WILSON BISSI

WS-TDD - UMA ABORDAGEM ÁGIL PARA O DESENVOLVIMENTO DE SERVIÇOS WEB

Dissertação submetida ao Programa de Pós-Graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná como requisito parcial para a obtenção do título de Mestre em Computação Aplicada.

Curitiba PR
Março de 2016

WILSON BISSI

WS-TDD - UMA ABORDAGEM ÁGIL PARA O DESENVOLVIMENTO DE SERVIÇOS WEB

Dissertação submetida ao Programa de Pós-Graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná como requisito parcial para a obtenção do título de Mestre em Computação Aplicada.

Área de concentração: *Engenharia de Sistemas Computacionais*

Orientador: Prof. Dr. Adolfo Gustavo S. S. Neto
Coorientadora: Prof.ª Dr.ª Maria Cláudia F. P. Emer

Curitiba PR
Março de 2016

Dados Internacionais de Catalogação na Publicação

B623w 2016

Bissi, Wilson
WS-TDD : uma abordagem ágil para o desenvolvimento de serviços web / Wilson Bissi.-- 2016.
xxiv, 126 p.: il.; 30 cm

Texto em português, com resumo em inglês.
Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Computação Aplicada, Curitiba, 2016.
Bibliografia: p. 71-84.

1. Software - Testes. 2. Serviços da web - Desenvolvimento. 3. Arquitetura orientada a serviços (Computador). 4. Desenvolvimento ágil de software. 5. Medição de software. 6. Software - Controle de qualidade. 7. Métodos de simulação. 8. Computação - Dissertações. I. Seca Neto, Adolfo Gustavo Serra, orient. II. Emer, Maria Cláudia Figueiredo Pereira, coorient. III. Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Computação Aplicada. IV. Título.

CDD: Ed. 22 -- 621.39

Biblioteca Central da UTFPR, Câmpus Curitiba

ATA DE DEFESA DE DISSERTAÇÃO DE MESTRADO Nº 40

Aos 23 dias do mês de março de 2016 realizou-se na sala B-204 a sessão pública de Defesa da Dissertação de Mestrado intitulada "WS-TDD - Uma Abordagem Ágil Para O Desenvolvimento De Serviços Web", apresentada pelo aluno **Wilson Bissi** como requisito parcial para a obtenção do título de Mestre em Computação Aplicada, na área de concentração “Engenharia de Sistemas Computacionais”, linha de pesquisa “Engenharia de Software”.

Constituição da Banca Examinadora:

Prof. Dr. Adolfo Gustavo Serra Seca Neto, UTFPR - CT (Presidente) _____

Prof. Dr. Marco Aurélio Wehrmeister, UTFPR- CT _____

Profª. Drª. Rafaela Mantovani Fontana, UFPR _____

Em conformidade com os regulamentos do Programa de Pós-Graduação em Computação aplicada e da Universidade Tecnológica Federal do Paraná, o trabalho apresentado foi considerado _____ (aprovado/reprovado) pela banca examinadora. No caso de aprovação, a mesma está condicionada ao cumprimento integral das exigências da banca examinadora, registradas no verso desta ata, da entrega da versão final da dissertação em conformidade com as normas da UTFPR e da entrega da documentação necessária à elaboração do diploma, em até _____ dias desta data.

Ciente (assinatura do aluno): _____

(para uso da coordenação)

A Coordenação do PPGCA/UTFPR declara que foram cumpridos todos os requisitos exigidos pelo programa para a obtenção do título de Mestre.

Curitiba PR, _____/_____/_____ _____

"A Ata de Defesa original está arquivada na Secretaria do PPGCA".

Dedico este trabalho a Deus por ter me dado força, sabedoria e paciência para superar todas as etapas em busca desta conquista pessoal. Aos meus pais Irço e Tereza, à minha esposa Franciéli, exemplo de vida e de um amor sem limites. E a todos que desenvolvem software zelando pela qualidade e melhoria contínua.

Agradecimentos

A Deus pela oportunidade e por ter me dado as habilidades necessárias para realizar este trabalho.

Aos meus pais, Irço Bissi (*in memoriam*) e Tereza M. Bissi que sempre me apoiaram e ensinaram os principais valores éticos e de caráter, sendo os maiores responsáveis pela minha trajetória acadêmica, profissional e familiar. Amo vocês!

À minha esposa Franciéli Vieira, pelo incentivo, apoio e compreensão durante o período do mestrado, onde tive que abdicar de vários momentos juntos para me dedicar aos estudos. Ela que sempre ouviu atentamente as ideias e definições iniciais do projeto, mesmo sem entender profundamente do que se tratava este estudo. Porém, seu otimismo e confiança me ajudaram a chegar até aqui, Obrigado Fram. À Rita de Cássia que me apoiou e incentivou durante todo esse período.

Ao meu orientador Prof. Dr. Adolfo Gustavo Serra Seca Neto e a minha coorientadora Prof.ª Dr.ª Maria Cláudia Figueiredo Pereira Emer pela confiança, dedicação e pelos ensinamentos que contribuíram para o amadurecimento das minhas ideias.

Ao Prof. Dr. Laudelino Cordeiro Bastos e ao Prof. Dr. Marco Aurélio Wehrmeister por participarem dos seminários de qualificação, pelas revisões e pelos direcionamentos realizados. A Prof.ª Dr.ª Rafaela Mantovani Fontana pelas contribuições realizadas na dissertação.

Agradeço também as empresas que cederam espaço para a realização dos experimentos. Aos voluntários que responderam os questionários e aos que participaram dos experimentos. Aos colegas de trabalho que contribuíram para a realização deste projeto, em especial a Luiz Mattos, Samuel Augusto, Sandro Simas, Tiago Manosso. Sem a colaboração de todos vocês este estudo não seria possível.

Por fim, agradeço a UTFPR pelo mestrado que estou concluindo e a todos os professores e colaboradores do DAINF pela troca de experiência e pelo suporte acadêmico fornecido.

Tudo o que é seu encontrará uma
maneira de chegar até você.
(Chico Xavier)

Resumo

Test Driven Development (TDD) é uma prática ágil que ganhou popularidade ao ser definida como parte fundamental na *eXtreme Programming* (XP). Essa prática determina que os testes devem ser escritos antes da implementação do código. TDD e seus efeitos têm sido amplamente estudados e comparados com a prática *Test Last Development* (TLD) em diversos trabalhos. Entretanto, poucos estudos abordam TDD no desenvolvimento de *Web Services* (WS), devido à complexidade em testar as dependências entre os componentes distribuídos e as particularidades da *Service Oriented Architecture* (SOA). Este trabalho tem por objetivo definir e validar uma abordagem para o desenvolvimento de WS baseada na prática de TDD, denominada WS-TDD. Essa abordagem guia os desenvolvedores no uso de TDD durante o desenvolvimento de WS, sugerindo ferramentas e técnicas para lidar com as dependências e as particularidades de SOA, com foco na criação dos testes unitários e integrados automatizados na linguagem Java. No intuito de definir e validar a abordagem proposta, quatro métodos de pesquisa foram executados: (i) questionário presencial; (ii) experimento; (iii) entrevista presencial com cada participante do experimento e (iv) triangulação dos resultados com as pessoas que participaram nos três métodos anteriores. De acordo com os resultados obtidos, a WS-TDD mostrou-se mais eficiente quando comparada a TLD, aumentando a qualidade interna do software e a produtividade dos desenvolvedores. No entanto, a qualidade externa do software diminuiu, apresentando um maior número de defeitos quando comparada a TLD. Por fim, é importante destacar que a abordagem proposta surge como uma alternativa simples e prática para a adoção de TDD no desenvolvimento de WS, trazendo benefícios a qualidade interna e contribuindo para aumentar a produtividade dos desenvolvedores. Porém, a qualidade externa do software diminuiu ao utilizar a WS-TDD.

Palavras-chave: TDD, Práticas Ágeis, Serviços Web, SOA, Qualidade de Software.

Abstract

Test Driven Development (TDD) is an agile practice that gained popularity when defined as a fundamental part in eXtreme Programming (XP). This practice determines that the tests should be written before implementing the code. TDD and its effects have been widely studied and compared with the Test Last Development (TLD) in several studies. However, few studies address TDD practice in the development of Web Services (WS), due to the complexity of testing the dependencies among distributed components and the specific characteristics of Service Oriented Architecture (SOA). This study aims to define and validate an approach to develop WS based on the practice of TDD, called WS-TDD. This approach guides developers to use TDD to develop WS, suggesting tools and techniques to deal with SOA particularities and dependencies, focusing on the creation of the unitary and integrated automated tests in Java. In order to define and validate the proposed approach, four research methods have been carried out: (i) questionnaire; (ii) practical experiment; (iii) personal interview with each participant in the experiment and (iv) triangulation of the results with the people who participated in the three previous methods. According to the obtained results, WS-TDD was more efficient compared to TLD, increasing internal software quality and developer productivity. However, the external software quality has decreased due to a greater number of defects compared to the TLD approach. Finally, it is important to highlight that the proposed approach is a simple and practical alternative for the adoption of TDD in the development of WS, bringing benefits to internal quality and contributing to increase the developers' productivity. However, the external software quality has decreased when using WS-TDD.

Keywords: TDD, Agile Practices, Web Services, SOA, Software Quality.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

ACM	*Association for Computing Machinery*
BPMN	*Business Process Modeling Notation*
BPEL	*Business Process Execution Language*
CFG	*Control Flow Graph*
CORBA	*Common Object Request Broker Architecture*
EJB	*Enterprise JavaBeans*
HTTP	*Hyper Text Transfer Protocol*
IEC	*International Engineering Consortium*
IEEE	*Institute of Electrical and Electronics Engineers*
ISO	*International Organization for Standardization*
LoC	*Lines of Code*
RFC	*Response for a Class*
NBR	Norma Brasileira
PPGCA	Programa de Pós-Graduação em Computação Aplicada
REST	*Representational State Transfer*
SOA	*Service Oriented Architecture*
SOAP	*Simple Object Access Protocol*
SWEBOK	*Software Engineering Body of Knowledge*
TDD	*Test Driven Development*
TLD	*Test Last Development*
UDDI	*Universal Description, Discovery and Integration*
UML	*Unified Modeling Language*
VCS	*Version Control System*
W3C	*World Wide Web Consortium*
WS	*Web Services*
WS-TDD	*Web Service-Test Driven Development*
WSDL	*Web Services Description Language*
XML	*eXtensible Markup Language*
XP	*Extreme Programming*

Capítulo 1

Introdução

Desenvolvimento Guiado por Testes (TDD - *Test Driven Development*) é uma prática de desenvolvimento de software em que os testes de unidades automatizados são escritos de forma incremental antes da implementação do código fonte (BECK, 2002). O desenvolvedor só deve escrever código relacionado a uma funcionalidade se este for coberto anteriormente por um teste daquela funcionalidade. Por exemplo, antes de escrever o código de um método que soma dois números, o desenvolvedor deve escrever um teste de unidade que especifica que o resultado da soma de 2 com 3 é 5.

TDD ganhou visibilidade ao ser definida como parte fundamental do processo de desenvolvimento denominado Programação Extrema (XP - *eXtreme Programming*) em Beck (1999). Porém, atualmente esta prática é usada na indústria de software de maneira independente de XP (PANCUR et al., 2003).

De acordo com o relatório anual conduzido pela VersionOne (2015), baseado em um questionário *online*, cerca de 34% dos entrevistados afirmaram utilizar a prática de TDD durante o desenvolvimento. Um estudo semelhante feito no Brasil por Melo et al. (2013), apontou que 39,5% dos entrevistados utilizam TDD.

Apesar deste percentual ser expressivo, TDD ainda é menos utilizado que outras práticas ágeis, como testes unitários, integração contínua e padrões de código. Os estudos apresentados em VersionOne (2015) e em Melo et al. (2013) foram realizados com pessoas que utilizam ou possuem interesse de utilizar algum método ou prática ágil, ou seja, esse percentual não pode ser generalizado para todos os desenvolvedores de software.

Diversos trabalhos têm estudado os efeitos que a prática de TDD produz no desenvolvimento de software dentro do cenário acadêmico (universidades) ou na própria indústria (empresas), comparando a prática de TDD com o *Test Last Development* (TLD). Porém, a maioria desses trabalhos falham ao apresentar resultados conclusivos no que diz respeito à produtividade e à qualidade do software desenvolvido (RAFIQUE; MISIC, 2013).

Com relação a arquitetura de software flexível e reutilizável, a Arquitetura Orientada a Serviços (SOA - *Service Oriented Architecture*) tornou-se a principal referência para o desenvolvimento de aplicações distribuídas por meio da integração de serviços disponíveis na web (SHARMA; HELLMANN; MAURER, 2012). Segundo Ribarov et al. (2007), SOA tem se destacado por oferecer um padrão arquitetural voltado para reduzir os custos despendidos na tentativa de uma integração rápida e flexível dos sistemas de uma organização.

Para Zambiasi (2012), o que motiva a adoção de SOA é a necessidade de aumentar a agilidade na implementação de novos produtos e processos e na composição de novas aplicações

por meio da orquestração de componentes já existentes, além de permitir o reúso de sistemas e aplicações legadas.

Os componentes que constituem a arquitetura SOA são chamados de serviços. Os serviços são entidades autônomas e independentes de plataformas que podem ser descritos, publicados, descobertos e montados dinamicamente para o desenvolvimento rápido, de baixo custo, interoperáveis e reutilizáveis (PAPAZOGLOU et al., 2007). A implementação de um serviço para integração de aplicações é feita usando o conceito de Serviço Web (WS - *Web Service*).

De acordo com Barbir et al. (2007), WS tornou-se uma tecnologia-chave para permitir o desenvolvimento de componentes com baixo acoplamento, linguagem neutra e independente de plataforma para conectar aplicações em diversas organizações.

Bai et al. (2006) afirmam que SOA traz novas preocupações e desafios para testar aplicações baseadas em serviços pelos seguintes motivos: (1) Flexibilidade de configuração do sistema; (2) Interoperabilidade entre componentes desenvolvidos por terceiros; (3) Detecção de falhas em tempo de execução; (4) Avaliação da confiabilidade; (5) Falta de transparência na implementação.

Se por um lado os serviços web oferecem muitos benefícios, como a garantia de interoperabilidade, modularidade e reutilização (W3C, 2015), por outro eles introduzem uma série de desafios e preocupações para os pesquisadores, desenvolvedores e testadores de software. Bozkurt, Harman e Hassoun (2010) afirmam que testar serviços web é mais difícil se comparado com os sistemas tradicionais por duas razões principais: (1) a natureza complexa dos WS e (2) as limitações que ocorrem devido a natureza de SOA.

O desenvolvedor encontra inúmeras limitações ao elaborar os testes unitários ou integrados dos serviços web como, por exemplo, a falta de acesso ao código fonte dos componentes do qual o WS depende (muitas vezes são provenientes de terceiros), a falta de informação sobre os componentes, a dependência dos servidores de aplicação e a falta de visibilidade do impacto que a alteração poderá causar em outros módulos do software. Essas limitações tornam a adoção de TDD no desenvolvimento de WS em SOA um desafio devido à complexidade do ambiente.

Segundo Koltai et al. (2011), pela perspectiva do desenvolvedor, elaborar testes unitários ou introduzir a prática de TDD durante o desenvolvimento, torna-se muito mais desafiador neste cenário em que as aplicações são implementadas por meio de um sistema de computação distribuída.

1.1 Motivação

As limitações percebidas pelo autor ao tentar adotar a prática de TDD no desenvolvimento de serviços web motivaram esta pesquisa. As principais limitações identificadas foram:

- Como lidar com a dependência de ambientes complexos para realizar o teste unitário e integrado de serviços web;
- Falta de acesso ao código fonte dos componentes externos, no caso de composição entre serviços web;
- Faltam experimentos sobre teste de WS no ambiente corporativo;

- Faltam orientações para lidar com a falta de informação (dependências externas) dos serviços web durante a implementação;
- Faltam guias para orientar os desenvolvedores no uso de ferramentas para facilitar a adoção de TDD no desenvolvimento de WS.

A literatura atual confirma essas limitações enfatizando que o surgimento do paradigma de desenvolvimento orientado a serviços exige novas abordagens para verificação e validação que considerem as características deste paradigma. Na literatura, não foi encontrado nenhum experimento sobre o uso de TDD em ambientes distribuídos, apenas com sistemas centralizados. Além disso, a quantidade de experimentos que utilizam profissionais da indústria em seus estudos é menor do que os aplicados no ambiente acadêmico.

A crescente popularidade da prática de TDD, tanto na academia quanto na indústria, e os benefícios da prática de TDD mencionados nos livros, porém pouco explorados, também formam a motivação inicial do trabalho.

A maioria dos estudos acadêmicos que fazem a comparação entre TDD e TLD baseia-se na construção de novas aplicações em plataformas centralizadas. Existem poucos estudos que fazem essa comparação entre as abordagens ou tentam viabilizar a prática de TDD em plataformas distribuídas.

A revisão sistemática da literatura realizada neste trabalho não identificou nenhum estudo ou abordagem que direcione o desenvolvedor na implementação de serviços web, o trabalho que mais se aproximou a este tema foi o de Besson et al. (2015), focado na coreografia de serviços web. Portanto, a novidade que a WS-TDD traz neste estudo é a aplicação de TDD focado no desenvolvimento de serviços web na arquitetura orientada a serviços.

1.2 Objetivos

A prática de TDD tem sido utilizada no desenvolvimento de aplicações centralizadas e são raros os estudos que abordam essa prática em outros paradigmas de desenvolvimento. A possibilidade de usar TDD no desenvolvimento de WS direcionou os objetivos definidos a seguir.

1.2.1 Objetivo Geral

O objetivo geral desta dissertação é definir e avaliar uma abordagem para o desenvolvimento de serviços web baseada na prática de TDD e direcionada para a arquitetura orientada a serviços.

1.2.2 Objetivos Específicos

Este trabalho é composto pelos seguintes objetivos específicos:

- Obter a percepção dos desenvolvedores sobre o uso da prática de TDD no desenvolvimento de software em geral;
- Combinar o fluxo de TDD com técnicas do desenvolvimento de serviços web para atender as necessidades específicas desta tecnologia;

- Especificar uma abordagem (WS-TDD) voltada para a adoção do TDD no desenvolvimento de WS que utilize as ferramentas e técnicas existentes;
- Sugerir alternativas para lidar com a falta de informação dos serviços web e de seus componentes externos na fase de desenvolvimento e de teste (unitário e integrado);
- Avaliar os efeitos que a WS-TDD gera na produtividade, na qualidade interna e na qualidade externa de sistemas baseados em WS;
- Aumentar a qualidade interna e a qualidade externa do software baseado em WS a partir do uso da WS-TDD, se comparado com o TLD;
- Conduzir estudos no ambiente corporativo para validar e aperfeiçoar a WS-TDD comparando-a com a TLD;
- Levantar as percepções dos desenvolvedores quanto as vantagens e desvantagens na utilização da WS-TDD.

1.3 Contribuições

As principais contribuições deste trabalho são:

- Apresentação de uma revisão sistemática da literatura sobre estudos que comparam TDD com TLD;
- Mapeamento da percepção dos desenvolvedores sobre TDD em empresas locais;
- Criação de uma abordagem para aplicação de TDD no desenvolvimento de WS, denominada WS-TDD;
- Análise dos experimentos realizados para avaliar a WS-TDD no ambiente corporativo;
- Documentação para os desenvolvedores interessados adotarem e avaliarem a abordagem proposta;
- Artigo intitulado "*The Effects of Test Driven Development on Internal Quality, External Quality and Productivity: A systematic review*" publicado na revista *Information and Software Technology* (v. 74, p. 45 - 54, 2016).

A abordagem WS-TDD, criada neste trabalho, se propõe a guiar os desenvolvedores na implementação serviços web, oferecendo ferramentas e práticas que os auxiliem durante o desenvolvimento usando TDD. A WS-TDD oferece suporte ao desenvolvedor para transpor as limitações e viabilizar a adoção de TDD no desenvolvimento de WS.

Além disso, este trabalho contribui para diminuir a carência de experimentos no mundo real e apresenta soluções que melhoram a capacidade de teste em sistemas orientados a serviços. Essas necessidades são apresentadas em Bozkurt, Harman e Hassoun (2013) como itens que ainda carecem de novas investigações.

1.4 Organização da Dissertação

Este trabalho está dividido da seguinte forma:

- O Capítulo 2 detalha os principais conceitos utilizados e discute alguns estudos sobre TDD;
- O Capítulo 3 descreve o método de pesquisa utilizado para o desenvolvimento do trabalho e as formas de coleta e análise dos dados das atividades de pesquisa executadas;
- O Capítulo 4 apresenta a WS-TDD, abordagem criada neste trabalho e utilizada nos experimentos e nas entrevistas;
- O Capítulo 5 mostra os resultados obtidos nos questionários, nos experimentos, nas entrevistas e na triangulação desses resultados;
- O Capítulo 6 discute os resultados obtidos e as ameaças a validade deste trabalho;
- O Capítulo 7 conclui o trabalho realizado e apresenta algumas oportunidades de trabalhos futuros relacionado ao tema.

Adicionalmente ao corpo principal do trabalho, os apêndices estão distribuídos da seguinte forma:

- O Apêndice A mostra o método usado na revisão sistemática da literatura;
- O Apêndice B faz uma análise detalhada dos principais artigos relacionados ao tema deste trabalho;
- O Apêndice C apresenta o questionário usado na pesquisa sobre TDD nas três empresas de Curitiba;
- O Apêndice D apresenta o formulário usado na entrevista com os profissionais que participaram do experimento;
- O Apêndice E detalha todos os requisitos solicitados na implementação dos dois exercícios usados no experimento;
- O Apêndice F descreve com mais detalhes os resultados obtidos com o questionário de pesquisa sobre TDD.
- O Apêndice G apresenta um exemplo do formulário de entrevista respondido por um dos participantes.

Capítulo 2

Fundamentação Teórica

2.1 Métodos Ágeis

O desenvolvimento ágil de software é um paradigma que fornece um método para organizar o desenvolvimento de software complexo com vários participantes, acomodando as mudanças constantes do projeto (STRODE et al., 2012).

O manifesto para o desenvolvimento ágil de software[1] foi publicado com o objetivo de concentrar os valores e os princípios que fundamentam o desenvolvimento ágil de software.

O manifesto ágil valoriza indivíduos e interação, software em funcionamento, colaboração com o cliente e respostas as mudanças (BECK et al., 2001). Esses são os quatro valores que se espera de qualquer método ágil.

Entende-se que as mudanças de requisitos são bem-vindas, mesmo em estágios finais de desenvolvimento, porque o software é construído de forma incremental (LEFFINGWELL, 2011). Os métodos ágeis ajudam e até mesmo incentivam as mudanças de requisitos durante o projeto de software; essa é uma característica forte presente nesses métodos.

As principais razões que motivam a adoção de métodos ágeis nas empresas brasileiras, segundo Melo et al. (2013) são: Aumento de produtividade (91%), Gerenciamento da mudança de prioridade (86%) e Aumento da qualidade de software (83%).

Atualmente, há pelo menos treze métodos ágeis de desenvolvimento de software diferentes. De acordo com Dyba, Dingsoyr e Hanssen (2007), os métodos mais populares são Extreme Programming (XP) e Scrum. Segundo o relatório anual da *Version One* sobre Métodos Ágeis, Scrum, Kanban, XP e suas variações (Scrum/XP Híbrido, Scrumban) estão entre as metodologias mais populares, representando 78% da amostra pesquisada (VERSIONONE, 2015).

Uma pesquisa realizada no Brasil, similar a *Annual State of Agile Report* da *Version One*, revelou que 60,8% das empresas adotam métodos ágeis em mais da metade dos seus projetos. Desse total, os métodos ágeis Scrum, Kanban, XP e suas variações representam 84,9% do métodos utilizados (MELO et al., 2013). Os resultados apresentados em Melo et al. (2013) e VersionOne (2015) não podem ser generalizados pois o foco principal dessas pesquisas foram pessoas que utilizam ou possuem interesse em utilizar os métodos ágeis.

Cada método de desenvolvimento ágil procura especializar-se em determinados pontos do ciclo de vida do software. Por exemplo, o XP oferece diversas práticas voltadas para o desenvolvimento de software (BECK, 1999), enquanto o Scrum se especializa em práticas

[1] http://www.agilemanifesto.org/

voltadas para a comunicação e gerenciamento do projeto de software (PIKKARAINEN et al., 2008).

Uma das práticas que o método XP incorporou em sua essência foi a criação e execução do teste unitário automatizado antes do desenvolvimento da funcionalidade. Essa prática ficou conhecida como TDD, e ganhou mais visibilidade ao ser incluída no XP (BECK, 1999). Porém, atualmente esta prática é usada na indústria de software de maneira independente ao XP (PANCUR et al., 2003).

2.1.1 Desenvolvimento Guiado por Testes

TDD é uma prática para a construção de software, tendo como ideia central a criação de software na forma incremental, por meio de testes para guiar o desenvolvimento (KOLTAI et al., 2011). Os desenvolvedores primeiramente identificam as características e escrevem os testes correspondentes para expressar a funcionalidade desejada. Somente depois que o teste for escrito é que a funcionalidade deve ser implementada.

Beck (2002) define TDD como um conjunto de técnicas que incentivam o desenvolvimento de projetos simples e a elaboração de um conjunto de testes. Apesar do foco em criar o teste unitário automatizado, TDD não é exatamente uma técnica de testes (BECK, 2001). Ela deve ser considerada como uma técnica de projeto (*design*) de software (DAMM; LUNDBERG; OLSSON, 2005).

Ao praticar TDD, os testes de unidade são usados para guiar o projeto de código a ser desenvolvido (KIM; PARK; WU, 2006). As duas regras básicas da TDD são: (1) Escreva um novo código somente depois que um teste automatizado falhou, e (2) eliminar a duplicação refatorando o código (LUZ; NETO; NORONHA, 2013).

TDD é conhecida pelo ciclo “vermelho-verde-refatore” (BECK, 2002), que consiste nos seguintes passos:

1. Projetar e adicionar um teste;
2. Executar todos os testes e verificar a nova falha relativa ao teste adicionado no passo 1 (vermelho);
3. Adicionar código novo que seja suficiente para satisfazer o novo teste;
4. Executar todos os testes, repetir o passo 3 se necessário, até que todos os testes passem (verde);
5. Refatorar para melhorar a estrutura do código ou dos testes;
6. Executar todos os testes após a refatoração para garantir que todos os testes estão passando.

Como observado no ciclo “vermelho-verde-refatore”, antes de implementar uma nova funcionalidade, um teste de unidade deve ser escrito e só depois que ele falhar, o código da funcionalidade deve ser desenvolvido. No final deste ciclo, o desenvolvedor deve refatorar o código e os testes antes de iniciar o desenvolvimento da próxima funcionalidade. Essa refatoração é necessária para melhorar as estruturas internas do software.

A comparação entre TDD e TLD serve de base para muitos estudos atualmente. A Figura 2.1 apresenta o fluxo de desenvolvimento executado em cada uma das práticas, esclarecendo as diferenças de cada fluxo.

Os fluxos apresentados na Figura 2.1 são referentes a implementação de uma nova funcionalidade, denominada N. A principal diferença que pode-se observar entre TDD e TLD na Figura 2.1 é que no fluxo de TDD, a segunda etapa é escrever o teste e em seguida executá-lo; caso ele falhe, o código fonte da funcionalidade é escrito para passar no teste e se necessário, o código é refatorado. Enquanto que no fluxo TLD, a segunda etapa é usada para escrever o código fonte da funcionalidade e em seguida o teste é escrito e executado.

No TLD, os desenvolvedores podem escrever os testes iterativamente após a conclusão do código da funcionalidade ou optar por escrever todos os testes no final da implementação de todo o sistema. Isto depende do modelo de processo de desenvolvimento especificado para o desenvolvimento do software. Porém, em TLD um teste de unidade deve ser escrito somente após o código da funcionalidade ter sido finalizado.

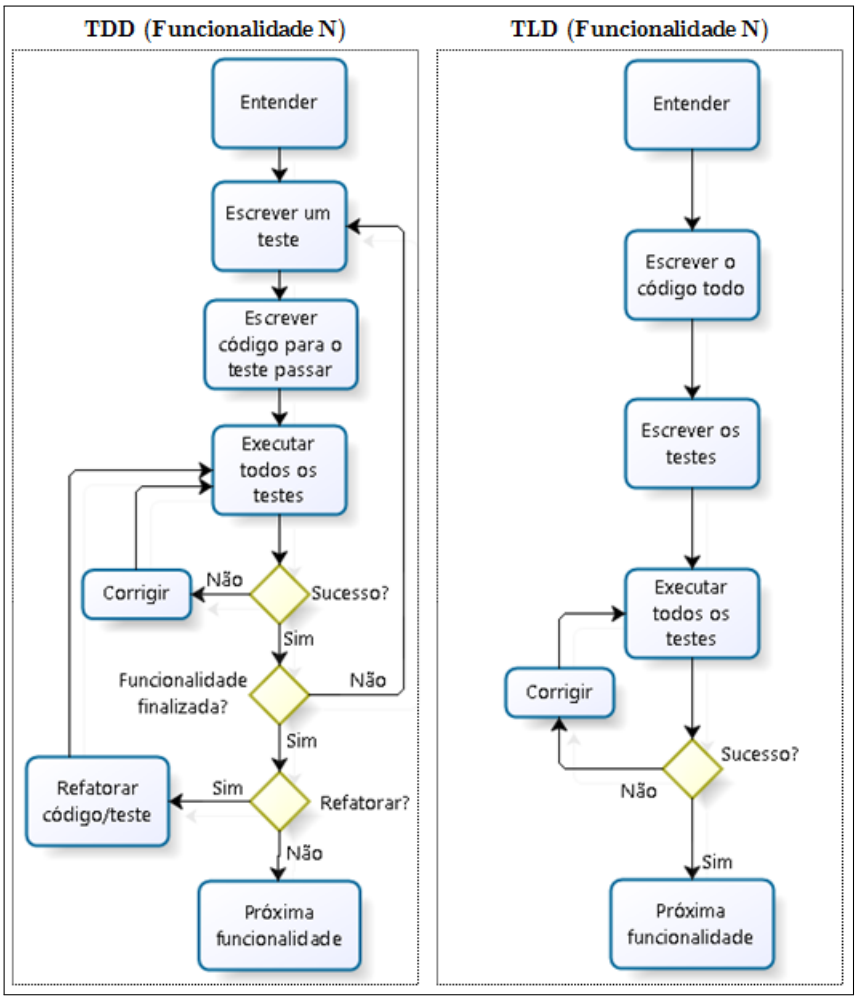

Figura 2.1: Comparação entre TDD e TLD, adaptado de Pancur e Ciglaric (2011).

2.2 Arquitetura Orientada a Serviços

A arquitetura orientada a serviços foi concebida a partir dos conceito do desenvolvimento baseado em componentes. Essa arquitetura e conceito visam atingir os mesmos objetivos que são: atingir um alto nível de reúso, independência de tecnologias e diminuição do esforço no desenvolvimento de software (HEINEMAN; COUNCILL, 2001), (STAA, 2000) e (KRUEGER, 1992).

A reutilização de software é o processo de criação de sistemas a partir de componentes de software existentes (KRUEGER, 1992). A reutilização de componentes pode aumentar a produtividade, diminuir o tempo necessário para o desenvolvimento e apresentar maior demanda por componentes de qualidade (LI; WANG; ZHANG, 2008).

SOA e suas implementações de serviços web promovem uma arquitetura baseada em padrões abertos e de baixo acoplamento para a integração de aplicativos em um ambiente heterogêneo distribuído (W3C, 2015).

Soluções que utilizam SOA destinam-se a satisfazer os objetivos de negócio adicionando requisitos de qualidade como, por exemplo: a integração fácil e flexível com sistemas legados (interoperabilidade); processos de negócios otimizados (manutenibilidade); custos reduzidos (modificabilidade); e agilidade para lidar com mudanças rápidas de processos de negócios (extensibilidade) (BIANCO et al., 2011). Para Gu e Lago (2008), o desenvolvimento orientado a serviços tem se tornado gradualmente mais popular e dominante na engenharia de software moderna.

As organizações criam suas funcionalidades com base nos requisitos de interoperabilidade, manutenibilidade, modificabilidade e extensibilidade procurando resolver problemas específicos. Essas funcionalidades são modeladas por meio de um conjunto de serviços que formam a arquitetura e podem ser invocados por meio da descrição de suas interfaces, uma vez que estas estão publicadas e descobertas por meio da rede (BOOTH et al., 2004).

Apesar de muitas definições citarem a utilização de serviços no formato de WS dentro de SOA, é importante ressaltar que os serviços web correspondem a apenas uma das maneiras de disponibilizar serviços por meio de SOA (ERL, 2007). Outros componentes e padrões abertos podem ser usados para disponibilizar funcionalidades por meio de serviços.

2.2.1 Serviços Web

O recente desenvolvimento da tecnologia web marca o início de uma nova era de computação orientada a serviços. Em particular, os serviços web permitem que os aplicativos se comuniquem uns com os outros por meio da Internet (GOTTSCHALK et al., 2002). O WS é um componente suportado por um conjunto de padrões abertos baseados em *eXtensible Markup Language* (XML), definido pelo *World Wide Web Consortium* (W3C), e o mais utilizado em SOA (PALACIOS; GARCÍA-FANJUL; TUYA, 2011).

O WS é um software projetado para suportar interação interoperável[2] máquina a máquina por meio da rede. Ele tem uma interface descrita em um formato processável[3] por computador denominado *Web Services Description Language* (WSDL) (W3C, 2015). O WS é o responsável por formalizar o acordo "contratual" entre fornecedor e consumidor da informação.

[2] É a capacidade de um sistema se comunicar de forma transparente com outro sistema.

[3] Que se consegue processar ou receber processamento de um sistema de computador.

> O serviço web é considerado um componente especializado de software e assim compartilha da mesma dificuldade relatada no desenvolvimento baseado em componentes, no qual os consumidores não conseguem testar adequadamente o componente disponibilizado para uso (MERILINNA; MATINLASSI, 2006, p.173).

A interação de outros sistemas com o serviço web é feita de uma maneira pré definida baseada no seu contrato, usando mensagens em *Simple Object Access Protocol* (SOAP), normalmente transmitidas por meio do protocolo *Hyper Text Transfer Protocol* (HTTP) com uma serialização em XML em conjunto com outros padrões relacionados à web semântica. O *Universal Description, Discovery and Integration* (UDDI), é uma iniciativa da indústria que permite às empresas publicarem seus serviços web. O serviço de registro UDDI é responsável por gerenciar os metadados dos serviços expostos na rede interna ou externa. Ele provê as informações necessárias para que os usuários possam pesquisar e descobrir os serviços web publicados na rede (OASIS, 2016).

A Figura 2.2 ilustra a interação entre o provedor e o usuário que consome o serviço disponibilizado e descoberto pelo serviço de registro dentro da infraestrutura. Tanto a publicação quanto a localização de um serviço são feitas com base no WSDL. Já o vínculo (*binding*) entre o provedor e o consumidor é estabelecido dinamicamente em tempo de execução.

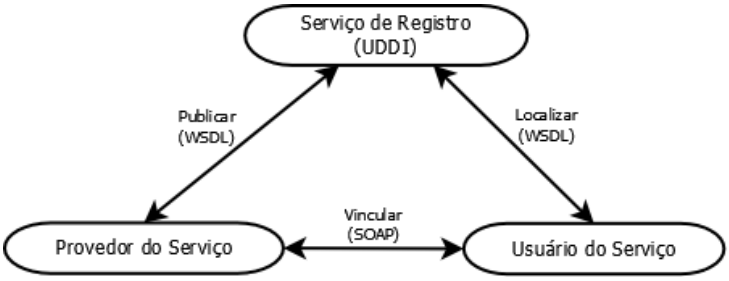

Figura 2.2: Arquitetura do serviço web, adaptado de Bozkurt, Harman e Hassoun (2010).

Estudos específicos sugerem a separação de serviços web em dois grupos, os baseados em SOAP e os de estilo *Representational State Transfer* (REST), mas esta distinção ainda não é precisa (KALIN, 2009). A forma mais comum de implementação de WS é utilizando mensagens SOAP, devido as suas características, tais como: independência de plataforma, linguagem e transporte; padronização de mensagem; tratamento de erros; atomicidade nas transações; e suporte a níveis de segurança específicos, como o WS-Security (ERL, 2007).

Os serviços web representam um novo paradigma para a construção de aplicações de computação distribuída e oferecem vantagens sobre plataformas de *middleware* convencionais, como o *Common Object Request Broker Architecture* (CORBA) (CHEN; ROMANOVSKY, 2008). Essas vantagens vão desde simplicidade e padronização nas trocas de mensagens baseadas em XML, até a flexibilidade e independência de plataforma que este paradigma provê.

> Um WS pode conter vários métodos que comunicam-se com outros componentes e aplicações por meio da web. Esses métodos do serviço web têm interfaces de invocação que assemelham-se aos métodos de software convencionais, mas são consideravelmente mais complexos na implementação (PRAS; MARTIN-FLATIN, 2008, p.252).

Os estudos conduzidos por este trabalho são focados no desenvolvimento de WS baseado em SOAP devido a popularidade e maior adoção nas empresas pesquisadas.

2.3 Qualidade do Produto de Software

Para Schumacher (2007), qualidade é estar em conformidade com os requisitos dos clientes, neste caso qualidade se exprime a partir da satisfação de todas as necessidades expressas pelo cliente durante a análise de requisitos.

Segundo a norma ABNT (1994), qualidade é a habilidade de satisfazer as necessidades explícitas e implícitas do usuário. É necessário entender as reais necessidades do usuário detalhadamente e representá-las nos requisitos do software para que satisfaça às suas necessidades.

A qualidade de software é a concordância com os requisitos funcionais e de desempenho claramente colocados, padrões de desenvolvimento explicitamente documentados e características implícitas que são esperadas de todo software profissionalmente desenvolvido (PRESSMAN, 1997). Para medir e melhorar a qualidade de software é preciso definir as características de qualidade que se está interessado e, então, decidir como elas serão medidas (KITCHENHAM, 1996).

A qualidade do produto de software é outra área extensa e abrangente no processo de desenvolvimento de software. Por este motivo, esta seção aborda apenas os principais termos e conceitos relacionados ao contexto deste trabalho. Alguns conceitos são apresentados de maneira resumida apenas para contextualização, como é o caso das categorias que compõem o modelo de qualidade apresentado nesta seção.

Existem três visões diferentes da qualidade do produto nos estágios do ciclo de vida do software, são elas: Qualidade em Uso (QU), Qualidade Interna (QI) e Qualidade Externa (QE). Essas visões são definidas a seguir:

- **Qualidade em Uso:** É a visão da qualidade do produto de software do ponto de vista do usuário, quando este produto é usado em um contexto de uso e ambiente especificado. Ela mede o quanto os usuários podem atingir seus objetivos num determinado ambiente e não as propriedades do software em si (ABNT, 2003). Esses objetivos são divididos em: Efetividade, Produtividade, Segurança e Satisfação.

- **Qualidade Interna:** É a totalidade das características do produto de software do ponto de vista interno. A qualidade interna é medida e avaliada com relação aos requisitos de qualidade interna (ABNT, 2003). Esses requisitos são usados para definir as estratégias de arquitetura e desenvolvimento apoiando-se em métricas internas do paradigma de desenvolvimento de software. Essas características são percebidas pelos arquitetos e desenvolvedores que criaram ou irão manter o software, não sendo perceptível para o usuário final.

- **Qualidade Externa:** É a totalidade das características do produto de software do ponto de vista externo. É a qualidade quando o software é executado, o qual é tipicamente medido e avaliado enquanto está sendo testado num ambiente simulado, com dados simulados e usando métricas externas (ABNT, 2003). Porém, mesmo com os testes alguns defeitos podem continuar na versão que será disponibilizada ao usuário.

A Figura 2.3 apresenta as categorias dos atributos de qualidade que compõem o modelo apresentado em ABNT (2003). As seis categorias principais são: Funcionalidade (oferecer funções que atendam às necessidades explícitas e implícitas); Confiabilidade (manter um nível de desempenho especificado), Usabilidade (ser compreendido, aprendido, operado e atraente ao usuário), Eficiência (apresentar desempenho apropriado, relativo à quantidade de recursos usados), Manutenibilidade (facilidade em ser modificado) e Portabilidade (facilidade em ser transferido de um ambiente para outro).

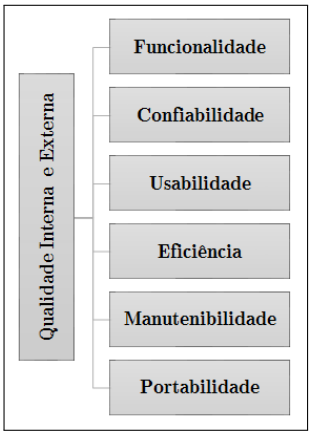

Figura 2.3: Modelo de qualidade para QI e QE, adaptado de ABNT (2003).

