UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

RASTREABILIDADE DE PONTO DE FUNÇÃO

Paulo José Azevedo Vianna Ferreira

Orientador
Prof. Dr. Márcio de Oliveira Barros

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2010

RASTREABILIDADE DE PONTO DE FUNÇÃO

Paulo José Azevedo Vianna Ferreira

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO), APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA.

Aprovada por:

Márcio de Oliveira Barros, D.Sc. – PPGI/UNIRIO

Eber Assis Schmitz, Ph.D. – NCE/UFRJ

Sean Wolfgand Matsui Siqueira, D.Sc. – PPGI/UNIRIO

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2010

Ferreira, Paulo José Azevedo Vianna.
F383 Rastreabilidade de ponto de função / Paulo José Azevedo Vianna Ferreira, 2010.
xiii, 82f. + DVD

Orientador: Márcio de Oliveira Barros.
Dissertação (Mestrado em Informática) – Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2010.

1. Sistemas de informação. 2. Rastreabilidade de requisitos. 3. Análise de ponto de função. 4. Análise de impacto de mudanças. I. Barros, Márcio de Oliveira. II Universidade do Estado do Rio de Janeiro (2003-). Centro de Ciências Exatas e Tecnologia. Curso de Mestrado em Informática. III Título.

CDD - 004.2

“Write, & find ecstasy in writing! Work, & be our bed in working! [...]
Hold! Hold! Bear up in thy rapture; [...]
Ah! Ah! What do I feel? Is the word exhausted?
[...]
But exceed! exceed!”

Liber AL vel Legis

AGRADECIMENTOS

À minha esposa Aline, que me apoiou com muita paciência no decorrer desta pesquisa, compreendendo a necessidade de tamanho afinco, além de ter sido responsável por inúmeras revisões e comentários.

Aos meus pais José Maria e Joselícia, diligentes na minha educação e que, apesar das diversas reclamações sobre a minha ausência, compreenderam a importância desta pesquisa para meu desenvolvimento pessoal e profissional.

Ao Prof. Dr. Márcio de Oliveira Barros, orientador valoroso, que mostrou o caminho a ser trilhado, manteve-me perseverante ao ultrapassar as dificuldades e reconheceu meu esforço para alcançar os resultados.

Aos professores do PPGI/UNIRIO, muitos dos quais também foram meus professores de graduação, que inúmeras vezes responderam minhas perguntas por email, *chat* ou pelos corredores.

Ao Prof. Dr. Luiz Amâncio Machado de Sousa Jr., ex-professor da minha graduação, que praticamente me intimou a fazer a inscrição para o Mestrado e auxiliou imensamente com as formalizações matemáticas.

Aos professores membros das bancas dos “temidos” Seminários de Acompanhamento Discente, que ofereceram importantes contribuições para a evolução da pesquisa.

Aos demais alunos do Mestrado em Informática, que compartilharam informações, felicidades e ânsias durante esse período.

Às inúmeras pessoas do meio profissional e pessoal, que contribuíram com informações e sugestões no decorrer da pesquisa.

À UNIRIO, pela sua importância na minha formação profissional, tendo sido a instituição responsável pelo meu curso de graduação e agora, pela oportunidade no Mestrado.

À Cultura Inglesa, pelo voto de confiança e compreensão das minhas ausências para aulas, estudos e apresentações.

Ao povo brasileiro, por patrocinar o ensino nas instituições públicas.

FERREIRA, Paulo José Azevedo Vianna. **Rastreabilidade de Ponto de Função**. UNIRIO, 2010. 82 páginas. Dissertação de Mestrado. Programa de Pós-Graduação em Informática, UNIRIO.

RESUMO

O desenvolvimento de *software* pode obter interessantes benefícios por meio da rastreabilidade de requisitos, incluindo facilidades na análise de impacto, melhor entendimento de um programa, facilidades na manutenção, reuso de componentes e avaliação do progresso do desenvolvimento. Entretanto, apesar da existência de ferramentas acadêmicas e comerciais de suporte à rastreabilidade, ela não é amplamente adotada nas organizações devido a pressões por prazos, falta de ferramentas totalmente automáticas e outros desafios. Sistemas de Informação (SI) não são uma exceção a esse cenário. Algumas organizações de desenvolvimento de SI possuem dificuldades para argumentar claramente sobre o custo de alterar funcionalidades apesar da sua capacidade de estimar o custo de criação de um novo SI usando a Análise de Ponto de Função (APF). Elas não conseguem afirmar precisamente quais códigos-fonte sofrem impacto por mudanças nos requisitos. Como a APF necessita de informações obtidas dos usuários, a pesquisa propõe uma técnica para rastrear as funções de dados e transacionais da APF para o código-fonte. Isso poderia trazer um novo recurso na negociação entre organizações de desenvolvimento de SI e seus clientes, além de poder ser generalizado para outros benefícios da rastreabilidade. A técnica proposta tem como princípio fundamental requerer informações básicas dos *stakeholders*, deixando o trabalho pesado por conta de processos automatizados.

Palavras-chave: Rastreabilidade de Requisitos, Análise de Ponto de Função, Análise de Impacto de Mudanças, Sistemas de Informação

ABSTRACT

Software development can achieve interesting benefits through requirements traceability, including easier impact analysis, improved program comprehension, easier maintenance, components reuse and evaluation of development progress and completeness. Despite the existence of academic and commercial tools to support traceability, it is not widely adopted in organizations due to deadline pressures, lack of fully automated tools, among other challenges. Information Systems (IS) are not an exception to the former scenario. Some IS development organizations complain on their inability to clearly argument on the cost of changing features in spite of their ability to estimate the cost of creating a new IS by using Function Point Analysis (FPA). They cannot accurately state which source codes are impacted by changes in requirements. As FPA requires information collectable from users, the research proposes a specific technique of tracing data and transactional functions from FPA to the source code. It could bring a new resource on negotiation between IS development organizations and their clients, as long as to be generalized to other benefits of traceability. The main guideline of the proposed technique is to require basic information from stakeholders leaving the hard work to automated processes.

Keywords: Requirements Traceability, Function Point Analysis, Change Impact Analysis, Information Systems

ÍNDICE

I. Introdução 1

1.1. Motivação 1

1.2. Objetivo 3

1.3. Avaliação da Proposta 4

1.4. Organização do trabalho 5

II. Rastreabilidade de Requisitos 6

2.1. Definições 9

2.2. Benefícios 10

2.3. Dificuldades 13

2.4. Classificação 15

2.4.1. Rastreabilidade pré e pós-especificação 16

2.4.2. Rastreabilidade horizontal e vertical 16

2.5. Representação 17

2.5.1. Meta-modelos 19

2.6. Técnicas 21

2.6.1. Abordagens 22

2.6.2. Ferramentas 25

2.7. Considerações finais 26

III. Rastreabilidade de Ponto de Função 28

3.1. Análise de Ponto de Função 30

3.2. Uma Proposta de Rastreabilidade de Ponto de Função 34

3.2.1. Informações de entrada 35

3.2.2. Etapa das Funções de Dados 37

3.2.3. Etapa das Funções Transacionais 41

3.2.4. Pesos das ligações 45

3.3. Considerações finais 47

IV. Avaliação e Implementação da Proposta 48
4.1. Estudos de Caso 49
4.1.1. Compiere 49
4.1.1.1. Execução 50
4.1.1.2. Análise 52
4.1.2. Moodle 54
4.1.2.1. Execução da Etapa das Funções de Dados 54
4.1.2.2. Execução da Etapa das Funções Transacionais 56
4.1.2.3. Automatização de testes 58
4.1.2.4. Análise 60
4.1.3. Sistema de Inscrições Online 62
4.1.3.1. Execução da Etapa das Funções de Dados 63
4.1.3.2. Execução da Etapa das Funções Transacionais 64
4.1.3.3. Análise 65
4.2. Protótipo 67
4.3. Considerações finais 71
V. Conclusão 73
5.1. Contribuições 75
5.2. Limitações 75
5.3. Trabalhos Futuros 77
VI. Referências Bibliográficas 79

ÍNDICE DE FIGURAS

Figura II-1: Rastreabilidade pré e pós-especificação (GOTEL e FINKELSTEIN, 1994) 16

Figura II-2: Rastreabilidade horizontal e vertical (GOTEL, 1995 *apud* GENVIGIR, 2009, p. 40) 17

Figura II-3: Meta-modelo de rastreabilidade de RAMESH e JARKE (2001) 20

Figura II-4: Meta-modelo de rastreabilidade de TORANZO *et al.* (2002) 21

Figura II-5: Meta-modelo de rastreabilidade de GENVIGIR (2009) 21

Figura III-1: Visão geral da técnica proposta 34

Figura III-2: Ligação entre as funções de dados e o esquema de banco 38

Figura III-3: Ligação entre o esquema de banco e o código-fonte 38

Figura III-4: Pesos das ligações das funções de dados 39

Figura III-5: Exemplo de cálculo do peso de ligações entre tabelas e partes de código 39

Figura III-6: Ligação indireta entre as funções de dados e o código-fonte 39

Figura III-7: Exemplo de cálculo do peso de ligações entre funções de dados e partes de código 40

Figura III-8: Ligação entre as funções transacionais e os casos de teste 42

Figura III-9: Ligação entre os casos de teste e o código-fonte 42

Figura III-10: Pesos das ligações das funções transacionais 43

Figura III-11: Exemplo de cálculo do peso de ligações entre casos de teste e partes de código 43

Figura III-12: Ligação indireta entre as funções transacionais e o código-fonte 44

Figura III-13: Exemplo de cálculo do peso de ligações entre funções transacionais e partes de código 45

Figura IV-1: Dados obtidos no estudo de caso sobre o Compiere 52

Figura IV-2: Interface do Selenium IDE 59

Figura IV-3: Dados obtidos na etapa das funções de dados do estudo de caso sobre o Moodle 60

Figura IV-4: Dados obtidos na etapa das funções transacionais do estudo de caso sobre o Moodle 61

Figura IV-5: Dados obtidos na etapa das funções de dados do estudo de caso sobre o SIO 66

Figura IV-6: Dados obtidos na etapa das funções transacionais do estudo de caso sobre o SIO 66

Figura IV-7: Interface do FPT Plugin 68

ÍNDICE DE TABELAS

Tabela II-1: Exemplo de matriz de rastreabilidade 19
Tabela III-1: Complexidade das Funções de Dados 31
Tabela III-2: Complexidade das Funções Transacionais 32
Tabela III-3: Contribuição em PFNA 32
Tabela III-4: Características gerais do sistema 33
Tabela IV-1: Dados consolidados dos estudos de caso 72

LISTA DE SIGLAS

AIE: Arquivos de Interface Externa

ALI: Arquivos Lógicos Internos

APF: Análise de Ponto de Função (ou FPA, de Function Point Analysis)

CRM: Customer Relationship Management

CE: Consultas Externas

DDL: Data Definition Language

DID: Descrição de Itens de Dados

DML: Data Manipulation Language

EE: Entradas Externas

ERP: Enterprise Resource Planning

FAV: Fator de Ajuste de Valor

GPL: GNU General Public License

GTI: Grau Total de Influência

IDE: Integrated Development Environment

IREQ: Regras Entre-Requisitos (Inter-REQuirement rules)

LMS: Learning Management System

PFA: Ponto de Função Ajustados

PFNA: Ponto de Função Não Ajustados

ROM: Regras Requisitos-para-Modelo (Requirement-to-Object-Model rules)

SDL: Storage Definition Language

SE: Saídas Externas

SI: Sistemas de Informação (ou IS, de Information Systems)

SIO: Sistema de Inscrições Online (sistema avaliado em um dos estudos de caso)

TAR: Tipos de Arquivo Referenciado

TED: Tipos de Elemento Dado

TER: Tipos de Elemento Registro

VDL: View Definition Language

I. Introdução

1.1. Motivação

Atualmente as organizações têm a necessidade de respostas rápidas da TI às demandas de negócio para torná-las cada vez mais competitivas. A qualidade dos projetos, dos processos, dos sistemas e da informação entregue são fatores de extrema relevância para suportar a competitividade dessas organizações. Nesse contexto, foram propostas diversas soluções, incluindo padrões e metodologias para desenvolvimento de *software*.

Em padrões para o desenvolvimento de *software*, como o MIL-STD-498 (DOD, 1994), IEEE-830 (IEEE, 1998), CMMI (SEI, 2006) e MPS.BR (SOFTEX, 2009), a Gerência de Requisitos é tida como essencial para o sucesso do projeto. Um dos motivos é que erros na especificação de requisitos e a falta de controle das alterações aplicadas sobre ela fazem com que os erros se propaguem pelo ciclo de desenvolvimento, sendo encontrados apenas na fase de testes. A correção desses erros de forma tardia, na fase de testes, requer um esforço 30 vezes superior ao de correção durante as etapas iniciais do projeto (BOEHM, 1981 *apud* WATKINS e NEAL, 1994, p. 104).

A Rastreabilidade de Requisitos é um dos processos que compõem a Gerência de Requisitos (SEI, 2006). O IEEE-830 considera que uma boa especificação de requisitos deve ser rastreável. O objetivo da rastreabilidade é relacionar entre si os elementos produzidos durante o desenvolvimento de *software*, permitindo entender e seguir a origem dos requisitos pelos refinamentos realizados durante o ciclo de desenvolvimento.

Diversos benefícios podem ser percebidos pelo uso da rastreabilidade, como: maior facilidade de entender o código de um *software* e, por isso, ter maior facilidade para analisar o impacto de mudanças e efetivamente alterá-lo; maior visibilidade das regras de negócio e do andamento do projeto; dentre outras que serão abordadas mais adiante.

Entretanto, seu uso não é comum nas organizações, pois enquanto as técnicas manuais de rastreabilidade são onerosas e suscetíveis a erros, as técnicas automáticas não são maduras o suficiente para reduzir o esforço desta atividade e manter seus resultados consistentes ao longo do tempo (GRECHANIK *et al.*, 2007).

Para que seja alcançado maior retorno, deve-se ter disponíveis ferramentas adequadas e com alto grau de automatização. Assim, o esforço da equipe na criação e manutenção das ligações de rastreabilidade pode deixar de ser tão dispendioso e suscetível a erros. A própria integração com o processo de desenvolvimento e manutenção é uma grande oportunidade para coletar informações de rastreabilidade.

A Análise de Ponto de Função (APF) é uma técnica cujo objetivo é mensurar a complexidade (ou tamanho) de um *software*. Ela é amplamente utilizada para basear estimativas de esforço, tempo e custo para a realização de projetos de desenvolvimento de novos Sistemas de Informação. A métrica numérica obtida como resultado da APF se baseia em descrições de alto nível dos requisitos do *software*, compostos pelos dados manipulados pelo sistema (funções de dados) e pelas transações por ele oferecidas (funções transacionais), ajustados pela complexidade dos requisitos não funcionais.

Entretanto, um sistema está inserido em um ambiente de constante evolução e deve se adaptar a mudanças nas necessidades do negócio (REF). Mudanças são solicitadas em situações como correção de defeitos, extensão e detalhamento das funcionalidades ou melhor desempenho (LEHMAN e RAMIL, 2003). Em determinados cenários são criados projetos de manutenção de *software* com o objetivo de contemplar uma série de mudanças.

Em alguns casos, as estimativas de esforço, tempo e custo para projetos de manutenção de *software* também podem ser produzidas pela APF, quando se consegue perceber que a quantidade de dados manipulados pelo sistema e transações oferecidas por ele está sendo alterada. Entretanto, algumas (e talvez freqüentes) alterações são feitas nos requisitos em um nível mais detalhado que a APF se propõe a modelar, não estando relacionadas diretamente ao tamanho do *software*. Esse caso é chamado manutenção de zero ponto por função (GARMUS e HERRON, 2001). Como exemplos, têm-se alterações no comportamento de uma transação, na regra de validação ou tamanho de um campo.

Para esses casos, as estimativas podem se basear na análise de impacto de mudanças, que é uma forma de estimar os artefatos a serem alterados devido a uma mudança proposta (BOHNER e ARNOLD, 1996). A análise de impacto tem como objetivo mapear o escopo das mudanças para melhor planejamento e avaliação do seu custo/benefício. Ela pode ser tratada por abordagens baseadas em análise de

dependências ou em rastreabilidade (BOHNER E ARNOLD, 1996), que se desdobram em técnicas como (a) referência cruzada, que identifica partes de um programa que contém referências a uma variável ou processo, (b) corte do programa (*program slicing)*, que decompõe o sistema em seus dados e processos, (c) grade de conceitos (*concept lattice*), que procura grupos com atributos comuns, (d) consulta a documentos e especificação, (e) de uso da rastreabilidade (ARNOLD e BOHNER, 1993), dentre outras.

1.2. Objetivo

Um dos benefícios da rastreabilidade é justamente oferecer informações que facilitem a atividade da análise de impacto de mudanças. Consideremos um cenário onde um sistema tenha recebido estimativas iniciais usando a APF, tenha sido desenvolvido, implantado e recentemente tenha recebido uma série de solicitações de mudança dos requisitos. Seria interessante para uma empresa de desenvolvimento de software comprovar sistematicamente o esforço (e custo) necessário para realizar essas alterações, indicando que aquela mudança provoca alterações nos ponto de função do sistema e em determinados artefatos. Também é interessante oferecer à equipe responsável pela manutenção do *software* informações sobre onde no código-fonte estão localizadas as linhas de código que deverão ser alteradas, com terminologia próxima à conhecida pelo usuário, oferecida pela APF.

O objetivo principal do presente trabalho é explorar o relacionamento entre a Análise de Ponto de Função e o código-fonte de um sistema. O foco da pesquisa é indicar o volume de código que deve ser analisado e, possivelmente, alterado quando uma alteração nos requisitos é proposta, além de indicar os requisitos que devem ser avaliados quando um módulo de código-fonte precisa ser alterado.

O presente trabalho propõe abordar esse problema através de uma técnica semi-automática para criação e manutenção de ligações de rastreabilidade entre as funções de dados e o código-fonte e entre as funções transacionais e o código-fonte, relacionando assim os requisitos do sistema ao código-fonte produzido. A técnica deve ser semi-automática para não onerar demasiadamente a equipe com atividades de rastreabilidade, exigindo que as informações de entrada da técnica sejam obtidas pela equipe com esforço relativamente baixo.

Essas ligações devem permitir análises como: (a) identificar as partes do código-fonte que manipulam determinada função de dados; (b) identificar as partes do código-fonte que fazem parte da lógica de processamento de uma determinada função

transacional; e (c) determinar quais funções de dados e funções transacionais estão relacionadas a uma determinada parte de código que esteja sendo analisada.

Além dos benefícios trazidos para a análise de impacto, outros benefícios podem ser alcançados de maneira colateral pelo uso da rastreabilidade entre os dados e transações identificados pela APF e o código-fonte do sistema, como: (a) facilitar a comunicação entre usuários e a equipe técnica, pois ambos lidariam com uma visão única, de alto nível do projeto, oferecida pela APF; (b) facilitar o entendimento do programador sobre a lógica do sistema, devido a uma rápida análise dos artefatos que estão relacionados aos requisitos do sistema; e (c) maior facilidade de manutenção do sistema, devido ao item anterior.

A técnica determina etapas independentes para criação das ligações para as funções de dados e para as funções transacionais. Enquanto a etapa para relacionar as funções de dados ao código-fonte se baseia em uma análise de padrões de ocorrências de *queries* no código-fonte, a etapa para relacionar as funções transacionais ao código-fonte se baseia na monitoração do código executado durante a realização de casos de teste usando uma ferramenta de cobertura de código.

1.3. Avaliação da Proposta

Para avaliar a aplicação da técnica, perceber indícios de suas características e oferecer subsídios para algumas conclusões, foram realizados três estudos de caso.

No primeiro estudo de caso analisamos a criação de ligações de rastreabilidade entre as funções de dados e o código-fonte do sistema Compiere[1], um sistema de *Enterprise Resource Planning* (ERP) e *Customer Relationship Management* (CRM) *open source* escrito em Java.

No segundo estudo de caso analisamos a criação de ligações entre as funções de dados e o código-fonte do sistema Moodle[2] e entre as funções transacionais e o código-fonte de um dos módulos desse sistema. O Moodle é um *Learning Management System* (LMS) *open source* escrito em PHP.

No terceiro estudo de caso realizamos a aplicação completa da técnica proposta, analisando a criação de ligações entre as funções de dados e transacionais do Sistema de Inscrições Online do PPGI/UNIRIO[3] e seu código-fonte. Ele é um sistema escrito em PHP, cujo código-fonte e esquema de banco de dados foram disponibilizados para uso da pesquisa.

[1] http://www.compiere.com
[2] http://moodle.org
[3] http://www.uniriotec.br/~inscricaoppgi

Um protótipo de ferramenta foi produzido para demonstrar como os programadores e os arquitetos poderiam ter acesso às informações de rastreabilidade em suas ferramentas de trabalho. Uma ferramenta freqüentemente utilizada por esses profissionais é o ambiente de desenvolvimento integrado (IDE – *Integrated Development Environment*) de *software*. Para desenvolver essa ferramenta, selecionamos um IDE muito utilizado no desenvolvimento de aplicações Java – o Eclipse[4]. O protótipo construído é um *plugin* para o Eclipse, que facilita a visualização das ligações de rastreabilidade entre as funções de dados e transacionais e o código-fonte.

1.4. Organização do trabalho

Este trabalho está organizado em cinco capítulos: o primeiro compreende esta introdução. O segundo capítulo apresenta o resultado de uma revisão da literatura sobre a área de Rastreabilidade de Requisitos. O terceiro capítulo apresenta a técnica proposta e inclui uma breve apresentação sobre a Análise de Ponto de Função. O quarto capítulo apresenta os estudos de caso e resultados obtidos. Finalmente, o quinto capítulo apresenta algumas conclusões e sugestões de trabalhos futuros.

[4] http://www.eclipse.org

II. Rastreabilidade de Requisitos

O aumento da competitividade entre as organizações levou a uma crescente demanda por respostas rápidas da TI às necessidades de negócio. Sistemas computacionais cada vez mais complexos foram produzidos, oferecendo informações cada vez mais rebuscadas para apoio à tomada de decisão. Uma série de padrões, melhores práticas, processos, técnicas, ferramentas e tecnologias foram desenvolvidas para suportar essa estrutura. Cada vez maior era a preocupação com a qualidade dos processos, dos sistemas e da informação entregue. As décadas de 80 e 90 demonstraram esse cenário pelo surgimento de importantes padrões de qualidade para desenvolvimento de *software*, incluindo o DOD-STD-2167 (DOD, 1985 *apud* DOD, 1988), IEEE-830 (IEEE, 1984) e o CMM (SEI, 1991 *apud* SEI, 2006).

Da mesma forma que a TI precisava evoluir, esses padrões também precisavam refletir essa evolução. O original DOD-STD-2167 evoluiu e foi mesclado a outro padrão militar, sendo compilado como o atual MIL-STD-498 (DOD, 1994). O IEEE-830 sofreu revisões até sua versão atual (IEEE, 1998). O CMM passou a se chamar CMMI e foi reorganizado em três versões, conforme o propósito de utilização: uma para processos de desenvolvimento (SEI, 2006), uma para aquisição (SEI, 2007) e uma para serviços (SEI, 2009). Após um processo de adequação à realidade nacional, o CMMI para desenvolvimento foi mesclado a outros padrões, como o ISO/IEC 12207 e ISO/IEC 15504, chegando à criação em 2005 do padrão MPS.BR, tendo sido atualizado recentemente (SOFTEX, 2009).

O MIL-STD-498 é um padrão para uniformizar os projetos de desenvolvimento de *software* militar, sendo exigido legal e contratualmente pelo Departamento de Defesa dos EUA. Todo desenvolvimento realizado por equipes do governo e por empresas contratadas ou subcontratadas deve estar aderente a este padrão. Ele estabelece um total de 22 descrições de itens de dados (DID) necessárias aos projetos, que são os artefatos que devem ser entregues pelos executantes do projeto. Essas DID podem ser agrupadas em sete categorias: (a) planos, (b) conceitos e requisitos, (c) projeto (*design*), (d) testes, (e) manuais de usuário e operação, (f) manuais de suporte e, naturalmente, o (g) *software*.

O padrão CMMI foi criado como um guia para que as organizações possam aumentar o seu poder de gestão e comprovar sua excelência para o mercado. Ele propõe um conjunto de 22 áreas de processo, que listam as melhores práticas relacionadas a essas áreas. O CMMI distribui as áreas de processo em níveis, que podem ser usados como caminhos para a evolução das organizações ou para graduá-las nas áreas de processo.

Dois tipos de níveis existem no CMMI: (a) níveis de capacidade; e (b) níveis de maturidade. A chamada representação contínua é indicada para a melhoria de processos específicos, conforme a necessidade da organização. Ela distribui as áreas de processo em seis níveis de capacidade, que estão dispostos em uma escala numérica crescente de zero a cinco. A chamada representação por estágios é indicada para a melhoria gradual de um conjunto de áreas de processo. Ela distribui as áreas de processo em cinco níveis de maturidade, que estão dispostos em uma escala numérica crescente de um a cinco. O nível de capacidade zero e o nível de maturidade um indicam a existência de processos incompletos ou a inexistência de processos estruturados na organização.

O MPS.BR (Melhoria de Processo do *Software* Brasileiro) também faz uma distribuição das áreas de processo, mas em sete níveis de maturidade, em uma escala alfabética decrescente de A a G, onde o nível G é o mais básico e o nível A é o mais completo. Os processos no MPS.BR têm seus propósitos e resultados esperados definidos no padrão. A avaliação do nível de maturidade da organização é realizada pela análise dos resultados obtidos para nove atributos de processo (características mensuráveis de um processo). Estes atributos estão distribuídos nos níveis do MPS.BR, determinando quais práticas são necessárias a atingir cada nível.

O IEEE-830 é um padrão para criação e manutenção de especificações de requisitos. Ele determina um conjunto de melhores práticas e informações necessárias em uma especificação de requisitos com qualidade. Ele define que devem ser cobertos aspectos como: (a) funcionalidades, (b) interfaces externas – com outros sistemas ou com usuários, (c) desempenho, (d) atributos do *software* – escalabilidade, portabilidade, segurança, entre outros. e (e) restrições de projeto – como linguagens de programação, limitação de recursos, entre outras. Algumas propriedades de uma boa especificação de requisitos são indicadas, incluindo: (a) correta, (b) não ambígua, (c) completa, (d) consistente, (e) priorizada de acordo com relevância ou estabilidade, (f) verificável, (g) modificável, e (h) rastreável.

A Gerência de Requisitos (SEI, 2006, SOFTEX, 2009) é considerada uma área essencial ao sucesso do projeto. Seus processos são encontrados no primeiro nível de maturidade tanto do MPS.BR (nível G) quanto da representação por estágios do CMMI

(nível 2)[5]. Isso demonstra como a qualidade de uma especificação de requisitos é essencial para o adequado desenrolar do projeto, mantendo custo, prazo, risco e qualidade do produto dentro do planejado. Erros na especificação de requisitos de um *software* podem se propagar até a implementação, por vezes sendo encontrados apenas na fase de testes, o que requer a revisão de todos os artefatos de desenvolvimento, desde a especificação de requisitos até os casos de teste e implementações (BOEHM, 1981 *apud* WATKINS e NEAL, 1994, p. 104).

Dentre os processos da Gerência de Requisitos, a Rastreabilidade de Requisitos (DOD, 1994, IEEE, 1998, SEI, 2006, SOFTEX, 2009) em especial é indicada nos quatro padrões previamente citados. No entanto, este processo apresenta uma série de dificuldades relativas à sua utilização. Suas técnicas manuais são custosas, suscetíveis a erro e difíceis de ser mantidas ao longo do tempo. Não obstante, atualmente não existem técnicas e ferramentas suficientemente maduras para manter informações de rastreabilidade de maneira automática (GRECHANIK *et al.*, 2007).

O MIL-STD-498 determina a necessidade de manter informações de rastreabilidade entre: (a) componentes de sistema e requisitos de sistema, (b) requisitos de itens de configuração e requisitos de sistema, (c) unidades de *software* e requisitos de itens de configuração, (d) casos de teste e requisitos de itens de configuração, (e) casos de teste e requisitos de sistema e (f) códigos-fonte dos itens de configuração e unidades de *software*. A saber, um item de configuração (DOD, 1994) é uma parte de um *software* que cumpre uma função para o usuário, devendo ser configurada, controlada e documentada independentemente, enquanto uma unidade de *software* é um elemento do projeto de um item de configuração. Também devem ser mantidas informações de rastreabilidade nas seguintes descrições de itens de dados (DID): (a) descrição de projeto de sistema/subsistema, (b) descrição de projeto de *software* e (c) especificação de produto de *software*.

