção 3.2.3 (*Identificar aspectos em matriz de adjacência*) ter sido realizada para todas as descrições textuais de caso de uso. Neste exemplo, a matriz auxiliar será criada para o requisito não-funcional desempenho.

Essa matriz irá conter os aspectos que se encontram espalhados pelas matrizes de adjacência do RNF. A verificação e o tratamento dos aspectos espalhados em matrizes de adjacência serão realizados a partir dessa matriz. O diagrama de atividades apresentado pela Figura 18 ilustra esse subprocesso.

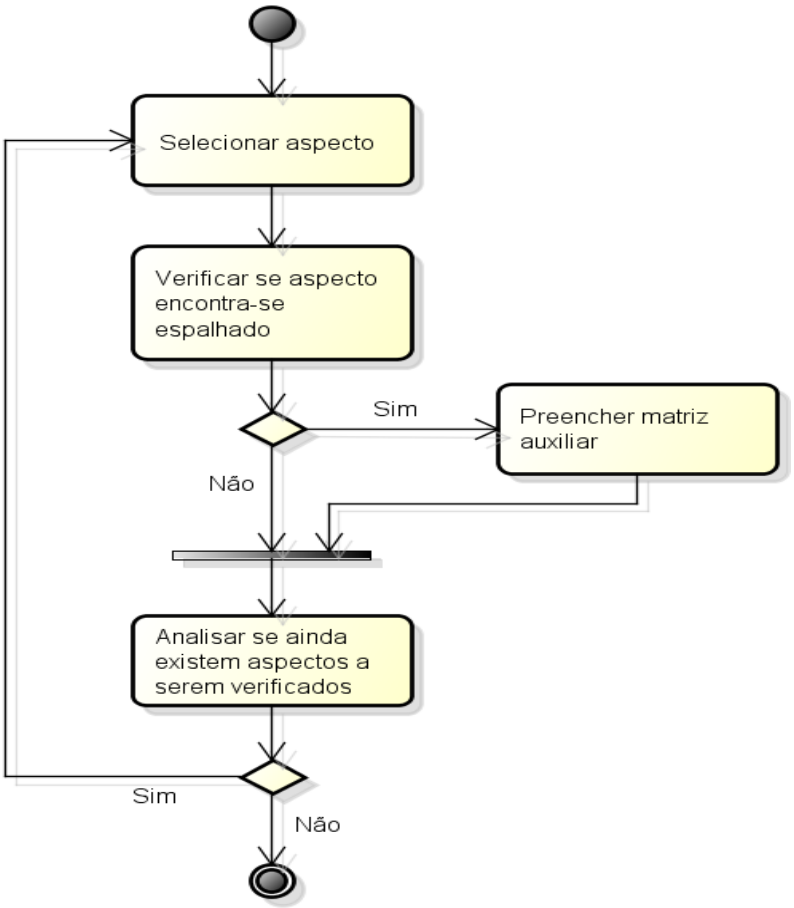


Figura 18 – Subprocesso para identificar e tratar espalhamento de aspectos em matriz de adjacência

Fonte: Autoria Própria

A subatividade *Selecionar aspecto* consiste em selecionar um dos aspectos identificados nas matrizes de adjacência. Nesse caso, o primeiro aspecto selecionado será D_1.

A subatividade *Verificar se aspecto encontra-se espalhado* consiste em analisar se o aspecto selecionado foi identificado em mais de uma matriz de adjacência. Por exemplo, o atributo de aspecto D_1 será verificado em cada matriz de adjacência.

A subatividade *Preencher matriz auxiliar* é realizada somente se foi identificado espalhamento do atributo de aspecto. A matriz auxiliar deverá registrar o nome do caso de uso, o passo no qual o aspecto foi identificado e o próprio aspecto. Após inserir essas informações na matriz auxiliar, esse passo pode ser removido da sua matriz original. Ou seja, a matriz original deverá conter apenas aspectos que não apresentam a característica de espalhamento, enquanto que a matriz auxiliar irá conter apenas aspectos que possuem essa característica. Isso pode resultar no

desaparecimento de algumas matrizes originais, mas isso se torna irrelevante, porque seus aspectos já terão sido movidos para alguma matriz auxiliar.

A subatividade *Analisar se ainda existem aspectos a serem verificados* é realizada para verificar se ainda existem atributos de aspectos que devem passar por esse subprocesso. Por exemplo, após verificar o atributo de aspecto D_1, deve-se averiguar se ainda restam atributos de aspectos do tipo desempenho a serem verificados. Isso se repete para todos os atributos de aspectos identificados nas matrizes de adjacência. Esse trabalho considera indiferente a ordem com que esses atributos são verificados.

O Quadro 11 ilustra o resultado final da realização do subprocesso ilustrado pela Figura 18 considerando os casos de uso ‘Cadastro de Obra’ e ‘Consultar Cadastro Obra’.

Matriz auxiliar – Aspectos do Tipo Desempenho								
	D_1	$D_{...}$	D_5	D_6	$D_{...}$	D_{11}	$D_{...}$	D_{15}
Cadastro de Obra – FB P_2	X			X				
Cadastro de Obra – FB P_3				X				
Cadastro de Obra – FB P_4				X		X		
Cadastro de Obra – FB P_5	X		X	X		X		
Cadastro de Obra – FB P_6								X
Consultar Cadastro de Obra – FB P_3	X			X				X

Quadro 11 – Matriz auxiliar preenchida com aspectos do tipo desempenho
Fonte: Autoria Própria

Esse quadro apresenta os aspectos do tipo desempenho que foram identificados em mais de uma matriz de adjacência. Se nenhum espalhamento for identificado, essa matriz auxiliar não existirá. A próxima seção apresenta a última atividade do método proposto.

3.3 TRATAR ASPECTOS IDENTIFICADOS

As entradas para essa atividade são as matrizes originais e as matrizes auxiliares criadas nas atividades das seções 3.2.3 e 3.2.4, respectivamente. Após a realização da subatividade *Preencher matriz auxiliar* é possível que algumas

matrizes de adjacência fiquem vazias. Nesse caso, elas devem ser eliminadas. Essa atividade demonstra como representar os aspectos dentro das descrições textuais de caso de uso. O Quadro 12 ilustra o modelo de descrição textual utilizado para tal representação. Lembrando que se for usado as matrizes de espalhamento geradas na seção 3.2.4 a quantidade total de passos básicos será *y* e não *m* e para os fluxos alternativos é *w* e não *t*.

<<Identificador do caso de uso>>	**<<Título da descrição textual>>**
<< Breve descritivo >>	
<<Tipo de fluxo>>	
TF_1 P_1 << descrição do passo >> <<*Join point* >>	
$TF_{...}$ $P_{...}$ << descrição do passo >> <<*Join point* >>	
TF_m P_t << descrição do passo >> <<*Join point* >>	
Tipos de aspectos	
<<Tipo de aspecto$_1$ >>	
<<Tipo de aspecto$_{...}$ >>	
<<Tipo de aspecto$_j$ >>	

Quadro 12 – Descrição textual utilizada para representar os aspectos identificados
Fonte: Autoria Própria

Este quadro apresenta o modelo de descrição textual que será utilizada para representar os aspectos identificados na matriz de adjacência. Esse modelo é composto por:

- Identificador do caso de uso: um código que identifique o caso de uso.
- Título da descrição textual: representa o nome do caso de uso.
- Breve descritivo: informações sobre a finalidade do caso de uso.
- Tipo de fluxo: indica se o fluxo é do tipo principal ou alternativo. Na presença dos dois tipos, deve-se ilustrar o fluxo alternativo após o fluxo principal.
- P_1 ... P_t: representam os passos dos casos de uso, sendo *m* o total de passos.
- TF_1 ... TF_m: representam os tipos de fluxo. Os valores possíveis são FB (fluxo básico ou principal), FA (fluxo alternativo) e S (subfluxo).
- Descrição do passo: diz respeito à descrição do passo.

- *Join point:* utilizado para representar a ocorrência de aspecto dentro do respectivo passo.
- Tipo de aspecto$_{1\ ...}$ Tipo de aspecto$_{j:}$ representa os tipos de aspectos que foram identificados nessa descrição textual.

O conceito de *join point e pointcut* utilizado dentro do método proposto é semelhante àquele usado na programação orientada a aspectos. A única diferença é que dentro do método proposto os aspectos não são codificados, apenas representados.

Um *join point* refere-se a um passo da descrição textual, enquanto que um *pointcut* diz respeito a um tipo de aspecto. O Quadro 13 ilustra a estrutura de um *pointcut* utilizado por esse método.

<<Tipo do aspecto >>	
$UC_1 – TF_1\ P_1$	<<nome do atributo>>
$UC_{...} – TF_{...}\ P_{...}$	<<nome do atributo>>
$UC_a – TF_m\ P_t$	<<nome do atributo>>

Quadro 13 – *Poincut* representando um tipo de aspecto
Fonte: Autoria Própria

Neste quadro um *poincut é* utilizado para representar um tipo de aspecto. Esse *pointcut* é composto por:

- $UC_1...UC_{a:}$ representa a qual caso de uso (*Use Case*) o passo pertence, onde *a* é o total de casos de uso.
- $P_1...P_{t:}$ representam os passos do fluxo dos casos de uso, sendo *y* o total de passos.
- TF_1 ... TF_m: representam os tipos de fluxo. Os valores possíveis são FB (fluxo básico ou principal), FA (fluxo alternativo) e S (subfluxo), sendo que um subfluxo pode conter um ou mais fluxos alternativos.