2.3.1 Teste de Software

Teste é o processo de executar um programa com o intuito de encontrar defeitos (MYERS; SANDLER; BADGETT, 2011). A afirmação de Dijkstra (1970) é complementar a anterior e descreve que o teste de software pode ser usado para mostrar a presença de defeitos, mas nunca para mostrar a sua ausência. Portanto, o objetivo do teste é fornecer informações sobre a qualidade do software de acordo com requisitos definidos no início de seu desenvolvimento. Com isso, espera-se aumentar a qualidade do software desenvolvido por meio da prevenção dos defeitos e da execução de testes e validações antes da entrega do software para o usuário.

Teste de software é uma área extensa do desenvolvimento de software que depende de vários fatores, como por exemplo: a plataforma de desenvolvimento, o modelo e o processo de desenvolvimento, o contexto do sistema a ser desenvolvido e de outras variáveis. Além disso, existem diversos modelos, fases, técnicas, estratégias e conceitos de teste de software. Porém, esta seção aborda apenas os principais conceitos relacionados ao contexto deste trabalho.

As técnicas e os níveis de teste de software são definidas a seguir:

- **Caixa Branca:** Avalia o comportamento interno do componente. Ela é de responsabilidade do desenvolvedor, pois é necessário ter acesso ao código fonte. Segundo Myers, Sandler e Badgett (2011), os testes são baseados nas estruturas internas do código fonte e visam examinar a lógica escrita no programa.

- **Caixa Preta:** Baseada na especificação da interface do componente (contrato), o testador não possui conhecimento da estrutura interna do programa e os dados são derivados da

interface exposta. Para Myers, Sandler e Badgett (2011), o testador deve concentrar-se em encontrar circunstâncias na qual o programa não se comporta de acordo com as suas especificações.

- **Caixa Cinza:** Utiliza estruturas de dados e algoritmos, parcialmente conhecidos, como base para criação dos testes, que é de responsabilidade do desenvolvedor. Ela é uma combinação das técnicas caixa branca e caixa preta. O que diferencia esta técnica das demais é que o acesso a estruturas internas do software é limitado. Segundo Li et al. (2008), esta limitação de acesso é muito comum no desenvolvimento de aplicações distribuídas e principalmente em serviços web.

Fazendo uma breve contextualização sobre o nível de teste, a seguir são definidos os conceitos e objetivos do teste de unidade e teste de integração:

- **Teste de Unidade:** Examina o comportamento de uma unidade distinta de trabalho (MASSOL et al., 2010). Esse nível de teste deve assegurar que as unidades menores (módulo, classe ou método) funcionem de acordo com o que foi especificado, independente do restante do sistema.

- **Teste de Integração:** Visa encontrar falhas na interação entre as unidades do sistema (ROCHA; MALDONADO; WEBER, 2001). Neste nível o objetivo é identificar pontos de falhas na comunicação entre as unidades do software desenvolvidas anteriormente.

2.3.2 Teste de Serviços Web

As pesquisas na área de teste de serviços web ainda estão em estágios iniciais e um dos grandes problemas que retarda a expansão do uso de serviços web é a dificuldade em testá-los do lado consumidor do serviço (BOZKURT; HARMAN; HASSOUN, 2010).

De acordo com a revisão sistemática descrita no Apêndice A.3.2, inúmeras contribuições têm sido apresentadas na literatura, principalmente nas áreas de testes de integração, unitário, regressão e não-funcional. Porém, grande parte dessas pesquisas são direcionadas à técnica de teste caixa preta.

Para Bozkurt, Harman e Hassoun (2013), existem outras necessidades abertas e que precisam de novas investigações: 1) Carência de estudos de caso do mundo real; 2) Soluções que possam gerar dados de teste realistas; 3) Soluções para reduzir o custo de testes em sistemas orientado a serviços; 4) Soluções que melhoram a capacidade de teste em sistemas orientado a serviços; 5) Soluções que combinem testes e verificação em sistemas orientado a serviços; 6) Modelagem e validação em sistemas orientados a serviços.

Dentre essas questões, a mais importante atualmente é a falta de exemplos do mundo real em funcionamento e disponíveis (BOZKURT; HARMAN; HASSOUN, 2013). Dos 167 trabalhos selecionados, apenas 11% tinham aplicação no mundo real e 71% não tiveram nenhuma validação experimental (BOZKURT; HARMAN; HASSOUN, 2013).

Atualmente, faltam estudos que abordem ferramentas para o desenvolvimento de testes de unidade em serviço web (ZAKARIA et al., 2009). Ainda, segundo os autores, isso resulta em práticas ineficientes em teste e depuração de serviços e processos de negócios automatizados.

O presente trabalho conduziu experimentos com profissionais no ambiente corporativo, com foco nas técnicas de teste caixa branca e caixa cinza, visando colaborar com novos estudos nesta área.

2.4 Métricas de Software

DeMarco (1986) diz que é importante medir e controlar qualquer processo de produção. Um conjunto de métricas bem definidas é fundamental para medir e controlar um processo.

A métrica é um método de medição para determinar uma medida ou uma característica de determinado objeto (KOSCIANSKI; SOARES, 2007). Segundo Schumacher (2007), é necessário estabelecer métricas para rastrear a qualidade do software. Medindo, poderemos saber se as ações de melhoria deram resultado.

Sob o ponto de vista de medição, existem duas subdivisões: medidas diretas (quantitativas) e indiretas (qualitativas) (PRESSMAN, 1997). A diferença entre as duas é que a medida direta envolve apenas uma variável, enquanto medida indireta envolve uma n-tupla em seu domínio (KANER; BOND, 2004).

De acordo com a norma IEEE 1061, a medida direta são medidas que não precisa de outra mensuração ou de qualquer outro atributo. Enquanto que medida indireta são as medidas extraídas com a utilização de uma ou mais medidas diretas que podem estar combinas com constantes IEEE (1998). Um exemplo de medida direto em software são as linhas de código e um exemplo de media indireta é a eficiência, complexidade e produtividade do software.

As comparações realizadas neste trabalho foram feitas com base em métricas relacionadas a qualidade interna e qualidade externa do software e na produtividade dos desenvolvedores enquanto implementavam os exercícios. As métricas técnicas de qualidade interna foram escolhidas com base nas métricas do paradigma orientado a objetos. Além disso, outras métricas foram definidas para quantificar a qualidade externa e a produtividade dos desenvolvedores, viabilizando a comparação dos resultados obtidos. Os detalhes das métricas utilizadas são apresentados na Seção 3.2.2.

2.5 Trabalhos Correlatos sobre TDD

Os estudos encontrados sobre a prática de TDD no desenvolvimento de software têm por objetivo realizar uma comparação entre as práticas de TDD e TLD. Os artigos foram identificados no processo de revisão sistemática descrito no Apêndice B.1 e publicado em Bissi, Neto e Emer (2016).

Cada estudo fez a comparação seguindo um contexto específico; não há uniformidade entre as métricas e parâmetros utilizados entre os estudos e isto dificulta a comparação entre eles.

Esses trabalhos apresentaram uma divisão clara em dois grupos relacionados aos cenários, sendo eles acadêmico e industrial. Apesar dos trabalhos serem realizados em ambiente industrial com profissionais de mercado, alguns estudos não foram realizados utilizando projetos reais, ou seja, foram utilizados exemplos fictícios. De maneira geral, as comparações consideraram a qualidade interna, qualidade externa e a produtividade no desenvolvimento das aplicações.

A métrica mais utilizada como indicador que determina a qualidade interna foi a cobertura de testes no código fonte. Essa métrica determina a quantidade de testes unitários feitos para cobrir o código fonte do sistema. De acordo com Bissi, Neto e Emer (2016), ela é considerada fraca para medir a qualidade interna, quando usada isoladamente. Outras métricas como a coesão, complexidade e acoplamento deveriam ser consideradas para identificar a variação na qualidade interna.

Somente os trabalhos Janzen e Saiedian (2008a), Janzen e Saiedian (2006), Pancur e Ciglaric (2011), Aniche e Gerosa (2012) e Siniaalto e Abrahamsson (2007) consideraram as métricas de coesão, complexidade e acoplamento de código para realizar a comparação entre as práticas. O uso dessas métricas traz mais segurança aos resultados apresentados pelos estudos.

Outros estudos aprofundaram-se em características específicas no desenvolvimento de software. Aniche e Gerosa (2012) procuraram identificar padrões que expliquem a hipótese de que a prática do TDD melhora o projeto de classes. Segundo os autores, a prática do TDD pode influenciar o projeto de classe, mas ela não guia o desenvolvedor automaticamente para um bom projeto de classe. Isso é parte resultante do *feedback* que o TDD passa, podendo variar de acordo com a experiência de cada desenvolvedor.

Com relação a qualidade externa, a forma de medição foi praticamente a mesma em todos os trabalhos, sendo extraída com base no número de casos de teste que falharam ou passaram ao fim do desenvolvimento do software, gerando uma densidade de defeitos por linha de código. Apenas Pancur et al. (2003) apontaram que houve uma diminuição na qualidade externa.

A produtividade foi a categoria que mais apresentou diferenças nos resultados de acordo com o cenário utilizado. No cenário acadêmico, apenas Vu et al. (2009) concluiram que houve uma queda na produtividade, os demais observaram que a produtividade se manteve ou aumentou. No cenário industrial, nenhum estudo identificou que a produtividade aumentou. A produtividade foi medida considerando o tempo gasto para implementar cada funcionalidade.

George e Williams (2003) identificaram que existe uma correlação moderada entre o tempo gasto e a qualidade resultante. Essa correlação pode justificar a diminuição na produtividade em sistemas que apresentam menos defeitos em sua versão final.

Dois estudos sobre a adoção de TDD no desenvolvimento de serviços web trazem contribuições significativas para a área. Eles foram identificados no processo de revisão sistemática descrito no Apêndice B.2.

Em Laranjeiro e Vieira (2009), o foco da pesquisa foi nos requisitos não funcionais do WS, mais especificamente a robustez. Os autores propuseram uma extensão ao fluxo TDD para viabilizar a inclusão de testes de robustez em serviços web. Já em Besson et al. (2015) o foco da pesquisa foi na criação e utilização de uma ferramenta, denominada de Rehearsal, para facilitar os testes de software no desenvolvimento de coreografia de serviços.

Atualmente existem poucos trabalhos que procuram analisar a adoção de TDD no desenvolvimento de serviços web. Esta área de estudo ainda é recente e novos trabalhos devem ser propostos para conduzir novos avanços nesta área.

2.6 Considerações Finais do Capítulo

Este capítulo apresentou uma visão geral sobre os conceitos utilizados. Os métodos ágeis e arquitetura orientada a serviços são os principais temas deste trabalho e por isso foram descritos em mais detalhes. Já o teste de software, qualidade do produto de software e as métricas de software foram descritos superficialmente para embasar o leitor sobre os assuntos tratados ao longo dessa dissertação.

Além disso, a seção anterior destaca e comenta alguns estudos correlatos sobre TDD, fazendo referência a revisão sistemática conduzida neste trabalho e descrita no Apêndice B.1 e Apêndice B.2.

O próximo capítulo apresenta o método de pesquisa utilizado para o desenvolvimento de todo o trabalho, descrevendo as principais atividades executadas desde o início até o fim desta dissertação. O foco está no detalhamento da Revisão Sistemática, Pesquisa Quantitativa, Experimento e Entrevista.

Capítulo 3

Método de Pesquisa

Este capítulo descreve o método de pesquisa utilizado para o desenvolvimento deste trabalho, desde a definição dos objetos de estudo até a apresentação do trabalho final. A Figura 3.1, escrita em notação *Business Process Modeling Notation* (BPMN), mostra o fluxo completo da execução deste trabalho.

A raia Definição do Projeto de Pesquisa representa as atividades executadas durante a definição do projeto de pesquisa. O objeto de estudo foi definido e refinado durante a execução da revisão sistemática da literatura; Depois de consolidar as informações o escopo foi delimitado e o método de pesquisa foi definido; Em seguida, o projeto de pesquisa é descrito e planejado em sua totalidade, antes da execução.

A raia Execução do Projeto de Pesquisa contempla desde a execução do projeto de pesquisa definido e planejado até a escrita e apresentação dessa dissertação de mestrado para a banca examinadora. O questionário presencial para mapeamento da percepção dos desenvolvedores sobre TDD é aplicado a 116 profissionais e seus resultados analisados; as respostas deste questionário contribuem para a elaboração da abordagem WS-TDD; após a definição da WS-TDD, os experimentos de validação da abordagem são definidos e aplicados a 23 desenvolvedores de software; em seguida os dados são coletados, analisados, e os resultados obtidos são consolidados; por fim, a escrita da dissertação é finalizada e o trabalho final apresentado a banca examinadora.

A atividade destacada em negrito e sublinhada representa a principal contribuição do mestrado, a construção da abordagem WS-TDD, descrita em detalhes no próximo capítulo.

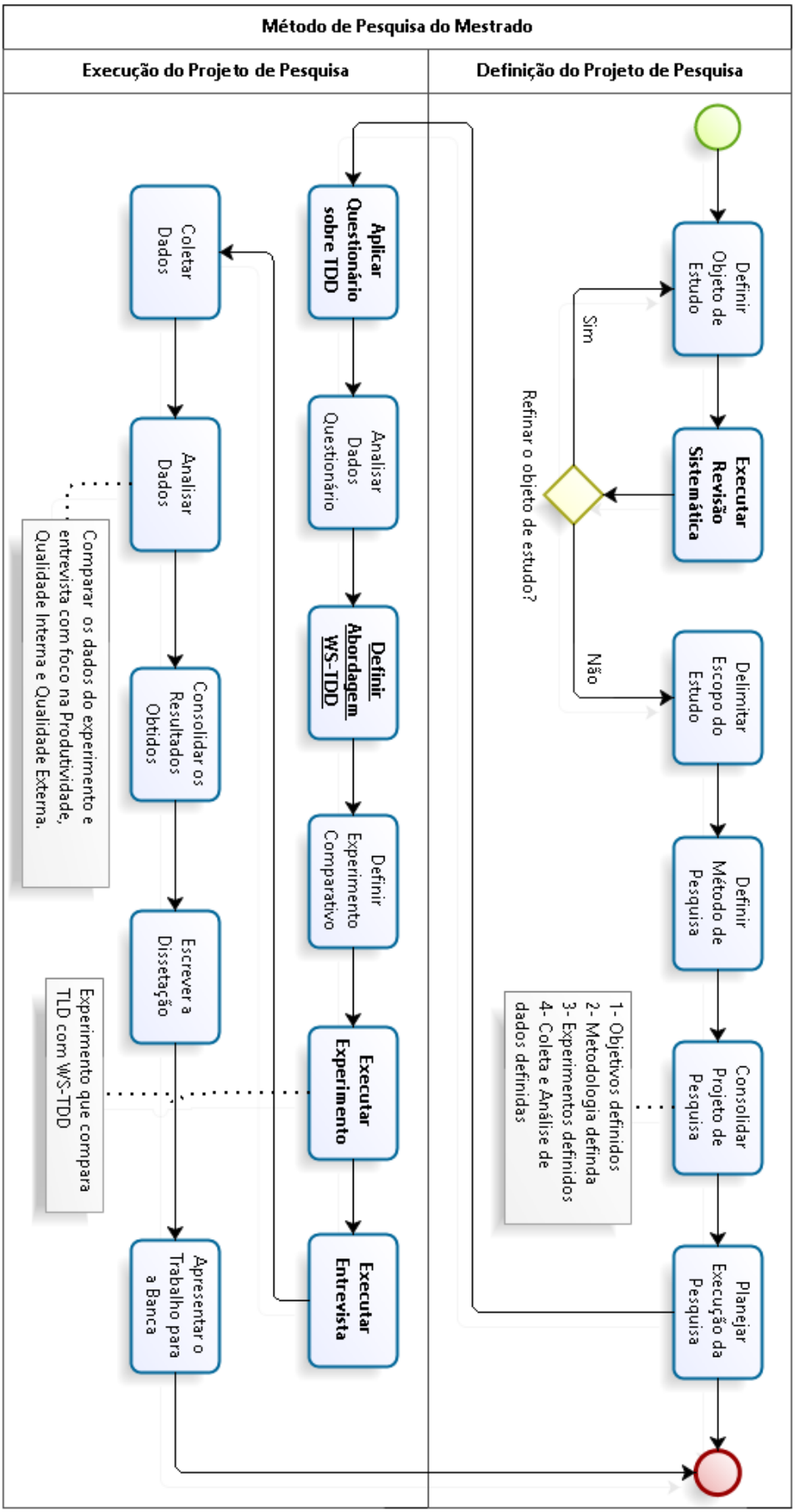

Figura 3.1: Fluxo de pesquisa do mestrado. Fonte: Autoria própria.

Métodos específicos são necessários para ajudar a estabelecer uma base de engenharia e de ciência para a engenharia de software (TRAVASSOS et al., 2008). Este trabalho realizou quatro atividades específicas dentro do método de pesquisa, são elas: Revisão Sistemática; Pesquisa Quantitativa; Experimento e Entrevista. Essas atividades estão destacadas em negrito na Figura 3.1, o detalhamento de cada uma delas é feito a seguir.

1. Para identificar novas oportunidades de pesquisas e as lacunas existentes nos trabalhos atuais, foi realizada uma **revisão sistemática da literatura**. O método dessa revisão é descrito no Apêndice A e a análise detalhada está no Apêndice B. O estudo de Bissi, Neto e Emer (2016) também traz os detalhes desta revisão;

   1.1 A revisão da literatura contribuiu para identificar oportunidades de pesquisa e mostrar que os experimentos que buscam identificar os efeitos de TDD na qualidade de software são focados apenas em sistemas centralizados. Além disso, o resultado desta revisão confirmou as lacunas apontadas na Seção 1.1.

2. Para identificar a importância do tema escolhido, as necessidades atuais dos desenvolvedores e o mapeamento do conhecimento sobre TDD em três empresas locais, foi realizada uma **pesquisa quantitativa** aplicando um questionário com 19 perguntas sobre TDD. O questionário é apresentado no Apêndice C e detalhado na Seção 3.1.

   2.1 O questionário contribuiu para identificar a importância de TDD, mapear o nível de conhecimento dos profissionais sobre TDD, identificar as barreiras encontradas ao tentar adotar TDD (Questões 3, 5, 6, 9 e 16). O questionário coletou informações que ajudaram na criação da WS-TDD no que diz respeito ao uso de WS no desenvolvimento de software e no uso de ferramentas e objetos simulados para auxiliar a escrita de testes unitários com WS (Questões 14 e 15). Além disso, o questionário obteve a percepção dos profissionais quanto aos efeitos que TDD pode trazer para a qualidade interna e qualidade externa do software e na produtividade dos desenvolvedores (Questões 17, 18 e 19). As principais questões que contribuíram para este trabalho estão detalhadas na Seção 5.1.

3. A partir da revisão sistemática da literatura e da pesquisa quantitativa, foi proposta a WS-TDD. Para a concepção e a validação desta abordagem foi realizado um **experimento** com desenvolvedores que atuam no mercado de tecnologia da informação em Curitiba. Este experimento é detalhado na Seção 3.2.

4. Para refinar e melhorar o entendimento dos resultados obtidos com o experimento, foi realizada uma **entrevista** com os participantes após a execução do experimento. O questionário para a entrevista é apresentado no Apêndice D e a entrevista detalhada na Seção 3.3.

3.1 Questionário

O questionário de pesquisa foi aplicado entre os meses de Março e Junho de 2015 a profissionais da área de tecnologia da informação de três empresas diferentes. Primeiramente o questionário foi aplicado em uma empresa de telecomunicação (denominada Empresa A), depois em uma empresa de desenvolvimento de software (denominada Empresa B) e por fim no

grupo de tecnologia da informação de um órgão público estadual (denominada Empresa C). O questionário foi aplicado durante o expediente de trabalho dos profissionais.

> Questionário é um método de coletar dados no campo, de interagir com o campo composto por uma série ordenada de questões a respeito de variáveis e situações que o pesquisador deseja investigar. Tais questões são apresentadas a um respondente, por escrito, para que ele responda também dessa forma, independentemente de ser a apresentação e a resposta em papel ou em computador. A escolha do meio é sempre do pesquisador. (VERGARA, 2013, p.39).

Para se ter uma ideia de dimensão dessas empresas, a Empresa A possui aproximadamente 200 desenvolvedores, a Empresa B possui cerca de 70 desenvolvedores e a Empresa C possui em torno de 35 desenvolvedores. É importante ressaltar que essa quantidade é referente ao setor em que o questionário foi aplicado e não o total de desenvolvedores presentes na empresa como um todo. Todas elas são empresas de grande porte, com mais de 500 funcionários[1].

O objetivo deste questionário foi mapear o conhecimento, o tempo de experiência e a opinião dos profissionais de tecnologia de informação sobre a prática de TDD no desenvolvimento de software, mais especificamente no desenvolvimento de serviço web. Este questionário de pesquisa foi elaborado e aplicado em três empresas locais, conduzindo uma pesquisa quantitativa acerca do uso de TDD durante o desenvolvimento de software.

Todas as empresas citadas estão localizadas na cidade de Curitiba no estado do Paraná e foram escolhidas por estarem próximas e utilizarem as tecnologias citadas na pesquisa. Em média os profissionais demoraram em torno de 20 minutos para responder e devolver o questionário de pesquisa.

A extensão do questionário não deve ser tal que desanime o respondente a responder (VERGARA, 2013). Um questionário deve ser montado de tal forma que demore cerca de 20 a 30 minutos para ser respondido (MARCONI; LAKATOS, 2005). Portanto, este questionário está dentro das boas práticas citadas anteriormente pela literatura.

As Empresas A e B, no qual o questionário de pesquisa foi aplicado, utilizam a linguagem Java para o desenvolvimento de software e a Empresa C utiliza a linguagem C# para o desenvolvimento de suas aplicações. Todos os profissionais da área de tecnologia da informação que participaram desta pesquisa têm contato direto com o desenvolvimento de software e possuem experiência nesta área. Todos eles atuam ou já atuaram como desenvolvedores de software.

O questionário de pesquisa possui quatro páginas (ver Apêndice C): a primeira contém uma breve apresentação dos pesquisadores e o tema de pesquisa abordado, a segunda página traz o termo de consentimento livre e esclarecido solicitado pelo Comitê de Ética em Pesquisa da UTFPR.

As duas últimas páginas são compostas pelas informações pessoais como nome, e-mail, idade e empresa em que o profissional trabalha e 19 perguntas fechadas sobre o tema de pesquisa: 10 delas são de múltipla escolha e 9 delas são escalonadas, utilizando Likert como escala de razão.

As questões de 1 a 3 identificam a formação e a experiência dos desenvolvedores. As questões de 4 a 10 mapeiam os conhecimentos técnicos sobre ferramentas, linguagens e práticas no desenvolvimento de software. As questões de 11 a 15 mapeiam o conhecimento em serviços web, objetos simulados e teste unitário de código. Por fim, as questões de 16 a 19

[1] Critério de classificação usado: http://www.sebrae-sc.com.br/leis/default.asp?vcdtexto=4154

visam identificar a opinião dos desenvolvedores com relação a prática de TDD. O formato das perguntas e respostas que compõem este questionário foi baseado nos modelos apresentados e discutidos em Vieira (2009) e Vergara (2013).

O questionário foi impresso e entregue pessoalmente para cada um dos 116 profissionais que aceitaram participar da pesquisa. Todos participantes responderam e devolveram o formulário respondido. Por não atender o pré-requisito de conhecer a prática de TDD, 10 questionários foram removidos da amostragem final de pesquisa. Portanto, dos 116 questionários recebidos, 106 foram analisados. A distribuição dos questionários respondidos para cada uma das três empresas é apresentada a seguir:

- Empresa A: 76 Questionários respondidos, representa 71,70% da amostragem;
- Empresa B: 22 Questionários respondidos, representa 20,75% da amostragem;
- Empresa C: 8 Questionários respondidos, representa 7,55% da amostragem.

As respostas dos questionários podem ser trianguladas com outros métodos de interação com o campo, tais como a técnica de construção, a técnica de complemento, a entrevista, a observação e outros (VERGARA, 2013). O Capítulo 5 apresenta os resultados do trabalho e da triangulação das respostas do questionário com resultados obtidos no experimento e na entrevista. A triangulação foi realizada selecionando apenas os profissionais que participaram dos três métodos de pesquisa.

3.1.1 Limitações do Questionário

A utilização de questionários, apesar de trazer várias possibilidades de adequação à pesquisa, como a inclusão de maior número de respondentes, possui também algumas limitações (VERGARA, 2013).

Quanto ao formato de questões fechadas utilizado, Marconi e Lakatos (2005) destacam como limitação a menor liberdade nas respostas e o risco de respostas distorcidas, em virtude da opções disponíveis. Para reduzir este risco, grande parte das questões tinham como opção de escolha uma alternativa que permitisse o respondente a não opinar ou escrever uma resposta que lhe atenda.

Para mitigar a possibilidade de baixa devolução, foi solicitada a resposta do questionário durante o horário de expediente. Em grande parte, a devolução foi feita no mesmo dia. Para as exceções, foi usado o contato telefônico para lembrar da devolução do questionário. Com isso, todos os questionários foram respondidos e devolvidos.

O questionário foi distribuído para os profissionais de tecnologia de informação que conhecem as tecnologias apresentadas e atuam em uma das três empresas citadas. Essas empresas foram escolhidas devido à proximidade que o autor possui com cada uma e também por serem empresas selecionadas para a realização do experimento proposto na Seção 3.2. Dessa forma, os resultados do questionário só podem ser generalizados para os desenvolvedores que trabalham nessas empresas, não sendo generalizável para outros desenvolvedores.

O entendimento das questões descritas no questionário também é outra limitação a ser considerada. Para mitigá-la o autor questionou os participantes, no momento da entrega do formulário, se restaram dúvidas sobre questionário. Caso positivo, o autor realizava os esclarecimentos necessários.

3.2 Experimento

Um experimento foi conduzido com 23 desenvolvedores de software que trabalham nas Empresas A e B entre os meses de Setembro e Outubro de 2015. Essas empresas foram escolhidas para realizar o experimento pois o ambiente de desenvolvimento delas é baseado em Java, que é o foco deste experimento, e também por terem participado do primeiro questionário.

O objetivo deste experimento é investigar e comparar os resultados obtidos ao aplicar as abordagens TLD e WS-TDD nos dois exercícios propostos. A comparação entre as abordagens é feita em três categorias: qualidade interna, qualidade externa e produtividade. Essas categorias são descritas na Seção 3.2.2.

Os desenvolvedores selecionados para participar deste experimento possuem conhecimento na linguagem de programação Java e no desenvolvimento de serviços web. Porém, o tempo de experiência em desenvolvimento de software varia para cada um deles. Os desenvolvedores foram divididos em dois grupos no qual a ordem de execução dos exercícios (usando TDD e TLD) foram alternadas.

Os dois exercícios implementados pelos participantes possuem o mesmo número de parâmetros de entrada e saída, a mesma quantidade de regras de negócio e maneiras similares de validá-las.

O intuito de deixar os exercícios similares foi igualar a complexidade de implementação encontrada nos dois exercícios para que a comparação entre TLD e WS-TDD seja mais coerente. O contexto dos dois exercícios é diferente para não induzir o participante a fazer a mesma implementação, da mesma maneira e com o mesmo código, ou seja, tentou-se evitar ao máximo que qualquer vantagem surja durante o desenvolvimento dos exercícios.

O foco da implementação das regras propostas nestes exercícios é provocar mudanças nos indicadores de qualidade interna, qualidade externa e produtividade de acordo com as decisões de cada participante no momento do desenvolvimento.

Após a execução do experimento, os desenvolvedores são direcionados a uma entrevista sobre a implementação dos exercícios que acabaram de realizar e as abordagens utilizadas. Os detalhes da entrevista são descritos na Seção 3.3.

3.2.1 Estrutura do Experimento

Antes de realizar o experimento oficialmente, foi realizado um experimento teste com dois profissionais para identificar possíveis problemas na compreensão dos exercícios e também para verificar se o tempo previsto para realização era factível. Após os ajustes, a distribuição de tempo ficou prevista em 8 horas no total, considerando o intervalo para refeição, sendo distribuído da maneira a seguir:

- 30 Minutos: Apresentação do Exercício 1 e da abordagem TLD;
- 150 Minutos: Tempo para os participantes implementarem o Exercício 1;
- 80 Minutos: Intervalo para refeição (almoço);
- 40 Minutos: Apresentação do Exercício 2, explicação sobre ágil, TDD e WS-TDD;
- 150 Minutos: Tempo para os participantes implementarem o Exercício 2;

- 30 Minutos: Tempo para realização da entrevista sobre o experimento.

Para evitar qualquer viés nos resultados do experimento devido sequência de execução dos exercícios, a seguinte estratégia foi adotada: Os desenvolvedores foram separados em dois grupos, o primeiro grupo seguiu o sequenciamento apresentado anteriormente, ou seja, foi apresentada a técnica e o exercício seguindo o TLD e em seguida a técnica e o exercício referente a WS-TDD. A ordem de apresentação da técnica foi invertida para o segundo grupo de desenvolvedores, iniciando com a WS-TDD e finalizando com a TLD.

Foram executadas três turmas para cada um dos dois grupos. No Grupo 1 participaram 4 desenvolvedores em cada turma, totalizando 12 participantes. No Grupo 2 participaram 4 desenvolvedores em 2 turmas e 3 desenvolvedores na terceira turma (houve uma desistência), totalizando 11 participantes.

Os dois exercícios propostos foram implementados na linguagem Java (versão 1.7.40) utilizando a IDE Eclipse (versão 4.4.2) e o arcabouço de teste TestNG (versão 6.9). O servidor de aplicação para a implantação dos exercícios é o WebLogic Server (versão 12.1.1). Essas ferramentas foram escolhidas por serem de uso comum nas empresas em que o experimento foi aplicado. O Apêndice E apresenta os detalhes dos dois exercícios implementados neste experimento.

A Figura 3.2 apresenta o fluxo utilizado para a execução do experimento. Na primeira etapa, os desenvolvedores são convocados para participar do experimento, em seguida o instrutor realiza a apresentação da abordagem TLD ou WS-TDD e na sequência os desenvolvedores irão implementar o exercício da abordagem explicada. Como dito anteriormente, a ordem de apresentação da abordagem e da aplicação dos exercícios sobre TLD e WS-TDD é alternada para cada um dos grupos, com o intuito de eliminar qualquer viés que impacte nas análises dos resultados.

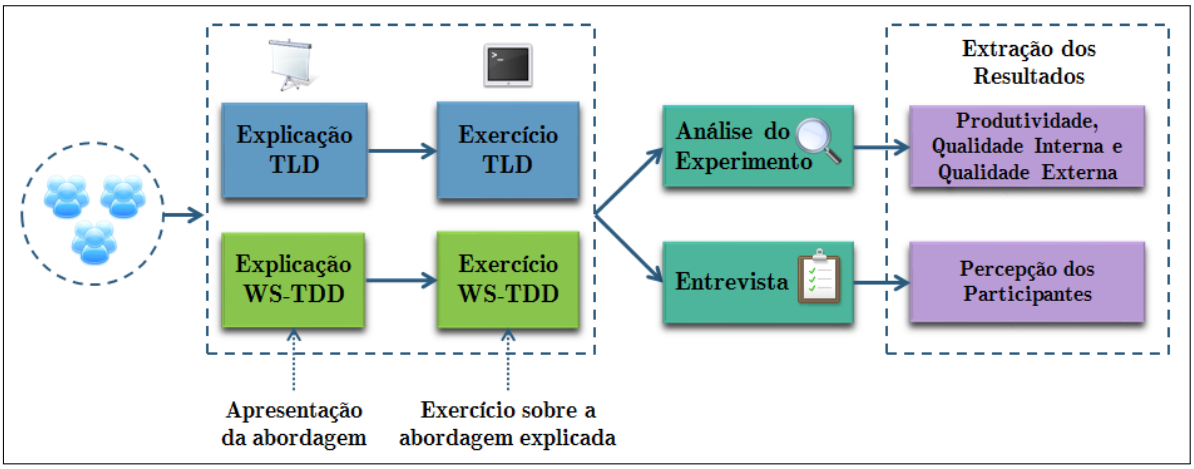

Figura 3.2: Fluxo detalhado do experimento. Fonte: Autoria própria.

Ao finalizarem a implementação dos dois exercícios propostos, os desenvolvedores enviaram o código produzido para um sistema de controle de versão. Em seguida o código foi analisado com base nas métricas de qualidade interna, qualidade externa e produtividade, descritas na Seção 3.2.2.

3.2.2 Métricas Observadas no Experimento

As métricas utilizadas para extrair informações dos exercícios realizados durante o experimento estão divididas em três categorias: Qualidade Interna (métricas técnicas); Qualidade Externa (métricas de qualidade); Produtividade (métricas de produtividade).

O objetivo dessas métricas é fornecer informações que possibilitem a comparação entre as abordagens TLD e WS-TDD no que diz respeito a qualidade interna do código, qualidade externa da aplicação e produtividade do desenvolvedor em cada um dos exercícios realizados. Usando essas métricas é possível comparar igualmente os resultados obtidos, independentemente da abordagem utilizada.

Qualidade Interna

A qualidade interna é aferida por meio de um sistema de análise de código conhecido como Sonar (versão 3.7.4). Esse sistema, por meio de seus *plugins* (como o Squid, FindBugs, PMD e CheckStyle), provê informações sobre as métricas de código conhecidas da orientação a objetos. As métricas que foram consideradas neste experimento estão listadas na Tabela 3.1. Elas foram selecionadas por serem as mais utilizadas atualmente para identificar a qualidade do código fonte desenvolvido em linguagens orientadas a objetos.

- **RFC (*Response for a Class*):** Indica a presença de muitos métodos na classe o que aumenta a complexidade.
- ***Complexity, Complexity per Class, Complexity per Method*:** É usada a complexidade ciclomática de McCabe (1976), que determina se o código fonte é mais complexo, de acordo com as estruturas, variáveis e regras utilizadas no desenvolvimento. Essa métrica apresenta 3 formas de medição: Complexidade geral do projeto; Complexidade das classes; e Complexidade dos métodos.
- ***Coverage* e *Line Coverage*:** *Coverage* é a porcentagem de código produtivo coberto por teste automatizados. *Line Coverage* é a porcentagem de linha de código produtivo coberto pelos testes automatizados escritos.
- ***Uncovered Lines*:** É a quantidade de linhas de código que não foram cobertas pelos testes automatizados.
- **LoC (*Lines of Code*):** Quantidade de linhas de código produtivo, desconsiderando os comentários e linhas em branco.
- ***Lines*:** Quantidade total de linhas de código.
- **Violações *Blocker, Critical, Major, Minor* e *Info*:** Quantidade de violações das regras usadas pelos *plugins* do Sonar citados no início desta seção. Os 5 tipos de violações são separadas de acordo com o grau de severidade definida. Considerando do maior nível de severidade para o menor, a sequência é *Blocker, Critical, Major, Minor* e *Info*. No subitem a seguir são apresentados alguns exemplos dessas violações de acordo com sua criticidade.

  - *Blocker*: *"Throwable and Error should not be caughy"*, essa violação diz que não se deve capturar a exceção raiz da linguagem, mas sim a exceção especializada para fazer o tratamento correto do problema ocorrido.
  - *Critical*: *"Use a logger to log this exception"*, essa violação diz que a exceção ocorrida deve ser escrita usando o comando para log (*"LOGGER"*) ao invés de mostrá-la apenas no console do servidor. Isso facilita a investigação de possíveis defeitos no sistema.
  - *Major*: *"Inheritance tree of classes should not be too deep"*, essa violação diz que uma classe não pode conter um número excessivo de superclasses. No caso dessa violação a quantidade considerada aceitável é de no máximo 5.
  - *Minor*: *"Constant names should comply with a naming convention"*, essa violação diz que a constante deve ser nomeada seguindo a convenção de nomenclatura definida pela linguagem Java. Essa padrão deve obedecer a seguinte expressão regular: "^[A-Z][A-Z0-9]*(_[A-Z0-9]+)*$".
  - *Info*: *"Do not forget to remove this deprecated code someday"*, essa violação diz que o código com anotação de depreciado deve ser removido um dia do sistema. Esse é o nível mais brando de criticidade e normalmente não afeta o desempenho ou a complexidade do sistema.

- ***Rules Compliance* (Conformidade com as Regras):** É o percentual de conformidade com as regras e boas práticas de desenvolvimento baseado em uma listagem com 516 regras usadas habitualmente na comunidade de desenvolvimento de software. Esse indicador é um compilado das violações (*Blocker, Critical, Major, Minor* e *Info*) citadas no item anterior. As regras podem ser obtidas juntamente com o Sonar[2] na versão 3.7.4.

Todas as métricas listadas na Tabela 3.1 são implementadas e extraídas pelo Sonar de forma automatizada. A métrica RFC é baseada nos princípios de Chidamber e Kemerer (1994). Já a métrica de complexidade segue os princípios de McCabe (1976). As demais métricas[3] listadas são definidas pela própria ferramenta utilizada. Essas métricas são conhecidas da comunidade de desenvolvimento de software e amplamente utilizadas por projetos na linguagem Java. A descrição das métricas permaneceu em inglês, pois o sistema utilizado para aferição adota essa nomenclatura.

[2]Disponível em: http://downloads.sonarsource.com/sonarqube/sonar-3.7.4.zip
[3]http://docs.sonarqube.org/display/SONAR/Metric+Definitions

Tabela 3.1: Métricas selecionadas e forma de avaliação.

Métrica	**Avaliação**
RFC	Menor melhor.
Complexity	Menor melhor.
Complexity per Class	Menor melhor.
Complexity per Method	Menor melhor.
Coverage	Maior melhor.
Line Coverage	Maior melhor.
Uncovered Lines	Menor melhor.
LoC	Menor melhor.
Lines	Menor melhor.
Violação *Blocker*	Menor melhor.
Violação *Critical*	Menor melhor.
Violação *Major*	Menor melhor.
Violação *Minor*	Menor melhor.
Violação *Info*	Menor melhor.
Rules Compliance	Maior melhor.

Qualidade Externa

A qualidade externa foi aferida por meio de um conjunto de testes automatizados que valida o código fonte criado pelos desenvolvedores, tentando identificar defeitos por meio de várias requisições para os métodos e classes criadas no experimento.

O autor construiu um conjunto de 24 casos de testes para validar o código fonte gerado pelos desenvolvedores nos dois exercícios propostos. Esse conjunto de testes foi criado baseada na técnica de teste funcional e estrutural e executados automaticamente. O código fonte dos casos de teste estão disponíveis no GitHub[4] mantido pelo autor.

Para validar os parâmetros de entrada dos métodos, os testes foram baseados na técnica funcional, mais especificamente no critério da análise de valor limite, que auxilia na seleção de um pequeno conjunto de casos de teste, mantendo uma boa cobertura de código (DELAMARO; MALDONADO; JINO, 2007). Segundo Pressman (1997), no critério da análise de valor limite, ao invés de selecionar-se qualquer elemento de uma classe, os casos de teste são escolhidos nas fronteiras das classes, pois nesses pontos se concentra um grande número de erros. Nessa categoria de teste, foram criados 17 casos de teste automatizados para o exercício.

Já na validação das regras de negócio implementadas pelos desenvolvedores, os testes foram baseados na técnica estrutural, usando o critério baseado em fluxo de controle, mais especificamente o critério Todos-Nós. Nessa categoria de teste, foram criados 7 casos de teste automatizados para o exercício.

O critério Todos-Nós exige que a execução do programa passe, ao menos uma vez, em cada vértice do grafo de fluxo, ou seja, que cada comando do programa seja executado pelo menos uma vez (DELAMARO; MALDONADO; JINO, 2007).

As unidades de medidas usadas para aferir os resultados estão listadas a seguir:

[4]https://github.com/wbissi/thesis

1. **Quantidade de Casos de Teste que Falharam:** Quantidade de defeitos encontrados por meio do conjunto de testes automatizados executados. Essa métrica é contabilizada de acordo com os casos de teste que falharam no conjunto de testes executado.

2. **Quantidade de Casos de Teste com Sucesso:** Quantidade total de casos de teste que obtiveram sucesso durante a execução do conjunto de testes automatizados.

3. **Densidade de Defeitos:** Quantidade de Casos de Teste que falharam dividido pela Quantidade de Linhas de Código produzida.

A forma de medição dessa métrica foi semi-automatizada. O conjunto de teste executado no código fonte escrito é automatizada e constituída de testes unitários automatizados utilizando o arcabouço TestNG. A contagem de casos de teste que falharam ou obtiveram sucesso foi feita pela ferramenta juntamente com a IDE Eclipse. O cálculo de densidade de defeitos foi feito manualmente, dividindo a quantidade de defeitos encontrados dividido pela Quantidade de Linha de Código (LoC).

Produtividade

Em geral, a produtividade é definida como a quantidade de trabalho realizado durante um certo período de tempo em uma atividade específica (FUCCI et al., 2015). Desse modo, a produtividade dos desenvolvedores foi aferida cronometrando o tempo gasto no desenvolvimento completo dos exercícios e também comparando o tempo dispendido com a quantidade de código fonte gerado. As duas métricas que foram utilizadas para consolidar esses dados são apresentadas a seguir:

1. **Tempo de Implementação da Funcionalidade:** Tempo gasto pelo desenvolvedor para implementar e testar as todas as funcionalidades propostas em cada exercício.

