AUGUSTO CÉSAR CARDOZO DE OLIVEIRA

UMA PROPOSTA DE ADAPTAÇÃO DO ACCEPTANCE TDD

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para a conclusão do curso de MBA em Tecnologia de Software.

Área de Concentração: Tecnologia de Software

Orientador: Prof. Dr. Kechi Hirama

São Paulo
2012

FICHA CATALOGRÁFICA

Oliveira, Augusto César Cardozo de
Uma proposta de adaptação do Acceptance TDD / A.C.C. de Oliveira. -- São Paulo, 2012.
113 p.

Monografia (MBA em Tecnologia de Software) – Escola Politécnica da Universidade de São Paulo. Programa de Educação Continuada em Engenharia.

1. Desenvolvimento de software I. Universidade de São Paulo. Escola Politécnica. Programa de Educação Continuada em Engenharia II. t.

DEDICATÓRIA

Dedico este trabalho aos meus pais João Cardozo (in memoriam) e Rita Cristina pelo apoio incondicional, dedicação e perseverança na minha qualificação.

AGRADECIMENTOS

À Deus por iluminar os meus caminhos e fortalecer-me nas adversidades.

Ao PECE, Programa de Educação Continuada em Engenharia, e aos professores do curso de MBA em Tecnologia de Software 2010-2011 pela oportunidade em compartilhar experiências e conhecimentos que contribuíram para a minha formação acadêmica e também pessoal.

Ao Professor Doutor Kechi Hirama pela sua confiança, dedicação, orientação e incentivo, fundamentais para a conclusão deste trabalho.

Aos meus pais pela sua doação, amor e confiança.

À minha namorada, Majory Melo, pelo apoio, compreensão e paciência nos momentos de ausência em função deste trabalho.

RESUMO

Recentemente, a qualidade de software é considerada a principal característica competitiva entre as organizações. A baixa qualidade do produto de software pode oferecer impactos negativos aos negócios, além de comprometer a credibilidade de uma organização. Portanto, é muito importante planejar e conduzir as atividades de testes de maneira sistemática.

Este trabalho tem como objetivo caracterizar a atividade de teste de software, em particular os testes de aceitação, aplicada dentro do contexto dos métodos ágeis de desenvolvimento de software, abordando a técnica do *Acceptance Test Driven Development* (ATDD) para assegurar a clareza das necessidades de negócio do cliente, garantir a implementação adequada do produto de software e a satisfação do cliente.

Este trabalho propõe uma adaptação do ATDD prezando pela execução manual dos testes de aceitação, e apresenta também diretrizes para o uso desta proposta.

ABSTRACT

Currently, the software quality is considered the main competitive feature among the organizations. Low software product quality can offer negative impacts to business, also undermine the credibility of an organization. Therefore, it is very important to plan and conduct testing activities in a systematic way.

This work aimed to characterize the activity of software testing, in particular the acceptance tests, applied within the context of agile software development, addressing the technique of Acceptance Test Driven Development (ATDD), to ensure the clarity of the business needs of the client, ensure the proper implementation of the software product and customer satisfaction.

This work proposes an adaptation of ATDD valuing manual execution of acceptance tests, and also presents guidelines for the use of this proposal.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

A qualidade de software tornou-se um fator de competitividade empresarial em função da grande demanda por produtos de software e o elevado nível de exigência dos clientes que almejam softwares confiáveis e eficientes com baixo custo e prazo.

A atividade de teste de software contribui para a melhoria da qualidade do software, revelando a presença de defeitos no software, minimizando os riscos para o negócio e garantindo a aderência às necessidades do cliente.

Este trabalho explora os testes de aceitação no contexto dos métodos ágeis, abordando o *Acceptance Test Driven Development* (Desenvolvimento Dirigido a Testes de Aceitação) como um instrumento de comunicação colaborativa que visa assegurar a clareza dos requisitos de negócio, por conseqüência, o adequado desenvolvimento do produto de software e a satisfação do cliente.

1.1 Motivações

O uso de softwares já faz parte do cotidiano de praticamente todas as pessoas. Muitos dos produtos e serviços incorporam de alguma forma, computadores e softwares com finalidades diversas. Nesse contexto, as empresas inseridas no mercado globalizado, que é dinâmico e competitivo, utilizam de produtos de software como vantagem competitiva para apoiar suas operações e negócios. Tornando-se constantes as demandas para inovar, aumentar a produtividade e a qualidade, e disponibilizar para os clientes produtos e serviços com menor custo e maior agilidade.

Embora a Tecnologia da Informação seja uma aliada na busca destes objetivos, os projetos de software são, cada vez mais, vistos nas empresas não como investimentos, mas sim, como custos que devem ser controlados. As equipes de desenvolvimento são constantemente pressionadas para reduzir custos, prazo e produzir produtos que assegure valor ao cliente. Para

acompanhar esse dinamismo faz-se necessário lidar de forma mais eficiente com as mudanças de requisitos, melhorando a comunicação com o cliente e promovendo maior sinergia da equipe do projeto.

Sob esta ótica, a indústria de software tem exigido que a Engenharia de Software desenvolva a capacidade de se adaptar a mudanças. Historicamente, as abordagens de desenvolvimento tradicionais, orientadas a planos, com nível mais rígido de atividades, exigem densa documentação, dificultando que os requisitos dos softwares sejam mutáveis. Além disso, o pouco envolvimento do cliente, que retém o conhecimento do negócio, ou até mesmo a sua ausência durante o ciclo de vida do software fez com que essa abordagem de desenvolvimento de software se mostrasse inapropriada à atual realidade.

Como um contraponto a burocracia dos processos tradicionais da Engenharia de Software, diversos métodos ágeis de desenvolvimento estão sendo usados em projetos de software. O enfoque dessa abordagem está direcionado às pessoas. Assim, a abordagem ágil de desenvolvimento de software, cujo foco é a satisfação do cliente, preza pela simplicidade, entrega imediata de funcionalidade de valor ao cliente, documentação mínima necessária e predisposição às mudanças (ABRAHAMSSON et al., 2002. p.11-12.).

Considerando que algumas das razões pelas quais projetos de software tendem ao insucesso é a ausência de objetivos claros de negócio, o que remete a identificação de defeitos tardiamente no ciclo de desenvolvimento, e o pouco envolvimento do cliente no projeto, esse que por sua vez é responsável em fornecer informações íntegras e concisas para subsidiar o desenvolvimento do produto de software, essa situação controversa revela uma deficiência no uso de valores como comunicação e *feedback* praticados nos métodos ágeis, o que justifica algumas das razões para esse insucesso.

Com o cliente inserido no processo de desenvolvimento de software e a sua satisfação como norteador da qualidade, a atividade de teste no contexto dos métodos ágeis possui um papel relevante para assegurar a aderência das necessidades e expectativas do cliente, minimizar a probabilidade da ocorrência de defeitos no software e os seus riscos para o negócio.

Diferentemente das abordagens tradicionais de desenvolvimento onde os testes ocorrem tardiamente, os testes em métodos ágeis devem ocorrer freqüentemente, procurando identificar os defeitos desde as fases iniciais do desenvolvimento, com o apoio permanente do cliente através de seus *feedbacks*.

Entre as práticas de testes adotadas na abordagem ágil de desenvolvimento de software destacam-se o *Test Driven Development* (TDD) e o *Acceptance* TDD, ambas utilizando-se da automação de testes.

O TDD é voltado aos desenvolvedores, aborda o nível de teste de unidade, e a integração destas unidades, seu foco está na legibilidade do código-fonte, prezando por um bom projeto (*design*) que promova um código limpo e funcional (BECK, 2003. p.11.).

Em contrapartida, o *Acceptance* TDD, conhecido também como ATDD (Desenvolvimento Dirigido a Testes de Aceitação) aborda o nível de teste de aceitação, sob a ótica do cliente, e visa construir um produto de software aderente aos requisitos de negócio, promovendo a satisfação do cliente perante a funcionalidade desenvolvida, independente da legibilidade e boa estrutura do código-fonte (KOSKELA, 2007. p.324.).

Contudo, a automação dos testes de aceitação pode exigir dos desenvolvedores um amplo esforço na elaboração e manutenção dos testes, sobrecarregando-os, pois tornam-se responsáveis por conceber e manter o código da aplicação, o código dos testes de unidade e aceitação. Além disso, os testes de aceitação automatizados são restritivos à algumas interfaces gráficas, o que pode inviabilizar a automação desses testes, fazendo-se necessário o uso da execução manual dos testes de aceitação.

Sob o pano de fundo dos métodos ágeis, o ATDD é usado como um instrumento de comunicação colaborativa no processo de desenvolvimento de software para assegurar a clareza das necessidades de negócio do cliente, por consequência, reduzir o número de defeitos detectados durante o ciclo de vida do software em função de requisitos incompletos e ambíguos, portanto, garantir

a implementação adequada do produto de software e minimizar os custos de desenvolvimento e manutenção do software.

1.2 Objetivo

O objetivo deste trabalho é propor uma adaptação de um processo de teste de aceitação de software baseado na abordagem do ATDD (*Acceptance Test Driven Development*), mantendo a aderência do produto de software com as necessidades de negócio do cliente.

Essa proposta aborda o ciclo de vida do teste de aceitação, prezando pela execução manual dos testes de aceitação e fornece um instrumento de comunicação colaborativa no projeto de desenvolvimento de software.

1.3 Justificativas

Em suas obras, Koskela (2007) e Pugh (2011) apresentam as práticas e o processo do ATDD no desenvolvimento de software. Ambos os trabalhos tem suas similaridades, no entanto, como diz o provérbio chinês "Existem muitos caminhos para o topo da montanha, mas a vista é sempre a mesma". E muitos dos caminhos compartilham a mesma trilha por parte da viagem (PUGH, 2011. p.5.).

O ATDD é uma prática recente, e há poucos estudos que apresentam os detalhes que envolvem o ciclo de vida do teste de aceitação. A maioria dos trabalhos concentram-se em evidenciar os resultados de experimentos acadêmicos, avaliando a eficácia do uso de ferramentas automatizadas de testes de aceitação sob a perspectiva dos requisitos. Alguns exemplos desses trabalhos são:

- Melnik, Read e Maurer (2004. p.2.) propuseram um experimento em dois ambientes acadêmicos distintos, sob condições específicas, para avaliar o uso da ferramenta FIT (*Framework for Integrated Tests*) como uma especificação de requisitos funcional para apoiar os desenvolvedores, e

realizaram uma aferição para verificar se o uso desta ferramenta ajuda a mitigar as falhas provenientes de requisitos inconsistentes;

- Read, Melnik e Maurer (2005. p.1.) realizaram uma nova experiência introduzindo o uso da ferramenta FIT em um curso de Engenharia de Software. A premissa foi que os estudantes não possuíssem conhecimento algum da ferramenta FIT. Logo, este estudo avaliou a habilidade de aprendizagem do uso da ferramenta durante os projetos acadêmicos. A finalidade do estudo concentrou-se em educar os estudantes; investigar suas percepções e opiniões sobre os testes de aceitação e a ferramenta FIT; E por fim, determinar se a ferramenta é eficaz como documento de requisitos;

- Ricca, et al. (2007, p.280.) propuseram experimentos controlados para avaliar a adoção da ferramenta FIT como suporte à compreensão de requisitos. Este estudo tem similaridades com o trabalho proposto por Melnik, Read e Maurer (2004). No entanto, há algumas diferenças, dentre elas, a mais relevante é o uso da ferramenta FIT para melhor entendimento dos requisitos, portanto, não há execução automatizada dos testes;

- Sauvé, Neto e Neto (2007. p.82.) avaliaram o uso da ferramenta *EasyAccept* no projeto *Smart Action*, desenvolvido pela Universidade Federal de Campina Grande (UFCG da Paraíba), para a companhia de energia CHESF. O projeto consumiu mais de dezoito meses e o relato dos benefícios e problemas deparados com o uso da ferramenta *EasyAccept* neste período foram relatados neste trabalho;

- Hanssen e Haugset (2009. p.4.) realizaram um estudo de caso na indústria para avaliar qualitativamente os esforços e benefícios no uso da ferramenta FIT. Este estudo foi realizado sob a perspectiva de um único projeto e o método de análise adotado foi o uso de entrevista estruturada com questões abertas à quatro desenvolvedores. O escopo

dessa entrevista teve seu foco em três aspectos do FIT: Sobre o Uso; os Efeitos e Resultados; e por fim, a Experiência.