O CMMI estabelece a rastreabilidade de requisitos na prática específica de manutenção de rastreabilidade bidirecional dos requisitos. O MPS.BR estabelece a rastreabilidade bidirecional entre requisitos e produtos de trabalho como um dos resultados esperados do processo da Gerência de Requisitos. Por último, o IEEE-830 considera que uma boa especificação de requisitos deve ser rastreável.

[5] Apesar do nível inicial da representação por estágios do CMMI ser o nível 1, este se trata de um nível de referência, indicando que a organização possui processos *ad-hoc* e caóticos, que dependem da habilidade dos profissionais e não da maturidade dos processos. Assim, o nível de maturidade onde começam a ser inseridas as áreas de processo é o nível 2 (gerenciado).

2.1. Definições

Ao longo dos anos foram dadas diversas definições para a rastreabilidade de requisitos, sendo algumas específicas para determinados problemas de pesquisa e outras generalizáveis. GOTEL e FINKELSTEIN (1994) realizaram uma ampla avaliação sobre o problema da rastreabilidade e perceberam que existia muita controvérsia sobre sua definição. Baseados nas definições existentes, incluindo a do IEEE-830 (IEEE, 1984), mais aceita à época, propuseram uma definição que persiste sendo a mais citada e aceita até os dias atuais. A seguir serão discutidas algumas definições para rastreabilidade de requisitos:

- GOTEL e FINKELSTEIN (1994) - "se refere à habilidade de descrever e seguir o ciclo de vida de um requisito tanto para frente quanto para trás (ou seja, das suas origens, através do desenvolvimento e especificação, para sua subseqüente implantação e uso, e através de períodos de refinamento e iteração em quaisquer dessas fases)";
- IEEE-830 (IEEE, 1998) - "uma especificação de requisitos é rastreável se a origem de cada um dos seus requisitos é clara e facilita a referência de cada requisito em desenvolvimentos futuros ou na documentação de melhorias";
- CMMI (SEI, 2006) - "associação entre requisitos e outros requisitos relacionados, implementações e verificações";
- HAMILTON e BEEBY (1991) *apud* RAMESH *et al.* (1997), p. 398 - habilidade de "descobrir a história de cada funcionalidade de um sistema";
- EDWARDS e HOWELL (1992) *apud* RAMESH *et al.* (1997), p. 398 - técnica que "provê um relacionamento entre requisitos, projeto e a implementação do sistema";
- GREENSPAN e MCGOWAN (1978) *apud* RAMESH *et al.* (1997), p. 398 - método para "examinar as técnicas de descrição de sistemas e como elas permitem que mudanças em qualquer parte dos requisitos de projeto, especificação e implementação sejam rastreadas por todo o sistema".

Como se pode perceber pelas definições apresentadas, em especial a de GOTEL e FINKELSTEIN (1994), a rastreabilidade de requisitos se refere à habilidade de identificar as origens de uma especificação de requisitos (reuniões, idéias, decisões de clientes e usuários) e relacioná-las aos produtos de trabalho delas derivados (modelos de análise e projeto, código-fonte, modelos de persistência, casos de teste, manuais, entre outros).

2.2. Benefícios

As técnicas de rastreabilidade, quando devidamente utilizadas, são capazes de prover uma série de benefícios às empresas de desenvolvimento de *software* e seus clientes. Esses benefícios alcançam direta e indiretamente diversos dos profissionais envolvidos nos projetos de desenvolvimento de *software*, assim como usuários finais. A literatura elenca uma série desses benefícios, conforme estão dispostos a seguir.

- Compreensão do *software*: Quando um desenvolvedor analisa um código-fonte com o objetivo de entender seu funcionamento, ele associa seu conhecimento sobre o mundo com as estruturas do *software* (BIGGERSTAFF *et al.*, 1993). Essa associação pode ocorrer de maneira *top-down* (dos conceitos para o *software*), *bottom-up* (do *software* para os conceitos) ou por uma combinação das duas (ANTONIOL *et al.*, 1999). Rastreando documentos de requisitos, análise e projeto de um *software* para o código-fonte pode-se auxiliar desenvolvedores na compreensão do *software* (ANTONIOL *et al.*, 1999).
- Análise de impacto (de alterações): Os requisitos de um *software* freqüentemente são alterados durante e após seu desenvolvimento. Uma mudança nos requisitos pode afetar a documentação, o código-fonte, os casos de teste, entre outros artefatos que compõem o *software* (ANTONIOL *et al.*, 1999). A identificação dos artefatos que precisarão ser alterados (ANTONIOL *et al.*, 1999, KANNENBERG e SAIEDIAN, 2009) para que uma mudança seja realizada é chamada de análise de impacto (ARNOLD e BOHNER, 1993, RAMESH *et al.*, 1997). A rastreabilidade pode oferecer informações sobre o impacto de mudanças, baseando decisões sobre a realização da mudança e estimativas (ANTONIOL *et al.*, 1999).
- Manutenção: Enquanto a análise de impacto indica as alterações necessárias ao atendimento de uma solicitação de mudança que está sendo avaliada, no momento em que a manutenção será implementada, a mesma informação da análise de impacto auxilia a identificação dos documentos, modelos e módulos de código-fonte que devem ser alterados, além dos casos de teste que devem ser novamente executados (WATKINS e NEAL, 1994).
- Identificação de componentes candidatos a reuso: O reuso de *software* é a utilização de *software* existente na construção de um novo *software*. Seu objetivo é aumentar a qualidade e produtividade do *software* desenvolvido (FRAKES e KANG, 2005). Bons projetos de desenvolvimento deveriam ter a preocupação com o reuso. Os conceitos de domínio e as regras de negócio estão presentes na documentação, chegando até o código-fonte e outros

produtos de trabalho criados durante o projeto. Quando a documentação de um *software* é rastreada para seu código-fonte, é proporcionado um importante auxílio para localização de componentes candidatos a reuso. Consultas poderiam ser feitas para identificar os componentes que devem ser preparados quando se deseja que uma funcionalidade existente seja reutilizada (ANTONIOL *et al*., 1999).

- Criação de casos de teste: A fase de testes tem o objetivo de determinar que todos os requisitos do sistema estejam implementados, cumpram seus objetivos e satisfaçam as necessidades dos usuários. Isso é feito verificando se todas as condições ambientais estão sendo cobertas, interagindo corretamente com o usuário, o *hardware* e com interfaces com outros sistemas (IEEE, 2008). Aspectos não funcionais, como desempenho e obediência às melhores práticas, também podem ser observados nos testes. A cobertura de teste (IEEE, 2008) é uma métrica que pode avaliar quanto os testes abrangem os requisitos e, sob outra ótica, quanto os testes abrangem os códigos produzidos. A identificação de áreas de código responsáveis pela implementação de uma funcionalidade específica auxilia a percepção dos requisitos que não estejam cobertos pelos testes, além dos códigos que não estejam relacionados a requisitos (ANTONIOL *et al*., 2002).
- Progresso da implementação: Documentos de especificação descrevem os requisitos que devem se tornar documentos e modelos de análise e projeto, sendo implementados no código-fonte e posteriormente testados. MACMILLAN e VOSBURGH (1986) *apud* RAMESH *et al.* (1997) demonstram o uso da rastreabilidade para determinar quanto os requisitos foram implementados em um *software*, um indicador de progresso da implementação. Isso é feito por meio de uma matriz de rastreabilidade, que relaciona os requisitos a funções do *software* existentes nos códigos. Da mesma forma, rastreando-se requisitos para documentos de análise e projeto, pode-se identificar quantos requisitos foram analisados e projetados ou rastreando-se códigos para testes, realizar a análise de cobertura de testes (KANNENBERG e SAIEDIAN, 2009).
- Verificação e validação: Nas etapas de verificação e validação de um projeto de desenvolvimento de *software* é analisado se os requisitos foram apropriadamente documentados, implementados e testados (KANNENBERG e SAIEDIAN, 2009, WATKINS e NEAL, 1994). O objetivo dessas etapas é garantir que o sistema atende seus requisitos e foi corretamente implementado. Segundo KANNENBERG e SAIEDIAN (2009), é nas etapas de verificação e validação que a rastreabilidade demonstra seus maiores benefícios, pois facilita

a identificação de onde os requisitos foram documentados, implementados e testados. Entretanto, conforme indicam os trabalhos de RAMESH e DHAR (1992) e CURTIS *et al.* (1988) *apud* RAMESH *et al.* (1997), para comprovar o cumprimento dos requisitos é necessário não apenas mapear os produtos originados por eles, mas também suas origens nas discussões com os *stakeholders*, o que dá origem a mais um benefício da rastreabilidade: a visibilidade do projeto, que será discutida a seguir.

- Visibilidade: Problemas na comunicação entre os envolvidos em um projeto freqüentemente provocam atrasos, estouros de orçamento, produtos que não atendem aos requisitos e cancelamento de projetos (RAMESH *et al.*, 1997). A rastreabilidade oferece um importante recurso para que a origem de cada requisito seja corretamente entendida e propagada para os demais produtos de trabalho, indicando a importância desse requisito, como foi implementado e testado. Com esse avanço, ocorre uma melhora na comunicação entre os clientes e a equipe técnica, reduzindo os problemas decorrentes de uma comunicação deficiente (KANNENBERG e SAIEDIAN, 2009).
- Satisfação dos clientes: Devido à maior visibilidade dos clientes sobre o progresso do projeto, ao acompanhamento do prazo esperado e à garantia de que eles terão o produto que solicitaram, aumenta-se a satisfação dos clientes (KANNENBERG e SAIEDIAN, 2009).
- Avaliação e aceitação de projetos: Estimativas de tamanho, escopo, recursos e custo de um novo projeto de desenvolvimento de *software* podem ser feitas com base nos requisitos deste *software*. Entretanto, em um projeto de mudança de um *software* existente é necessário avaliar o impacto dessas mudanças nos componentes da versão atual do *software*. Como a rastreabilidade auxilia na análise de impacto, podem ser oferecidas estimativas mais precisas aos responsáveis pela avaliação e aprovação de projetos (RAMESH *et al.*, 1997).
- Auditoria: A auditoria em um projeto de desenvolvimento de *software* tem como objetivo garantir que um conjunto de produtos de trabalho está compatível com critérios específicos, por exemplo, requisitos (SEI, 2006). Dentre as atividades realizadas, pode ser avaliado se os requisitos foram devidamente implementados e validados, garantindo que o cliente receberá o produto que solicitou. Também pode ser avaliado se existem casos de teste adequados aos requisitos, garantindo que o produto terá a qualidade esperada. Com ligações de rastreabilidade que demonstrem o cumprimento dos requisitos e testes,

pode-se comprovar que os requisitos estão validados e foram testados, chegando a uma melhor avaliação na auditoria (WATKINS e NEAL, 1994).

- Redução de custos de correção de defeitos: O custo de correção de defeitos na fase de testes é considerado 30 vezes superior ao custo de correção durante as etapas iniciais do processo de desenvolvimento de *software* (BOEHM, 1981 *apud* WATKINS e NEAL, 1994). A rastreabilidade auxilia no entendimento e visibilidade dos requisitos. Com isso, inconsistências são encontradas e podem ser resolvidas antes de provocarem maiores problemas na documentação, na implementação e nos testes (WATKINS e NEAL, 1994).

2.3. Dificuldades

Apesar de a rastreabilidade oferecer os benefícios citados na seção anterior e ser uma prática indicada por padrões como o CMMI e o MPS.BR, nem sempre ela é tida como um processo obrigatório para a produção de um *software*, muitas vezes tratada como uma atividade de suporte, que influencia na qualidade do produto final. Com a pressão pelo cumprimento de prazos, a atividade de manutenção de ligações de rastreabilidade acaba sendo deixada de lado (GOTEL e FINKELSTEIN, 1994). Isso aliado à imaturidade das técnicas e ferramentas de rastreabilidade automática atuais, ao grande esforço e suscetibilidade a erros das técnicas manuais de rastreabilidade e à falta de treinamento e conscientização dos profissionais envolvidos, torna a rastreabilidade um processo que recebe pouca adesão das organizações. A seguir, serão discutidos em mais detalhes esses desafios e dificuldades.

- Entendimentos divergentes: Executivos tendem a perceber a rastreabilidade como um processo necessário apenas para cumprir padrões de desenvolvimento (RAMESH, 1998). Enquanto isso, alguns analistas e desenvolvedores conseguem entender a importância da rastreabilidade e buscam implementá-la adequadamente. Entretanto, grande parte dos profissionais envolvidos em desenvolvimento de *software* não está consciente da importância da rastreabilidade. A melhor forma de evitar os diferentes entendimentos sobre a rastreabilidade seria estabelecer uma visão única sobre ela para todos os projetos da organização (KANNENBERG e SAIEDIAN, 2009), incluindo esforços de conscientização sobre sua relevância e benefícios. É importante que os profissionais envolvidos e beneficiados com o processo de rastreabilidade sejam conscientizados da sua importância para que se alcance com sucesso seus benefícios (RAMESH *et al.*, 1997).

- Falta de treinamento: Técnicas de rastreabilidade não são comumente abordadas em cursos de graduação e treinamentos corporativos. Devido à falta de treinamento, quando profissionais despreparados são inseridos em atividades relacionadas à rastreabilidade, acabam por se sentirem sobrecarregados, com baixa produtividade e tendo impressão de estarem desempenhando atividades desnecessárias, sem claros benefícios (KANNENBERG e SAIEDIAN, 2009).
- Custo: O custo de se seguir processos de rastreabilidade aumenta proporcionalmente à complexidade do *software* (HEINDL e BIFFL, 2005). Inicialmente, as atividades de rastreabilidade podem dar a impressão de aumentar o custo dos projetos. Entretanto, considerando que problemas nos requisitos são detectados de maneira mais rápida quando é feita a rastreabilidade, através do seu uso consegue-se reduzir o custo com correções e atrasos nos projetos. Outros benefícios da rastreabilidade, como facilitar mudanças, acabam por provocar o retorno do investimento no médio e longo prazo. Para reduzir o custo de se adotar processos de rastreabilidade pode-se adotar a abordagem baseada em valor, onde os requisitos são classificados por ordem de relevância (valor) e apenas os mais relevantes são mapeados para os demais produtos de trabalho (KANNENBERG e SAIEDIAN, 2009).
- Técnicas manuais: Conforme mencionado previamente, o custo de se manter a rastreabilidade aumenta proporcionalmente à complexidade do *software*. Isso se deve ao esforço necessário para identificar os produtos de trabalho derivados dos requisitos e manter a rastreabilidade consistente à medida que mudanças são solicitadas ao longo do tempo. Quando a rastreabilidade é controlada por processos manuais, com sua documentação residindo em documentos textuais, planilhas ou ferramentas específicas (CLELAND-HUANG *et al.*, 2007), contendo inúmeros elementos a serem rastreados (requisitos, modelos, códigos etc.). Assim, o esforço e o tempo necessários para manter as ligações de rastreabilidade atualizadas são altos e a atividade é suscetível a erros (CLELAND-HUANG *et al.*, 2007, KANNENBERG e SAIEDIAN, 2009). Encontrar esses erros se torna uma atividade ainda mais difícil.
- Falta de maturidade das ferramentas: Uma boa ferramenta de apoio à rastreabilidade deveria oferecer facilidades para reduzir o esforço necessário à criação e manutenção das informações de rastreabilidade de um projeto, contemplando todos os tipos de produtos de trabalho e as alterações motivadas por mudanças nos requisitos ou em outros produtos de trabalho. Entretanto, as ferramentas comerciais disponíveis atualmente ainda são

simplistas (RAMESH e JARKE, 2001), não suportando a automatização de grande parte das atividades de rastreabilidade e o controle de mudanças nos requisitos. A maior parte das ferramentas automáticas ainda está em fase de pesquisa, disponível apenas no meio acadêmico. Uma pequena parcela das organizações que realizam atividades de rastreabilidade o faz por meio de ferramentas construídas internamente para atender às suas próprias necessidades, não as disponibilizando publicamente (RAMESH e JARKE, 2001). A falta de maturidade das ferramentas é um problema que havia sido identificado por GOTEL e FINKELSTEIN (1994) e que, apesar de inúmeras pesquisas, continua persistindo, conforme declaram KANNENBERG e SAIEDIAN (2009). Com isso, ainda é necessário que estudos sejam feitos para que as ferramentas rastreabilidade alcancem um nível de maturidade que permita sua ampla utilização.

- Controle de mudanças: A falta de maturidade das técnicas e ferramentas atuais de rastreabilidade leva a uma dificuldade em controlar as alterações motivadas por mudanças nos requisitos ou em outros produtos de trabalho. As técnicas manuais exigem muito esforço e tempo para a realização deste controle e são suscetíveis a erros (CLELAND-HUANG *et al.*, 2007, KANNENBERG e SAIEDIAN, 2009). Ademais, exigem grande comprometimento dos profissionais responsáveis por manter as informações de rastreabilidade. As ferramentas de mercado auxiliam na identificação dos potenciais impactos, mas não são capazes de automatizar a atualização das ligações, necessitando que isso seja feito manualmente (KANNENBERG e SAIEDIAN, 2009).

2.4. Classificação

A rastreabilidade deve ser bidirecional para permitir rastrear (para frente) dos requisitos para seus sucessivos refinamentos e rastrear (para trás) dos refinamentos até sua origem. Essa propriedade é necessária a todas as técnicas de rastreabilidade. Caso contrário, seria considerado um recurso de utilização limitada (GENVIGIR, 2009). Ainda segundo GENVIGIR (2009), existem duas principais classificações para a rastreabilidade: (a) pré e pós-especificação de requisitos[6] e (b) horizontal e vertical.

[6] Os termos originais de GOTEL e FINKELSTEIN (1994) são "*pre-requirements specification traceability*" e "*post-requirements specification traceability*". Como os prefixos se referem à especificação, consideramos mais adequada a tradução "rastreabilidade pré-especificação de requisitos" e "rastreabilidade pós-especificação de requisitos", ao contrário de alguns trabalhos em português que utilizam a tradução "pré-rastreabilidade" e "pós-rastreabilidade".

2.4.1. Rastreabilidade pré e pós-especificação

A definição de rastreabilidade de requisitos presente no padrão IEEE-830 (IEEE, 1998) considera que "uma especificação de requisitos é rastreável se a origem de cada um dos seus requisitos é clara e facilita a referência de cada requisito em desenvolvimentos futuros ou na documentação de melhorias". Analisando essa definição, já existente na primeira versão do IEEE-830 (IEEE, 1984), GOTEL e FINKELSTEIN (1994) perceberam que há a necessidade de não apenas rastrear os produtos de trabalho do ciclo de desenvolvimento de *software* (como a especificação de requisitos, documentos de análise, projeto, códigos, testes etc.), mas também identificar as origens dos requisitos diretamente nas discussões sobre as necessidades de clientes, anteriores à produção da especificação de requisitos.

Essa necessidade levou à classificação da rastreabilidade em dois tipos, dependendo da sua relação com a especificação de requisitos: a rastreabilidade pré-especificação de requisitos e a rastreabilidade pós-especificação de requisitos (Figura II-1). A rastreabilidade pré-especificação se refere aos "aspectos do ciclo de vida de um requisito anteriores à sua inclusão na especificação" e a rastreabilidade pós-especificação se refere aos "aspectos do ciclo de vida de um requisito que resultam da sua inclusão na especificação" (GOTEL e FINKELSTEIN, 1994).

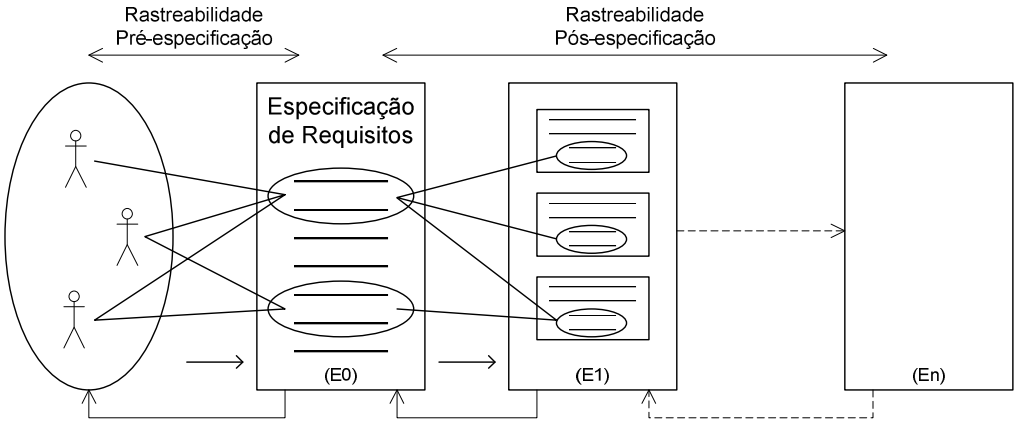


Figura II-1: Rastreabilidade pré e pós-especificação (GOTEL e FINKELSTEIN, 1994)

2.4.2. Rastreabilidade horizontal e vertical

De acordo com a Figura II-1, as discussões entre *stakeholders* fundamentam as regras determinadas pela especificação de requisitos. Os requisitos são propagados para os demais produtos de trabalho que, com o avanço do projeto, terminam por estar relacionados desde sua origem até a etapa final do

desenvolvimento. Entretanto, a evolução dos requisitos, motivada por mudanças nas necessidades dos clientes e usuários, provoca a realização de alterações nos demais produtos de trabalho. O controle da evolução das versões de um mesmo produto de trabalho em uma determinada fase do ciclo de desenvolvimento é feito pela rastreabilidade horizontal. O controle dos refinamentos de um produto de trabalho para o produto seguinte é feito pela rastreabilidade vertical (Figura II-2).

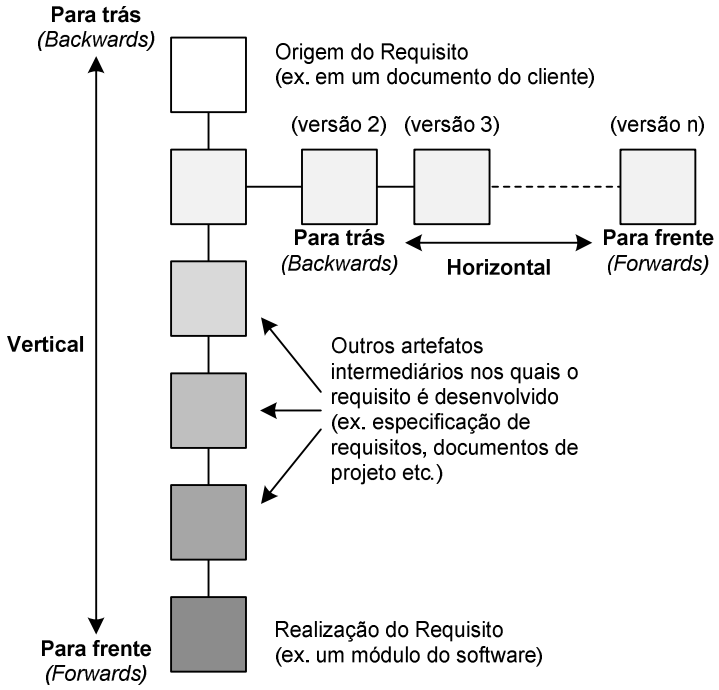


Figura II-2: Rastreabilidade horizontal e vertical (GOTEL, 1995 *apud* GENVIGIR, 2009, p. 40)

2.5. Representação

Cada organização que se proponha a utilizar técnicas de rastreabilidade precisa decidir quais benefícios pretende alcançar através do uso desta técnica. Dependendo desses benefícios, as técnicas de rastreabilidade devem ser capazes de representar os tipos de informação necessários a alcançá-los. Em determinadas situações, é necessário representar os usuários e sua influência sobre os requisitos, porque e quando os requisitos serão necessários e as características do projeto que será desenvolvido. Essas são informações de contexto relevantes para o entendimento de como foi dada origem à especificação de requisitos. SAYÃO e LEITE (2005) propõem quatro níveis de informação que auxiliam na classificação, entendimento e contextualização das informações de rastreabilidade, são eles: (a) ambiental (leis, padrões etc.), (b) organizacional (regras, processos etc.), (c) gerencial (tarefas, restrições etc.) e (d) desenvolvimento (requisitos, diagramas, programas etc.).

Genericamente pode-se entender que as informações de rastreabilidade são representadas como ligações bidirecionais entre dois elementos. Esses elementos são de um tipo específico. Uma ligação geralmente relaciona elementos de tipos distintos, mas também é possível relacionar elementos do mesmo tipo.

- Tipo: é uma "forma fundamental comum a todos os indivíduos de uma mesma espécie", segundo o dicionário Michaelis (WEISZFLOG, 2010). Os tipos de elemento na rastreabilidade são classes de produtos de trabalho e suas subdivisões, que influenciam ou são influenciados pelos requisitos. Tópicos de discussão de reuniões, requisitos, classes, atributos e métodos de diagramas de classes, arquivos de código-fonte, classes, atributos, métodos e parâmetros de código-fonte, casos de teste e testes de unidade são exemplos de tipos de elemento rastreáveis.
- Elemento: é uma "parte integrante e fundamental de uma coisa" (WEISZFLOG, 2010). Os elementos na rastreabilidade são instâncias dos tipos que tenham sido produzidos durante o processo de desenvolvimento. A discussão sobre um determinado requisito, o requisito funcional *nº 418* no documento de especificação de requisitos, a classe *"Candidato"* no diagrama de classes, seu atributo *"nome"* e método *"inscreve()"*, sua implementação no código-fonte e o caso de teste "*Verifica se o candidato está inscrito*" são exemplos de elementos rastreáveis.
- Ligação: é o "ato ou efeito de ligar", "junção, união, laço, vínculo" (WEISZFLOG, 2010). As ligações de rastreabilidade determinam que um elemento está relacionado a outro elemento, indicando que, de alguma forma, existe um refinamento ou relação entre eles. Como exemplo de refinamento, a implementação da classe *"Candidato"* no código-fonte pode ser rastreada a partir da classe *"Candidato"* no diagrama de classes. Como exemplo de relação, o requisito funcional *"O sistema não pode permitir a inscrição de um candidato já inscrito*" está relacionado ao requisito *"O sistema deve emitir um relatório de inscritos*".

As técnicas de rastreabilidade existentes representam informações de rastreabilidade de maneiras diversas, baseando-se na idéia de ligações entre elementos. GENVIGIR (2009) cita uma série de técnicas existentes para expressar as ligações de rastreabilidade: (a) referência cruzada, (b) matrizes, (c) dependência de frases-chaves, (d) integração de documentos, (e) hipertextos, (f) grafos, (g) métodos formais, (h) esquemas dinâmicos, (i) suposição da manutenção da verdade e (j) redes de confiança. Uma forma bem comum de representação dos elementos e ligações na

literatura é sob a forma de matrizes de rastreabilidade, conforme exemplificado na Tabela II-1.

Tabela II-1: Exemplo de matriz de rastreabilidade

	REQ #9	REQ #61	REQ #111	REQ #220	REQ #418	REQ #777
Candidato.php		✓	✓	✓		✓
Inscricao.php	✓			✓	✓	✓
Curso.php	✓			✓	✓	✓
Regiao.php		✓				✓

O exemplo demonstra que o arquivo *"Regiao.php"* está ligado aos requisitos *"REQ#61"* e *"REQ#777"*, indicando que as regras destes dois requisitos motivaram a criação desse arquivo. No sentido contrário da ligação, esse arquivo implementa as regras destes dois requisitos.

2.5.1. Meta-modelos

Até o momento foi abordado como representar as informações de rastreabilidade, com foco nos elementos e ligações. Nesta seção, serão abordadas quais informações de rastreabilidade devem ser representadas, com foco nos tipos de elementos. A literatura apresenta meta-modelos capazes de determinar quais informações devem ser rastreadas. Durante a argumentação sobre seu meta-modelo, GENVIGIR (2009) destaca dois meta-modelos existentes na literatura: RAMESH e JARKE (2001) e TORANZO *et al.* (2002). Esses dois meta-modelos e o de GENVIGIR (2009) serão discutidos a seguir.

- RAMESH e JARKE (2001): capturam informações de três dimensões: fontes (origem dos requisitos), interessados (pessoas envolvidas ou beneficiadas) e objetos (objetos relacionados aos produtos de trabalho gerados no desenvolvimento). Existem quatro tipos de ligações, que podem ser classificadas como relacionadas ao produto e relacionadas ao processo. As ligações de satisfação e dependência são relacionadas ao produto, descrevendo as propriedades dos objetos e os relacionamentos entre eles. As ligações de evolução e razão são relacionadas ao processo, representando o histórico das ações executadas no processo (Figura II-3).