2. **Quantidade de Linhas de Código por Minuto:** Quantidade de linhas de código escrita por minuto, aferida ao fim da implementação de cada exercício. Esse cálculo é feito considerando o valor total de minutos gasto para a implementação de cada exercício, dividido pela quantidade de linhas de código escritas no programa, considerando o código de testes automatizados.

A forma de medição dessa métrica foi semi-automatizada, pois o tempo de implementação da funcionalidade foi obtido medindo o tempo de início até o fim do desenvolvimento de cada exercício, utilizando um cronômetro. A soma total de linhas de código foi obtida de forma automatizada pela métrica LoC, explicada na Seção 3.2.2. O cálculo do número de linhas de código por minuto foi feito por meio da divisão do tempo dispendido em minutos, pela quantidade total de linhas gerada, obtido pela métrica LoC.

3.2.3 Limitações do Experimento

O experimento fez uso da linguagem Java e os arcabouços desta tecnologia para a implementação dos exercícios propostos. Portanto os resultados obtidos não podem ser generalizados para outras linguagens ou tecnologias.

O tamanho do projeto implementado pelos desenvolvedores é pequeno e não ultrapassou 500 novas linhas de código. Contudo, esses exercícios exemplificam o desenvolvimento

de métodos dos serviços web e a interação entre eles, comumente utilizada no desenvolvimento de aplicações dentro da arquitetura orientada a serviços. Uma vez que o experimento foi executado com aplicações pequenas e o tamanho da aplicação desenvolvida pode influenciar nos resultados obtidos, os resultados do experimento realizado não podem ser generalizados para todos os tamanhos de aplicações.

As métricas de código utilizadas na comparação dos resultados obtidos do experimento são provenientes do paradigma orientado a objetos. O uso de métricas e paradigmas diferentes podem levar a outros resultados. Optou-se por utilizar essas métricas por serem mais difundidas e, portanto, mais utilizadas na literatura, facilitando a comparação com o referencial teórico encontrado na fundamentação teórica.

As métricas relacionadas à produtividade podem variar de acordo com o experiência do desenvolvedor em alguma tecnologia utilizada no experimento. Para diminuir essa variação, os grupos foram montados com desenvolvedores que possuíam o tempo de experiência aproximados.

3.3 Entrevista

Entrevistas são úteis quando se quer obter informações que estão internalizadas no indivíduo e que dizem respeito a experiências vividas ou tendências futuras (VERGARA, 2013). Por isso o formato de entrevista foi escolhido por possibilitar a coleta de informações e expressões do entrevistado e para conduzir uma pesquisa qualitativa acerca das abordagens WS-TDD e TLD.

A estrutura de uma entrevista individual poder ser: fechada, semiaberta e aberta (VERGARA, 2013). O formato das perguntas e respostas que compõem esta entrevista foi baseado nos modelos apresentados e discutidos em Vergara (2013). O formulário usado para a entrevista está detalhado no Apêndice D. Já o Apêndice G mostra um formulário respondido por um dos participantes da entrevista.

As questões 3 até a 5 seguem a estrutura fechada, as demais seguem a estrutura semiaberta, possibilitando o entrevistado revelar não só sua opinião acerca das abordagens, como também detalhar alguma informação que achar mais relevante.

O objetivo da entrevista é mapear a opinião dos desenvolvedores que participaram do experimento e obter evidências (positivas ou negativas) sobre a utilidade da abordagem WS-TDD comparando-a com TLD. Com as respostas das perguntas pretendeu-se mapear pontos que dificultaram o entendimento da abordagem WS-TDD e identificar, de acordo com a opinião de cada desenvolvedor, qual das abordagens mais os ajudaram na implementação dos exercícios propostos e qual delas os desenvolvedores optariam por utilizar no próximo projeto de software.

A entrevista foi realizada individualmente na mesma sala em que o experimento foi executado, logo após a execução do experimento. Cada resposta foi armazenada digitalmente em um documento de texto, que pode ser aberto por qualquer editor de texto avançado, como por exemplo o Microsoft Word.

3.3.1 Limitações da Entrevista

Para evitar as limitações de comunicação na entrevista, as questões formuladas são curtas e não exigem muita reflexão do entrevistado.

Para evitar que o entrevistador influenciasse, consciente ou inconscientemente, as respostas dos entrevistados - por suas opiniões ou pelo cargo que ocupa - outra pessoa foi treinada pelo pesquisador para realizar a pesquisa com os participantes do experimento. A pessoa que foi treinada para realizar a entrevista possui conhecimento em desenvolvimento de software porém não trabalha na mesma equipe das pessoas que participaram do experimento.

3.4 Considerações Finais do Capítulo

Este capítulo descreveu o método de pesquisa utilizado, as atividades executadas, os métodos de coleta e a forma de análise dos dados para cada atividade de pesquisa. A Figura 3.1 mostra as principais atividades executadas em todo o processo de pesquisa do mestrado.

Os critérios adotados para a análise de cada uma das atividades (Revisão Sistemática, Pesquisa Quantitativa, Experimento e Entrevista) foram detalhados nas seções anteriores. Além disso, as limitações de cada atividade foram descritas no final de cada seção.

O próximo capítulo detalha a abordagem teórica criada neste trabalho, denominada WS-TDD. A WS-TDD é voltada para a construção de serviços web e divide-se em 3 fases: Contrato, Teste Caixa Branca e Teste Caixa Cinza que são detalhadas a seguir.

Capítulo 4

Abordagem WS-TDD

A abordagem WS-TDD, proposta neste trabalho, é uma extensão da prática de TDD orientada para o desenvolvimento e teste de serviços web em SOA. A WS-TDD é uma abordagem conceitual apoiada por ferramentas e foi criada com base nos princípios que regem a prática de TDD, juntamente com as necessidades encontradas no levantamento bibliográfico sobre testes de serviços web, na experiência profissional do autor e nas respostas do questionário de pesquisa respondido por 116 pessoas que atuam diretamente com o desenvolvimento de WS. Os detalhes do questionário de pesquisa estão no Capítulo 3 e no Apêndice C.

Os dois principais objetos de estudo que compõem a WS-TDD são: o desenvolvimento de serviços web e a prática de TDD. Estes objetos foram escolhidos com base nas observações feitas pelo autor sobre as dificuldades e limitações da adoção de TDD durante o desenvolvimento de serviços web em SOA.

Dentro do processo de desenvolvimento de software, a WS-TDD restringe-se ao seguinte escopo:

1. Metodologia: Ágil, usando TDD;
2. Arquitetura: SOA;
3. Componente da Arquitetura: Serviços Web baseado em SOAP;
4. Linguagem de Programação: Java;
5. Fase do ciclo de vida SOA: Desenvolvimento de Serviços;
6. Perspectiva dos envolvidos: Desenvolvedores e Integradores de Serviços;
7. Nível de Teste: Unitário e Integrado;
8. Técnicas de Teste: Caixa Branca e Caixa Cinza.

Ainda sobre a WS-TDD, vale ressaltar que:

- A WS-TDD não é uma ferramenta, mas sim uma abordagem apoiada por ferramentas que deve ser usada para a construção de serviços web auxiliando nos testes unitários e integrados nos casos em que os serviços possuem dependências internas ou externas.

- A construção dessa abordagem foi baseada nas motivações detalhadas no início deste trabalho, na revisão da literatura, nas evidências obtidas pelo questionário e na experiência profissional do autor.

4.1 Detalhamento da WS-TDD

A WS-TDD reúne um conjunto de conceitos e informações que descrevem como o desenvolvedor poderá realizar a implementação de serviços web, com foco na prática de TDD. Essa abordagem compreende dois cenários comuns no desenvolvimento de serviços web. O primeiro é quando o desenvolvedor está codificando seu próprio WS e não tem dependência de nenhum outro WS, neste caso ele cria somente o teste unitário (caixa branca). Isso é possível porque ele tem o controle de todo o código fonte.

O segundo cenário é quando o desenvolvedor está codificando seu próprio WS mas tem dependência de um ou mais WS. Neste caso ele cria o teste unitário (caixa branca) e simula a dependência dos outros WS criando um teste integrado (caixa cinza). Neste último cenário pode não ser possível realizar o teste caixa branca da dependência por que ela pode ser um artefato de terceiros do qual o desenvolvedor não terá acesso ao código fonte.

O objetivo dessa abordagem conceitual é guiar o desenvolvedor no projeto, desenvolvimento e teste de serviços web, trazendo os benefícios da prática de TDD. O processo que apoia a abordagem WS-TDD possui 3 fases: Contrato, Teste Caixa Branca e Teste Caixa Cinza. Cada fase é detalhada com entradas, ferramentas/técnicas e saídas. Todas as fases de teste desta abordagem são suportadas pelo fluxo de TDD e contemplam as técnicas de teste caixa branca e caixa cinza.

Os arcabouços (em inglês, *frameworks*) sugeridos para auxiliar na aplicação dos conceitos descritos nesta abordagem são voltados para a linguagem de programação Java. Para outras linguagens deve-se buscar as implementações equivalentes dos arcabouços aqui apresentados. É importante ressaltar que a WS-TDD é composta por um referencial teórico que direciona o desenvolvedor durante as etapas de implementação de um serviço web.

A Figura 4.1 representa de modo resumido a abordagem WS-TDD. A Fase 1 é composta pela Especificação e Definição do Contrato do WS. Na Fase 2 o Teste Unitário (Caixa Branca) e a Implementação são realizadas seguindo o Fluxo TDD. Por fim, na Fase 3 o Teste Integrado (Caixa Cinza) e a Implementação são realizados seguindo o Fluxo TDD, finalizando o WS. Durante a implementação do WS nas Fases 2 e 3, os testes sempre são criados antes do código do WS. O retângulo localizado na parte inferior da Figura 4.1 mostra o fluxo de TDD que deve ser seguido durante a implementação do WS na Fase 2 e 3 da abordagem.

O diagrama de atividade da Figura 4.2 apresenta o fluxo das atividades que são desenvolvidas em cada uma das fases do processo que suporta a abordagem WS-TDD. O fluxo apresentado neste diagrama refere-se à construção de um método WS desde o início (definição do contrato) até o fim do desenvolvimento (implementação das funcionalidades e dos testes caixa cinza). O fluxo apresentado deve ser repetido para cada novo método WS. As próximas seções farão o detalhamento de cada uma das três fases.

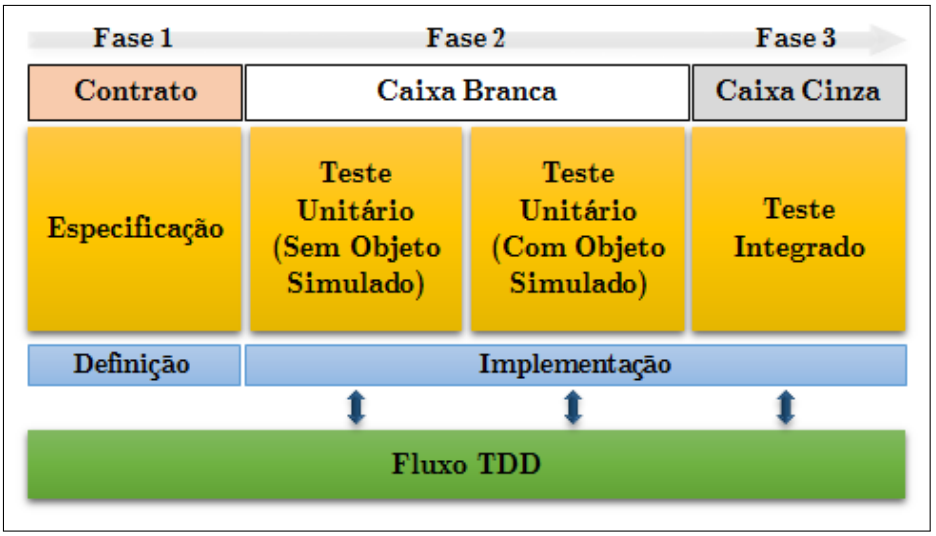

Figura 4.1: Representação da abordagem WS-TDD. Fonte: Autoria própria.

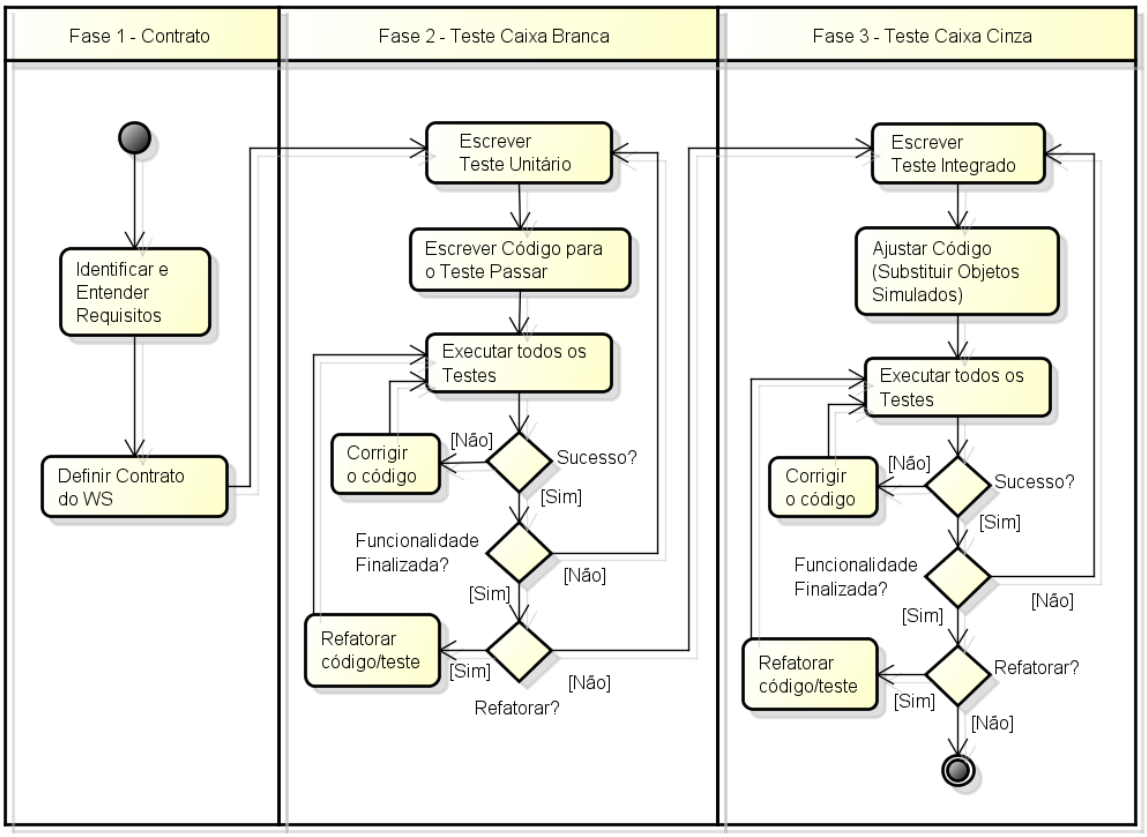

Figura 4.2: Diagrama de atividades da abordagem WS-TDD. Fonte: Autoria própria.

4.1.1 WS-TDD Fase 1 - Contrato

É a fase de preparação para o desenvolvimento do WS e dos testes. Nesta fase as atividades de detalhamento das pré-condições, dos parâmetros de entrada, das exceções que devem ser tratadas, das pós-condições e dos parâmetros de saída do método do serviço web devem ser realizadas. Essa sequência de atividades é inspirada na abordagem *Design by Contract* (DbC) descrito em Meyer (1992). A DbC direciona o desenvolvedor a definir especificações formais, precisas e verificáveis para as interfaces dos componentes (MEYER, 1992). A Figura 4.3 apresenta o fluxo resumido das etapas presentes na abordagem DbC.

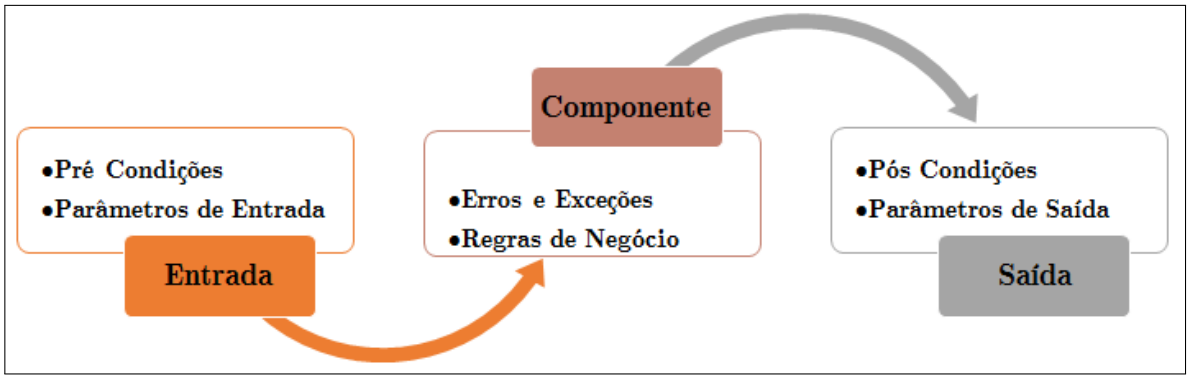

Figura 4.3: Representação resumida das etapas da abordagem DbC, adaptado de Meyer (1992).

O objetivo dessa fase é aprimorar e o contrato do componente que está sendo criado, induzindo o desenvolvedor a pensar em como o seu componente será utilizado posteriormente e não somente em como ele será implementado. Depois da definição do contrato do serviço web, as regras de negócio devem ser definidas para dar início à escrita dos testes unitários automatizados.

Um editor de XML deve ser utilizado para auxiliar o desenvolvedor na definição do contrato do serviço web. Isso permitirá ao desenvolvedor criar o primeiro rascunho do WSDL a ser desenvolvido e ir evoluindo-o a medida em que as entradas, descritas na Figura 4.4, são aprimoradas.

A Figura 4.4 apresenta as principais entradas, ferramentas/técnicas e saídas desta fase. Pode-se observar que são esperados os parâmetros, as pré condições, as pós condições, as exceções e as regras como entrada. Em seguida a ferramenta e abordagem utilizadas são: um editor XML e a abordagem DbC para manipular as entradas. Por fim, o contrato WSDL do serviço web será a saída desta fase.

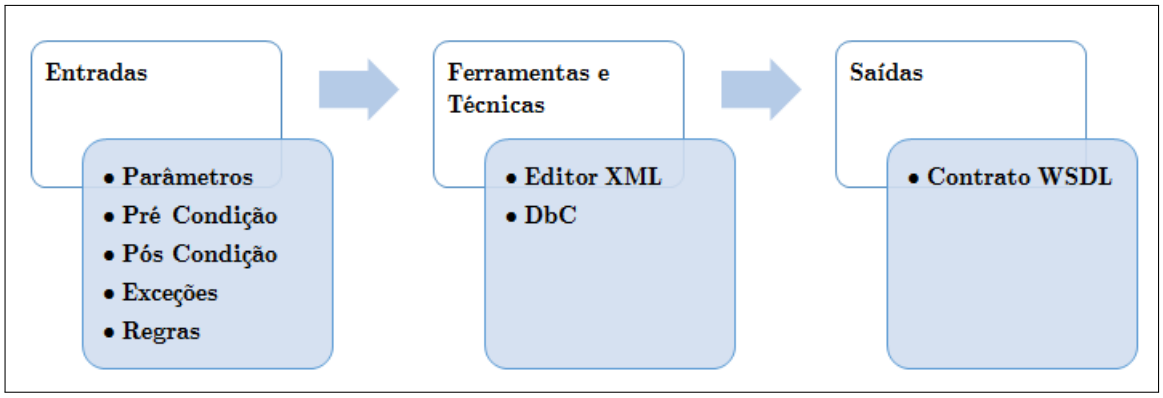

Figura 4.4: Principais atividades da Fase 1. Fonte: Autoria própria.

4.1.2 WS-TDD Fase 2 - Teste Caixa Branca

Esta é a fase mais importante da abordagem. Nela o desenvolvimento do serviço web é feito de maneira incremental, criando os testes e em seguida o código fonte. Caso necessário, o código e o teste devem ser refatorados para melhorar a qualidade interna. Nesta fase é esperado que o desenvolvedor crie os testes unitários em duas etapas: a primeira sem objeto simulado (em inglês, *mock*), validando a menor unidade do componente, e a segunda com objeto simulado para testar a interação entre os componentes.

A Figura 4.5 ilustra o conceito de teste unitário, com e sem objeto simulado. As caixas retangulares no início da figura são as classes de testes automatizados, os círculos representam os componentes de software implementados (depois da escrita dos testes), os retângulos tracejados representam os objetos simulados criados para simular algum comportamento externo, como por exemplo o acesso a um serviço web de um parceiro externo.

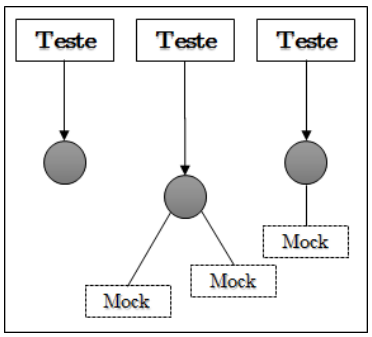

Figura 4.5: Representação dos testes unitários. Fonte: Autoria própria.

Na **primeira etapa** o desenvolvedor inicia o desenvolvimento da classe de teste automatizada, pensando nos parâmetros de entrada e saída e nas primeiras regras que ele deverá implementar no método do serviço web para uma determinada funcionalidade. Nesta etapa, o

JUnit[1] ou o TestNG[2] são as ferramentas sugeridas para auxiliar o desenvolvedor na criação do teste unitário automatizado.

Durante a implementação das classes de teste, o fluxo TDD deve ser seguido para o desenvolvimento incremental da funcionalidade. O foco desta etapa é criar o teste para validar a menor parte do componente, ou seja, os testes unitários que não possuem dependência de outros objetos. Caso o desenvolvedor necessite integrar o seu teste unitário com algum outro objeto complexo, essa etapa é finalizada e a segunda etapa deve ser iniciada.

Na **segunda etapa** o desenvolvedor incrementa o desenvolvimento do teste unitário feito na etapa anterior. Nela o desenvolvedor passa a pensar nas interações e dependências que seu teste terá com outros objetos para que seja possível executá-lo completamente. No caso das dependências técnicas como, por exemplo, um servidor de aplicação, o arcabouço Arquillian[3] pode ser usado para simulá-lo, não sendo necessário a instalação local de um servidor de aplicação, poupando tempo e recursos.

Nos pontos em que existe dependência de objetos complexos ou serviços externos ao contexto da aplicação que está sendo desenvolvida, o desenvolvedor deverá utilizar objetos simulados para isolar essa dependência. O objeto simulado é um componente fictício que possui as características de um objeto real.

O desenvolvedor deve configurar o objeto simulado de acordo com o comportamento esperado e possibilitar a interação entre esses objetos, satisfazendo as dependências dos objetos complexos. Para este cenário os arcabouços de simulação de objetos como o EasyMock[4], Mockito[5] e PowerMock[6] devem ser utilizados para facilitar a implementação. Durante a implementação das classes de teste, o fluxo TDD deve ser seguido para o desenvolvimento incremental da funcionalidade.

A Figura 4.6 apresenta as principais entradas, ferramentas/técnicas e saídas desta fase. Pode-se observar que é esperado o contrato do serviço web (WSDL) e as regras de negócio como entrada, em seguida as ferramentas e técnicas utilizadas são os arcabouços de teste unitário (JUnit, TestNG), simuladores de servidor de aplicação (Arquillian), objetos simulados (EasyMock, Mockito, PowerMock), testes unitários e simulações de objetos e servidores de aplicação para manipular as entradas. Por fim, o desenvolvimento do serviço web juntamente com suas validações e o conjunto de testes unitários são a saída dessa fase.

Os testes criados até esta etapa ainda são isolados de qualquer dependência externa, tanto técnica (servidor de aplicação, objetos complexos) quanto de negócio (acesso serviço web externo). Com isso, o desenvolvedor consegue testar a interação entre os objetos, a troca de mensagens internas entre os objetos e a composição de serviços internos e externos sem a dependência real deles. Nesta etapa todos os comportamentos das dependências são simulados. As duas etapas abordadas nesta fase estão relacionadas à técnica de teste caixa branca.

4.1.3 WS-TDD Fase 3 - Teste Caixa Cinza

Após definir o contrato na Fase 1 e criar os testes unitários na Fase 2, a Fase 3 direciona o desenvolvedor para a evolução das classes de testes unitários que utilizam objetos

[1] http://junit.org/
[2] http://testng.org/
[3] http://arquillian.org/
[4] http://easymock.org/
[5] http://mockito.org/
[6] http://github.com/jayway/powermock

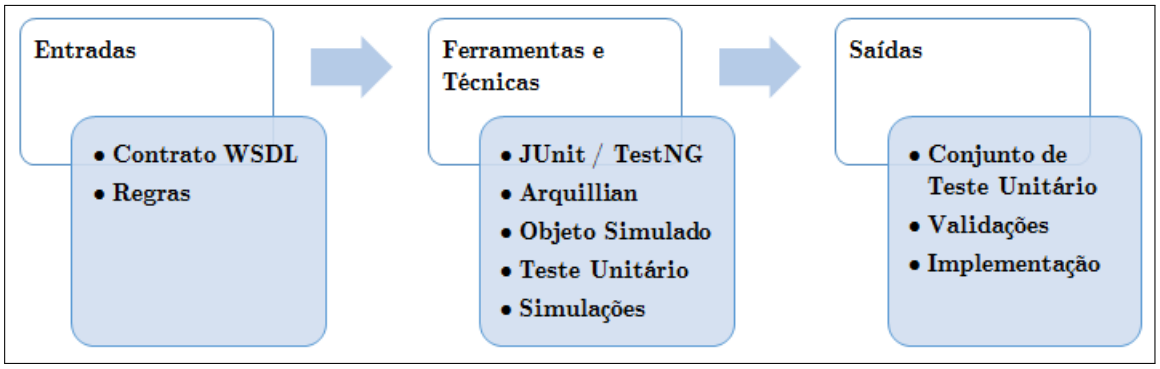

Figura 4.6: Principais atividades da Fase 2. Fonte: Autoria própria.

simulados. Nesta fase, novos testes devem ser criados com base nos testes unitários definidos na Fase 2, substituindo os objetos simulados por objetos reais, invocando os serviços reais existentes, caso não haja nenhuma restrição.

A Figura 4.7 ilustra o conceito de teste integrado com outros componentes de software. As caixas retangulares em cima da figura são as classes de testes automatizados e os círculos representam os componentes de software implementados na aplicação depois da escrita dos testes. Já o cilindro representa o banco de dados e o quadrado um módulo externo à aplicação. A ligação entre as formas dentro da figura representa a interação entre os componentes desenvolvidos e os componentes externos à aplicação.

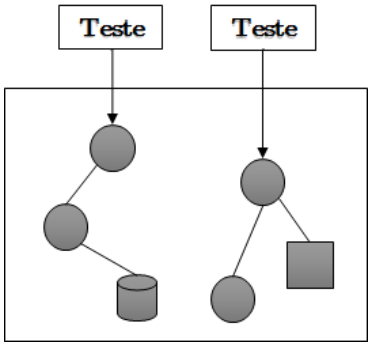

Figura 4.7: Representação dos testes integrados. Fonte: Autoria própria.

As restrições aos serviços externos podem ser com relação à inviabilidade de se utilizar objetos reais, devido à complexidade ou até mesmo restrições financeiras, como no caso da invocação de serviços externos que podem resultar em cobranças de acordo com o número de requisições realizadas. Porém, caso não haja restrições, o teste integrado automatizado deve ser criado com o objetivo de validar a troca de mensagens externas e a interação entre o componente construído e os outros serviços web que este possa vir a invocar para compor determinada funcionalidade, como no caso de composição de serviços. Este teste é voltado para a técnica caixa cinza uma vez que não existe acesso ao código dos serviços externos. Neste cenário o desenvolvedor baseia-se apenas no contrato exposto (WSDL) para a definição dos testes.

A Figura 4.8 apresenta as principais entradas, ferramentas/técnicas e saídas desta fase. Pode-se observar que é esperado como entrada o conjunto de teste unitário, as validações, a

implementação do WS e as regras de negócio. Em seguida, as ferramentas e técnicas utilizadas são os arcabouços de teste unitário (JUnit, TestNG), teste integrado dos objetos e o contrato do WS externo (WSDL) para manipular as entradas. Por fim, os ajustes na implementação do serviço web com os objetos reais e o conjunto de testes integrados são a saída dessa fase.

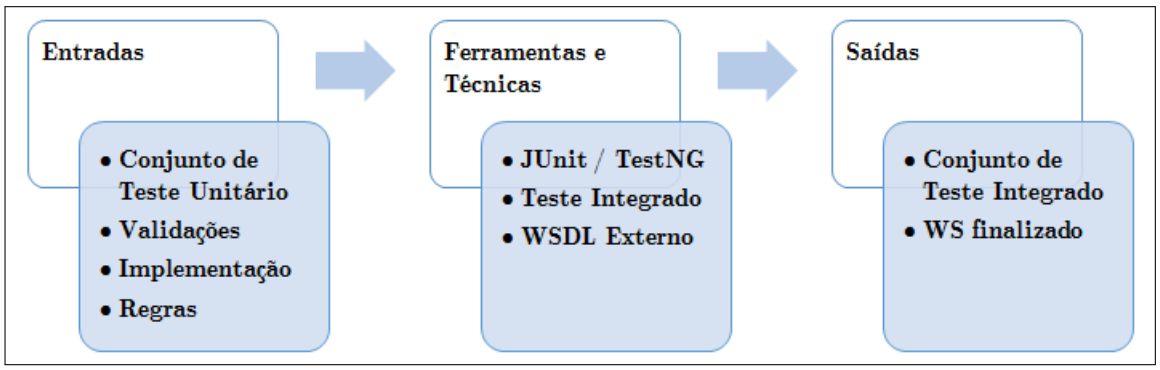

Figura 4.8: Principais atividades da Fase 3. Fonte: Autoria própria.

Para a construção do teste integrado automatizado as mesmas ferramentas citadas nas fases anteriores podem ser utilizadas. Ao fim desta fase o desenvolvedor pode refatorar o código em que está trabalhando, buscando aprimorá-lo ou então iniciar o desenvolvimento de um novo serviços web a partir de uma especificação, conforme descrito na Fase 1 - Contrato. Espera-se que ao finalizar a Fase 3 da abordagem a funcionalidade tenha sido implementada completamente.

4.1.4 Limitações da WS-TDD

A WS-TDD foi concebida para ser aplicada no desenvolvimento de serviços web baseado em SOAP dentro de uma arquitetura orientada a serviços, direcionando os desenvolvedores a utilizarem ferramentas e técnicas da linguagem Java durante a implementação da funcionalidade e dos testes unitários e integrados necessários, simplificando a adoção de TDD neste contexto. A WS-TDD não foi aplicada em nenhum outro contexto de desenvolvimento diferente do definido nesta seção. Os detalhes sobre a aplicação da WS-TDD na execução de um experimento são descritos na Seção 3.2.

Para utilizar a abordagem WS-TDD em uma outra linguagem de desenvolvimento, as ferramentas e arcabouços citados devem ser substituídos por seus equivalentes na linguagem escolhida.

A WS-TDD deve ser usada para a construção de serviços web auxiliando nos testes unitários e integrados nos casos em que os serviços possuem dependências internas ou externas. A abordagem foi projetada para suprir desenvolvimento de serviços web que atuam nos bastidores (em inglês, *back-end*), ou seja, sem interfaces diretas com o usuário. A interface com o usuário é vista aqui como um outro tipo de desenvolvimento que não faz parte do escopo da WS-TDD. Também é importante observar que a cobertura de testes usada nos experimentos para a validação da qualidade externa pode ter influenciado nos resultados apresentados no Capítulo 5.

4.2 Considerações Finais do Capítulo

Este capítulo apresentou as três fases da WS-TDD (Contrato, Teste Caixa Branca e Teste Caixa Cinza), detalhando as entradas, ferramentas e técnicas, e saídas esperadas em cada uma delas.

A Figura 4.1 mostra um visão macro da abordagem enquanto que a Figura 4.2 apresenta o fluxo das atividades para cada uma das fases da WS-TDD. A seção anterior descreveu as limitações identificadas na WS-TDD.

O próximo capítulo apresenta os resultados obtidos no questionário sobre TDD, juntamente com os resultados do experimento e da entrevista que compararam o uso da WS-TDD com a TLD. Essas atividades foram desempenhadas com profissionais que atuam em empresas locais.

Capítulo 5

Resultados

Este capítulo apresenta os resultados obtidos nos três métodos de pesquisa utilizados neste trabalho. São elas:

- Questionário;
- Experimento;
- Entrevista.

Para a aplicação dos três métodos de pesquisa, os profissionais selecionados são provenientes de amostragem não probabilística. De acordo com Vieira (2009), a amostragem não probabilística é constituída por *n* unidades reunidas em uma amostra, simplesmente porque o pesquisador tem acesso fácil a elas. Além disso, a Seção 5.4 realiza uma triangulação dos resultados obtidos somente com as pessoas que participaram dos três métodos de pesquisa citados anteriormente.

5.1 Questionário

Esta seção apresenta os resultados obtidos com o questionário de pesquisa sobre TDD, definido na Seção 3.1. A confiabilidade das respostas obtidas por meio de um questionário é uma das principais preocupações ao elaborar e aplicar um questionário. Para medir a confiabilidade do questionário aplicado, o coeficiente alfa de Cronbach foi utilizado para as 9 questões escalonadas (intervalo da questão 11 até a 19), que utilizam o Likert como escala de razão, vide Apêndice C. Para Cozby (2003), a confiabilidade de um instrumento de medição refere-se ao grau em que sua repetida aplicação, ao mesmo sujeito ou objeto, produz resultados iguais.

Para o cálculo do alfa de Cronbach as opções de respostas do questionário devem ser escalonadas (VIEIRA, 2009). Por este motivo, as outras 10 questões de múltipla escolha (intervalo da questão 1 até a 10), sem escala, não foram consideradas no cálculo de confiabilidade. O alfa de Cronbach é uma medida de consistência interna e é, definitivamente, a mais utilizada (VIEIRA, 2009). Por se tratar de uma medida bastante difundida e confiável, optou-se por utilizá-la para a validação deste questionário.

O questionário é considerado confiável caso o coeficiente alfa de Cronbach esteja acima de 0,70 (GLIEM; GLIEM, 2003). O coeficiente calculado para este questionário foi de 0,784. Portanto, ele é considerado confiável dentro da análise realizada. As análises realizadas

nas seções à seguir são referentes às questões 3, 5, 6, 9, 14, 15, 16, 17, 18 e 19 do questionário, as demais questões são descritas e analisadas no Apêndice F. As questões apresentadas neste capítulo foram selecionadas por serem as mais relevantes do questionário no contexto deste trabalho.

5.1.1 Conhecimento sobre TDD

Esta seção apresenta uma visão sobre o aprendizado e uso da prática de TDD pelos participantes. As questões aqui detalhadas são referentes a como o participante aprendeu TDD, onde aplica TDD e o tempo de experiência em TDD. Essas análises são referentes às questões 3, 5 e 6 do questionário.

Os participantes foram questionados sobre **há quanto tempo praticam TDD profissionalmente** (Questão 3). Referente a este questionamento, a Figura 5.1 mostra que cerca de 42,45% dos participantes disseram que nunca praticaram TDD profissionalmente. Já 23,58% deles informaram que praticam TDD de 1 a 6 meses. No intervalo de 7 a 12 meses estão 13,21% dos participantes. Na faixa de 13 a 24 meses estão outros 12,26% dos participantes. Por fim, 8,49% informaram que praticam TDD profissionalmente há mais de 25 meses.

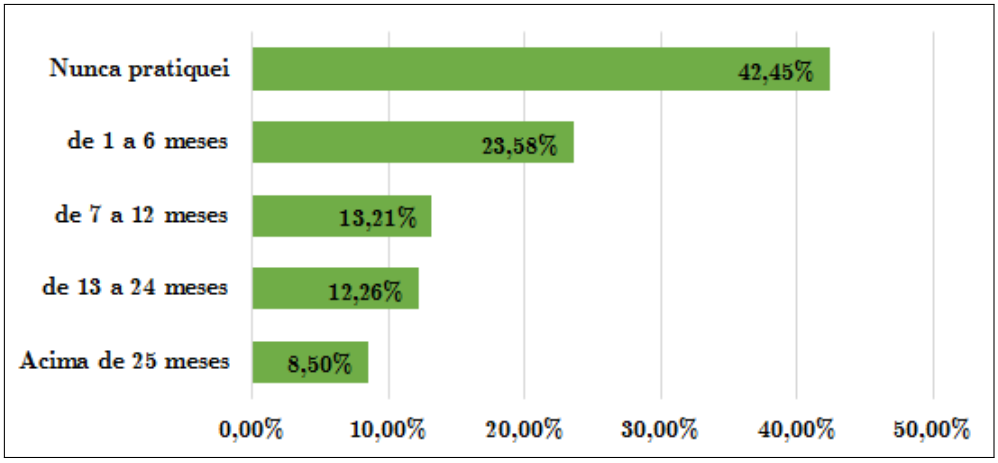

Figura 5.1: Tempo que utiliza TDD profissionalmente. Fonte: Autoria própria.

Na Questão 5, os participantes foram questionados sobre **como eles aprenderam TDD**. Conforme apresentado da Figura 5.2, a Internet, com 67,92% das escolhas, é o meio mais usado para aprender TDD; em seguida com 42,45% o Trabalho é outra fonte de aprendizado da prática. O livro teve 23,58% das escolhas, a universidade teve 20,75% das escolhas e 4,72% escolheram outra forma de aprendizado, como por exemplo conferências e *workshops*. Nesta questão é possível selecionar mais de uma resposta, portanto a soma total das opções é superior a 100%.

A Figura 5.3 mostra os **ambientes em que os participantes praticam TDD** (Questão 6), a resposta da maioria foi em projetos pessoais com 47,17%, em seguida no trabalho com 41,51%, seguido por universidade com 12,26% e projetos abertos com 5,66%. Cerca de 24,53% informaram que praticam em outros ambientes. Nesta questão é possível selecionar mais de uma resposta, portanto a soma total das opções é superior a 100%.

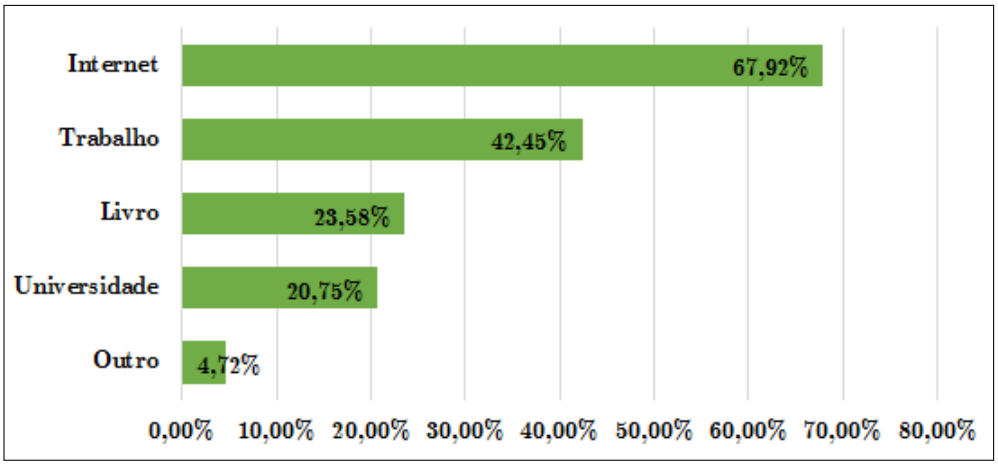

Figura 5.2: Como o participante aprendeu TDD. Fonte: Autoria própria.

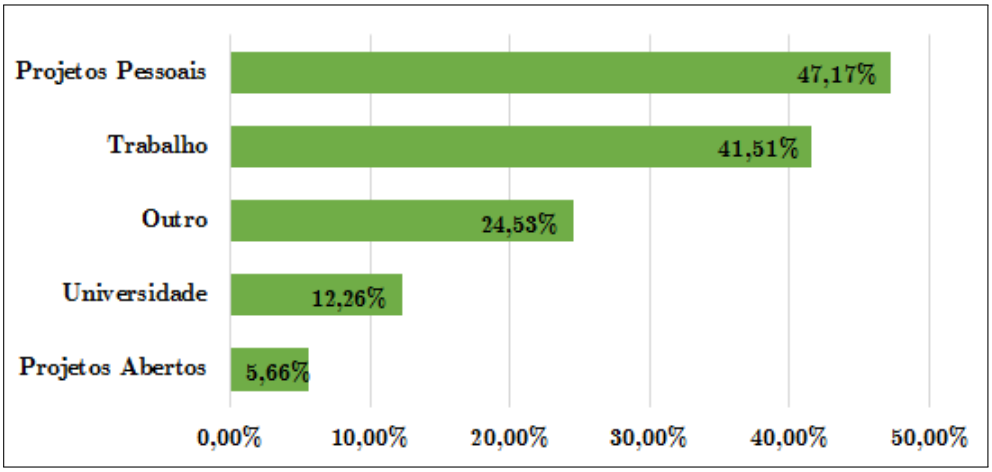

Figura 5.3: Ambiente que o participante pratica TDD. Fonte: Autoria própria.