A seguir será ilustrado um exemplo de uso utilizando o conteúdo da descrição textual *Consultar Cadastro de Obra* e seus aspectos identificados. O Quadro 14 ilustra os aspectos representados dentro da descrição textual.

UC004	Consultar Cadastro de Obra
Breve Descritivo:	Este Caso de Uso descreve o processo da consulta de um cadastro de obra.
Fluxo Principal	
P1- O usuário insere algum dado sobre a obra para recuperar o cadastro da mesma.	
P2- O usuário tenta consultar o cadastro da obra.	
P3- O sistema recupera, a partir do dado inserido, o cadastro e disponibiliza-o para o usuário. ***Join point***	
Tipos de aspectos	
Desempenho	

Quadro 14 – Representação de aspectos dentro de descrição textual
Fonte: Autoria Própria

A seção 'Tipos de aspectos' da descrição textual ilustrada pelo Quadro 14 indica que existem aspectos do tipo desempenho dentro dessa descrição textual. O Quadro 15 apresenta detalhes sobre os aspectos identificados.

Aspecto Desempenho	
Consultar Cadastro de Obra – FB P_3	$D_{1.}$
Consultar Cadastro de Obra – FB P_3	$D_{6.}$
Consultar Cadastro de Obra – FB P_3	$D_{15.}$

Quadro 15 – Detalhes sobre os aspectos identificados
Fonte: Autoria Própria

Esse quadro apresenta quais aspectos foram identificados na descrição textual 'Consultar Cadastro de Obra'.

Após a realização dessa atividade, os aspectos identificados estão preparados para serem utilizados ao longo das próximas atividades do ciclo de desenvolvimento de software.

3.4 VANTAGENS DO MÉTODO PROPOSTO

A aplicação do método proposto não exige que os analistas tenham conhecimento sobre aspectos, uma vez que esses são identificados a partir das similaridades com requisitos não-funcionais.

Por utilizar descrições de casos de uso, a aplicação do método proposto torna-se menos trabalhosa, pois essas descrições geralmente estão prontas no momento da aplicação do método.

A montagem e preenchimento das matrizes de adjacência são atividades fundamentais e exigem grande atenção do analista, pois a identificação de aspectos depende do modo como essas atividades são realizadas.

O próximo capítulo apresenta a aplicação do método proposto em um estudo de caso.

4 APLICAÇÃO DO MÉTODO PROPOSTO

Este capítulo descreve a aplicação do método proposto o qual têm como objeto de entrada as descrições textuais de caso de uso. As descrições textuais utilizadas são provenientes de trabalhos desenvolvidos pelo Grupo de Pesquisa em Engenharia de Software (GPES). A seção 4.1 justifica a escolha dessas descrições. A seção 4.2 apresenta detalhadamente a aplicação do método sobre apenas duas descrições textuais. Por fim, a seção 4.3 mostra um resumo dos atributos de aspectos identificados em todas as descrições textuais.

4.1 JUSTIFICATIVA PARA A ESCOLHA DAS DESCRIÇÕES TEXTUAIS

O objetivo do Grupo de Pesquisa em Engenharia de Software (GPES) é estudar a aplicação da engenharia de s*oftware* no tema Engenharia de Produção. Dentro desse contexto, busca-se aprofundar sua aplicação na apresentação de novas metodologias voltadas para gestão, melhoria e otimização do processo produtivo.

Atualmente o GPES está desenvolvendo um *framework* de domínio para formação de preço de venda chamado *FrameMK*. As áreas de estudo relacionadas à engenharia de software são: acessibilidade, linhas de produto, *frameworks*, especificação formal e aspectos.

Vários estudos têm sido realizados dentro da área de aspectos, foco deste trabalho, sendo eles:

- Sistemas de *Login*, Cadastro de Usuário e Fale Conosco (TORRENS; MATOS, 2010): esses sistemas fazem parte do *framework* de formação de preço de venda e foram desenvolvidos considerando questões como usabilidade e confiabilidade do usuário. Os aspectos identificados e desenvolvidos passaram por nenhum processo de modelagem. Esses sistemas foram desenvolvidos para o ambiente *desktop*.
- Modelagem de Sistemas baseados em aspectos (KOSSOSKI, 2010): esse trabalho apresenta métodos para modelagem de aspectos. Os

mesmos são aplicados sobre os sistemas desenvolvidos por Torrens e Matos (2010).

As descrições textuais utilizadas para a aplicação do método proposto foram retiradas do trabalho de Kossoski (2010). A justificativa para essa escolha reside no fato de que essas descrições textuais foram utilizadas por outros métodos de identificação de aspectos. A partir dessa semelhança foi possível comparar os resultados obtidos pelo método proposto e os atingidos pelos métodos apresentados em Kossoski (2010).

Além disto, o resultado alcançado pelo método proposto também será incorporado nos sistemas de Login, Fale Conosco e Cadastro de Usuário, melhorando o entrelaçamento tanto de análise quanto de código.

4.2 IDENTIFICAÇÃO DE ASPECTOS UTILIZANDO O MÉTODO PROPOSTO

Esta seção descreve o processo de identificação de aspectos. Cada subseção representa uma atividade do método proposto e irá descrever o que foi realizado.

4.2.1 Identificar requisitos não funcionais

A identificação de requisitos não funcionais pode ser realizada por meio do uso de qualquer método disponível na literatura. Para essa aplicação do método utilizou-se os requisitos não-funcionais relatados no trabalho de Mairizia et al. (2010). Os requisitos não-funcionais e seus atributos estão disponíveis no Apêndice A.

O trabalho apresentado por Mairizia et al.(2010) ilustra os requisitos-não funcionais normalmente identificados dentro do desenvolvimento de software. Por esse motivo, optou-se por utilizar seu trabalho como abordagem para identificação de requisitos não-funcionais.

4.2.2 Montar matriz de adjacência

A montagem de matriz de adjacência foi realizada usando as onze descrições textuais presentes no trabalho de Kossoski (2010). São elas:

- Sistema de Login: Logar no Sistema, Preencher Campos Obrigatórios e Conferir Login.
- Sistema de Cadastro de Usuários: Cadastrar Usuário, Preencher Campos Obrigatórios, Validar Campos Obrigatórios e Validar Dados Específicos.
- Sistema Fale Conosco: Enviar Mensagem, Preencher Campos Obrigatórios, Validar Campos Obrigatórios e Validar Dados Específicos.

Nesta e nas outras atividades de aplicação do método será exibido apenas o resultado da análise de duas das onze descrições, a saber, *Logar no Sistema* e *Conferir Login*, sendo que o resultado final para todas as descrições se encontra ilustrado no Apêndice E. O Quadro 16 ilustra a primeira descrição textual, Logar no Sistema.

Descrição do Caso de Uso: Logar no Sistema
Descrição breve Este caso de uso inicia quando o usuário executa a aplicação.
Fluxo Básico **B1. Logar no Sistema** O caso de uso inicia quando um usuário quer acessar o sistema. 1. O sistema pede que o usuário informe usuário e senha. 2. O usuário digita o seu nome de usuário e sua senha no sistema e clica no botão logar. 3. O sistema verifica a inserção dos dados através de Conferir Login. 4. O sistema autoriza o acesso do usuário, mostrando-lhe a tela inicial do sistema com suas configurações pessoais. 5. O caso de uso termina. **Fluxo Alternativo** **A1. Preenchimento de Campos** No passo 3 caso ocorra algum erro no preenchimento de campos obrigatórios ou na validação dos dados informados, o sistema limpará os campos e mostrará a tela de *login* novamente.

Quadro 16 – Descrição textual Logar no Sistema
Fonte: Adaptado de Kossoski (2010, p. 67)

Esse quadro ilustra uma descrição textual detalhando o processo de autenticação de usuário para acesso a um determinado sistema. A realização do Passo 3 utiliza outro caso de uso, *Conferir Login*, para validar os dados do usuário e realizar a autenticação do mesmo.

O Quadro 17 apresenta a segunda descrição utilizada para aplicação do método proposto. Essa descrição textual contêm os passos realizados para validar as informações inseridas pelo usuário durante o processo de autenticação do sistema.

Descrição do Caso de Uso: Conferir Login
Descrição Breve: O caso de uso verifica se os campos usuário e senha que foram informados pelo usuário correspondem as informações inseridas no banco de dados.
Fluxo básico ... **Subfluxos** **S1. Conferir Login** 1. O sistema verifica se usuário informado possui mais que cinco caracteres incluindo letras e números. 2. O sistema executa um algoritmo de *hash* na senha informada. 3. O sistema verifica se usuário e senha submetidos ao banco de dados correspondem aos dados de entrada. **Precondições** Dados do usuário. **Pós-Condições** Usuário e senha validados. **Fluxos Alternativos** **S1-A1. Problemas na Validação** No passo 1 se usuário informado não possuir letra(s) e número(s) e mais que cinco caracteres no total, será alertado ao usuário este problema. No passo 3 se os dados informados não correspondem no banco de dados, será mostrada uma mensagem alertando que as informações digitadas são incorretas ou não existem; o caso de uso termina.