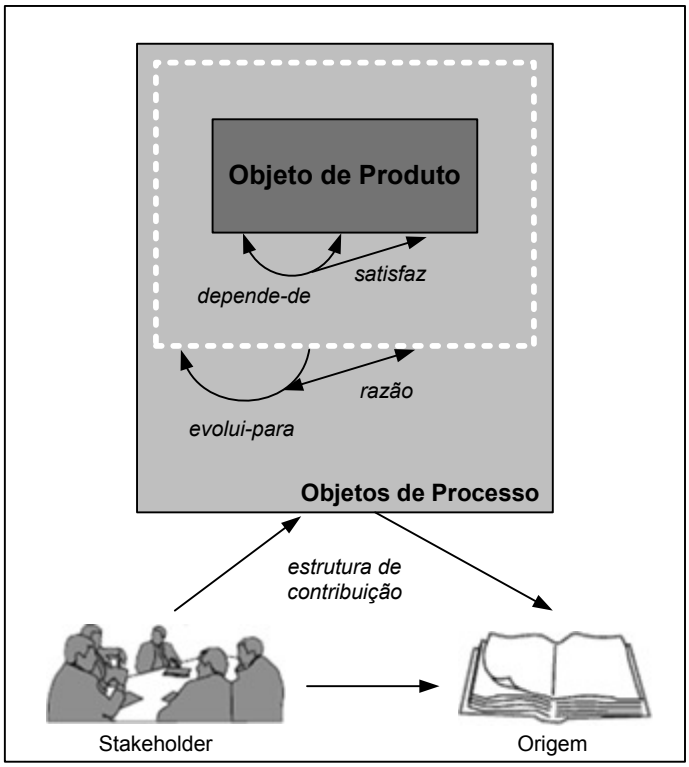


Figura II-3: Meta-modelo de rastreabilidade de RAMESH e JARKE (2001)

- TORANZO *et al.* (2002): capturam informações de quatro dimensões: ambiental (o ambiente onde a organização está inserida, com suas leis e padrões), organizacional (características da organização, objetivos e regras), gerencial (associam tarefas a requisitos e restrições) e desenvolvimento (relacionada aos produtos de trabalho gerados durante o desenvolvimento). Representa seis diferentes tipos de ligação: (a) ligações de satisfação, que indicam que o elemento destino deve satisfazer o elemento origem; (b) de recurso, que indicam que o elemento destino é um recurso do elemento origem; (c) de responsabilidade, demonstrando a responsabilidade, participação ou ação humana sobre um produto de trabalho; (d) de representação, demonstrando a modelagem dos requisitos em outra linguagem; (e) de alocação, indicando que o elemento origem está inserido (é um subsistema) no elemento destino; e, finalmente, (f) de agregação, indicando uma composição de elementos (Figura II-4).

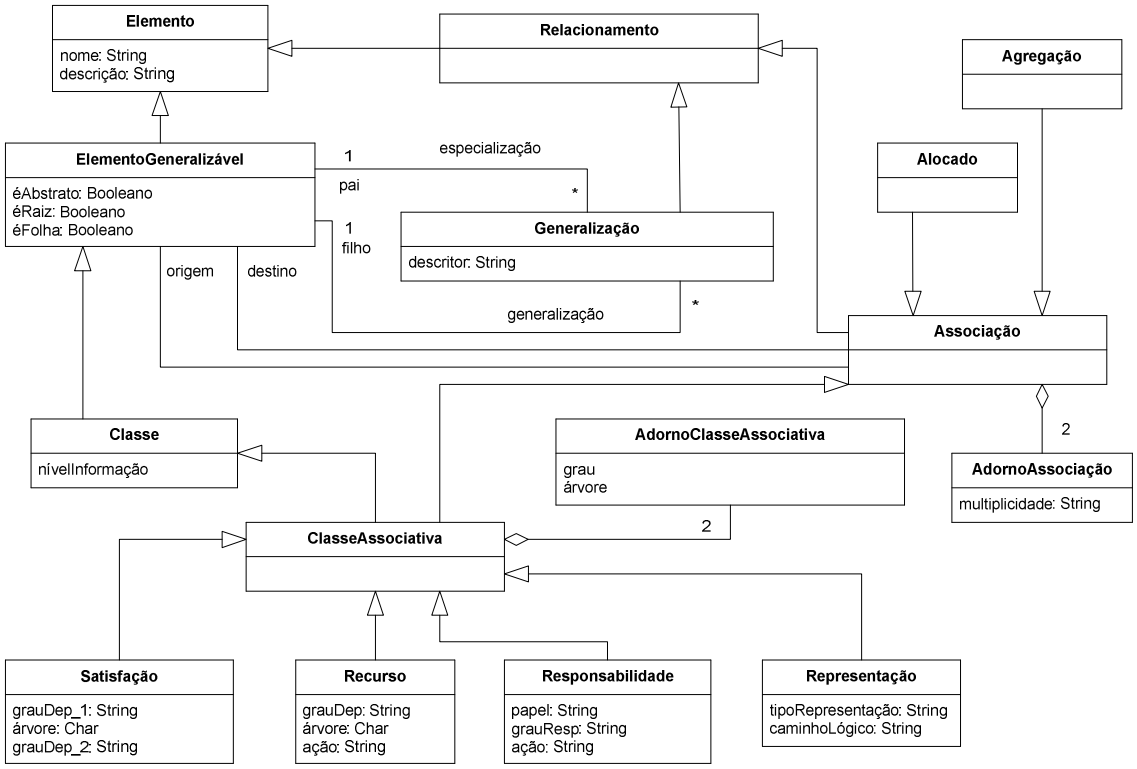


Figura II-4: Meta-modelo de rastreabilidade de TORANZO *et al.* (2002)

- GENVIGIR (2009): propõe um meta-modelo para representação de informações de rastreabilidade altamente generalizável. Ele define uma estrutura básica que pode ser estendida de acordo com as necessidades de cada organização. O meta-modelo é composto por quatro elementos: tipo de artefato (define um tipo de elemento, podendo também representar um tipo de ligação), atributos do tipo (propriedades inerentes a um tipo de artefato), artefato (os artefatos propriamente ditos) e instância (valores dos atributos do tipo) (Figura II-5).

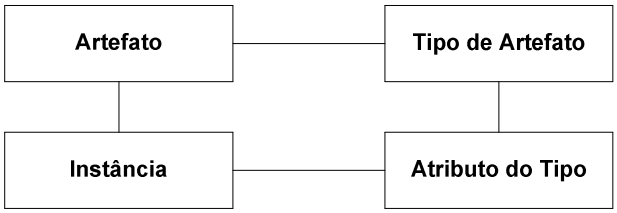


Figura II-5: Meta-modelo de rastreabilidade de GENVIGIR (2009)

2.6. Técnicas

Após mais de 30 anos de pesquisas em rastreabilidade de requisitos, a literatura recente indica que este ainda é um problema em aberto, onde ainda não

existe uma solução definitiva que possa ser aplicada em todas as situações e com garantia de alta produtividade. Entretanto, muitos avanços foram feitos ao longo desses anos, com a criação de uma série de técnicas e ferramentas úteis para a prática da rastreabilidade em contextos específicos.

2.6.1. Abordagens

ROCHIMAH *et al.* (2007) fazem uma avaliação das técnicas de rastreabilidade mais conhecidas, baseando-se na revisão de aproximadamente cem artigos recentes relacionados à área de rastreabilidade de requisitos. Como resultado, eles isolam e comparam as características de cada técnica, chegando às seguintes abordagens: (a) baseada em valor, (b) baseada em cenários, (c) recuperação de informação, (d) baseada em modelos de características, (e) baseada em eventos, (f) baseada em regras, e (g) baseada em hipertexto. A seguir, estas abordagens serão discutidas isoladamente, apresentando suas características.

- Baseada em valor: sua fundamentação está no paradigma da engenharia de *software* baseada em valor, que defende que o valor agregado por cada requisito, caso de uso e defeito deve ser levado em consideração no processo de desenvolvimento (EGYED *et al.*, 2005). HEINDL e BIFFL (2005) apresentam um estudo de caso onde a criação de ligações de rastreabilidade é realizada em 5 fases: (a) definição de requisitos, (b) priorização, (c) empacotamento, onde são formados grupos de requisitos, (d) ligação, e (e) avaliação. EGYED *et al.* (2005) avalia a relação custo/benefício de se aumentar a qualidade das ligações de rastreabilidade de acordo com três aspectos-chave: (a) granularidade dos artefatos, (b) valor dos artefatos, e (c) momento de realização da rastreabilidade (no início ou após o projeto). Ao final, sugere que a qualidade das informações de rastreabilidade aumenta em uma proporção inferior ao o esforço despendido para prover as informações de entrada. Essa conclusão é validada pelo uso da sua ferramenta de rastreabilidade, chamada TraceAnalyzer.
- Baseada em cenários: é fundamentada na idéia de que a identificação dos trechos de código acionados durante a execução de cenários de utilização (casos de teste) oferece rastros (*footprints*), os quais podem ser usados para gerar ligações de rastreabilidade entre esses trechos de código e os cenários. Isso é demonstrado por EGYED (2001) e EGYED e GRÜNBACHER (2002), que realizam um processo composto pelas etapas de definição de hipóteses,

atomização, generalização e refino pelos quais os rastros são submetidos, chegando a ligações de rastreabilidade entre cenários, elementos de modelo e do código. BREITMAN (2000) propõe um *framework* para evolução de cenários que, dentre seus requisitos, exige que sejam rastreados cenários para outros produtos de trabalho do desenvolvimento, como a especificação de requisitos e os diagramas de classes.

- Recuperação de informação: faz uso de técnicas de mineração de texto para comparar elementos e gerar ligações entre eles. Seu uso é intensivo para rastreamento entre documentos de especificação, modelos de análise e de projeto e código. Comumente faz uso de modelos probabilísticos ou de espaços vetoriais para comparação entre os elementos. ANTONIOL *et al.* (1999, 2000, 2002) demonstram o uso dessa abordagem na geração de ligações entre documentos de requisitos e código. GRECHANIK *et al.* (2007) geram ligações entre elementos de diagramas de casos de uso e do código-fonte. Essa abordagem tem como premissa que sejam dados nomes significativos para os elementos dos documentos, modelos e, principalmente, do código, facilitando a comparação (ANTONIOL *et al.*, 1999). Além das previamente citadas, existe vasta literatura sobre essa abordagem (CLELAND-HUANG *et al.*, 2005 *apud* ROCHIMAH *et al.*, 2007, MARCUS e MALETIC, 2003, PINHEIRO e GOGUEN, 1996, RAMESH e DHAR, 1992).
- Baseada em modelos de características: modelos de características (*feature models*) descrevem uma visão geral sobre os requisitos e a variabilidade de uma linha de produto. Eles são utilizados para que um cliente possa descrever um produto sob o seu ponto de vista (ROCHIMAH *et al.*, 2007). É um modelo onde as características são os vértices e os relacionamentos entre características são as arestas de um grafo. PASHOV e RIEBISH (2004) *apud* ROCHIMAH *et al.* (2007) e RIEBISH (2004) *apud* ROCHIMAH *et al.* (2007) descrevem uma abordagem para gerar ligações de rastreabilidade entre requisitos, características e outros produtos de trabalho gerados pelo processo de desenvolvimento de *software*. Três tipos de características são representados pelo modelo: (a) funcionais, (b) de interface, e (c) parâmetros. Estas são ligadas por três tipos de relações: (a) hierárquicas, (b) refinamentos, e (c) restrições entre características que possuem alguma influência nas decisões sobre o produto. Essas restrições podem ser dos tipos: requer, exclui ou agrupamento-múltiplo.
- Baseada em eventos: ao contrário das demais, o foco desta abordagem não é propor um método para geração e manutenção de ligações de rastreabilidade,

mas o gerenciamento de ligações consistentes (ROCHIMAH *et al.*, 2007). Uma relação *publisher-subscriber* é estabelecida entre os elementos participantes de uma ligação de rastreabilidade, onde uma relação de dependência entre dois elementos é representada pela inscrição de um elemento no elemento do qual depende. Com o objetivo de gerenciar este mecanismo de publicação, é proposto por CLELAND-HUANG *et al.* (2002a, 2002b, 2003) *apud* ROCHIMAH *et al.* (2007) um processo e *framework* de controle de eventos, que auxilia na análise de impacto de mudanças. O método é composto por um (a) gerenciador de requisitos, (b) servidor de eventos, e (c) gerenciador de publicação.

- Baseada em regras: utiliza regras heurísticas para manter ligações de rastreabilidade com alto grau de automatização. SPANOUDAKIS (2002) propõe o uso de regras para rastrear documentos em linguagem natural (especificações de requisitos, casos de uso, dentre outros) e modelos UML. Dois tipos de regras são usados: requisitos-para-modelo (ROM) e entre-requisitos (IREQ)[7]. Enquanto as regras ROM geram ligações entre requisitos e objetos de modelo pela identificação de *lexicons* e análise de similaridade entre padrões sintáticos de ambos, as regras IREQ geram ligações entre os elementos de um documento de requisitos ou entre os elementos de diferentes tipos de documentos de requisitos. O processo consiste de quatro etapas: (a) marcação gramatical, (b) conversão em XML, (c) ligação de artefatos, e (d) ligação de partes dos artefatos (ROCHIMAH *et al.*, 2007).
- Baseada em hipertexto: é fundamentada na idéia que as ligações de rastreabilidade podem ser representadas por *hiperlinks* entre um elemento âncora e um elemento alvo (ex. causais, não-causais e navegações). A abordagem faz uso da linguagem XML para representação de elementos e ligações. Ligações complexas e versionamento são propostas por MALETIC *et al.* (2003a, 2003b) *apud* ROCHIMAH *et al.* (2007) e a geração de ligações por meio de hipermídia e integração de informação é proposta por SHERBA *et al.* (2003) *apud* ROCHIMAH *et al.* (2007) e SHERBA (2005) *apud* ROCHIMAH *et al.* (2007).

[7] As regras heurísticas de SPANOUDAKIS (2002) são originalmente chamadas em inglês, "*requirement-to-object-model*" (ROM) e "*inter-requirement*" (IREQ) *rules*. Enquanto os nomes das regras foram traduzidos para facilitar a compreensão durante a leitura, seus acrônimos foram mantidos com a grafia inglesa para facilitar os leitores que porventura desejem fazer pesquisas por essas expressões em motores de busca.

2.6.2. Ferramentas

O INCOSE (International Council on Systems Engineering) mantém constantes pesquisas sobre Engenharia de *Software*, incluindo uma lista[8] de 25 ferramentas de Gerência de Requisitos, onde um dos seus critérios de avaliação é a funcionalidade de manter informações de rastreabilidade. Sua última atualização ocorreu em março de 2010. É mantida também uma taxonomia de ferramentas[9], onde existe uma hierarquia iniciada na Engenharia de *Software*, passando por Engenharia[10], Engenharia de Requisitos, Gerenciamento de Requisitos, chegando à Rastreabilidade de Requisitos. Sua última atualização ocorreu em setembro de 2004.

Dentre as ferramentas encontradas nas pesquisas realizadas pelo INCOSE, foram selecionadas quatro: CORE[11], Cradle[12], DOORS[13] e RequisitePro[14], todas possuindo recursos de rastreabilidade. O critério utilizado para seleção destas quatro ferramentas foi analisar o relatório de consolidado[15] das pesquisas realizadas pelo INCOSE sobre as 25 ferramentas de Gerência de Requisitos listadas. As ferramentas menos conhecidas foram eliminadas, restando apenas cinco. Dessas, a Enterprise Architect foi avaliada como fraca em captura de requisitos e rastreabilidade, sendo descartada. Restaram quatro ferramentas que oferecem suporte ao gerenciamento de requisitos e permitem algumas análises como será abordado a seguir.

A suíte CORE da Vitech Corporation oferece suporte ao gerenciamento de requisitos, análise de comportamento, arquitetura, simulação, testes, verificação, validação e comprovação do completo desenvolvimento e documentação. A suíte permite rastrear requisitos para elementos de arquitetura. Como exemplo, ela pode ser usada para mapear processos de negócio para sistemas, seus módulos e serviços (*web services*), além de diagramas de classes para componentes (VITECH, 2008). Devido ao foco desta suíte em modelos, a principal forma de visualização das ligações de rastreabilidade é por meio de um diagrama hierárquico, possuindo também consultas e relatórios personalizáveis.

A suíte Cradle da 3SL (Structured Software Systems Ltd.) oferece suporte ao gerenciamento de requisitos, análise e projeto, modelagem de processos de negócio,

8 http://www.incose.org/ProductsPubs/products/rmsurvey.aspx
9 http://www.incose.org/ProductsPubs/products/SETools/tooltax/se_tools_taxonomy.html
10 O termo “Engenharia” nessa taxonomia se refere realmente a uma subárea da Engenharia de Software, dentre os outros termos: gerenciamento, compartilhamento de informações e suporte a infra-estrutura.
11 http://www.vitechcorp.com
12 http://www.threesl.com/pages/products/overview.php
13 http://www-01.ibm.com/software/awdtools/doors
14 http://www-01.ibm.com/software/awdtools/reqpro
15 http://www.incose.org/ProductsPubs/products/INCOSERMToolSurveyConsolidatedResults.xls

arquitetura, análise de desempenho, métricas, verificação, validação, testes, riscos e configuração (controle de mudanças e versões). Elementos de documentos de todo ciclo de vida podem ser rastreados (CRADLE, 2008), como especificações de requisitos funcionais e de sistema, funções, processos, modelos ou mesmo sistemas inteiros. As ligações podem ser estabelecidas direta ou indiretamente, a última por meio de ligações transitivas (GERSTING, 1995). A visualização das ligações de rastreabilidade pode ser feita por meio de árvores, matrizes, diagramas hierárquicos ou referências cruzadas, além de consultas e relatórios personalizáveis. A suíte é capaz de importar dados de ferramentas externas de Engenharia de *Software* como Rational Rose, ClearCase, DOORS, RequisitePro e outras, a partir de documentos Word, Excel e outros. Além de fazer engenharia reversa de código-fonte (C, Ada e Pascal) gerando modelos controlados pela suíte.

A linha de produto IBM Rational inclui as ferramentas RequisitePro e DOORS. Ambas oferecem suporte ao gerenciamento de requisitos e permitem o seu relacionamento com documentos externos, construídos através de ferramentas especializadas. O RequisitePro utiliza matrizes e árvores e o DOORS utiliza *hiperlinks* para visualização e manutenção das ligações de rastreabilidade (SAYÃO e LEITE, 2005). Um aspecto a ressaltar sobre o RequisitePro é sua capacidade de registrar o raciocínio dos interessados no projeto (*stakeholders*), anterior ao levantamento de requisitos, por meio de discussões sobre os requisitos que são registradas na ferramenta.

De maneira geral, todas as ferramentas citadas se propõem a apoiar a transformação de um conjunto de requisitos em um produto composto por diversos elementos por elas gerenciados. Elas ainda permitem a atuação dos interessados no projeto em um ambiente colaborativo, garantindo que todos tenham acesso a informações consistentes e atualizadas.

2.7. Considerações finais

Conforme abordado neste capítulo, o objetivo da rastreabilidade é relacionar os elementos produzidos durante o desenvolvimento de *software*, permitindo entender a origem dos requisitos e segui-la pelos refinamentos realizados durante o ciclo de desenvolvimento.

Foram analisados diversos benefícios da realização da rastreabilidade em projetos de *software*, incluindo: (a) compreensão do *software*, (b) análise de impacto, (c) manutenção, (d) identificação de componentes candidatos a reuso, (e) criação de casos de teste, (f) progresso da implementação, (g) verificação e validação, (h)

visibilidade, (i) satisfação dos clientes, (j) avaliação e aceitação de projetos, (k) auditoria, e (l) redução de custos com *bugs*.

Entretanto, algumas dificuldades relacionadas ao uso da rastreabilidade foram expostas, como: (a) entendimentos divergentes, (b) falta de treinamento, (c) custo, (d) técnicas manuais, (e) falta de maturidade das ferramentas, e (f) controle de mudanças. Estas dificuldades fazem da rastreabilidade um problema em aberto (KANNENBERG e SAIEDIAN, 2009), com potencial para ser explorado por pesquisas nessa área.

Mais de 30 anos de pesquisas tiveram como resultado uma série de abordagens para manutenção de informações de rastreabilidade, variando de técnicas manuais a automáticas, aplicáveis do início até o término do ciclo de desenvolvimento, da origem dos requisitos aos seus refinamentos e podendo relacionar diversas versões de um mesmo elemento.

Foram expostas ainda classificações das informações oferecidas pelas técnicas de rastreabilidade como: pré e pós-especificação de requisitos, horizontal e vertical, genericamente como as informações são representadas: tipo, elemento e ligação, os meta-modelos de representação de GENVIGIR (2009), RAMESH e JARKE (2001) e TORANZO *et al.* (2002), bem como diversas abordagens e uma análise de algumas ferramentas comerciais.

III. Rastreabilidade de Ponto de Função

Conforme foi abordado no capítulo anterior, a rastreabilidade de requisitos oferece diversos benefícios às organizações. Entretanto, ela não é largamente utilizada devido à falta de ferramentas adequadas, maduras e que possuam alto grau de automatização, não sendo capazes de aumentar significativamente a produtividade do trabalho de rastreabilidade (KANNENBERG e SAIEDIAN, 2009). Em um cenário de constante pressão por prazos, as informações de rastreabilidade tendem a se deteriorar, pois a equipe começa a falhar na tarefa de manter estas ligações atualizadas (GOTEL e FINKELSTEIN, 1994).

A existência de ferramentas adequadas para automatização das tarefas de rastreabilidade é essencial para o alcance dos seus benefícios (CLELAND-HUANG *et al.*, 2007). Elas podem fazer com que o esforço de criação e manutenção das ligações de rastreabilidade não seja tão dispendioso e suscetível a erros quanto o derivado de um processo manual. A integração com as ferramentas nativas da equipe de projeto e o processo de desenvolvimento e manutenção de *software* é importante para a coleta de ligações de rastreabilidade (CLELAND-HUANG *et al.*, 2007).

A falta de controle de mudanças nos requisitos é uma das principais causas de falha em projetos de *software* (CERPA e VERNER, 2009). Mudanças nos requisitos afetam diretamente o sucesso do projeto, pois envolvem alterações no planejamento, renegociação de prazos e custos e outros efeitos adversos sobre o projeto. As organizações estão começando a perceber que existe a necessidade de controlar e estimar o esforço de alterações nos requisitos de sistemas, mas as ferramentas e técnicas atuais não atendem completamente a esta expectativa (LAVAZZA e VALETO, 2000), mesmo já sendo capazes de estimar o esforço para a implementação dos requisitos usando a Análise de Ponto de Função (APF) e outros métodos.

A APF pode ser usada para estimar o esforço de alterações nos requisitos quando pode ser percebida uma mudança na quantidade de dados ou transações do sistema. Entretanto, mudanças mais sutis podem não afetar a quantidade de ponto de função do sistema, pois estão em um nível de detalhe mais refinado do projeto. Esse

caso é conhecido como manutenção de zero ponto por função (GARMUS e HERRON, 2001).

A identificação das atividades necessárias ao atendimento de uma solicitação de mudança é chamada análise de impacto (LAVAZZA e VALETO, 2000). Em uma abordagem manual, a análise de impacto de mudanças nos requisitos seria feita com o apoio de um arquiteto, analista e/ou programador que possuíssem conhecimento sobre o projeto. Eles analisariam a documentação e possivelmente o código-fonte do sistema, procurando identificar quais seriam os artefatos impactados e estimar o esforço (e o tempo) necessário para que as alterações fossem realizadas.

A proposta descrita neste capítulo tem o objetivo de auxiliar a análise de impacto de mudanças nos requisitos. Em primeira instância, os beneficiados seriam gerentes de projeto, analistas, arquitetos e programadores envolvidos em atividades de manutenção de sistemas. Busca-se alcançar esse objetivo oferecendo a esses profissionais informações de rastreabilidade de requisitos obtidas de forma semi-automática, sem onerar significativamente a equipe. Devido à ampla utilização da APF em projetos de desenvolvimento de Sistemas de Informação (SI) e à mesma representar um modelo de alto nível dos requisitos do sistema, percebeu-se o potencial de oferecer informações de rastreabilidade que determinassem onde os requisitos (funções de dados e funções transacionais) identificados pela APF estão implementados no código-fonte, auxiliando assim na análise de impacto.

Essas informações são oferecidas sob a forma de dois tipos de ligações de rastreabilidade: entre funções de dados e partes do código-fonte e entre funções transacionais e partes do código-fonte. A técnica proposta divide o código-fonte em partes que podem ter diferentes níveis de granularidade, identificando de arquivos de código-fonte (granularidade grossa) até linhas de código individuais (granularidade fina)[16]. As ligações entre funções de dados ou transacionais e partes de código-fonte recebem um peso, que busca determinar, para um requisito sob análise, que arquivos de código-fonte (ou quaisquer partes de código-fonte) são mais relevantes para a sua implementação, facilitando a identificação dos prováveis pontos de alteração caso seja necessária uma mudança relacionada com o requisito.

Com essas ligações, acreditamos que a análise de impacto de mudanças nos requisitos em projetos de SI seria facilitada. Alguns benefícios colaterais da

[16] Após algumas pesquisas sobre o termo granularidade em diversos contextos, a melhor tradução encontrada para os termos originais em inglês "*coarse-grained*" e "*fine-grained granularity*" foi "granularidade grossa" e "fina". A granularidade grossa indica que um elemento será dividido em partes maiores, com menor nível de detalhe e quantidade. A granularidade fina indica que um elemento será dividido em partes menores, com maior nível de detalhe e quantidade.

rastreabilidade também poderiam ser alcançados, apesar do presente trabalho não se propor a oferecer diretamente esses benefícios e nem apresentar maiores evidências no sentido de indicar que possam ser atingidos com o uso da técnica proposta. São eles: (a) melhor entendimento de código-fonte do sistema; (b) maior facilidade na manutenção do sistema; e (c) apoio na identificação de componentes reutilizáveis.

A Seção 3.1 apresenta uma introdução à APF. A Seção 3.2 e suas subseções descrevem o processo pelo qual a técnica proposta cria ligações bidirecionais de rastreabilidade. A Seção 3.2.1 determina as informações de entrada que a técnica precisa obter sobre o SI. A Seção 3.2.2 detalha o processo para as ligações entre as funções de dados e os artefatos de código-fonte. A Seção 3.2.3 para as ligações entre as funções transacionais e os artefatos de código-fonte. A Seção 3.2.4 apresenta detalhes sobre os pesos das ligações. Ao final, a seção 3.3 observa brevemente algumas considerações gerais sobre a proposta.

3.1. Análise de Ponto de Função

Originalmente, a APF foi utilizada pela IBM na realização de medições de produtividade e estimativas de esforço (ALBRECHT, 1979). Entretanto, ao longo do tempo ela se tornou um método largamente utilizado em projetos de desenvolvimento de SI para realização de estimativas não apenas de esforço, mas de tempo, custos e recursos necessários. Essas estimativas são relevantes na negociação entre empresas de desenvolvimento de *software* e seus clientes.

A APF leva em consideração dados e transações do sistema reconhecíveis pelos usuários (ALBRECHT e GAFFNEY, 1983, GARMUS e HERRON, 2001), o que unifica a linguagem usada na negociação com os clientes. Duas perspectivas sobre o sistema devem ser obtidas dos usuários pelo analista: os dados manipulados (funções de dados) e as transações oferecidas pelo sistema (funções transacionais)[17].

A estimativa pela APF é dividida em duas fases principais: o cálculo do número de ponto de função não ajustados (PFNA) e o ajuste do anterior para alcançar o número de ponto de função ajustados (PFA), com a aplicação de 14 características gerais do sistema pré-definidas que levam em consideração aspectos não funcionais do sistema. O esforço necessário ao desenvolvimento do sistema é obtido a partir do número de PFA. Cada função de dados e cada função transacional possuem individualmente certo número de PFNA, que quando somados, chegam ao total de PFNA do sistema.

[17] Os termos “funções de dados” e “funções transacionais” são traduções dos termos originais em inglês “*data functions*” e “*transactional functions*”, conforme apresentado em IFPUG (2010).

As funções de dados são identificadas pela análise de quais são os dados (entidades) manipulados ou consultados pelo sistema. Estes são organizados em grupos de dados logicamente relacionados. Os grupos de dados podem ser classificados em Arquivos Lógicos Internos (ALI), quando são mantidos pelo próprio sistema (inserindo novos dados ou modificando dados existentes no grupo), ou Arquivos de Interface Externa (AIE), quando são apenas consultados pelo sistema que está sendo analisado, mas são mantidos por outro sistema.

Os grupos de dados estão relacionados a subgrupos de dados, denominados Tipos de Elemento Registro (TER), e possuem atributos atômicos, não repetitivos e reconhecíveis pelo usuário, denominados Tipos de Elemento Dado (TED). A complexidade de uma função de dados é baseada na quantidade de TER e TED que possuem e determinada em uma escala ordinal de três valores (baixa, média, alta) de acordo com a Tabela III-1.