5.1.2 Uso de Objetos Simulados nos Testes

Esta seção apresenta uma visão sobre a necessidade de usar objetos simulados durante o desenvolvimento dos testes unitários da aplicação. Essas análises são referentes às questões 14 e 15 do questionário.

A Figura 5.4 apresenta o resultado da Questão 14, que refere-se ao **uso de objetos simulados para testar *Enterprise JavaBeans* (EJB) ou WS**. Referente ao uso de objetos simulados para auxiliar a escrita de testes unitários automatizados, cerca de 12,26% disseram concordar fortemente e 33,96% disseram concordar. Já 24,53% afirmaram que é indiferente; 10,38% discordaram; 6,60% discordam fortemente e 12,27% não opinaram. O percentual de pessoas que não opinaram foi removido dos gráficos 5.4 e 5.5.

A Figura 5.5 apresenta o resultado da Questão 15, que refere-se à **necessidade de utilizar objetos simulados na criação de teste unitário**. Sobre essa necessidade, os participantes responderam da seguinte maneira: Cerca de 9,43% disseram concordar fortemente e

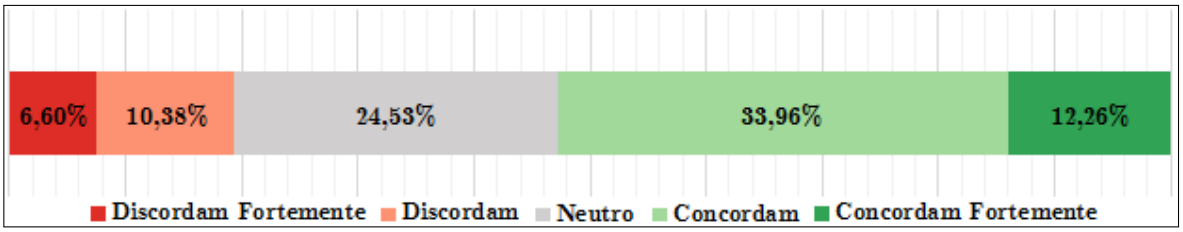

Figura 5.4: Uso de objetos simulados para testar EJB ou WS. Fonte: Autoria própria.

27,36% disseram concordar com a afirmação. Já 27,36% disseram ser indiferente; 12,26% disseram discordar; 5,66% disseram discordar fortemente e 17,93% não quiseram opinar a respeito. Considerando somente as respostas dos participantes que disseram concordar com a afirmação, o número de 36,79% revela a necessidade de usar objetos simulados para testar componentes externos.

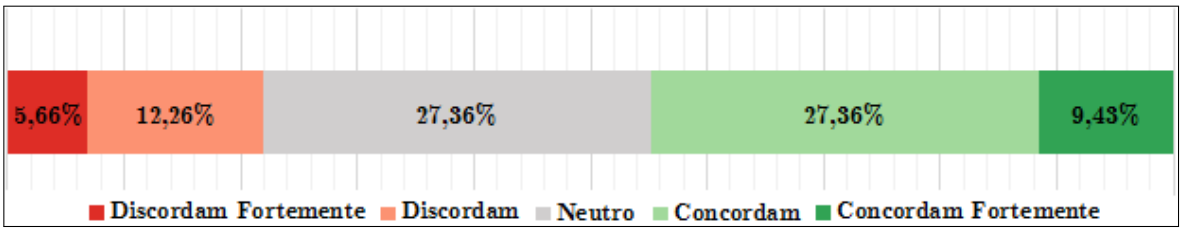

Figura 5.5: Necessidade de usar objetos simulados nos testes unitários. Fonte: Autoria própria.

5.1.3 Barreiras e Percepções sobre TDD

As análises aqui apresentadas são referentes a barreiras que os desenvolvedores encontraram ao tentar aplicar TDD na empresa, dificuldades ao adotar a prática e a percepção deles sobre os impactos de TDD quanto à redução de defeitos, qualidade de código e produtividade. Essas análises são referentes às questões 9, 16, 17, 18 e 19 do questionário.

Quando perguntado sobre **o que dificulta a adoção da prática de TDD nas empresas onde os participantes trabalham** (Questão 9), conforme apresentado na Figura 5.6, cerca de 16,04% informaram que não há dificuldade em adotar a prática. Já 36,78% informam que a falta de tempo é o principal motivo; 16,04% informaram que falta apoio da gestão para justificar o tempo investido na adoção da prática; 4,72% afirmam que ainda faltam ferramentas para atender as necessidades específicas de desenvolvimento dos testes unitários; 16,04% alegam que a falta de conhecimento técnico é o grande impeditivo. Cerca de 10,38% informaram que outras razões, como por exemplo: a complexidade do código, a ausência de padronização e o código legado, são o que dificultam a adoção da prática de TDD na empresa.

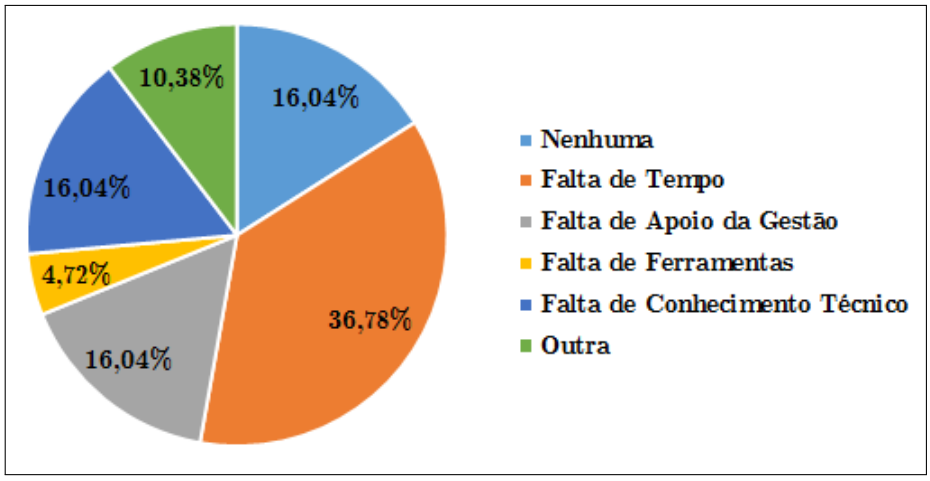

Figura 5.6: Dificuldade em adotar TDD na empresa. Fonte: Autoria própria.

A Figura 5.7 apresenta o resultado da Questão 16, que refere-se a **dificuldade em aplicar TDD pela primeira vez em um projeto**. Os resultados mostram que 24,53% concordam que não tiveram dificuldades em aplicar TDD pela primeira vez. Já 21,70% disseram que é indiferente; 33,97% discordam, ou seja, tiveram problemas ao aplicar TDD pela primeira vez e cerca de 19,80% dos participantes decidiram não opinar sobre essa questão. O percentual de pessoas que não opinaram foi removido dos gráficos 5.7, 5.8, 5.9 e 5.10.

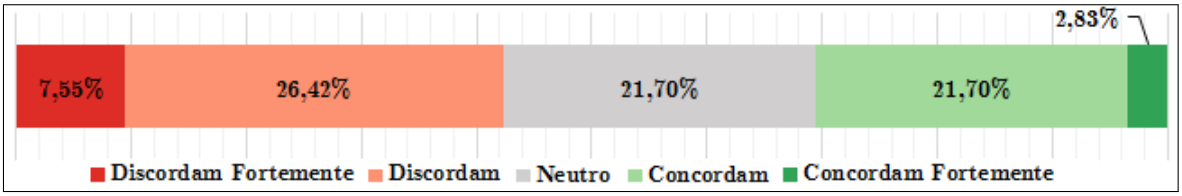

Figura 5.7: Não houve dificuldade em aplicar TDD. Fonte: Autoria própria.

A Questão 17 pergunta aos participantes se **eles acreditam que TDD ajuda a reduzir os defeitos do software**. A grande maioria, com 86,79%, concorda que TDD ajuda a reduzir os defeitos. Já 8,49% disseram que é indiferente; 2,83% discordam da afirmação, e 1,89% não opinaram a respeito. A Figura 5.8 detalha o resultado desse questionamento.

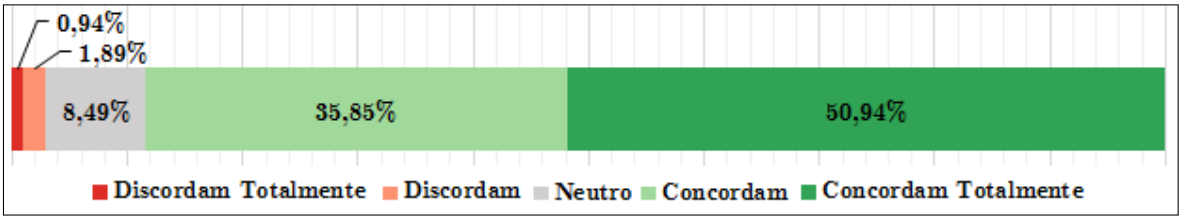

Figura 5.8: TDD ajuda a reduzir a quantidade de defeitos. Fonte: Autoria própria.

A Figura 5.9 mostra que a grande maioria com 84,91% dos participantes concorda que **TDD ajuda a melhorar a qualidade de código** (Questão 18). Já 9,43% disseram que é

indiferente; 3,77% discordam da afirmação e 1,89% não opinaram. O objetivo desta questão é identificar a percepção dos participantes sobre os efeitos que a prática de TDD gera na qualidade do código fonte final.

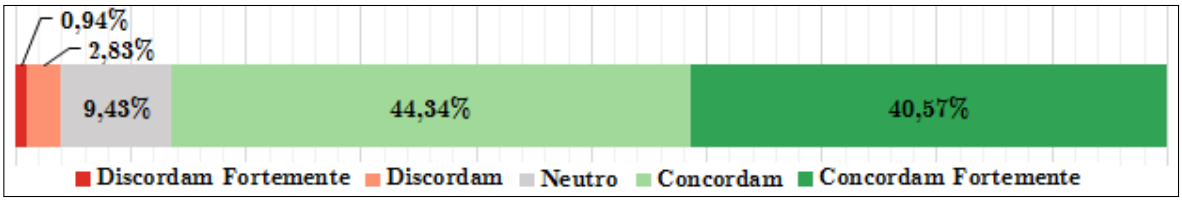

Figura 5.9: TDD melhora a qualidade do código fonte. Fonte: Autoria própria.

A Questão 19 refere-se ao **impacto de TDD na produtividade**. Neste quesito, cerca de 65,09% dos participantes concordam que TDD ajuda a aumentar a produtividade. Já 19,81% disseram que é indiferente; 10,38% discordam e 4,72% não opinaram. Conforme apresentado na Figura 5.10, as escolhas ficaram mais divididas, mas mesmo assim a maioria dos participantes acreditam que existe um aumento na produtividade ao utilizar TDD.

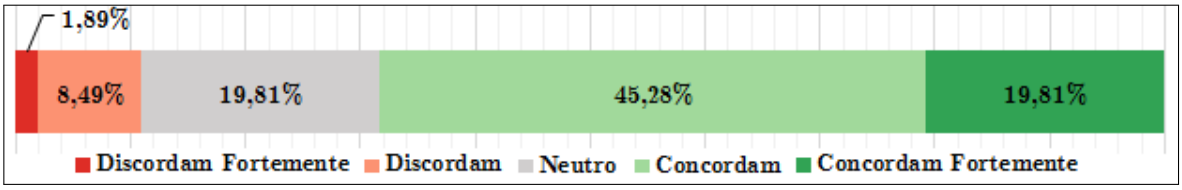

Figura 5.10: TDD pode aumentar a produtividade. Fonte: Autoria própria.

5.2 Experimento

Esta seção apresenta os resultados obtidos com o experimento, definido na Seção 3.2. Este experimento visa comparar as abordagens TLD e WS-TDD observando os resultados obtidos na qualidade interna, qualidade externa do software e na produtividade do desenvolvedor.

Os detalhes da estrutura do experimento e as métricas observadas estão descritas na Seção 3.2.1 e Seção 3.2.2, respectivamente. As análises estão separadas em dois grupos. O que diferencia um do outro é a sequência com que os exercícios foram apresentados e executados. Todos os participantes possuem conhecimento e experiência no desenvolvimento de serviços web e na linguagem Java.

Os profissionais do Grupo 1 receberam instruções da abordagem TLD e em seguida executaram o Exercício 1 (descrito na Seção E.1). Após finalizarem o Exercício 1, foi apresentada a abordagem WS-TDD e em seguida executaram o Exercício 2 (descrito na Seção E.2). Ao todo 12 pessoas participaram do Grupo 1. Já os profissionais do Grupo 2 receberam instruções da abordagem WS-TDD e em seguida executaram o Exercício 2. Após finalizarem o Exercício 2, foi apresentada a abordagem TLD e em seguida executaram o Exercício 1. Ao todo 11 pessoas participaram do Grupo 2.

5.2.1 Qualidade Interna

A Tabela 5.1 apresenta a média dos resultados obtidos no experimento realizado com os 23 participantes, relacionados a qualidade interna do software desenvolvido. As métricas consideradas estão detalhadas na Seção 3.2.2. A coluna **Var.** apresenta a variação do resultado comparando a WS-TDD com a TLD. O símbolo (+) significa que o resultado obtido com a WS-TDD foi melhor se comparado a TLD. Já o símbolo (-) significa que o resultado obtido com a WS-TDD foi pior se comparado a TLD.

Tabela 5.1: Resultados obtidos referente a qualidade interna do software.

Métrica	**Abordagem**	**Grupo 1**	**Grupo 2**	**Geral**	**Var.**
RFC	TLD	6,9	7,4	7,1	-
	WS-TDD	7,6	8,9	8,2	
Complexity	TLD	40,3	31,7	36,0	+
	WS-TDD	26,8	27,5	27,2	
Complexity per Class	TLD	6,8	6,0	6,4	-
	WS-TDD	6,3	6,7	6,5	
Complexity per Method	TLD	4,6	3,7	4,2	+
	WS-TDD	4,1	3,8	3,9	
Coverage	TLD	66,84%	58,43%	62,63%	+
	WS-TDD	75,63%	73,64%	74,63%	
Line Coverage	TLD	70,32%	62,61%	66,46%	+
	WS-TDD	80,56%	78,98%	79,77%	
Uncovered Lines	TLD	40,3	31,7	36,0	+
	WS-TDD	26,8	27,5	27,2	
LoC	TLD	161,3	150,4	155,8	+
	WS-TDD	109,9	113,6	111,8	
Lines	TLD	226,1	217,4	221,7	+
	WS-TDD	157,8	165,2	161,5	
Violação *Blocker*	TLD	0,6	1,0	0,8	+
	WS-TDD	0,4	0,5	0,4	
Violação *Critical*	TLD	2,3	0,2	1,2	+
	WS-TDD	0,3	0,1	0,2	
Violação *Major*	TLD	9,3	4,5	6,9	+
	WS-TDD	5,8	3,7	4,7	
Violação *Minor*	TLD	3,1	4,2	3,6	+
	WS-TDD	1,5	1,2	1,3	
Violação *Info*	TLD	15,3	5,5	10,4	-
	WS-TDD	14,3	7,5	10,9	
Rules Compliance	TLD	72,00%	75,18%	73,59%	+
	WS-TDD	80,78%	86,35%	83,56%	

A métrica RFC é relacionada ao projeto de software desenvolvido. Como pode-se observar na Tabela 5.1, a RFC aumentou quando os desenvolvedores utilizaram a WS-TDD, mas o valor ainda ficou dentro do limite máximo que, segundo Hitschfeld et al. (2006), é de 36. Quanto menor for essa métrica melhor é a estrutura da classe criada.

Conforme apresentado na Tabela 5.1, a complexidade geral do software reduziu em 24,53% ao utilizar a WS-TDD. A complexidade por classe praticamente se manteve estável e variou muito pouco quando as abordagens foram aplicadas. Ainda assim ao utilizar a WS-TDD a complexidade por classe aumentou em 1,60% aproximadamente. Já a complexidade por método diminuiu em 7% quando a WS-TDD foi utilizada. De forma geral, a WS-TDD reduziu a complexidade do código desenvolvido durante o experimento.

Com relação as métricas de cobertura de código por testes unitários, nota-se que a WS-TDD foi mais efetiva e trouxe resultados positivos nas três métricas. A cobertura geral do código aumento em 12,00%; o percentual de linhas de código cobertas por um teste aumentou em 13,31% e a quantidade de linhas de código descobertas caiu de 21,47 para 8,41 em média. A WS-TDD aumentou a cobertura de código e diminui o número de linhas de código descobertas.

Em geral, a quantidade de LoC e *lines* escritas pelos desenvolvedores reduziram ao utilizarem a WS-TDD, sendo que para LoC houve uma redução de 28,28% e para *lines* a redução foi de 27,16%. Os resultados mostram que ao utilizar a WS-TDD os desenvolvedores escreveram menos linhas de código quando comparada a TLD.

Ao utilizar a WS-TDD as violações de regras categorizadas como *Blocker*, *Critical*, *Major* e *Minor* tiveram reduções. Somente a violação *Info* teve um leve aumento, porém está é a violação mais branda dentre as cinco listadas, conforme explicado na Seção 3.2.2. A Figura 5.11 mostra o total de violações encontradas nos experimentos, separadas por categoria, abordagem e grupo.

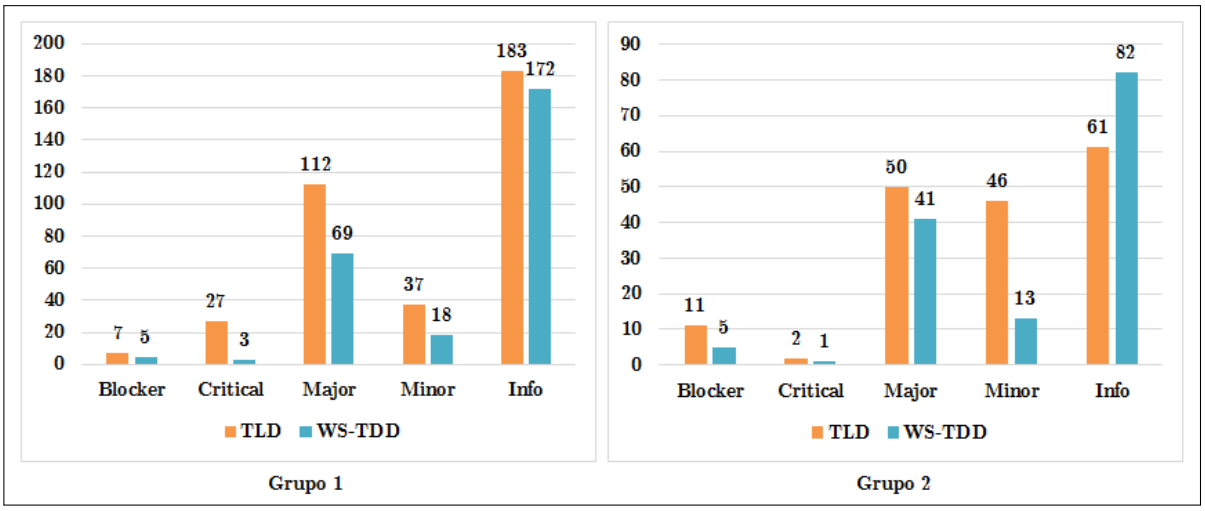

Figura 5.11: Quantidade de violações dos Grupos 1 e 2. Fonte: Autoria própria.

As violações mencionadas anteriormente geram as principais informações para a composição da métrica de *Rules Compliance*. Neste quesito, a WS-TDD também elevou o número de conformidade com as regras em aproximadamente 10% chegando a 83,56%, em média. A Figura 5.12 apresenta essa métrica comparada individualmente entre os desenvolvedores do Grupo 1, enquanto que a Figura 5.13 traz a comparação para o Grupo 2.

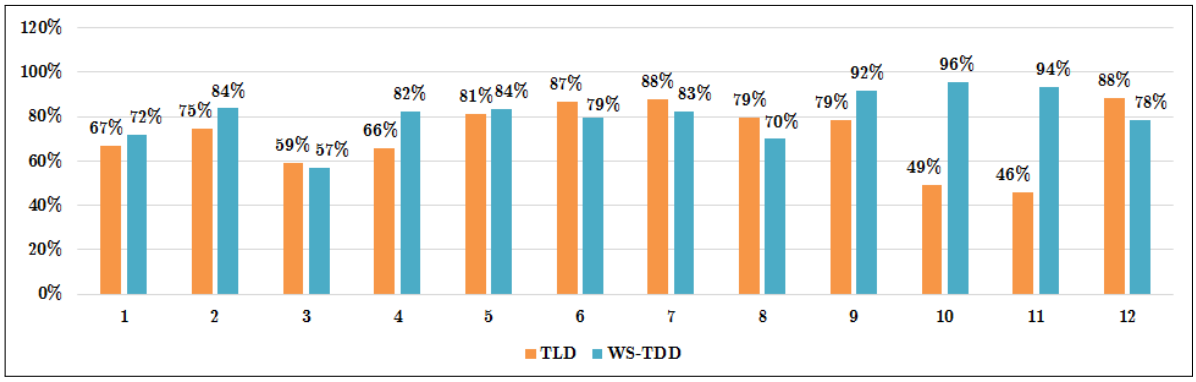

Figura 5.12: Percentual de conformidade com as regras do Grupo 1. Fonte: Autoria própria.

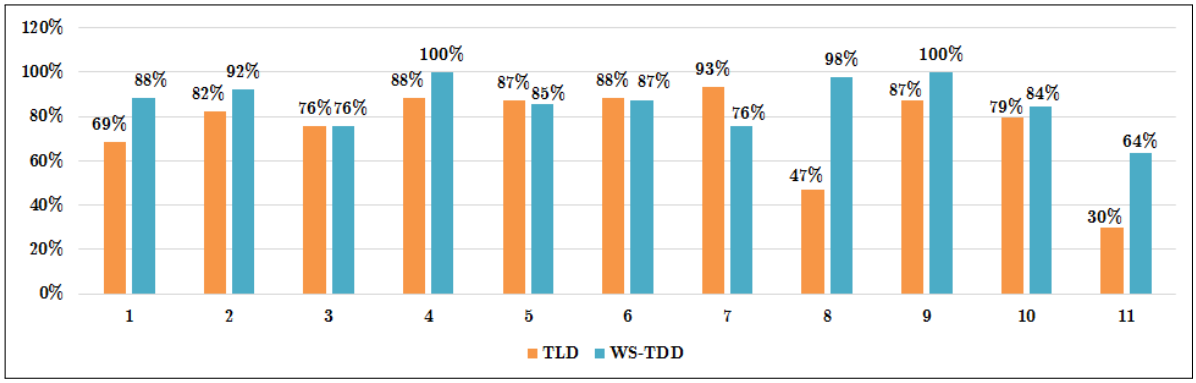

Figura 5.13: Percentual de conformidade com as regras do Grupo 2. Fonte: Autoria própria.

5.2.2 Qualidade Externa

A Tabela 5.2 apresenta a média dos resultados obtidos com os 23 participantes do experimento, relacionados à qualidade externa do software desenvolvido por cada um dos participantes. As métricas consideradas estão detalhadas na Seção 3.2.2. As métricas **Qtde. de caso de teste com falha** (quanto menor melhor) e **Qtde. de caso de teste com sucesso** (quanto maior melhor) possuem um valor entre 0 e 24. Já a **Densidade de Defeitos** (quanto menor melhor) possui um valor variável que depende da quantidade de defeitos e da quantidade de linhas de código escrita. A coluna **Var.** apresenta a variação do resultado comparando a WS-TDD com a TLD. O símbolo (+) significa que o resultado obtido com a WS-TDD foi melhor se comparado a TLD. Já o símbolo (-) significa que o resultado obtido com a WS-TDD foi pior se comparado a TLD.

A primeira métrica apresentada na Tabela 5.2 é referente aos casos de teste que falharam, ou seja, são defeitos que foram identificados. Na maior parte são validações aos parâmetros de entrada e saída que não foram implementadas. Na média, quando os desenvolvedores utilizaram WS-TDD produziram mais defeitos afetando negativamente a qualidade externa. Isso foi identificado tanto no Grupo 1 quanto no Grupo 2.

Tabela 5.2: Resultados obtidos referente a qualidade externa.

Métrica	Abordagem	Grupo 1	Grupo 2	Geral	Var.
Qtde. de caso de teste com falha	TLD	3,17	6,09	4,63	-
	WS-TDD	5,33	6,18	5,76	
Qtde. de caso de teste com sucesso	TLD	20,83	17,91	19,37	-
	WS-TDD	18,67	17,82	18,24	
Densidade de defeitos	TLD	0,0276	0,0423	0,0350	-
	WS-TDD	0,0445	0,0553	0,0499	

A segunda métrica refere-se aos casos de teste que obtiveram sucesso. Essa métrica é reflexo dos resultados obtidos na primeira. Uma vez que existem 24 casos de teste, retirando os que falharam, o restante são os que obtiveram sucesso. Como as falhas aumentaram ao usar a WS-TDD, consequentemente a quantidade de casos de teste que obtiveram sucesso diminuíram ao usar esta abordagem.

Observando as duas métricas descritas anteriormente, identificou-se que ao utilizar a WS-TDD, aproximadamente 1 caso de teste falhou a mais do que quando a TLD foi utilizada. A cobertura de testes usada pelo autor para validar o código pode ter influenciado nesse resultado, porém a mesma cobertura foi aplicada a todos os experimentos mantendo a equivalência nas análises.

Por fim, a terceira métrica é relacionada a quantidade de defeitos encontrados ao confrontá-los a quantidade de linhas de código escritas. Essa métrica também é influenciada pelos valores das métricas anteriores e como pode-se observar na Tabela 5.2, ao utilizarem a abordagem WS-TDD a densidade de defeitos aumentou, guiada pelo aumento no número de falhas e pela diminuição das linhas de códigos escritas ao utilizar esta abordagem.

5.2.3 Produtividade

A Tabela 5.3 apresenta a média dos resultados obtidos no experimento realizado com os 23 participantes, com relação a produtividade de cada um deles. As métricas de produtividade consideradas estão detalhadas na Seção 3.2.2. As métricas **Tempo de implementação** (quanto menor melhor) e **LoC por minuto** (quanto maior melhor) não possuem um intervalo de valores pré-definidos pois variam de acordo com o tempo que cada desenvolvedor levou para concluir cada exercício. A coluna **Var.** apresenta a variação do resultado comparando a WS-TDD com a TLD. O símbolo (+) significa que o resultado obtido com a WS-TDD foi melhor se comparado a TLD. Já o símbolo (-) significa que o resultado obtido com a WS-TDD foi pior se comparado a TLD.

Tabela 5.3: Resultados obtidos referente a produtividade.

Métrica	Abordagem	Grupo 1	Grupo 2	Geral	Var.
Tempo de implementação (min.)	TLD	121,33	153,00	137,17	+
	WS-TDD	97,33	125,00	111,17	
LoC por minuto	TLD	1,42	1,06	1,24	-
	WS-TDD	1,27	1,05	1,16	

A primeira métrica apresentada na Tabela 5.3 é relacionada ao tempo gasto, em minutos, para implementar as funcionalidades. A Figura 5.14 e a Figura 5.15 mostram a comparação

individual entre o tempo gasto pelos desenvolvedores durante a implementação dos exercícios utilizando a abordagem TLD e a WS-TDD. Apenas 4 desenvolvedores, dos 23 que participaram, finalizaram o exercício em menos tempo ao utilizar a TLD. Na média, quando os desenvolvedores utilizaram a WS-TDD fizeram o exercício em menos tempo do que quando utilizaram a TLD, tanto no Grupo 1 quanto no Grupo 2. Em geral, o tempo reduziu em 18,96% ao utilizar a WS-TDD se comparado com a TLD.

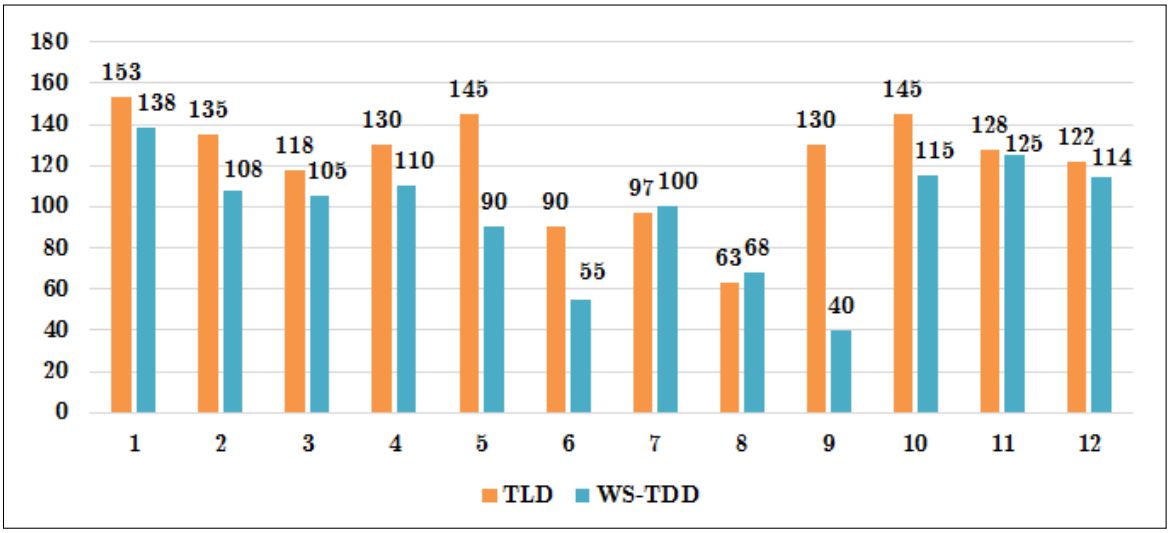

Figura 5.14: Tempos gastos pelos desenvolvedores do Grupo 1. Fonte: Autoria própria.

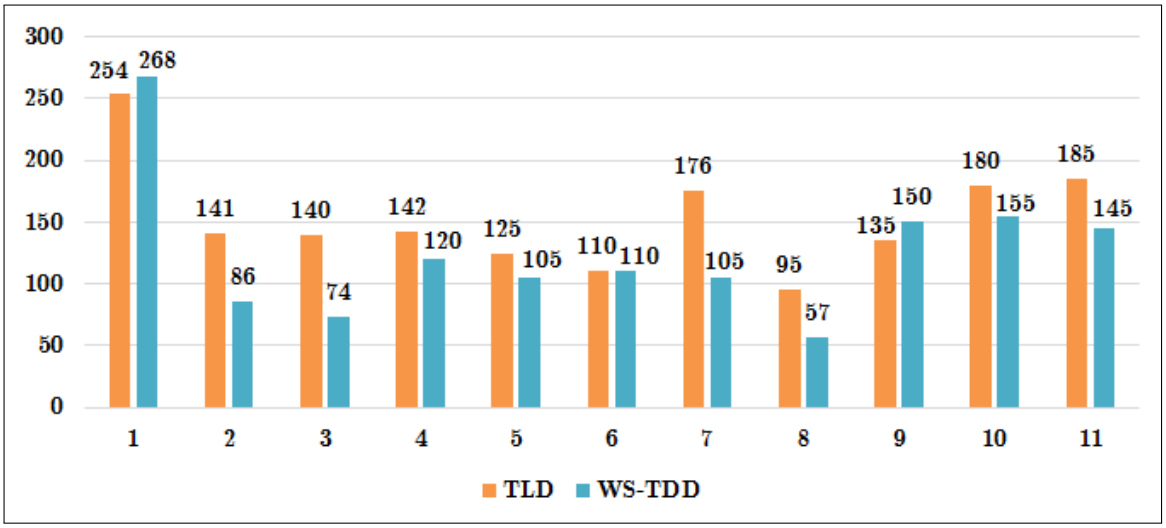

Figura 5.15: Tempos gastos pelos desenvolvedores do Grupo 2. Fonte: Autoria própria.

A segunda métrica de produtividade visa identificar a quantidade de linhas de código escritas por minuto. Ao utilizar a WS-TDD os desenvolvedores escreveram menos linhas de código do que quando utilizaram a TLD, tanto no Grupo 1 quanto no Grupo 2. Em geral, a quantidade de linhas de código reduziu em 6,45% ao utilizar a WS-TDD se comparado com a TLD.

Em nenhuma das métricas houve variações relevantes entre os resultados obtidos no Grupo 1 e no Grupo 2 dos experimentos, isso confirma que a ordem de execução e aplicação dos experimentos não influenciaram nos resultados obtidos.

5.3 Entrevista

Esta seção apresenta os resultados obtidos com a entrevista individual, definida na Seção 3.3. O foco da entrevista é obter a opinião dos profissionais que participaram do experimento sobre a abordagem TLD e a WS-TDD. O resultado da entrevista é usado como complemento ao resultado do experimento e também para aprimorar a WS-TDD.

Todos os 23 profissionais que participaram do experimento também participaram da entrevista. A Tabela 5.4 apresenta as três respostas que mais apareceram nas Questões 1 e 2, quando os entrevistados foram questionados sobre qual foi a maior dificuldade em desenvolver e testar o Exercício 1 (TLD) e o Exercício 2 (WS-TDD).

Tabela 5.4: Principais barreiras encontradas ao desenvolver os exercícios.

Abordagem	**Resposta 1**	**Resposta 2**	**Resposta 3**
TLD	Descobrir os erros tardiamente	Entendimento dos requisitos	Identificar as falhas no código
WS-TDD	Entender o novo paradigma	Pensar na abrangência do teste	Criar o teste antes do código

Ao utilizar a TLD as respostas foram relacionadas ao fato de "descobrir" os erros, requisitos e falhas durante a implementação do exercício. Os participantes se mantiveram focados em fazer funcionar o código que foi escrito inicialmente, ainda que eles não tivessem entendido totalmente o que era esperado como funcionalidade final.

Com a WS-TDD as respostas se voltaram para o entendimento do novo paradigma (método, fases e ferramentas) e principalmente pelo fato de criar os testes antes do código. Além disso, o foco dos participantes foi em desenvolver testes que tivessem a maior abrangência possível para o código que seria escrito na sequência, ou seja, aumentando a cobertura de código.

A Tabela 5.5 apresenta o resultado das questões 3 até 7. As colunas TLD, WS-TDD e Indiferente, referem-se a opção selecionada pelo entrevistado e os valores contidos nessa coluna estão relacionados a quantidade de pessoas que selecionaram essa opção. A soma dessas três colunas é 23.

Tabela 5.5: Opinião dos entrevistados sobre as abordagens.

Questão	**TLD**	**WS-TDD**	**Indiferente**
Na sua opinião, qual abordagem produz a solução mais correta em menos tempo?	1	19	3
Na sua opinião, qual abordagem produz código mais simples e com mais reúso?	0	19	4
Na sua opinião, qual abordagem produz a solução com menos defeito?	1	21	1
Qual abordagem mais te motivou a testar?	3	18	2
Qual abordagem simplificou o desenvolvimento e a escrita de testes automatizados?	1	20	2

Em todas as questões descritas na Tabela 5.5, a abordagem WS-TDD foi a opção mais selecionada, mostrando que os entrevistados acreditam no potencial da WS-TDD no que diz respeito ao desenvolvimento de uma solução correta, simplificada, com menos defeitos e que motive a criação de testes automatizados durante o desenvolvimento. A opção Indiferente apareceu em outras respostas, nas quais os entrevistados justificaram que somente com o exercício realizado não foi possível identificar diferença nos assuntos que foram questionados. Já a TLD foi a opção menos selecionada.

Todos os participantes disseram que considerariam utilizar a WS-TDD no próximo projeto de software com serviços web. As principais vantagens citadas pelos participantes ao utilizar a WS-TDD foram:

1. Mais liberdade e segurança para fazer as alterações no código (refatoração);
2. Maior compreensão dos requisitos (mais assertividade e menos retrabalho);
3. Diminuição do tempo de desenvolvimento ao usar as ferramentas e técnicas citadas (independência dos servidores de aplicação e componentes externos);
4. Melhora na qualidade de código e do sistema (maior cobertura nos testes escritos);
5. Diminuição do tempo gasto com depuração de código procurando defeitos (*debug*).

Já as principais desvantagens citadas pelos participantes ao utilizar a WS-TDD foram:

1. Adaptação à nova abordagem e ao novo paradigma;
2. A configuração das ferramentas e arcabouços pode ser demorada;
3. Resistência a mudança por parte dos desenvolvedores;
4. Em validações simples e com poucas regras de negócio, talvez não seja necessário seguir todas as fases da abordagem;
5. Usar a abordagem em software legados com tecnologias antigas.

5.4 Triangulação dos Resultados

A triangulação surge como forma de amenizar problemas de credibilidade em pesquisas, ao adotar como estratégia de investigação múltiplas visadas e métodos de obtenção de informações (AZEVEDO et al., 2013).

Esta seção faz uma triangulação dos resultados obtidos no questionário, experimento e entrevista. A triangulação dos resultados foi feita selecionando somente os profissionais que participaram dos três métodos de pesquisa executados. Devido a esta restrição o total de participantes caiu para 14. No caso do experimento e da entrevista que possuem uma subdivisão em grupos, coincidentemente, os grupos ficaram divididos igualmente com 7 participantes em cada. O foco de selecionar o mesmo profissional[1] que participou dos três métodos de pesquisa para a triangulação dos resultados foi de identificar alguma variação (positiva ou negativa) nos resultados encontrados inicialmente com as 23 pessoas.

Essa triangulação tem por objetivo verificar se o método de pesquisa aplicado influenciou nos resultados obtidos, fazendo uma comparação entre o questionário, experimento e entrevista. O foco dessa triangulação foi observar nos três métodos (questionário, experimento e entrevista) os resultados obtidos nas três categorias principais: qualidade interna, qualidade externa e produtividade. Sendo assim, essa seção apresenta os resultados comuns obtidos nessas três categorias.

A Questão 17 perguntou aos participantes se **eles acreditam que TDD ajuda a reduzir os defeitos do software**. A grande maioria com 78,57% dos participante, concordam que TDD ajuda a reduzir os defeitos. Já 14,29% disseram que é indiferente; 7,14% discordam da afirmação. A Figura 5.16 detalha o resultado desse questionamento.

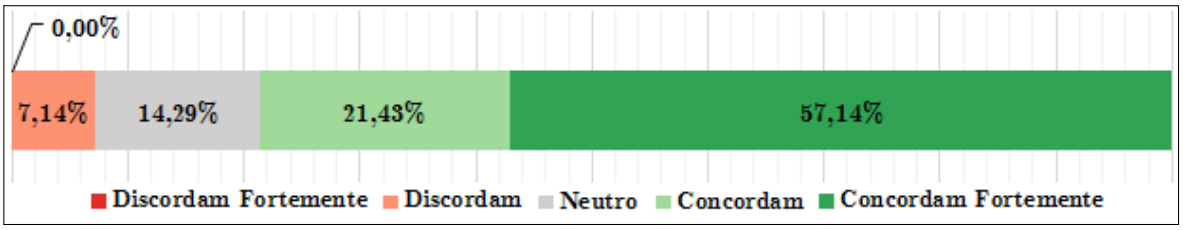

Figura 5.16: TDD ajuda a reduzir a quantidade de defeitos. Fonte: Autoria própria.

A Figura 5.17 mostra que a grande maioria dos participantes com 78,57% concordam que **TDD ajuda a melhorar a qualidade de código** (Questão 18). Já 7,14% disseram que é indiferente e 14,29% discordam da afirmação. O objetivo desta questão é identificar a percepção dos participantes sobre os efeitos que a prática de TDD gera na qualidade do código fonte.

[1]Refere-se ao mesmo profissional como a pessoa que participou do questionário, experimento e da entrevista.

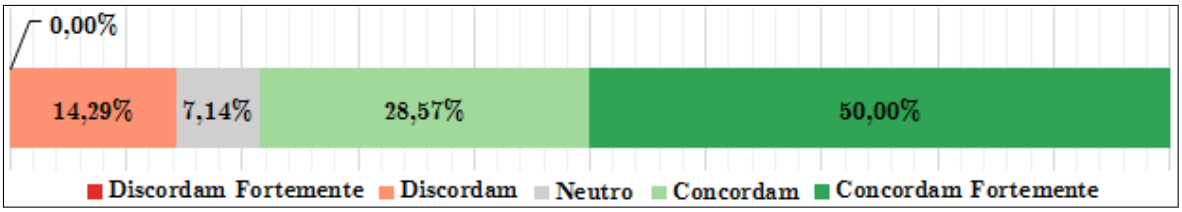

Figura 5.17: TDD melhora a qualidade do código fonte. Fonte: Autoria própria.

A Questão 19 refere-se ao **impacto de TDD na produtividade**. Neste quesito, cerca de 42,86% dos participantes concordam que TDD ajuda a aumentar a produtividade. Já 35,71% disseram que é indiferente e 21,43% discordam. Conforme apresentado na Figura 5.18, as escolhas ficaram mais divididas, mas ainda assim a maioria dos participantes acreditam que existe um aumento na produtividade ao utilizar TDD.