O uso de ferramentas de automação de testes é importante, principalmente, em projeto de desenvolvimento ágil que preza por ciclos pequenos de desenvolvimento e *feedbacks* constantes. No entanto, os estudos citados não apresentam as etapas e atividades que permeiam os testes de aceitação, desde sua concepção, execução e encerramento. Então, sob esta perspectiva, o ATDD é explorado no presente trabalho cujo propósito é propor uma adaptação do ATDD, abordando a execução manual dos testes de aceitação, e algumas diretrizes para o uso do ATDD proposto.

Procura-se com este trabalho propor um conjunto de atividades estruturadas para os testes de aceitação baseando-se nas práticas do ATDD. Além disso, pretende-se encorajar os profissionais da Engenharia de Software ao uso dessa abordagem expondo os aspectos inerentes ao uso desta proposta no projeto de software.

1.4 Estrutura do Trabalho

O trabalho está organizado em seis capítulos cuja breve descrição dos respectivos é apresentada a seguir.

Capítulo 1 - INTRODUÇÃO

Nesse capítulo é apresentada as motivações, o objetivo, as justificativas e a estrutura do trabalho.

Capítulo 2 – TEST DRIVEN DEVELOPMENT - TDD

Nesse capítulo são apresentadas as características da técnica do *Test Driven Development* (TDD) e os resultados do seu uso no projeto de software.

Capítulo 3 - ACCEPTANCE TDD - ATDD

Nesse capítulo são apresentadas as práticas associadas a atividade de testes de aceitação no contexto dos métodos ágeis, abordando o *Acceptance* TDD (ATDD). As características do ATDD, assim como os resultados do seu uso no projeto de software também são descritos nesse capítulo.

Capítulo 4 - ADAPTAÇÃO DO ATDD

Nesse capítulo é apresentada uma proposta de adaptação do ATDD, descrevendo as atividades e os produtos de engenharia de software que permeiam as etapas desta proposta.

Capítulo 5 - DIRETRIZES PARA O USO DO ATDD PROPOSTO

Nesse capítulo são apresentadas algumas diretrizes para o uso do ATDD proposto, assim como são inferidos os principais resultados da sua aplicação em um projeto de desenvolvimento de software.

Capítulo 6 - CONSIDERAÇÕES FINAIS

Esse capítulo conclui este trabalho, destacando as contribuições para a área de teste de software no contexto dos métodos ágeis e, por fim, a descrição de possíveis trabalhos futuros.

REFERÊNCIAS

Relacionam as referências bibliográficas usadas como fundamentação às afirmações do presente trabalho.

2. TEST DRIVEN DEVELOPMENT - TDD

A atividade de teste vem crescendo em importância ao longo dos anos e isto está diretamente relacionado à necessidade de produzir produtos de software que atendam as exigências dos clientes que são cada vez maiores.

Em projetos ágeis o desenvolvimento do software é realizado em ciclos curtos e iterativos e devido a isso os testes são executados frequentemente durante todo o ciclo de vida do software.

Os métodos ágeis enfatizam os níveis de teste de unidade, cujo esforço concentra-se em garantir a aderência dos menores componentes do código às especificações funcionais e arquiteturais, e teste de aceitação, cujo objetivo é assegurar a aderência do software aos requisitos de negócio.

Além de promover a detecção de defeitos no software desde as fases iniciais do projeto, os testes no contexto ágil possibilitam que os desenvolvedores analisem com mais critério o projeto do software, isto porque os testes são escritos, impreterivelmente, antes do código da aplicação. Então, os desenvolvedores são capazes de prover soluções mais robustas. Além disso, os testes são utilizados como um instrumento de comunicação e *feedback* que permite nortear a qualidade do desenvolvimento do software.

Neste capítulo são apresentadas as características do TDD e a atuação desta técnica no projeto de desenvolvimento de software.

2.1 TDD

Concebido por Kent Beck, o TDD é uma pratica do *eXtreme Programming* (XP) e está associado aos métodos ágeis de desenvolvimento de software. O TDD promove o desenvolvimento incremental do código que deve passar pelos testes previamente escritos (BECK, 2003. p.7).

Escrever os testes antes do código faz com que os desenvolvedores avaliem previamente os diversos aspectos da funcionalidade e as

circunstâncias à serem consideradas, definindo o escopo e a decisão arquitetural do software (KOSKELA, 2007. p.16.).

O TDD aborda os testes de unidade e a integração destas unidades, assegurando a implementação de uma funcionalidade ou estória de usuário em conformidade aos requisitos do sistema. Além disso, o TDD preza pelo uso de ferramentas de automação de testes para garantir a capacidade e cobertura dos testes, por conseqüência, a integridade do software ao acréscimo de funções e o dinamismo das mudanças (ASTELS, 2003. p.6.), promovendo a confiança dos desenvolvedores (BECK, 2003. p.197.).

O TDD e testes de unidade são conceitos inter-relacionados, porém com escopos e objetivos diferentes. Realizar os testes de unidade por si só não significa praticar o TDD. Essa distinção é importante visto que os principais benefícios de usar TDD estão relacionados à entender como essa técnica funciona e aplicá-la corretamente.

Na seção seguinte são descritas as etapas que compreendem o TDD, as práticas e os resultados do seu uso no projeto de desenvolvimento de software.

2.2 O ciclo de TDD

O TDD prevê as seguintes etapas do desenvolvimento de software: Teste, Código e Projeto. A ordem das etapas é inversa da abordagem tradicional de desenvolvimento de software. No TDD a primeira etapa é escrever o teste, então o código e, por fim, o projeto. Esta última etapa tem um significado diferente no contexto do TDD, onde ao final o código é reescrito, aprimorando o projeto inicial, conhecido como refatoração (KOSKELA, 2007. p.15.).

O ciclo de TDD, também é conhecido como *Red-Green-Refactor*, e segundo Beck (2003. p.x-prefácio.) este é o mantra do TDD. A alusão às cores dá-se em função das ferramentas de testes que sinalizam a evolução das etapas do TDD de acordo com as cores vermelho (*Red*), que significa que o teste falhou, e verde (*Green*), que o teste foi bem sucedido.

A Figura 1 ilustra o ciclo do TDD (JEFFRIES e MELNIK, 2007. p.25).

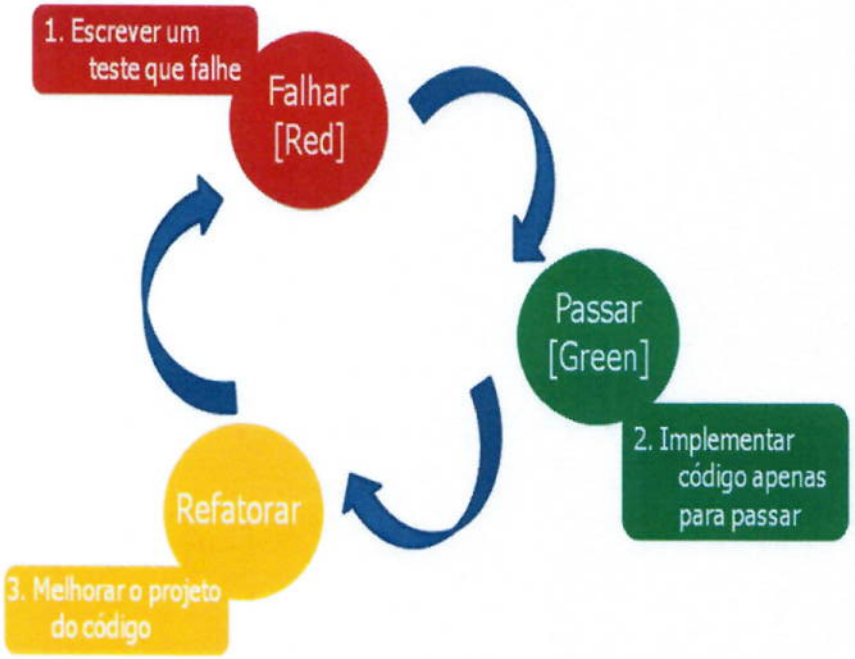

Figura 1 - Ciclo do TDD (adaptado de JEFFRIES e MELNIK, 2007. p.25).

Uma breve descrição das etapas que compreendem o ciclo de TDD é apresentada a seguir.

2.2.1 Escrever o teste

O ciclo de TDD inicia-se com a escrita de um teste e este deve falhar (*Red*), pois ainda não há uma funcionalidade presente.

Segundo BECK (2003. p.204.), o TDD é uma ferramenta de projeto cuja motivação é encorajar os desenvolvedores a refletir e planejar melhor a solução, privilegiando a simplicidade do desenvolvimento.

Iniciar com um teste falho obriga o desenvolvedor a justificar cada linha de código. Então, o teste torna-se o termômetro do projeto e de acordo com o seu tamanho e dificuldade determina-se a complexidade da implementação.

Nesta primeira etapa do ciclo de TDD o teste é escrito para uma pequena funcionalidade à ser desenvolvida, implementando um caso de teste especifico de como a aplicação deve invocar esta funcionalidade e qual o resultado

esperado. Logo, cada funcionalidade é desdobrada em comportamentos que são necessários para cumprir os requisitos do sistema. Em seguida, para cada comportamento escreve-se um teste de unidade automatizado.

Uma boa prática é escrever os testes de uma forma independente, ou seja, testes que não dependam de outros testes para passarem. Evitando situações onde um teste falho implique na falha de outros testes. Ou ainda, um caso mais sutil e raro, a execução de um teste fazer um teste subseqüente ser bem sucedido (BECK, 2003. p. 97-98).

Agrupar os testes através de algum critério também é uma boa prática, pois permite que determinadas partes do sistema sejam testadas sem que seja preciso executar todos os testes. Isto pode ser útil na execução dos testes de regressão após alterações no código (ASTELS, 2003. p.43.).

2.2.2 Escrever o código

Após escrever o teste que falhou, deve-se escrever o código que atenda aos requisitos do sistema. Nesta etapa do ciclo de TDD deve-se escrever apenas o código suficiente para satisfazer o teste previamente escrito.

Uma boa prática é implementar o código pouco a pouco (*baby steps*), obtendo o *feedback* dos testes das pequenas frações do código e assim evitando criar códigos complexos desnecessariamente. Também é importante a prática de simular o código antes de sua construção, permitindo um melhor planejamento pelo desenvolvedor (BECK, 2003. p.13.).

O TDD prega por desenvolver somente o que é realmente necessário no momento, prezando pela simplicidade. Uma vez que o código implementado está plenamente satisfatório (*Green*), ou seja, o código foi bem sucedido em todos os testes de unidade, a refatoração deve melhorar o código e encontrar uma arquitetura robusta e satisfatória.

2.2.3 Refatorar o código

Esta etapa é responsável por tornar o TDD sustentável. Neste momento melhorias são realizadas no código-fonte, retirando o código mal estruturado e implementando melhores arquiteturas.

Em função dos pequenos ciclos iterativos é previsível que em pouco tempo o código estará desbalanceado, difícil de entender e manter. Neste contexto a refatoração tem relevância no TDD, pois é uma técnica disciplinada de reestruturação do código existente, alterando sua estrutura interna sem alterar o comportamento externo (FOWLER, 1999. p.9.).

A refatoração é utilizada no TDD como um instrumento para tornar o código limpo e legível. Portanto, é recomendável aplicá-la tanto no código da aplicação como no código de teste (ASTELS, 2003. p.43.).

Para garantir o rápido *feedback* das mudanças que ocorrem no sistema, o uso de ferramentas de testes automatizados faz-se necessário para a adoção do TDD, fornecendo uma rede de proteção contra a introdução inadvertida de erros. Além disso, as ferramentas de testes apóiam os testes regressivos de maneira eficiente (KOSKELA, 2007. p.30.).