Tabela III-1: Complexidade das Funções de Dados

	Arquivos Lógicos Internos / Arquivos de Interface Externa		
	1-19 TED	**20-50 TED**	**51+ TED**
1 TER	Baixa	Baixa	Média
2-5 TER	Baixa	Média	Alta
6+ TER	Média	Alta	Alta

As funções transacionais são processos elementares que manipulam dados à medida que estes atravessam as fronteiras do sistema. Elas podem ser classificadas em três tipos: Entradas Externas (EE), quando são processos que recebem dados de fora do sistema, e Saídas Externas (SE) ou Consultas Externas (CE), quando são processos que enviam dados para fora do sistema. O que diferencia uma SE de uma CE é a sua dependência de processamento complexo dos dados. Quando há a necessidade de fórmulas matemáticas, criação de dados derivados, conversões de valores equivalentes, filtragem e seleção de dados por múltiplos critérios, reordenação ou reorganização de dados, análises condicionais ou a alteração do estado do sistema, a função transacional é classificada como SE. Caso contrário, ela é classificada como CE (GARMUS e HERRON, 2001).

A complexidade de uma função transacional é baseada (a) na sua classificação como EE, SE ou CE; (b) na quantidade de grupos de dados distintos que ela manipula, denominados Tipos de Arquivo Referenciado (TAR); e (c) na quantidade de TED que ela obtém do ou exibe para o usuário. Essa complexidade é determinada em uma escala ordinal de três valores (baixa, média, alta) de acordo com a Tabela III-2.

Tabela III-2: Complexidade das Funções Transacionais

	Entradas Externas		
	1-4 TED	**5-15 TED**	**16+ TED**
0-1 TAR	Baixa	Baixa	Média
2 TAR	Baixa	Média	Alta
3+ TAR	Média	Alta	Alta
	Saídas Externas / Consultas Externas		
	1-5 TED	**6-19 TED**	**20+ TED**
0-1 TAR	Baixa	Baixa	Média
2-3 TAR	Baixa	Média	Alta
4+ TAR	Média	Alta	Alta

A contribuição individual em PFNA de cada função de dados e transacional para o total de PFNA do sistema é derivada de suas complexidades. Esta conversão de uma escala qualitativa (complexidade) para uma escala quantitativa (PFNA) tem como base não apenas a complexidade da função, mas também sua classificação como ALI, AIE, EE, SE ou CE, como pode ser observado na Tabela III-3.

Tabela III-3: Contribuição em PFNA

	ALI	**AIE**	**EE**	**SE**	**CE**
Baixa	7	5	3	4	3
Média	10	7	4	5	4
Alta	15	10	6	7	6

O total de PFNA do sistema é obtido pela soma das contribuições individuais de todas as funções de dados e transacionais, de acordo com a fórmula abaixo, onde ALI_a, AIE_b, EE_c, SE_d e CE_e são a contribuição individual em PFNA de cada uma das funções e as variáveis *f*, *g*, *h*, *i* e *j* são a quantidade de ALI, AIE, EE, SE e CE, respectivamente.

$$PFNA = \sum_{a=1}^{f} ALI_a + \sum_{b=1}^{g} AIE_b + \sum_{c=1}^{h} EE_c + \sum_{d=1}^{i} SE_d + \sum_{e=1}^{j} CE_e$$

A influência dos requisitos não funcionais sobre a complexidade final de desenvolvimento de um sistema deve ser levada em consideração. Para isso, os graus de influência de 14 características gerais do sistema (Tabela III-4) devem ser avaliados para que seja ajustada a quantidade total de PFNA obtida para o sistema, gerando o número de ponto de função ajustados (PFA). Um grau de influência é uma nota em uma escala numérica de zero a cinco, que deve ser atribuída a cada uma das 14 características, variando de nenhuma influência (zero) a altamente influente (cinco).

Tabela III-4: Características gerais do sistema

1	Mecanismos de comunicação
2	Processamento distribuído
3	Desempenho crítico
4	Execução em ambiente pesado
5	Alta freqüência de transações
6	Entrada de dados online
7	Interface com o usuário complexa
8	Atualização online dos arquivos
9	Processamento complexo
10	Reusabilidade
11	Facilidade de instalação
12	Facilidade de operação
13	Usável em diversas empresas
14	Facilidade na mudança de requisitos

O grau total de influência (GTI) é um índice formado pela soma dos graus atribuídos a todas as características gerais do sistema. Formalmente, o GTI pode ser definido pela fórmula abaixo, onde N_i é o grau atribuído à *i-ésima* característica.

$$GTI = \sum_{i=1}^{14} N_i \, , 0 \leq N_i \leq 5$$

O GTI é usado para calcular o fator de ajuste de valor (FAV), um índice que representa quanto é estimado que os requisitos não-funcionais devem influenciar na complexidade de desenvolvimento do sistema. Essa influência pode fazer com que os PFNA tenham uma redução de até 35% ou um aumento de até 35% em relação ao seu valor original. Tal ajuste pode ser percebido pela fórmula abaixo, recordando que o GTI pode variar entre zero e setenta e, conseqüentemente, o FAV pode variar entre 0,65 e 1,35.

$$FAV = 0{,}65 + 0{,}01 \cdot GTI$$

Finalmente, a quantidade de PFA representa uma estimativa da complexidade de desenvolvimento do sistema considerando seus requisitos funcionais e os ajustes não-funcionais. Os PFA são obtidos calculando-se o produto dos PFNA pelo FAV, conforme demonstrado na fórmula a seguir.

$$PFA = PFNA \cdot FAV$$

A APF oferece um modelo que descreve os dados utilizados e as transações oferecidas por um sistema. Esse modelo habilita as organizações a estimar o esforço, tempo e custo de projetos de criação de um novo *software*. Dessa maneira, a APF é útil tanto para a identificação inicial dos requisitos de um sistema quanto para negociações entre a empresa de desenvolvimento de *software* e seus clientes.

3.2. Uma Proposta de Rastreabilidade de Ponto de Função

O modelo de ponto de função representa os requisitos de um SI como um conjunto de funções de dados (ALI e AIE) e funções transacionais (EE, SE e CE). A seguir, será descrito como criar ligações bidirecionais de rastreabilidade entre os ALI e os artefatos de código-fonte e entre as funções transacionais e o código-fonte.

A Figura III-1 oferece uma visão geral da técnica proposta, que divide em duas etapas a criação das ligações de rastreabilidade. Essas duas etapas não são interdependentes e, por isso, podem ser realizadas paralelamente. São elas: (a) uma busca feita no código-fonte procurando em quais partes deste estão inseridos os comandos que manipulam cada uma das funções de dados; e (b) uma análise da execução do código-fonte para identificar quais partes do mesmo oferecem suporte a cada uma das funções transacionais.

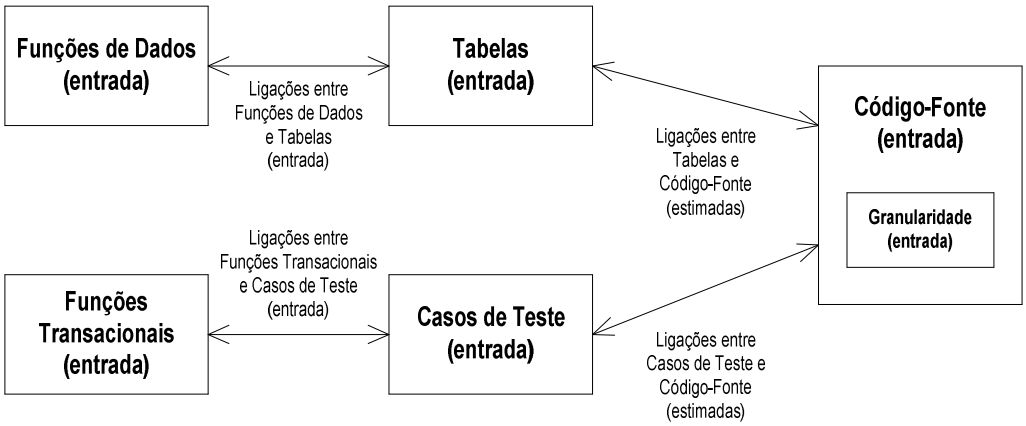


Figura III-1: Visão geral da técnica proposta

O código-fonte do sistema é necessário em ambas as etapas da técnica. Entrando na etapa das funções de dados, é necessário conhecer as funções de dados e seus relacionamentos com as tabelas do banco de dados. A quantidade de ocorrências de tabelas no código-fonte do sistema é usada para estimar o relacionamento entre as funções de dados e o código-fonte. Entrando na etapa das funções transacionais, é necessária uma lista de funções transacionais e seus

relacionamentos com casos de teste. A quantidade de linhas acionadas no código-fonte durante a execução de um caso de teste é usada para estimar o relacionamento entre as funções transacionais e o código-fonte. A granularidade em que o código-fonte será rastreado também precisa ser informada.

Ao realizar um trabalho de rastreabilidade, deve-se decidir o nível de granularidade apropriado em que serão estabelecidas as ligações. Por exemplo, na técnica proposta, referente ao código-fonte, podem ser estabelecidas ligações para diretórios, pacotes, arquivos, classes, métodos, estruturas de programação (*i.e.* iterações, condições etc.) e linhas de código individuais. Esses são diferentes níveis de granularidade em que a estrutura do código-fonte pode ser analisada e entendida. Sendo assim, conforme recomenda CLELAND-HUANG *et al.* (2007), foi necessário definir em quais destes níveis de granularidade serão criadas e mantidas as ligações de rastreabilidade ao utilizar a técnica proposta.

A escolha por um nível de granularidade mais grosso ou mais fino para a criação de ligações de rastreabilidade influencia no esforço para a criação destas ligações e no retorno, em termos de informação, provido para os desenvolvedores por estas ligações de rastreabilidade. Esses aspectos foram extensamente analisados no trabalho de EGYED *et al.* (2005) e não fazem parte do escopo do presente trabalho. Ainda, essa escolha depende das necessidades de cada projeto e organização, além das ferramentas disponíveis para auxiliar na automatização da atividade de criação de ligações. A técnica proposta permite a livre escolha por um ou outro nível de granularidade do código-fonte, de acordo com a necessidade específica. Nos estudos de caso realizados na Seção 4.1 foi optou-se pela granularidade de arquivo.

3.2.1. Informações de entrada

A técnica proposta é classificada como semi-automática, pois gera as ligações por meio de mecanismos computacionais automatizados, mas necessita de algumas informações que somente podem ser obtidas manualmente. A seguir, são detalhadas as informações de entrada necessárias à aplicação da técnica:

- Funções de dados: Tipo de elemento que representa os dados utilizados pelo sistema, identificados durante o levantamento das funções de dados da APF. É composto por uma lista dos ALI manipulados pelo sistema. Os AIE não precisam ser informados, pois a técnica não considera dados mantidos por outros sistemas e consultados pelo sistema sob análise, considerando apenas os dados mantidos pelo próprio sistema.

- Tabelas do banco de dados: Tipo de elemento que representa a estrutura de persistência do sistema, composto por uma lista com os nomes das tabelas que formam seu banco de dados.
- Rastreabilidade entre as funções de dados e as tabelas do banco de dados: Ligações de rastreabilidade produzidas manualmente, indicando quais tabelas do banco de dados do sistema representam cada um dos ALI. É importante notar que um ALI representa um grupamento lógico de dados, podendo estar distribuído em diversas tabelas físicas do banco de dados do sistema. Por outro lado, foi assumido que o modelo físico foi construído de forma que uma tabela não represente mais de um elemento lógico dentre os grupamentos de dados manipulados pela aplicação.
- Funções transacionais: Tipo de elemento que representa as transações do sistema identificadas durante o levantamento das funções transacionais feito para a APF. Deve ser composto por uma lista das transações oferecidas pelo sistema e sua classificação como EE, SE e CE. Os TAR não precisam ser informados, pois a técnica não os utiliza durante seu processo de criação de ligações de rastreabilidade.
- Casos de teste: Tipo de elemento que representa os casos de teste do sistema, juntamente com suas ações, pré-condições e pós-condições. Devem existir casos de teste para avaliar todas as funções transacionais providas pelo sistema. Caso contrário, a técnica não será capaz de identificar as partes do código-fonte que suportam as funções transacionais que não estão cobertas pelos casos de teste.
- Rastreabilidade entre as funções transacionais e os casos de teste: Ligações de rastreabilidade produzidas manualmente, indicando quais casos de teste avaliam cada função transacional.
- Código-fonte: Tipo de elemento que representa a lógica interna do sistema como uma estrutura de diretórios e arquivos que contemplam suas classes, rotinas e *scripts*. O código-fonte precisa estar configurado em um ambiente em que o sistema possa ser executado, incluindo banco de dados e qualquer outro recurso utilizado pelo sistema.
- Granularidade de código desejada: Varia com a necessidade do projeto ou da organização. Determina o nível de granularidade em que o código-fonte será analisado. A decisão pelo grão do código-fonte que será analisado influenciará as ligações de rastreabilidade que serão criadas para ligar as tabelas do banco de dados ao código-fonte e os casos de teste ao código-fonte.

Alguns sistemas podem não ter a APF produzida ou esta esteja desatualizada. Ligações de rastreabilidade entre as funções de dados e as tabelas e entre as funções transacionais e os casos de teste dificilmente serão encontrados já existentes em uma situação real. A presença de um conjunto de casos de teste com ampla cobertura de código (suficiente para avaliar todas as funções transacionais) pode não ser uma realidade válida para todos os sistemas.

Deve-se esclarecer que a obrigatoriedade dessas informações de entrada fomenta a avaliação do esforço de produzir as funções de dados e transacionais tardiamente, estabelecer ligações de rastreabilidade manualmente e produzir casos de teste, em especial em sistemas de grande porte e alta complexidade, o que pode não ser uma tarefa trivial do ponto de vista de tempo, custo e recursos.

3.2.2. Etapa das Funções de Dados

A primeira etapa que compõe a técnica relaciona as funções de dados ao código-fonte do sistema. Seu objetivo é descobrir quais partes do código-fonte manipulam as funções de dados e estabelecer ligações de rastreabilidade entre elas. O nível de granularidade desejado para o código-fonte determina como este é dividido em partes, sendo que ligações de rastreabilidade serão criadas entre cada parte do código e os ALI manipulados por elas.

Geralmente não existem referências objetivas sobre quais funções de dados estão sendo manipuladas em determinada parte do código-fonte. Essas referências apenas estariam presentes caso fosse feito um trabalho manual de marcação das partes de código durante o desenvolvimento, o que poderia tornar a sua manutenção onerosa e suscetível a erros, conforme abordado no capítulo anterior.

ELMASRI e NAVATHE (2005) consideram que o modelo de dados relacional básico representa um banco de dados como uma coleção de tabelas. Como as funções de dados definem quais dados são necessários para que o sistema realize as operações desejadas pelo usuário, as tabelas do modelo relacional são criadas a partir de refinamentos cuja origem pode ser traçada até as funções de dados identificadas para ele. Por esse motivo, o conjunto de tabelas de um sistema pode ser entendido como a implementação das funções de dados, estando intimamente relacionados.

Para que o sistema manipule seus dados, o código-fonte deve referenciar as tabelas do sistema executando comandos de consulta e manipulação dos dados. Quando comandos *SQL* são utilizados no código-fonte com esse objetivo, ao invés de abstrações, como *web services*, *stored procedures*, *views* ou mapeamentos objeto-relacionais, as ocorrências desses comandos podem ser usadas como referências

objetivas aos dados que são manipulados por uma parte do código-fonte. Assim, as tabelas do banco de dados podem ser usadas como uma ponte entre as funções de dados e o código-fonte. Comandos em *data definition language (DDL)*, *storage definition language (SDL)* e *view definition language (VDL)* (ELMASRI e NAVATHE, 2005) devem ser desconsiderados, pois não manipulam os dados, mas a estrutura em que são armazenados. Apenas os comandos de recuperação, inserção, alteração e remoção de dados em *data manipulation language (DML)* são relevantes, e apenas os que atuem sobre tabelas, pois as *views* não são tratadas pela técnica proposta.

Entretanto, mesmo existindo similaridades entre as funções de dados e as tabelas de um sistema, não existem referências objetivas entre elas. Por conta disso, é necessário que o usuário crie manualmente ligações de rastreabilidade entre as funções de dados e as tabelas do banco de dados do sistema (Figura III-2). Enquanto um ALI deve ser rastreado para todas as tabelas criadas como parte de sua implementação, uma tabela deve ser rastreada para apenas um ALI. As tabelas que não representam entidades reconhecíveis pelo usuário são consideradas tabelas de implementação e não devem ser rastreadas para os ALI.

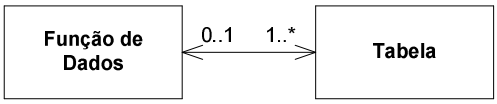


Figura III-2: Ligação entre as funções de dados e o esquema de banco

Com a criação manual dessas ligações, as funções de dados passam a estar relacionadas objetivamente às tabelas do banco de dados do sistema. Sabendo-se que existe um conjunto de *queries* inserido no código-fonte e que elas manipulam as tabelas do banco de dados, estas *queries* podem ser usadas para criar ligações de rastreabilidade entre as tabelas e as partes do código-fonte (Figura III-3).

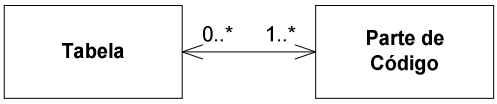


Figura III-3: Ligação entre o esquema de banco e o código-fonte

Todas as partes do código-fonte devem ser analisadas em busca das *queries*. Cada uma delas pode apresentar referências a zero ou mais tabelas. Para cada tabela referenciada em uma parte do código é criada uma ligação de rastreabilidade entre a tabela e a parte do código. A quantidade de referências a uma tabela em uma parte do código é registrada como o peso da ligação entre elas (Figura III-4a).

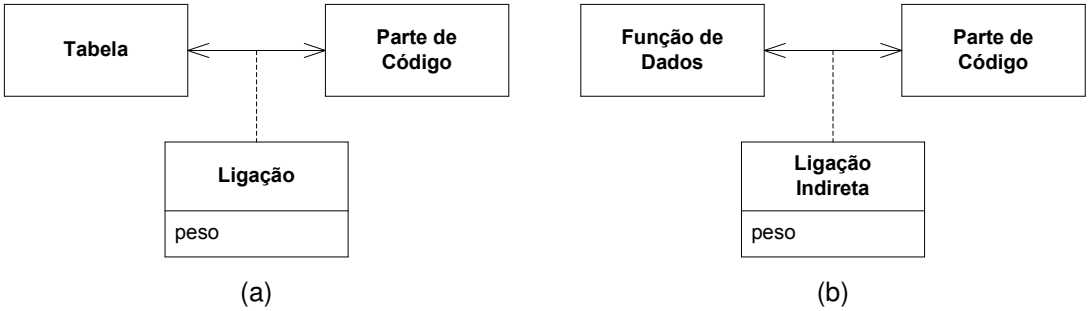


Figura III-4: Pesos das ligações das funções de dados

Exemplificando, em um caso onde o nível de granularidade desejado seja o de arquivo, existam cinco *queries* em um arquivo, um total de oito tabelas sendo referenciadas nestas *queries,* uma dessas tabelas sendo referenciada três vezes e as demais apenas uma vez (Figura III-5a), devem existir oito ligações de rastreabilidade (uma para cada tabela) e o arquivo de código-fonte. A ligação entre a tabela referenciada três vezes e o arquivo tem peso igual a três, enquanto as demais têm peso igual a um (Figura III-5b). Detalhes sobre como os pesos são utilizados pela técnica proposta serão abordados mais adiante neste capítulo.

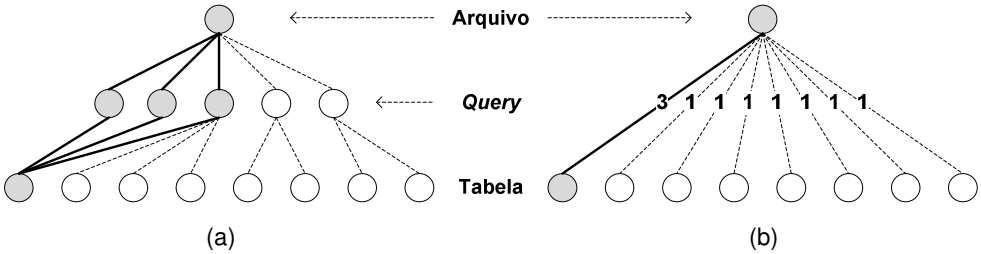


Figura III-5: Exemplo de cálculo do peso de ligações entre tabelas e partes de código

Pelo processo descrito nesta seção, as funções de dados e as tabelas são relacionadas por ligações de rastreabilidade, assim como o são as tabelas e o código-fonte. Aplicando-se a regra da transitividade de relações (GERSTING, 1995), as funções de dados e o código-fonte passam a ser ligados indiretamente por intermédio das tabelas do banco de dados do sistema (Figura III-6).

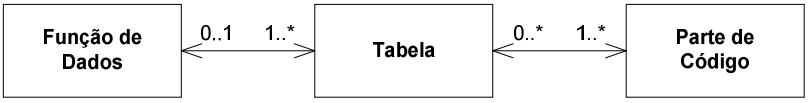


Figura III-6: Ligação indireta entre as funções de dados e o código-fonte

Essa ligação indireta também possui um peso (Figura III-4b), derivado das suas ligações diretas originais. O peso da ligação entre uma função de dados e uma parte de código é obtido pela soma dos pesos das ligações diretas entre todas as tabelas

relacionadas tanto pela função de dados quanto pela parte de código, de acordo com a fórmula a seguir.

$FD = \{ fd_i \mid i \in \mathbb{N}^* \}$
Conjunto das funções de dados (ALI)
$TB = \{ tb_j \mid j \in \mathbb{N}^* \}$
Conjunto das tabelas do banco de dados
$CF = \{ cf_k \mid k \in \mathbb{N}^* \}$
Conjunto das partes de código
$LIGFDTB = \{ (fd_i, tb_j) \mid i, j \in \mathbb{N}^* \wedge fd_i \in FD \wedge tb_j \in TB \}$
Conjunto das ligações entre funções de dados e tabelas
$LIGTBCF = \{ (tb_j, cf_k) \mid j, k \in \mathbb{N}^* \wedge tb_j \in TB \wedge cf_k \in CF \}$
Conjunto das ligações entre tabelas e partes de código
$LIGFDCF = LIGFDTB \circ LIGTBCF$
Conjunto das ligações transitivas entre funções de dados e partes de código, obtidas por meio de composição
$p_{jk} = peso\ da\ ligação\ (tb_j, cf_k) \in LIGTBCF$
$p_{jk} \in \mathbb{N}^*$
$w_{ik} = peso\ da\ ligação\ (fd_i, cf_k) \in LIGFDCF$
$w_{ik} \in \mathbb{N}^*$
$w_{ik} = \sum_{j \mid (fd_i, tb_j) \in LIGFDTB \wedge (tb_j, cf_k) \in LIGTBCF} p_{jk}$

Exemplificando, considere um caso onde determinada função de dados esteja ligada a duas tabelas e estas tabelas estejam ligadas a um total de cinco arquivos, sendo que um desses arquivos é referenciado por ambas as tabelas, com ligações que tenham pesos três e oito. Os demais arquivos estejam sendo referenciados por uma única tabela, sempre possuindo peso unitário em suas ligações (Figura III-7a). Assim, deve existir uma ligação de rastreabilidade entre a função de dados e o arquivo referenciado pelas duas tabelas, com peso igual a onze, e outras quatro ligações entre a função de dados e cada um dos demais arquivos referenciados por uma única tabela, com peso igual a um (Figura III-7b).

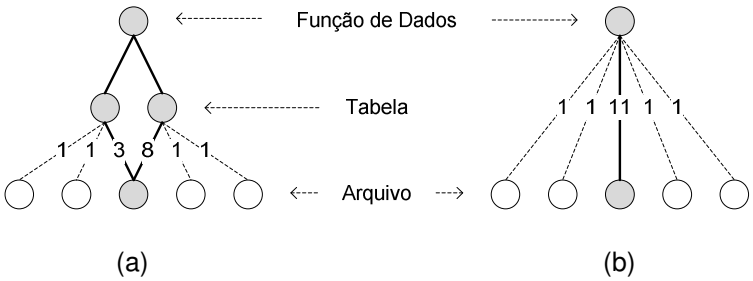


Figura III-7: Exemplo de cálculo do peso de ligações entre funções de dados e partes de código

3.2.3. Etapa das Funções Transacionais

A segunda etapa que compõe a técnica relaciona as funções transacionais ao código-fonte do sistema. Seu objetivo é descobrir quais partes do código-fonte implementam que funções transacionais e estabelecer ligações de rastreabilidade entre elas. O nível de granularidade desejado para o código-fonte determina como o código é dividido e ligações de rastreabilidade são criadas entre cada parte do código e as transações (EE, SE e CE) implementadas por elas.

Entretanto, geralmente não existem referências objetivas sobre quais funções transacionais são implementadas em uma determinada parte do código-fonte. Assim como ocorre com as funções de dados, essas referências estariam presentes apenas caso fosse realizado um trabalho manual de marcação das partes do código durante o seu desenvolvimento.

Um caso de teste é a especificação de um conjunto de entradas, condições de execução e resultados esperados para um objetivo particular, exercendo um caminho de execução específico pelo programa e verificando o cumprimento de um requisito (IEEE, 2008). Como as funções transacionais declaram quais operações um sistema oferece de acordo com uma visão inicial obtida junto aos usuários, os casos de teste são projetados para avaliar se as regras determinadas pelos refinamentos das funções transacionais estão sendo cumpridas. Por esse motivo, foi considerado que a execução de um caso de teste exerce um caminho de execução específico para realizar uma operação e que esta operação pode ser rastreada até uma função transacional identificada na APF.

Durante a execução de um caso de teste, dependendo das entradas e condições de execução, uma determinada seqüência ordenada de linhas de código é ativada. Identificando-se as linhas de código que são ativadas pelos casos de teste que verificam uma determinada operação oferecida pelo sistema, chega-se às ligações entre essas operações e as partes do código-fonte. Assim, os casos de teste podem ser usados como uma ponte entre as funções transacionais e o código-fonte. Essa estratégia se baseia na abordagem de rastreabilidade baseada em cenários (EGYED, 2001), onde a informação das linhas de código que estão sendo acionadas durante a execução de cenários de utilização do sistema (por exemplo, casos de teste) é usada na criação de ligações de rastreabilidade.

Entretanto, mesmo existindo essa relação entre as funções transacionais e os casos de teste, não existem referências objetivas entre eles. Por conta disso, é necessário que sejam criadas manualmente ligações de rastreabilidade entre as funções transacionais e os casos de teste (Figura III-8). Enquanto uma transação EE,

SE ou CE deve ser rastreada para todos os seus casos de teste, um caso de teste deve ser rastreado para uma única EE, SE ou CE, sendo que os casos de teste que não manipulam dados reconhecidos pelo usuário (casos de teste desenvolvidos para avaliar um algoritmo interno do sistema, que não seja diretamente identificado com uma operação específica) não devem ser rastreados.

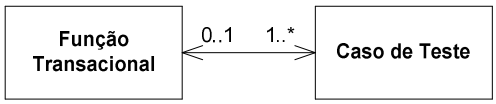


Figura III-8: Ligação entre as funções transacionais e os casos de teste

Com a criação manual dessas ligações, as funções transacionais do sistema passam a estar relacionadas objetivamente aos seus casos de teste. Sabendo-se que os casos de teste provocam a ativação de uma seqüência de linhas no código-fonte, o registro destas ativações pode ser usado para criar ligações de rastreabilidade entre os casos de teste e o código-fonte (Figura III-9). Uma ferramenta de cobertura de código, capaz de monitorar a execução do *software* enquanto testes estão sendo executados, identifica as partes do código que são acionadas e registra estas ativações em um *log*. Mais tarde, esta informação pode ser utilizada para relacionar os casos de teste com o código-fonte.

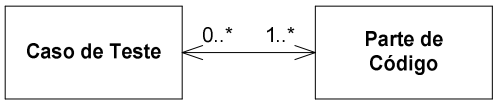


Figura III-9: Ligação entre os casos de teste e o código-fonte

De acordo com a capacidade da ferramenta de cobertura de código, é possível monitorar a execução do código-fonte em diferentes níveis de granularidade, como arquivos, classes, métodos ou linhas de código individuais. Ferramentas deste tipo geralmente podem ser usadas de duas formas distintas: intrusiva, onde o código-fonte é instrumentado, recebendo um conjunto de linhas que acionam a ferramenta; e não-intrusiva, onde basta que seja feita uma configuração no compilador, interpretador ou no ambiente de execução, sem a necessidade de alteração do código propriamente dito. A segunda forma de uso é mais recomendada para a técnica proposta, pois não exige que a instrumentação do código seja feita toda vez que o código-fonte for alterado devido a requisições de mudança, otimização de algoritmos e outros motivos.