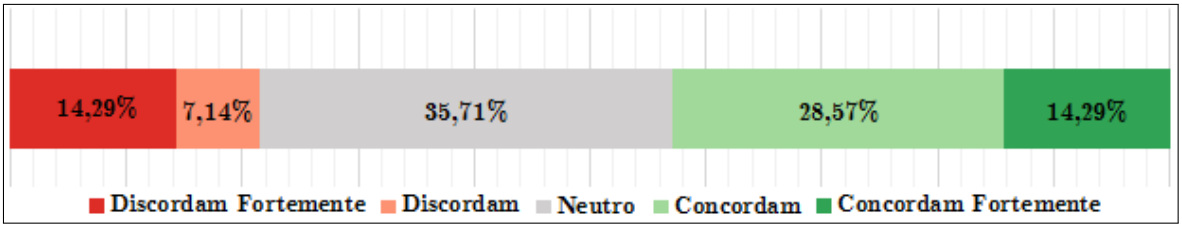

Figura 5.18: TDD pode aumentar a produtividade. Fonte: Autoria própria.

Conforme apresentado nas Figuras 5.16, 5.17 e 5.18, a grande maioria dos participantes acreditam que TDD reduz defeitos (QE), melhora a qualidade de código (QI), e aumenta a produtividade (PROD).

A Tabela 5.6 apresenta a média dos resultados obtidos no experimento com todos os 14 participantes selecionados, relacionado à qualidade interna do software desenvolvido. Os resultados mostram que ao utilizar a WS-TDD a qualidade interna aumentou, diminuindo a complexidade do código, aumentando a cobertura de testes, diminuindo a quantidade de linhas de código escrita, reduzindo as violações e aumentando a conformidade com as regras de qualidade definidas pela ferramenta Sonar. Portanto, pode-se confirmar que a qualidade interna do código aumentou, o que era esperado por 78,57% dos participantes do questionário. A coluna **Var.** apresenta a variação do resultado comparando a WS-TDD com a TLD. O símbolo (+) significa que o resultado obtido com a WS-TDD foi melhor se comparado a TLD. Já o símbolo (-) significa que o resultado obtido com a WS-TDD foi pior se comparado a TLD.

Tabela 5.6: Resultados obtidos referente a qualidade interna do software.

Métrica	**Abordagem**	**Grupo 1**	**Grupo 2**	**Geral**	**Var.**
RFC	TLD	6,9	7,0	6,9	-
	WS-TDD	7,6	8,3	7,9	
Complexity	TLD	38,6	32,6	35,6	+
	WS-TDD	29,9	23,7	26,8	
Complexity per Class	TLD	6,6	6,2	6,4	+
	WS-TDD	6,7	5,9	6,3	
Complexity per Method	TLD	4,0	3,8	3,9	+
	WS-TDD	4,1	3,5	3,8	
Coverage	TLD	70,41%	55,46%	62,94%	+
	WS-TDD	76,90%	72,84%	74,87%	
Line Coverage	TLD	75,59%	59,43%	67,51%	+
	WS-TDD	81,71%	78,19%	79,95%	
Uncovered Lines	TLD	17,9	22,6	20,2	+
	WS-TDD	8,9	8,9	8,9	
LoC	TLD	151,1	161,3	156,2	+
	WS-TDD	116,4	109,0	112,7	
Lines	TLD	219,0	234,1	226,6	+
	WS-TDD	170,7	164,4	167,6	
Violação *Blocker*	TLD	0,3	0,4	0,4	+
	WS-TDD	0,1	0,1	0,1	
Violação *Critical*	TLD	2,6	0,3	1,4	+
	WS-TDD	0,1	0,1	0,1	
Violação *Major*	TLD	10,3	3,3	6,8	+
	WS-TDD	7,4	2,6	5,0	
Violação *Minor*	TLD	5,0	6,4	5,7	+
	WS-TDD	2,4	1,9	2,1	
Violação *Info*	TLD	14,1	1,4	7,8	+
	WS-TDD	11,9	2,1	7,0	
Rules Compliance	TLD	66,13%	79,06%	72,59%	+
	WS-TDD	81,66%	91,33%	86,49%	

A Tabela 5.7 apresenta a média dos resultados obtidos no experimento com os 14 participantes selecionados, relacionado a qualidade externa do software desenvolvido. Os resultados mostram que ao utilizar a WS-TDD (baseada em TDD) a qualidade externa diminuiu, aumentando a quantidade de falhas identificadas pelo conjunto de testes executado. Dessa forma, pode-se dizer que a qualidade externa diminuiu, algo relevante e que não era esperado por este estudo e nem pela maioria dos participantes do questionário, no qual 78,57% acreditavam que a prática de TDD reduziria a quantidade de defeitos do software. A coluna **Var.** apresenta a variação do resultado comparando a WS-TDD com a TLD. O símbolo (+) significa que o resultado obtido com a WS-TDD foi melhor se comparado a TLD. Já o símbolo (-) significa que o resultado obtido com a WS-TDD foi pior se comparado a TLD.

A Tabela 5.8 apresenta a média dos resultados obtidos no experimento com os 14 participantes selecionados, relacionado a produtividade no desenvolvimento de software. Os

Tabela 5.7: Resultados obtidos referente a qualidade externa.

Métrica	Abordagem	Grupo 1	Grupo 2	Geral	Var.
Qtde. de caso de teste com falha	TLD	2,86	6,00	4,43	-
	WS-TDD	4,43	6,43	5,43	
Qtde. de caso de teste com sucesso	TLD	21,14	18,00	19,57	-
	WS-TDD	19,57	17,57	18,57	
Densidade de defeitos	TLD	0,0245	0,0386	0,0315	-
	WS-TDD	0,0293	0,0592	0,0442	

resultados mostram que ao utilizar a WS-TDD a produtividade aumentou, pois o tempo para a implementação dos requisitos foi menor e a qualidade de LoC escritas por minuto praticamente manteve-se a mesma. Pode-se dizer que a produtividade aumentou, o que era esperado por 65,09% dos participantes do questionário. A coluna **Var.** apresenta a variação do resultado comparando a WS-TDD com a TLD. O símbolo (+) significa que o resultado obtido com a WS-TDD foi melhor se comparado a TLD. Já o símbolo (-) significa que o resultado obtido com a WS-TDD foi pior se comparado a TLD.

Tabela 5.8: Resultados obtidos referente a produtividade.

Métrica	Abordagem	Grupo 1	Grupo 2	Geral	Var.
Tempo de implementação (min.)	TLD	133,00	152,57	142,79	+
	WS-TDD	106,43	127,29	116,86	
LoC por minuto	TLD	1,15	1,16	1,15	+
	WS-TDD	1,28	1,04	1,16	

Em todas as questões descritas na Tabela 5.9, a abordagem WS-TDD foi a opção mais selecionada, no que diz respeito ao desenvolvimento de uma solução correta, simplificada, com menos defeitos e que motive a criação de testes automatizados durante o desenvolvimento. A opção Indiferente apareceu em outras respostas, no qual os entrevistados justificaram que somente com o exercício realizado não foi possível identificar diferença nos itens que foram questionados. Já a TLD foi a opção menos selecionada. A soma das três colunas com os resultados é 14.

Tabela 5.9: Opinião dos entrevistados sobre o uso das abordagens.

Questão	TLD	WS-TDD	Indiferente
Na sua opinião, qual abordagem produz a solução mais correta em menos tempo?	0	13	1
Na sua opinião, qual abordagem produz código mais simples e com mais reúso?	0	11	3
Na sua opinião, qual abordagem produz a solução com menos defeito?	0	13	1
Qual abordagem mais te motivou a testar?	3	10	1
Qual abordagem simplificou o desenvolvimento e a escrita dos testes automatizados?	0	12	2

5.5 Considerações Finais do Capítulo

Este capítulo apresentou em detalhes os resultados obtidos em cada um dos métodos de pesquisa (questionário, experimento, entrevista). Os resultados obtidos com o uso de cada uma das abordagens (WS-TDD e TLD) foram comparados e separados por grupo.

Adicionalmente foi realizado uma triangulação dos resultados com as pessoas que participaram desses três métodos, criando um novo conjunto de amostragem. Os resultados sugerem que a WS-TDD aumentou a qualidade interna do software e a produtividade dos desenvolvedores. Porém, a qualidade externa do software diminuiu.

O próximo capítulo discute os resultados obtidos em cada um dos três métodos e faz uma comparação dos resultados desta pesquisa com os encontrados na literatura.

Capítulo 6

Discussão

Este capítulo discute os principais resultados obtidos neste trabalho e os comparam com trabalhos da revisão sistemática que fizeram estudos similares na indústria utilizando TDD. Os detalhes da revisão sistemática estão no Apêndice B. Além da discussão dos resultados e da literatura, as principais ameaças à validade do trabalho também são apresentadas.

6.1 Discussão dos Resultados Obtidos

Esta seção discute os resultados obtidos neste trabalho em cada umas das atividades de pesquisa realizada.

6.1.1 Questionário

Analisando as questões 3, 5 e 6 identificou-se que a divulgação da prática de TDD no meio acadêmico teve apenas 20,75%; isso pode ser considerado baixo se comparado com o aprendizado no ambiente de trabalho, que foi mais que o dobro. O ambiente que mais usado para aplicar a prática de TDD, segundos os participantes, é em Projetos Pessoais ou no Trabalho.

Cerca de 57,55% dos participantes afirmaram que praticam TDD profissionalmente durante o desenvolvimento de software. Esse número é superior aos obtidos pelas pesquisas de VersionOne (2015) (34%) e de Melo et al. (2013) (39,5%). Porém, apesar de TDD ser uma prática que tornou-se conhecida no início de 2000, somente 8,49% dos participantes utilizam essa prática há mais de 2 anos e 42,45% nunca utilizaram TDD, portanto o uso dessa prática ainda é considerado pequeno.

Ao analisar os resultados das Questões 16 até 19, fica claro que na opinião dos participantes a prática de TDD ajuda a melhorar o código fonte, contribui para a redução de defeitos e aumenta a produtividade. Deste ponto de vista, pode-se dizer que os participantes percebem ou acreditam em alguns benefícios ao utilizar TDD. A percepção dos participantes mapeada neste questionário vai de encontro com os benefícios relatados pelos autores renomados na indústria, tais como: Beck (2002) e Martin (2006).

Novas abordagens e ferramentas que simplifiquem a adoção de TDD podem contribuir para a popularização da prática. Pois, conforme apresentado na Figura 5.7, cerca de 33,97% dos participante informaram que tiveram dificuldades ao aplicar TDD. Além disso, existem barreiras e rejeições por parte das empresas ao tentar adotar TDD no desenvolvimento de software.

6.1.2 Experimento

Qualidade Interna: Como apresentado na Seção 5.2.1, a WS-TDD mostrou-se eficiente e contribuiu para o aumento da qualidade interna, diminuindo a complexidade do código, aumentando a cobertura de testes, diminuindo a quantidade de linhas de código escrita, reduzindo as violações e aumentando a conformidade com as regras de qualidade definidas pela ferramenta Sonar. O aumento de conformidade com as regras é percebido nos dois grupos e pela grande maioria dos desenvolvedores que participaram do experimento, conforme apresentado na Figura 5.12 e Figura 5.13.

Qualidade Externa: Em geral, a quantidade de casos de teste que obtiveram sucesso reduziu em 4,70% e a densidade de defeitos por LoC aumentou em 0,0149 (ou aproximadamente 30%, se considerarmos o valor máximo de 0,0499 obtido na média geral dos resultados) ao utilizar a WS-TDD se comparado com a TLD. Esses resultados não eram esperados, uma vez que as métricas de cobertura de código foi maior em todos os casos conforme apresentado na Seção 5.2.1. A qualidade externa foi impactada negativamente ao utilizar a WS-TDD. Esses efeitos ainda precisam ser detalhados e analisados separadamente em outros experimentos para identificar quais outras variáveis, não usadas neste trabalho, poderiam ter influenciado nesse resultado. Uma variável inicial que pode ter influência direta neste resultado é a cobertura dos testes criados pelo autor para fazer a validação.

Produtividade: Os desenvolvedores escreveram menos linhas de código por minuto quando utilizaram a WS-TDD. Além disso, eles finalizaram os exercícios em menos tempo do que com a TLD. Além da produtividade maior, o código gerado com a WS-TDD foi mais enxuto. A refatoração constante do código na WS-TDD pode ter levado a este resultado, porém outros estudos que observem esse comportamento precisam ser conduzidos para validar essa hipótese.

6.1.3 Entrevista

As entrevistas com os profissionais que participaram dos experimentos trouxeram informações valiosas sobre a percepção de cada um. Como descrito anteriormente, a WS-TDD obteve a maior engajamento do que a TLD de acordo com as respostas obtidas. Em resumo, os participantes perceberam que ao usar a WS-TDD eles obtiveram uma maior cobertura de código, maior compreensão dos requisitos, maior qualidade de código e, consequentemente, um sistema com menos defeitos. O ponto de atenção ao adotar WS-TDD, assim como no TDD, é a mudança cultural dos desenvolvedores na forma de desenvolvimento de aplicações.

6.1.4 Triangulação dos Resultados

Em linhas gerais, os resultados obtidos pelo experimento na qualidade interna e na produtividade mantiveram-se condizentes com os resultados do questionário. Porém, no que diz respeito a qualidade externa o resultado do experimento diverge do que foi obtido no questionário, mostrando que os participantes acreditavam que adotar a prática de criar testes antes do código acarretaria em um software com menos defeitos, no entanto isso mostrou-se falso. Os resultados da entrevista também mostram um entendimento de que a WS-TDD produz código com maior qualidade interna, com menos defeitos e em menos tempo. Esses resultados estão alinhados com os obtidos no questionário.

6.1.5 Considerações dos Resultados

Conforme os resultados apresentados nos questionários e na entrevista, a maioria dos participantes disseram que a qualidade interna, a qualidade externa e a produtividade aumentam quando a prática de TDD ou WS-TDD é utilizada durante o desenvolvimento de software. O experimento comprovou na prática que a opinião dos participantes é verdadeira no que tange a qualidade interna e a produtividade, porém a qualidade externa teve um efeito contrário do esperado e diminuiu ao utilizar a WS-TDD.

A diminuição da qualidade externa identificada no experimento pode estar relacionada à cobertura dos testes criados e executados externamente ao software ou também pela falta de experiência dos participantes no desenvolvimento de serviços web. Essas hipóteses podem ser uma das causas para a diminuição da qualidade externa e não somente o fato de ter usado a abordagem WS-TDD no desenvolvimento dos sistemas, ou seja, pode ter havido uma combinação de fatores que culminaram para a redução da qualidade externa.

6.2 Discussão dos Resultados com a Literatura

Esta seção discute os resultados obtidos com a WS-TDD e os comparam com os estudos sobre TDD no desenvolvimento de software identificados na literatura por meio da revisão sistemática de Bissi, Neto e Emer (2016). Como os estudos sobre TDD no desenvolvimento de WS, apresentados no Apêndice B.2, referem-se a conceitos e ferramentas sem que algum experimento tenha sido realizado comparando os resultados entre TDD e TLD, não foi possível fazer a discussão dos resultados destes estudos com os deste trabalho. Portanto, esta seção limita-se a discutir os estudos sobre TDD no desenvolvimento de software.

6.2.1 Qualidade Interna

Conforme apresentado em detalhes na Seção 5.2, a WS-TDD aumentou a qualidade interna do software nos experimentos realizados. A Tabela 6.1 apresenta todos os 15 estudos com experimentos, encontrados e classificados de acordo com o resultado, em que (+) significa que houve aumento, (=) mostra que os resultados se manteve e (-) significa que houve uma queda na qualidade interna.

Tabela 6.1: Comparação dos efeitos de TDD com TLD na qualidade interna.

Cenário	**(+) QI**	**(=) QI**	**(-) QI**
Acadêmico	Geras, Smith e Miller (2004), Janzen e Saiedian (2008a), Vu et al. (2009), Janzen, Turner e Saiedian (2007), Erdogmus, Morisio e Torchiano (2005), Pancur e Ciglaric (2011)	Janzen e Saiedian (2008b), Madeyski (2010)	Pancur et al. (2003), Janzen e Saiedian (2006)
Industrial	George e Williams (2004), George e Williams (2003), Maximilien e Williams (2003), Janzen e Saiedian (2008a), Canfora et al. (2006)		

Cerca de 73% dos experimentos em ambiente acadêmico e industrial, encontrados na revisão sistemática, identificaram que a qualidade interna aumentou ao usar TDD. A variação nos resultados aconteceu somente no ambiente acadêmico. No ambiente industrial todos os trabalhos identificaram que a qualidade interna aumentou, sendo este o mesmo resultado obtido no experimento realizado neste trabalho usado a WS-TDD. O resultado obtido apenas reforçou os já existentes na literatura.

6.2.2 Qualidade Externa

A qualidade externa do software diminuiu, em média, para todos os grupos no qual o experimento foi realizado ao utilizar a WS-TDD quando comparado a TLD. Esse resultado é apresentado em detalhes na Seção 5.2.

A Tabela 6.2 apresenta todos os 9 estudos com experimentos, encontrados e classificados de acordo com o resultado na qualidade externa. Cerca de 78% dos experimentos em ambiente acadêmico e industrial, encontrados na revisão sistemática, identificaram que a qualidade externa aumentou. A variação nos resultados novamente aconteceu no ambiente acadêmico. No ambiente industrial todos os trabalhos identificaram que a qualidade externa aumentou.

Tabela 6.2: Comparação dos efeitos de TDD com TLD na qualidade externa.

Cenário	**(+) QE**	**(=) QE**	**(-) QE**
Acadêmico	Geras, Smith e Miller (2004), Vu et al. (2009), Edwards (2004), Pancur e Ciglaric (2011)	Gupta e Jalote (2007)	Pancur et al. (2003)
Industrial	George e Williams (2004), George e Williams (2003), Maximilien e Williams (2003)		

Os resultados obtidos por este trabalho no ambiente industrial assemelham-se aos resultados descritos em Pancur e Ciglaric (2011) no ambiente acadêmico, no qual também foi identificado que a qualidade externa do software diminuiu ao utilizar TDD se comparado com TLD. Por se tratar de um resultado que difere dos obtidos na literatura, é provável que outras variáveis não observadas (como o domínio da aplicação, contexto de negócio e experiência dos participantes) tenham contribuído para este resultado.

6.2.3 Produtividade

A produtividade dos desenvolvedores aumentou, em média, para os dois grupos em que o experimento foi realizado ao utilizar a WS-TDD quando comparado a TLD, seja pelo menor número de linhas de código ou pelo tempo gasto para implementar as funcionalidades solicitadas. Esse resultado também é apresentado em detalhes na Seção 5.2.

A Tabela 6.3 apresenta todos os 11 estudos com experimentos, encontrados e classificados de acordo com o resultado na produtividade dos desenvolvedores. Somente 27% dos experimentos em ambiente acadêmico e industrial, encontrados na revisão sistemática, identificaram que a produtividade aumentou. A variação nos resultados aconteceu nos dois ambientes, sendo que no ambiente industrial nenhum trabalho identificou aumento na produtividade.

O experimento realizado neste trabalho foi no ambiente industrial e os resultados obtidos são superiores aos da literatura apresentada na Tabela 6.3. No contexto industrial, este é o primeiro trabalho que identificou um aumento da produtividade ao usar um abordagem baseada em TDD. Isso pode ter ocorrido devido ao uso das ferramentas e técnicas que suportam a WS-TDD e foram detalhadas no Capítulo 4.

Tabela 6.3: Comparação dos efeitos de TDD com TLD na produtividade.

Método	**Cenário**	**(+) Prd.**	**(=) Prd.**	**(-) Prd.**
Experimento	Acadêmico	Gupta e Jalote (2007), Janzen e Saiedian (2006), Pancur e Ciglaric (2011)	Geras, Smith e Miller (2004), Janzen e Saiedian (2008b), Erdogmus, Morisio e Torchiano (2005)	Vu et al. (2009)
Experimento	Industrial		Maximilien e Williams (2003)	George e Williams (2004), George e Williams (2003), Canfora et al. (2006)

6.3 Ameaças à Validade da Pesquisa

A pesquisa realizada foi extensiva em seu método e na quantidade de profissionais que participaram. Porém, como em todo estudo, existem ameaças à validade do trabalho realizado. As principais ameaças a este estudo são:

- O estudo foi realizado em três empresas, portanto não é possível generalizar os resultados obtidos;
- O foco na linguagem Java, nas ferramentas detalhadas no experimento e no paradigma de orientação em objetos podem influenciar os resultados obtidos, caso o mesmo experimento seja realizado com outra linguagem ou paradigma;
- O experimento foi realizado com dois grupos de desenvolvedores totalizando 23 participantes. O número de participantes foi pequeno e outros grupos poderiam ter sido formados para diminuir ainda mais alguma variação nos resultados obtidos;
- As métricas de produtividade foram escolhidas com base em outros trabalhos encontrados para facilitar a comparação entre eles. Porém, elas podem variar de acordo com a experiência do desenvolvedor. Além disso, a quantidade de defeitos encontrados não foi contabilizada como retrabalho por que os defeitos não foram corrigidos. O tempo gasto na correção poderia impactar na produtividade;
- Os exercícios possuem dois contextos de negócios diferentes, apesar de terem a mesma quantidade de regras e funcionalidades. Porém, a familiaridade do desenvolvedor em um dos assuntos poderia reduzir o tempo de implementação;

- O tempo de experiência dos participantes no experimento não foi considerado nos resultados obtidos, porém a grande maioria possui de 5 a 6 anos de experiência em desenvolvimento de software na linguagem Java.

6.4 Considerações Finais do Capítulo

Este trabalho trouxe resultados primários diferentes dos conhecidos na literatura. Com relação à qualidade interna observou-se um aumento em todos os grupos de desenvolvedores enquanto que na literatura alguns trabalhos identificaram que a qualidade interna diminuiu. Na produtividade, enquanto que na literatura nenhum experimento no ambiente industrial identificou um aumento, este trabalho observou que a produtividade dos desenvolvedores aumentaram nos dois grupos.

Por fim, na qualidade externa o resultado foi diferente, enquanto que neste trabalho observou-se que a qualidade externa diminuiu, na literatura a maioria dos trabalhos identificaram que a qualidade externa aumentou.

Outro comportamento apresentado neste estudo foi que uma qualidade interna maior e uma cobertura de código superior não reflete diretamente em um menor número de defeitos externos. Essa afirmação pode ser feita com base nos resultados obtidos no experimento deste trabalho.

Este capítulo discutiu os principais resultados obtidos e os comparou com os trabalhos encontrados na literatura. O próximo capítulo faz a conclusão deste trabalho, apresentando as principais contribuições e sugerindo alguns trabalhos futuros para esta linha de pesquisa.

Capítulo 7

Conclusão

O presente trabalho definiu e validou uma abordagem para o desenvolvimento de WS baseada em TDD, denominada de WS-TDD. Inicialmente a opinião dos desenvolvedores sobre o uso da prática de TDD no desenvolvimento de software em geral foi obtida por meio de um questionário presencial aplicado a 116 profissionais. Os resultados do questionário e a experiência do autor contribuíram para a definição e construção da WS-TDD. Essa abordagem é uma extensão do fluxo de TDD combinada com ferramentas e técnicas específicas para suportar as particularidades do desenvolvimento de serviços web.

A abordagem WS-TDD foi especificada e seu processo orienta os desenvolvedores com ferramentas e técnicas (JUnit, TestNG, Arquillian, EasyMock, Mockito, Powermock, DbC, teste unitário, objetos simulados) de acordo com a fase executada na abordagem. As ferramentas e técnicas descritas auxiliam os desenvolvedores a progredirem na implementação das funcionalidades e oferecem alternativas para lidar com a dependência de componentes externos e a falta de informação entre eles na fase de desenvolvimento e de teste (unitário e integrado).

O experimento conduzido no ambiente corporativo contou com a participação de 23 profissionais. Com base neste experimento, os efeitos que a WS-TDD gera na qualidade interna, qualidade externa e produtividade foram aferidos. De modo geral, a WS-TDD mostrou-se eficiente ao aumentar a qualidade interna e a produtividade dos desenvolvedores, mas em contrapartida, a qualidade externa do software diminuiu ao usar a WS-TDD quando comparada a TLD. Dessa forma, os objetivos específicos de aumentar a qualidade interna e produtividade foram atingidos, porém o objetivo de aumentar a qualidade externa não foi alcançado. O resultado obtido na qualidade externa precisa ser estudado com mais detalhes em outros trabalhos, pois alguns fatores como a cobertura dos casos de testes utilizados podem ter influenciado nesses resultados.

Ao finalizar a implementação do experimento, os participantes foram entrevistados e suas percepções quanto às vantagens e desvantagens na utilização da WS-TDD foram registradas. A grande maioria relatou que ao adotar a WS-TDD o entendimento dos requisitos melhorou e o código final ficou mais estruturado além de aumentar a cobertura de testes para o código desenvolvido. Essa percepção foi confirmada ao analisar os dados do experimento.

A triangulação dos resultados do questionário, experimento e entrevista trouxe resultados similares aos relatados pelos experimentos individualmente. A triangulação serviu para confirmar os resultados obtidos nos três métodos de pesquisa executados durante este trabalho.

Os resultados obtidos na qualidade interna são similares aos identificados na revisão sistemática, no qual ao usar a prática de TDD foi percebido um aumento na qualidade interna.

Os resultados obtidos na qualidade externa são diferentes dos encontrados na revisão sistemática, ou seja, este trabalho identificou que a qualidade externa diminuiu enquanto a literatura aponta que a qualidade externa aumentou ao usar TDD. Os resultados obtidos na produtividade são diferentes dos encontrados na revisão sistemática, ou seja, este trabalho identificou que a produtividade aumentou enquanto a literatura aponta que a produtividade diminuiu ou se manteve a mesma ao usar TDD no cenário industrial.

Como nenhum experimento que fizesse a comparação entre TDD e TLD no desenvolvimento de serviços web foi encontrado na literatura, os resultados obtidos neste trabalho foram comparados com os estudos que utilizaram TDD no desenvolvimento de software centralizado.

As informações apresentadas neste trabalho servirão de referência para os interessados em utilizar TDD durante o desenvolvimento de software, especialmente na construção de serviços web. As evidências geradas a partir dos três métodos de pesquisa utilizados, somados aos estudos sobre TDD, servirão de base para novas pesquisas na área de Engenharia de Software.

No decorrer desta pesquisa de mestrado, o artigo "*The Effects of Test Driven Development on Internal Quality, External Quality and Productivity: A systematic review*" foi publicado na revista internacional *Information and Software Technology* (v. 74, p. 45 - 54, 2016), mostrando o interesse da academia sobre este assunto.

Por fim, as contribuições deste trabalho de mestrado foram: (1) Mapeamento sobre TDD e desenvolvimento de software nas empresas locais; (2) Criação, aplicação e validação da abordagem WS-TDD; (3) Execução de experimentos e entrevistas com desenvolvedores nas empresas locais; e (4) Um artigo publicado na revista *Information and Software Technology*.

Este trabalho aplicou TDD no desenvolvimento de serviços web para integrar aplicações em um ambiente com sistemas distribuídos conectados por uma rede. Esse ambiente é diferente dos utilizados em todos os trabalhos encontrados na revisão sistemática que contemplavam apenas aplicações centralizadas. A WS-TDD viabilizou o uso de TDD no desenvolvimento de serviços web e mostrou que há benefícios para a qualidade interna e para a produtividade dos desenvolvedores.

7.1 Trabalhos Futuros

Como trabalho futuro sugere-se expandir o uso da abordagem WS-TDD para outros desenvolvedores e ambientes de desenvolvimento com o intuito de obter novos resultados que ampliem os descritos neste trabalho. A partir do direcionamento da WS-TDD novas ferramentas que integram os conceitos aqui apresentados podem ser construídas com o objetivo de facilitar o uso da abordagem.

Novos experimentos que utilizem a WS-TDD também são importantes para confrontar os resultados aqui apresentados, principalmente com relação a qualidade externa do software que diminuiu ao utilizar a WS-TDD.

Estudos futuros também devem ser dirigidos para validar a abordagem WS-TDD em outras linguagens e paradigmas além de Java, orientação a objetos e da arquitetura orientada a serviços.

Expandir e adequar o uso da WS-TDD para o desenvolvimento serviços web de estilo REST, pois este trabalho considerou apenas os serviços web baseados em SOAP, que são os mais comuns atualmente.

As métricas de qualidade interna do software utilizadas são baseadas nas métricas de orientação a objetos, embora sejam bastante difundidas na comunidade, outras métricas direcionadas a arquitetura orientada a serviços são necessárias para aumentar a precisão dos resultados e validar as métricas existentes, principalmente as relacionadas ao reúso de componentes de software.

Referências Bibliográficas

ABNT. *NBR ISO 8402 : Gestão da qualidade e garantia da qualidade - Terminologia*. [S.l.], 1994. 14 p.

ABNT. *NBR ISO/IEC 9126-1 : Engenharia de software - Qualidade de produto Parte 1: Modelo de qualidade*. [S.l.], 2003. 21 p.

ANICHE, Mauricio Finavaro; GEROSA, Marco Aurélio. Most common mistakes in Test-Driven Development practice: Results from an online survey with developers. In: *Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops*. Washington, DC, USA: IEEE Computer Society, 2010. (ICSTW '10), p. 469–478. ISBN 978-0-7695-4050-4. Disponível em: <http://dx.doi.org/10.1109/ICSTW.2010.16>.

ANICHE, Mauricio Finavaro; GEROSA, Marco Aurelio. How the practice of TDD influences class design in object-oriented systems: Patterns of unit tests feedback. In: *Proceedings of the 2012 26th Brazilian Symposium on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2012. (SBES '12), p. 1–10. ISBN 978-0-7695-4868-5. Disponível em: <http://dx.doi.org/10.1109/SBES.2012.14>.

AZEVEDO, Carlos Eduardo Franco; OLIVEIRA, Leonel Gois Lima; GONZALEZ, Rafael Kuramoto; ABDALLA, Márcio Moutinho. A estratégia de triangulação: objetivos, possibilidades, limitações e proximidades com o pragmatismo. *V Encontro de Ensino e Pesquisa em Administração e contabilidade (ANPAD). Brasília*, 2013.

BAI, Xiaoying; DAI, Guilan; XU, Dezheng; TSAI, Wei-Tek. A multi-agent based framework for collaborative testing on web services. In: *Proceedings of the The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06)*. Washington, DC, USA: IEEE Computer Society, 2006. (SEUS-WCCIA '06), p. 205–210. ISBN 0-7695-2560-1.

BAI, Xiaoying; DONG, Wenli; TSAI, Wei-Tek; CHEN, Yinong. WSDL-based automatic test case generation for web services testing. In: *Service-Oriented System Engineering, 2005. SOSE 2005. IEEE International Workshop*. [S.l.: s.n.], 2005. p. 207–212.

BARBIR, Abbie; HOBBS, Chris; BERTINO, Elisa; HIRSCH, Frederick; MARTINO, Lorenzo D. Challenges of testing web services and security in SOA implementations. In: BARESI, Luciano; DI NITTO, Elisabetta (Ed.). *Test and Analysis of Web Services*. Springer Berlin Heidelberg, 2007. p. 395–440. ISBN 978-3-540-72911-2. Disponível em: <http://dx.doi.org/10.1007/978-3-540-72912-9_14>.

BARTOLINI, Cesare; BERTOLINO, Antonia; ELBAUM, Sebastian; MARCHETTI, Eda. Whitening SOA testing. In: *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*. New York, NY, USA: ACM, 2009. (ESEC/FSE '09), p. 161–170. ISBN 978-1-60558-001-2. Disponível em: <http://doi.acm.org/10.1145/1595696.1595721>.

BARTOLINI, Cesare; BERTOLINO, Antonia; ELBAUM, Sebastian; MARCHETTI, Eda. Bringing white-box testing to service oriented architectures through a service oriented approach. *Journal of Systems and Software*, v. 84, n. 4, p. 655 – 668, 2011. ISSN 0164-1212. The Ninth International Conference on Quality Software.

BARTOLINI, Cesare; BERTOLINO, Antonia; MARCHETTI, Eda. Introducing service-oriented coverage testing. In: *23rd IEEE/ACM International Conference on Automated Software Engineering - Workshops, 2008.* [S.l.: s.n.], 2008. (ASE Workshops 2008), p. 57–64.

BARTOLINI, Cesare; BERTOLINO, Antonia; MARCHETTI, Eda; PARISSIS, Ioannis. Data flow-based validation of web services compositions: Perspectives and examples. In: LEMOS, Rogério; GIANDOMENICO, Felicita; GACEK, Cristina; MUCCINI, Henry; VIEIRA, Marlon (Ed.). *Architecting Dependable Systems V*. Berlin, Heidelberg: Springer-Verlag, 2008. p. 298–325. ISBN 978-3-540-85570-5. Disponível em: <http://dx.doi.org/10.1007/978-3-540-85571-2_13>.

BARTOLINI, Cesare; BERTOLINO, Antonia; MARCHETTI, Eda; POLINI, Andrea. WS-TAXI: A WSDL-based testing tool for web services. In: *International Conference on Software Testing Verification and Validation, 2009.* [S.l.: s.n.], 2009. (ICST '09), p. 326–335.

BECK, Kent. *Extreme Programming Explained: Embrace Change*. 1. ed. Boston: Addison-Wesley, 1999. ISBN 0201616416.

BECK, Kent. Aim, fire [test-first coding]. *IEEE Software*, v. 18, n. 5, p. 87–89, September 2001. ISSN 0740-7459.

BECK, Kent. *Test Driven Development: by example*. 1. ed. Boston: Addison-Wesley, 2002. ISBN 0321146530.

BECK, Kent; BEEDLE, Mike; BENNEKUM, Arie van; COCKBURN, Alistair; CUNNINGHAM, Ward; FOWLER, Martin; GRENNING, James; HIGHSMITH, Jim; HUNT, Andrew; JEFFRIES, Ron; KERN, Jon; MARICK, Brian; MARTIN, Robert C.; MELLOR, Steve; SCHWABER, Ken; SUTHERLAND, Jeff; THOMAS, Dave. *Manifesto for Agile Software Development*. 2001. <http://www.agilemanifesto.org/>. Acessado em 05/01/2016.

BELLI, Fevzi; LINSCHULTE, Michael. Event-driven modeling and testing of web services. In: *32nd Annual IEEE International Computer Software and Applications, 2008.* [S.l.: s.n.], 2008, (COMPSAC '08). p. 1168–1173.

BERTOLINO, Antonia; GAO, Jinghua; MARCHETTI, Eda; POLINI, Andrea. Automatic test data generation for XML schema-based partition testing. In: *Second International Workshop on Automation of Software Test , 2007.* [S.l.: s.n.], 2007. (AST '07), p. 4–4.

BESSON, Felipe; MOURA, Paulo; KON, Fabio; MILOJICIC, Dejan. Bringing test-driven development to web service choreographies. *Journal of Systems and Software*, v. 99, n. 0, p. 135 – 154, 2015. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/S016412121400209X>.

BHAT, Thirumalesh; NAGAPPAN, Nachiappan. Evaluating the efficacy of Test-Driven Development: Industrial case studies. In: *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*. New York, NY, USA: ACM, 2006. (ISESE '06), p. 356–363. ISBN 1-59593-218-6. Disponível em: <http://doi.acm.org/10.1145/1159733.1159787>.

BIANCO, Philip; LEWIS, Grace A.; MERSON, Paulo; SIMANTA, Soumya. *Architecting Service-Oriented Systems*. [S.l.], 2011. Technical Report CMU/SEI-2011-TN-008, Carnegie Mellon.

BISSI, Wilson; NETO, Adolfo Gustavo Serra Seca; EMER, Maria Claudia Figueiredo Pereira. The effects of test driven development on internal quality, external quality and productivity: A systematic review. *Information and Software Technology*, v. 74, p. 45 – 54, June 2016. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584916300222>.

BOOTH, David; HAAS, Hugo; MCCABE, Francis; NEWCOMER, Eric; CHAMPION, Michael; FERRIS, Chris; ORCHARD, David. *Web Services Architecture*. [S.l.], 2004. 91 p.

BOZKURT, Mustafa; HARMAN, Mark; HASSOUN, Youssef. *Testing Web Services: A Survey*. [S.l.], 2010. Technical Report TR-10-01, King's College London.

BOZKURT, Mustafa; HARMAN, Mark; HASSOUN, Youssef. Testing and Verification in Service-Oriented Architecture: A survey. *Software Testing, Verification and Reliability*, John Wiley & Sons, v. 23, n. 4, p. 261–313, 2013. ISSN 1099-1689. Disponível em: <http://dx.doi.org/10.1002/stvr.1470>.

CANFORA, Gerardo; CIMITILE, Aniello; GARCIA, Felix; PIATTINI, Mario; VISAGGIO, Corrado Aaron. Evaluating advantages of test driven development: A controlled experiment with professionals. In: *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*. New York, NY, USA: ACM, 2006. (ISESE '06), p. 364–371. ISBN 1-59593-218-6. Disponível em: <http://doi.acm.org/10.1145/1159733.1159788>.

CANFORA, Gerardo; DI PENTA, Massimiliano. Testing services and service-centric systems: Challenges and opportunities. *IT Professional*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 8, n. 2, p. 10–17, mar. 2006. ISSN 1520-9202. Disponível em: <http://dx.doi.org/10.1109/MITP.2006.51>.

CAUSEVIC, Adnan; SUNDMARK, Daniel; PUNNEKKAT, Sasikumar. Factors limiting industrial adoption of test driven development: A systematic review. In: *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*. [S.l.: s.n.], 2011. p. 337–346.

CHEN, Yuhul; ROMANOVSKY, Alexander. Improving the dependability of web services integration. *IT Professional*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 10, n. 3, p. 29–35, May 2008. ISSN 1520-9202. Disponível em: <http://dx.doi.org/10.1109/MITP.2008.49>.

CHIDAMBER, Shyam R.; KEMERER, Chris F. A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, v. 20, n. 6, p. 476–493, June 1994. ISSN 0098-5589.

COZBY, Paul. *Métodos de Pesquisa Em Ciências do Comportamento*. 1. ed. São Paulo: Atlas, 2003. ISBN 9788522433636.

DAMM, Lars-Ola; LUNDBERG, Lars. Results from introducing component-level test automation and Test-Driven Development. *Journal of Systems and Software*, v. 79, n. 7, p. 1001 – 1014, 2006. ISSN 0164-1212. Selected papers from the 11th Asia Pacific Software Engineering Conference (APSEC2004). Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121205001573>.

DAMM, Lars-Ola; LUNDBERG, Lars; OLSSON, David. Introducing test automation and test-driven development: An experience report. *Electronic Notes in Theoretical Computer Science*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 116, p. 3–15, January 2005. ISSN 1571-0661. Disponível em: <http://dx.doi.org/10.1016/j.entcs.2004.02.090>.

DELAMARO, Marcio Eduardo; MALDONADO, Jose Carlos; JINO, Mario. *Introdução ao Teste de Software*. 1. ed. Rio de Janeiro: Campus - Elsevier, 2007. ISBN 8535226346.

DEMARCO, Tom. *Controlling Software Projects: Management, Measurement, and Estimates*. 1. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1986. ISBN 0131717111.

DI PENTA, Massimiliano; BRUNO, Marcello; ESPOSITO, Gianpiero; MAZZA, Valentina; CANFORA, Gerardo. Test and analysis of web services. In: . [S.l.]: Springer-Verlag, 2007. cap. Web Services Regression Testing, p. 205–234. ISBN 978-3-540-72911-2.

DIJKSTRA, Edsger Wybe. *Notes on structured programming*. [S.l.], 1970. Technical Report 70-WSK03, Eindhoven.

DYBA, Tore; DINGSOYR, Torgeir; HANSSEN, Geir K. Applying systematic reviews to diverse study types: An experience report. In: *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*. Washington, DC, USA: IEEE Computer Society, 2007. (ESEM '07), p. 225–234. ISBN 0-7695-2886-4. Disponível em: <http://dx.doi.org/10.1109/ESEM.2007.21>.

EDWARDS, Stephen H. Using software testing to move students from trial-and-error to reflection-in-action. In: *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*. New York, NY, USA: ACM, 2004. (SIGCSE '04), p. 26–30. ISBN 1-58113-798-2. Disponível em: <http://doi.acm.org/10.1145/971300.971312>.

ERDOGMUS, Hakan; MORISIO, Maurizio; TORCHIANO, Marco. On the effectiveness of the test-first approach to programming. *Software Engineering, IEEE Transactions on*, v. 31, n. 3, p. 226–237, March 2005. ISSN 0098-5589.

ERL, Thomas. *SOA: Principles of Service Design*. 1. ed. Boston: Prentice Hall, 2007. ISBN 0132344823.

FEUDJIO, Vouffo Alain-Georges; SCHIEFERDECKER, Ina. Availability testing for web services. In: *Telektronikk*. [S.l.: s.n.], 2009. p. 81–89. ISSN 0085-7130.

FUCCI, Davide; TURHAN, Burak; JURISTO, Natalia; DIESTE, Oscar; TOSUN-MISIRLI, Ayse; OIVO, Markku. Towards an operationalization of test-driven development skills: An industrial empirical study. *Information and Software Technology*, v. 68, p. 82 – 97, 2015. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584915001469>.

GEORGE, Boby; WILLIAMS, Laurie. An initial investigation of test driven development in industry. In: *Proceedings of the 2003 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2003. (SAC '03), p. 1135–1139. ISBN 1-58113-624-2. Disponível em: <http://doi.acm.org/10.1145/952532.952753>.