Quadro 17 – Descrição textual Conferir Login
Fonte: Kossoski (2010, p. 68-69)

A partir dessas duas descrições textuais as matrizes de adjacência foram montadas. Dentro dessa atividade essas matrizes ainda estão vazias. Por esse motivo, optou-se por exibi-las na próxima seção, onde ocorre a identificação de aspectos.

4.2.3 Identificar aspectos em matriz de adjacência

Após realizar a montagem das matrizes de adjacência a atividade para identificação de aspectos foi executada.

A matriz ilustrada pelo Quadro 18 apresenta os atributos de aspectos do tipo segurança identificados para o caso de uso *Logar no Sistema*.

Logar no Sistema - Segurança					
	S_1	S_2	S_3	S_4	S_5
FB P_1					
FB P_2			X	X	
FB P_3	X	X			X
FB P_4	X				
FB P_5					
FA_1P_1	X	X			X

Quadro 18 – Matriz de adjacência Segurança para a descrição Logar no Sistema
Fonte: Autoria Própria

Este quadro ilustra uma das matrizes de adjacência representando os atributos de aspectos identificados. As outras matrizes estão disponíveis no Apêndice B.

4.2.4 Verificar espalhamento de aspectos em matriz de adjacência

Durante a realização dessa atividade observou-se o espalhamento de vários atributos de aspectos. A Figura 19 ilustra o espalhamento dos atributos de aspecto do tipo segurança.

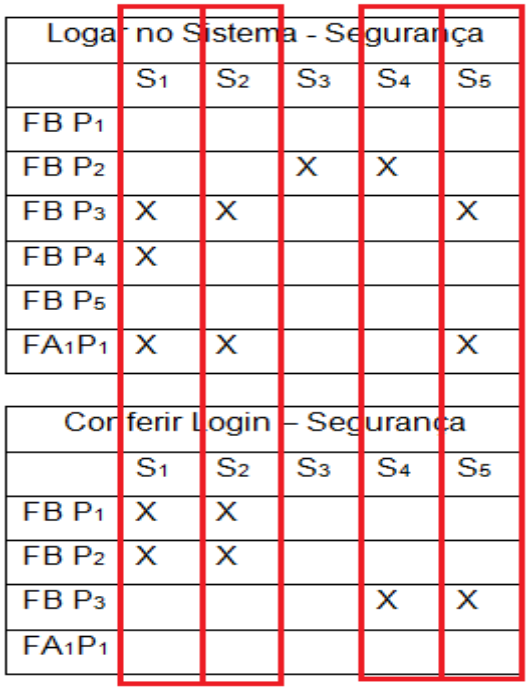


Figura 19 – Espalhamento de atributos de aspectos
Fonte: Autoria Própria

Essa figura ilustra um exemplo de espalhamento de atributos de aspectos que ocorreu durante a aplicação do método proposto. Como existe o espalhamento de código, executou-se a atividade seguinte do método: *Tratar Espalhamento em Matriz de Adjacência*, descrita a seguir.

4.2.5 Tratar espalhamento em matriz de adjacência

O espalhamento de aspectos é um fenômeno comum durante a aplicação do método proposto. Para eliminar esse espalhamento, várias matrizes auxiliares foram criadas para armazenar os atributos de aspectos espalhados. O Quadro 19 apresenta a matriz auxiliar que foi utilizada para tratar o espalhamento dos atributos de aspecto do tipo segurança.

Matriz Auxiliar - Segurança					
	S_1	S_2	S_3	S_4	S_5
Logar no Sistema – FB P_2				X	
Logar no Sistema – FB P_3	X	X			X
Logar no Sistema – FB P_4	X				
Logar no Sistema – FA_1P_1	X	X			X
Conferir Login – FB P_1	X	X			
Conferir Login – FB P_2	X	X			
Conferir Login – FB P_3				X	X

Quadro 19 - Matriz auxiliar para atributos de aspecto do tipo segurança
Fonte: Autoria Própria

As outras matrizes auxiliares desenvolvidas durante a aplicação do método proposto estão disponíveis no Apêndice C.

4.2.6 Tratar aspectos identificados

A última atividade do método proposto foi realizada para representar os aspectos identificados.

O Quadro 20 apresenta a descrição textual utilizada para representar esses aspectos. Esse quadro apresenta os aspectos identificados na descrição textual

Logar no Sistema a partir das matrizes de adjacência desenvolvidas anteriormente. Dentro dessa descrição, apenas o passo cinco não apresentou aspectos.

UC001	**Logar no Sistema**
Breve Descritivo:	Este Caso de Uso inicia quando o usuário executa a aplicação.
Fluxo Principal	
P1- O sistema pede que o usuário informe usuário e senha. ***Join point***	
P2- O usuário digita o seu nome de usuário e sua senha no sistema e clica no botão logar. ***Join point***	
P3- O sistema verifica a inserção dos dados através de Preencher Campos Obrigatórios. ***Join point***	
P4- O sistema autoriza o acesso do usuário, mostrando-lhe a tela inicial do sistema com suas configurações pessoais. ***Join point***	
P5- O caso de uso termina.	
Fluxo Alternativo	
FA1 – No passo 3 caso ocorra algum erro no preenchimento de campos obrigatórios ou na validação dos dados informados, o sistema limpará os campos e mostrará a tela de *login* novamente. ***Join point***	
Tipos de aspectos	
Segurança	
Usabilidade	
Manutenibilidade	
Confiabilidade	
Desempenho	

Quadro 20 – Identificação de aspectos dentro da descrição textual Logar no Sistema
Fonte: Autoria Própria

O Quadro 21 apresenta os aspectos identificados na descrição textual *Conferir Login*. Todos os passos dessa descrição apresentam comportamentos transversais. Os atributos de aspectos identificados para essas descrições estão disponíveis no Apêndice D, exceto para o tipo usabilidade, que é ilustrado pelo Quadro 22.

UC001	Conferir Login
Breve Descritivo:	O caso de uso verifica se os campos usuário e senha que foram informados pelo usuário correspondem as informações inseridas no banco de dados.
Sub fluxo	
S1- O sistema verifica se o usuário informado possui mais que cinco caracteres incluindo letras e números. ***Join point***	
S2- O sistema executa um algoritmo de *hash* na senha informada. ***Join point***	
S3- O sistema verifica se usuário e senha submetidos ao banco de dados correspondem aos dados de entrada. ***Join point***	
Fluxo Alternativo	
FA1 - No passo 1 se o usuário informado não possuir letra(s) e número(s) e mais que cinco caracteres no total, será alertado ao usuário este problema. No passo 3 se os dados informados não correspondem no banco de dados, será mostrada uma mensagem alertando que as informações digitadas são incorretas ou não existem; o caso de uso termina. ***Join point***	
Tipos de aspectos	
Segurança	
Usabilidade	
Manutenibilidade	
Confiabilidade	
Desempenho	

Quadro 21 – Identificação de aspectos dentro da descrição textual Conferir Login
Fonte: Autoria Própria

Utilizando as informações da seção Tipos de Aspectos é possível identificar quais tipos de aspecto essa descrição possui. O Quadro 22 ilustra os atributos de aspectos identificados e em qual passo esses foram encontrados.

Aspecto Usabilidade	
Logar no Sistema – FB P_1	U_1, U_2, U_6, U_8, U_9, U_{11}, U_{12}.
Logar no Sistema – FB P_2	U_6, U_8, U_9.
Logar no Sistema – FB P_3	U_1, U_2, U_6, U_8, U_9, U_{11}, U_{12}.
Logar no Sistema – FB P_4	U_2, U_8.
Logar no Sistema – FA_1P_1	U_1, U_2, U_6, U_8, U_9, U_{11}, U_{12}.
Conferir Login – FA_1P_1	U_1, U_2, U_8, U_9, U_{11}, U_{12}.

Quadro 22 – Detalhes sobre os aspectos identificados do tipo usabilidade
Fonte: Autoria Própria

Esse quadro ilustra atributos de aspecto do tipo usabilidade e podem ser codificados de diversas formas. Exemplos disso são as mensagens de sucesso e erro que são exibidas para os usuários.

4.3 ATRIBUTOS DE ASPECTOS IDENTIFICADOS PARA OS SISTEMAS DE LOGIN, CADASTRO DE USUÁRIO E FALE CONOSCO

A seção anterior apresentou detalhadamente o processo de aplicação do método proposto. Nele foram utilizadas apenas duas descrições textuais. Essa seção apresenta alguns dos atributos de aspectos identificados nas outras descrições textuais presentes no trabalho de Kossoski (2010).