Como a técnica proposta exige que o sistema seja submetido a testes, o código-fonte deve estar em um ambiente em que possa ser executado, com todos os seus recursos configurados (por exemplo, variáveis de ambiente e banco de dados).

Este é um pré-requisito das ferramentas de cobertura de código, pois somente assim elas podem monitorar a ativação das partes de código enquanto o sistema é testado.

Todos os casos de teste devem ser executados. Os registros de *log* gerados pela ferramenta durante a execução de um caso de teste devem ser relacionados a esse caso de teste. Considerando o nível de granularidade desejado, para cada parte de código-fonte obtida do *log* e o caso de teste que a acionou é criada uma ligação de rastreabilidade (Figura III-10a). Considerando o nível de granularidade em que a ferramenta é capaz de monitorar o código-fonte, que deve ser igual ou inferior ao nível desejado para as ligações, a quantidade de partes de código distintas identificadas pela ferramenta como tendo sido ativadas durante a execução do caso de teste é registrada como o peso da ligação entre o caso de teste e a parte de código.

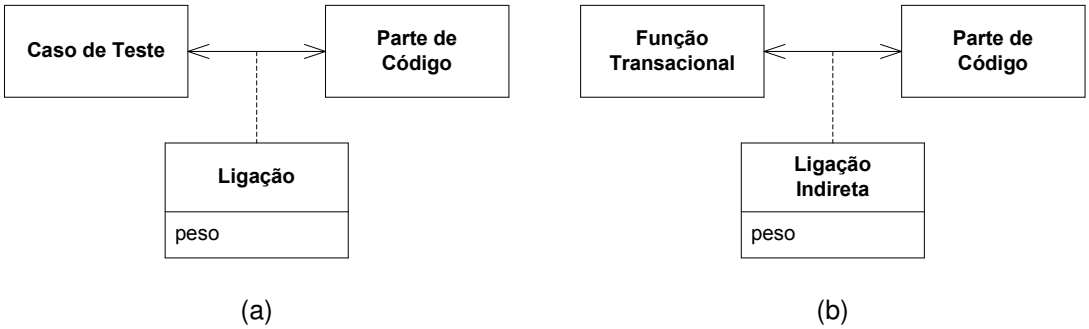


Figura III-10: Pesos das ligações das funções transacionais

Exemplificando, em um caso onde o nível de granularidade desejado seja o de arquivo, o nível de granularidade identificável pela ferramenta de cobertura de código seja o de linha de código e um caso de teste tenha acionado 100 linhas distintas de um arquivo (Figura III-11a), deve existir uma ligação de rastreabilidade entre o caso de teste e o arquivo com peso igual a 100 (Figura III-11b). Detalhes sobre como os pesos são utilizados pela técnica proposta serão abordados mais adiante neste capítulo.

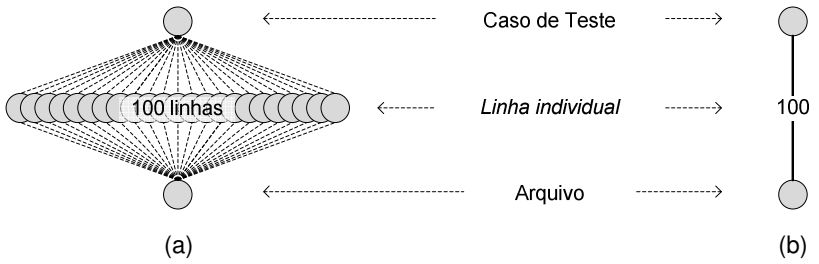


Figura III-11: Exemplo de cálculo do peso de ligações entre casos de teste e partes de código

Pelo processo descrito nesta seção, as funções transacionais e os casos de teste são relacionados por ligações de rastreabilidade, assim como o são os casos de teste e o código-fonte. Aplicando-se a regra da transitividade de relações (GERSTING,

1995), as funções transacionais e o código-fonte passam a ser ligados indiretamente por intermédio dos casos de teste (Figura III-12).

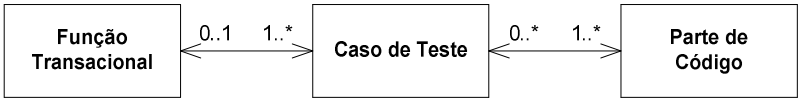


Figura III-12: Ligação indireta entre as funções transacionais e o código-fonte

Essa ligação indireta também possui um peso (Figura III-10b), derivado das suas ligações diretas originais. O peso da ligação entre uma função transacional e uma parte de código é obtido pela soma dos pesos das ligações diretas entre todos os casos de teste relacionados tanto pela função transacional quanto pela parte de código, de acordo com a fórmula abaixo.

$FT = \{ ft_i \mid i \in \mathbb{N}^* \}$
Conjunto das funções transacionais (EE, SE e CE)
$CT = \{ ct_j \mid j \in \mathbb{N}^* \}$
Conjunto dos casos de teste
$CF = \{ cf_k \mid k \in \mathbb{N}^* \}$
Conjunto das partes de código
$LIGFTCT = \{ (ft_i, ct_j) \mid i,j \in \mathbb{N}^* \wedge ft_i \in FT \wedge ct_j \in CT \}$
Conjunto das ligações entre funções transacionais e casos de teste
$LIGCTCF = \{ (ct_j, cf_k) \mid j,k \in \mathbb{N}^* \wedge ct_j \in CT \wedge cf_k \in CF \}$
Conjunto das ligações entre casos de teste e partes de código
$LIGFTCF = LIGFTCT \circ LIGCTCF$
Conjunto das ligações transitivas entre funções transacionais e partes de código, obtidas por meio de composição
$p_{jk} = peso\ da\ ligação\ (ct_j, cf_k) \in LIGCTCF$
$p_{jk} \in \mathbb{N}^*$
$w_{ik} = peso\ da\ ligação\ (ft_i, cf_k) \in LIGFTCF$
$w_{ik} \in \mathbb{N}^*$
$w_{ik} = \sum_{j \mid (ft_i, ct_j) \in LIGFTCT \wedge (ct_j, cf_k) \in LIGCTCF} p_{jk}$

Exemplificando, considere um caso onde determinada função transacional esteja ligada a oito casos de teste e estes casos de teste estejam ligados a um total de cinco arquivos, sendo que um desses arquivos é referenciado por dois casos de teste, com ligações de pesos 100 e 150 (Figura III-13a). Desconsideremos as demais ligações para os fins desse exemplo. Assim, deve existir uma ligação entre a função transacional e o arquivo referenciado pelos dois casos de teste, com peso igual a 250 (Figura III-13b). Já as ligações desconsideradas darão origem a mais quatro ligações entre funções transacionais e arquivos de código-fonte. Seus detalhes foram ocultados apenas para facilitar o entendimento do exemplo.

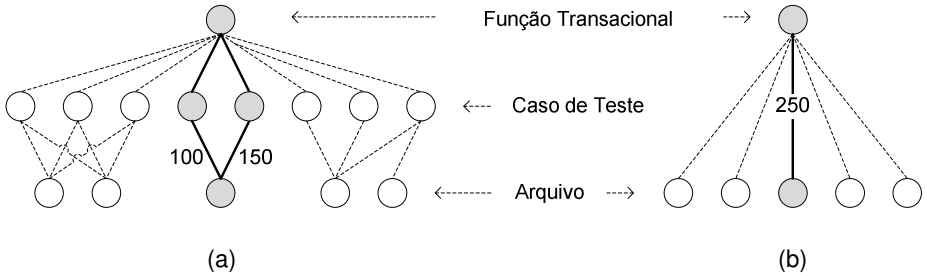


Figura III-13: Exemplo de cálculo do peso de ligações entre funções transacionais e partes de código

3.2.4. Pesos das ligações

A utilização de pesos para indicar que uma ligação de rastreabilidade é mais forte que outra não é comum na literatura. Dentre os trabalhos pesquisados, as ligações de rastreabilidade costumam indicar apenas que dois elementos estão relacionados, sem uma visão ponderada da relação entre eles.

Alguns trabalhos (ANTONIOL *et al.*, 2000, ANTONIOL *et al.*, 2002, GRECHANIK *et al.*, 2007) calculam a probabilidade de dois elementos serem relacionados e utilizam esta probabilidade para decidir sobre a validade de uma ligação. Entretanto, estes trabalhos não permitem comparar duas ligações e indicar que o elemento rastreado por uma ligação é mais relevante do que o elemento rastreado pela outra sob uma determinada perspectiva (na técnica ora proposta, a participação na implementação de uma função transacional ou de dados).

A técnica proposta oferece pesos em quatro tipos de ligação: (a) diretas entre as tabelas e o código-fonte; (b) diretas entre os casos de teste e o código-fonte; (c) indiretas entre as funções de dados e o código-fonte; e (d) indiretas entre as funções transacionais e o código-fonte. Analisando os pesos das ligações de um elemento qualquer se chega à conclusão de quais elementos rastreados a partir deste são potencialmente mais relevantes que os demais.

Exemplificando o uso dos pesos na análise de imapacto, considere um sistema onde existam 30 funções de dados e 100 funções transacionais. Dentre as funções de dados, uma em especial está ligada indiretamente a cinco partes de código, cujos pesos em ordem decrescente são {16, 13, 13, 5, 2}. Se um programador precisa identificar os pontos de alteração para modificar um comportamento relacionado a essa função de dados, as partes de código que possuem maior peso (16 e 13, por exemplo) tendem a ser os mais prováveis pontos de alteração.

O peso da ligação entre uma função de dados e uma parte de código é derivado da quantidade de ocorrências de tabelas no código-fonte. Já o peso da ligação entre uma função transacional e uma parte de código é derivado da

quantidade de partes de código acionadas pelos casos de teste que avaliam a função transacional.

Pelas diferenças inerentes aos processos com que são medidos, os pesos dos dois tipos de ligação não são comparáveis e não existem garantias de que a ordem de grandeza dos dois pesos seja similar. Por exemplo, se a ferramenta de cobertura de código gera o *log* de execução no nível das linhas de código individuais, a quantidade de linhas acionadas em um arquivo de código-fonte tende a ser muito superior à quantidade de *queries* e de tabelas nelas referenciadas. Para que os dois tipos de pesos sejam comparáveis, é necessário que sejam convertidos para uma ordem de grandeza comum, como o intervalo (0,1], ou seja, variando de um valor próximo a zero até o valor um. O peso zero é excluído, pois representaria uma ligação inexistente.

Os pesos convertidos foram denominados como índices de relevância da ligação. Para uma ligação entre uma função de dados ou transacional e uma parte de código, o valor desse índice é resultado da razão entre o peso dessa ligação e a soma dos pesos de todas as ligações dessa função de dados ou transacional com qualquer parte de código, o que pode ser observado nas fórmulas apresentadas logo adiante. Em outras palavras, o índice é um percentual que determina quanto a parte de código em questão é responsável pela implementação daquela função de dados ou transacional, frente à totalidade de partes de código que participam da implementação daquela função de dados ou transacional.

Exemplificando, para uma função de dados que esteja ligada a três partes de código *{A, B, C}* com pesos *{100, 70, 50}* respectivamente, o índice de relevância da ligação com *A* é resultado do cálculo $IR_A = 100 / (100 + 70 + 50) = 0{,}45$, o que sugere que 45% da lógica da função de dados está implementada na parte de código *A*.

$v_{ik} = peso\ da\ ligação\ (fd_i, cf_k) \in LIGFDCF$
$v_{ik} \in \mathbb{N}^*$
$w_{jk} = peso\ da\ ligação\ (ft_j, cf_k) \in LIGFTCF$
$w_{jk} \in \mathbb{N}^*$

$a_{ik} = índice\ de\ relevância\ da\ ligação\ (fd_i, cf_k) \in LIGFDCF$
$a_{ik} \in \mathbb{Q}^* \cap (0,1]$
$$a_{ik} = \frac{v_{ik}}{\sum_{x|x\in\mathbb{N}^*\wedge(fd_i,cf_x)\in LIGFDCF} v_{ix}}$$

$b_{jk} = índice\ de\ relevância\ da\ ligação\ (ft_j, cf_k) \in LIGFTCF$
$b_{jk} \in \mathbb{Q}^* \cap (0,1]$
$$b_{jk} = \frac{v_{jk}}{\sum_{y|y\in\mathbb{N}^*\wedge(ft_j,cf_y)\in LIGFTCF} w_{jy}}$$

3.3. Considerações finais

Este capítulo apresentou a técnica proposta para criação de ligações de rastreabilidade entre ponto de função e o código-fonte de um Sistema de Informação. O processo de aplicação da técnica proposta é semi-automático, pois exige que algumas informações de entrada sejam criadas manualmente, enquanto utiliza processos automáticos para realizar tarefas onerosas, repetitivas e suscetíveis a erros.

Entretanto, o esforço necessário à produção dessas informações de entrada deve ser comparado com os objetivos desejados na utilização da técnica proposta. Trabalhos futuros podem avaliar soluções para facilitar a criação ou a engenharia reversa das listas de funções de dados e transacionais, da criação das ligações de rastreabilidade de entrada e de casos de teste adequados.

As ligações de rastreabilidade criadas permitem navegação bidirecional entre o código-fonte e as funções de dados ou transacionais, propiciando o entendimento de quais partes do código-fonte manipulam uma função de dados ou processam uma função transacional, assim como quais funções de dados são manipuladas por uma parte de código-fonte e quais funções transacionais têm parte da sua lógica implementada em uma parte de código-fonte. A proposta traz ainda pesos nas ligações, permitindo que a relevância de um conjunto de ligações seja comparada.

O próximo capítulo apresentará a forma como foram conduzidos três estudos de caso onde esta técnica foi aplicada a sistemas reais e um protótipo que demonstra como as ligações criadas podem ser utilizadas por uma equipe de desenvolvimento de *software*.

IV. Avaliação e Implementação da Proposta

Nos capítulos anteriores foi apresentada uma revisão sobre a área da Rastreabilidade de Requisitos, uma introdução sobre a Análise de Ponto de Função e uma proposta de técnica para criação semi-automática de ligações de rastreabilidade entre funções de dados e transacionais e o código-fonte de um Sistema de Informação. Este capítulo apresenta a metodologia aplicada para avaliação da técnica, os resultados obtidos e algumas considerações sobre os mesmos.

Foram realizados dois estudos de caso que auxiliaram no desenvolvimento da técnica proposta e demonstraram sua aplicação em sistemas reais. O primeiro avaliou a criação de ligações para as funções de dados do Compiere[18], um *Enterprise Resource Planning* (ERP) e *Customer Relationship Management* (CRM) *open source* escrito em Java. O segundo avaliou não apenas a criação de ligações para as funções de dados, mas também para as funções transacionais do módulo de administração de usuários do Moodle[19], um *Learning Management System* (LMS) *open source* escrito em PHP. As funções transacionais do Moodle não foram avaliadas por completo, pois devido ao porte do sistema, identificar todas as suas funções transacionais e produzir casos de teste consumiria mais tempo do que o disponível para a realização desta pesquisa. Esse esforço se caracteriza como uma dificuldade no cenário de uso da técnica, não a uma ameaça interna da própria, pois a produção e execução de casos de teste fazem parte do processo de desenvolvimento tradicional, não sendo uma atividade exclusivamente realizada para aplicação da técnica proposta. Situações onde esses casos de teste não estejam adequados à aplicação da técnica devem sofrer uma maior avaliação quanto ao seu custo/benefício.

Um terceiro estudo de caso foi realizado para demonstrar a viabilidade de aplicação da completa da técnica em um SI – neste caso, o Sistema de Inscrições Online do PPGI[20]. Ele é um sistema de menor porte (se comparado ao Compiere e Moodle) escrito em PHP, que teve seu código-fonte disponibilizado para a pesquisa.

[18] http://www.compiere.com
[19] http://moodle.org/
[20] http://www.uniriotec.br/~inscricaoppgi

Enquanto os estudos de caso demonstram a viabilidade da criação de ligações de rastreabilidade, o uso dessas ligações é apresentado em um protótipo, cujo objetivo é mostrar como os programadores e arquitetos poderiam ter acesso a informações de rastreabilidade diretamente em suas ferramentas de trabalho, conforme é recomendado por CLELAND-HUANG *et al. (*2007). O protótipo foi desenvolvido como um *plugin* para o ambiente de desenvolvimento (IDE) Eclipse, facilitando a visualização das ligações de rastreabilidade entre as funções de dados ou transacionais e o código-fonte.

A escolha por sistemas *open source* nos estudos de caso deveu-se ao livre acesso ao código-fonte destes sistemas e à possibilidade de reprodução dos resultados da pesquisa por outros pesquisadores. Entretanto, a avaliação de sistemas que não sejam *open source* deve ser objeto de estudo de trabalhos futuros, pois a indústria produz *software* de forma significativamente diferente daquela utilizada pela comunidade *open source. Trabalhos futuros também devem avaliar* sistemas de diversos portes, linguagens, arquiteturas e *frameworks,* para generalizar a aplicabilidade da técnica proposta e motivar seu amadurecimento, aspectos que não estão incluídos no escopo do presente trabalho.

Este capítulo apresenta o trabalho desenvolvido e os dados obtidos na pesquisa. Ele é dividido da seguinte forma: a Seção 4.1 e suas subseções apresentam os três Estudos de caso. A Seção 4.2 apresenta o protótipo de ferramenta de rastreabilidade. Ao final, a Seção 4.3 apresenta algumas considerações finais sobre os dados obtidos.

4.1. Estudos de Caso

Esta seção apresenta os três estudos de caso realizados durante a pesquisa: o primeiro sobre as funções de dados do Compiere; o segundo sobre as funções de dados do Moodle e as funções transacionais do módulo de administração de usuários; e o terceiro sobre as funções de dados e transacionais do Sistema de Inscrições Online do PPGI.

4.1.1. Compiere

A condução deste estudo de caso tem o objetivo de apresentar a aplicação de uma primeira versão da técnica proposta na criação de ligações de rastreabilidade entre as funções de dados e o código-fonte de um Sistema de Informação. O sistema selecionado para o estudo de caso foi o Compiere, um ERP e CRM que realiza a

integração de dados e processos de diversas áreas de uma organização, como finanças, aquisições, materiais, vendas e outros. O Compiere está disponível sob a licença *open source GNU General Public License* (GPL)[21]. Assim, ele pode ser utilizado e distribuído gratuitamente e seu código-fonte está disponível para acesso e modificação por terceiros.

4.1.1.1. Execução

Foi desenvolvida uma técnica para identificar em quais arquivos de código-fonte as funções de dados de um sistema são manipuladas. Para isso, foram usadas as ocorrências de *queries* de manipulação de tabelas no código-fonte, de maneira similar à apresentada na Seção 3.2.2.

Para a coleta das informações de entrada, o primeiro passo do estudo de caso foi obter o código-fonte do Compiere, acessando seu repositório de controle de versão no dia 7 de novembro de 2008. Um total de 5.015 arquivos foi recebido, sendo 2.277 arquivos de código-fonte escritos em Java (01_EC_Compiere_Arquivos.xlsx, no DVD anexo). Os demais arquivos são relacionados a textos TXT, interface *web* com HTML, CSS, JS e JSP, configurações em XML, bibliotecas JAR, *scripts* SH e BAT, imagens GIF e PNG, áudio WAV, *queries* SQL, e planilhas XLS.

O passo seguinte foi obter uma lista das funções de dados do sistema. Em muitas situações reais, essa informação já estaria disponível, pois as funções de dados teriam sido listadas como forma de orçar o esforço necessário para o desenvolvimento do sistema. Entretanto, como a pesquisa não possuía essa lista de funções de dados, ela foi construída por um processo de engenharia reversa semi-automático, analisando as tabelas que compunham o banco de dados do sistema.

Esse passo foi decomposto em um conjunto de atividades. A primeira delas foi obter uma lista das tabelas pertencentes ao banco de dados do sistema. Para isto, foi analisado um *script* Postgres[22] que continha os comandos SQL para criação de 465 tabelas (02_EC_Compiere_Tabelas.xlsx, no DVD anexo) utilizadas pelo sistema. Foi detectado um padrão de nomenclatura ao qual quase todas as tabelas do Compiere obedecem. Seus nomes iniciam com uma letra seguida de um *underscore* e um conjunto de nomes intercalados por *underscores* (por exemplo, *M_Product_Category*). Analisando os nomes das tabelas, a pesquisa detectou que aquelas iniciadas pelas letras "*C*" ou "*M*" pareciam se referir a entidades do domínio de ERP e CRM. As 211 tabelas com esses prefixos foram selecionadas e foram descartadas as demais (03_EC_Compiere_TabelasSelecionadas.xlsx, no DVD anexo).

[21] http://www.gnu.org/licenses/gpl.html
[22] http://www.postgresql.org/

Analisando as 211 tabelas selecionadas, percebeu-se que existiam aquelas cujos nomes diferiam apenas pela adição de um sufixo composto por um segundo *underscore* seguido por um nome (por exemplo, *M_Product, M_Product_Acct e M_Product_Category*). Um *script* foi desenvolvido para analisar as tabelas de acordo com esse padrão, compondo grupos de tabelas com nome similar. Cada tabela poderia pertencer a apenas um grupo e cada tabela cujo nome não fosse similar ao de nenhuma outra deveria ser isolada em um grupo próprio. Aplicadas estas heurísticas, foram encontrados 162 grupos de tabelas (04_EC_Compiere_FuncoesDeDados.xlsx, no DVD anexo). Considerando cada grupo de tabelas como a implementação de uma função de dados, foi convencionado que a tabela de nome mais curto no grupo deveria nomear a função de dados (*M_Product*, no exemplo anterior).

O passo seguinte foi estabelecer as ligações entre as funções de dados e as tabelas do seu grupo, compondo 211 ligações (05_EC_Compiere_Lig_FD_Tabelas.xlsx, no DVD anexo). Cabe ressaltar que as 254 tabelas que não possuem o prefixo *"C"* ou *"M"* não foram rastreadas para funções de dados, por exemplo, as de prefixo *"AD"*, que se referem a *logs* de acesso, impressão, agendamento, dentre outros.

Foram necessárias cerca de 40 horas para compor as informações de entrada da técnica. Entrando na aplicação da técnica propriamente dita, foram estabelecidas as ligações entre as tabelas e o código-fonte do sistema. A princípio, essas ligações seriam criadas analisando manualmente o código-fonte arquivo por arquivo. Entretanto, como seria necessário demasiado esforço para realizar essa análise, foi identificado o objetivo de cada módulo de código-fonte e eliminados os que foram considerados não serem responsáveis por manipular o banco de dados, por exemplo, os diretórios referentes aos módulos de autenticação LDAP e marcação Jakarta ECS. Apenas os 1.667 arquivos de código-fonte Java dos diretórios *"ad"* e *"base"* foram mantidos (06_EC_Compiere_ArquivosSelecionados.xlsx, no DVD anexo).

Foi produzido um *script* para localizar as *queries* (e as tabelas por elas referenciadas) presentes nos arquivos de código-fonte dos diretórios selecionados para análise. Foram encontradas 33.523 referências a tabelas em 700 arquivos de código-fonte. Eliminando referências duplicadas entre uma dada tabela e um dado arquivo, foram formadas 2.608 ligações entre apenas 83 tabelas e os 700 arquivos (07_EC_Compiere_Lig_Tabelas_Arquivos.xlsx, no DVD anexo). Cada uma destas ligações associa um arquivo a uma tabela e recebe um peso que indica a quantidade de referências no arquivo para a tabela. Cabe ressaltar que 128 tabelas não foram rastreadas para os 1.667 arquivos selecionados. Uma análise, posterior à execução do estudo de caso, demonstrou que existiam referências a algumas tabelas nos módulos

de código-fonte descartados. Isso vem a corroborar com a visão de que o descarte de módulos de código-fonte pode gerar problemas nos resultados da técnica proposta.

Finalmente, o último passo foi criar as ligações entre as funções de dados e os arquivos de código-fonte. À época da realização deste estudo de caso, foi considerado que todo arquivo que referenciasse o banco de dados do sistema deveria estar relacionado a uma única função de dados. Quando tabelas de mais de uma função de dados eram encontradas em um arquivo, a função de dados com maior quantidade de referências era selecionada para a ele ser relacionada.

Um *script* foi produzido para estabelecer ligações entre um arquivo e a função de dados mais referenciada por ele. A quantidade de referências nesse contexto é a soma dos pesos das ligações entre todas as tabelas rastreáveis pela função de dados e o arquivo. Quando mais de uma função de dados possuía a maior quantidade de referências, uma delas era escolhida manualmente, analisando-se o propósito do arquivo com base no seu nome e conteúdo. Foram obtidas 700 ligações de rastreabilidade entre 76 funções de dados e os 700 arquivos de código-fonte (08_EC_Compiere_Lig_FD_Arquivos.xlsx, no DVD anexo). Cabe ressaltar que 86 funções de dados não foram rastreadas, o que é preocupante quanto à precisão da técnica. Por isso, com a evolução da técnica não é mais efetuado o descarte de arquivos e tabelas, o que ocorreu no estudo de caso sobre o Moodle (Seção 4.1.2). Essas funções de dados não rastreadas se devem à presença de referências a tabelas nos módulos do código-fonte descartados e à ligação de apenas uma função de dados a um arquivo.

4.1.1.2. Análise

Consolidando os dados obtidos no estudo de caso sobre o Compiere, chegou-se aos conjuntos e relações expostos na Figura IV-1.

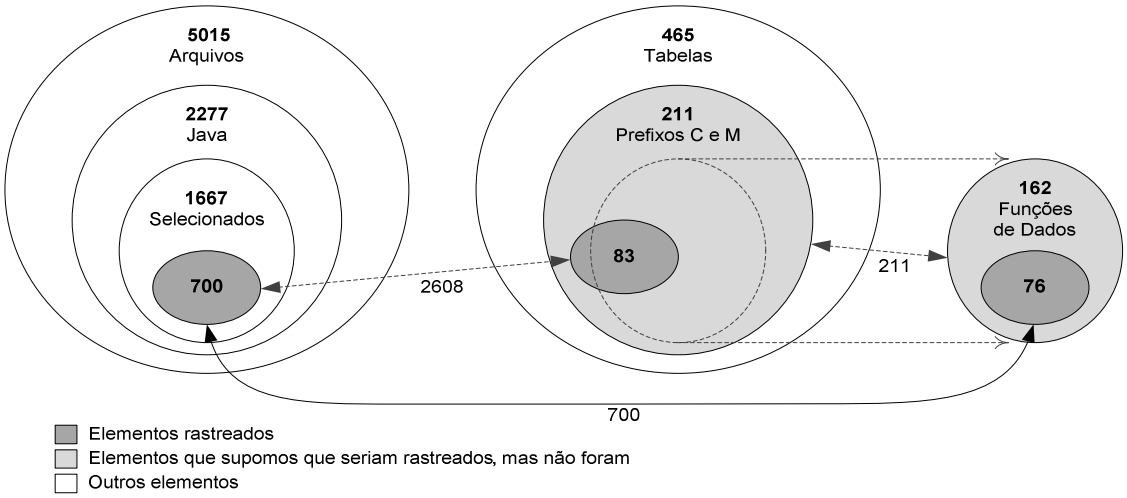


Figura IV-1: Dados obtidos no estudo de caso sobre o Compiere

Os dados obtidos não se apresentavam conforme o esperado. Analisando os quantitativos dos elementos e ligações, foi identificado que algumas informações potencialmente valiosas foram perdidas. Dentre essas perdas, destacam-se 128 tabelas (61%) com prefixo *"C"* ou *"M"* e 86 funções de dados (53%) não referenciadas pelo código-fonte. Uma busca realizada posteriormente à condução do estudo de caso demonstrou que existiam referências a tabelas com prefixo *"C"* e *"M"* nos módulos de código-fonte descartados. Essa situação nos levou a modificações na primeira versão da técnica, que seriam verificadas no estudo de caso seguinte, sobre o Moodle.

Foi concluído que não deve ser feito nenhum descarte prévio de elementos (sejam eles de funções de dados, tabelas ou código-fonte) que possam participar de ligações de rastreabilidade. Quando apenas alguns arquivos de código-fonte e tabelas foram selecionados para a análise, qualquer mudança de entendimento (por exemplo, perceber que determinado arquivo ou tabela foi descartada indevidamente) obrigaria uma nova execução dos *scripts* de busca e análise citados na seção anterior. Aliando à busca automatizada pelas tabelas no código-fonte ter reduzido o esforço necessário para relacionar as funções de dados aos arquivos, todos os elementos passaram a ser mantidos, ainda que não os rastreando.