GEORGE, Boby; WILLIAMS, Laurie. A structured experiment of test-driven development. *Information and Software Technology*, v. 46, n. 5, p. 337 – 342, 2004. ISSN 0950-5849. Special Issue on Software Engineering, Applications, Practices and Tools from the ACM Symposium on Applied Computing 2003. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584903002040>.

GERAS, Adam; SMITH, Michael; MILLER, James. A prototype empirical evaluation of test driven development. In: *Proceedings 10th International Symposium on Software Metrics, 2004.* [S.l.: s.n.], 2004. p. 405–416. ISSN 1530-1435.

GLIEM, Joseph A.; GLIEM, Rosemary R. Calculating, interpreting, and reporting Cronbach's alpha reliability coefficient for Likert-type scales. In: MIDWEST RESEARCH-TO-PRACTICE CONFERENCE IN ADULT, CONTINUING, AND COMMUNITY EDUCATION. Columbus, Ohio, USA, 2003. p. 82–88.

GOTTSCHALK, K.; GRAHAM, S.; KREGER, H.; SNELL, J. Introduction to web services architecture. *IBM Systems Journal*, IBM Corp., Riverton, NJ, USA, v. 41, n. 2, p. 170–177, April 2002. ISSN 0018-8670. Disponível em: <http://dx.doi.org/10.1147/sj.412.0170>.

GU, Qing; LAGO, Patricia. SOA process decisions: New challenges in architectural knowledge modeling. In: *Proceedings of the 3rd International Workshop on Sharing and Reusing Architectural Knowledge*. New York, NY, USA: ACM, 2008. (SHARK '08), p. 3–10. ISBN 978-1-60558-038-8. Disponível em: <http://doi.acm.org/10.1145/1370062.1370065>.

GUPTA, Atul; JALOTE, Pankaj. An experimental evaluation of the effectiveness and efficiency of the test driven development. In: *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*. Washington, DC, USA: IEEE Computer Society, 2007. (ESEM '07), p. 285–294. ISBN 0-7695-2886-4. Disponível em: <http://dx.doi.org/10.1109/ESEM.2007.20>.

HAMILL, Paul; ALEXANDER, David; SHASHARINA, Svetlana. Web service validation enabling Test-Driven Development of service-oriented applications. In: *Proceedings of the 2009 Congress on Services - I*. Washington, DC, USA: IEEE Computer Society,

2009. (SERVICES '09), p. 467–470. ISBN 978-0-7695-3708-5. Disponível em: <http://dx.doi.org/10.1109/SERVICES-I.2009.113>.

HANNA, Samer; MUNRO, Malcolm. An approach for wsdl-based automated robustness testing of web services. In: *Information Systems Development*. [S.l.]: Springer-Verlag, 2009. p. 1093–1104. ISBN 978-0-387-78577-6.

HEINEMAN, George T.; COUNCILL, William T. *Component-Based Software Engineering: Putting the Pieces Together*. 1. ed. Boston: Addison-Wesley, 2001. ISBN 076868207X.

HITSCHFELD, Nancy; LILLO, Carlos; CáCERES, Ana; BASTARRICA, María Cecilia; RIVARA, Maria-Cecilia. Building a 3d meshing framework using good software engineering practices. In: OCHOA, Sergio F.; ROMAN, Gruia-Catalin (Ed.). *Advanced Software Engineering: Expanding the Frontiers of Software Technology*. Springer US, 2006, (IFIP International Federation for Information Processing, v. 219). p. 162–170. ISBN 978-0-387-34828-5. Disponível em: <http://dx.doi.org/10.1007/978-0-387-34831-5_13>.

HOU, Shan-Shan; ZHANG, Lu; LAN, Qian; MEI, Hong; SUN, Jia-Su. Generating effective test sequences for BPEL testing. In: *9th International Conference on Quality Software*. [S.l.: s.n.], 2009. (QSIC '09), p. 331–340. ISSN 1550-6002.

HUANG, Hai; TSAI, Wei-Tek.; PAUL, Raymond. Automated model checking and testing for composite web services. In: *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*. [S.l.: s.n.], 2005. p. 300–307.

IEEE. *IEEE Std 1061 : IEEE Standard for a Software Quality Metrics Methodology*. [S.l.], 1998. 0-26 p.

JALALI, Samireh; WOHLIN, Claes. Systematic literature studies: Database searches vs. backward snowballing. In: *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. New York, NY, USA: ACM, 2012. (ESEM '12), p. 29–38. ISBN 978-1-4503-1056-7. Disponível em: <http://doi.acm.org/10.1145/2372251.2372257>.

JANZEN, David; SAIEDIAN, Hossein. Does test-driven development really improve software design quality? *IEEE Software*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 25, n. 2, p. 77–84, March 2008. ISSN 0740-7459. Disponível em: <http://dx.doi.org/10.1109/MS.2008.34>.

JANZEN, David; SAIEDIAN, Hossein. Test-Driven Learning in early programming courses. In: *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*. New York, NY, USA: ACM, 2008. (SIGCSE '08), p. 532–536. ISBN 978-1-59593-799-5. Disponível em: <http://doi.acm.org/10.1145/1352135.1352315>.

JANZEN, David S.; SAIEDIAN, Hossein. On the influence of Test-Driven Development on software design. In: *Proceedings of the 19th Conference on Software Engineering Education & Training*. Washington, DC, USA: IEEE Computer Society, 2006. (CSEET '06), p. 141–148. ISBN 0-7695-2557-1. Disponível em: <http://dx.doi.org/10.1109/CSEET.2006.25>.

JANZEN, David S.; TURNER, Clark S.; SAIEDIAN, Hossein. Empirical software engineering in industry short courses. In: *Software Engineering Education Training, 2007. CSEET '07. 20th Conference on*. [S.l.: s.n.], 2007. p. 89–96. ISSN 1093-0175.

JEFFRIES, Ron; MELNIK, Grigori. TDD: The art of fearless programming. *IEEE Software*, v. 24, n. 3, p. 24–30, May 2007. ISSN 0740-7459.

JIANG, Ying; HOU, Shan-Shan; SHAN, Jin-Hui; ZHANG, Lu; XIE, Bing. Contract-based mutation for testing components. In: *Proceedings of the 21st IEEE International Conference on Software Maintenance*. [S.l.: s.n.], 2005. (ICSM '05), p. 483–492. ISSN 1063-6773.

JOSEFSSON, Marc. Making architectural design phase obsolete? TDD as a design method. In: *T-76.650 Seminar course on SQA in Agile Software Development*. [S.l.: s.n.], 2004. p. 1–7. Site: <http://www.soberit.hut.fi/t-76.5650/spring_2004/papers/m.josefsson_76650_final.pdf> [Último Acesso em: 07 Julho 2014].

KALAMEGAM, Poonkavithai; GOD, Zayaraz. A survey on testing SOA built using web services. *International Journal of Software Engineering and Its Applications*, p. 91 – 104, 2012.

KALIN, Martin. *Java Web Services: Up and Running*. 1. ed. Sebastopol: O'Reilly Media, 2009. ISBN 059652112X.

KANER, Cem; BOND, Walter P. Software engineering metrics: What do they measure and how do we know? In: *Proceedings of the 2004 10th International Software Metrics Symposium*. [S.l.]: IEEE Computer Society, 2004. (METRICS '04), p. 1–12. ISBN 0-7695-2371-4.

KIM, Taeksu; PARK, Chanjin; WU, Chisu. Mock object models for test driven development. In: *Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications*. Washington, DC, USA: IEEE Computer Society, 2006. (SERA '06), p. 221–228. ISBN 0-7695-2656-X. Disponível em: <http://dx.doi.org/10.1109/SERA.2006.49>.

KITCHENHAM, Barbara. *DESMET: A method for evaluating Software Engineering methods and tools*. [S.l.], 1996. Technical Report TR96-09, Keele University Report.

KITCHENHAM, Barbara. Procedures for performing systematic reviews. *Departament of Computer Science, Keele University*, v. 33, p. 1–26, 2004. ISSN 1353-7776.

KITCHENHAM, Barbara; BRERETON, O. Pearl; BUDGEN, David; TURNER, Mark; BAILEY, John; LINKMAN, Stephen. Systematic literature reviews in software engineering : A systematic literature review. *Information and Software Technology*, v. 51, n. 1, p. 7 – 15, 2009. ISSN 0950-5849. Special Section - Most Cited Articles in 2002 and Regular Research Papers. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584908001390>.

KITCHENHAM, Barbara; CHARTERS, Stuart. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. [S.l.], 2007. Technical Report EBSE-2007-01, Keele University and Durham University Joint Report.

KOLTAI, Benji; WARNICK, Jeffrey; AGBAJI, Ruth; NILAN, Sean. Test-driven development. *ACM Transactions on Computational Logic (TOCL)*, ACM, New York, NY, USA, v. 5, p. 1–21, 2011. ISSN 1529-3785.

KOSCIANSKI, André; SOARES, Michel dos Santos. *Qualidade de Software*. 2. ed. [S.l.]: Novatec, 2007. ISBN 9788575221129.

KRUEGER, Charles W. Software reuse. *ACM Computing Surveys*, ACM, New York, NY, USA, v. 24, n. 2, p. 131–183, June 1992. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/130844.130856>.

LADAN, Mohamad I. Web services testing approaches: A survey and a classification. In: ZAVORAL, Filip; YAGHOB, Jakub; PICHAPPAN, Pit; EL-QAWASMEH, Eyas (Ed.). *Networked Digital Technologies: Second International Conference*. Springer, 2010. (Communications in Computer and Information Science, v. 88), p. 70–79. ISBN 978-3-642-14305-2. Disponível em: <http://dblp.uni-trier.de/db/conf/ndt/ndt2010-2.html\#Ladan10>.

LARANJEIRO, Nuno; VIEIRA, Marco. Extending Test-Driven Development for robust web services. In: *Proceedings of the 2009 Second International Conference on Dependability*. Washington, DC, USA: IEEE Computer Society, 2009. (DEPEND '09), p. 122–127. ISBN 978-0-7695-3666-8. Disponível em: <http://dx.doi.org/10.1109/DEPEND.2009.25>.

LEFFINGWELL, Dean. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. 1. ed. Boston: Addison-Wesley, 2011. ISBN 0321635841.

LENZ, Chris; CHIMIAK-OPOKA, Joanna; BREU, Ruth. Model-driven testing of SOA-based software. In: *In Proceedings of the Workshop on Software Engineering Methods for Service-Oriented Architecture*. Hannover, Germany: Leibniz Universität Hannover, 2007. (SEMSOA 2007), p. 99–110.

LI, Liangming; WANG, Zhijian; ZHANG, Xuejie. An approach to testing based component composition. In: *ISECS International Colloquium on Computing, Communication, Control, and Management*. [S.l.: s.n.], 2008. (CCCM '08, v. 1), p. 735–739.

LI, Zhong Jie; TAN, H.F.; LIU, H.H.; ZHU, Jun; MITSUMORI, Naomi M. Business-process-driven gray-box SOA testing. *IBM Systems Journal*, v. 47, n. 3, p. 457–472, April 2008. ISSN 0018-8670.

LIANG, Donglin; XU, Kai. Testing scenario implementation with behavior contracts. In: *30th Annual International Computer Software and Applications Conference*. [S.l.: s.n.], 2006. (COMPSAC '06, v. 1), p. 395–402. ISSN 0730-3157.

LUZ, Ramiro Batista da; NETO, Adolfo Gustavo Serra Seca; NORONHA, Robinson Vida. Teaching TDD, the coding Dojo style. In: *Proceedings of the 2013 IEEE 13th International Conference on Advanced Learning Technologies*. Washington, DC, USA: IEEE Computer Society, 2013. (ICALT '13), p. 371–375. ISBN 978-0-7695-5009-1. Disponível em: <http://dx.doi.org/10.1109/ICALT.2013.114>.

MADEYSKI, Lech. The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment. *Information and Software Technology*, v. 52, n. 2, p. 169 – 184, 2010. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584909001487>.

MADEYSKI, Lech; SZALA, Lukasz. The impact of test-driven development on software development productivity - an empirical study. In: ABRAHAMSSON, Pekka; BADDOO, Nathan; MARGARIA, Tiziana; MESSNARZ, Richard (Ed.). *Software Process Improvement*. Springer Berlin Heidelberg, 2007, (Lecture Notes in Computer Science, v. 4764). p. 200–211. ISBN 978-3-540-74765-9. Disponível em: <http://dx.doi.org/10.1007/978-3-540-75381-0_18>.

MARCONI, Marina de Andrade; LAKATOS, Eva Maria. *Fundamentos de metodologia científica*. 6. ed. São Paulo: Atlas, 2005. ISBN 9788522440153.

MARTIN, Robert. *Agile Principles, Patterns, and Practices in C#*. 1. ed. [S.l.]: Prentice Hall, 2006. ISBN 0131857258.

MASSOL, Vincent; TAHCHIEV, Petar; LEME, Felipe; GREGORY, Gary. *Component-Based Software Engineering: Putting the Pieces Together*. 2. ed. Stamford: Manning Publications, 2010. ISBN 1935182021.

MAXIMILIEN, E. Michael; WILLIAMS, Laurie. Assessing Test-Driven Development at IBM. In: *Software Engineering, 2003. Proceedings. 25th International Conference on*. [S.l.: s.n.], 2003. p. 564–569. ISSN 0270-5257.

MAYER, Philip; LüBKE, Daniel. Towards a BPEL unit testing framework. In: *Proceedings of the 2006 Workshop on Testing, Analysis, and Verification of Web Services and Applications*. New York, NY, USA: ACM, 2006. (TAV-WEB '06), p. 33–42. ISBN 1-59593-458-8. Disponível em: <http://doi.acm.org/10.1145/1145718.1145723>.

MCCABE, Thomas J. A complexity measure. *IEEE Transactions on Software Engineering*, IEEE Press, Piscataway, NJ, USA, v. 2, n. 4, p. 308–320, jul 1976. ISSN 0098-5589. Disponível em: <http://dx.doi.org/10.1109/TSE.1976.233837>.

MEI, Hong; ZHANG, Lu. A framework for testing web services and its supporting tool. In: *IEEE International Workshop Service-Oriented System Engineering*. [S.l.: s.n.], 2005. (SOSE '05), p. 199–206.

MELO, Claudia de Oliveira; SANTOS, Viviane; KATAYAMA, Eduardo; CORBUCCI, Hugo; PRIKLADNICKI, Rafael; GOLDMAN, Alfredo; KON, Fabio. The evolution of agile software development in brazil. *Journal of the Brazilian Computer Society*, v. 19, n. 4, p. 523–552, 2013. ISSN 1678-4804. Disponível em: <http://dx.doi.org/10.1007/s13173-013-0114-x>.

MERILINNA, Janne; MATINLASSI, Mari. State of the art and practice of OpenSource component integration. In: *32nd EUROMICRO Conference on Software Engineering and Advanced Applications, 2006. SEAA '06.* [S.l.: s.n.], 2006. p. 170–177. ISSN 1089-6503.

MEYER, Bertrand. Applying design by contract. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 25, n. 10, p. 40–51, October 1992. ISSN 0018-9162. Disponível em: <http://dx.doi.org/10.1109/2.161279>.

MUNIR, Hussan; MOAYYED, Misagh; PETERSEN, Kai. Considering rigor and relevance when evaluating test driven development: A systematic review. *Information and Software Technology*, Butterworth-Heinemann, Newton, MA, USA, v. 56, n. 4, p. 375–394, April 2014. ISSN 0950-5849. Disponível em: <http://dx.doi.org/10.1016/j.infsof.2014.01.002>.

MYERS, Glenford J.; SANDLER, Corey; BADGETT, Tom. *The Art of Software Testing*. 3. ed. [S.l.]: Wiley, 2011. ISBN 9781118031964.

NAGAPPAN, Nachiappan; MAXIMILIEN, E. Michael; BHAT, Thirumalesh; WILLIAMS, Laurie. Realizing quality improvement through test driven development: Results and experiences of four industrial teams. *Empirical Software Engineering*, Kluwer Academic Publishers, Hingham, MA, USA, v. 13, n. 3, p. 289–302, June 2008. ISSN 1382-3256. Disponível em: <http://dx.doi.org/10.1007/s10664-008-9062-z>.

NEBUT, Clémentine; FLEUREY, Franck; TRAON, Yves Le; JÉZÉQUEL, Jean-Marc. Requirements by contracts allow automated system testing. In: *14th International Symposium on Software Reliability Engineering, 2003*. [S.l.: s.n.], 2003. (ISSRE '03), p. 85–96. ISSN 1071-9458.

NOIKAJANA, Siripol; SUWANNASART, Taratip. Web service test case generation based on decision table (short paper). In: *Proceedings of the 2008 The Eighth International Conference on Quality Software*. Washington, DC, USA: IEEE Computer Society, 2008. (QSIC '08), p. 321–326. ISBN 978-0-7695-3312-4. Disponível em: <http://dx.doi.org/10.1109/QSIC.2008.7>.

OASIS. *UDDI Specification*. 2016. <https://www.oasis-open.org/committees/uddi-spec/faq.php>. Acessado em 05/01/2016.

PALACIOS, Marcos; GARCÍA-FANJUL, José; TUYA, Javier. Testing in service oriented architectures with dynamic binding: A mapping study. *Information and Software Technology*, Butterworth-Heinemann, Newton, MA, USA, v. 53, n. 3, p. 171–189, March 2011. ISSN 0950-5849. Disponível em: <http://dx.doi.org/10.1016/j.infsof.2010.11.014>.

PANCUR, Matjaz; CIGLARIC, Mojca. Impact of Test-Driven Development on productivity, code and tests: A controlled experiment. *Information and Software Technology*, v. 53, n. 6, p. 557 – 573, 2011. ISSN 0950-5849. Special Section: Best papers from the (APSEC). Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584911000346>.

PANCUR, Matjaz; CIGLARIC, Mojca; TRAMPUS, Matej; VIDMAR, Tone. Towards empirical evaluation of Test-Driven Development in a university environment. In: *EUROCON 2003. Computer as a Tool. The IEEE Region 8*. [S.l.: s.n.], 2003. v. 2, p. 83–86 vol.2.

PAPAZOGLOU, M.P.; TRAVERSO, P.; DUSTDAR, S.; LEYMANN, F. Service-Oriented Computing: State of the art and research challenges. *Computer*, v. 40, n. 11, p. 38–45, Nov 2007. ISSN 0018-9162.

PETTICREW, Mark; ROBERTS, Helen. *Systematic Reviews in the Social Sciences: A Practical Guide*. 1. ed. Oxford: Wiley-Blackwell, 2005. ISBN 1405121106.

PIKKARAINEN, Minna; HAIKARA, J.; SALO, Outi; ABRAHAMSSON, Pekka; STILL, Jari. The impact of agile practices on communication in software development. *Journal Empirical Software Engineering*, Kluwer Academic Publishers, Hingham, MA, USA, v. 13, n. 3, p. 303–337, June 2008. ISSN 1382-3256. Disponível em: <http://dx.doi.org/10.1007/s10664-008-9065-9>.

PRAS, Aiko; MARTIN-FLATIN, Jean-Philippe. What can web services bring to integrated management? *Handbook of network and system administration*, Elsevier Science Limited, p. 241 – 294, 2008.

PRESSMAN, Roger S. *Software Engineering: A Practitioner's Approach*. 4. ed. [S.l.]: McGraw-Hill, 1997. ISBN 0070521824.

RAFIQUE, Yahya; MISIC, Vojislav B. The effects of test-driven development on external quality and productivity: A meta-analysis. *IEEE Transactions on Software Engineering*, IEEE Press, Piscataway, NJ, USA, v. 39, n. 6, p. 835–856, June 2013. ISSN 0098-5589. Disponível em: <http://dx.doi.org/10.1109/TSE.2012.28>.

RIBAROV, Lachezar; MANOVA, Ilina; ILIEVA, Sylvia; BLVD, James Bourchier. Testing in a Service-Oriented World. In: *Proceedings of the International Conference on Information Technologies (InfoTech-2007)*. Bulgaria: [s.n.], 2007. p. 1–10.

ROCHA, Ana Regina Cavalcanti; MALDONADO, José Carlos; WEBER, Kival Chaver. *Qualidade de Software: Teoria e Prática*. 1. ed. [S.l.]: Prentice Hall, 2001. ISBN 8587918540.

SANCHEZ, Julio Cesar; WILLIAMS, Laurie; MAXIMILIEN, E. Michael. On the sustained use of a Test-Driven Development practice at IBM. In: *Agile Conference (AGILE), 2007*. [S.l.: s.n.], 2007. p. 5–14.

SCHNEIDER, Vitali; GERMAN, Reinhard. Integration of Test-Driven Agile simulation approach in Service-Oriented tool environment. In: *Proceedings of the 46th Annual Simulation Symposium*. San Diego, CA, USA: Society for Computer Simulation International, 2013. (ANSS 13), p. 11:1–11:7. ISBN 978-1-62748-030-7. Disponível em: <http://dl.acm.org/citation.cfm?id=2499604.2499615>.

SCHUMACHER, Vera Rejane Niedersberg. *Qualidade de Software*. 3. ed. Palhoça: UnisulVirtual, 2007. ISBN 978-85-7817-000-4.

SFETSOS, Panagiotis; STAMELOS, Ioannis. Empirical studies on quality in agile practices: A systematic literature review. In: *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*. [S.l.: s.n.], 2010. p. 44–53.

SHARMA, Abhishek; HELLMANN, Theodore D.; MAURER, Frank. Testing of web services – a systematic mapping. In: *Proceedings of the 2012 IEEE Eighth World Congress on Services*. Washington, DC, USA: IEEE Computer Society, 2012. (SERVICES '12), p. 346–352. ISBN 978-0-7695-4756-5. Disponível em: <http://dx.doi.org/10.1109/SERVICES.2012.21>.

SINIAALTO, Maria; ABRAHAMSSON, Pekka. A comparative case study on the impact of Test-Driven Development on program design and test coverage. In: *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*. Washington,

DC, USA: IEEE Computer Society, 2007. (ESEM '07), p. 275–284. ISBN 0-7695-2886-4. Disponível em: <http://dx.doi.org/10.1109/ESEM.2007.2>.

SLYNGSTAD, Odd Petter N.; LI, Jingyue; CONRADI, Reidar; RØNNEBERG, Harald; LANDRE, Einar; WESENBERG, Harald. The impact of test driven development on the evolution of a reusable framework of components - an industrial case study. In: *Proceedings of the 2008 The Third International Conference on Software Engineering Advances*. Washington, DC, USA: IEEE Computer Society, 2008. (ICSEA '08), p. 214–223. ISBN 978-0-7695-3372-8. Disponível em: <http://dx.doi.org/10.1109/ICSEA.2008.8>.

STAA, Arndt Von. *Programação Modular*. 2. ed. Rio de Janeiro: Campus Elsevier, 2000. ISBN 8535206086.

STRODE, Diane E.; HUFF, Sid L.; HOPE, Beverley; LINK, Sebastian. Coordination in co-located agile software development projects. *The Journal of Systems and Software*, Elsevier Science Inc., New York, NY, USA, v. 85, n. 6, p. 1222–1238, jun. 2012. ISSN 0164-1212. Disponível em: <http://dx.doi.org/10.1016/j.jss.2012.02.017>.

TAHIR, Abbas; TOSI, Davide; MORASCA, Sandro. A systematic review on the functional testing of semantic web services. *Journal of Systems and Software*, Elsevier Science Inc., New York, NY, USA, v. 86, n. 11, p. 2877–2889, November 2013. ISSN 0164-1212. Disponível em: <http://dx.doi.org/10.1016/j.jss.2013.06.064>.

TRAINA, Agma Juci Machado; JUNIOR, Caetano Traina. Como fazer pesquisa bibliográfica. In: . São Paulo, SP, Brasil: Sociedade Brasileira de Computação, 2009. v. 2, n. 2.

TRAVASSOS, Guilherme H.; SANTOS, Paulo Sérgio Medeiros dos; MIAN, Paula Gomes; BIOLCHINI, Jorge. An environment to support large scale experimentation in software engineering. In: *13th IEEE International Conference on Engineering of Complex Computer Systems, 2008. ICECCS 2008.* [S.l.: s.n.], 2008. p. 193–202.

TSAI, Wei-Tek; PAUL, Ray; SONG, Weiwei; CAO, Zhibin. Coyote: An XML-Based framework for web services testing. In: *Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering*. Washington, DC, USA: IEEE Computer Society, 2002. (HASE '02), p. 173–. ISBN 0-7695-1769-2. Disponível em: <http://dl.acm.org/citation.cfm?id=795685.797694>.

TSAI, Wei-Tek; PAUL, Ray; WANG, Yamin; FAN, Chun; WANG, Dong. Extending WSDL to facilitate web services testing. In: *Proceedings. 7th IEEE International Symposium on High Assurance Systems Engineering, 2002.* [S.l.: s.n.], 2002. p. 171–172. ISSN 1530-2059.

TSAI, Wei-Tek; ZHANG, Dawei; PAUL, Raymond; CHEN, Yinong. Stochastic voting algorithms for web services group testing. In: *Proceedings of the Fifth International Conference on Quality Software*. Washington, DC, USA: IEEE Computer Society, 2005. (QSIC '05), p. 99–108. ISBN 0-7695-2472-9. Disponível em: <http://dx.doi.org/10.1109/QSIC.2005.58>.

TURNU, Ivana; MELIS, Marco; CAU, Alessandra; SETZU, Alessio; CONCAS, Giulio; MANNARO, Katiuscia. Modeling and simulation of open source development using an agile practice. *Journal of Systems Architecture*, v. 52, n. 11, p. 610 – 618,

2006. ISSN 1383-7621. Agile Methodologies for Software Production. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1383762106000634>.

VERGARA, Sylvia Constant. *Métodos de Coleta de Dados no Campo*. 2. ed. São Paulo: Atlas, 2013. ISBN 8522470537.

VERSIONONE. *9th Annual State of Agile Survey*. [S.l.], 2015. 1-16 p. Site: <https://www.versionone.com/pdf/state-of-agile-development-survey-ninth.pdf> [Último Acesso em: 26 Setembro 2015].

VIEIRA, Sônia. *Como Elaborar Questionários*. 1. ed. São Paulo: Atlas, 2009. ISBN 9788522455737.

VU, John Huan; FROJD, Niklas; SHENKEL-THEROLF, Clay; JANZEN, David S. Evaluating Test-Driven Development in an industry-sponsored capstone project. In: *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*. Washington, DC, USA: IEEE Computer Society, 2009. (ITNG '09), p. 229–234. ISBN 978-0-7695-3596-8. Disponível em: <http://dx.doi.org/10.1109/ITNG.2009.11>.

W3C. *Web Services Glossary*. 2015. <http://www.w3.org/TR/ws-gloss/>. Acessado em 05/01/2015.

YENDURI, Sumanth; PERKINS, Louise A. Impact of using Test-Driven Development: A case study. In: ARABNIA, Hamid R.; REZA, Hassan (Ed.). *Proceedings of the International Conference on Software Engineering Research and Practice & Conference on Programming Languages and Compilers, SERP 2006, Las Vegas, Nevada, USA, June 26-29, 2006, Volume 1*. [S.l.]: CSREA Press, 2006. p. 126–129. ISBN 1-932415-90-4.

YUAN, Yuan; LI, Zhongjie; SUN, Wei. A graph-search based approach to BPEL4WS test generation. In: *International Conference on Software Engineering Advances*. [S.l.: s.n.], 2006. p. 14–14.

YUE, Qiang; XU, Zhiwei; YU, Haiyan; LI, Wei; ZHA, Li. An approach to debugging grid or web services. In: *Web Services, 2007. ICWS 2007. IEEE International Conference on*. [S.l.: s.n.], 2007. p. 330–337.

ZAKARIA, Zulfa; ATAN, Rodziah; GHANI, Abdul Azim Abdul; SANI, Nor Fazlida Mohd. Unit testing approaches for BPEL: A systematic review. In: *Asia-Pacific Software Engineering Conference (APSEC)*. [S.l.: s.n.], 2009. p. 316–322. ISSN 1530-1362.

ZAMBIASI, Saulo Popov. *Uma Arquitetura de Referência para Softwares Assistentes Pessoais Baseada na Arquitetura Orientada a Serviços*. Tese (Doutorado em Engenharia de Automação e Sistemas) — Universidade Federal de Santa Catarina, Florianópolis - SC - Brasil, Março 2012. 295 pgs.

ZHANG, Jia; XU, Di. A mobile agent-supported web services testing platform. In: *Embedded and Ubiquitous Computing, 2008. EUC '08. IEEE/IFIP International Conference on*. [S.l.: s.n.], 2008. v. 2, p. 637–644.

ZHE, Li; MAIBAUM, Tom. An approach to integration testing of object-oriented programs. In: *Seventh International Conference on Quality Software, 2007*. [S.l.: s.n.], 2007. (QSIC '07), p. 268–273. ISSN 1550-6002.

Apêndice A

Método de Pesquisa da Revisão Sistemática

Esta seção apresenta o método utilizado para realizar a revisão sistemática da literatura conduzido neste trabalho.

Uma revisão sistemática é um estudo empírico em que uma questão de pesquisa ou hipótese é abordada para coletar e agregar evidências de uma série de estudos primários, por meio de um processo sistemático de pesquisa e extração de dados (CAUSEVIC; SUNDMARK; PUNNEKKAT, 2011).

O processo utilizado neste trabalho foi baseado no modelo proposto por Kitchenham et al. (2009). Optou-se pela revisão sistemática por ser a maneira mais difundida para elaborar síntese de trabalhos científicos de forma rigorosa, objetiva, repetível e com valor científico.

Com o objetivo de ampliar os resultados da busca dos estudos primários referente a este projeto, optou-se por realizar duas buscas independentes nas bibliotecas digitais. A primeira busca concentrou-se em encontrar trabalhos diversos sobre TDD e a segundo foi mais restrita e focou em localizar trabalhos referente ao uso de TDD no ambiente SOA e na construção de serviços web.

A.1 Processo de Busca

O processo de busca da revisão sistemática iniciou-se a partir da definição do protocolo de pesquisa, que define o propósito da revisão. Kitchenham e Charters (2007) destacam que definir as questões de pesquisa ou hipóteses é parte essencial da revisão sistemática pois elas irão guiar toda a metodologia da revisão. Portanto, como primeira tarefa, foram definidas as questões de pesquisas a seguir.

Questões referentes a busca de trabalhos sobre TDD:

1. Quais os principais efeitos obtidos ao aplicar a prática de TDD no desenvolvimento de software?

2. Em quais paradigmas de desenvolvimento a prática de TDD normalmente é aplicada?

3. Como a prática de TDD influencia a produtividade, a qualidade interna e a qualidade externa do software?

Questões referentes a busca de trabalhos sobre a adoção de TDD no desenvolvimento de serviços web:

1. Existe algum estudo que explora a aplicação de TDD no desenvolvimento de serviços web?
2. Quais os principais desafios ao aplicar TDD no desenvolvimento de serviços web?
3. Quais os efeitos obtidos ao aplicar a prática de TDD no desenvolvimento de serviços web?

Depois de definido as questões de pesquisa, a identificação dos estudos primários foi realizada nas bases de conhecimentos listadas na Tabela A.1 por meio da *string* de pesquisa definida na Seção A.2. Após a execução da pesquisa nas bases de conhecimento, os trabalhos foram selecionados e classificados conforme apresentado na Seção A.3. Já a Seção A.4 apresenta os critérios utilizados para selecionar os artigos.

A.2 Identificação dos Estudos Primários

Com o objetivo de encontrar os estudos relevantes, primeiramente foram identificados e selecionados os termos de busca mais importantes. Para defini-los, foi aplicada uma diretriz muito utilizada na área médica para identificar a eficácia de um tratamento, no qual sugere-se o uso de três pontos de vista: População, Intervenção e Resultados.

Essas diretrizes são apresentadas inicialmente em Kitchenham (2004) e estendidas mais tarde em Petticrew e Roberts (2005) com a seguinte definição:

- População: pode ser qualquer papel específico da engenharia de software ou a área de aplicação da pesquisa.
- Intervenção: são as tecnologias de software que tratam de questões específicas.
- Resultados: estão relacionados a fatores de importância aos pesquisadores.

Os termos foram definidos para cada um dos três pontos de vista em duas fases diferentes. A primeira fase considerou apenas os trabalhos relacionados a prática de TDD durante o desenvolvimento de software. Já a segunda fase considerou o uso de TDD no desenvolvimento de serviços web. A separação da busca em duas fases foi necessária para trazer um maior número de estudos referente a área de interesse deste trabalho.

Termos referentes a busca de trabalhos sobre TDD:

- **População:** *Software Engineers, Software Developers, Programmers.*
- **Intervenção:** *Test Driven Development, Test-Driven Development, TDD, Test First Development.*
- **Resultados:** *Code Quality, Quality Improvement, Design Improvement, Improved Software Development, Internal Quality, External Quality, Productivity.*

Termos referentes a busca de trabalhos sobre a adoção de TDD no desenvolvimento de serviços web:

- **População:** *Software Engineers, Software Developers, Programmers.*
- **Intervenção:** *(Web Services AND TDD), (Web Services AND Test-Driven Development), (SOA AND TDD), (SOA AND Test-Driven Development).*
- **Resultados:** *Code Quality, Quality Improvement, Design Improvement, Improved Software Development, Internal Quality, External Quality, Productivity.*

Além dos termos apresentados, também foi considerada a tradução livre para o português. A construção da *string* de pesquisa e a notação lógica para a busca é definida a seguir:

(P_1 OR P_2 ... OR P_n) AND (I_1 OR I_2 ... OR I_n)

No qual as P_n referem-se aos termos de População e I_n referem-se aos termos de Intervenção. Eles são relacionados por meio de operadores booleano *AND* e *OR*. Para ampliar os resultados da pesquisa somente os termos População e Intervenção foram considerados.

A inclusão do campo Resultados na pesquisa reduz notavelmente o volume de estudos relevantes identificados e portanto, aumenta o risco da falta destes estudos na pesquisa inicial (TAHIR; TOSI; MORASCA, 2013).

A *string* de pesquisa resultante foi utilizada para realizar a busca nas bibliotecas digitas apresentadas na Tabela A.1:

Tabela A.1: Relação das bibliotecas *online* pesquisadas.

Fonte	**Data da Pesquisa**
IEEE Xplore	18/12/2014
ScienceDirect	18/12/2014
ACM Digital Library	19/12/2014
CiteSeerx	20/12/2014

Traina e Junior (2009) afirmam que os ambientes principais e mais utilizados para realizar uma pesquisa bibliográfica, são as bibliotecas digitais das sociedades científicas, tais como: ACM e IEEE, que oferecem recursos para que sejam pesquisados artigos de veículos patrocinados por elas. Além dessas duas bibliotecas digitais citadas, este trabalho também considerou a ScienceDirect e a CiteSeerx para a pesquisa bibliográfica.

Durante a busca nas bibliotecas digitais descritas, algumas alterações foram necessárias para atender a configuração dos critérios de busca em cada site. A busca por artigos ficou restrita ao intervalo dos anos de publicação (de 1999 até 2014) nos idiomas inglês e português.

Esse intervalo de datas foi escolhido por considerar que o termo TDD passou a ser utilizado com mais frequência após o advento dos métodos ágeis, em particular o método XP que surgiu em 1999 conforme sugere Beck (1999). Os campos considerados na busca dos artigos foram: Resumo, Título e Palavras-Chave *(TITLE-ABS-KEY)*.

A.3 Processo de Seleção dos Artigos

O processo de seleção dos artigos foi realizado em quatro etapas de filtragem até chegar ao conjunto final dos estudos primários analisados. A Figura A.1 e a Figura A.2 detalham o processo executado.

Como as duas buscas por artigos foram realizadas de maneira independente uma da outra, o processo de identificação e seleção dos artigos também foi realizado de maneira independente e detalhado na Seção A.3.1 e Seção A.3.2.

A.3.1 Processo de Seleção dos Artigos sobre TDD no desenvolvimento de software

O fluxo apresentado na Figura A.1 corresponde a identificação e seleção dos artigos sobre a prática de TDD em ambientes diversos no desenvolvimento de software.

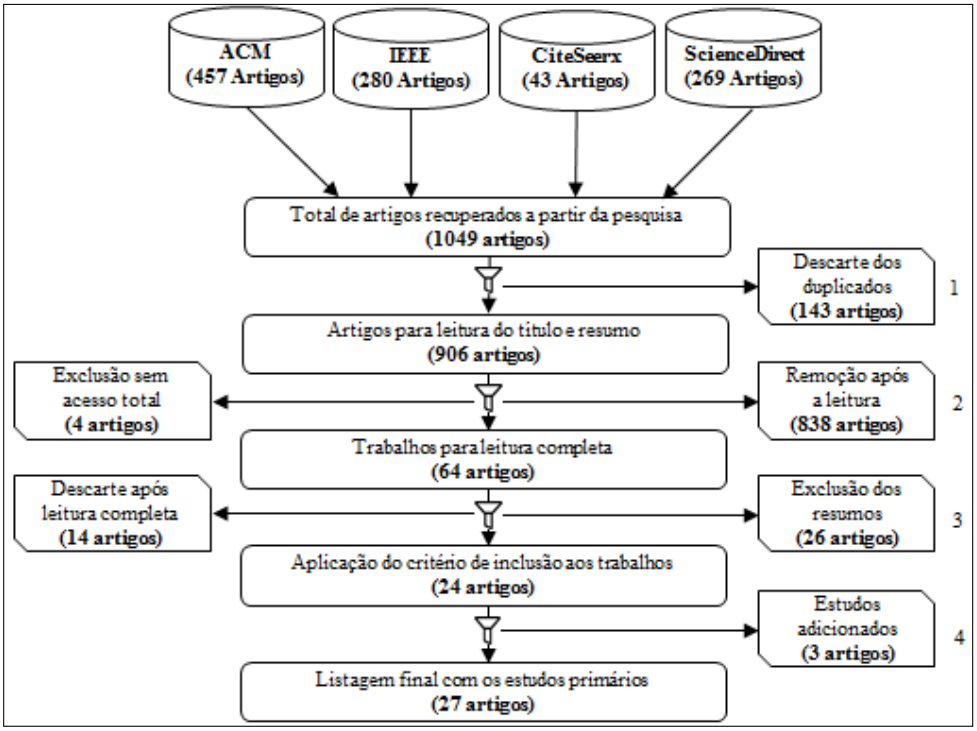

Figura A.1: Processo de revisão dos artigos sobre TDD. Fonte: Autoria própria.

Inicialmente foram obtidos 1049 artigos a partir da primeira *string* de pesquisa sobre TDD executada nas bases de conhecimentos descritas na Tabela A.1. Em seguida foram removidos 143 artigos que estavam em duplicidade. Na próxima fase foram retirados 4 artigos por não haver acesso completo ao material publicado e 838 trabalhos foram removidos após a leitura do título e resumo, restando 64 artigos para análise.

A maior parte dos artigos removidos nesta etapa foi por que não atenderam aos critérios inclusão 3 e 4 da Seção A.4 e também por que não faziam parte do contexto de experimentos em desenvolvimento de software.

Deste montante, os artigos que continham menos de 4 páginas também foram removidos por se tratar de resumos, nessa classificação foram encontrados 26 artigos. Além dos resumos, 14 trabalhos foram removidos após a leitura completa, por estarem aderente aos critérios de exclusão 1, 2 e 3 apresentado na Seção A.4.

Para complementar a lista de artigos encontrados, foi utilizada a técnica de amostragem conhecida como bola de neve (em inglês, *snowball*) descrita em Jalali e Wohlin (2012).

Essa técnica visa analisar as referências dos artigos, em especial as revisões sistemáticas, com o objetivo de encontrar estudos importantes que não foram selecionados durante as buscas nas bases de conhecimento.

Durante a leitura das referências dos trabalhos de revisão sistemática de Jeffries e Melnik (2007), Rafique e Misic (2013), Sfetsos e Stamelos (2010) e Munir, Moayyed e Petersen (2014), foram incluídos três novos artigos ao conjunto final, consolidando um total de 27 trabalhos selecionados sobre a aplicação de TDD no desenvolvimento de software.

A.3.2 Processo de Seleção dos Artigos sobre TDD no desenvolvimento de WS em SOA

O fluxo apresentado na Figura A.2 corresponde a identificação e seleção dos artigos sobre a prática de TDD no desenvolvimento de serviços web dentro da uma arquitetura orientada a serviços. Nesta busca os critérios foram mais restritos e direcionados para o desenvolvimento de WS dentro de uma arquitetura orientada a serviços.