O Quadro 23 apresenta os atributos de aspecto identificados para a descrição textual *Cadastrar Usuário*, do sistema Cadastro de Usuário.

Cadastro de Usuário – Cadastrar Usuário	
Cadastrar Usuário – FB P_1	U_1, U_2, U_6, U_8, U_{11}
Cadastrar Usuário – FB P_2	S_2, S_4, U_1, U_2, U_6, U_8, U_{11}
Cadastrar Usuário – FB P_4	S_4, C_1, C_2, C_4, C_7, C_8, D_1, D_6, D_9, D_{10}
Cadastrar Usuário – FB P_6	S_1, S_2, C_7, C_8, D_6, D_{10}
Cadastrar Usuário – FA_1P_1	S_2, S_4, U_1, U_2, U_6, U_8, U_{11}, C_7, C_8, D_1, D_6, D_{10}
Cadastrar Usuário – FA_2P_1	S_2, U_1, U_2, U_6, U_8, U_{11}, D_1

Quadro 23 – Atributos de aspecto identificados na descrição textual Cadastrar Usuário
Fonte: Autoria Própria

Após a aplicação do método proposto, concluiu-se que os atributos de aspecto do tipo manutenibilidade estão implicitamente presentes em cada passo das descrições textuais. Por esse motivo eles não estão representados. O Quadro 24 ilustra os atributos de aspectos identificados na descrição textual *Enviar Mensagem*, do sistema Fale Conosco. Os outros atributos identificados estão disponíveis no Apêndice E.

Fale Conosco – Enviar Mensagem	
Enviar Mensagem – FB P_2	U_1, U_2, U_6, U_8, U_{11}
Enviar Mensagem – FA_1 P_1	S_2, S_4, U_1, U_2, U_6, U_8, U_{11}

Quadro 24 – Atributos de aspecto para a descrição textual Enviar Mensagem
Fonte: Autoria Própria

O próximo capítulo apresenta a implementação de alguns dos aspectos identificados durante a aplicação do método proposto.

5 RESULTADOS

O objetivo do método proposto é identificar atributos de aspectos dentro da fase de análise de sistemas. Para ilustrar a relevância dos atributos de aspectos identificados, esse capítulo apresenta a codificação de alguns deles e depois faz uma análise comparativa entre os métodos da literatura e o método proposto. A seção 5.1 apresenta a implementação dos atributos de aspectos identificados no capítulo 4. A seção 5.2 realiza um comparativo entre o método proposto e àqueles apresentados no capítulo 2.

5.1 CODIFICAÇÃO DE ASPECTOS

O processo de codificação dos atributos de aspectos identificados pelo método proposto foi realizado utilizando as linguagens *JavaScript* (*JavaScript*, 2012) e *AspectJ* (*AspectJ*, 2012).

Javascript é a principal linguagem de *script* utilizada em páginas *web* para fornecer níveis de interatividade que não seriam possíveis apenas com HTML (*HyperText Markup Language*). Para realizar a implementação dos atributos de aspectos do tipo usabilidade de forma satisfatória, foi necessário utilizar o *plugin jQuery AOP* (*jQuery AOP*, 2012). Esse *plugin* permite adicionar os recursos da orientação a aspectos dentro da linguagem *Javascript*. Os atributos de aspectos implementados são ilustrados pelos Quadros 25 e 26.

A Figura 20 ilustra um aspecto implementado por meio da linguagem *Javascript*. A responsabilidade desse aspecto é verificar se cada campo obrigatório de um formulário está vazio. Se sim, esse aspecto exibe um alerta ao usuário e destaca, com uma determinada cor, que o campo está vazio. É possível utilizar esse aspecto em várias páginas HTML.

```
jQuery.aop.before( {target: window, method: 'validarCamposVazios()'},
    function() {
        var form = $("#submitForm :input");
        var value = '';
        isFormValid = true;

        $(form).each(function(){

            if($.trim($(this).val().trim()) == ""){

                if(((($(this).attr('alt') + "").indexOf('*')) != -1){
                    $(this).addClass("inputFieldAlert");

                    value = $(this).attr("alt") + "";

                    value = value.replace('*', '');
                    value = value.replace(':', '');
                    value = value.replace(/\(.*?\)/, '');

                    alertMessage += value + "\n";

                    isFormValid = false;
                }
            }
            else{
                $(this).removeClass("inputFieldAlert");
            }
        });
    }
);
```

Figura 20 – Codificação de aspecto utilizando Javascript
Fonte: Autoria Própria

Esta figura apresenta um aspecto que representa a implementação de vários atributos de aspectos (Quadro 25). Esses atributos referem-se ao sistema Cadastro de Usuário.

Preencher Campos Obrigatórios – S_1P_1	S_1, S_2, C_2, D_1
Preencher Campos Obrigatórios- $S_1A_1P_1$	U_1, U_2, U_6, U_8, U_{11}
Validar Campos Obrigatórios – S_1P_1	S_2, C_2, C_4, D_1, D_6
Validar Campos Obrigatórios – $S_1A_1P_1$	U_1, U_2, U_8, U_9, U_{11}
Validar Dados Específicos – S_1P_1	S_2, D_1
Validar Dados Específicos – S_1P_2	S_2, D_1
Validar Dados Específicos – $S_1A_1P_1$	U_1, U_2, U_6, U_8, U_{11}
Validar Dados Específicos – $S_1A_1P_2$	U_1, U_2, U_6, U_8, U_{11}
Validar Dados Específicos – S_2P_1	S_2,D_1
Validar Dados Específicos – S_2P_2	S_2,D_1
Validar Dados Específicos – $S_2A_1P_1$	U_1, U_2, U_6, U_8, U_{11}
Validar Dados Específicos – $S_2A_1P_2$	U_1, U_2, U_6, U_8, U_{11}
Validar Dados Específicos – S_3P_1	S_2,D_1
Validar Dados Específicos – S_3P_2	S_2,D_1
Validar Dados Específicos – $S_3A_1P_1$	U_1, U_2, U_6, U_8, U_{11}
Validar Dados Específicos – $S_3A_1P_2$	U_1, U_2, U_6, U_8, U_{11}

Quadro 25 – Atributos de aspecto implementados para o sistema Cadastro de Usuário
Fonte: Autoria Própria

A linguagem *JavaScript* foi utilizada para implementar atributos de aspecto porque permite implementar satisfatoriamente atributos do tipo usabilidade e possui alto nível de compatibilidade com a maioria dos navegadores *Web*.

Por sua vez, a linguagem *AspectJ* foi usada para implementar outros atributos de aspectos (Quadro 26). A Figura 21 ilustra um aspecto de *logging* codificado com essa linguagem. O uso de *logging* permite registrar qualquer tipo de evento ocorrido dentro de um software. A partir das informações registradas dentro dos arquivos de *log* é possível aumentar tanto o nível de segurança quanto o desempenho de um software.

```
1  package aspectos;
2
3  import javax.servlet.http.HttpServletRequest;
10
11 public aspect LoggerLoginLogout {
12
13     pointcut login(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) :
14         args(mapping, form, request, response) &&
15         execution(* app.LoginAction.execute(ActionMapping, ActionForm, HttpServletRequest, HttpServletResponse));
16
17     pointcut logout(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) :
18         args(mapping, form, request, response) &&
19         execution(* app.LogoutAction.unspecified(ActionMapping, ActionForm, HttpServletRequest, HttpServletResponse));
20
21
22     after(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) :
23                                                                   login(mapping, form, request, response){
24         if(request.getSession().getAttribute("user") != null){
25             Logging.log("Usuário " + request.getSession().getAttribute("user") + " realizou login.");
26         }
27     }
28     before(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) :
29                                                                 logout(mapping, form, request, response){
30         Logging.log("Usuário " + request.getSession().getAttribute("user") + " realizou logout.");
31     }
32 }
```

Figura 21 – Codificação de aspecto utilizando AspectJ
Fonte: Autoria Própria

A responsabilidade desse aspecto é registrar, em um arquivo de *log*, toda ocorrência de *log in* e *log out* dentro de um sistema. A linha 13 define o *pointcut login*, responsável por identificar os *join points* onde ocorre o processo de *login*. A linha 22 inicia o *advice* que implementa esse *poincut*. O *advice* registra todas as tentativas de *login* realizadas com sucesso.

A linha 17 mostra o *poincut logout* é responsável por capturar os *join points* onde ocorre o processo de *logout*. O *advice* iniciado na linha 28 registra toda ocorrência do processo de *logout* realizado dentro do sistema.

O Quadro 26 mostra os atributos de aspectos, sistema Fale Conosco, que foram codificados no aspecto ilustrado pela Figura 21.

Logar no Sistema – FB P_2	S_4
Conferir Login – FB P_3	S_4

Quadro 26 – Atributos de aspectos implementados para o sistema de Login
Fonte: Autoria Própria

Os atributos de aspectos apresentados nessa seção foram implementados utilizando duas linguagens distintas. Porém, a função do método proposto é apenas identificar atributos de aspectos dentro da fase de análise de sistemas. A linguagem que será utilizada para codificar os atributos identificados é decisão daqueles que irão desenvolver o sistema.