Usar apenas o nome das tabelas para identificar e rastrear as funções de dados, sem considerar outros aspectos semânticos e contextuais sobre essas tabelas, é uma abordagem questionável. Tabelas de nomes completamente diferentes entre si poderiam compor perfeitamente uma única função de dados, pois o nome de cada tabela é atribuído conforme o desejo de seus desenvolvedores, que nem sempre terão a intenção de demonstrar os grupamentos de dados em seus nomes. Por exemplo, uma função de dados *"Localidade"* pode ser mapeada nas tabelas *"C_Location"*, *"C_City"* e *"C_Country"*, que possuem nomes diferentes. Esse mapeamento passaria a ser um dado de entrada para a técnica, possível de ser produzido mesmo que manualmente.

Rastrear um arquivo para apenas uma função de dados provoca a perda de informação das outras funções de dados que porventura sejam manipuladas pelo arquivo. Usar a quantidade de referências a tabelas como único critério para identificar essa função de dados pode gerar inconsistências, pois muitas junções relacionais com uma mesma tabela fariam com que a função de dados a ela relacionada sobressaísse. A técnica passou então a usar a quantidade de referências para indicar a relevância de uma função de dados em um arquivo, sem descartar os arquivos que os pesos indiquem ter menor relevância.

Mais da metade (53%) das funções de dados não foi rastreada, devido à perda de informações provocada pelo descarte de arquivos e tabelas. A existência dessas

funções de dados não manipuladas no código-fonte indicaria funções de dados e tabelas sem utilidade ou a presença de uma camada de abstração (uma das limitações da proposta).

Ligações entre as funções de dados e os arquivos de código-fonte não eram estabelecidas automaticamente após a criação dos dois tipos de ligações diretas. Havia ainda a necessidade de identificar as funções de dados mais relevantes, com eventual intervenção manual. Essas ligações indicam apenas que dois elementos estão relacionados, sem uma visão ponderada, pois a quantidade de referências era suprimida. O próximo estudo de caso mudaria ambas as abordagens.

4.1.2. Moodle

A condução deste estudo de caso tem o objetivo de apresentar a aplicação da técnica proposta na criação de ligações de rastreabilidade entre as funções de dados e o código-fonte, bem como a criação de ligações entre as funções transacionais e o código-fonte de um Sistema de Informação.

Foi desenvolvida neste estudo de caso a etapa das funções transacionais, que identifica em quais arquivos de código-fonte as funções transacionais de um sistema são implementadas, fazendo uso das linhas de código acionadas pela execução de casos de teste, conforme foi apresentado na Seção 3.2.3.

O sistema selecionado para o estudo de caso é um LMS, que gerencia cursos, currículos, turmas e conteúdo de ensino. Assim como o Compiere, o Moodle está disponível sob a licença *open source GNU General Public License* (GPL), estando seu código-fonte disponível para acesso e modificação por terceiros.

A aplicação da técnica a mais um sistema, que inclusive utilize uma linguagem de programação diferente do sistema utilizado no primeiro estudo de caso constitui um passo em direção à generalização da técnica proposta.

4.1.2.1. Execução da Etapa das Funções de Dados

Para a coleta das informações de entrada, foi definido o nível de granularidade das ligações como o de arquivo. O primeiro passo da etapa de criação das ligações entre as funções de dados e o código-fonte do Moodle foi obter seu código-fonte acessando seu repositório de controle de versão no dia 10 de junho de 2009. Um total de 30.960 arquivos foi recebido, sendo 10.170 arquivos de código-fonte escritos em PHP (01_EC_Moodle_ ArquivosFD.xlsx, no DVD anexo).

O passo seguinte foi obter uma lista das funções de dados do sistema. Esse passo foi decomposto em um conjunto de atividades. A primeira delas foi baixar e

instalar o *build* do Moodle para permitir a segunda atividade, a identificação das tabelas pela administração do MySQL. Foram identificadas 198 tabelas (02_EC_Moodle_Tabelas.xlsx, no DVD anexo) utilizadas pelo sistema. A terceira atividade foi realizar um agrupamento inicial de tabelas que pareciam representar uma mesma função de dados. A quarta atividade foi realizar testes *ad hoc* para confirmar se essa função de dados proposta era coerente no domínio do sistema e também encontrar as tabelas que tiveram dados alterados pelo teste realizado. Isso foi baseado na abordagem de testes utilizada por TAN *et al.* (2009), que extrai o modelo conceitual de um sistema como um diagrama de classes (pela similaridade existente entre um diagrama de classes de domínio e um esquema de banco de dados e, indiretamente, com das funções de dados). Com a execução dessas atividades, 28 funções de dados foram identificadas (03_EC_Moodle_FuncoesDeDados.xlsx, no DVD anexo).

É importante ressaltar que o código-fonte obtido do repositório de controle de versão não foi instalado, pois existiam diversas versões de módulos no diretório *"contrib"* e estas não estavam inseridas no diretório principal chamado *"moodle"*. Como os códigos do sistema passam por um processo de *build*, suas funcionalidades não estavam plenamente disponíveis ao fazer uma instalação a partir da estrutura de diretórios obtida do repositório.

O passo seguinte foi estabelecer as ligações entre as funções de dados e as tabelas com base nos testes realizados, compondo 108 ligações (04_EC_Moodle_Lig_FD_Tabelas.xlsx, no DVD anexo). Cabe ressaltar que 90 tabelas não foram rastreadas, pois não foi identificada relação com nenhuma função de dados, sendo responsáveis por *logs* de *queries* SQL, *backups*, configurações, *cache* de conteúdo, um mecanismo interno de controle de eventos, dentre outras.

Foram necessárias cerca de 60 horas para compor as informações de entrada da etapa das funções de dados. Entrando na aplicação da técnica propriamente dita, foram estabelecidas as ligações entre as tabelas e o código-fonte do sistema. A busca por referências a tabelas foi feita no código-fonte obtido do repositório de controle de versão. O Moodle foi projetado para isolar a manipulação de dados em um único arquivo, que provê um conjunto de métodos genéricos de acesso a tabelas, enquanto os demais arquivos de código-fonte utilizam esses métodos. Foi produzido um *script* para localizar as referências a tabelas nos arquivos de código-fonte. Dessa vez não foram descartados arquivos para reduzir o escopo de busca. Foram encontradas 77.981 referências a tabelas em 1.403 arquivos de código-fonte. Eliminando referências duplicadas entre uma dada tabela e um dado arquivo, foram formadas 4.024 ligações entre as 192 tabelas e os 1.403 arquivos

(05_EC_Moodle_Lig_Tabelas_Arquivos.xlsx, no DVD anexo). Essas ligações receberam um peso, que indica a quantidade de referências a uma tabela realizadas em um arquivo.

Cabe ressaltar que dentre as seis tabelas que não foram rastreadas para o código-fonte, uma (*scale_history*) foi rastreada para uma função de dados enquanto as outras cinco (*adodb_logsql*, *block_search_documents*, *role_sortorder*, *sessions2* e *webdav_locks*) não foram rastreadas para nenhuma função de dados. A tabela *scale_history* armazena uma cópia (*backup*) das configurações de escalas de notas (por exemplo, A+, A, A-, B, ...). Ela não é referenciada no código-fonte, pois utiliza uma camada de abstração que desconhecíamos, uma classe chamada *"XMLDBTable"*, criada para facilitar a manipulação de dados no Moodle.

As ligações entre as funções de dados e os arquivos de código-fonte foram obtidas transitivamente por meio das ligações entre as funções de dados e as tabelas e entre as tabelas e os arquivos de código-fonte. O peso da ligação entre uma função de dados e um arquivo de código-fonte foi calculado pela soma dos pesos das ligações entre todas as tabelas rastreáveis pela função de dados e o arquivo. Foram obtidas 2.076 ligações entre as 28 funções de dados e os 1.316 arquivos de código-fonte (06_EC_Moodle_Lig_FD_Arquivos.xlsx, no DVD anexo), totalizando 56.257 referências a funções de dados no código-fonte (por intermédio das referências a tabelas rastreadas para funções de dados). As 21.724 referências desconsideradas são referentes às 90 tabelas não rastreadas para funções de dados, mas cabe ressaltar que todas as funções de dados foram rastreadas.

4.1.2.2. Execução da Etapa das Funções Transacionais

Para a coleta das informações de entrada, o primeiro passo da etapa de criação das ligações entre as funções transacionais e o código-fonte do Moodle foi reinstalar um *build* do sistema, para que ele pudesse ser testado. No dia 11 de fevereiro de 2010 (oito meses após o *download* do *build* usado para as funções de dados[23]), um novo *build* do Moodle foi baixado, na versão 1.9.7, contendo 5.169 arquivos, sendo 2.346 arquivos de código-fonte escritos em PHP (07_EC_Moodle_ArquivosFT.xlsx, no DVD anexo).

O passo seguinte foi obter uma lista das funções transacionais do sistema. Ela foi construída manualmente, descobrindo quais são as operações oferecidas na interface com o usuário. Foram identificadas 35 funções transacionais no módulo de administração de usuários do Moodle (08_EC_Moodle_FuncoesTransacionais.xlsx, no

[23] Nesse período estava sendo conduzido um estudo exploratório sobre Evolução de Software usando a etapa das funções de dados da técnica proposta. Entretanto, não foram obtidos resultados relevantes a serem citados neste trabalho.

DVD anexo). Por restrições de tempo, a análise não foi estendida às funções transacionais dos outros módulos, como os dos acessos de alunos e professores, nem aos demais módulos do acesso de administrador do Moodle.

Em seguida, foram elaborados casos de teste que avaliassem as transações oferecidas pelo sistema. Cada função transacional exige que seja produzido pelo menos um caso de teste. Ao final, foram produzidos 43 casos de teste (09_EC_Moodle_CasosDeTeste.xlsx, no DVD anexo). Um desses casos de teste (o *logout*) foi criado apenas para permitir que fosse automatizada uma sequência específica de *logins* e *logouts* necessária aos testes e não tem origem a partir de uma função transacional.

O passo seguinte foi estabelecer as ligações entre as funções transacionais e os casos de teste. Essas ligações foram criadas manualmente durante a produção dos casos de teste (Seção 4.1.2.3), relacionando-os às funções transacionais que eles se propunham a testar. Foram compostas 42 ligações entre as 35 funções transacionais e 42 casos de teste (10_EC_Moodle_Lig_FT_CT.xlsx, no DVD anexo). O caso de teste *"Logout"* não foi rastreado para uma função transacional, pois não altera dados em tabelas de domínio e apenas manipula controles internos do sistema, como variáveis de sessão no servidor *web*.

Foram necessárias cerca de 45 horas para compor as informações de entrada da etapa das funções transacionais, desconsiderando outras 15 horas utilizadas para levantar casos de teste de módulos que não eram de administração de alunos e, portanto, não foram utilizados. Entrando na aplicação da técnica propriamente dita, foram estabelecidas as ligações entre os casos de teste e o código-fonte do sistema. Para isso, foi necessário monitorar a execução do sistema com uma ferramenta de cobertura de código. A ferramenta escolhida foi a XDebug[24], que pode ser acoplada ao servidor *web* onde o sistema está instalado, monitorando as linhas de código que estão sendo acionadas através de uma funcionalidade chamada rastreamento de funções (não confundir com a terminologia utilizada neste trabalho).

Com o sistema apto a ser monitorado, os casos de teste foram individual e progressivamente sendo executados (Seção 4.1.2.3), enquanto a execução do código era monitorada pela ferramenta de cobertura de código. Ao término da execução de cada caso de teste, os arquivos de *log* gerados pela ferramenta de cobertura de código eram copiados para uma pasta associada ao caso de teste.

Assim, foi produzido um *script* para analisar os *logs* gerados e obter a quantidade de acionamentos a linhas individuais (não repetidas) nos arquivos de

[24] http://www.xdebug.org/

código-fonte. Para cada caso de teste e arquivo de código-fonte encontrado no *log* foi criada uma ligação de rastreabilidade. Foram obtidas 2.698 ligações entre 43 casos de teste e 252 arquivos (11_EC_Moodle_Lig_CT_Arquivos.xlsx, no DVD anexo). Essas ligações receberam um peso, que indica a quantidade de linhas individuais acionadas por um caso de teste em um arquivo.

As ligações entre as funções transacionais e os arquivos de código-fonte foram obtidas transitivamente por meio das ligações entre as funções transacionais e os casos de teste e entre os casos de teste e os arquivos de código-fonte. O peso da ligação entre uma função transacional e um arquivo de código-fonte é calculado pela soma dos pesos das ligações entre todos os casos de teste rastreáveis pela função transacional e o arquivo. Foram obtidas 2.666 ligações entre 35 funções transacionais e 251 arquivos de código-fonte (12_EC_Moodle_Lig_FT_Arquivos.xlsx, no DVD anexo). Cabe ressaltar que a redução na quantidade de arquivos referenciados se deve a um deles ter sido acionado apenas pelo caso de teste *"Logout"*, que não estava rastreado para uma função transacional.

4.1.2.3. Automatização de testes

A integração ao processo de desenvolvimento é um dos requisitos das técnicas de rastreabilidade automática (CLELAND-HUANG *et al.*, 2007). Um dos objetivos desse requisito é facilitar a manutenção das ligações de rastreabilidade sem adicionar uma maior carga de trabalho à equipe.

Percebendo o grande esforço e tempo necessários à execução dos casos de teste neste estudo de caso, foi considerado que iria contra esse requisito a necessidade de executá-los manualmente a cada vez que fosse necessário realizar uma manutenção no sistema, exclusivamente com o objetivo de manter atualizadas as ligações de rastreabilidade.

Portanto, considerando essa ameaça à aplicabilidade da técnica, um estudo foi realizado para permitir a automatização dos casos de teste, viabilizando sua aplicação em um processo de desenvolvimento que utilize testes automatizados.

A ferramenta Selenium IDE[25] foi selecionada para automatizar os casos de teste. Ela pode ser instalada como *plugin* em um *browser*, permitindo que sejam gravados os cliques e digitações realizadas por um analista de testes para serem posteriormente reproduzidos automaticamente. A Figura IV-2 demonstra a interface da ferramenta, exibindo a suíte de testes e as ações de um dos seus casos de teste. Os testes automáticos podem ser executados pelo próprio *browser* ou exportados para algumas linguagens de programação, dentre elas, Java. Os casos de testes

[25] http://seleniumhq.org/projects/ide/

automatizados no Selenium IDE foram exportados para uso com o *framework* de testes JUnit[26], para Java.

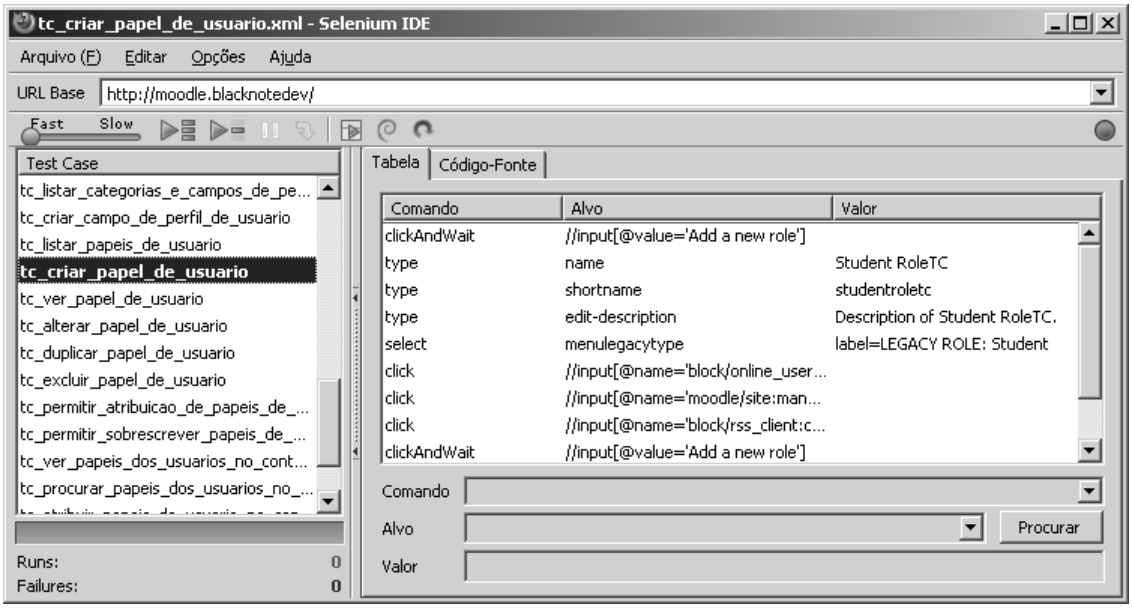


Figura IV-2: Interface do Selenium IDE

Foi desenvolvida uma suíte de testes que contemplava a execução de todos os casos de teste, usando recursos da ferramenta Selenium IDE. Essa suíte e os casos de teste foram alterados para, logo após a execução de um caso de teste, moverem os *logs* gerados para um diretório específico relacionado àquele caso de teste.

Porém, três dentre os 43 casos de teste selecionados não puderam ser automatizados. Os casos de teste *"Procurar usuário*" e *"Criar usuário (pelo próprio usuário)*" possuem um campo cujo identificador é gerado dinamicamente e a ferramenta não pôde registrar qual era o campo a atribuir um valor necessário. O caso de teste *"Criar usuário (pelo administrador)*" possui um campo validado por Javascript, que verifica a quantidade de teclas pressionadas para calcular a quantidade de caracteres inseridos. Essa forma de validação não reconheceu o valor atribuído pela ferramenta ao campo.

Como esses três casos de teste referentes às funções transacionais tiveram problemas com a automatização, considera-se que o sistema não foi projetado para testes de interface automáticos (ao menos usando a ferramenta Selenium IDE). Caso essas funcionalidades fossem desenvolvidas de outra forma, poderia ser viabilizado o teste completamente automático.

[26] http://www.junit.org/

Com a quase totalidade dos casos de teste identificados para o Moodle tendo sido automatizada, temos um indício de que a criação de ligações para as funções transacionais pode ser integrada ao processo de desenvolvimento tradicional, bastando para isso que se tenha a devida atenção com a compatibilidade às ferramentas de automatização de testes de interface, o que facilitaria não apenas a aplicação da técnica de rastreabilidade proposta, mas também os testes.

4.1.2.4. Análise

Consolidando os dados obtidos pela criação das ligações entre as funções de dados e o código-fonte no estudo de caso sobre o Moodle, chegou-se aos conjuntos e relações expostos na Figura IV-3.

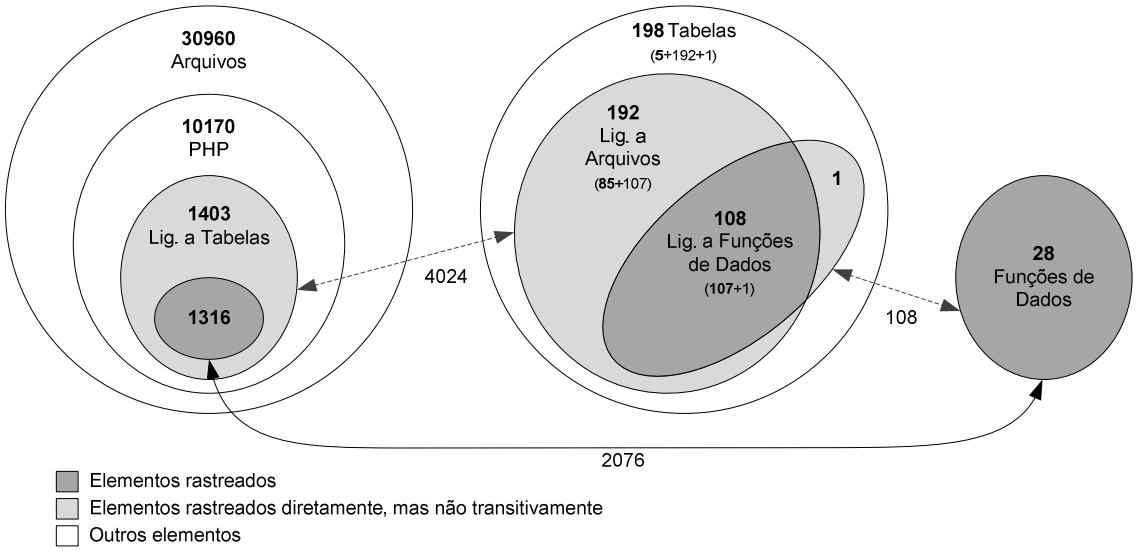


Figura IV-3: Dados obtidos na etapa das funções de dados do estudo de caso sobre o Moodle

Neste estudo de caso, tabelas que foram consideradas como não relacionadas a funções de dados não foram descartadas, ao contrário da abordagem do estudo de caso anterior. Isso adicionou maior flexibilidade de correção de erros, possibilitando atualizar as ligações indiretas sem precisar fazer novas buscas ou executar os casos de teste mais de uma vez.

Diversas tabelas não foram rastreadas para funções de dados: 90 de um total de 198 tabelas (45%). Aparentemente, no esquema de banco de dados do Moodle, muitas tabelas são utilizadas para mecanismos internos de implementação, não relacionados a funções de dados.

O arquivo com métodos genéricos de acesso a dados pode ser considerado uma camada de abstração. Entretanto, devido à presença dos nomes das tabelas nas chamadas a esses métodos, foi possível localizar essas chamadas ao invés das

queries. Isso traz um indício de que pode ser viável ultrapassar a limitação da técnica proposta quando camadas de abstração são utilizadas. Entretanto, esse aspecto não chegou a ser investigado no presente trabalho.

Das 198 tabelas do sistema, seis (3%) não foram rastreadas para arquivos. Apesar de um número relativamente baixo, tendeu-se a considerar que todas as tabelas deveriam estar presentes no código-fonte, caso contrário, o sistema teria tabelas sem utilidade. O motivo dessa perda não chegou a ser investigado no presente trabalho.

Ao final, todas as 28 funções de dados identificadas foram rastreadas para 1.316 dos 10.170 arquivos de código-fonte (13%), um relativamente baixo percentual, mas esperado, pois 85 das 198 tabelas rastreadas para os arquivos de código-fonte (42%) não estavam ligadas a funções de dados. Ainda, um total de 87 dos 1.403 arquivos rastreados para tabelas (6%) não foi relacionado às funções de dados, pois as tabelas manipuladas pelos arquivos não estavam relacionadas às funções de dados, um percentual baixo se comparado aos 58% de tabelas não ligadas a funções de dados.

A necessidade de *download* de um novo código-fonte para permitir a realização da etapa de criação das ligações das funções transacionais suscitou um fator previamente desconhecido: utilizar uma duas versões distintas do código-fonte na etapa de criação de ligações para as funções de dados torna desconexos os arquivos e ligações entre as etapas da técnica proposta.

Consolidando os dados obtidos pela criação das ligações entre as funções de transacionais e o código-fonte no estudo de caso sobre o Moodle, chegou-se aos conjuntos e relações expostos na Figura IV-4.

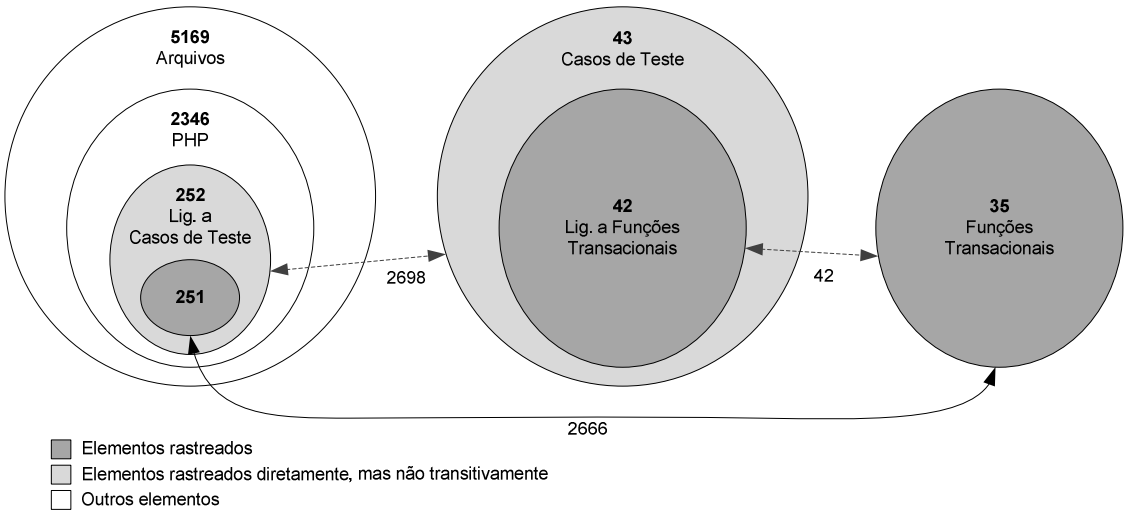


Figura IV-4: Dados obtidos na etapa das funções transacionais do estudo de caso sobre o Moodle

Analisando os quantitativos dos elementos e ligações, foram verificadas algumas perdas de informação (o causador e o impacto dessas perdas não chegou a ser investidado), pois devido a restrições de tempo, apenas uma parte do sistema teve casos de teste implementados e executados na etapa de criação das ligações para funções transacionais. Isso levou à realização do terceiro estudo de caso, que endereçou um sistema de menor porte, onde a pesquisa pôde ser feita para todo o sistema.

O *download* do código-fonte para a etapa das funções transacionais reduziu a quantidade de arquivos de 30.960 para 5.169 (17% do anterior) e de arquivos de código-fonte de 10.170 para 2.346 (23%), quase todos oriundos do diretório *"contrib"*, que possui diversas versões de módulos de extensão das funcionalidades do Moodle.

Existia 1 de 43 casos de teste (2%) não relacionado a funções transacionais. Apesar de inicialmente ele parecer desnecessário à técnica, é de execução obrigatória para a sequência de testes. Por exemplo, para que seja executado o caso de teste *"Login como professor (sucesso)*" após a execução do caso de teste *"Login como administrador (sucesso)*" (ambos relacionados a funções transacionais), é necessário executar o *"Logout*" (que não está relacionado) para que seja encerrado o estado atual da sessão e a ferramenta de automatização de testes possa ter autonomia.

Apenas o módulo de administração de usuários teve funções transacionais elicitadas e casos de teste planejados devido a limitações de tempo para a pesquisa. Por esse motivo, os dados obtidos para as ligações das funções transacionais ficaram seriamente limitados. Isso fez com que fosse considerado desnecessário avaliar os índices de relevância neste estudo de caso, pois dados incorretos seriam obtidos a partir de um conjunto incompleto de funções transacionais. Entretanto, o estudo de caso seguinte iria contemplar todos os aspectos da técnica proposta, pois se trata de um sistema de pequeno porte se comparado aos dois anteriores, viabilizando a produção completa de todas as entradas da técnica.

4.1.3. Sistema de Inscrições Online

A condução deste estudo de caso tem o objetivo de apresentar a aplicação da técnica proposta na criação de ligações de rastreabilidade entre as funções de dados e o código-fonte, bem como a criação de ligações entre as funções transacionais e o código-fonte de um Sistema de Informação.

O sistema selecionado para o estudo de caso foi o Sistema de Inscrições Online do PPGI (doravante chamado de SIO), que realiza o cadastro de candidatos e suas inscrições para um edital de seleção. Ele foi desenvolvido na UNIRIO e é

utilizado para gerenciar as inscrições no curso de Mestrado oferecido pelo PPGI/UNIRIO. O acesso ao seu código-fonte foi permitido pelo seu desenvolvedor.

4.1.3.1. Execução da Etapa das Funções de Dados

Para a coleta das informações de entrada, foi definido o nível de granularidade das ligações como o de arquivo. O primeiro passo da etapa de criação das ligações entre as funções de dados e o código-fonte do SIO foi obter seu código-fonte, entregue pelo seu desenvolvedor no dia 15 de abril de 2010. Um total de 67 arquivos foi recebido, incluindo 42 arquivos de código-fonte escritos em PHP (01_EC_SIO_Arquivos.xlsx, no DVD anexo) e um *script* de criação do banco de dados MySQL.

O passo seguinte foi obter uma lista das funções de dados do sistema. Ela foi construída manualmente, analisando as tabelas que compunham o banco de dados, testando o sistema e observando os dados inseridos no banco de dados. As funções de dados identificadas foram confirmadas com o desenvolvedor. Esse passo foi decomposto em uma série de atividades. A primeira delas foi obter uma lista das tabelas pertencentes ao banco de dados, analisando o *script* de criação do banco de dados, que continha 8 tabelas (02_EC_SIO_Tabelas.xlsx, no DVD anexo). A segunda atividade foi executar o *script* de criação do banco de dados e instalar o sistema. A terceira atividade foi realizar um agrupamento de tabelas que pareciam representar uma mesma função de dados para obter uma lista inicial de funções de dados. A quarta atividade foi realizar testes de execução *ad hoc* no sistema para confirmar se cada função de dados proposta era coerente no domínio do sistema e encontrar as tabelas que tiveram seus dados alterados pelo teste realizado. A quinta e última atividade foi verificar com o desenvolvedor se as funções de dados estavam corretas. Seis funções de dados foram identificadas (03_EC_SIO_FuncoesDeDados.xlsx, no DVD anexo).