2.3 Fluxograma do TDD

O ciclo de TDD é apresentado na literatura sucintamente como três etapas: Testar, Códificar e Projetar (Refatorar). Esta é uma maneira simplista de sumarizar esta técnica. Portanto, para possibilitar uma melhor visualização de como estas etapas se relacionam, a Figura 2 ilustra o fluxograma do TDD.

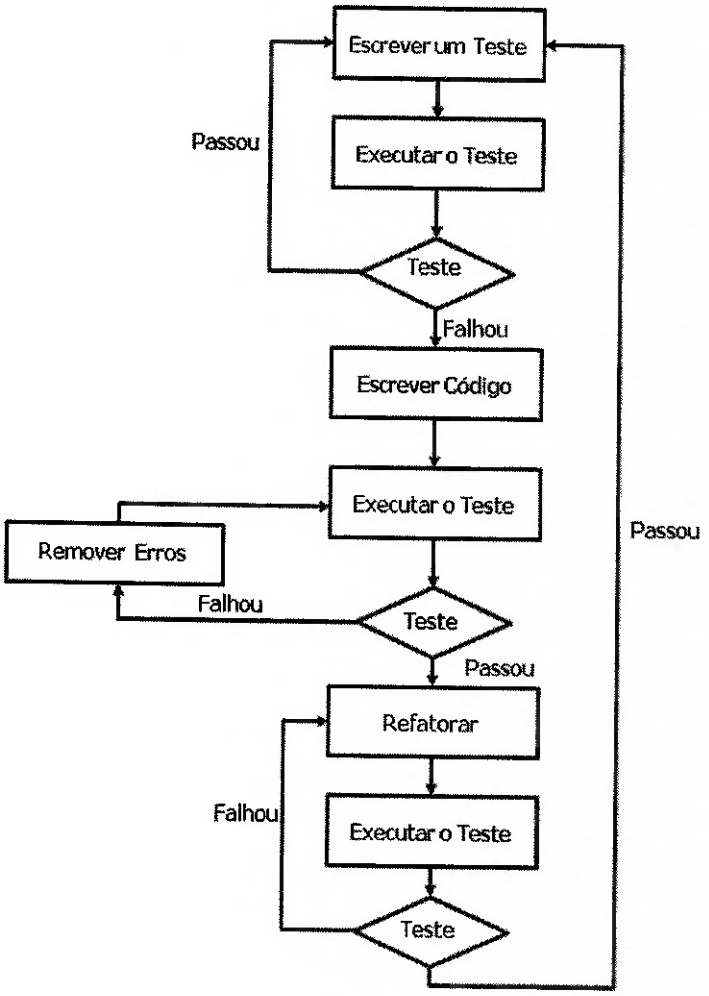

Figura 2 – Fluxograma do TDD (SINIAALTO, 2006. p.6.).

Segue uma breve descrição das etapas fundamentais do TDD, de acordo com o fluxograma ilustrado na Figura 2:

a) Escrever um teste para uma pequena funcionalidade;
b) Executar o teste para vê-lo falhar;
c) Escrever um código para a funcionalidade;
d) Executar o teste para vê-lo passar;
e) Refatorar o código (aplicação e teste); e
f) Executar todos os testes para validar se a refatoração não alterou o comportamento externo.

Na seção seguinte são descritos os resultados do TDD no projeto de desenvolvimento de software.

2.4 Resultados do TDD no projeto de software

O uso adequado do TDD pode melhorar a qualidade do código, reduzir a densidade de defeitos e prover uma melhor manutenibilidade do software (ANICHE e GEROSA, 2010. p.469.).

Aplicar os preceitos do TDD com disciplina aumenta a confiança dos desenvolvedores para com os impactos causados pelas alterações no software. Além disso, permite tornar o código limpo, flexível e documentá-lo de forma não ambígua através dos testes (MARTIN, 2007. p.35-36.).

Embora o uso do TDD possibilite esses resultados positivos a sua adoção em projetos ágeis ainda tem pouca representatividade de acordo com pesquisa divulgada recentemente (VERSION ONE, 2011. p.4.). Uma das razões para esse cenário é a dificuldade de aplicar estritamente os preceitos do TDD, assim equívocos como não visualizar o teste falhar, não refatorar o código da aplicação e o código de teste, e ignorar o princípio da simplicidade comprometem a aderência da técnica (ANICHE e GEROSA, 2010. p.473.). Além disso, a inexperiência dos desenvolvedores com os testes automatizados deve ser considerada como um fator de risco na adoção do TDD (PULEIO, 2006. p.2.).

2.5 Considerações do Capítulo

O TDD fomenta o desenvolvimento sustentável do software, encorajando os desenvolvedores a um bom projeto e maximiza as chances de que o código seja implementado de maneira correta. Contudo, esta técnica deve ser praticada com disciplina pelos desenvolvedores prezando pelo uso estrito dos preceitos do TDD de modo que sobressaiam os resultados positivos do seu uso no projeto de desenvolvimento do software.

A técnica do TDD promove valor, principalmente, aos desenvolvedores do software, pois assegura a boa legibilidade do código e provê mecanismos que aumentam a confiança dos desenvolvedores para com a qualidade interna do software, através dos testes regressivos automatizados.

No entanto, os clientes, raramente, estão interessados em adquirir código, mas sim, software que suporte seus negócios, promovendo produtividade e eficiência operacional.

Sob esta ótica os métodos ágeis enfatizam os testes de aceitação, abordando o *Acceptance* TDD cujo objetivo é assegurar a aderência do produto de software aos requisitos de negócio, promovendo valor ao cliente perante as funcionalidades desenvolvidas, independente da legibilidade e boa estrutura do código. O *Acceptance* TDD estabelece a qualidade no software sob a ótica do cliente (qualidade externa).

O capitulo seguinte apresenta as características do *Acceptance* TDD, suas práticas e resultados no projeto de desenvolvimento de software.

3. ACCEPTANCE TDD - ATDD

Compreender exatamente quais são as necessidades do cliente é um dos grandes desafios do desenvolvimento de software. Visando meios para compreender melhor as necessidades do cliente, a abordagem ágil de desenvolvimento de software propõe a técnica do *Acceptance Test Driven Development* (*Acceptance* TDD ou, simplesmente, ATDD), que consiste em criar testes de requisitos de negócio antes de implementar o código, fazendo com que a implementação das funções sejam guiadas por testes de aceitação desenvolvidos com a colaboração do cliente (KOSKELA, 2007, p.324.).

O objetivo do ATDD é encorajar uma comunicação clara das necessidades de negócio através do uso de exemplos concretos de funções de software. Assim, os testes são parte integrante da especificação de requisitos de um produto de software. Então, requisitos e testes estão ligados, e não se pode ter um sem o outro (PUGH, 2011. p.27).

O ATDD visa solucionar o que geralmente é falho em um projeto de desenvolvimento de software: a colaboração entre o cliente e o desenvolvedor, que ao invés de basear-se somente em uma documentação formal, que muitas vezes esta sujeita a erros, existe uma interação estreita com o responsável pelo requisito, o cliente, que deve prover as informações de forma explícita, lúcida e inequívoca de suas necessidades. Esta colaboração além de propiciar uma sinergia entre as partes favorece ao desenvolvimento de produtos de software cujas funções estão alinhadas com as reais expectativas do cliente.

Neste capítulo são apresentadas as características do ATDD, o seu processo e os resultados de aplicá-lo no projeto de desenvolvimento de software.

3.1 Testes de Aceitação

Na literatura é possível encontrar vários termos diferentes associado ao conceito de teste de aceitação. Alguns exemplos são ilustrados na Tabela 1. Apesar da variedade de termos é importante compreender a essência dos testes de aceitação.

O teste de aceitação tem seu foco em verificar que o sistema atende aos requisitos do cliente. A motivação dos testes de aceitação é mostrar que o software funciona ao invés de encontrar defeitos, no entanto, defeitos podem ser identificados como resultado desses testes.

A norma IEEE 1012 (1998. p.4 ; p.30.) define teste de aceitação como testes formais conduzidos pelo cliente para determinar se o sistema satisfaz ou não seus critérios de aceitação, determinando também se o sistema deve ou não ser aceito.

Tabela 1 - Sinônimos para "*Acceptance Tests*" (MELNIK, 2007. p.12.)

Termo	**Apresentado / Usado por**
functional tests	BECK, *Extreme Programming Explained*
customer tests	JEFRIES, BECK, *Extreme Programming Explained*
customer-inspired tests	BECK, *Extreme Programming Explained*
story-tests and story-test-driven development	KERIEVSKY
specification by example	FOWLER
coaching tests	MARICK
examples, business-facing example, and example-driven development	MARICK
conditions of satisfaction	COHN
scenario tests	KANER
keyword-driven test	KANER, BACK, PETTICHORD
soap opera tests	BUWALDA
formal qualification test	e.g. DOD
user acceptance tests (UAT)	e.g. IEEE610
client acceptance tests	e.g. IEEE610
system tests	e.g. IEEE610

Geralmente, os testes de aceitação são realizados por um grupo restrito de usuários finais do sistema, que simulam operações de rotina do sistema de modo a verificar se o seu comportamento está em conformidade com o solicitado. Os testes de aceitação podem contemplar os testes funcionais, de recuperação de falhas, de segurança, de configuração, e de desempenho.

Contudo, os testes de aceitação apenas tem eficácia com o apóio do cliente na definição dos critérios de aceitação, a sua ausência torna difícil validar se o software está sendo construído de forma correta. Logo, ter o cliente próximo a equipe de desenvolvimento para definir o sistema através de cenários é de fundamental importância para o sucesso do projeto de software.

Melnik (2007, p.13.) define cenário como uma descrição de um possível conjunto de eventos que pode ocorrer. O principal objetivo de desenvolver cenários é estimular a reflexão sobre possíveis ocorrências e suas suposições, oportunidades e riscos.

A ausência de testes de aceitação e a correta compreensão dos requisitos são as principais razões para inúmeras falhas de projeto de desenvolvimento de software.

Definir os requisitos com clareza através de cenários e critérios de aceitação oferecem boas possibilidades para a construção de software que atenda às expectativas do cliente. Neste contexto, os métodos ágeis de desenvolvimento de software propõem o ATDD cujos conceitos e práticas são descritos nas seções seguintes.

3.2 ATDD

O ATDD, proveniente do desenvolvimento ágil de software, move os testes de aceitação para antes da codificação. O processo é similar ao TDD, apenas abordando em um nível de teste diferente. Enquanto o TDD trata de comportamento de pequenas unidades, o ATDD trata de comportamento de todo sistema (KOSKELA, 2007. p.334.).

Fazendo uma alusão ao conceito de qualidade interna, perspectiva do desenvolverdor, e qualidade externa, perspectiva do cliente, o TDD refere-se à qualidade interna do código, enquanto o ATDD refere-se à qualidade externa do código, conforme ilustrado na Figura 3.

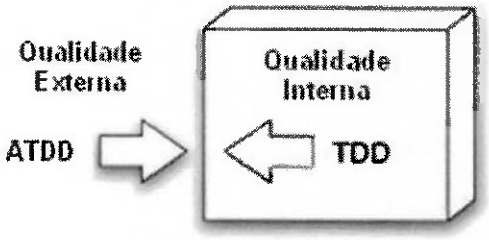

Figura 3 - Perspectivas de qualidade externa e interna (KOSKELA, 2007. p.7.)

No ATDD, os testes de aceitação são elaborados com base nas estórias de usuário, que são escritas em pequenos cartões contendo poucas palavras ou frases que dentro de um contexto transmitem significado ao cliente e desenvolvedor, e que, geralmente, expressam quem faz o quê.

As estórias de usuários são sucintas para representar os requisitos do sistema, ou seja, não pretendem documentá-lo, atuando como uma promessa de diálogo futuro entre o cliente e o desenvolvedor (KOSKELA, 2007. p.325.). Neste compromisso de reunir os *stakeholders* do projeto, é realizada uma discussão sobre os critérios de aceitação que envolvem os requisitos do cliente, onde ele deve descrever, através de exemplos concretos de funções, o comportamento correto do sistema através de sua perspectiva – condições de satisfação. Com o apoio da equipe técnica, essencialmente, desenvolvedor e testador, os exemplos especificados pelos usuários são traduzidos em forma de testes automatizados, isto é, testes de aceitação executáveis (*executable acceptance tests*), que posteriormente devem ajudar os desenvolvedores a determinar quando as novas funções estão completas ou verificar de forma ágil e eficiente se alguma função existente apresentou falha.