Portanto, nota-se que os atributos identificados pelo método são possíveis de serem implementados. Isto demonstra que a teoria e a prática são aliadas para a criação de sistemas com um menor nível de espalhamento de código.

5.2 COMPARATIVO ENTRE OS MÉTODOS

Esta seção irá apresentar um breve comparativo entre o método proposto e os métodos apresentados no capítulo 2 para identificação de aspectos em nível de análise.

5.2.1 Abordagem de modelagem de caso de uso para aquisição de *early aspects*

A técnica proposta por Kong e Yuan (2009) faz uso da metodologia de Engenharia de Software *Rational Unified Process* (RUP) para identificação de aspectos.

Essa identificação ocorre a partir da análise das propriedades de caso de uso e são inseridas de forma iterativa até que o caso de uso esteja construído. O

Quadro 27 apresenta as principais diferenças entre essa técnica e o método proposto.

Kong e Yuan (2009)	Método proposto
A identificação de aspectos pode ser realizada durante ou após a criação dos casos de uso e descrição textual.	O processo de identificação de aspecto recebe, como entrada, as descrições textuais.
Aspectos são identificados por caso de uso.	Atributos de aspectos são identificados por passo de descrição textual.
Uma das propriedades utilizadas para identificação de aspectos é RNF.	A propriedade utilizada para identificar atributos de aspectos é RNF e cada RNF é classificado por atributos.
Utiliza o modelo iterativo proposto pela metodologia RUP para a criação dos casos de uso.	Processo iterativo ocorre no momento de verificar e tratar espalhamento de atributos de aspectos em matriz de adjacência.

Quadro 27 – Comparativo entre o método proposto e a técnica proposta por Kong e Yuan (2009)
Fonte: Autoria Própria

As principais diferenças dizem respeito ao local onde os aspectos são identificados e a maneira como os RNFs são tratados.

No método proposto, cada passo da descrição textual é analisado para verificar a existência de atributos de aspectos. Já na abordagem de Kong e Yuan (2009), os aspectos são identificados por caso de uso. Ao exigir que o analista procure atributos de aspectos em cada passo das descrições textuais, o método proposto possibilita maior aquisição de conhecimento sobre a regra de negócio do sistema, além de aumentar as chances de identificar atributos de aspectos.

A maneira como cada método utiliza os RNFs também é diferente. Enquanto Kong e Yuan (2009) consideram os RNFs como mais uma opção para identificar aspectos, o método proposto considera os RNFs a principal fonte para identificação de aspectos. A identificação de atributos de aspecto a partir de RNFs é justificada pelas semelhanças entre aspectos e RNFs apresentadas nesse trabalho.

5.2.2 Identificação de aspectos baseada em *Use Case Slice*

A unidade *use case slice* foi inserida dentro da técnica de caso de uso para tornar possível a identificação de aspectos. O Quadro 28 apresenta as principais

diferenças entre a identificação de aspectos baseada em *use case slice* e o método proposto.

Jacobson e NG (2004)	Método proposto
Utiliza *use case slice* para armazenar comportamento transversal.	Os atributos de aspectos são identificados e armazenados em matrizes de adjacência.
A identificação de aspectos tende a ser baseada em requisitos funcionais.	Atributos de aspectos são identificados a partir dos RNFs.

Quadro 28 - Comparativo entre o método proposto e o método apresentado por Jacobson e NG (2004)
Fonte: Autoria Própria

No método de Jacobson e NG (2004), *use case slice* é utilizado para representar um comportamento transversal e os relacionamentos <<*include*>> e <<*extend*>> são usados para tratar o espalhamento de aspectos. Já no método proposto, as matrizes de adjacência são utilizadas para armazenar os atributos de aspectos identificados e as matrizes auxiliares tratam o espalhamento de atributos de aspectos. Por fim, esses atributos de aspecto são representados em suas respectivas descrições textuais.

A principal diferença entre esses métodos está no objeto utilizado para identificar aspectos. Na técnica de caso de uso, os requisitos funcionais são utilizados para identificar aspectos. Por sua vez, o método proposto utiliza os requisitos não-funcionais para realizar a identificação de atributos de aspecto.

5.2.3 Identificação de candidatos a aspectos utilizando responsabilidades

A abordagem para identificação de aspectos baseada em responsabilidade (HORNUNG, 2010) apresenta uma maneira de identificar candidatos a aspectos durante a análise de requisitos. O Quadro 29 apresenta as principais diferenças entre ela e o método proposto.

Hornung (2010)	Método Proposto
Utiliza casos de uso para representar os candidatos a aspectos identificados.	Utiliza matriz de adjacência para identificar os atributos de aspectos e descrição textual para representá-los.
O critério 'Responsabilidade = Verbo + Complemento' costuma induzir a identificação de aspectos a partir de requisitos funcionais.	Utiliza atributos de RNFs para identificar atributos de aspectos.

Quadro 29 – Comparativo entre o método proposto e a abordagem apresentada por Hornung
Fonte: Autoria Própria

Apesar de ambos os métodos receberem como entrada descrições textuais de caso de uso, eles diferem no uso dessas entradas.

Em sua abordagem, Hornung (2010) utiliza casos de uso para representar os aspectos identificados. Enquanto isso, dentro do método proposto, os atributos de aspectos são identificados em matrizes de adjacência e representados nas suas respectivas descrições textuais.

A implementação dos atributos de aspectos identificados pelo método proposto permite que as classes e aspectos desenvolvidos possuam alto nível de coesão.

5.2.4 RESUMO DO COMPARATIVO ENTRE MÉTODOS

Após realizar esse comparativo constatou-se que os aspectos identificados por esses métodos diferem dos atributos de aspecto levantados pelo método proposto. Isso acontece porque o método proposto utiliza RNFs como referência para a identificação de atributos de aspectos, enquanto que os outros métodos utilizam RFs para o mesmo fim.

A implementação de um atributo de aspecto identificado pelo método proposto pode resultar em vários aspectos identificados pelos métodos anteriores. Por exemplo, a codificação do atributo de aspecto S_2(Integridade) pode resultar nos seguintes aspectos:

- Verificar se campo está vazio;
- Verificar se CPF é válido;
- Verificar se CNPJ é válido.

O analista deve identificar como esses atributos de aspecto deverão ser codificados. Entretanto, esse processo já está fora do escopo do método proposto.

Da mesma forma, um aspecto pode representar um ou mais atributos de aspectos. Por exemplo, o aspecto ‘Validar CPF’, descrito por Kossoski (2010, p. 98), pode representar os atributos de aspectos S_1(Confiabilidade) e S_2(Integridade). Vale lembrar que os métodos descritos por Kossoski não consideram atributos de RNFs para identificar aspectos, mas sim RFs.

Apesar disso, esse comportamento não é suficiente para indicar qual método consegue identificar o maior número de aspectos.

6 CONCLUSÃO

Este trabalho apresentou um método para realizar a identificação de atributos de aspectos durante a fase de análise de sistemas e demonstrou a implementação de alguns desses aspectos. Os métodos de identificação de aspectos descritos por Jacobson e NG (2004), Kong e Yuan (2009) e Hornung (2010) também foram apresentados em que se mostraram suas vantagens e desvantagens.

Ao realizar um estudo de caso, constatou-se que o método proposto é eficaz na identificação de atributos de aspectos. Os métodos apresentados na literatura utilizam de forma sucinta os RNFs para identificação de aspectos, por outro lado, o método proposto enfatiza a importância dos RNFs e explora seus atributos (MAIRIZIA et al., 2010) para realizar a identificação de atributos de aspectos. O resultado disso é que os aspectos identificados por meio do método proposto são diferentes daqueles identificados pelos outros métodos.

Demonstraram-se exemplos de como os atributos de aspectos podem ser codificados. Apesar de o método propor apenas a identificação de atributos de aspectos a nível de análise, vários atributos foram implementados com o intuito de tornar mais evidente a aplicação dos aspectos levantados por meio método proposto.

A identificação de aspectos em fases anteriores à de programação possibilita o desenvolvimento de softwares mais modulares e com maiores níveis de manutenibilidade. Problemas como entrelaçamento e espalhamento de código são tratados de maneira eficaz, porque os atributos de aspectos, identificados na fase de análise, proporcionam maior nível de coesão das classes e aspectos do sistema.

6.1 TRABALHOS FUTUROS

A aplicação do método proposto sem o auxilio de uma ferramenta CASE (*Computer-Aided* Software *Engineering*) torna-se muito trabalhosa e propensa a erros. Um trabalho futuro que pode ser realizado é o desenvolvimento de uma ferramenta para auxiliar o analista durante o processo de identificação de atributos de aspectos.

Outro trabalho futuro possível de ser realizado é o tratamento do espalhamento de atributos de aspectos entre sistemas. Atualmente, o método proposto trata o espalhamento de atributos de aspectos apenas dentro de um sistema ou subsistema. O tratamento desse espalhamento permitirá maior modularidade no desenvolvimento de software e possibilitará a identificação de padrões de aspectos entre sistemas.