Em seguida, foram estabelecidas as ligações entre as funções de dados e as tabelas com base nos testes realizados, compondo oito ligações (04_EC_SIO_Lig_FD_Tabelas.xlsx, no DVD anexo). Cabe ressaltar que todas as tabelas foram rastreadas.

Foram necessárias cerca de 8 horas para compor as informações de entrada da etapa das funções de dados. Entrando na aplicação da técnica propriamente dita, foram estabelecidas as ligações entre as tabelas e o código-fonte do sistema. Foi produzido um *script* para localizar as referências a tabelas nos arquivos de código-fonte. Foram encontradas 51 referências a tabelas em 11 arquivos de código-fonte. Eliminando referências duplicadas entre uma dada tabela e um dado arquivo, foram

formadas 22 ligações entre as 8 tabelas e os 11 arquivos (05_EC_SIO_Lig_Tabelas_Arquivos.xlsx, no DVD anexo). Essas ligações receberam um peso, que indica a quantidade de referências entre a mesma tabela e arquivo.

As ligações entre as funções de dados e os arquivos de código-fonte foram obtidas transitivamente por meio das ligações entre as funções de dados e as tabelas e entre as tabelas e os arquivos de código-fonte. O peso da ligação entre uma função de dados e um arquivo de código-fonte foi calculado pela soma dos pesos das ligações entre todas as tabelas rastreáveis pela função de dados e o arquivo. Foram obtidas 20 ligações (com índices de relevância calculados) entre as 6 funções de dados e os 11 arquivos de código-fonte (06_EC_SIO_Lig_FD_Arquivos.xlsx, no DVD anexo), totalizando as mesmas 51 referências. Cabe ressaltar que todas as funções de dados foram rastreadas.

4.1.3.2. Execução da Etapa das Funções Transacionais

Para a coleta das informações de entrada, o primeiro passo da etapa de criação das ligações de rastreabilidade entre as funções transacionais e o código-fonte do SIO foi obter uma lista das suas funções transacionais. Diferentemente do estudo de caso anterior, onde o sistema foi reinstalado, neste foi utilizada a mesma versão do código-fonte da etapa anterior. A lista das funções transacionais foi construída manualmente, a partir das operações oferecidas na interface com o usuário do sistema. Foram identificadas 17 funções transacionais (07_EC_SIO_FuncoesTransacionais.xlsx, no DVD anexo), posteriormente confirmadas com o desenvolvedor do sistema.

Em seguida, foram elaborados casos de teste que avaliassem as transações oferecidas pelo sistema. Cada função transacional necessita que seja produzido ao menos um caso de teste e, ao final, foram produzidos 22 casos de teste (08_EC_SIO_CasosDeTeste.xlsx, no DVD anexo).

O passo seguinte foi estabelecer as ligações entre as funções transacionais e os casos de teste. Essas ligações foram criadas manualmente durante a produção dos casos de teste, relacionando-os às funções transacionais que eles se propunham a testar, compondo 20 ligações entre as 17 funções transacionais e 20 casos de teste (09_EC_SIO_Lig_FT_CT.xlsx, no DVD anexo). Dois casos de teste não foram rastreados por não estar relacionados a funções transacionais: *"Exibir página inicial (formulário de login)*" e *"Fazer logout*". Estas operações não manipulam dados em arquivos internos ou interfaces externas no sistema e, portanto, não foram consideradas funções transacionais.

Foram necessárias cerca de 12 horas para compor as informações de entrada da etapa das funções transacionais. Entrando na aplicação da técnica propriamente dita, foram estabelecidas as ligações entre os casos de teste e o código-fonte do sistema. Para isso, foi necessário fazer com que o sistema fosse monitorado por uma ferramenta de cobertura de código. A ferramenta escolhida foi novamente a XDebug (já utilizada em um estudo de caso anterior, ver Seção 4.1.2.2). Os 22 casos de teste foram executados individualmente, enquanto a execução era monitorada pela ferramenta de cobertura de código. Ao término da execução de cada caso de teste, os arquivos de *log* gerados pela ferramenta de cobertura de código eram copiados para uma pasta associada ao caso de teste.

Assim, foi produzido um *script* para analisar os *logs* gerados e obter a quantidade de acionamentos a linhas individuais (não repetidas) nos arquivos de código-fonte. Para cada caso de teste e arquivo de código-fonte encontrado no *log* foi criada uma ligação de rastreabilidade. Foram obtidas 212 ligações entre os 22 casos de teste e 37 arquivos (10_EC_SIO_Lig_CT_Arquivos.xlsx, no DVD anexo). Essas ligações receberam um peso, que indica a quantidade de linhas individuais acionadas por um caso de teste em um arquivo. Cabe ressaltar que 5 arquivos de código-fonte não foram rastreados.

As ligações entre as funções transacionais e os arquivos de código-fonte foram obtidas transitivamente por meio das ligações entre as funções transacionais e os casos de teste e entre os casos de teste e os arquivos de código-fonte. O peso da ligação entre uma função transacional e um arquivo de código-fonte foi calculado pela soma dos pesos das ligações entre todos os casos de teste rastreáveis pela função transacional e o arquivo. Foram obtidas 175 ligações (com índices de relevância calculados) entre as 17 funções transacionais e 36 arquivos de código-fonte (11_EC_SIO_Lig_FT_Arquivos.xlsx, no DVD anexo). Cabe ressaltar que a redução na quantidade de arquivos referenciados se deve ao arquivo ter sido acionado apenas pelo caso de teste "*Fazer logout*", que não foi rastreado para uma função transacional.

4.1.3.3. Análise

Consolidando os dados obtidos pela criação das ligações entre as funções de dados e o código-fonte no estudo de caso sobre o SIO, chegou-se aos conjuntos e relações expostos na Figura IV-5.

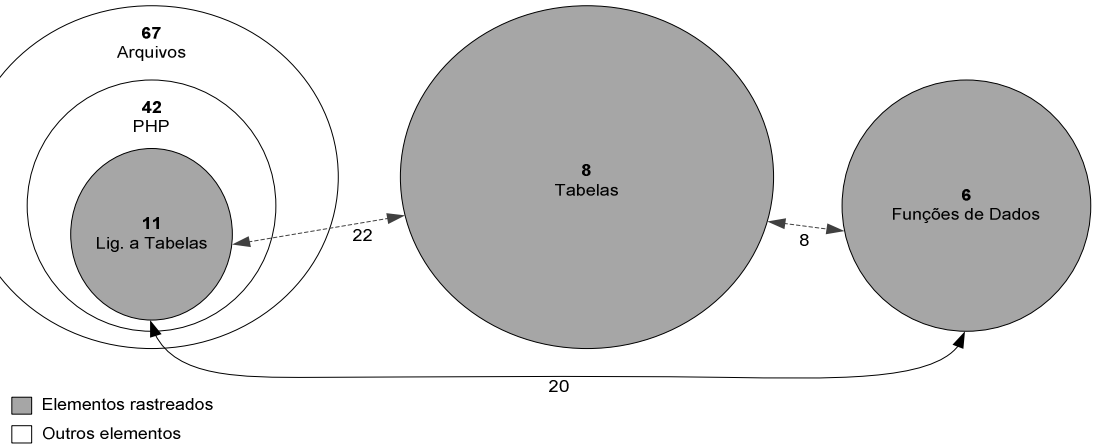


Figura IV-5: Dados obtidos na etapa das funções de dados do estudo de caso sobre o SIO

Analisando os quantitativos dos elementos e ligações, foi verificado que quase não ocorreu perda de informação, pois um maior percentual de artefatos foi rastreado quando comparamos com os demais estudos de caso. Além disso, apenas neste estudo de caso todas as tabelas foram rastreadas para funções de dados, talvez por se tratar de um sistema de pequeno porte, que não possui a mesma complexidade dos sistemas objeto dos estudos de caso anteriores.

Não existe uma camada de abstração na manipulação de dados deste sistema e foi verificado que as suas 8 tabelas estão referenciadas em 11 de 42 arquivos de código-fonte (26%). A presença de todas as tabelas no código-fonte é um comportamento que foi percebido apenas neste estudo de caso.

Ao final, todas as 6 funções de dados identificadas foram rastreadas para os mesmos 11 arquivos de código-fonte. Os índices de relevância calculados, quando somados, chegam sempre à mesma quantidade das funções de dados (seis).

Consolidando os dados obtidos pela criação das ligações entre as funções transacionais e o código-fonte no estudo de caso sobre o SIO, chegou-se aos conjuntos e relações expostos na Figura IV-6.

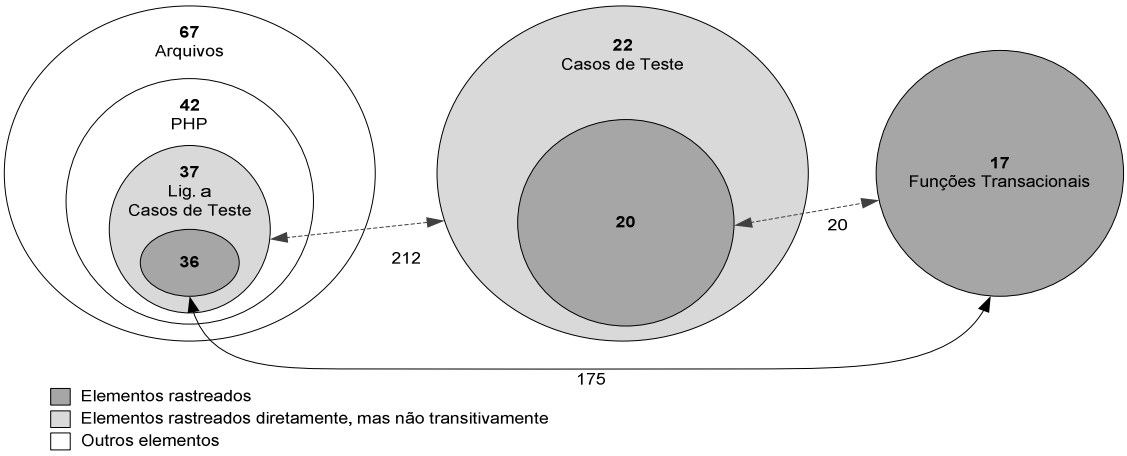


Figura IV-6: Dados obtidos na etapa das funções transacionais do estudo de caso sobre o SIO

Existiam 2 de 22 casos de teste (9%) não relacionados a funções transacionais. Como todas as operações oferecidas pelo sistema tiveram casos de teste implementados, foi simulada uma situação real, onde todo o sistema foi testado e ligações de rastreabilidade foram criadas entre todas as funções transacionais e os módulos de código-fonte.

Era esperado que todos os 42 arquivos de código-fonte fossem rastreados para os casos de teste. Contudo, 5 deles (12%) não foram rastreados. Percebeu-se que um desses arquivos possui apenas *tags* HTML (*footer.php*) e os outros 4 se referiam às funcionalidades de *"Curso de graduação"* e *"Esqueci minha senha"*, que foram parcialmente implementadas e, por isso, ainda não estão acessíveis no sistema.

Finalmente, como os casos de teste não relacionados a funções transacionais não influenciam os pesos das ligações transitivas e o arquivo *"usuario_logout.php"* está relacionado a um caso de teste não relacionado a funções transacionais, as ligações para esse arquivo não são consideradas.

4.2. Protótipo

Esta seção apresentará o FPT Plugin (de Function Point Tracing Plugin), um protótipo de ferramenta de rastreabilidade desenvolvido para demonstrar como as ligações criadas pela técnica proposta na Seção 3.2 e avaliada nos estudos de caso apresentados na Seção 4.1 poderiam ser utilizadas de forma integrada ao ambiente de desenvolvimento. O protótipo é um *plugin* para o Eclipse, que terá suas funcionalidades descritas e algumas potenciais aplicações ensaiadas nas próximas seções. Sua arquitetura é descrita brevemente no Anexo I.

O desenvolvimento do protótipo foi motivado pela necessidade de exibir informações de rastreabilidade relevantes e adequadas a um determinado público, pois a simples existência de ligações de rastreabilidade não faz com que seus benefícios sejam alcançados.

Alguns dos possíveis usuários de informações de rastreabilidade são arquitetos de *software* e programadores. Esses profissionais freqüentemente utilizam um ambiente de desenvolvimento integrado (IDE) em sua rotina de trabalho e poderiam se beneficiar por terem informações de rastreabilidade inseridas nas suas ferramentas de trabalho, auxiliando-os nas suas tarefas.

Dentre essas informações, no contexto da técnica proposta, foi decidido auxiliar a localização mais rápida dos artefatos de código-fonte responsáveis por implementar certa função de dados ou transacional, bem como apresentar as funções de dados e transacionais que se utilizam de um artefato de código-fonte. Com essas informações

disponíveis, a intenção é melhorar a compreensão do *software*, a análise de impacto de uma mudança, a manutenção e, em menor escala, os demais benefícios relacionados na Seção 2.2.

Entretanto, as afirmações acima postuladas se tratam de hipóteses baseadas nesses benefícios esperados das abordagens tradicionais de rastreabilidade (Seção 2.2). A presente pesquisa não se propõe a comprovar que esses benefícios são efetivamente alcançados com a ferramenta proposta e que traz aumento de produtividade em uma situação real – a intenção é apenas indicar um possível uso para as ligações de rastreabilidade geradas pela técnica.

O *FTP Plugin* é uma extensão (*plugin*) para o ambiente de desenvolvimento integrado Eclipse, muito utilizado para o desenvolvimento de *software* com a linguagem Java, mas também compatível com outras linguagens, como C, C++ e PHP. Sua interface com o usuário (Figura IV-7) é composta por duas visões (abas) que podem ser inseridas no ambiente de trabalho do desenvolvedor, estendendo as funcionalidades do Eclipse: *FPT Navigator* e *FPT File Details*.

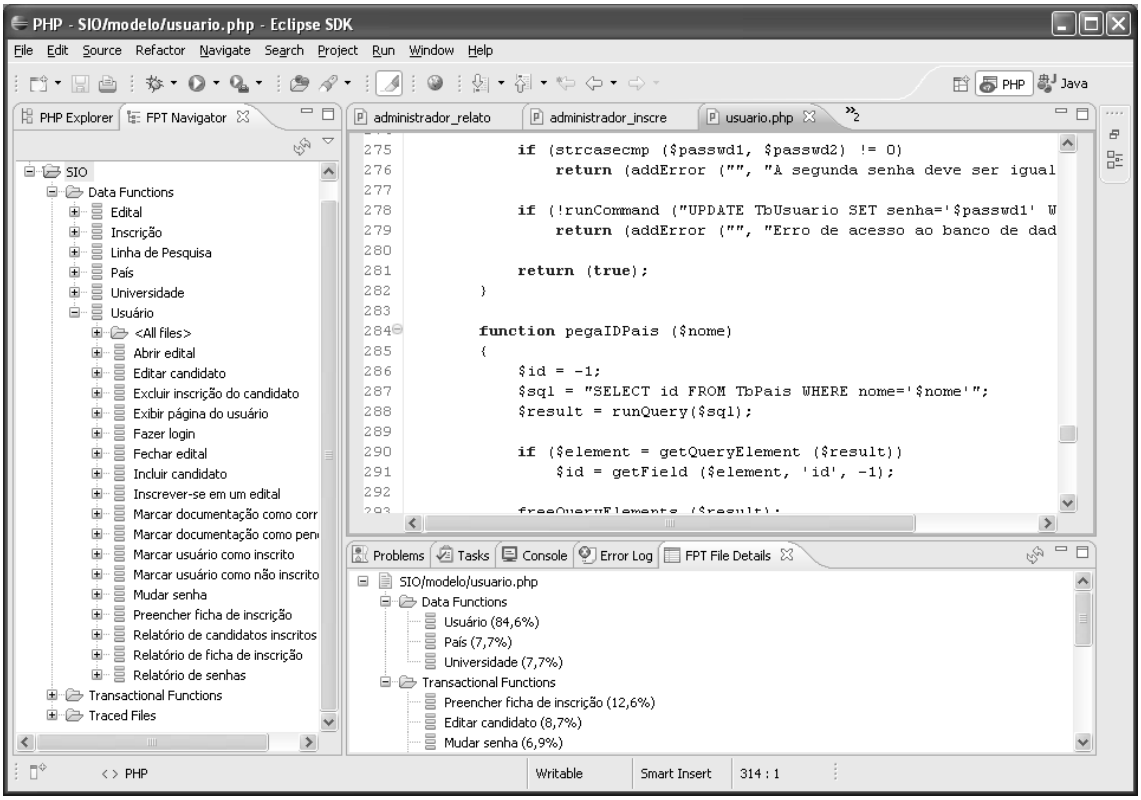


Figura IV-7: Interface do FPT Plugin

A visão *FPT Navigator* oferece diversas informações de rastreabilidade sobre o projeto ativo no momento. Ela possui uma lista das funções de dados (opção *"Data Functions"*) e uma lista das funções transacionais (opção *"Transactional Functions"*),

possibilitando a identificação dos arquivos de código-fonte a elas relacionadas. Possui ainda uma lista dos arquivos de código-fonte (opção *"Traced Files"*) que possuem ligações de rastreabilidade, possibilitando acesso direto aos mesmos e ignorando arquivos não relacionados às funções de dados ou transacionais.

Ao abrir a opção *"Data Functions"* na árvore com o nome do projeto, são exibidas todas as funções de dados do sistema. Clicando em uma delas, é exibida uma opção *"<All Files>"* e opções para as funções transacionais que tiverem arquivos em comum com a função de dados selecionada. A opção *"<All Files>"* lista todos os arquivos relacionados àquela função de dados. Ela possui ainda, opções para cada função transacional, onde são listados todos os arquivos relacionados tanto à função de dados quanto à função transacional selecionada. Um exemplo de uso prático desses recursos seria possibilitar a um arquiteto de sistemas identificar rapidamente quais arquivos podem ser impactados por uma alteração na função de dados *"Inscrição"*, mais especificamente na função transacional *"Excluir inscrição do candidato"*. Conhecendo esses arquivos, caberia agora uma análise focada no conteúdo dos mesmos.

Ao abrir a opção *"Transactional Functions"* na árvore com o nome do projeto, são exibidas todas as funções transacionais do sistema. Clicando em uma delas, é exibida uma opção *"<All Files>"* e opções para as funções de dados que tiverem arquivos em comum com a função transacional selecionada. A opção *"<All Files>"* lista todos os arquivos relacionados àquela função transacional. Ela possui ainda, opções para cada função de dados, onde são listados todos os arquivos relacionados tanto à função transacional quanto à função de dados selecionada. Um exemplo de uso seria possibilitar a identificação rápida de quais arquivos podem ser impactados por uma alteração na função transacional *"Inscrever-se em um edital"*, mais especificamente na função de dados *"Inscrição"*.

Ao abrir a opção *"Traced Files"* na árvore com o nome do projeto, são exibidos todos os arquivos que possuem ligações de rastreabilidade seja para uma função de dados ou para uma função transacional qualquer. Esta opção deve ser utilizada para abrir um arquivo cujo nome seja conhecido sem a necessidade de acessá-lo por meio de uma função de dados ou transacional. É uma alternativa à visão padrão do Eclipse para acessar os arquivos do projeto.

A visão *FPT File Details* exibe informações de rastreabilidade sobre o arquivo de código-fonte aberto e visível no momento. São listadas todas as funções de dados e transacionais que possuam ligações de rastreabilidade para o arquivo. Como exemplo, pode-se ter uma situação onde um desenvolvedor precisa fazer uma alteração no comportamento da função de dados *"Inscrição"* e abriu um arquivo a ela

relacionado (usando a visão *"FPT Navigator"*). Após abrir esse arquivo, o desenvolvedor identificou (usando a visão *"FPT File Details"*) que a alteração proposta também impactaria as funções de dados *"Usuário"* e *"Edital"*, submetendo a situação para avaliação do arquiteto.

Podem ser exemplificadas algumas outras situações onde o *plugin* poderia ser útil:

- Um desenvolvedor recentemente inserido no projeto precisa entender como o sistema foi estruturado internamente. Neste caso, ele utiliza o *FPT Navigator* e identifica os dados e transações oferecidos pelo sistema; verifica os arquivos de código-fonte que implementam esses dados e transações e começa a perceber de maneira *ad hoc* (auxiliado pelos pesos) quais são os principais arquivos em cada funcionalidade;
- Um desenvolvedor precisa adicionar campos de dados (TED, na nomenclatura da APF) em uma função de dados. Utiliza o *FPT Navigator* e identifica os arquivos onde essa função de dados é manipulada. Então, tem o escopo de busca reduzido para localizar os locais que precisa alterar;
- Um arquiteto é informado de que diversos módulos do sistema estão com problemas de desempenho; realiza um teste em uma transação e identifica que um método está levando mais tempo para ser executado do que aparentemente deveria, sendo o provável gerador da lentidão nessa transação. Ao abrir o arquivo que possui esse método, percebe pela visão *FPT File Details* que que este arquivo está presente em diversas transações. Identifica um trecho de código que deve ser otimizado e o encaminha para um desenvolvedor. Com o sistema atualizado, é percebida uma melhora significativa no desempenho geral do sistema.

Mais aplicações poderiam ser expostas para o protótipo proposto. Entretanto, apenas esses exemplos são suficientes para apresentar brevemente algumas situações em que ele poderia ser utilizado, não esgotando todas as suas possibilidades e sua efetividade no processo de desenvolvimento de *software*.

O uso *ad hoc* do protótipo com as ligações de rastreabilidade obtidas nos estudos de caso do Moodle e do Sistema de Inscrições Online do PPGI demonstrou que as visões oferecidas parecem facilitar a identificação dos arquivos referenciados por uma determinada funcionalidade dentre os diversos arquivos disponíveis.

Os percentuais exibidos no protótipo ao lado de funções de dados, funções transacionais e arquivos são referentes aos índices de relevância das ligações. Freqüentemente eram encontradas funções de dados e transacionais cujos arquivos com maior índice de relevância possuíam nomes que se assemelhavam aos das

funcionalidades a que estão relacionados e que faziam sentido no contexto dos seus sistemas. Isso demonstra que a quantidade de referências a tabelas (para funções de dados) e a linhas acionadas (para funções transacionais), assim como os índices de relevância, são indicadores que podem auxiliar a descobrir quais são os arquivos mais importantes na implementação de uma funcionalidade.

Ao abrir um arquivo de código-fonte são exibidas as funções de dados e transacionais a que ele está relacionado e o índice de relevância da ligação. Essa visão permite que sejam identificadas as funcionalidades onde aquele arquivo é utilizado, de onde se conclui que a alteração desse arquivo terá possíveis impactos nessas funcionalidades.

4.3. Considerações finais

Este capítulo apresentou os estudos realizados para avaliar a técnica de rastreabilidade proposta. Durante o primeiro estudo de caso foi desenvolvida a etapa das funções de dados. O segundo estudo de caso aplicou essa mesma etapa com algumas modificações e incluiu o desenvolvimento da etapa de análise das funções transacionais. Nele, foi considerado demorado (para os objetivos desta pesquisa) criar os casos de teste quando estes não foram previamente produzidos (ou não se tinha acesso aos mesmos). O terceiro estudo de caso consistiu na aplicação completa da técnica, que ao final obteve dados sobre o sistema que, analisados de maneira *ad hoc*, foram considerados consistentes.

O protótipo desenvolvido apresenta como informações obtidas das ligações de rastreabilidade criadas pela técnica podem ser inseridas em uma ferramenta de trabalho de um arquiteto ou desenvolvedor de *software*, possivelmente facilitando seu entendimento sobre a lógica de um sistema, a manutenção e a análise de impacto de mudanças.

Conforme foi percebido nos estudos de caso, o tempo necessário para a criação das informações de entrada para a técnica, caso as mesmas não existam, não pode ser desprezado. Atividades como a engenharia reversa das funções de dados e transacionais a partir dos artefatos produzidos para o sistema, estabelecer as ligações de rastreabilidade de entrada e produzir casos de teste demandaram um esforço razoável na pesquisa. Isso deve ser levado em consideração na aplicação prática da técnica e trabalhos futuros deveriam se encarregar de melhorar este cenário, oferecendo melhores recursos obter as informações de entrada da técnica.

Os principais indicadores obtidos nos estudos de caso são as quantidades de cada tipo de elemento que participa das ligações e a quantidade de ligações. Para

efeito de comparação entre os estudos de caso, esses indicadores foram tabulados na Tabela IV-1, apresentada logo adiante.

Deve ser ressaltado que o foco da avaliação realizada foi detectar onde a técnica deveria melhorar durante seu desenvolvimento e buscar traços que indicassem que poderia ser aplicável em uma situação real na indústria. Entretanto, não foram realizados estudos que pudessem ser mais assertivos sobre essa aplicabilidade e comparação com outras técnicas automáticas ou semi-automáticas de rastreabilidade. Tais estudos devem ser objeto de trabalhos futuros.

Tabela IV-1: Dados consolidados dos estudos de caso

INDICADOR	COMPIERE		MOODLE		SIO	
	Nº	%	Nº	%	Nº	%
Arquivos	5.010	100%	30.960 5.169	100%	67	100%
Arquivos de Código-Fonte	2.277	45%	10.170 2.346	33% 45%	42	63%
Tabelas	465	100%	198	100%	8	100%
Arquivos de Código-Fonte ligados a Tabelas	700	31%	1.403	14%	11	26%
Tabelas ligadas a Arquivos de Códigos-Fonte	83	18%	192	97%	8	100%
Ligações entre Arquivos de Código-Fonte e Tabelas	162	100%	28	100%	6	100%
Funções de Dados	211	45%	108	55%	8	100%
Tabelas ligadas a Funções de Dados	162	100%	28	100%	6	100%
Funções de Dados ligadas a Tabelas	211	100%	108	100%	8	100%
Ligações entre Tabelas e Funções de Dados	700	31%	1.316	13%	11	26%
Arquivos de Código-Fonte ligados a Funções de Dados	76	47%	28	100%	6	100%
Funções de Dados ligadas a Arquivos de Código-Fonte	2.608	100%	4.024	100%	22	100%
Ligações Arqs. de Código-Fonte e Funções de Dados	700	100%	2.076	100%	20	100%
Casos de Teste	-	-	291	100%	22	100%
Arquivos de Código-fonte ligados a Casos de Teste	-	-	252	11%	37	88%
Casos de Teste ligados a Arquivos de Código-Fonte	-	-	43	15%	22	100%
Ligações Arquivos de Código-Fonte e Casos de Teste	-	-	2.698	100%	212	100%
Funções Transacionais	-	-	205	100%	17	100%
Casos de Teste ligados a Funções Transacionais	-	-	228	78%	20	91%
Funções Transacionais ligadas a Casos de Teste	-	-	205	100%	17	100%
Ligações Casos de Teste e Funções Transacionais	-	-	228	100%	20	100%
Arqs. Código-Fonte ligados a Funções Transacionais	-	-	251	11%	36	86%
Funções Transacionais lig. a Arquivos de Código-Fonte	-	-	35	17%	17	100%
Lig. Arquivos Código-Fonte e Funções Transacionais	-	-	2.666	100%	175	100%

V. Conclusão

A necessidade por uma TI ágil em atender mudanças no negócio motivou o surgimento de inúmeros padrões e metodologias para desenvolvimento de *software*. A Rastreabilidade de Requisitos é uma prática recomendada por diversos padrões relacionados à Gerência de Requisitos para permitir entender mais rapidamente a lógica de um sistema, mantê-lo, analisar impacto de mudanças, dentre outros. Por outro lado, a Análise de Ponto de Função é uma técnica muito aplicada para basear estimativas de esforço, tempo e custo.

Atualmente a rastreabilidade está presente como uma das funcionalidades de algumas ferramentas de Gerenciamento de Requisitos. Entretanto, elas dependem de entradas manuais. Enquanto isso, ferramentas acadêmicas que se propõem a automatizar a rastreabilidade não são maduras o suficiente para atender as necessidades de negócio da indústria.

A literatura indica que é necessário aumentar a automatização das ferramentas de rastreabilidade, reduzindo o esforço manual para geração das entradas, obtendo informações do processo de desenvolvimento e exibindo essas informações diretamente nas ferramentas utilizadas pelos profissionais que dela se beneficiarão (CLELAND-HUANG *et al.*, 2007).