O ATDD não determina uma técnica de implementação do software, mas considera o uso do TDD mais adequado ao espírito do desenvolvimento ágil. Nesta etapa, em geral, uma estória é dividida pelos desenvolvedores em um conjunto de tarefas necessárias para criar a funcionalidade. Então, os

desenvolvedores atuam sob essas tarefas usando quaisquer ferramentas necessárias, incluindo o TDD. Quando uma determinada tarefa é concluída, o desenvolvedor passa para a próxima tarefa, e assim por diante, até que a estória seja concluída, evidenciado pelo sucesso na execução de testes de aceitação.

Nas seções seguintes são apresentadas em detalhes as características do ATDD.

3.2.1 Propriedades do ATDD

Segundo Koskela (2007. p.328.) o ATDD possui cinco propriedades, descritas a seguir.

- **De propriedade do cliente:** Os testes de aceitação devem ser de propriedade do cliente, onde seu principal propósito é especificar os critérios de aceitação para a estória do usuário. Além disso, o cliente é o especialista de negócio que tem habilidades específicas para esta tarefa. Logo, o cliente torna-se responsável por escrever os testes de aceitação. Com isso, é possível evitar um problema muito comum com os testes de aceitação escritos por desenvolvedores, que, geralmente, especificam os aspectos técnicos da implementação ao invés de especificar os aspectos da funcionalidade.

- **Escrito em conjunto com o cliente, desenvolvedor e testador:** Embora o cliente seja o único proprietário dos testes, isto não significa que ele deve ser o único a escrevê-lo. A equipe técnica deve suportar o cliente nesta tarefa. Escrever os testes em conjunto melhora a comunicação entre os *stakeholders*, dissemina o conhecimento e esclarece o entendimento do sistema, possibilitando produzir software de qualidade.

Pugh (2011. p.3.) refere-se a colaboração entre o cliente, desenvolvedor e testador como a tríade, onde:

- O Cliente pode ser representado pelo *Product Owner*, analistas de negócios e especialistas de domínio, cujas responsabilidades são determinar os requisitos, criar os testes de aceitação, e definir as prioridades.

- O Desenvolvedor implementa os requisitos e assegura que o software atende aos critérios de aceitação. Também apóia o testador no processo de automatização dos testes de aceitação.

- O Testador apoia o cliente no desenvolvimento dos testes de aceitação, e verifica se a implementação faz o que deve fazer e não faz o que não deve fazer.

- **Sobre *o quê* e não *como*:** Uma das principais características que torna as estórias de usuários apropriadas à entrega antecipada de valor é o seu foco em descrever o que agrega real valor para o cliente ao invés dos mecanismos de como obter este valor. Estórias empenham-se em transmitir as necessidades e desejos – *o quê* e, às vezes, *por quê*– e deixam as questões de implementação – *como* – de lado. Geralmente, o cliente não se importa como será implementado o software, desde que ele atenda a suas necessidades e expectativas.

- **Expresso na linguagem de domínio do problema:** Uma importante propriedade dos testes de aceitação, no contexto do ATDD, é usar a linguagem de domínio do cliente e não jargões de programação. Esta é uma premissa fundamental para ter o cliente envolvido no processo de criação e validação dos testes. O uso de jargões técnicos torna os testes vulneráveis a falhas, pois desvia o foco do cliente à verdadeira necessidade: especificar a coisa certa.

- **Conciso, preciso e inequívoco:** Os testes de aceitação são escritos para verificar um único aspecto ou cenário relevante para a estória do usuário. Logo, esses testes devem ser bem organizados, fáceis de entender e traduzíveis para testes executáveis. Quanto menor a ambigüidade, melhor é para mitigar erros. As estórias podem ser escritas como lembretes simples na forma de uma lista com marcadores, ou pode-se optar por explicá-las como frases completas que descreve o comportamento esperado. Em ambos os casos, o objetivo é fornecer informações suficientes para lembrar as coisas importantes que precisam ser discutidas e testadas, ao invés de documentar os detalhes com antecedência. A idéia é manter os testes de aceitação simples e concisos, portanto, evitar escrever detalhes que são fáceis de ser identificados posteriormente na implementação, que podem ser interpretados erroneamente, e que não acrescentam informação crucial devem ser omitidos.

3.2.2 Perfil do cliente

Como é possível verificar nas propriedades do ATDD, o cliente tem um papel indispensável à viabilidade desta técnica.

Na prática existem variações de clientes em um projeto de software. O cliente que paga pelo produto de software, quem usará o sistema, em alguns casos, quem venderá o sistema, etc. (KOSKELA 2007. p.348.).

No contexto do ATDD, o papel do cliente deve ser preenchido por alguém que detenha um amplo conhecimento do domínio do negócio.

Por vezes, projetos complexos exigem competências que não cabem somente a uma única pessoa, fazendo-se necessário um grupo de indivíduos que represente o papel do cliente.

Portanto, há algumas propriedades recomendáveis ao papel do cliente no ATDD (KOSKELA, 2007. p.349):

- O cliente é membro ativo da equipe e deve estar comprometido com o sucesso do projeto;
- O cliente tem autoridade para tomar decisões e deve ter a capacidade de compreender as implicações técnicas destas decisões com o apoio da equipe técnica;
- Por fim, o cliente deve ser capaz de explicar com clareza o domínio da aplicação para a equipe do projeto. Todavia, um cliente capaz de comunicar-se efetivamente com a equipe do projeto é um fator essencial para o sucesso do projeto.

Uma vez que conhecemos as principais características inerentes ao ATDD, na seção seguinte são apresentadas as etapas que compreendem esta técnica.

3.3 O ciclo de ATDD

Segundo Koskela (2007) o ATDD pode ser expresso com o ciclo ilustrado na Figura 4. Esse ciclo continua por toda a iteração enquanto houver estórias para implementar, começando na escolha de uma estória de usuário, em seguida, os testes de aceitação são escritos para estória em questão, então, esses testes são automatizados, tornando-se testes executáveis e, por fim, a funcionalidade é implementada para passar pelos testes de aceitação executáveis.

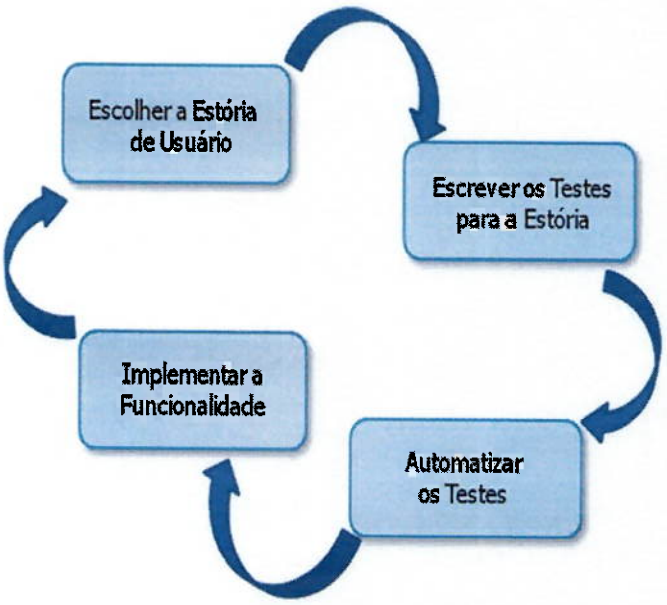

Figura 4 - Ciclo de ATDD (KOSKELA, 2007. p.335.).

A seguir, são descritos os detalhes do ciclo de ATDD.

3.3.1 Escolher a Estória de Usuário

A primeira etapa consiste em decidir em qual estória de usuário trabalhar. As estórias são provenientes das reuniões de planejamento realizadas durante todo o projeto de desenvolvimento do software. Nesses encontros, os clientes informam suas necessidades e expectativas através de exemplos concretos de funções, ilustrando como o sistema deve funcionar para cada situação.

Os desenvolvedores e testadores, tipicamente, fazem perguntas sobre essas características, tornando um meio de intenso aprendizado e discussão. Algumas dessas informações ficam documentadas em cartões de estória, onde o cliente prioriza as estórias pelo seu valor de negócio, incluindo o risco do negócio, e o risco técnico é estimado pela equipe de desenvolvimento.

3.3.2 Escrever os Testes para a Estória

Após escolher a estória, a segunda etapa é escrever os testes de aceitação para esta estória. Esta tarefa é de responsabilidade do cliente. Logo, os membros da equipe técnica, desenvolvedores e testadores, devem estar junto com o cliente e começar a esboçar uma lista de testes para a estória em

questão, de modo que esta lista contemple os cenários e aspectos da estória que serão testados e irão garantir que a funcionalidade será implementada corretamente.

Uma vez esboçada a lista de testes, é iniciada a elaboração do teste de aceitação propriamente dito, acrescentando os detalhes e discutindo sobre as características, especificidades sobre a interface do usuário, a melhor forma de funcionamento, etc.

3.3.3 Automatizar os testes

A terceira etapa do ATDD é transformar os testes de aceitação em testes executáveis que retorne um resultado simples: sucesso ou falha.

Existem diversas ferramentas que realizam a automatização desses testes. As ferramentas mais conhecidas utilizam o conceito *table-based*, onde a premissa é usar o formato tabular, linhas e colunas, para facilitar a especificação dos testes de modo que sejam legíveis para o homem e a máquina.

Uma vez que o teste é expresso em um formato tabular, a ferramenta interpreta este conteúdo em tabelas HTML e transforma-os em seqüencias de métodos à ser invocados e interpretados pelo código do sistema sob teste.

Na maioria das vezes, não existe uma ferramenta disponível capaz de compreender os testes na linguagem de domínio, em formato tabular, e ser capaz de ligar os testes em chamadas para o sistema sob teste. Na prática, esta comunicação deve ser implementada pelos desenvolvedores usando uma linguagem de programação.

Simplificando esta dualidade de transformar os testes de aceitação em testes executáveis, devem-se expressar os testes em um formato tabular, legível ao homem e máquina, e escrever o código que conecte os testes ao sistema a ser testado.

Um aspecto a ser considerado é a necessidade de desenvolver todos os testes executáveis no início do projeto ou automatizá-los em pequenos passos

com o processo de implementação, semelhante ao TDD, onde implementa-se um teste, em seguida, implementa o código para o parte da estória, então, implementa-se um outro teste, e assim por diante. A desvantagem em automatizar previamente todos os testes de aceitação é o risco de ter retrabalho durante o projeto. Logo, implementar os testes executáveis de forma gradativa, além de ser mais prudente, pode evitar retrabalho.

Portanto, uma vez concluída esta etapa, tem-se um teste de aceitação que pode executar e informar que a funcionalidade especificada está ausente. A próxima etapa é, naturalmente, fazer o teste passar, isto é, implementar a funcionalidade para satisfazer o teste que falhou.

3.3.4 Implementar a funcionalidade

A quarta etapa do ciclo de ATDD consiste em implementar a funcionalidade. O ATDD não determina uma técnica de implementação para a funcionalidade, mas considera como uma melhor prática o uso do TDD.

Usualmente, uma estória representa uma funcionalidade de valor para o cliente. Esta estória é desdobrada em um conjunto de tarefas necessárias para criar esta funcionalidade. Então, os desenvolvedores atuam sob essas tarefas usando quaisquer ferramentas necessárias para garantir a implementação das funções, incluindo o TDD.

Na prática, este processo contempla inúmeras iterações pequenas dentro de iterações. A Figura 5 ilustra a transição entre os processos de ATDD e TDD.