Um terceiro trabalho futuro refere-se à criação de um método dentro da fase de projeto orientado a aspectos. Esse método receberia como entrada os atributos de aspectos identificados e iria gerar, como saída, diagramas representando esses atributos dentro do sistema a ser implementado.

REFERÊNCIAS

ASLAM, Toheed; DOHERTY, Jesse; DUBRAU, Anton; HENDREN, Laurie. AspectMatlab: an aspect-oriented scientific programming language. **Proceedings…** 9th International Conference On Aspect-Oriented Software Development, 2010, Rennes e St. Malo.

AspectJ. Disponível em: < http://www.eclipse.org/aspectj/ >. Acesso em: 01 de Maio 2012.

CHAVEZ, Christina v. F. G. **Um Enfoque Baseado em Modelos para o *Design* Orientado a Aspectos**. 2004. 298 f. Tese (Doutorado em Informática) – Departamento de Informática, Pontifícia Universidade Católica, Rio de Janeiro, 2004.

CHUNG, Lawrence; Nixon, Brian A.; ERIC, Yu; MYLOPOULOS, John. **Non-Functional Requirements in Software Engineering.** 1. ed. Norwell: Springer, 1999.

CYSNEIROS, Luiz M. **Requisitos Não Funcionais: Da Elicitação ao Modelo Conceitual.** 2001. 224 f. Tese de Doutorado (Doutorado em Ciência da Computação) – Programa de Pós-Graduação em Ciência da Computação, Pontifícia Universidade Católica, Rio de Janeiro, 2001.

DOUENCE, Rémi; FRITZ, Thomas; LORIANT, Nicolas; MENAUD, Jean-Marc; SÉGURA-DEVILLECHAISE, Marc; SÜDHOLT, Mario. An expressive aspect language for system applications with Arachne. **Proceedings …** 4th International Conference on Aspect-Oriented Software Development, 2005, Chicago.

ELRAD, Tzilla; AKSIT, Mehmet; KICZALES, Gregor; LIEBERHERR, Karl; OSSHER, Harold. Discussing aspects of AOP. **Communications of the ACM**. New York, v. 44, n. 10, p. 33-38, Out. 2001.

FONTOURA, M.; PREE, W.; RUMPE, B. UML-F: A modeling language for object-oriented frameworks. **Proceedings…** 14th European Conference on Object-Oriented Programming, 2000, Sophia Antipolis e Cannes.

FOWLER, Martin; BECK, Kent; BRANT, John; OPDYKE, William; ROBERTS, Don. **Refactoring: Improving the Design of Existing Code.** New York: Addison-Wesley Professional, 1999.

GASTALDO, Denise L.; MIDORIKAWA, Edson T. Processo de Engenharia de Requisitos Aplicado a Requisitos Não-Funcionais de Desempenho – Um Estudo de Caso. **In:** Workshop em Engenharia de Requisitos, 2003, Piracicaba.

HORNUNG, Rafael. **Um método para identificação antecipada de candidatos a aspectos no desenvolvimento de frameworks de domínio**. 2010. 83 f. Dissertação (Mestrado em Informática) - Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná. Curitiba, 2010.

JACOBSON, Ivar; NG, Pan-Wei. **Aspect-Oriented Software Development with Use Cases**. Upper Saddle River: Pearson Education, 2004.

JavaScript. Disponível em: < http://en.wikipedia.org/wiki/JavaScript >. Acesso em: 01 de Maio 2012.

jQuery AOP. Disponível em: < http://code.google.com/p/jquery-aop/ >. Acesso em: 01 de Maio de 2012.

KICZALES, Gregor; HILSDALE, Erik; HUGUNIN, Jim; KERSTEN, Mik; PALM, Jeffrey; GRISWOLD, William. Getting started with ASPECTJ. **Communications of the ACM**. New York, v. 44, n. 10, p. 59-65, Out. 2001.

KONG, L;YUAN, T. Use Case Modeling Approach for Early Aspect Acquisition. **In:** ACM SIGSOFT Software Engineering Notes. 2009, New York.

KOSSOSKI, Clayton. **Modelagem de Sistemas Baseada em Orientação a Aspectos Usando Casos de Uso e Responsabilidades**. 2010. 106 f. Trabalho de Conclusão de Curso (Graduação) – Curso Superior em Análise e Desenvolvimento de Sistemas. Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2010.

LADDAD, Ramnivas. **Aspectj in Action: Practical Aspect-Oriented Programming.** 1 ed. New York: Manning Publications, 2003.

MCHEICK, Hamid; SIOUD, Aymen. Comparison of Garbage Collector prototypes for C++ applications**. In:** IEEE/ACS International Conference on Computer Systems and Applications (AICCSA), 2009, Rabat.

MAIRIZA, Dewi; ZOWGHI, Didar; NURMULIANI, Nurie. An Investigation Into the Notion of Non-Functional Requirements. **Proceedings**...2010 Acm Symposium on Applied Computing, 2010, Sierre.

MATOS, Simone Nasser; FERNANDES, Clovis Torres. **Abordagem dirigida por responsabilidades voltada ao desenvolvimento de *frameworks* de domínio**. Relatório Técnico Científico ITA, São José dos Campos SP, 2007.

MUNNELLY, Jennifer; CLARKE, Siobhán. HL& healthcare information management using aspect-oriented programming. **In:** 22th IEEE International Symposium on Computer-Based Medical Systems, 2009, Albuquerque.

NOEL, Steven; JAJODIA, Sushil. Understanding complex network attack graphs through clustered adjacency matrices. **In:** 21st Annual Computer Security Appplications Conference, 2005, Tucson.

PEDERSON, Knut H; CONSTANTINIDES, Constantinos. AspectAda – Aspect Oriented Programming for Ada95. **In:** SIGAda'05, 2005, New York.

RASHID, Awais; SAWYER, Peter; MOREIRA, Ana; ARAÚJO, João. Early Aspects: a Model for Aspect-Oriented Requirements Engineering. **In:** IEEE Joint Conference on Requirements Engineering, 2002, Essen.

RASHID, Awais; COTTENIER, Thomas, GREENWOOD, Phil, CHITCHYAN, Ruzanna, MEUNIER, Regine, COELHO, Roberta, SÜDHOLT, Mario, JOOSEN, Wouter. Aspect-Oriented Software Development in Practice: Tales from AOSD-Europe. **Computer IEEE Journals.** Los Alamitos, v. 43, p. 19-26, 2010.

SPINCZYK, Olaf; GAL, Andreas; SCHRÖDER-PREIKSCHAT, Wolfgang. AspectC++: an aspect-oriented extension to the C++ programming language. **Proceedings** … Fortieth International Conference on Tools Pacific: Objects for Internet, Mobile and Embedded Applications, 2002, Darlinghurst.

TAEHO, Kim, HONGCHUL, Lee. Establishment of a security system using Aspect Oriented Programming**. In:** International Conference on Control, Automation and Systems, 2008, p. Seoul.

TAO LONG; CHEN, D.; RONGGONG SONG. Measure Large Scale Network Security Using Adjaceny Matrix Attack Graphs. **In:** 5th International Conference on Future Information Technology (FutureTech), 2010, Busan.

TOLEDO, Rodolfo; LEGER, Paul; TANTER, Éric. AspectScript: Expressive Aspects for the Web. **Proceedings** … 9th International Conference on Aspect-Oriented Software Development, 2010, Rennes e St. Malo.

TORRENS, I. C; MATOS, S.N. M. Identificação e Implementação dos Aspectos no Desenvolvimento dos Aplicativos de Login, Cadastro de Usuário e Fale Conosco. **In:** SICITE, 2010, Cornélio Procópio.

XAVIER, Laís. **Integração de Requisitos Não-Funcionais a Processos de Negócios: Integrando BPMN e RNF**. 2009. 100 f. Dissertação (Mestrado em Ciência da Computação) - Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Pernambuco, 2009.

ZHENGYAN, Dong. Aspect Oriented Programming Technology and the Strategy of Its Implementation. **In:** International Conference on Intelligence Science And Information Engineering (ISIE), 2011, Wuhan.

WEN-LIN LIU;CHUNG-HORNG LUNG; AJILA, Samuel. Impact of Aspect-Oriented Programming on Software Performance: A Case Study of Leader/Followers and Half-Sync/Half-Async Architectures**. In:** IEEE 35th Annual Computer Software and Applications Conference (COMPSAC), 2011, Munich.

APÊNDICE A - Quadro de Requisitos Não Funcionais

QUADRO DO REQUISITO NÃO FUNCIONAL - MANUTENIBILIDADE

O Quadro 30 ilustra os atributos do requisito não funcional do tipo manutenibilidade.

Manutenibilidade	
M_1	Capacidade de ser testado
M_2	Capacidade de ser compreendido
M_3	Capacidade de ser modificado
M_4	Capacidade de ser analisado

Quadro 30 – Requisito Não Funcional Manutenibilidade
Fonte: Adaptado de Mairizia et al (2010)

Os atributos apresentados por este quadro referem-se a capacidade do software em sofrer alterações. Tais alterações podem ser para corrigir um defeito ou realizar uma melhoria no software.