O presente trabalho aborda o uso da rastreabilidade de requisitos como auxílio à realização de estimativas em projetos de manutenção de *software*. Nesse sentido, a Análise de Ponto de Função é aplicada como uma representação básica dos requisitos, pois freqüentemente é utilizada para representar o escopo de um projeto de criação de um novo Sistema de Informação.

A solução proposta é uma técnica semi-automática de rastreabilidade que permite estabelecer ligações entre elementos de modelo da Análise de Ponto de Função (funções de dados e transacionais) e elementos do código-fonte do sistema (métodos, classes, arquivos, dentre outros). As funções de dados são representações dos dados manipulados pelo sistema em uma esfera conceitual e não podem ser encontradas diretamente no produto desenvolvido. Devido a isso, as tabelas do banco de dados do sistema são utilizadas como uma ponte entre as funções de dados e o

código-fonte, pois em diversos sistemas podem ser encontradas referências a essas tabelas no seu código-fonte. As funções transacionais são representações das transações (operações) oferecidas pelo sistema, mas também não podem ser encontradas diretamente no produto desenvolvido. Por isso, os casos de teste são utilizados como uma ponte entre as funções transacionais e o código-fonte. Quando executados, os casos de teste acionam uma seqüência de linhas de código. Monitorando esses acionamentos, pode-se identificar quais casos de teste acionam determinadas partes do código-fonte.

A avaliação da aplicação da técnica proposta foi feita através de três estudos de caso e um protótipo. No primeiro estudo de caso foi analisada a criação de ligações de rastreabilidade para as funções de dados do Compiere. No segundo, foi analisada a criação de ligações para as funções de dados e transacionais do Moodle, incluindo a automatização dos casos de teste. No terceiro, foi analisada a criação de ligações para as funções de dados e transacionais do Sistema de Inscrições Online do PPGI. O protótipo demonstra como programadores e arquitetos podem ter acesso a informações de rastreabilidade diretamente em seu ambiente de desenvolvimento integrado (IDE).

A cada estudo de caso que se seguia, os dados contemplavam um maior percentual dos elementos nos conjuntos de funções de dados, transacionais, tabelas, casos de teste e arquivos de código-fonte. Isso demonstra a evolução da técnica, ocorrida devido aos ajustes realizados durante a realização dos estudos de caso. A automatização dos casos de teste, realizada no estudo de caso sobre o Moodle, demonstrou que não é necessária a execução manual dos casos de teste para apoiar a criação de ligações de rastreabilidade, mantendo a prática de testes automáticos utilizada atualmente, sem adicionar esse esforço à carga de trabalho da equipe. O protótipo produzido demonstra a possibilidade de exibir à equipe os arquivos necessários a uma função de dados ou transacional e, inversamente, as funções de dados e transacionais dependentes de um determinado arquivo.

A importância do índice de relevância proposto para as ligações de rastreabilidade é evidenciada no protótipo, pois indica percentualmente quanto uma função de dados ou transacional está relacionada a um arquivo e, inversamente, quanto um arquivo está relacionado a uma função de dados ou transacional. Isso permite ordená-los iniciando pelo mais relevante, o que pode indicar o provável artefato que sofrerá impacto no caso de uma manutenção.

5.1. Contribuições

A seguir são apresentadas as contribuições mais relevantes deste trabalho:

- Identificação de um problema referente à análise de impacto de alterações em projetos de manutenção de *software*;
- Uma revisão bibliográfica sobre a área da Rastreabilidade de Requisitos, realizada com base em trabalhos clássicos e incluindo os mais recentes desenvolvimentos e propostas na área;
- Definição e implementação de uma técnica para estabelecer ligações entre elementos da Análise de Ponto de Função e elementos do código-fonte de um sistema;
- Definição, planejamento, execução e análise dos resultados de três estudos de caso de aplicação da técnica proposta, realizados com sistemas de duas linguagens de programação distintas, portes distintos, onde dois sistemas são *open source* e um foi produzido internamente para uso do PPGI/UNIRIO;
- Definição, planejamento, desenvolvimento e análise de um protótipo de ferramenta de rastreabilidade que usa os resultados da técnica proposta.

5.2. Limitações

A técnica proposta neste trabalho é uma simplificação de uma realidade complexa e inerente ao contexto da Rastreabilidade de Requisitos. A seguir, são apresentadas algumas questões que podem comprometer a confiabilidade da proposta.

- Camadas de abstração do acesso a dados podem provocar desvios nos resultados da etapa das funções de dados. A técnica proposta utiliza as referências a tabelas nos arquivos de código-fonte do sistema. Com isso, caso o sistema utilize *web services*, *stored procedures*, *views*, mapeamento objeto-relacional ou outros tipos de camadas de abstração que retirem a necessidade de manter referências a tabelas nos arquivos de código-fonte, não são geradas essas ligações de rastreabilidade;
- Para um sistema que possua o acesso a dados em uma camada isolada das demais (como regras de negócio e interface com o usuário), os arquivos de código-fonte que não estão na camada de acesso a dados não são rastreados para as funções de dados;

- Dados manipulados em integrações entre sistemas não são mapeados por ligações de rastreabilidade. Isso ocorre porque integrações entre sistemas geralmente utilizam camadas de abstração. Assim, os Arquivos de Interface Externa não são mapeados, à exceção de sistemas que façam integração por meio de *shared database*, onde um sistema tem acesso direto ao banco de dados de outro sistema, uma prática não recomendada;
- A qualidade das ligações de rastreabilidade obtidas como resultado da aplicação da técnica em um sistema depende diretamente da qualidade das entradas fornecidas manualmente pela equipe;
- A dependência que a técnica proposta possui de informações de entrada manuais, que podem ser oriundas de um processo de análise e desenvolvimento (as funções de dados e transacionais, as ligações de rastreabilidade de entrada e os casos de teste), com esforço que pode ser inviável, em especial no caso de sistemas de larga escala;
- Mudanças nos requisitos ou no entendimento sobre os mesmos exige alterações manuais nos dados de entrada da técnica. Dependendo da natureza da mudança, pode ser necessária uma nova execução da etapa. Por exemplo, caso seja inserida uma nova função de dados, é necessário mapear suas tabelas e realizar uma nova busca de tabelas no código-fonte. Mas caso uma tabela esteja ligada a uma função de dados indevida e isso seja corrigido, as ligações indiretas entre as funções de dados e os arquivos de código-fonte imediatamente determinam os mapeamentos. Caso uma nova função transacional seja produzida, é necessário criar casos de teste, executá-los e monitorá-los. Mas caso uma função transacional esteja relacionada a um caso de teste indevido, as ligações indiretas entre as funções transacionais e os arquivos de código-fonte imediatamente determinam os mapeamentos;
- A técnica recomenda que os testes da interface com o usuário sejam automatizados para evitar o esforço adicional de testes manuais do sistema. Caso não seja possível a total automatização, é necessário que na primeira aplicação da técnica e em toda mudança que provoque impacto em um módulo alterado, os casos de teste deste módulo sejam executados manualmente. É importante frisar que seria uma boa prática realizar esses testes, mesmo em um projeto de desenvolvimento de *software* sem preocupação com rastreabilidade;
- Bibliotecas e códigos reutilizados em diversos pontos do código-fonte de um sistema podem elevar indevidamente a relevância de uma função de dados ou transacional. Por exemplo, caso um método com muitas linhas de código seja

executado nos testes de diversas funções transacionais e outro método com poucas linhas de código seja executado nos testes de uma única função transacional, pode-se concluir que o método longo seja mais relevante para essa função transacional que o método curto, quando na realidade um método usado exclusivamente em uma função transacional poderia ser mais relevante.

5.3. Trabalhos Futuros

Apesar das limitações expostas, a proposta oferece recursos para uma abordagem inicial de solução do problema. Entretanto, ela não possui o objetivo de definir e criar uma técnica que contemple todas as suas nuances, mas apenas abordar seus principais aspectos. Algumas evoluções da técnica proposta podem ser citadas, para ampliar sua maturidade e buscar sua generalização.

- Realizar estudos de caso, criando ligações de rastreabilidade para diversos sistemas aplicando a técnica proposta e outras técnicas de rastreabilidade conhecidas. As métricas *precision* e *recall* devem ser comparadas para os resultados das técnicas utilizadas, avaliando a técnica proposta frente às demais. Essas métricas são amplamente utilizadas na avaliação de técnicas de rastreabilidade. *Precision* determina a exatidão das ligações de rastreabilidade obtidas pela técnica, enquanto *recall* determina quão completas estão estas ligações (ANTONIOL *et al.*, 1999, CLELAND-HUANG *et al.*, 2007).;
- Avaliar se os benefícios da rastreabilidade são realmente alcançados pelo uso da técnica proposta, em especial se as informações de rastreabilidade obtidas facilitam a análise de impacto e, consequentemente, a estimar o esforço de construção de uma solicitação de mudança em um sistema de tenha sido analisado sob a ótica da Análise de Ponto de Função. Avaliar outros benefícios, como a compreensão do código-fonte, a manutenção, a identificação de componentes candidatos a reuso e outros são realmente alcançados. Essas avaliações devem ser feitas em sistemas de diversos portes, suportados pelos mais diversos tipos de equipe;
- Avaliar se a contagem de referências a tabelas e de acionamentos de linhas de código é suficiente para definir os pesos das ligações das funções de dados e transacionais, respectivamente. Verificar se existem outras métricas que podem ser utilizadas, alcançando melhores resultados. Possivelmente, a própria complexidade das funções de dados e transacionais adiciona mais um fator a ser considerado ao calcular o peso de uma ligação;

- Realizar estudos de caso usando outros níveis de granularidade que não sejam o de arquivo. Aplicar a técnica e observar a criação de ligações de rastreabilidade para linhas de código e métodos seria uma forma de entender como ela se comporta ao aumentar o nível de granularidade. Aplicá-la em pacotes e diretórios mostraria o comportamento ao reduzir o nível de granularidade. Fatores como o esforço necessário, benefícios e dificuldades precisam ser expostos e analisados;
- Buscar uma solução para a limitação imposta pelas camadas de abstração, permitindo que sistemas desenvolvidos usando *web services*, *stored procedures*, *views*, mapeamentos objeto-relacionais e outras arquiteturas que evitem referências a tabelas no código-fonte possam ser beneficiadas pela técnica proposta. No cenário atual de desenvolvimento de sistemas, onde se busca reduzir o acoplamento entre as camadas de um software e de integrações entre sistemas, formando soluções maiores, é necessário que a técnica não esteja limitada aos sistemas que não utilizem camadas de abstração;
- Habilitar a técnica proposta a criar ligações de rastreabilidade (na etapa das funções de dados) não apenas para os artefatos de código-fonte que possuam queries de manipulação do banco e dados, mas também para os artefatos de código-fonte que dependam da manipulação feita por outros artefatos (a exemplo de artefatos de regras de negócio e interface com o usuário). Uma sugestão seria o uso de grafos de dependências (CHEN e RAJLICH, 2000) para alcançar, a partir dos artefatos que manipulam o banco de dados, os demais artefatos que deles dependam. Isso deve aumentar substancialmente a quantidade de arquivos rastreados pela técnica, identificando também artefatos responsáveis pelas regras de negócio, interface com o usuário e outros;
- Reduzir a dependência de informações de entrada que podem não existir previamente ou não tenham sido mantidas atualizadas. Pode-se estudar como melhorar a criação das listas de funções de dados e transacionais, a criação de casos de teste e das ligações de rastreabilidade de entrada.
- A avaliação *ad hoc* feita sobre o protótipo não é assertiva ao ponto de indicar que o protótipo oferece os benefícios a que se propõe. Trabalhos futuros devem ser realizados para verificar se esses e os demais benefícios da rastreabilidade podem ser alcançados, principalmente o que essa pesquisa se propõe a oferecer, que é oferecer dados que facilitem mensurar o esforço, tempo e custo necessários a um projeto de manutenção de *software*, baseado na análise de impacto feita com apoio da rastreabilidade.

VI. Referências Bibliográficas

ALBRECHT, A. J., 1979, "Measuring Application Development Productivity". In: *Proceedings of IBM Applications Development Symposium*, pp. 83-92, Monterey, CA, EUA, Out.

ALBRECHT, A. J. e GAFFNEY, J. E., 1983, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", *IEEE Transactions on Software Engineering*, v. 9, n. 6 (Nov), pp. 639-648.

ANTONIOL, G., CANFORA, G., DE LUCIA, A. et al., 1999, "Recovering Code to Documentation Links in OO Systems". In: *Proceedings of the 6th Working Conference on Reverse Engineering*, pp. 136-144, Atlanta, GA, EUA, Out.

ANTONIOL, G., CASAZZA, G. e CIMITILE, A., 2000, "Traceability Recovery by Modeling Programmer Behavior". In: *Proceedings of the 7th Working Conference on Reverse Engineering*, pp. 240-247, Brisbane, Queensland, Austrália, Nov.

ANTONIOL, G., CANFORA, G., CASAZZA, G. et al., 2002, "Recovering Traceability Links between Code and Documentation", *IEEE Transactions on Software Engineering*, v. 28, n. 10 (Out), pp. 970-983.

ARNOLD, R. S. e BOHNER, S. A., 1993, "Impact Analysis - Towards a Framework for Comparison". In: *Proceedings of the Conference on Software Maintenance*, pp. 292-301, Montréal, Quebec, Canada, Set.

BIGGERSTAFF, T. J., MITBANDER, B. G. e WEBSTER, D., 1993, "The Concept Assignment Problem in Program Understanding", In: *Proceedings of the 15th Conference on Software Engineering*, pp. 482-498, Baltimore, Maryland, EUA, Mai.

BOHNER, S. A. e ARNOLD, R. S., 1996, *Software Change Impact Analysis*. Los Alamitos, CA, EUA, IEEE Computer Society Press.

BREITMAN, K. K., 2000, *Evolução de Cenários*. Tese de Doutorado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ, Brasil, Mai.

CERPA, N. e VERNER, J. M., 2009, "Why did your Project Fail?", *Communications of the ACM*, v. 52, n. 12 (Dez), pp. 130-134.

CHEN, K. e RAJLICH, V., 2000, "Case Study of Feature Location using Dependence Graph", In: *Proceedings of the 8th International Workshop on Program Comprehension*, pp. 241-249, Limerick, Ireland, Jun.

CLELAND-HUANG, J. C., SETTIMI, R., ROMANOVA, E. et al., 2007, "Best Practices for Automated Traceability", *Computer*, v. 40, n. 6 (Jun), pp. 27-35.

CRADLE, 2008, "CRADLE, From concept to creation…", *18th Annual International Symposium of Incose, 6th Biennal European Systems Engineering Conference, Tool Vendor Challenge*, RT097-01, Utrecht, Holanda, 15-19 de Junho.

DOD, 1988, *Defense System Software Development*, DOD-STD-2167A, AMSC No. 4327, Department of Defense, Washington, D.C.

DOD, 1994, *Software Development and Documentation*, MIL-STD-498, AMSC No. N7069, Department of Defense, Washington, D.C.

EGYED, A., 2001, "A Scenario Driven Approach to Traceability", In: *Proceedings of the 23rd International Conference on Software Engineering*, pp. 123-132, Toronto, Canada, Mai.

EGYED, A. e GRÜNBACHER, P., 2002, "Automating Requirements Traceability: Beyond the Record & Replay Paradigm", In: *Proceedings of the 17th International Conference on Automated Software Engineering*, pp. 163-171, Edinburgh, Escócia, Set.

EGYED, A., BIFFL, S., HEINDL, M. et al., 2005, "Determining the Cost-Quality Trade-Off for Automated Software Traceability", In: *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, pp. 360-363, Long Beach, CA, EUA, Nov.

ELMASRI, R. e NAVATHE, S. B., 2005, *Sistemas de Banco de Dados*. 4 ed. São Paulo, Pearson Addison-Wesley.

FRAKES, W. B. e KANG, K., 2005, "Software Reuse Research: Status and Future", *IEEE Transactions on Software Engineering*, v. 31, n. 7 (Jul), pp. 529-536.

GARMUS, D. e HERRON, D., 2001, *Function Point Analysis: Measurement Practices for Successful Software Projects*. New Jersey, Pearson Addison-Wesley.

GENVIGIR, E. C., 2009, *Um Modelo para Rastreabilidade de Requisitos de Software Baseado em Generalização de Elos e Atributos*. Tese de Doutorado, Instituto Nacional de Pesquisas Espaciais, São José dos Campos, SP, Brasil, Mar.

GERSTING, J. L., 1995, *Fundamentos Matemáticos para a Ciência da Computação*. 3 ed. Rio de Janeiro, LTC Editora.

GOTEL, O. C. Z. e FINKELSTEIN, A. C. W., 1994, "An Analysis of the Requirements Traceability Problem", In: *Proceedings of the International Conference on Requirements Engineering*, pp. 94-101, Colorado Springs, CO, EUA, Abr.

GRECHANIK, M., MCKINLEY, K. S. e PERRY, D. E., 2007, “Recovering And Using Use-Case-Diagram-To-Source-Code Traceability Links”, In: *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGMOD Symposium on the Foundations of Software Engineering*, pp. 95-104, Dubrovnik, Croácia, Set.

HEINDL, M. e BIFFL, S., 2005, “A Case Study on Value-based Requirements Tracing”, In: *Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 60-69, Lisboa, Portugal, Set.

IEEE, 1984, *IEEE Guide to Software Requirements Specifications*, ANSI/IEEE STD 830-1984, Institute of Electrical and Electronics Engineers, New York, NY, EUA.

IEEE, 1998, *IEEE Recommended Practice to Software Requirements Specifications*, IEEE STD 830-1998, Institute of Electrical and Electronics Engineers, New York, NY, EUA.

IEEE, 2008, *IEEE Standard for Software and System Test Documentation*, IEEE STD 829-2008, Institute of Electrical and Electronics Engineers, New York, NY, EUA.

IFPUG, 2010, *Function Point Counting Practices Manual Release 4.3.1 (Excerpt)*, International Function Point Users Group, New Jersey, EUA.

KANNENBERG, A. e SAIEDIAN, H., 2009, “Why Software Requirements Traceability Remains a Challenge”, *Crosstalk – The Journal of Defense Software Engineering*, v. 22, n. 5 (Jul/Ago), pp. 14-19.

LAVAZZA, L. e VALETO, G., 2000, “Enhancing Requirements and Change Management through Process Modeling and Measurement”, In: *Proceedings of the 4th International Conference on Requirements Engineering*, pp. 106-118, Schaumburg, IL, EUA, Jun.

LEHMAN, M. M. e RAMIL, J. F., 2003, "Software evolution - Background, theory, practice", *Information Processing Letters*, v. 88, n. 1-2 (Out), p. 33-44.

MARCUS, A. e MALETIC, J. I., 2003, “Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing”, In: *Proceedings of the 25th International Conference on Software Engineering*, pp. 125-135, Portland, OR, EUA, Mai.

PINHEIRO, F. A. C. e GOGUEN, J. A., 1996, “An Object-Oriented Tool for Tracing Requirements”, *IEEE Software*, v. 13, n. 2 (Mar), pp.52-64.

RAMESH, B. e DHAR, V., 1992, “Supporting Systems Development by Capturing Deliberations During Requirements Engineering”, *IEEE Transactions on Software Engineering*, v. 18, n. 6 (Jun), pp. 498-510.

RAMESH, B., STUB, C., POWERS, T. et al., 1997, “Requirements Traceability: Theory and Practice”, *Annals of Software Engineering*, v. 3 (Dez), pp. 397-415.

RAMESH, B., 1998, “Factors Influencing Requirements Traceability Practice”, *Communications of the ACM*, v. 41, n. 12 (Dez), pp. 37-44.

RAMESH, B. e JARKE, M., 2001, “Towards Reference Models for Requirements Traceability”, *IEEE Transactions on Software Engineering*, v. 27, n. 1 (Jan), pp. 58-93.

ROCHIMAH, S., KADIR, W. M. N. W. e ABDULLAH, A. H., 2007, “An Evaluation of Traceability Approaches to Support Software Evolution”, In: Proceedings of the International Conference on Software Engineering Advances, pp. 19, Cap Esterel, França, Ago.

SAYÃO, M. e LEITE, J. C. S. P., 2005, *Rastreabilidade de Requisitos*. Monografias em Ciência da Computação, Departamento de Informática, Rio de Janeiro, RJ, Brasil, Mai.

SEI, 2006, *CMMI for Development, Version 1.2*, Software Engineering Institute, Pittsburgh, PA, EUA.

SEI, 2007, *CMMI for Acquisition, Version 1.2*, Software Engineering Institute, Pittsburgh, PA, EUA.

SEI, 2009, *CMMI for Services, Version 1.2*, Software Engineering Institute, Pittsburgh, PA, EUA.

SOFTEX, 2009, *MPS.BR - Melhoria de Processo do Software Brasileiro, Guia Geral*, SOFTEX – Associação para Promoção da Excelência do Software Brasileiro.

SPANOUDAKIS, G., 2002, “Plausible and Adaptive Requirement Traceability Structures”, In: *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, pp. 135-142, Ischia, Itália, Jul.

TAN, H. B. K., ZAO, Y. e KHANG, H. 2009, “Conceptual Data Model-Based Software Size Estimation for Information Systems”, ACM Transactions on Software Engineering and Methodology, v. 19, n. 2, Artigo 4 (Out), pp. 1-37.

TORANZO, M., CASTRO, J. F. B. e MELLO, E. 2002, “Uma Proposta para Melhorar o Rastreamento de Requisitos”, In: *Proceedings of the 5th International Workshop on Requirements Engineering*, pp. 194-209, Valencia, Espanha, Nov.

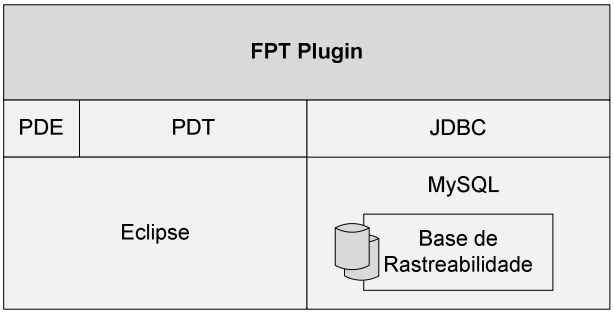
VITECH, 2008, “System Engineering & Architecting with CORE”, INCOSE WMA Chapter Meeting, Abr.

WATKINS, R. e NEAL, M., 1994, “Why and How of Requirements Tracing”, *IEEE Software*, v. 11, n. 4 (Jul), pp. 104-106.

WEISZFLOG, W., 2010, *Michaelis – Moderno Dicionário da Língua Portuguesa*. São Paulo, SP, Melhoramentos.

ANEXO I – ARQUITETURA DO PROTÓTIPO

O protótipo desenvolvido é um *plugin* para o ambiente de desenvolvimento *Eclipse*. A base para o seu desenvolvimento foi o próprio *Eclipse* em conjunto com um *framework* de criação de *plugins* chamado *Plug-in Development Environment (PDE)* [1]. A capacidade de o Eclipse controlar projetos na linguagem *PHP* foi oferecida pelo *framework PHP Development Tools (PDT)*[2]. O acesso ao banco de dados *MySQL* é feito por meio do *Sun Microsystem's JDBC Driver for MySQL (MySQL Connector/J)*[3]. A figura a seguir representa resumidamente a arquitetura do protótipo.



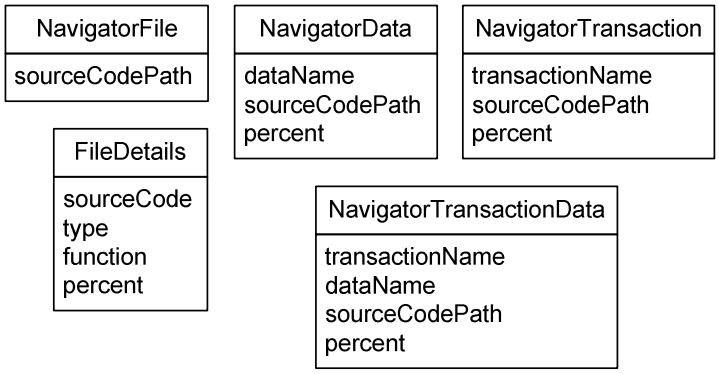
Arquitetura do FPTPlugin

O *plugin* armazena as ligações de rastreabilidade em um banco de dados MySQL com um esquema pré-determinado. Esse esquema é composto de cinco tabelas, sendo quatro utilizadas pela visão *FPT Navigator* e uma pela visão *FPT File Details*, como pode ser percebido pelos seus nomes. Os dados inseridos nessas tabelas são referências aos elementos rastreados (funções de dados, funções transacionais e arquivos de código-fonte), ligações e percentuais (calculados com base no índice de relevância).

Por uma questão de desempenho, o projeto do esquema de banco de dados não utilizou as representações propostas pelos meta-modelos apresentados na Seção 2.5.1 da Dissertação, que exigiriam agregações e cálculos em tempo de execução, devido à existência de junções relacionais (*joins*). Por esse motivo, o esquema de banco de dados passou por um processo (informalmente chamado) de "desnormalização", tornando o *plugin* capaz de fazer consultas com menor quantidade de agregações e cálculos em tempo de execução.

[1] http://www.eclipse.org/pde/
[2] http://www.eclipse.org/pdt/
[3] http://www.mysql.com/downloads/connector/j/

O esquema utilizado no protótipo será dificilmente aproveitado para outros fins que não esse protótipo, pois os dados são armazenados com uma estrutura muito próxima à das informações oferecidas pelo *plugin*. Para uso das informações de rastreabilidade geradas pela técnica proposta com outros fins, é necessário armazenar os dados conforme determinado nas Seções 3.2.2 e 3.2.3. É importante ressaltar que nos estudos de caso realizados, as informações de rastreabilidade foram armazenadas conforme recomendado pela técnica proposta. Para uso no protótipo, essas informações foram pré-processadas e transformadas para se adequarem ao esquema esperado pelo protótipo, que é representado a seguir.



Esquema de banco de dados do FPT Plugin

Perceba que a falta de normalização e de relacionamentos dificulta o entendimento estruturado de quais dados devem ser inseridos no banco de dados do protótipo. Esse entendimento é feito *ad hoc*, apoiado apenas pelo conhecimento pessoal, e os nomes das tabelas e seus campos. Para auxiliar o entendimento detalhado de cada uma das tabelas e campos no esquema de banco de dados utilizado pelo protótipo, seu dicionário de dados é apresentado a seguir, contendo o nome da tabela, seu objetivo, sua aplicação (como e em qual visão é utilizada), os campos que devem ter dados inseridos e seus significados.

Dicionário de dados do FPT Plugin

NavigatorFile		
Objetivo	Indicar os arquivos que foram rastreados	
Aplicação	Lista de arquivos na opção Traced Files dentro da visão FPT Navigator	
Campos	sourceCodePath	Caminho do arquivo relativo à home do projeto

NavigatorData		
Objetivo	Indicar os arquivos aos quais uma função de dados está relacionada	
Aplicação	Lista de funções de dados e seus arquivos correspondentes na opção Data Functions dentro da visão FPT Navigator	
Campos	dataName	Nome da função de dados
	sourceCodePath	Caminho do arquivo relativo à home do projeto
	Percent	Percentual baseado no peso da ligação entre a função de dados e o arquivo

NavigatorTransaction		
Objetivo	Indicar os arquivos aos quais uma função transacional está relacionada	
Aplicação	Lista de funções transacionais e seus arquivos correspondentes na opção Transactional Functions dentro da visão FPT Navigator	
Campos	transactionName	Nome da função transacional
	sourceCodePath	Caminho do arquivo relativo à home do projeto
	percent	Percentual baseado no peso da ligação entre a função transacional e o arquivo

FileDetails		
Objetivo	Indicar as funções de dados e transacionais a que um arquivo está relacionado	
Aplicação	Lista de funções de dados e transacionais relacionadas a um arquivo aberto, exibida dentro da visão FPT File Details	
Campos	sourceCode	Caminho do arquivo relativo à home do projeto
	type	Classificação da função de dados ou transacional (ALI, AIE, EE, SE, CE)
	function	Nome da função de dados ou transacional
	percent	Percentual baseado no peso da ligação entre a função e o arquivo

NavigatorTransactionData		
Objetivo	Indicar os arquivos aos quais uma função de dados e uma função transacional estão simultaneamente relacionados	
Aplicação	Lista de funções de dados e transacionais e arquivos na opção All Files inserida em cada função de dados e transacional exibida na visão FPT Navigator	
Campos	transactionName	Nome da função transacional
	dataName	Nome da função de dados
	sourceCodePath	Caminho do arquivo relativo à home do projeto
	percent	Percentual baseado na multiplicação (cruzamento) do índice de relevância da ligação entre a função de dados e o arquivo pelo índice de relevância da ligação entre a função transacional e o arquivo.