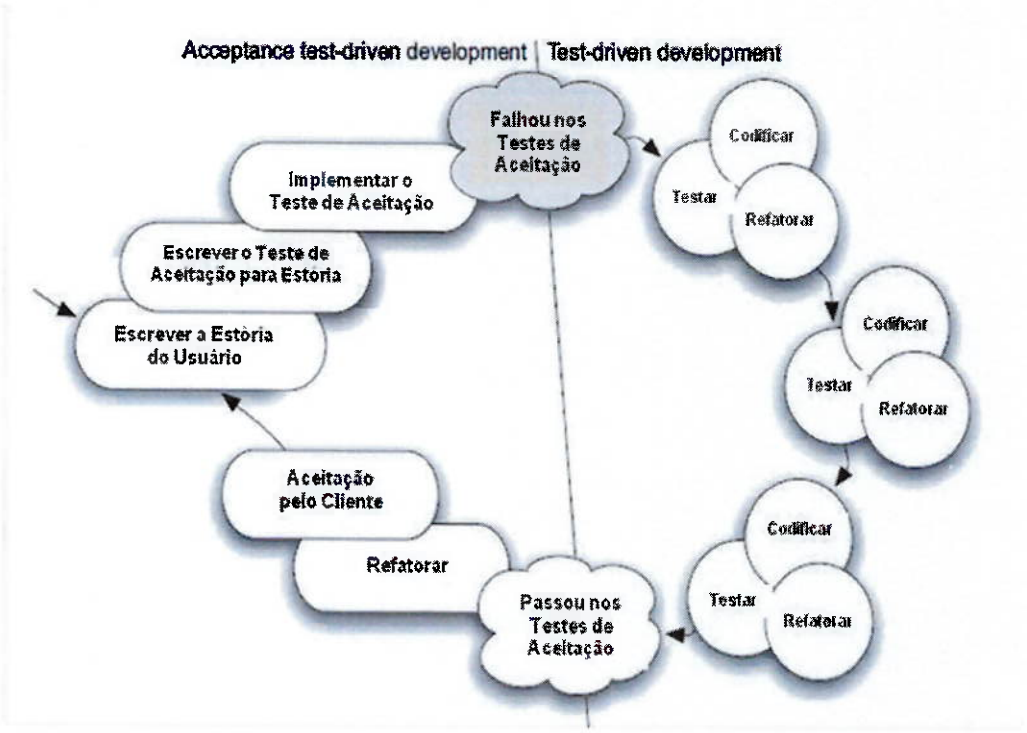

Figura 5 - Relacionamento entre o ATDD e TDD (KOSKELA, 2007, p.342.).

Como é ilustrado na Figura 5, a quarta etapa do ciclo de ATDD, implementa a funcionalidade necessária para corrigir o teste de aceitação que falhou inicialmente, e pode ser expandido em uma seqüência de pequenos ciclos de TDD (*Test-Code-Refactor*), construindo a funcionalidade ausente de forma fragmentada até que todos os testes de aceitação sejam bem sucedidos.

Enquanto o desenvolvedor está trabalhando em uma estória, frequentemente o cliente é consultado sobre como uma funcionalidade ou outra deveria funcionar. Desta forma, há ocasiões onde o desenvolvedor identifica um cenário de teste que o sistema provavelmente deve lidar, e que não foi previamente elaborado pelo cliente, desenvolvedor e testador. Logo, os testes de aceitação devem ser adicionados à lista após o aval do cliente, que deve refletir sobre o novo cenário de teste. Afinal, ele pode não atribuir tanto valor à um determinado aspecto ou funcionalidade da estória.

Nesta etapa do processo, dependendo de qual foi a opção adotada na etapa anterior, entre automatizar previamente todos os testes ou de forma gradativa, deve-se retornar a etapa 1 e escolher uma nova estória para trabalhar, ou retornar a etapa 3 e automatizar um outro teste, respectivamente.

Para contextualizar as quatro etapas do processo de ATDD proposta por Koskela (2007), a Figura 6 ilustra o respectivo fluxograma.

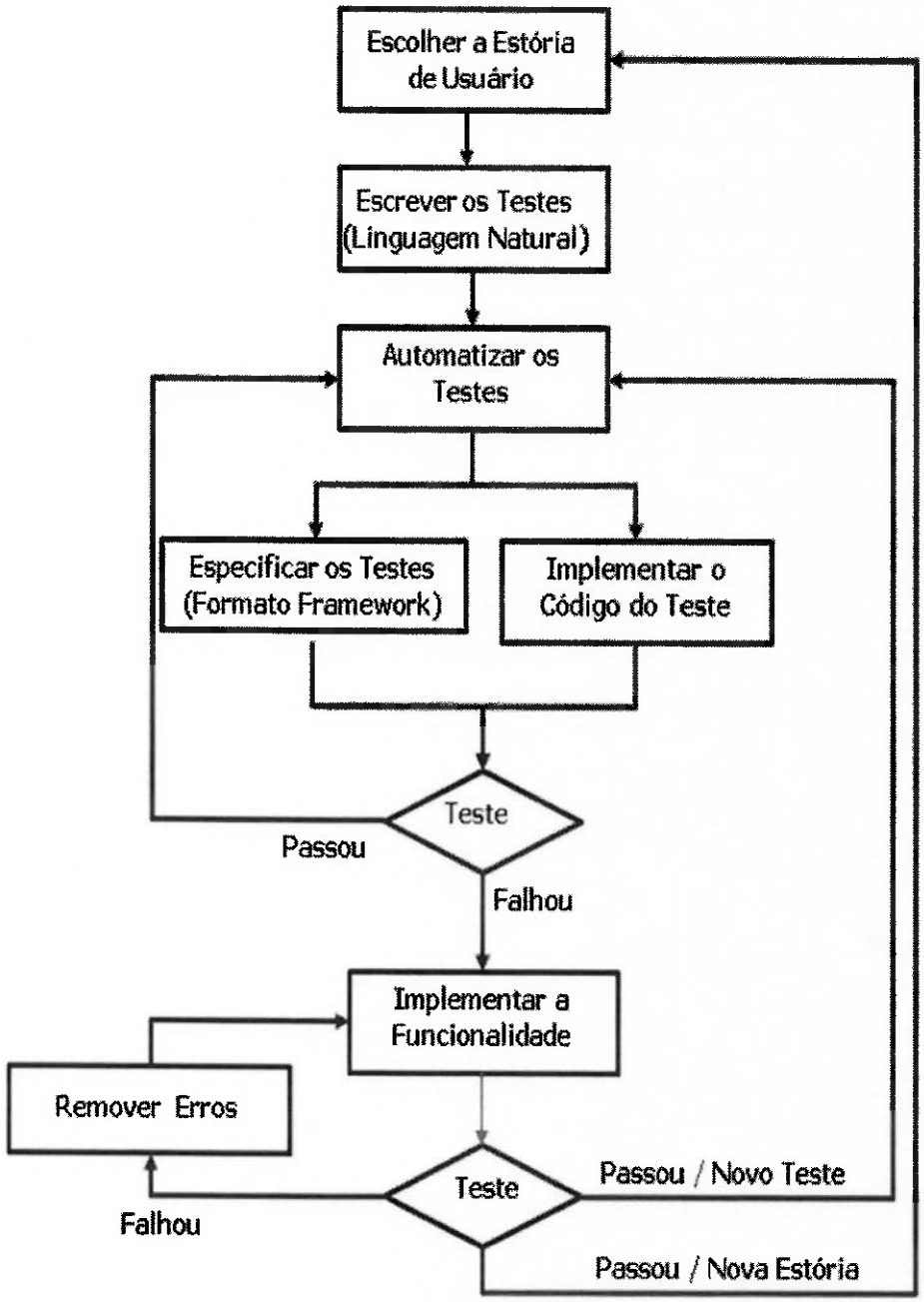

Figura 6 - Fluxograma do ATDD

Hendrickson (2008) descreveu em seu artigo uma abordagem ao ciclo ATDD, conforme ilustrado na Figura 7.

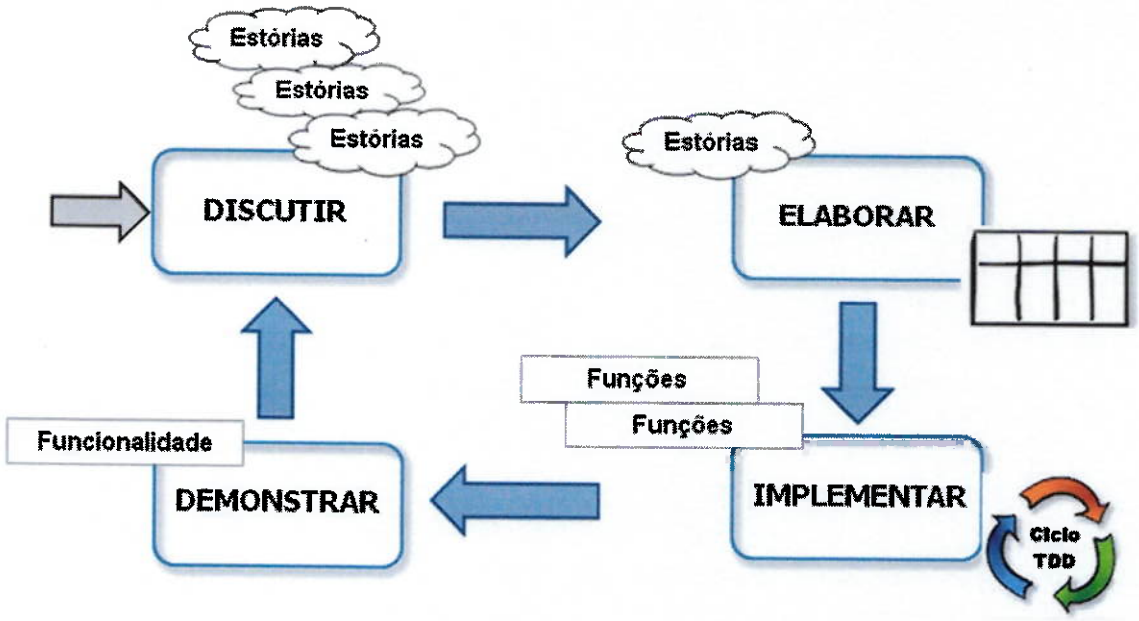

Figura 7 - ATDD sob a perspectiva de Hendrickson (adaptado de HENDRICKSON, 2008. p.3.)

A seguir, é feita uma breve descrição de cada uma das fases do ciclo ATDD.

a) Discutir

Durante a reunião de planejamento da iteração são discutidas as estórias de usuários. Nesse evento devem estar presentes todos os *stakeholders* do projeto que possuem informações relevantes sobre os requisitos do sistema. Esta etapa do processo corresponde às seguintes ações:

- Os critérios de aceitação são definidos para cada estória;

- A equipe técnica deve fazer as perguntas certas ao cliente cujas respostas devem resultar em exemplos de uso que podem ser escritos como testes/requisitos. Por exemplo: "Imagine o sistema pronto. Como o sistema seria utilizado e o que é esperado dele?" (LARMAN e VODDE, 2010. p.2.);

- O cliente deve pensar cuidadosamente sobre a funcionalidade e suas expectativas;

- As estórias dos usuários podem, e normalmente são desdobradas em estórias menores;

- Os testes de aceitação são esboçados com a colaboração do cliente para cada estória. Os testes são escritos em linguagem natural.

b) Elaborar

Uma vez que os testes foram esboçados com a colaboração do cliente é possível traduzi-los em um formato que seja legível para o *framework* de automação.

Para cada estória do usuário, os testes devem ser reescritos em formato aderente a um *framework* de automação, por exemplo, em formato tabular. Neste momento ainda não há a preocupação de como os testes serão automatizados, o foco deve estar na essência dos testes e os detalhes de implementação devem ser ignorados.

c) Implementar

Esta etapa do processo compreende às seguintes ações:

- Desenvolvedores executam os testes de aceitação executáveis para a estória e os testes falham, pois os testes ainda não estão ligados ao código de teste;

- Desenvolvedores implementam o código de teste que ligará os testes de aceitação e o código da aplicação;

- Desenvolvedores executam os testes de aceitação executáveis e os testes falham, pois o código da aplicação ainda não foi implementado;

- Desenvolvedores implementam o código da funcionalidade (estória) utilizando o TDD e executam os testes de aceitação, que devem ser bem sucedidos.

d) Demonstrar

Na última etapa do processo, é importante a execução de testes exploratórios manuais para verificar se há erros nos critérios de aceitação e eventuais riscos que não foram identificados durante o processo.