QUADRO DO REQUISITO NÃO FUNCIONAL - CONFIABILIDADE

O Quadro 31 ilustra os atributos do requisito não funcional do tipo confiabilidade.

Confiabilidade	
C_1	Precisão
C_2	Consistência
C_3	Disponibilidade
C_4	Integridade
C_5	Corretitude
C_6	Maturidade
C_7	Tolerância à falhas
C_8	Capacidade de recuperação

Quadro 31 – Requisito Não Funcional Confiabilidade
Fonte: Adaptado de Mairizia et al.(2010)

Os atributos do requisito não funcional do tipo confiabilidade referem-se à capacidade do software para operar sem falhas e manter um determinado nível de desempenho. Essa capacidade deve ser medida levando-se em consideração tanto circunstâncias normais quanto extraordinárias. Um exemplo de situação extraordinária é um pico de acesso em um determinado período do dia.

QUADRO DO REQUISITO NÃO FUNCIONAL - USABILIDADE

O Quadro 32 ilustra os atributos do requisito não funcional do tipo usabilidade.

Usabilidade	
U_1	Capacidade de ser aprendido
U_2	Compreensibilidade
U_3	Capacidade de ser operável
U_4	*Attractiveness*
U_5	Capacidade de ser usável
U_6	Facilidade de uso
U_7	Engenharia Humana
U_8	Amigável ao usuário
U_9	Fácil de memorizar
U_{10}	Eficiência
U_{11}	Tempo de reação do usuário
U_{12}	Produtividade do usuário

Quadro 32 – Requisito Não Funcional Usabilidade
Fonte: Adaptado de Mairizia et al.(2010)

Os atributos do requisto não funcional do tipo usabilidade referem-se às interações do usuário final com o sistema e o esforço necessário para aprender, operar, preparar entrada de dados e interpretar a resposta do sistema.

QUADRO DO REQUISITO NÃO FUNCIONAL - SEGURANÇA

O Quadro 33 ilustra os atributos do requisito não funcional do tipo segurança.

Segurança	
S_1	Confiabilidade
S_2	Integridade
S_3	Disponibilidade
S_4	Controle de acesso
S_5	Autenticação

Quadro 33 – Requisito Não Funcional Segurança
Fonte: Adaptado de Mairizia et al.(2010)

Os atributos de segurança dizem respeito aos requisitos sobre a prevenção de acesso não autorizado ao sistema.

APÊNDICE B - Matrizes de adjacência resultantes da aplicação do método

MATRIZES DE ADJACÊNCIA PARA O CASO DE USO *LOGAR NO SISTEMA*

O Quadro 34 apresenta a matriz de adjacência com os atributos de aspectos do tipo usabilidade identificados para o caso de uso *Logar no Sistema*.

Logar no Sistema – Usabilidade												
	U_1	U_2	U_3	U_4	U_5	U_6	U_7	U_8	U_9	U_{10}	U_{11}	U_{12}
FB P_1	X	X				X		X	X		X	X
FB P_2						X		X	X			
FB P_3	X	X				X		X	X		X	X
FB P_4		X						X				
FB P_5												
FA_1 P_1	X	X				X		X	X		X	X

Quadro 34 – Matriz de adjacência Usabilidade para a descrição Logar no Sistema
Fonte: Autoria Própria

A matriz ilustrada pelo Quadro 35 apresenta os atributos de aspectos do tipo confiabilidade identificados para o caso de uso *Logar no Sistema*.

Logar no Sistema – Confiabilidade								
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
FB P_1								
FB P_2								
FB P_3	X			X	X		X	X
FB P_4								
FB P_5								
FA_1 P_1	X			X	X		X	X

Quadro 35 – Matriz de adjacência Confiabilidade para a descrição Logar no Sistema
Fonte: Autoria Própria

O Quadro 36 apresenta a matriz de adjacência com os atributos de aspectos do tipo manutenibilidade identificados para o caso de uso *Logar no Sistema*.

Logar no Sistema - Manutenibilidade				
	M_1	M_2	M_3	M_4
FB P_1	X	X	X	X
FB P_2				
FB P_3	X	X	X	X
FB P_4	X	X	X	X
FB P_5				
FA_1 P_1	X	X	X	X

Quadro 36 – Matriz de adjacência Manutenibilidade para a descrição Logar no Sistema
Fonte: Autoria Própria

A matriz ilustrada pelo Quadro 37 apresenta os atributos de aspectos do tipo desempenho identificados para o caso de uso *Logar no Sistema*.

Logar no Sistema – Desempenho																
	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}	D_{16}
FB P_1																
FB P_2	X					X										
FB P_3	X					X			X	X					X	X
FB P_4	X					X										
FB P_5																
FA_1 P_1	X					X			X	X					X	X

Quadro 37 – Matriz de adjacência Desempenho para a descrição Logar no Sistema
Fonte: Autoria Própria

MATRIZES DE ADJACÊNCIA PARA O CASO DE USO *CONFERIR LOGIN*

A matriz ilustrada pelo Quadro 38 apresenta os atributos de aspectos do tipo segurança identificados para o caso de uso *Conferir Login*

Conferir Login – Segurança					
	S_1	S_2	S_3	S_4	S_5
FB P_1	X	X			
FB P_2	X	X			
FB P_3				X	X
FA_1P_1					

Quadro 38 – Matriz de adjacência Segurança para a descrição Conferir Login
Fonte: Autoria Própria

O Quadro 39 apresenta a matriz de adjacência com os atributos de aspectos do tipo usabilidade identificados para o caso de uso *Logar no Sistema*.

Conferir Login – Usabilidade												
	U_1	U_2	U_3	U_4	U_5	U_6	U_7	U_8	U_9	U_{10}	U_{11}	U_{12}
FB P_1												
FB P_2												
FB P_3												
FA_1 P_1	X	X						X	X		X	X

Quadro 39 – Matriz de adjacência Usabilidade para a descrição Conferir Login
Fonte: Autoria Própria

A matriz ilustrada pelo Quadro 40 apresenta os atributos de aspectos do tipo confiabilidade identificados para o caso de uso *Conferir Login*.

Conferir Login – Confiabilidade								
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
FB P_1	X						X	X
FB P_2				X				
FB P_3								
FA_1 P_1								

Quadro 40 – Matriz de adjacência Confiabilidade para a descrição Conferir Login
Fonte: Autoria Própria

O Quadro 41 apresenta a matriz de adjacência com os atributos de aspectos do tipo manutenibilidade identificados para o caso de uso *Logar no Sistema*.

Conferir Login - Manutenibilidade				
	M_1	M_2	M_3	M_4
FB P_1	X	X	X	X
FB P_2	X	X	X	X
FB P_3	X	X	X	X
FA_1 P_1				

Quadro 41 – Matriz de adjacência Manutenibilidade para a descrição Conferir Login
Fonte: Autoria Própria

A matriz ilustrada pelo Quadro 42 apresenta os atributos de aspectos do tipo desempenho identificados para o caso de uso *Conferir Login*.

Conferir Login - Desempenho																
	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}	D_{16}
FB P_1	X				X					X						
FB P_2						X				X	X	X				
FB P_3									X							
FA_1 P_1																

Quadro 42 – Matriz de adjacência Desempenho para a descrição Conferir Login
Fonte: Autoria Própria

APÊNDICE C - Matrizes auxiliares desenvolvidas durante a aplicação do método

MATRIZES AUXILIARES

A matriz ilustrada pelo Quadro 43 apresenta a matriz auxiliar do tipo usabilidade para os casos de Uso: *Logar no Sistema* e *Conferir Login*.

Matriz Auxiliar – Usabilidade												
	U_1	U_2	U_3	U4	U5	U6	U7	U8	U9	U10	U11	U12
Logar no Sistema – FB P_1	X	X						X	X		X	X
Logar no Sistema – FB P_2								X	X			
Logar no Sistema – FB P_3	X	X						X	X		X	X
Logar no Sistema – FB P_4		X						X				
Logar no Sistema – FA_1 P_1	X	X						X	X		X	X
Conferir Login – FA_1 P_1	X	X						X	X		X	X

Quadro 43 – Matriz auxiliar para o aspecto do tipo usabilidade
Fonte: Autoria Própria

O Quadro 44 apresenta a matriz auxiliar com os atributos de aspectos do tipo confiabilidade para os casos de Uso: *Logar no Sistema* e *Conferir Login*.

Matriz Auxiliar – Confiabilidade								
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
Logar no Sistema – FB P_3	X			X	X		X	X
Logar no Sistema – FA_1 P_1	X			X	X		X	X
Conferir Login – FB P_1	X						X	X
Conferir Login – FB P_2				X				

Quadro 44 – Matriz auxiliar para o aspecto do tipo confiabilidade
Fonte: Autoria Própria

A matriz ilustrada pelo Quadro 45 apresenta a matriz auxiliar do tipo manutenibilidade para os casos de Uso: *Logar no Sistema* e *Conferir Login*.