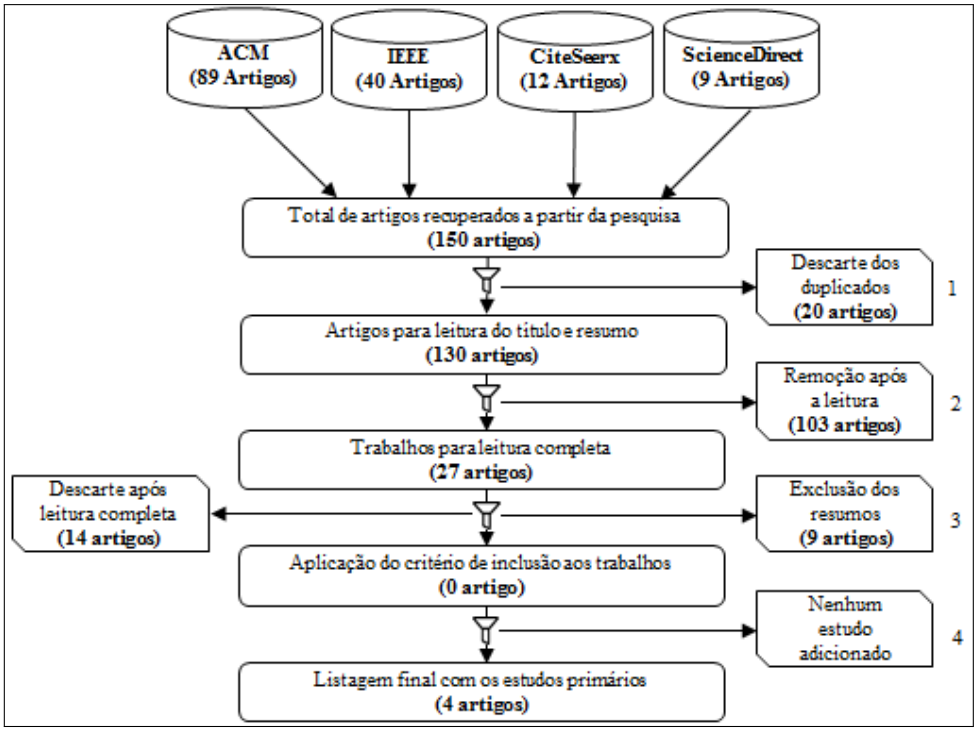

Figura A.2: Processo de revisão dos artigos sobre TDD no desenvolvimento de WS. Fonte: Autoria própria.

Inicialmente foram obtidos 150 artigos a partir da segunda *string* de pesquisa sobre a adoção de TDD no desenvolvimento de serviços web, executada nas bases de conhecimentos descritas na Tabela A.1. Em seguida foram removidos 20 artigos que estavam em duplicidade. Na fase seguinte 103 trabalhos foram removidos após a leitura do título e resumo, restando 27 artigos para análise.

Deste montante, os artigos que continham menos de 4 páginas também foram removidos por se tratar de resumos, nessa classificação foram encontrados 9 artigos. Além disso,

14 trabalhos foram removido após a leitura completa, por estarem aderente aos critérios de exclusão 1, 2 e 3 apresentado na Seção A.4.

Nenhum outro artigo adicional sobre TDD no desenvolvimento de serviços web foi encontrado nos trabalhos lidos. Dessa forma, apenas 4 artigos foram selecionados.

A.3.3 Avaliação dos Estudos Primários Encontrados

Para identificar e avaliar a qualidade dos estudos levantados no processo de seleção dos artigos, foram desenvolvidas sete perguntas de validação (PV), baseadas nas definições de Kitchenham e Charters (2007) e Dyba, Dingsoyr e Hanssen (2007).

As perguntas de validação foram divididas em três grupos Projeto, Análise e Conclusão. O grupo de Projeto contém as perguntas 1 e 2. O grupo de Análise possui as perguntas 3 e 4. O grupo de Conclusão é formado pelas últimas três perguntas, 5, 6 e 7. O detalhamento dessas questões é apresentado a seguir.

1. As técnicas apresentadas estão relacionadas com as questões de pesquisa que este trabalho deseja responder?
2. As métricas usadas nos estudos foram definidas por completo?
3. Existe alguma validação técnica confiável da abordagem apresentada?
4. Os participantes do estudo ou a unidade de observação foram definidos com clareza?
5. Os resultados são explicados e estão relacionados aos objetivos do estudo?
6. Os resultados obtidos foram validados e podem ser replicados?
7. As limitações do trabalho foram explicadas adequadamente?

Os trabalhos relacionados nesta revisão respondem a essas perguntas. Essa validação é necessária para aumentar a qualidade da revisão sistemática e entregar mais confiabilidade aos leitores.

A.4 Critérios de Inclusão e Exclusão dos Artigos

O critério de inclusão dos artigos encontrados nas bibliotecas digitais, relacionadas na Tabela A.1, considerou as 5 definições a seguir relacionada para incluir algum trabalho nesta revisão sistemática. A inclusão do artigo somente acontecerá se o trabalho candidato atenda a todas as definições a seguir e não viole nenhum dos critérios de exclusão.

1. Os artigos devem estar disponíveis em sua totalidade;
2. Os artigos precisam ter sido publicados previamente em algum meio científico (periódicos ou eventos);
3. Os trabalhos devem apresentar resultados sobre a prática de TDD no desenvolvimento de software, principalmente com foco em serviços web;

4. Os estudos precisam apresentar adequadamente a descrição do contexto em que o estudo foi conduzido;

5. Os resultados apresentados devem ser baseados em algum critério de medição.

Além do critério de inclusão, foi definido outros 6 critérios para exclusão dos trabalhos encontrados. Os trabalhos foram excluídos do processo de revisão sistemática, caso atenda algum dos critérios apresentados a seguir.

1. Artigos duplicados;

2. Estudos que não estão relacionados a prática de TDD no desenvolvimento de software;

3. Trabalhos de revisão sistemática acerca da prática de TDD;

4. Artigos que não estejam escritos em língua inglesa ou portuguesa;

5. Trabalhos que não foram publicados em algum meio científico;

6. Artigos que não estejam totalmente acessíveis;

O objetivo desses critérios é direcionar e manter o foco da leitura dos artigos e aumentar a assertividade na seleção dos trabalhos. Caso haja alguma exceção a essas regras, uma justificativa deve ser apresentada.

As seções anteriores que descreveram o processo de busca, identificação, seleção e os critérios de inclusão/exclusão dos artigos correspondem, na ordem cronológica, ao fluxo completo da execução da revisão sistemática.

Apêndice B

Análise Detalhada do Estado da Arte

Esta seção fornece uma análise detalhada dos principais artigos relacionados ao tema deste trabalho. As Seções B.1 e B.2 classificam e analisam os trabalhos encontrados a partir da revisão sistemática descrita no Apêndice A. Já a Seção B.3 apresenta uma visão geral sobre os artigos que abordam o teste de serviços web.

B.1 Análise dos Estudos sobre TDD no Desenvolvimento de Software

No total, 27 estudos foram selecionados e analisados a partir da primeira revisão da literatura, detalhada no Apêndice A. Esses estudos foram publicados entre os anos de 1999 até 2014 e fazem uma comparação entre as práticas de TDD e TLD. Na comparação entre as duas práticas de desenvolvimento é que os efeitos são identificados e detalhados.

A Tabela B.1 lista todos os trabalhos encontrados e selecionados referente ao uso de TDD no desenvolvimento de software. Essa tabela descreve ainda qual o ano de publicação, cenário em que o estudo foi aplicado, linguagem de programação utilizada no estudo e o perfil dos participantes (profissionais de mercado ou estudantes universitários).

Como pode-se observar na Tabela B.1, o cenário no qual os estudos foram aplicados ficaram divididos em dois tipos: Acadêmico (48,14%) e Industrial (44,46%). Outros dois estudos, ou 7,40% do total, usaram os dois cenários e para isso foi utilizada a denominação de Misto.

O intervalo dos anos de publicação considerado durante o processo de busca dos artigos foi de 1999 até 2014. A distribuição da quantidade de trabalhos por ano de publicação está descrita na Figura B.1. Dentro do escopo deste trabalho, os primeiros artigos publicados surgiram no ano de 2003 com três publicações, já no ano de 2006 esse número dobrou. As pesquisas continuaram durante os anos seguintes, porém com menor intensidade do que a apresentada no ano de 2006.

Relacionado a linguagem de programação, praticamente todos os estudos utilizaram Java ou C++ para a aplicação do trabalho. A maior parte dos estudos, cerca de 23 deles, foram publicados antes de 2010.

Os participantes dos estudos publicados também estão separados quase que na mesma proporção que os cenários foram usados. Existem dois tipos de participantes, os Estudantes e os Profissionais, cada um deles presentes em 44,46% e 48,14% dos estudos respectivamente.

Tabela B.1: Estudos empíricos sobre TDD.

Estudo	**Cenário**	**Linguagem**	**Participante**
Siniaalto e Abrahamsson (2007)	Industrial	NI*	Profissionais
Geras, Smith e Miller (2004)	Acadêmico	NI	Profissionais
George e Williams (2004)	Industrial	Java	Profissionais
Gupta e Jalote (2007)	Acadêmico	Java	Estudantes
George e Williams (2003)	Industrial	Java	Profissionais
Maximilien e Williams (2003)	Industrial	Java	Profissionais
Janzen e Saiedian (2008a)	Misto	Java	Misto
Vu et al. (2009)	Acadêmico	Java, ActionScript	Estudantes
Pancur et al. (2003)	Acadêmico	Java	Estudantes
Janzen, Turner e Saiedian (2007)	Acadêmico	Java, C++	Estudantes
Edwards (2004)	Acadêmico	Java	Estudantes
Janzen e Saiedian (2008b)	Acadêmico	C++	Estudantes
Canfora et al. (2006)	Industrial	Java	Profissionais
Bhat e Nagappan (2006)	Industrial	C, C#, C++	Profissionais
Turnu et al. (2006)	Industrial	NI	Estudantes
Janzen e Saiedian (2006)	Acadêmico	Java	Estudantes
Erdogmus, Morisio e Torchiano (2005)	Acadêmico	Java	Estudantes
Sanchez, Williams e Maximilien (2007)	Industrial	Java	Profissionais
Aniche e Gerosa (2012)	Industrial	Java	Profissionais
Slyngstad et al. (2008)	Industrial	Java	Profissionais
Pancur e Ciglaric (2011)	Acadêmico	Java	Estudantes
Aniche e Gerosa (2010)	Misto	NI	Misto
Damm e Lundberg (2006)	Industrial	Java, C++	Profissionais
Madeyski (2010)	Acadêmico	Java	Estudantes
Madeyski e Szala (2007)	Acadêmico	Java	Estudantes
Nagappan et al. (2008)	Industrial	Java, C++, .NET	Profissionais
Yenduri e Perkins (2006)	Acadêmico	Java	Estudantes

*NI: Não Informado.

Outros dois estudos, ou 7,40% do total, usaram os dois tipos de participantes e para isso foi utilizada a denominação de Misto.

Uma outra classificação realizada foi com relação ao método de estudo aplicado. Os 4 métodos de pesquisa encontrados nestes trabalhos foram: Experimento (57,14%), Estudo de Caso (32,14%), Questionário (7,14%) e Simulação (3,58%).

Janzen e Saiedian (2008a) apresentam dois métodos de validação, sendo um Experimento e o outro Estudo de Caso. Neste caso ele foi contabilizado duas vezes, uma em cada método. Portanto, a soma da quantidade de trabalhos apresentados na Tabela B.2 é maior que a quantidade total de trabalhos selecionados nesta revisão.

A Tabela B.2 apresenta uma listagem consolidada dos métodos de pesquisa encontrados nos estudos analisados. O método mais utilizado foi o Experimento com 16 estudos

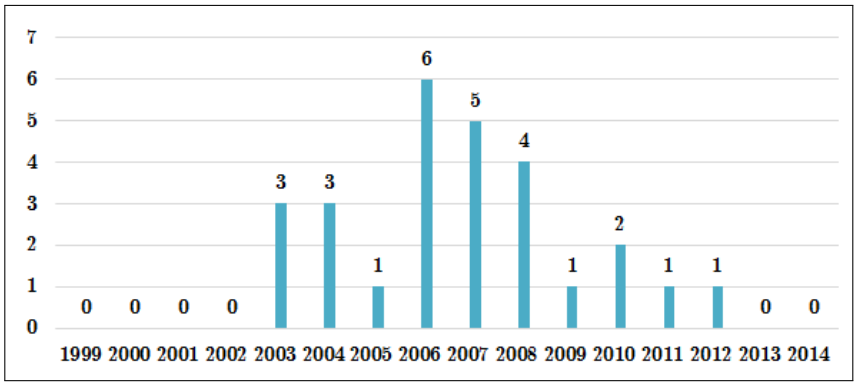

Figura B.1: Distribuição dos trabalhos por ano de publicação.

Tabela B.2: Mapeamento dos métodos de pesquisa usados nos artigos.

Método	**Estudos**	**Nº Estudos**
Experimento	Geras, Smith e Miller (2004), George e Williams (2004), Gupta e Jalote (2007), George e Williams (2003), Maximilien e Williams (2003), Janzen e Saiedian (2008a), Vu et al. (2009), Pancur et al. (2003), Janzen, Turner e Saiedian (2007), Edwards (2004), Janzen e Saiedian (2008b), Canfora et al. (2006), Janzen e Saiedian (2006), Erdogmus, Morisio e Torchiano (2005), Pancur e Ciglaric (2011), Madeyski (2010)	16
Estudo de Caso	Madeyski e Szala (2007), Janzen e Saiedian (2008a), Bhat e Nagappan (2006), Sanchez, Williams e Maximilien (2007), Slyngstad et al. (2008), Damm e Lundberg (2006), Madeyski e Szala (2007), Nagappan et al. (2008), Yenduri e Perkins (2006)	9
Questionário	Aniche e Gerosa (2010), Aniche e Gerosa (2012)	2
Simulação	Turnu et al. (2006)	1

classificados, seguido pelo Estudo de Caso com 9 trabalhos. Apenas dois estudos utilizaram o método Questionário e somente um estudo utilizou a Simulação para conduzir as pesquisas.

Após realizar a categorização dos estudos primários, a Seção B.1.1, Seção B.1.2 e Seção B.1.3 apresentarão os efeitos observados e descritos nos trabalhos.

As tabelas a seguir apresentam alguns trabalhos repetidos em mais de uma classificação porque eles utilizaram mais de um cenário ou método para aplicação do estudo, dessa forma são considerados estudos independentes. Os estudos com cenário misto (acadêmico e industrial) são: Janzen e Saiedian (2008a) e Aniche e Gerosa (2010). Janzen e Saiedian (2008a) utilizaram o experimento e estudo de caso para a aplicação do estudo.

Os resultados apresentados nesta seção não podem ser comparados completamente por que não há uma uniformidade e padronização nas métricas utilizadas pelos estudos. Mas as evidências foram resumidas e apresentadas nas tabelas baseando-se nas métricas mencionadas em cada um dos estudos analisados.

B.1.1 Qualidade Interna

A Tabela B.3 apresenta a lista dos trabalhos e os efeitos obtidos referentes a qualidade interna de software. O principal indicador utilizado para identificar os efeitos que a prática de TDD produz para a qualidade interna foi a cobertura de teste no código fonte gerado. É sabido que esse indicador analisado separadamente não garante que o código possui um elevado nível de qualidade interna. A cobertura de teste do código por sua vez, pode indicar que poucos defeitos sejam apresentados pelo software. Contudo, a forma de medição e os resultados dos trabalhos analisados não foram questionados neste trabalho.

Na Tabela B.3 os trabalhos contidos na coluna **(+) QI** são os que apresentaram resultados positivos e aumentaram a qualidade interna do software quando a prática de TDD foi aplicada. Na coluna **(=) QI** estão os trabalhos que não identificaram nenhum efeito ao aplicar as duas práticas. Já a coluna **(-) QI** contém os trabalhos que observaram efeitos negativos para a qualidade interna ao aplicar a prática de TDD, ou seja, a qualidade interna diminuiu.

Tabela B.3: Estudos que reportaram os efeitos de TDD para a qualidade interna.

Método	**Cenário**	**(+) QI**	**(=) QI**	**(-) QI**
Experimento	Acadêmico	Geras, Smith e Miller (2004), Janzen e Saiedian (2008a), Vu et al. (2009), Janzen, Turner e Saiedian (2007), Erdogmus, Morisio e Torchiano (2005), Pancur e Ciglaric (2011)	Janzen e Saiedian (2008b), Madeyski (2010)	Pancur et al. (2003), Janzen e Saiedian (2006)
Experimento	Industrial	George e Williams (2004), George e Williams (2003), Maximilien e Williams (2003), Janzen e Saiedian (2008a), Canfora et al. (2006)		
Estudo de Caso	Acadêmico	Janzen e Saiedian (2008a), Yenduri e Perkins (2006)		
Estudo de Caso	Industrial	Siniaalto e Abrahamsson (2007), Janzen e Saiedian (2008a), Bhat e Nagappan (2006), Sanchez, Williams e Maximilien (2007)		
Questionário	Acadêmico	Aniche e Gerosa (2010)		
Questionário	Industrial	Aniche e Gerosa (2010)	Aniche e Gerosa (2012)	
Simulação	Industrial	Turnu et al. (2006)		

Analisando a Tabela B.3 em detalhes, pode-se observar que os experimentos são a forma mais comum de validação dos estudos que visam identificar os efeitos de TDD na qualidade interna do software.

É visível uma diferença nos resultados quando comparado o cenário acadêmico com o industrial usando o experimento como método de validação. Os experimentos que foram aplicados no cenário industrial identificaram que TDD contribuiu para aumentar a qualidade interna do software, já no cenário acadêmico a maioria, cerca de 60% dos estudos, também identificaram que TDD ajuda aumentar a qualidade interna do software, porém Janzen e Saiedian (2008b) e Madeyski (2010) não identificaram diferenças entre o uso das práticas e os trabalhos. Pancur et al. (2003) e Janzen e Saiedian (2006) apontaram que a prática de TDD diminuiu a qualidade interna se comparado com a TLD.

Considerando somente os estudos únicos, ou seja, contabilizando os trabalhos de Janzen e Saiedian (2008a) e Aniche e Gerosa (2010) apenas uma vez, obtém-se um total de 21 trabalhos que fizeram a comparação entre TDD e TLD. Desse total, cerca de 76% disseram que houve melhora significativa na qualidade interna do software. Se considerar as repetições, esse total sobe para 80%.

Os estudos que visam identificar o impacto do TDD sobre a melhoria do código escrito e principalmente sobre o projeto de classes, ainda são muito limitados (JOSEFSSON, 2004). Aniche e Gerosa (2012) realizaram um novo levantamento sobre os trabalhos e também chegou a mesma conclusão, estudos nesta área ainda são limitados e por este motivo os impactos do TDD sobre a qualidade, focando em projeto de classe, ainda não podem ser explicados.

B.1.2 Qualidade Externa

A Tabela B.4 apresenta a lista dos trabalhos e os efeitos obtidos referentes a qualidade externa de software. O principal indicador utilizado para identificar os efeitos foi a execução de testes caixa preta para mensurar a quantidade de casos de teste que passaram ou falharam de acordo com a prática utilizada (TDD e TLD).

Na Tabela B.4 os trabalhos contidos na coluna **(+) QE** são os que apresentaram resultados positivos e foi percebido um aumento na qualidade externa do software quando a prática de TDD foi aplicada. Na coluna **(=) QE** estão os trabalhos que não identificaram nenhum efeito ao aplicar as duas práticas. Já a coluna **(-) QE** contém os trabalhos que observaram efeitos negativos para a qualidade externa ao aplicar a prática de TDD, ou seja, a qualidade externa diminuiu.

Tabela B.4: Estudos que reportaram os efeitos de TDD para a qualidade externa.

Método	**Cenário**	**(+) QE**	**(=) QE**	**(-) QE**
Experimento	Acadêmico	Geras, Smith e Miller (2004), Vu et al. (2009), Edwards (2004), Pancur e Ciglaric (2011)	Gupta e Jalote (2007)	Pancur et al. (2003)
Experimento	Industrial	George e Williams (2004), George e Williams (2003), Maximilien e Williams (2003)		
Estudo de Caso	Acadêmico	Yenduri e Perkins (2006)		
Estudo de Caso	Industrial	Bhat e Nagappan (2006), Slyngstad et al. (2008), Damm e Lundberg (2006), Nagappan et al. (2008)		
Questionário	Acadêmico	Aniche e Gerosa (2010)		
Questionário	Industrial	Aniche e Gerosa (2010)		
Simulação	Industrial	Turnu et al. (2006)		

Ao observar a Tabela B.4 em detalhes, foi possível perceber que os experimentos e os estudos de caso são as formas mais comuns de validação dos estudos que visam identificar os efeitos de TDD na qualidade externa do software.

Gupta e Jalote (2007) identificou que a qualidade externa do software se manteve a mesma ao comparar a prática de TDD com a TLD. Já Pancur et al. (2003) identificou que ao usar a prática de TDD a qualidade externa do software diminuiu. Esses dois estudos citados foram aplicados no cenário acadêmico. Todos os outros estudos listados na Tabela B.4 encontraram evidências de que TDD contribuiu para melhorar a qualidade externa do software.

Considerando somente os trabalhos únicos, ou seja, contabilizando o trabalho de Aniche e Gerosa (2010) apenas uma vez, obtém-se um total de 16 trabalhos que fizeram a comparação entre TDD e TLD. Desse total, cerca de 88% disseram que houve melhora significativa na qualidade externa do software.

B.1.3 Produtividade

A Tabela B.5 apresenta a lista dos trabalhos e os efeitos obtidos referentes a produtividade. O principal indicador utilizado para identificar se houve alteração na produtividade foi a comparação do tempo gasto para implementar uma nova funcionalidade e a quantidade de código gerado.

Na Tabela B.5 os trabalhos contidos na coluna **(+) Prd.** são os que apresentaram um aumento na produtividade dos desenvolvedores quando a prática de TDD foi aplicada. Na coluna **(=) Prd.** estão os trabalhos que não identificaram nenhum efeito ao aplicar as duas práticas. Já a coluna **(-) Prd.** contém os trabalhos que observaram uma diminuição na produtividade dos desenvolvedores quando a prática de TDD foi aplicada.

Tabela B.5: Estudos que reportaram os efeitos de TDD na produtividade.

Método	**Cenário**	**(+) Prd.**	**(=) Prd.**	**(-) Prd.**
Experimento	Acadêmico	Gupta e Jalote (2007), Janzen e Saiedian (2006), Pancur e Ciglaric (2011)	Geras, Smith e Miller (2004), Janzen e Saiedian (2008b), Erdogmus, Morisio e Torchiano (2005)	Vu et al. (2009)
Experimento	Industrial		Maximilien e Williams (2003)	George e Williams (2004), George e Williams (2003), Canfora et al. (2006)
Estudo de Caso	Acadêmico	Madeyski e Szala (2007), Yenduri e Perkins (2006)		
Estudo de Caso	Industrial		Siniaalto e Abrahamsson (2007)	Bhat e Nagappan (2006), Sanchez, Williams e Maximilien (2007), Nagappan et al. (2008)
Simulação	Industrial			Turnu et al. (2006)

Os resultados apresentados na Tabela B.5 são contraditórios ao comparar as diferenças entre os cenários industrial e acadêmico. Nenhum dos estudos aplicados em um cenário industrial apresentou um aumento na produtividade, no máximo se manteve. Já os estudos que foram aplicados no cenário acadêmico, apenas Vu et al. (2009) observou uma redução na produtividade.

Somente os trabalhos de Gupta e Jalote (2007), Janzen e Saiedian (2006), Pancur e Ciglaric (2011), Madeyski e Szala (2007) e Yenduri e Perkins (2006), aplicados em ambiente acadêmico apontaram que TDD aumentou a produtividade no desenvolvimento de software. A produtividade foi o quesito que mais gerou resultados diferenciados, mas a maioria dos trabalhos apontaram para uma diminuição na produtividade ao aplicar TDD frente a prática de TLD.

Conforme apresentado na Tabela B.5, 18 trabalhos fizeram a comparação entre TDD e TLD levando em consideração a produtividade. Desse total, cerca de 44% disseram que a produtividade dos desenvolvedores diminuiu, empatados com 28% cada, estão os demais trabalhos que indicaram que a produtividade se manteve ou aumentou.

B.2 Análise dos Estudos sobre TDD no Desenvolvimento de WS em SOA

No total, 4 estudos foram selecionados e analisados nesta segunda revisão da literatura. O cenário no qual os estudos foram aplicados ficaram divididos em dois tipos de cenários: Acadêmico (com 3 estudos) e Industrial (com 1 estudo).

A Tabela B.6 a seguir lista todos os trabalhos encontrados e selecionados referente ao uso de TDD no desenvolvimento de WS. Essa tabela descreve ainda qual o ano de publicação, cenário em que o estudo foi aplicado, linguagem de programação utilizada no estudo e o perfil dos participantes (profissionais de mercado ou estudantes universitários). O trabalho de Besson et al. (2015) foi retornado na busca que considerou os artigos publicados entre 1999 a 2014. O artigo já estava na base de conhecimento, mas ele foi publicado na edição de Janeiro/2015.

Tabela B.6: Estudos empíricos sobre TDD aplicado no desenvolvimento de WS.

Estudo	**Ano**	**Cenário**	**Linguagem**	**Participante**
Laranjeiro e Vieira (2009)	2009	Acadêmico	Java	Profissionais
Hamill, Alexander e Shasharina (2009)	2009	Industrial	Java, IDL, C++	Estudantes
Schneider e German (2013)	2013	Acadêmico	NI*	Estudantes
Besson et al. (2015)	2015	Acadêmico	Java	Estudantes

*NI: Não Informado.

Relacionado a linguagem de programação, praticamente todos os estudos utilizaram Java para a aplicação do trabalho, outras linguagens como C++ e IDL também foram utilizadas em alguns trabalhos. Os participantes classificados com Estudantes participaram em três dos quatro estudos e apenas um estudo contou com a participação de Profissionais.

Todos os 4 estudos foram publicados a partir de 2009, sendo que um deles foi publicado recentemente em 2015. Isso mostra que existe um interesse na comunidade científica sobre novos trabalhos nesta área.

Conforme apresentado anteriormente, poucos trabalhos sobre a adoção de TDD no desenvolvimento de serviços web foram encontrados. Além disso, dos 4 trabalhos analisados somente os artigos Laranjeiro e Vieira (2009), Hamill, Alexander e Shasharina (2009) e Besson et al. (2015) focam exatamente em TDD. Schneider e German (2013) apresentam uma abordagem análoga ao TDD e por isso também foi considerado neste estudo.

Laranjeiro e Vieira (2009) destacam a importância em diminuir os problemas de robustez, que segundo eles é um problema comum apresentado por diversos serviços web ao serem implantados nos servidores de produção. Com o objetivo de diminuir os problemas de robustez o estudo conduzido por Laranjeiro e Vieira (2009) fez uma extensão da prática de TDD voltada para eliminar problemas relacionados a robustez. Para validação do estudo três desenvolvedores experientes foram selecionados para executar o experimento.

O uso da prática de TDD, adaptada para robustez de serviços web, melhorou a qualidade do WS e nenhum problema relacionado a robustez foi encontrado no experimento. Já o experimento desenvolvido sem utilizar esta prática apresentou problemas de robustez (LARANJEIRO; VIEIRA, 2009).

O trabalho Hamill, Alexander e Shasharina (2009) ainda encontra-se nos estágios iniciais e de acordo com o material apresentado, não foram realizadas validações dos resultados apresentados. Os testes em serviços web são fundamentais para facilitar a adoção do TDD em SOA (HAMILL; ALEXANDER; SHASHARINA, 2009). Ainda segundo os autores essa abordagem traz diversas vantagens, no entanto, elas não são apresentadas. Este estudo não apresenta qualquer validação que comprove como eles chegaram a esta conclusão.

Em Schneider e German (2013), uma abordagem para o desenvolvimento ágil dirigido a testes visa combinar técnicas de simulação baseada em modelos para melhorar a qualidade no

desenvolvimento de software. Uma das ideias centrais dessa abordagem é viabilizar os testes o quanto antes no processo de desenvolvimento por meio de simulação a partir dos modelos definidos.

O objetivo da abordagem proposta por Schneider e German (2013) é detectar os problemas funcionais e de modelagem nas fases iniciais do processo de desenvolvimento de software. A implementação inicial da ferramenta foi denominada de VeriTAS. Os autores não comprovaram nenhum benefício concreto referente ao uso dessa abordagem, eles apenas mostraram ser viável o uso da ferramenta e da abordagem na construção de um serviço web.

Besson et al. (2015) focam em viabilizar o uso de TDD para a coreografias de serviços web para facilitar a construção de sistemas descentralizados. Os autores apresentam uma ferramenta denominada Rehearsal para suportar a criação de testes automatizados no tempo de desenvolvimento. Dentre os quatro estudos encontrados sobre TDD no desenvolvimento de WS, esse é o mais maduro e foi o que apresentou o maior número de validações dos resultados.

A ferramenta desenvolvida e apresentada por Besson et al. (2015) possui características para gerar clientes de serviços web, interceptação de mensagem, criação de objetos falsos, abstração da coreografia e etc. Essa ferramenta possui recursos adicionais quando comparada com outras, como por exemplo o SOCT (BARTOLINI et al., 2009), WS-TAXI (BARTOLINI et al., 2009) e BPELUnit (MAYER; LüBKE, 2006).

Tanto o método quanto a ferramenta apresentada permitem maior disciplina e robustez para o desenvolvimento de coreografias de serviços em SOA (BESSON et al., 2015).

Portanto, segundo as observações feitas nos quatro trabalhos apresentados nesta seção, a prática de TDD traz benefícios ao desenvolvimento de serviços web e precisa ser explorada para que novas ferramentas e abordagens sejam propostas.

B.3 Visão Geral sobre Testes em Serviços Web

Conforme descrito na Seção B.2 existem poucos estudos que tratem da adoção da prática de TDD no desenvolvimento de serviços web em aplicações distribuídas, se comparado uso dessa técnica para a criação de aplicações centralizadas. A partir desta premissa, uma outra busca baseada nas revisões sistemáticas elaboradas por Canfora e Di Penta (2006), Bozkurt, Harman e Hassoun (2013), Kalamegam e God (2012) foi realizada de maneira superficial com o objetivo de identificar quais as técnicas e métodos estão sendo estudados para testar e validar os serviços web.

As pesquisas na área de teste de serviços web ainda estão em estágios iniciais (CANFORA; DI PENTA, 2006). Ladan (2010) também afirma que o teste de serviços web é uma área recente de investigação. Inúmeras contribuições têm sido apresentadas na literatura, principalmente nas áreas de testes de integração, unitário, regressão e não-funcional.

Canfora e Di Penta (2006) mostram que ainda existe uma grande demanda por pesquisas para explorar essa área. Bozkurt, Harman e Hassoun (2013) apresenta algumas classificações das abordagens de testes em serviços web. As classificações sugeridas por Bozkurt, Harman e Hassoun (2013) são: (1) Geração de caso de teste; (2) Teste de partição; (3) Teste unitário de serviços web; (4) Teste baseado em modelo e verificação formal; (5) Teste baseado em contrato; (6) Teste baseado em defeitos; (7) Teste colaborativo; (8) Teste de regressão; (9) Teste de interoperabilidade; (10) Teste de integração.

Bai et al. (2005), Bartolini et al. (2009), Hanna e Munro (2009), Bartolini et al. (2008), Huang, Tsai e Paul (2005) apresentam técnicas para a geração de casos de teste para WS e

para *Web Services Business Process Execution Language* (WS-BPEL) a partir de contratos bem definidos, apoiados em especificações abertas como o WSDL.

Outros trabalhos que contribuem para a geração de casos de teste são os Tsai et al. (2002a), Di Penta et al. (2007), Bertolino et al. (2007) que por sua vez se baseiam em *schemas* XML bem definidos. Tsai et al. (2002a) e Di Penta et al. (2007) se voltam para o teste do código fonte dos serviços web baseando-se nestes contratos.

Tsai et al. (2005), Noikajana e Suwannasart (2008), Zhang e Xu (2008), Feudjio e Schieferdecker (2009), Yuan, Li e Sun (2006), Hou et al. (2009) e Belli e Linschulte (2008) também focam na geração casos de teste, porém, a partir de modelos. Nestes trabalhos é comum o uso do gráfico de controle de fluxos, ou *Control Flow Graph* (CFG), para mapear os processos escritos em *Business Process Execution Language* (BPEL), como nos trabalhos de Yuan, Li e Sun (2006) e Hou et al. (2009).

Os trabalhos que focam na geração de casos de teste baseados em contratos são: Tsai et al. (2002b), Mei e Zhang (2005), Nebut et al. (2003), Liang e Xu (2006), Jiang et al. (2005) e Zhe e Maibaum (2007). Dois deles, Tsai et al. (2002b) e Mei e Zhang (2005), propõe que o WSDL seja estendido para viabilizar a execução de testes caixa preta.

O foco dos estudos apresentados na Tabela B.7 está relacionado à geração de caso de testes ou dados para teste de serviços web e processos BPEL. Estes estudos foram conduzidos e direcionados, em sua maioria, para a técnica denominada caixa preta.

Tabela B.7: Classificação dos artigos sobre teste de serviços web.

Geração de Caso de Teste Baseado em:		
Especificação	**Modelos**	**Contrato**
Hanna e Munro (2009), Bartolini et al. (2008), Bai et al. (2005), Huang, Tsai e Paul (2005), Bartolini et al. (2009), Tsai et al. (2002a), Di Penta et al. (2007), Bertolino et al. (2007)	Tsai et al. (2005), Noikajana e Suwannasart (2008), Zhang e Xu (2008), Feudjio e Schieferdecker (2009), Yuan, Li e Sun (2006), Hou et al. (2009), Belli e Linschulte (2008)	Tsai et al. (2002b), Mei e Zhang (2005), Nebut et al. (2003), Liang e Xu (2006), Jiang et al. (2005), Zhe e Maibaum (2007)

Em todas as três classificações, os geradores de caso de testes podem criar vários casos rapidamente, porém a validação e a verificação das funcionalidades são feitas, na maior parte, manualmente.

Somente Tsai et al. (2002a) e Di Penta et al. (2007) focam os estudos na técnica caixa branca. Existem poucos trabalhos que buscam melhorias nas áreas de teste de unidade em WS.

Além dos estudos apresentados na Tabela B.7, os trabalhos Yue et al. (2007) e Lenz, Chimiak-Opoka e Breu (2007) estão relacionados a técnica de teste caixa preta, porém, propõem uma outra abordagem. Yue et al. (2007) recomendam uma ferramenta capaz de rastrear o comportamento de serviços, gravar informações de depuração (*debug*) e gerenciar o estado e comportamentos dos serviços depurados. Já Lenz, Chimiak-Opoka e Breu (2007) apresentam uma ferramenta capaz de gerar testes de unidade em JUnit a partir da especificação de requisitos e testes baseados em *Unified Modeling Language* (UML). Contudo, nenhum destes trabalhos apresentados sobre teste unitário utilizou TDD para o desenvolvimento e teste dos serviços.

Bartolini et al. (2011) apresentam uma abordagem de testes com *feedback* do código no modelo orientado a serviço. Essa abordagem foi nomeada originalmente de *Service Oriented*

Coverage Testing (SOCT) (BARTOLINI; BERTOLINO; MARCHETTI, 2008). O foco dessa abordagem e viabilizar a técnica de teste caixa branca no desenvolvimento de componentes na arquitetura SOA.

A técnica de teste caixa preta é mais utilizada pelos pesquisadores para desenvolver mecanismos que possibilitem o teste de componentes de terceiros sem o acesso ao código fonte. O uso da técnica caixa branca para o teste de serviços web ainda é pequena se comparado com a técnica caixa preta.

Apêndice C

Questionário sobre TDD

Este questionário possui quatro páginas sendo as duas primeiras referente a apresentação da pesquisa e o termo de consentimento livre e esclarecido. As outras duas contém os campos de identificação e as 19 perguntas de múltipla escolha sobre as experiências profissionais e os conhecimentos gerais de tecnologia da informação com o foco em testes unitários e a prática de TDD.

Todos os participantes da pesquisa assinaram o termo de consentimento livre e esclarecido, de acordo com as normas da UTFPR. Todo o conteúdo do questionário de pesquisa é apresentado a seguir.

Apresentação da Pesquisa

Este questionário faz parte de uma pesquisa de mestrado do Programa de Pós-Graduação em Computação Aplicada (PPGCA) do DAINF/UTFPR, conduzida pelos pesquisadores *Wilson Bissi, Adolfo Gustavo Serra Seca Neto e Maria Cláudia Figueiredo Pereira Emer*. A pesquisa é de cunho estritamente científico e refere-se ao desenvolvimento de software, com foco na prática de desenvolvimento guiado por teste (TDD).

Nota: O que é TDD? TDD é uma prática de desenvolvimento de software em que os testes unitários automatizados são escritos de forma incremental, antes da implementação do código fonte.

Pré-requisitos:

- Ter experiência em desenvolvimento de software;
- Conhecer o básico sobre TDD.

Participando você estará:

- Contribuindo para uma pesquisa científica na área da Computação, especificamente na área da Engenharia de Software;
- Expressando sua opinião sobre o desenvolvimento de software e ajudando o desenvolvimento de novos estudos na área de engenharia de software.

Os pesquisadores se comprometem a:

- Não divulgar, em qualquer meio, os dados ou informações pessoais dos participantes;
- Não divulgar, em qualquer meio, os dados ou informações individualizadas coletadas durante o processo de pesquisa com os participantes;
- Divulgar, em formato de dissertação, artigos e apresentações, apenas os dados consolidados, de maneira que não possam fornecer ou inferir a identificação de qualquer participante.

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

Eu _____, estou sendo convidado a participar de um estudo denominado: "TDD: Mapeamento de Uso na Indústria", cujo objetivo é identificar e mapear as características percebidas pelo voluntário da pesquisa na utilização do TDD durante o desenvolvimento de um software. Sei que para o avanço da pesquisa a participação de voluntários é de fundamental importância. Caso aceite participar desta pesquisa eu, responderei a um questionário de 19 questões sobre o assunto. Estou ciente de que minha privacidade será respeitada, ou seja, meu nome, ou qualquer outro dado confidencial, será mantido em sigilo. A elaboração final dos dados será feita de maneira codificada, respeitando o imperativo ético da confidencialidade. Estou ciente de que posso me recusar a participar do estudo, ou retirar meu consentimento a qualquer momento, sem precisar justificar, nem sofrer qualquer dano.

Os pesquisadores envolvidos com o referido projeto são, o professor Doutor Adolfo Gustavo Serra Seca Neto, a professora Doutora Maria Cláudia Figueiredo Pereira Emer, e o mestrando do Programa de Pós-Graduação em Computação Aplicada Wilson Bissi da Universidade Tecnológica Federal do Paraná, com quem poderei manter contato pelos e-mails: adolfo@utfpr.edu.br, mcemer@utfpr.edu.br, e wbissi@gmail.com. Estão garantidas todas as informações que eu queira saber antes, durante e depois do estudo.

Li, portanto, este termo, fui orientado quanto ao teor da pesquisa acima mencionada e compreendi a natureza e o objetivo do estudo do qual fui convidado a participar. Concordo, voluntariamente em participar desta pesquisa, sabendo que não receberei nem pagarei nenhum valor econômico por minha participação.

_____

Assinatura do sujeito de pesquisa

_____

Assinatura dos pesquisadores
Wilson Bissi / Adolfo Gustavo S. S. Neto / Maria Claudia F. P. Emer

Curitiba 12 de Junho de 2015

Questionário de Pesquisa

E-mail:	Idade:
Empresa:	Cargo:

1. Qual é a sua formação acadêmica? (escolha uma).

(A) Não Graduado.

(B) Graduado.

(C) Pós Graduado - Especialização.

(D) Pós Graduado - Mestrado.

(E) Pós Graduado - Doutorado.

2. Quanto tempo de experiência você tem em desenvolvimento de software? (escolha uma).

(A) de 0 a 2 anos.

(B) de 3 a 4 anos.

(C) de 5 a 6 anos.

(D) de 7 a 8 anos.

(E) acima de 9 anos.

3. Há quanto tempo você pratica ou praticou TDD? (escolha uma).

(A) Nunca pratiquei.
(B) de 1 a 6 meses.
(C) de 7 a 12 meses.
(D) de 13 a 24 meses.
(E) acima de 25 meses.

4. O que te impede de refatorar o código fonte ao identificar oportunidades de melhoria? (escolha uma).

(A) Nada.
(B) Falta de Tempo.
(C) Falta de Cobertura de Teste.
(D) Falta de Conhecimento.
(E) Falta de Visibilidade do Impacto da Alteração.

5. Como você aprendeu TDD? (escolha uma ou mais).

(A) Trabalho.
(B) Universidade.
(C) Livro.
(D) Internet.
(E) Outro:

6. Em qual ambiente você pratica TDD? (escolha uma ou mais).

(A) Universidade.
(B) Projetos Pessoais.
(C) Trabalho.
(D) Projetos Abertos.
(E) Outro:

7. Qual o tipo de aplicação que você costuma desenvolver? (escolha uma ou mais).

(A) Desktop.
(B) Web.
(C) Mobile.
(D) Back-end.
(E) Outra:

8. Em qual linguagem você normalmente escreve testes unitários automatizados? (escolha uma).

(A) Java.
(B) PHP.
(C) C#.
(D) C++.
(E) Outra:

9. Qual a maior dificuldade em aplicar TDD na sua empresa atualmente? (escolha uma).

(A) Nenhuma.
(B) Falta de Tempo.
(C) Falta de Apoio da Gestão.
(D) Falta de Ferramentas.
(E) Falta de Conhecimento Técnico.
(F) Outra:

10. Qual desses *frameworks* de teste e *mock* você conhece? (marque com X na frente).

(A) JUnit.
(B) Selenium.
(C) MockCreator.
(D) TestNG.
(E) Mockito.
(F) Mocha.
(G) CppUnit.
(H) JustMock.
(I) SnapTest.
(J) NUnit.
(K) Moq.
(L) PHPUnit.
(M) EasyMock.
(N) RhinoMock.
(O) JMock.
(P) Outro:

Para as demais questões, de 11 até 19, a seguinte instrução é apresentada no início do questionário: "As questões a seguir devem ser respondidas selecionando a opção que mais se enquadra em sua experiência. Assinale com X a opção mais adequada. Caso desconheça ou não saiba responder, assinale a opção **0 - Não se Aplica**. As opções **1 e 2** significam que você está discordando da afirmação. A opção **3** é neutra, e as opções **4 e 5** significam que você está concordando com a afirmação". Essas questões utilizam como base a escala de Likert para as respostas dos participantes.