Uma vez que os resultados da implementação satisfazem às expectativas da equipe de desenvolvimento é realizada uma demonstração da funcionalidade para os *stakeholders* de negócios, apresentando os resultados obtidos e os riscos em potencial identificados durante o processo.

Para a melhor compreensão de como o ciclo de ATDD se encaixa em uma iteração, a próxima seção apresenta uma visão do ATDD na linha do tempo.

3.4 ATDD dentro de uma iteração

As iterações devem ter duração de algumas horas a poucos dias para escrever os testes de aceitação, tornando-os testes executáveis e implementar a funcionalidade. Portanto, as etapas do ATDD é apenas uma fração de toda uma iteração feita de múltiplas, e pode ir além de dezenas de estórias de usuário, dependendo do tamanho da equipe e das estórias.

Para compreender melhor como o ciclo de ATDD se aplica a uma única estória e encaixa-se em uma iteração, a Figura 8 ilustra uma iteração na linha do tempo relacionado-a com as etapas do ATDD. Esta ilustração apresenta uma única iteração com cinco estórias de usuário à serem implementadas. Cada uma das barras representa uma única estória de usuário que move

através das etapas do ATDD (Escrever os Testes, Automatizar os Testes e Implementar as Estórias).

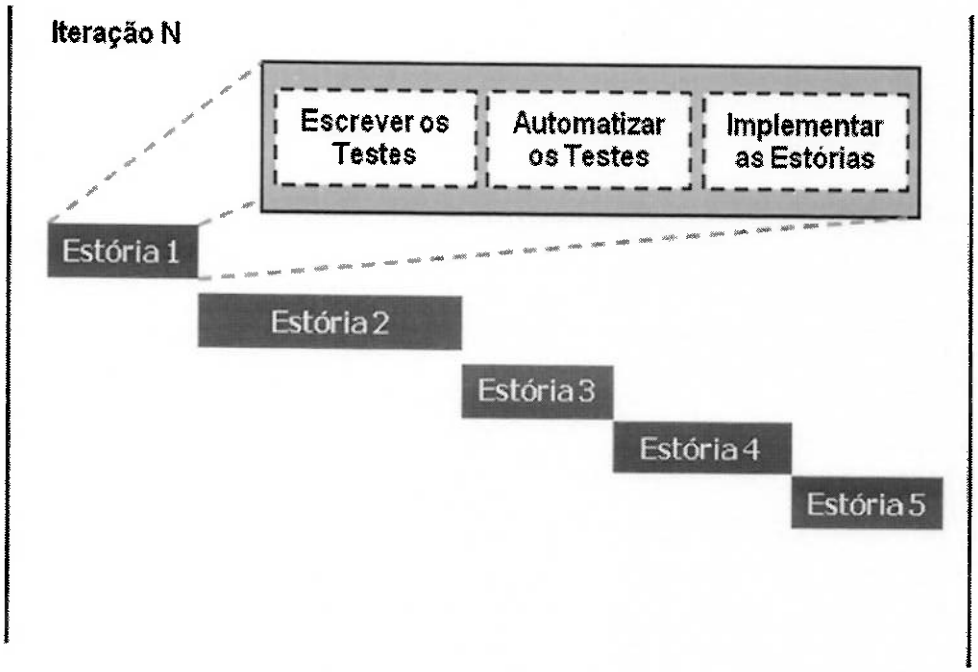

Figura 8 - ATDD na linha do tempo (adaptado de KOSKELA, 2007. p.344.).

Na prática, pode haver, e geralmente há, mais iterações em cada estória porque usualmente não se escreve e implementa todos os testes de aceitação de uma única vez, ao invés disto procede-se com os testes um a um.

É possível notar na Figura 8 que a iteração não contempla a etapa de escrever as estórias de usuário. Isto está associado a uma prática dos métodos ágeis, a reunião de planejamento, que é realizada no início de cada iteração onde o cliente discute os aspectos da estória e decide qual estória tem prioridade de implementação na iteração. Se as estórias são priorizadas nesta reunião, logo elas devem ter sido discutidas e escritas previamente.

Considerando o fato que há uma reunião de planejamento a cada iteração, é uma boa prática preparar a próxima iteração durante a iteração corrente, alocando uma pequena parcela de tempo para o pré-planejamento da próxima iteração. A ideia é estar preparado para a próxima iteração. Então, quanto maior o conhecimento adquirido previamente, mais rápido e produtivo será o planejamento da iteração.

3.5 Resultados do ATDD no projeto de software

Como já foi citado, um aspecto positivo do ATDD é que ele é um mecanismo de comunicação entre o cliente, desenvolvedor e testador, que além de esclarecer os requisitos do cliente, especifica como o sistema deve funcionar (PUGH, 2011. p.199.).

Outros resultados relevantes podem ser destacados na adoção do ATDD, como:

- **Guia de completude**

  Com a prática do ATDD é possível avaliar com clareza a evolução do desenvolvimento de software, identificando sua situação atual no projeto. O ATDD oferece um critério binário para isto, ao responder a seguinte questão: "Os testes para todas as estórias passaram?" (KOSKELA, 2011. p.351.).

  A relação de testes de aceitação bem sucedidos com o número total de testes podem fornecer um guia para o quanto da estória foi implementada. Por exemplo, se há dez testes de aceitação e três foram executados, a estória está "em média" 30% concluída.

  É possível que o caso de teste mais complexo não foi implementado. Assim, o esforço para implementar essa estória representa mais do que sua parte justa do esforço total. É por isso que este é apenas um guia para determinar a completude, ao invés de uma medida exata. (PUGH, 2011. p.199.)

- **Trabalho colaborativo**

  ATDD cria um ambiente cooperativo porque todos os envolvidos trabalham com os mesmos objetivos ao mesmo tempo. Esta é uma maneira mais efetiva de trabalhar, especialmente, considerando como cada indivíduo colabora com seus conhecimentos e habilidades para resolver os problemas.

- **Confiança e comprometimento**

  Os testes de aceitação são escritos para especificar o comportamento de uma estória de usuário. Quando o cliente escreve os testes de aceitação juntamente com a equipe de desenvolvimento, isto consolida o significado da estória do usuário porque há uma conexão direta entre o que o cliente especificou e o que foi implementado.

  O cliente vê que suas necessidades são efetivamente atendidas, e os desenvolvedores vêem que o trabalho realizado oferece valor ao cliente. Desta forma, a maneira como os testes são escritos no ATDD tem muito significado à ambas as partes.

- **Especificação através de exemplos**

  Um aspecto importante do ATDD é que os testes de aceitação representam especificação através de exemplos, é um modo de especificar requisitos por meio de exemplos inteligíveis ao invés de fórmulas complexas e sentenças ambíguas. Isto significa que testes expressos com exemplos concretos de funções são fáceis de compreender, validar e, principalmente, escrever, uma vez que o objetivo é ter o cliente envolvido na especificação e elaboração dos testes do sistema. Além disso, mau-entendimentos sobre os requisitos são minimizados.

- **Preenchendo a lacuna**

  Independente de quão bem os testes de unidade oferecem cobertura do código, eles não testam o sistema. Os testes de unidade verifica o código e os componentes isolados. Isto é, testes de unidade oferecem um bom indicativo se o código funciona, mas sem alguns testes adicionais não é possível saber se o sistema como um todo funciona como deveria. Embora os testes de unidade, especialmente na forma de *Test-First Programming*, seja uma prática extremamente útil na produção de software de qualidade, é preciso algo a mais para colocar todo o

sistema sob validação. Logo, as práticas de TDD e ATDD são complementares e não substituíveis.

- **Visualização do software em construção**

  Software por natureza é algo abstrato, de difícil compreensão. Mesmo sabendo qual o problema que deve ser solucionado, o cliente não tem a capacidade de abstrair a definição de uma solução. É muito mais fácil visualizar um software funcionando e, então, decidir se aquilo é ou não é a solução do problema. Desenvolvendo os testes de aceitação executáveis, os clientes terão uma oportunidade de visualizar o software funcionando, permitindo o aprendizado e o *feedback* para aprimorar os requisitos.

- **Apóio à estimativa**

  Uma estimativa pode ser necessária para a implementação de uma estória. O número e a complexidade dos testes de aceitação pode ser um guia para determinar o esforço necessário para implementar uma estória. Os testes para uma nova estória pode ser comparada para os testes executados em estórias concluídas. É possível desenvolver uma heurística própria de como o número e a complexidade pode influenciar no esforço. Grandes configurações personalizadas e numerosas mudanças de estado geralmente implicam em um esforço muito maior do que os testes com configurações de pequenas e poucas mudanças de estado (PUGH, 2011. p.200.).

- **Teste de Regressão**

  Definir testes de aceitação para todos os requisitos permite usar o conjunto de testes de aceitação como testes de regressão. Se uma alteração é feita para implementar um novo requisito, todos os testes de aceitação anteriores ainda devem ser bem sucedidos. Se os testes anteriores falham quando uma nova exigência é introduzida, é possível ter problemas no projeto do código. Contudo, se os testes de aceitação

são automatizados, executá-los rapidamente fornece um *feedback* imediato sobre as modificações no sistema.

Empregar o ATDD implica em investir um alto custo financeiro e intelectual para automatizar os testes de aceitação. A seguir, são descritos os fatores críticos que podem inviabilizar o uso do ATDD:

- **Automatização de testes**

  Estabelecer os testes de modo à automatizá-los pode ser um grande desafio mesmo fazendo uso de ferramentas, pois automatizar os testes exige grande esforço na elaboração e manutenção dos testes, além de muita disciplina e ampla experiência da equipe técnica na adoção das ferramentas de automação dos testes. (PARK e MAURER, 2008. p.20.).

- **Restrição a Interfaces de Usuário**

  Há situações onde a usabilidade de algumas interfaces de usuário, principalmente, as interfaces gráficas dificultam a automatização dos testes ao ponto de inviabilizá-los (KOSKELA, 2011. p.352.). É recomendável manter os testes de lógica de negócio separados dos testes de interfaces de usuário (PUGH, 2011. p.136). Hoje muitos especialistas concordam que a automação deve ser feito a nível abaixo da interface de usuário, nível este que correspondente à lógica de negócio (MELNIK, 2007. p.22.).

- **Manutenção em dobro**

  Alteração nos requisitos do sistema implica na atualização do código da aplicação e dos testes, ambos devem estar sempre sincronizados (MARTIN, 2007. p.34. ; PARK e MAURER, 2008. p.20.). Isto exige dos desenvolvedores, manutenção em dobro sobre os códigos, o que pode sobrecarregar os desenvolvedores e comprometer os prazos do projeto.

3.6 Ferramentas de Testes de Aceitação

Usar a ferramenta certa no desenvolvimento de software pode possibilitar muita facilidade à equipe de desenvolvimento.

Há uma variedade de *frameworks* que suportam os testes de aceitação executáveis, o FIT (*Framework for Integrated Tests*) é o mais popular na indústria de software.

Concebido por Cunningham (2002, fit.c2.com/wiki.cgi?FitDocumentation) o FIT permite compreender e escrever os testes de aceitação de maneira fácil, expressando-os através de um formato tabular (*Table-based*). Utilizando-se de ferramentas convencionais como o Word e Excel, ou HTML, os clientes especificam os testes na tabela (*Fixture table*), em suas linhas e colunas, usando a terminologia regular do negócio e organizando as afirmações acerca das regras de negócios que o software deve satisfazer. Então, um código de teste (*Fixture class*) é implementado pelos desenvolvedores para mapear os dados presentes nas tabelas e ligá-los ao software sob teste. Para escrever a *Fixture class* várias linguagens são suportadas pelo FIT, como o Java, C#, Ruby, C++, Python. Este código deve ser transparente aos clientes.

A Figura 9 ilustra o mecanismo adotado pelo FIT.