Matriz Auxiliar – Manutenibilidade				
	M_1	M_2	M_3	M_4
Conferir Login – FB P_1	X	X	X	X
Conferir Login – FB P_2	X	X	X	X
Conferir Login – FB P_3	X	X	X	X
Logar no Sistema – FB P_1	X	X	X	X
Logar no Sistema – FB P_3	X	X	X	X
Logar no Sistema – FB P_4	X	X	X	X
Logar no Sistema – FA_1P_1	X	X	X	X

Quadro 45 - Matriz auxiliar para o aspecto do tipo manutenibilidade
Fonte: Autoria Própria

O Quadro 46 apresenta a matriz auxiliar com os atributos de aspectos do tipo confiabilidade para os casos de Uso: *Logar no Sistema* e *Conferir Login*.

Matriz Auxiliar – Desempenho																
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15	D16
Conferir Login – FB P_1	X									X						
Conferir Login – FB P_2						X				X						
Conferir Login – FB P_3									X							
Logar no Sistema – FB P_2	X					X										
Logar no Sistema – FB P_3	X					X			X	X						
Logar no Sistema – FB P_4	X					X										
Logar no Sistema – FA_1 P_1	X					X			X	X						

Quadro 46 – Matriz auxiliar para o aspecto do tipo desempenho
Fonte: Autoria Própria

APÊNDICE D - Atributos de aspectos identificados em Logar no Sistema e Conferir Login

ASPECTOS IDENTIFICADOS PARA OS CASOS DE USO LOGAR NO SISTEMA E CONFERIR LOGIN

O Quadro 47 apresenta os atributos de aspectos do tipo segurança identificados para as descrições textuais *Logar no Sistema* e *Conferir Login*.

Aspecto Segurança	
Logar no Sistema – FB P_2	S_3, S_4.
Logar no Sistema – FB P_3	S_1, S_2, S_5.
Logar no Sistema – FB P_4	S_1.
Logar no Sistema – FA_1 P_1	S_1, S_2, S_5.
Conferir Login – FB P_1	S_1, S_2.
Conferir Login – FB P_2	S_1, S_2.
Conferir Login – FB P_3	S_4, S_5.

Quadro 47 – Detalhes sobre os aspectos identificados do tipo segurança
Fonte: Autoria Própria

Os atributos de aspecto do tipo desempenho, identificados nas descrições textuais *Logar no Sistema* e *Conferir Login* são ilustradas pelo Quadro 48.

Aspecto Desempenho	
Conferir Login – FB P_1	D_1, D_5, D_{10}.
Conferir Login – FB P_2	D_6, D_{10}, D_{11}, D_{12}.
Conferir Login – FB P_3	D_9.
Logar no Sistema – FB P_2	D_1, D_6.
Logar no Sistema – FB P_3	D_1, D_6, D_9, D_{10}, D_{15}, D_{16}.
Logar no Sistema – FB P_4	D_1, D_6.
Logar no Sistema – FA_1P_1	D_1, D_6, D_9, D_{10}, D_{15}, D_{16}.

Quadro 48 – Detalhes sobre os aspectos identificados do tipo desempenho
Fonte: Autoria Própria

O Quadro 49 ilustra os atributos de aspecto do tipo confiabilidade, enquanto o Quadro 50 ilustra os aspectos do tipo manutenibilidade.

Aspecto Confiabilidade	
Logar no Sistema – FB P_3	C_1, C_4, C_5, C_7, C_8.
Logar no Sistema – FA_1 P_1	C_1, C_4, C_5, C_7, C_8.
Conferir Login – FB P_1	C_1, C_7, C_8.
Conferir Login – FB P_2	C_4.

Quadro 49 – Detalhes sobre os aspectos identificados do tipo confiabilidade
Fonte: Autoria Própria

Aspecto Manutenibilidade	
Conferir Login – FB P_1	M_1, M_2, M_3, M_4.
Conferir Login – FB P_2	M_1, M_2, M_3, M_4.
Conferir Login – FB P_3	M_1, M_2, M_3, M_4.
Logar no Sistema – FB P_1	M_1, M_2, M_3, M_4.
Logar no Sistema – FB P_3	M_1, M_2, M_3, M_4.
Logar no Sistema – FB P_4	M_1, M_2, M_3, M_4.
Logar no Sistema – FA_1 P_1	M_1, M_2, M_3, M_4.

Quadro 50 – Detalhes sobre os aspectos identificados do tipo manutenibilidade
Fonte: Autoria Própria

Os atributos ilustrados pelo Quadro 50 estão fortemente relacionados à implementação dos atributos dos outros aspectos, uma vez que o aspecto manutenibilidade visa facilitar a realização de alterações dentro de um software. Unidades abstratas costumam ser utilizadas para codificar atributos de manutenibilidade.

APÊNDICE E - Atributos de aspectos dos sistemas Cadastro de Usuário e Fale Conosco

ATRIBUTOS DE ASPECTOS IDENTIFICADOS PARA O SISTEMA LOGIN, CADASTRO DE USUÁRIO E FALE CONOSCO

Cadastro de Usuário – Preencher Campos Obrigatórios	
Preencher Campos Obrigatórios – S_1 P_1	S_1, S_2, C_1, C_2, C_7, C_8, D_1, D_9, D_{10}
Preencher Campos Obrigatórios – $S_1A_1P_1$	U_1, U_2, U_6, U_8, U_9, U_{11}, C_7, C_8

Quadro 51 – Atributos de aspectos identificados em Preencher Campos Obrigatórios, sistema Cadastro de Usuário
Fonte: Autoria Própria

Cadastro de Usuário – Validar Campos Obrigatórios	
Validar Campos Obrigatórios – S_1 P_1	S_2, C_2, C_4, C_7, C_8, D_1, D_6, D_9, D_{10}
Validar Campos Obrigatórios – $S_1A_1P_1$	U_1, U_2, U_8, U_9, U_{11}

Quadro 52 – Atributos de aspectos identificados em Validar Campos Obrigatórios, sistema Cadastro de Usuário
Fonte: Autoria Própria

Cadastro de Usuário – Validar Dados Específicos	
Validar Dados Específicos – S_1 P_1	S_1, S_2, C_7, C_8, D_1, D_6, D_{10}
Validar Dados Específicos – S_1 P_2	S_1, S_2, C_7, C_8, D_1, D_6, D_{10}
Validar Dados Específicos – S_1 A_1 P_1	U_1, U_2, U_6, U_8, U_{11}
Validar Dados Específicos – S_1 A_1 P_2	U_1, U_2, U_6, U_8, U_{11}
Validar Dados Específicos – S_2 P_1	S_1, S_2, C_7, C_8, D_1, D_6, D_{10}
Validar Dados Específicos – S_2 P_2	S_1, S_2, C_7, C_8, D_1, D_6, D_{10}
Validar Dados Específicos – S_2 A_1 P_1	U_1, U_2, U_6, U_8, U_{11}
Validar Dados Específicos – S_2 A_1 P_2	U_1, U_2, U_6, U_8, U_{11}
Validar Dados Específicos – S_3 P_1	S_1, S_2, C_7, C_8, D_1, D_6, D_{10}
Validar Dados Específicos – S_3 P_2	S_1, S_2, C_7, C_8, D_1, D_6, D_{10}
Validar Dados Específicos – S_3 A_1 P_1	U_1, U_2, U_6, U_8, U_{11}
Validar Dados Específicos – S_3 A_1 P_2	U_1, U_2, U_6, U_8, U_{11}

Quadro 53 – Atributos de aspectos identificados em Validar Dados Específicos, sistema Cadastro de Usuário
Fonte: Autoria Própria

Fale Conosco – Preencher Campos Obrigatórios	
Preencher Campos Obrigatórios – S_1 P_1	S_1, S_2, C_7, C_8, D_6, D_9, D_{10}
Preencher Campos Obrigatórios – S_1 A_1 P_1	U_1, U_2, U_6, U_8, U_{11}

Quadro 54 – Atributos de aspectos identificados em Preencher Campos Obrigatórios, sistema Fale Conosco
Fonte: Autoria Própria

Fale Conosco – Validar Campos Obrigatórios	
Validar Campos Obrigatórios – S_1 P_1	S_1, S_2, C_7, C_8, D_1, D_6, D_9, D_{10}
Validar Campos Obrigatórios – S_1 A_1 P_1	U_1, U_2, U_6, U_8, U_{11}

Quadro 55 – Atributos de aspectos identificados em Validar Campos Obrigatórios, sistema Fale Conosco
Fonte: Autoria Própria

Fale Conosco – Validar Dados Específicos	
Validar Dados Específicos – S_1 P_1	C_1, C_2, C_7, C_8, D_6, D_9, D_{10}
Validar Dados Específicos – S_1 P_2	C_1, C_2, C_7, C_8, D_6, D_9, D_{10}
Validar Dados Específicos – S_1 A_1 P_1	U_1, U_2, U_6, U_8, U_{11}
Validar Dados Específicos – S_1 A_1 P_2	U_1, U_2, U_6, U_8, U_{11}

Quadro 56 – Atributos de aspectos identificados em Validar Dados Específicos, sistema Fale Conosco
Fonte: Autoria Própria