11. Eu normalmente utilizo serviços web (*web services*) nas soluções em que desenvolvo.

Não se aplica	Discordo Fortemente	Discordo	Indiferente	Concordo	Concordo Fortemente
0	1	2	3	4	5

12. Eu normalmente refatoro o código fonte existente.

Não se aplica	Discordo Fortemente	Discordo	Indiferente	Concordo	Concordo Fortemente
0	1	2	3	4	5

13. Eu normalmente crio testes unitários automatizados durante o desenvolvimento.

Não se aplica	Discordo Fortemente	Discordo	Indiferente	Concordo	Concordo Fortemente
0	1	2	3	4	5

14. Quando utilizo uma tecnologia como EJB ou *Web Service*, uso *mocks* para escrever o teste unitário.

Não se aplica	Discordo Fortemente	Discordo	Indiferente	Concordo	Concordo Fortemente
0	1	2	3	4	5

15. Eu normalmente preciso escrever teste unitário complexo que necessita de objetos *mock*.

Não se aplica	Discordo Fortemente	Discordo	Indiferente	Concordo	Concordo Fortemente
0	1	2	3	4	5

16. Eu não tive dificuldades em aplicar TDD na minha primeira experiência.

Não se aplica	Discordo Fortemente	Discordo	Indiferente	Concordo	Concordo Fortemente
0	1	2	3	4	5

17. A prática de TDD durante o desenvolvimento ajuda a reduzir os defeitos do software.

Não se aplica	Discordo Fortemente	Discordo	Indiferente	Concordo	Concordo Fortemente
0	1	2	3	4	5

18. A prática de TDD durante o desenvolvimento melhora a qualidade de código fonte.

Não se aplica	Discordo Fortemente	Discordo	Indiferente	Concordo	Concordo Fortemente
0	1	2	3	4	5

19. A prática de TDD durante o desenvolvimento pode aumentar a produtividade.

Não se aplica	Discordo Fortemente	Discordo	Indiferente	Concordo	Concordo Fortemente
0	1	2	3	4	5

Apêndice D

Questionário da Entrevista Presencial

Para obter e registrar a opinião dos desenvolvedores, uma entrevista individual foi realizada no final de cada experimento. Essa entrevista foi baseada em um questionário contendo 9 perguntas. Os primeiros campos para preenchimento da entrevista são referentes ao nome do profissional e a data de realização da entrevista. As questões que guiaram as entrevistas são descritas a seguir.

Perguntas da Entrevista

Nome:	Data:

1) Qual foi sua maior dificuldade técnica ao desenvolver e testar o Exercício 1 usando a abordagem TLD?
R:

2) Qual foi sua maior dificuldade técnica ao desenvolver e testar o Exercício 2 usando a abordagem WS-TDD?
R:

3) Na sua opinião, qual abordagem produz a solução mais correta em menos tempo?

(A) TLD. (B) WS-TDD. (C) Indiferente.

4) Na sua opinião, qual abordagem produz código mais simples e com mais reúso?

(A) TLD. (B) WS-TDD. (C) Indiferente.

5) Na sua opinião, qual abordagem produz a solução com menos defeito?

(A) TLD. (B) WS-TDD. (C) Indiferente.

6) Qual abordagem mais te motivou a testar? Explique.

(A) TLD. (B) WS-TDD. (C) Indiferente.

Explicação:

7) Qual abordagem simplificou o desenvolvimento e a escrita de testes automatizados? Explique.

(A) TLD. (B) WS-TDD. (C) Indiferente.

Explicação:

8) Você consideraria utilizar abordagem WS-TDD no próximo projeto com *web services*? Por que?
R:

9) Quais as vantagens e desvantagens que você percebeu ao utilizar a abordagem WS-TDD?
R:

Apêndice E

Descrição dos Exercícios Propostos no Experimento

Esta seção detalha todos os requisitos solicitados na implementação dos dois exercícios usados no experimento deste trabalho. Tanto o Exercício 1 quanto o Exercício 2 possuem o mesmo número parâmetros de entrada e saída, a mesma quantidade de regras de negócio e formas similares de validá-las.

Após a implementação de cada exercício, o desenvolvedor envia o código para um sistema de controle de versão (VCS - *Version Control System*). Depois que o código foi enviado ao VCS, os exercícios desenvolvidos localmente são apagados e um novo projeto em branco é entregue aos participantes para o desenvolvimento do próximo exercício. Dessa forma, impossibilitamos o desenvolvedor de fazer a cópia de qualquer parte do código fonte do exercício anterior.

Para auxiliar no desenvolvimento dos exercícios, dois diagramas da UML foram utilizados, são eles: o diagrama de componente e o diagrama de sequência.

E.1 Detalhamento do Exercício 1

Neste primeiro exercício os desenvolvedores implementam um serviço web usando o conceito de TLD. O exercício é referente à implementação simplificada de uma transação de cartão de crédito bancário. O nome da classe, o nome do método e as regras de negócio desenvolvidas no exercício ficaram de livre escolha do desenvolvedor.

Antes de começar o desenvolvimento, o participante recebeu uma estrutura inicial do projeto Java EE pré-configurado dentro do Eclipse IDE, cabendo ao participante se preocupar somente com o desenvolvimento do exercício de acordo com o enunciado proposto.

Enunciado do Exercício 1: “Para o Exercício 1 você deverá implementar um método WS respeitando as informações de entrada e saída, e as regras de negócio descritas a seguir. O objetivo desse método WS é receber os dados do cartão de crédito, juntamente com o valor e a quantidade de parcelas que o cliente deseja efetuar a compra. Com base nessas informações o seu método deverá fazer as validações necessárias descritas nas regras de negócio a seguir. Depois de realizar as validações preliminares, seu método deverá invocar um método do serviço web externo, da aplicação BankSystemWS denominado de BankServices.verifyCreditCardInfo. Atenção: para a implementação correta do exercício, os diagramas de componentes e de sequência devem ser seguidos”.

A Tabela E.1 apresenta os parâmetros de entrada que o método WS deve possuir.

Tabela E.1: Parâmetros de entrada do Exercício 1.

Atributo	**Tipo**	**Tamanho**	**Descrição**
NumeroCartao	String	16	Representa o número identificador do cartão de crédito.
Validade	String	5	Data limite em que o cartão de crédito é válido e pode ser aceito nas transações.
CVV	String	3	Código de segurança do cartão de crédito definido pelo sistema financeiro.
NomeTitular	String	150	Nome da pessoa responsável pelo cartão de crédito.
Valor	Double	ND*	Valor total da compra realizada.
NumParcelas	Integer	3	Número de parcelas que o cliente deseja dividir o valor total da compra.
JurosParcelado	Boolean	true/false	Indica se haverá ou não juros no parcelamento.

*ND: Não Definido.

A Tabela E.2 apresenta os parâmetros de saída que o método WS deve possuir.

Tabela E.2: Parâmetros de saída do Exercício 1.

Atributo	**Tipo**	**Tamanho**	**Descrição**
CodigoRetorno	String	3	Código da mensagem que é retornada pelo método do serviço web.
MensagemRetorno	String	150	Mensagem que é retornada pelo método do serviço web.

As regras de negócio que deverão ser implementadas neste exercício são descritas a seguir:

- Validar o tamanho dos parâmetros de entrada do método, com relação ao parâmetro NomeTitular. Este parâmetro deverá ter no mínimo 10 e no máximo 150 caracteres. Caso algum parâmetro esteja com tamanhos diferentes do previsto o código 3 deverá ser retornado juntamente com a mensagem “Dados Inválidos”. Para simplificar o exercício, apenas estaremos validando o tamanho dos campos, com exceção ao campo Validade que deve seguir a regra descrita no subitem;
  - O formato do campo Validade deve ser mês e ano com dois dígitos cada, por exemplo (04/15) para representar Abril de 2015. Caso contrário o método deverá finalizar com o código de retorno 3 e com a mensagem de “Dados Inválidos”.
- Verificar se o cliente, a partir do número do cartão, está na lista negra (*black list*), lista com clientes restritos que não devem efetuar a transação;
  - Se o cliente estiver na lista negra, o método deverá retornar que o cliente está negativado, o código de retorno deve ser 6 e mensagem será “Cliente Negativado”;

  - Se o cliente não estiver na lista negra, o método BankServices.verifyCreditCardInfo deverá ser chamado e o processamento continua.

- Validar o retorno da chamada do método do serviço web BankServices.verifyCreditCardInfo;
  - O cliente deve ser adicionado à lista negra caso o código de retorno do método BankServices.verifyCreditCardInfo seja igual a 4;
  - Se o retorno do método BankServices.verifyCreditCardInfo for igual a 1, a transação pode ser feita e o processamento do método deve continuar retornando o código 1 e a mensagem “Sucesso”;
  - Se o retorno do método BankServices.verifyCreditCardInfo for diferente de 1, a transação não pode ser feita e o retorno do método deverá conter o mesmo código e a mensagem retornada pelo método BankServices.verifyCreditCardInfo.

- Retornar o código e a mensagem do método implementado no exercício de acordo com as regras citadas anteriormente.

A Figura E.1 apresenta o diagrama de componentes do Exercício 1. A parte que está dentro do retângulo tracejado refere-se aos componentes que os profissionais precisam implementar durante a realização do exercício. A parte fora do retângulo tracejado refere-se ao serviço web externo que é invocado para validar os dados do cartão de crédito recebido pelo usuário. Portanto, os profissionais implementaram somente o componente CreditCardWS. O componente BankSystemWS já está implementado e disponível para acesso localmente.

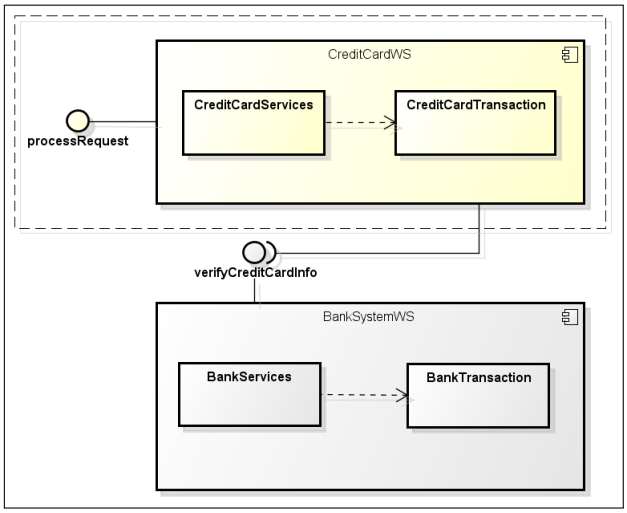

Figura E.1: Diagrama de componentes do Exercício 1. Fonte: Autoria própria.

A Figura E.2 representa, por meio do diagrama de sequência, as interações entre os métodos processRequest e createTransaction da aplicação CreditCardWS com o método verifyCreditCardInfo da aplicação externa BankSystemWS exposta pelo serviço web BankServices.

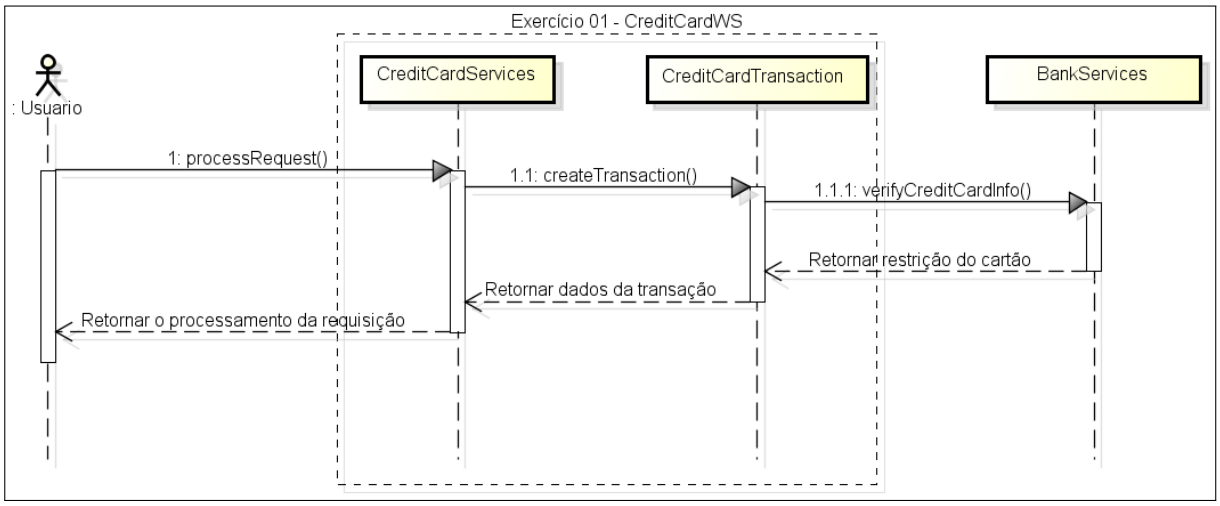

Figura E.2: Diagrama de sequência do Exercício 1. Fonte: Autoria própria.

E.2 Detalhamento do Exercício 2

Após a apresentação e a explicação da prática de TDD e da abordagem derivada de TDD com foco em WS, a WS-TDD, os participantes irão implementar o segundo exercício seguindo a abordagem WS-TDD. O exercício é referente à implementação simplificada de uma simulação de financiamento bancário. O nome da classe, o nome do método e as regras de negócio desenvolvidas no exercício ficaram de livre escolha do desenvolvedor.

Antes de começar o desenvolvimento, o participante recebeu uma estrutura inicial do projeto Java EE pré-configurado dentro do Eclipse IDE, cabendo ao participante se preocupar somente com o desenvolvimento do exercício de acordo com o enunciado proposto.

Enunciado do Exercício 2: "Para o Exercício 2 você deverá implementar um método WS respeitando as informações de entrada e saída, e as regras de negócio descritas a seguir. O objetivo desse método WS é receber os dados do cliente, juntamente com o valor e a quantidade de parcelas que o cliente deseja simular o financiamento. Com base nessas informações o seu método deverá fazer as validações necessárias descritas nas regras de negócio a seguir. Depois de realizar as validações preliminares, seu método deverá invocar um método do serviço web externo, da aplicação BankSystemWS denominado de BankServices.validateFunding. Atenção: para a implementação correta do exercício, os diagramas de componentes e de sequência devem ser seguidos".

A Tabela E.3 apresenta os parâmetros de entrada que o método WS deve possuir.

A Tabela E.4 apresenta os parâmetros de saída que o método WS deve possuir.

As regras de negócio que deverão ser implementadas neste exercício são descritas a seguir:

- Validar o tamanho dos parâmetros de entrada do método, com relação ao parâmetro Nome. Este parâmetro deverá ter no mínimo 10 e no máximo 150 caracteres. Caso algum parâmetro esteja com tamanhos diferentes do previsto o código 3 deverá ser retornado juntamente com a mensagem "Dados Inválidos". Para simplificar o exercício, apenas estaremos validando o tamanho dos atributos, com exceção ao atributo TipoJuros que deve seguir a regra descrita no subitem;

Tabela E.3: Parâmetros de entrada do Exercício 2.

Atributo	Tipo	Tamanho	Descrição
Documento	String	11	Representa o CPF (Cadastro de Pessoa Física) do cliente.
NumParcelas	String	3	Quantidade de parcelas que o financiamento irá conter.
TipoJuros	String	5	Descrição da tabela de juros que é aplicada ao financiamento.
Nome	String	150	Nome da pessoa responsável pelo financiamento.
Valor	Double	ND*	Valor total do financiamento solicitado.
RendaFamiliar	Double	ND	Renda bruta total familiar.
Correntista	Boolean	true/false	Indica se o cliente é ou não correntista do banco.

*ND: Não Definido.

Tabela E.4: Parâmetros de saída do Exercício 2.

Atributo	Tipo	Tamanho	Descrição
CodigoRetorno	String	3	Código da mensagem que é retornada pelo método do serviço web.
MensagemRetorno	String	150	Mensagem que é retornada pelo método do serviço web.

  - O atributo TipoJuros deve ser aceitar somente os valores SAC ou PRICE, caso contrário o método deverá finalizar com o código de retorno 3 e com a mensagem de "Dados Inválidos".

- Verificar se o cliente, a partir do número do documento, está na lista negra (*black list*), lista com clientes restritos que não devem efetuar a transação;
  - Se o cliente estiver na lista negra, o método deverá retornar que o cliente está negativado, o código de retorno deve ser 6 e mensagem será "Cliente Negativado";
  - Se o cliente não estiver na lista negra, o método BankServices.validateFunding deverá ser chamado e o processamento continua.
- Validar o retorno da chamada do método do serviço web BankServices.validateFunding;
  - O cliente deve ser adicionado na lista negra caso o código de retorno do método BankServices.validateFunding seja igual a 4;
  - Se o retorno do método BankServices.validateFunding for igual a 1, a simulação pode ser feita (foi aprovada) e o processamento do método deve continuar retornando o código 1 e a mensagem de "Sucesso";
  - Se o retorno do método BankServices.validateFunding for diferente de 1, a simulação não pode ser realizada (não foi aprovada) e o retorno do método deverá ser o mesmo código retornado pelo método BankServices.validateFunding.
- Retornar o código e a mensagem do método implementado no exercício de acordo com as regras citadas anteriormente.

A Figura E.3 apresenta o diagrama de componentes do Exercício 2. A parte que está dentro do retângulo tracejado refere-se aos componentes que os profissionais precisam implementar durante a realização do exercício. A parte fora do retângulo tracejado refere-se ao serviço web externo que é invocado para validar os dados do cliente na simulação de financiamento recebida pelo usuário. Portanto, os profissionais implementaram somente o componente FinancialWS. O componente BankSystemWS já está implementado e disponível para acesso localmente.

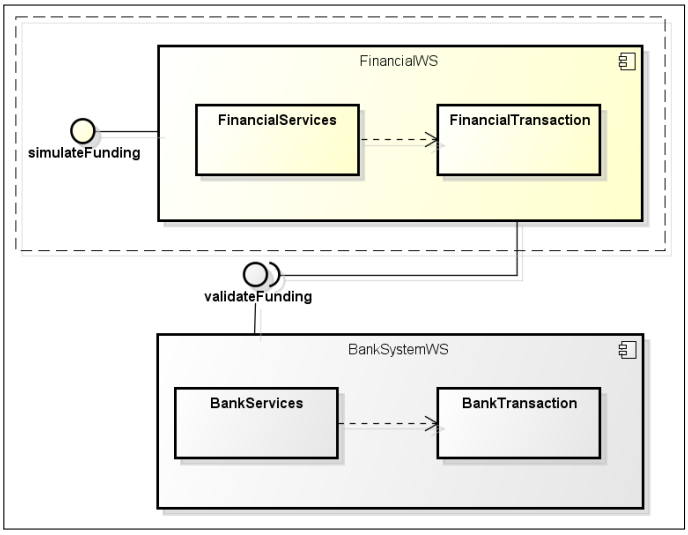

Figura E.3: Diagrama de componentes do Exercício 2. Fonte: Autoria própria.

A Figura E.4 representa, por meio do diagrama de sequência, as interações entre os métodos simulateFunding e sendSimulationToBank da aplicação FinancialWS com o método validateFunding da aplicação externa BankSystemWS exposta pelo serviço web BankServices.

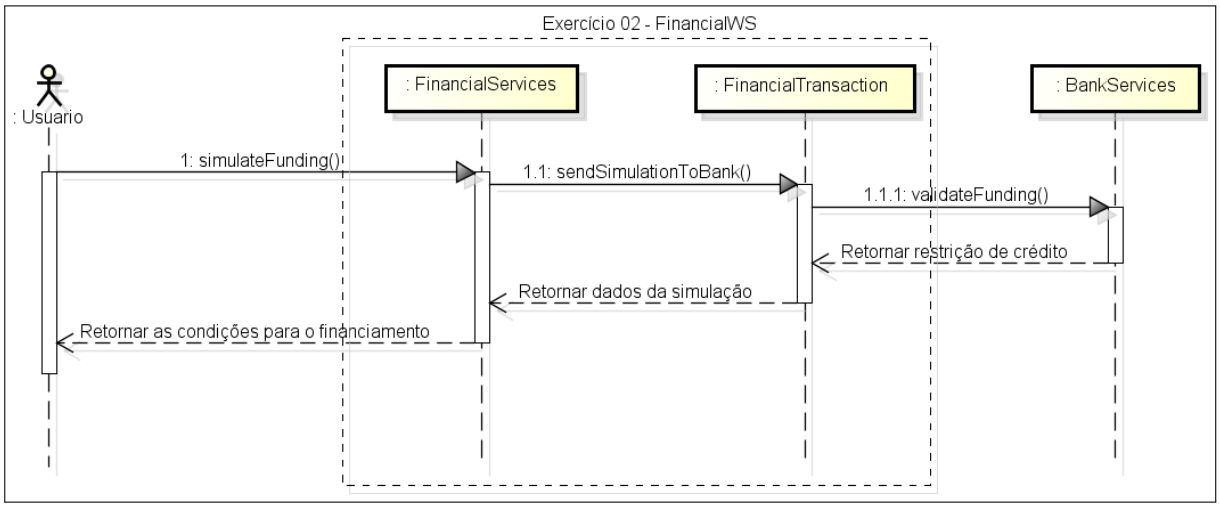

Figura E.4: Diagrama de sequência do Exercício 2. Fonte: Autoria própria.

Apêndice F

Resultados Obtidos no Questionário sobre TDD

Esta seção descreve com mais detalhes os resultados obtidos com o questionário de pesquisa descrito na Seção 3.1 e no Apêndice C.

F.1 Perfil dos Participantes

Esta seção apresenta a visão geral sobre o perfil dos participantes da pesquisa. As análises apresentadas são referentes a idade, cargo, formação acadêmica e tempo de experiência no desenvolvimento de software.

Dentre os intervalos de idade apresentados na Figura F.1, a maior concentração dos participantes ficou entre 26 a 30 anos, quase empatado com o intervalo de 31 a 35 anos. As pessoas presentes nesses dois intervalos equivalem a 75,47% da amostragem total. A média de idade dos participantes é de 30 anos.

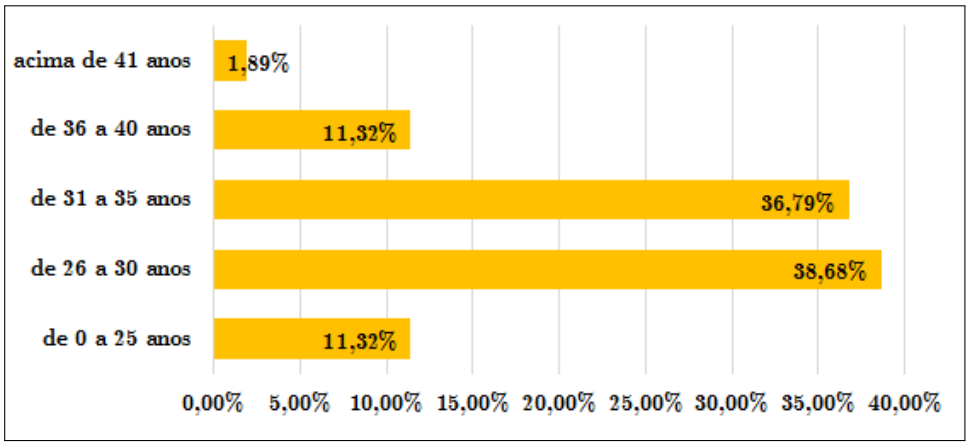

Figura F.1: Faixa de idade. Fonte: Autoria própria.

Referente ao cargo atual dos entrevistados cerca de 58,49% são desenvolvedores, 28,30% são arquitetos de software. Já os gestores, os analistas de negócio e os testadores representam 6,60%, 4,72% e 1,89% respectivamente. A Figura F.2 apresenta a distribuição dos

participantes para cada cargo. É importante ressaltar que todos os participantes tem contato com desenvolvimento de software atualmente e foram desenvolvedores em algum momento da carreira.

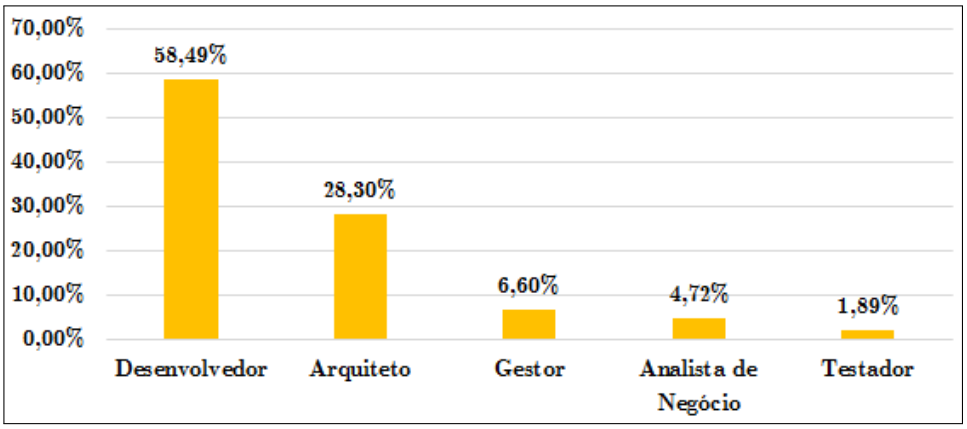

Figura F.2: Cargo atual. Fonte: Autoria própria.

Conforme apresentado na Figura F.3, apenas 6% dos participantes da pesquisa não possuem diploma de ensino superior. Os participantes que possuem somente a graduação representam 53% do total da amostra, sendo a grande maioria. Cerca de 36% deles possuem além da graduação uma especialização e 5% possuem mestrado na área de computação.

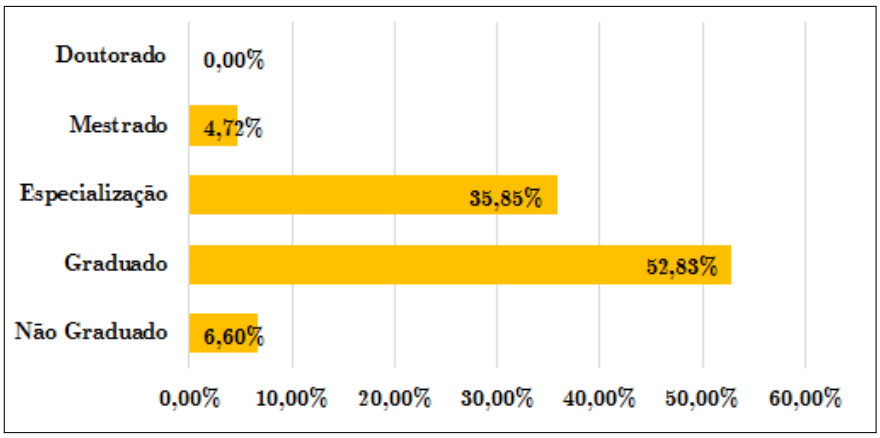

Figura F.3: Formação acadêmica. Fonte: Autoria própria.

Com relação ao tempo de experiência, o total de pessoas com mais de 9 anos de experiência é de 34,91%, no intervalo de 7 a 8 anos de experiência estão 28,30% dos participantes. Pessoas com até 2 anos de experiência representam 4,72%, na faixa de 3 a 4 anos são 12,26% e na faixa de 5 a 6 anos estão 19,81% dos participantes. Como observado na Figura F.4, cerca de 83% dos participantes possuem mais de 5 anos de experiencia, revelando que a grande maioria dos participantes são experientes na área de desenvolvimento de software.

A idade média dos participantes ficou em torno de 30 anos e cerca de 86,79% deles atuam como desenvolvedor ou arquiteto de software atualmente. Cerca de 94% dos profissionais

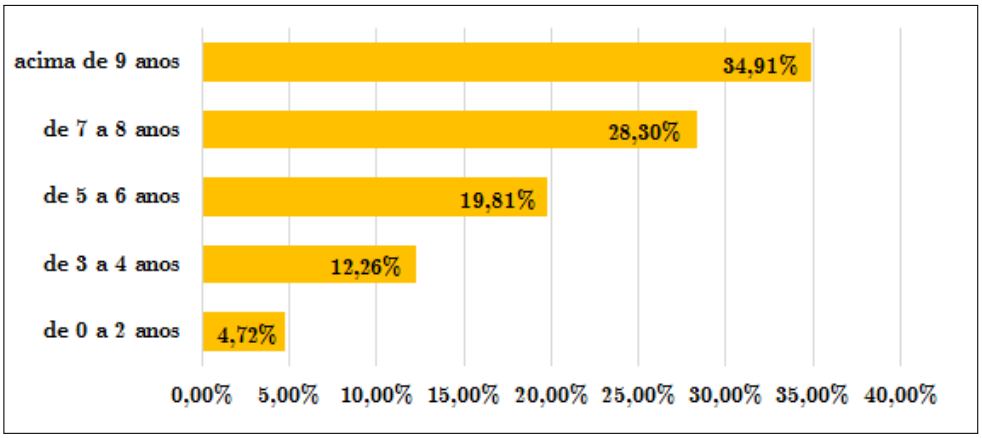

Figura F.4: Tempo de experiência. Fonte: Autoria própria.

que participaram deste questionário possuem graduação e 83% deles possuem mais de 5 anos de experiencia em desenvolvimento de software.

F.2 Tecnologias e Métodos Usados no Desenvolvimento de Software

Quando os participantes foram questionados sobre o que pode impedi-los de refatorar o código fonte existente em sistemas legados para melhorar a qualidade, diminuir a complexidade ou legibilidade a maior parte, cerca de 28,30%, informou que o que impede é a falta de visibilidade do impacto que a alteração que ele irá fazer poderá causar e por isso eles acabam deixando o código como está. A falta de tempo aparece com o maior volume das respostas, com 33,96%. A falta de cobertura de testes como motivo para não refatorar o código fonte representa 16,98%. Por fim, 18,87% afirmaram que nada os impediria de refatorar o código e apenas 1,89% informaram que é devido a falta de conhecimento do sistema. A Figura F.5 mostra a distribuição das respostas obtidas.

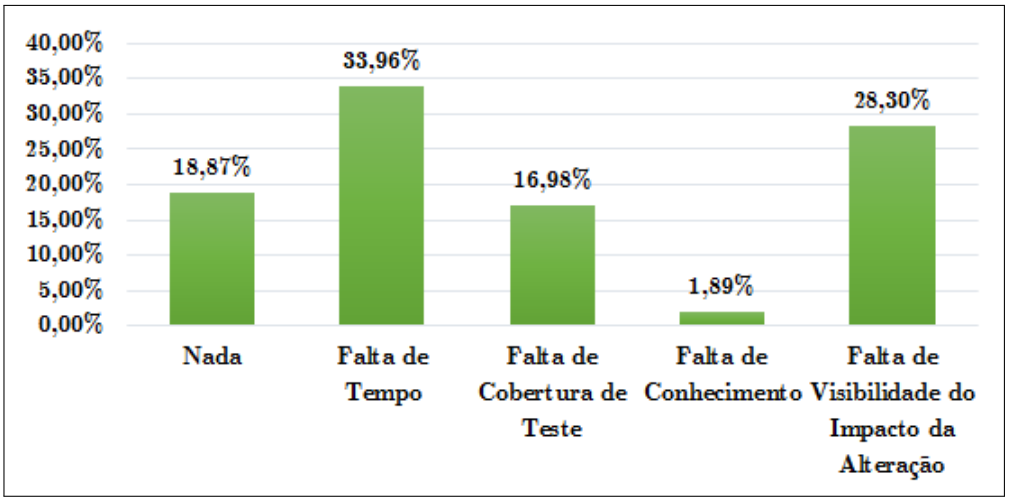

Figura F.5: Impedimentos para refatorar o código. Fonte: Autoria própria.

Quando questionamos sobre o tipo de aplicação que desenvolve, 80,19% dos participantes disseram que são aplicações Web, seguidos por aplicações back-end com 64,15%, aplicações mobile (para dispositivos móveis) com 26,42%, aplicações desktop com 22,64% e outro tipo de aplicação com 3,77%. Nesta questão é possível selecionar mais de uma resposta, portanto a soma total das opções é superior a 100%. A Figura F.6 mostra a distribuição das respostas obtidas.

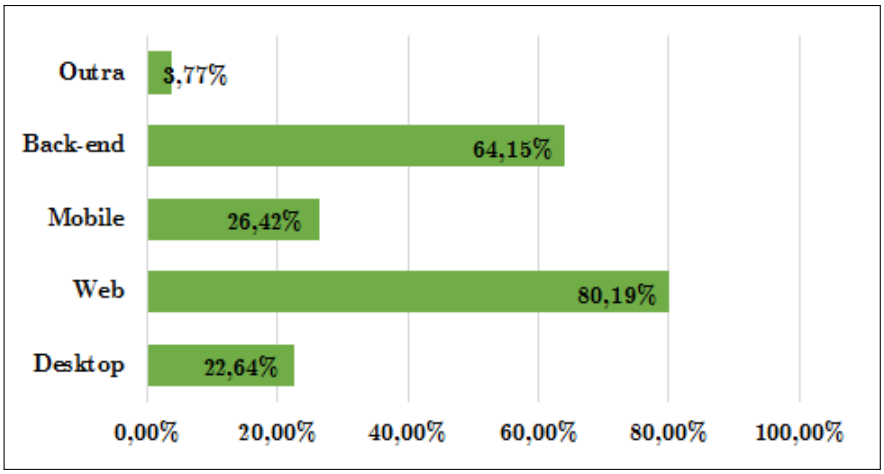

Figura F.6: Tipos de aplicações que desenvolve. Fonte: Autoria própria.

A Figura F.7 mostra que o uso da linguagem Java para a criação de testes unitários automatizados é expressivo, representado por cerca de 87,74% dos participantes. Atualmente a maioria dos profissionais utilizam Java para o desenvolvimento dos sistemas nas empresas em que a pesquisa foi aplicada. A linguagem da plataforma .NET da Microsoft, C# é a segunda com 6,60% e PHP aparece somente com 0,94%. Já 4,72% dos participantes informaram que utilizam outra linguagem para a criação dos testes, são elas: Ruby e JavaScript.

Os participantes foram questionados sobre quais os arcabouços auxiliam o desenvolvimento de teste unitário automatizado (Questão 10). O JUnit é o mais conhecido entre

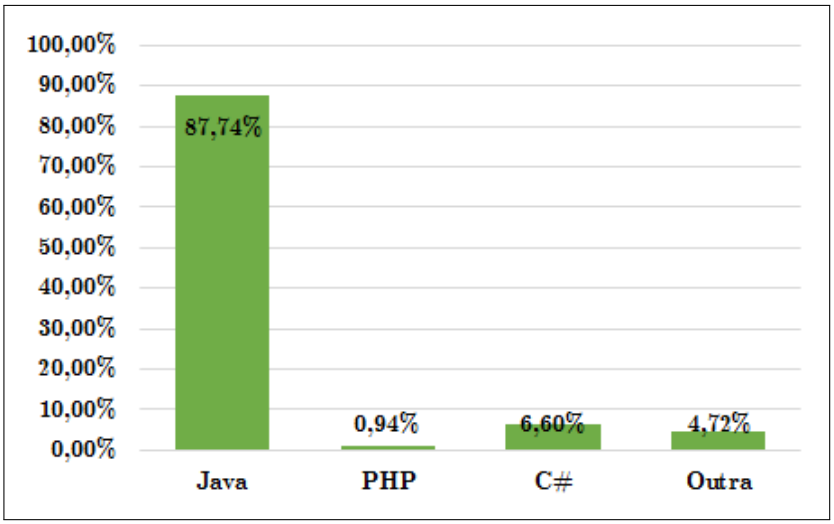

Figura F.7: Linguagem usada para os testes unitários. Fonte: Autoria própria.

os participantes com 95,28% das escolhas, em seguida o Selenium com 49,06% e o Mockito com 47,17% são também muito conhecidos. Na sequência o TestNG com 24,53%, o JMock com 21,70% e o EasyMock com 12,26% completam a lista dos 6 arcabouços mais conhecidos. Nesta questão é possível selecionar mais de uma resposta, portanto a soma total das opções é superior a 100%.

A Tabela F.1 consolida as respostas das questões 11 até 13. A coluna afirmação descreve a sentença utilizada para obter a opinião dos profissionais, as outras colunas apresentam os resultados obtidos em cada uma das opções selecionadas (NA:Não se Aplica, DF:Discordo Fortemente, D:Discordo, I:Indiferente, C:Concordo, CF:Concordo Fortemente).

Tabela F.1: Opiniões sobre as tecnologias e métodos de mercado.

Afirmação	**NA**	**DF**	**D**	**I**	**C**	**CF**
11-Eu normalmente utilizo serviços web nas soluções em que desenvolvo.	2,83%	0%	0%	7,55%	50,00%	39,62%
12-Eu normalmente refatoro o código fonte existente.	2,83%	0,94%	6,60%	26,42%	54,72%	8,49%
13-Eu normalmente crio testes unitários automatizados durante o desenvolvimento.	8,49%	6,60%	24,53%	28,30%	21,70%	10,38%

*NA:Não se Aplica, DF:Discordo Fortemente, D:Discordo, I:Indiferente, C:Concordo, CF:Concordo Fortemente.

Quando perguntado sobre o uso de serviços web na elaboração de uma solução de software, cerca a de 39,62% disseram que concordam fortemente e outros 50% concordam com essa afirmação. Apenas 7,55% indicaram como indiferente essa afirmação e 2,83% não opinaram. Dessa forma, de acordo com 89,62% dos participantes o uso de serviços web são necessários para a elaboração da solução.

A afirmação "Eu normalmente refatoro o código fonte existente", visa identificar se os participantes da pesquisa costumam refatorar o código fonte existente durante uma alteração

(correção ou nova funcionalidade) em um sistema existente. Cerca de 54,72% concordaram e 8,49% concordaram fortemente que ao fazer alguma modificação no sistema eles aproveitam para refatorar parte do código existente. Já 26,42% disseram indiferente, 6,60% discordaram, 0,94% disseram discordar fortemente, ou seja, 7,54% dos participantes dificilmente refatora o código fonte de um sistema, mesmo percebendo alguma melhoria. Outros 2,83% dos participantes não opinaram.

Já sobre a criação de teste unitário automatizado durante o desenvolvimento de software, 21,70% disseram concordar e 10,38% disseram concordar fortemente. Outros 28,30% afirmaram que é indiferente, 24,53% disseram discordar, 6,60% disseram discordar fortemente e 8,49% preferiram não opinar. O total de pessoas que afirmaram que criam testes unitários automatizados durante o desenvolvimento representa 32,08%.

Apêndice G

Questionário da Entrevista Presencial

Este apêndice apresenta um exemplo de resposta ao questionário de entrevista, que foi respondido por 23 participantes.

Perguntas da Entrevista

1) Qual foi sua maior dificuldade técnica ao desenvolver e testar o Exercício 1 usando a abordagem TLD?
R: Encontrar erros somente durante a execução dos testes.

2) Qual foi sua maior dificuldade técnica ao desenvolver e testar o Exercício 2 usando a abordagem WS-TDD?
R: No primeiro momento, tive dificuldade em escrever os testes antes do código esta pronto.

3) Na sua opinião, qual abordagem produz a solução mais correta em menos tempo?
R: WS-TDD, trabalhando dessa forma consegui encontrar erros e corrigi-los com mais agilidade e refatorar o código para ficar mais legível e de fácil manutenção.

4) Na sua opinião, qual abordagem produz código mais simples e com mais reúso?
R: WS-TDD.

5) Na sua opinião, qual abordagem produz a solução com menos defeito?
R: WS-TDD.

6) Qual abordagem mais te motivou a testar? Explique.
R: WS-TDD. Com o desenvolvimento e a criação dos testes trabalhando juntos, conseguimos fazer um código mais assertivo.

7) Qual abordagem simplificou o desenvolvimento e a escrita de testes automatizados? Explique.
R: WS-TDD. Durante a escrita dos testes é possível saber como será implementado o código, antecipando os erros e corrigindo com mais facilidade.

8) Você consideraria utilizar abordagem WS-TDD no próximo projeto com *web services*? Por que?

R: Sim, porque proporciona o aumento da produtividade, melhor desempenho no desenvolvimento/teste, qualidade do código.

9) Quais as vantagens e desvantagens que você percebeu ao utilizar a abordagem WS-TDD?
R: Vantagem: Assertividade, qualidade, desempenho no desenvolvimento. Desvantagem: No primeiro contato, é um pouco confuso criar os testes sem ter o código implementado, mas após o primeiro teste criado utilizando TDD fica mais fácil e produtivo.