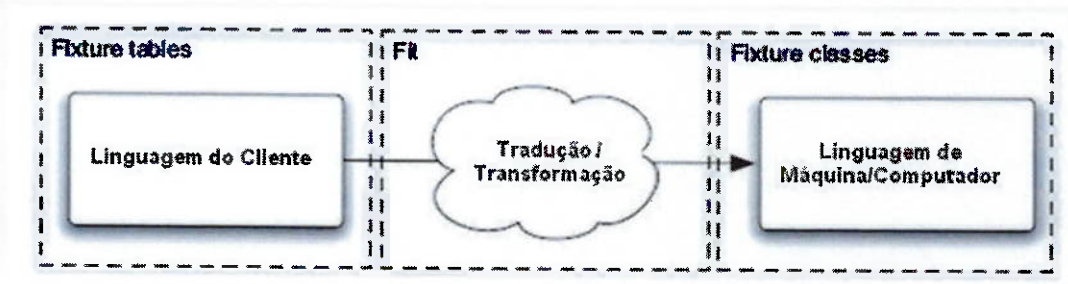

Figura 9 - Lógica da ferramenta FIT (KOSKELA, 2007. p.369.).

Uma vez que o código de teste e o código da aplicação estão implementados, o FIT pode atuar na execução dos testes que deve validar a aplicação candidata à produção. O resultado obtido pelo FIT tem o padrão apresentado na Figura 10, onde as células são destacadas com as cores verde e vermelha para indicar o sucesso e falha do teste, respectivamente. Quando o

teste falha, os resultados esperados e o atual são apresentados um ao lado do outro.

com.tddinaction.fit.fixtures.CalculatorFixture			
Esquerda	direita	Operador	resultado()
5	3	+	*8 esperado*
			2 atual
5	3	-	*2 esperado*
			8 atual
5	3	*	15
5	3	/	1.666666667

Figura 10 - Resultado apresentado pelo FIT (KOSKELA, 2007, p.375.)

O FIT apóia a uma necessidade fundamental no desenvolvimento de software, preencher a lacuna entre o cliente e o desenvolvedor em documentar com clareza as regras de negócio. Contudo, como o FIT foi projetado para trabalhar em tabelas HTML ou planilhas, então, Cunningham (2002) percebeu que seria eficiente se os testes pudessem ser criados e editados em uma wiki, tornando fácil o compartilhamento dos testes de aceitação entre os *stakeholders* do projeto de software. E assim, no ano de 2005, foi concebido o FitNesse.

FitNesse é uma wiki que provê uma ferramenta colaborativa de teste e documentação baseada na web cuja concepção é o FIT. Ela fornece uma maneira simples para que as equipes de forma colaborativa possam criar documentos, especificar testes, e até mesmo executar os testes através da Wiki website (FITNESSE Documentation Online).

Além dos *frameworks* FIT e FitNesse que utilizam do formato tabular para expressar os testes de aceitação, os *frameworks* que utilizam dos formatos de texto (*Text-based*) e linguagem de scripts (*Scripting Language*) também são adotados na indústria de software.

A Tabela 2 relaciona alguns *frameworks* de testes de aceitação reconhecidos na indústria de software, suas respectivas categorias e website.

Tabela 2 - *Framework* de testes de aceitação (adaptado de PUGH, 2011. p.298.).

Framework	**Categoria**	**Website**
Concordian	*Table-based*	http://www.concordion.org
Cumcuber	*Text-based*	http://cukes.info
Easyb	*Text-based*	http://www.easyb.org
Exactor	*Text-based*	http://exactor.sourceforge.net/
FIT	*Table-based*	http://fit.c2.com
FitNesse	*Table-based*	http://fitnesse.org
Jbehave	*Text-based*	http://jbehave.org
Robot	*Table-based*	http://cde.google.com/p/robotframework/
Selenium	*Table-based*	http://seleniumhq.org
Systir	*Scripting Language*	http://rubyforge.org/projects/systir
Watir	*Scripting Language*	http://watir.com

3.7 Considerações do Capítulo

O ATDD apóia diversos valores e algumas práticas ágeis, como por exemplo, a importância dos indivíduos e a colaboração entre eles de modo que favoreça uma comunicação eficaz, o respeito e benefício mútuo; a simplicidade dos exemplos para facilitar as soluções de problemas e o *feedback* como norteador da qualidade.

O ATDD preza pela execução automatizada dos testes, onde os testes de aceitação são transformados em testes executáveis cuja finalidade é validar as regras de negócio do software.

A automatização dos testes é importante, principalmente, em projetos de desenvolvimento cujas mudanças são constantes e o rápido *feedback* se faz necessário. No entanto, automatizar os testes de aceitação demanda alto investimento financeiro, esforço e exige habilidades técnicas que podem tornar os custos do projeto elevado, inviabilizando a automatização desses testes. Contudo, negligenciar os testes de aceitação, além de não ser uma melhor prática, pode comprometer a qualidade do software e, consequentemente, promover a insatisfação do cliente.

O capítulo seguinte descreve uma proposta de adaptação do ATDD cujos princípios são preservados, no entanto, a etapa de automatização dos testes é substituída pelo planejamento e a elaboração estruturada dos testes de modo a torná-los eficaz e legível para todos os envolvidos do projeto, fornecendo os artefatos para a execução manual dos testes de requisitos de negócio e mecanismos para o cliente validar se o produto de software satisfaz suas necessidades ao término de cada iteração.

A proposta de adaptação do ATDD tem como finalidade privilegiar a adoção dos testes de aceitação no projeto de desenvolvimento de software, de modo a mitigar os riscos de identificar defeitos no software quando implantado em produção, que além de não atender às expectativas do cliente podem causar impactos negativos ao negócio. Além disso, o ATDD proposto deve oferecer uma alternativa à automação dos testes de aceitação.

4. ADAPTAÇÃO DO ACCEPTANCE TDD

Introduzir testes de aceitação executáveis (*Executable Acceptance Tests*) no projeto de software é um grande desafio, principalmente, se a equipe técnica não possui experiência consolidada, pois a automação dos testes requer habilidades, tempo e esforço para escrever e manter os testes. Com a ausência de uma equipe disciplinada, os testes podem perder o sincronismo com o código da aplicação se a equipe não mantiver apropriadamente os artefatos de testes (MARTIN e MELNIK, 2008. p.57. ; PARK e MAURER, 2008. p.20.).

Os testes de aceitação executáveis podem ajudar a comunicar os requisitos, desempenhando um importante papel no ciclo de desenvolvimento de software porque todos na equipe do projeto de software estão envolvidos de alguma forma na elaboração, no desenvolvimento, e na manutenção dos testes. Logo, existe a confiança do cliente que os requisitos são compreendidos corretamente (PARK e MAURER, 2008. p.19.).

Embora a automação dos testes de aceitação exija disciplina, experiência e recursos financeiros que podem elevar os custos do projeto, e até mesmo inviabilizar os testes, simplesmente ignorar o nível de teste aceitação e inserir o produto de software no mercado pode implicar em riscos para a imagem de uma organização.

Independente da adoção dos níveis de testes que antecedem os testes de aceitação, dada a natureza da tecnologia, o domínio de negócio, entre outros aspectos, é provável que o novo software ainda contenha problemas inesperados (PUGH, 2011. p.89.).

As implicações de negligenciar ou subestimar a importância dos testes de aceitação podem variar de uma pequena inconveniência a uma paralisação total quando o software está em produção.

Para a credibilidade do produto de software é melhor identificar e corrigir os defeitos durante os testes de aceitação, antes da implantação do sistema em produção, do que identificá-los tardiamente quando o tempo necessário

para corrigi-los é mais escasso. Além disso, os custos envolvidos na identificação de um defeito em ambiente produtivo podem ser maiores e fora do âmbito do orçamento original. Para tornar as coisas ainda piores, o sistema pode ter sido formalmente entregue e as obrigações contratuais finalizadas. Logo, qualquer modificação custará mais dinheiro.

Portanto, os efeitos de uma falha em um software que não atende às expectativas do cliente, normas ou funcionalidade podem causar impactos negativos ao negócio de uma organização.

A proposta do presente capítulo é preservar os aspectos relevantes do ATDD no projeto de desenvolvimento de software. Porém, a etapa de automação dos testes de aceitação é substituída pelo planejamento e organização dos testes de modo que sejam fornecidos subsídios à execução manual dos testes de aceitação. Esta proposta oferece uma alternativa à adoção dos testes de aceitação no âmbito do desenvolvimento ágil de software dadas as dificuldades em implantar os testes de aceitação executáveis.

Uma adaptação do ATDD é apresentada neste capítulo, descrevendo as atividades e os produtos que permeiam todas as etapas desta proposta.

4.1 Papéis e Responsabilidades

Antes de apresentar a proposta de adaptação do ATDD é importante definir claramente os papéis e responsabilidades dos *stakeholders* de modo a estabelecer o foco de atuação, as autoridades e os limites de cada individuo atuante no projeto de desenvolvimento de software, em particular, na proposta do ATDD descrita neste capítulo.

Pugh (2011. p.3.) faz referência aos papéis essenciais para a viabilidade do ATDD como a tríade, composta por: desenvolvedor, testador e cliente. Koskela (2007. p.329.) também menciona estes três papéis na atuação da técnica do ATDD, mas ambos não descrevem em detalhes as responsabilidades de cada papel nas etapas que compreendem o ATDD.

Na prática existe uma série de atividades de testes que devem ser realizadas na aplicação do ATDD em um projeto de desenvolvimento de software. Certamente, um indivíduo pode atuar em mais de um papel, porém dependendo da dimensão do projeto torna-se inviável que um único indivíduo tenha condições de absorver e exercer com excelência inúmeras responsabilidades.

Esta proposta de adaptação do ATDD sugere empregar quatro equipes com responsabilidades específicas. A organização das equipes é ilustrada na Figura 11.

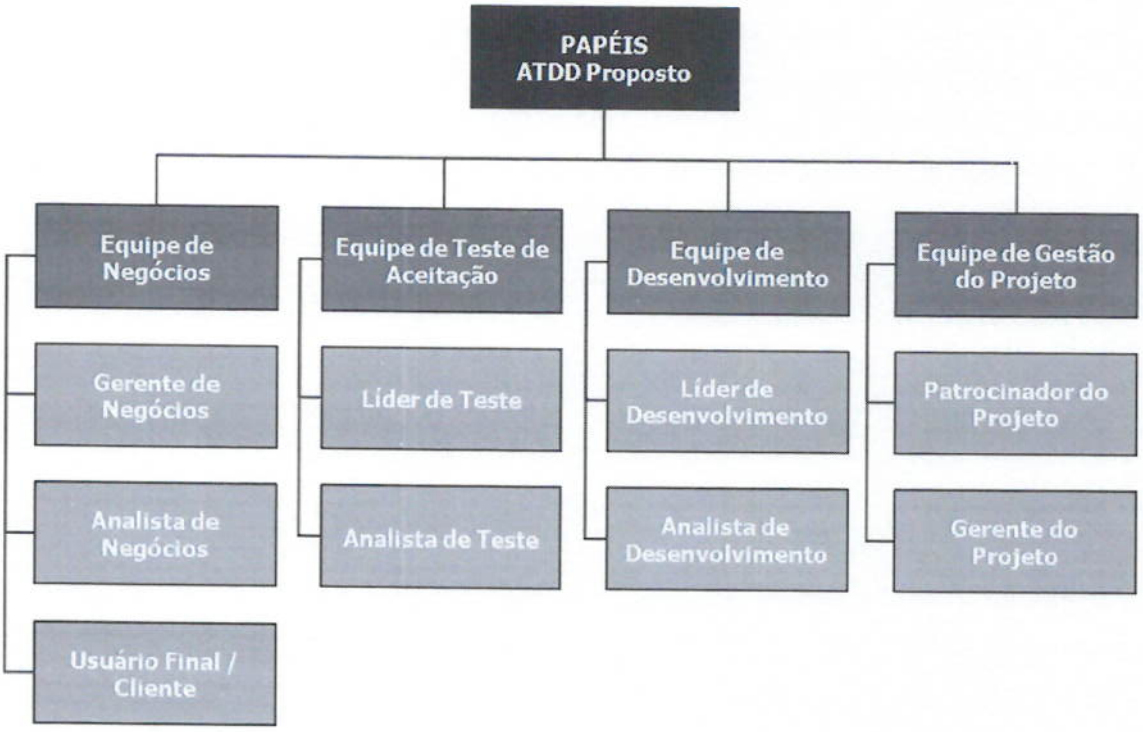

Figura 11 - Papéis da Proposta de Adaptação do ATDD

Essencialmente, as equipes mais atuantes na proposta do ATDD é a equipe de negócios, que representa os interesses do cliente, e a equipe técnica, composta pela equipe de testes de aceitação e desenvolvimento. A equipe de gestão do projeto tem participação nas etapas inicias, onde as definições de negócios são estabelecidas, e na etapa de encerramento do processo, quando é necessário conceder a aceitação formal do sistema.

Desta forma, a essência dos papéis no ATDD, cliente, desenvolvedor e testar, é preservada como prega a técnica. Aliás, o foco desta proposta não é modificar a atuação dos papéis e suas responsabilidades. Entretanto, o intuito de criar grupos e perfis com responsabilidades específicas é evidenciar as atividades de cada grupo que é composto por diferentes perfis dentro da proposta de adaptação do ATDD.

Os papéis e as principais responsabilidades de cada grupo são descritas a seguir (KRUCHTEN, 2000 ; PMI, 2004).

4.1.1 Equipe de Negócios

A equipe de negócio representa os interesses do cliente no projeto. Esta equipe é composta pelo gerente de negócios, analista de negócio e o usuário final / cliente, que devem ter amplo conhecimento do domínio de negócio. As responsabilidades destes papéis são relacionadas a seguir.

4.1.1.1 Gerente de Negócios

Responsável pela gerência e modelagem dos processos de negócio do projeto.

A Tabela 3 relaciona as principais responsabilidades do Gerente de Negócios (KRUCHTEN, 2000 ; PMI, 2004).

Tabela 3 - Principais Responsabilidades do Gerente de Negócios

Principais Responsabilidades
Atuar no entendimento do negócio e suas regras (necessidades dos clientes)
Definir e modelar os requisitos de negócio
Estimar o esforço do projeto
Colaborar na elaboração do Plano de Teste de Aceitação e aprová-lo
Acompanhar as atividades do projeto (realização dos Casos e *Scripts* de Testes, desenvolvimento das funcionalidades e correção dos defeitos, etc.), assegurando a integridade com os requisitos especificados
Conduzir a negociação com o cliente em eventuais mudanças de escopo do projeto
Ser mediador com os líderes de Teste e Desenvolvimento
Revisar e validar os entregáveis do projeto
Recomendar a aceitação do sistema a Equipe de Gestão do Projeto
Aceitar formalmente o novo sistema e recomendar sua implantação

4.1.1.2 Analista de Negócios

Responsável por prospectar oportunidades de negócio, criar novos produtos e soluções. Detém amplo conhecimento sobre um determinado ramo de negócio.

A Tabela 4 relaciona as principais responsabilidades do Analista de Negócios (KRUCHTEN, 2000).

Tabela 4 - Principais Responsabilidades do Analista de Negócios

Principais Responsabilidades
Colaborar na elaboração do Plano de Teste de Aceitação
Levantar os requisitos e apoiar na análise e projeto do sistema
Apoiar na preparação dos artefatos de testes
Coordenar a utilização de dados de teste
Acompanhar o desenvolvimento das funcionalidades e correção dos defeitos, assegurando a integridade com os requisitos especificados
Validar os entregáveis do projeto
Apoiar o Gerente de Negócios

4.1.1.3 Usuário Final / Cliente

Responsável por expressar com clareza as suas necessidades e expectativas sobre o comportamento do sistema, escrever os testes e critérios de aceitação, além de realizar os testes de aceitação do sistema.

A Tabela 5 relaciona as principais responsabilidades do Usuário Final / Cliente (KRUCHTEN, 2000).

Tabela 5 - Principais Responsabilidades do Usuário Final/ Cliente

Principais Responsabilidades
Colaborar na elaboração do Plano de Teste de Aceitação
Preparar os artefatos de testes (Casos e *Scripts* de Testes, critérios de aceitação, etc.)
Fornecer os dados de teste
Realizar os testes de aceitação de acordo com os artefatos de testes
Registrar os resultados dos testes e validá-los em conformidade com os critérios de aceitação
Documentar as ocorrências de testes
Reportar a conclusão dos testes

4.1.2 Equipe de Teste de Aceitação

Responsável por planejar, elaborar e manter os artefatos de testes, assim como apoiar os *stakeholders* do projeto na construção de um produto de software de qualidade. A equipe de teste de aceitação é composta pelo líder (ou gerente) de teste e o analista de teste. As responsabilidades destes papéis são relacionadas a seguir.

4.1.2.1 Líder de Teste

Responsável por liderar e gerenciar o projeto de teste de aceitação.

A Tabela 6 relaciona as principais responsabilidades do Líder de Teste (KRUCHTEN, 2000).

Tabela 6 - Principais Responsabilidades do Lider de Teste

Principais Responsabilidades
Preparar o Plano de Teste de Aceitação e obter os recursos necessários para a viabilidade dos testes de aceitação
Prover o treinamento da equipe responsável pelos testes de aceitação
Atribuir as tarefas à equipe de teste de aceitação
Supervisionar a preparação dos Casos de Teste e *Scripts* de Testes, com base nos requisitos de negócio
Gerenciar a configuração dos artefatos de testes
Reportar formalmente através de relatórios ao Gerente de Projeto e Negócios a evolução dos testes de aceitação
Gerir e assegurar a comunicação entre os *stakeholders* sobre as solicitações de mudanças e ocorrências de teste
Revisar os resultados dos testes
Certificar que os testes são repetidos sempre que necessário
Determinar a suspensão ou o cancelamento do teste na impossibilidade de prosseguir com os testes
Certificar que os testes são concluídos dentro do cronograma acordado
Gerir os relatórios de ocorrências de testes e recomendar prioridades de resolução
Solicitar a aceitação formal do sistema ao Analista ou Gerente de Negócios

4.1.2.2 Analista de Teste

Responsável por apoiar o usuário final /cliente na modelagem e realização dos Casos de Teste e *Scripts* de testes.

A Tabela 7 relaciona as principais responsabilidades do Analista de Teste (KRUCHTEN, 2000).

Tabela 7 - Principais Responsabilidades do Analista de Teste

Principais Responsabilidades
Colaborar na elaboração do Plano de Teste de Aceitação
Apoiar os usuários finais / clientes na preparação dos artefatos de testes
Apoiar os usuários finais /clientes na realização dos testes de aceitação
Reportar ao Líder de Teste a identificação de defeito com severidade crítica para tomada de decisão
Apoiar o Líder de Teste em suas atividades

4.1.3 Equipe de Desenvolvimento

Responsável por implementar o produto de software e gerenciar os aspectos do seu desenvolvimento. A equipe de desenvolvimento é composta pelo líder (ou gerente) de desenvolvimento e o analista de desenvolvimento. As responsabilidades destes papéis são relacionadas a seguir.

4.1.3.1 Líder de Desenvolvimento

Responsável por liderar e gerenciar o desenvolvimento do produto de software, mantendo a equipe unida, coesa e motivada. Apóia na solução dos problemas técnicos e na ausência de motivação, que são comuns em projetos complexos e com prazos desafiadores.

A Tabela 8 relaciona as principais responsabilidades do Líder de Desenvolvimento (KRUCHTEN, 2000).

Tabela 8 - Principais Responsabilidades do Líder de Desenvolvimento

Principais Responsabilidades
Colaborar na elaboração do Plano de Teste de Aceitação
Supervisionar a preparação do ambiente de teste de aceitação
Supervisionar a implementação do sistema
Assegurar que os entregáveis relacionados aos testes de aceitação são atendidos, conforme compromisso prévio
Gerenciar as versões do sistema
Mediador com os líderes de Negócios, Testes e Projeto

4.1.3.2 Analista de Desenvolvimento

Responsável por elaborar, implementar e manter um sistema computacional, transformando as necessidades do cliente em um produto de software. O analista de desenvolvimento é também conhecido como desenvolvedor ou programador.

A Tabela 9 relaciona as principais responsabilidades do Analista de Desenvolvimento (KRUCHTEN, 2000).

Tabela 9 - Principais Responsabilidades do Analista de Desenvolvimento

Principais Responsabilidades
Colaborar na elaboração do Plano de Teste de Aceitação
Implementar as funções aderentes aos requisitos de negócio e os critérios de aceitação
Realizar os testes de unidade antes de entregar o software
Estabelecer e manter o ambiente de Teste de Aceitação
Prover todos os requisitos de teste específicos
Definir a prioridade de solução dos defeitos
Solucionar os defeitos e liberar as versões atualizadas
Assegurar o estabelecimento dos parâmetros da aplicação
Migrar a aplicação para o ambiente apropriado de teste (ambiente de homologação / pré-produção)
Apoiar os usuários finais com os testes necessários

4.1.4 Equipe de Gestão do Projeto

Representa as maiores autoridades do projeto. A equipe de gestão do projeto é composta pelo patrocinador e o gerente do projeto. As responsabilidades destes papéis são relacionadas a seguir.

4.1.4.1 Patrocinador do Projeto

Responsável por fornecer os recursos financeiros e promover o projeto. A participação do Patrocinador é mais ativa nas fases iniciais do projeto. Este indivíduo tem o maior grau de autoridade no projeto.

A Tabela 10 relaciona as principais responsabilidades do Patrocinador do Projeto (PMI, 2004).

Tabela 10 - Principais Responsabilidades do Patrocinador do Projeto

Principais Responsabilidades
Definir os requisitos do negócio e os respectivos benefícios para o projeto
Estabelecer objetivos estratégicos para o projeto
Definir / identificar fatores críticos de sucesso
Assegurar que os recursos adequados (equipe, tecnologia, equipamentos, etc.) são disponibilizados para o Gerente do Projeto
Assegurar os treinamentos necessários à equipe do projeto para garantir o cumprimento dos objetivos
Monitorar continuamente o ambiente de negócios do projeto
Resolver conflitos e tomar a decisões no projeto
Atuar como executivo de contato com o cliente
Aprovar o Plano de Teste de Aceitação
Conceder a aceitação formal do sistema

4.1.4.2 Gerente do Projeto

Responsável por gerenciar o progresso do projeto através das variáveis, qualidade, custo e prazo, e verificar seus desvios. O foco do Gerente de Projeto é minimizar as falhas inerentes aos processos e garantir a qualidade dos produtos do projeto.

A Tabela 11 relaciona as principais responsabilidades do Gerente do Projeto (KRUCHTEN, 2000 ; PMI, 2004).

Tabela 11 - Principais Responsabilidades do Gerente de Projeto

Principais Responsabilidades
Definir e controlar os requisitos do produto
Definir e controlar os riscos do projeto
Definir e avaliar os fatores críticos de sucesso do projeto
Definir e controlar o cronograma
Alocar e gerenciar recursos
Coordenar interações entre os *stakeholders* do projeto
Assegurar que os prazos são mantidos dentro do planejado
Assegurar a qualidade dos produtos do projeto (aderente aos critérios de qualidade e padrões estabelecidos)
Aprovar o Plano de Teste de Aceitação
Elaborar relatórios de acompanhamento do projeto
Participar de reuniões de acompanhamento e de revisão do projeto
Conceder a aceitação formal do sistema

Um indivíduo pode assumir mais de um papel e/ou responsabilidades.

Os papéis e responsabilidades para os testes de aceitação devem estar claramente definidos e descritos no Plano de Teste de Aceitação.

4.2 Proposta de Adaptação do ATDD

A proposta de adaptação do ATDD descrita nesta seção tem como finalidade preservar os aspectos relevantes do ATDD, como a comunicação e a efetiva colaboração entre os *stakeholders*, para desenvolver um produto de software aderente as reais expectativas do cliente.

Esta proposta sugere a substituição da etapa de automação dos testes, onde os testes de aceitação são transformados em testes executáveis através de ferramentas específicas, pelo planejamento e organização dos testes cujos elementos essenciais são a documentação e sistematização dos testes. Os produtos de trabalho produzidos oferecem subsídios para a execução manual dos testes de aceitação.

Portanto, para suprir a lacuna da automação dos testes são introduzidas três etapas ao processo original de Hendrickson (2008. p.3.), destacadas em vermelho na Figura 12. A etapa Encerrar foi introduzida como um complemento a etapa Demonstrar, os detalhes destas etapas são descritos na seção 4.2.8.

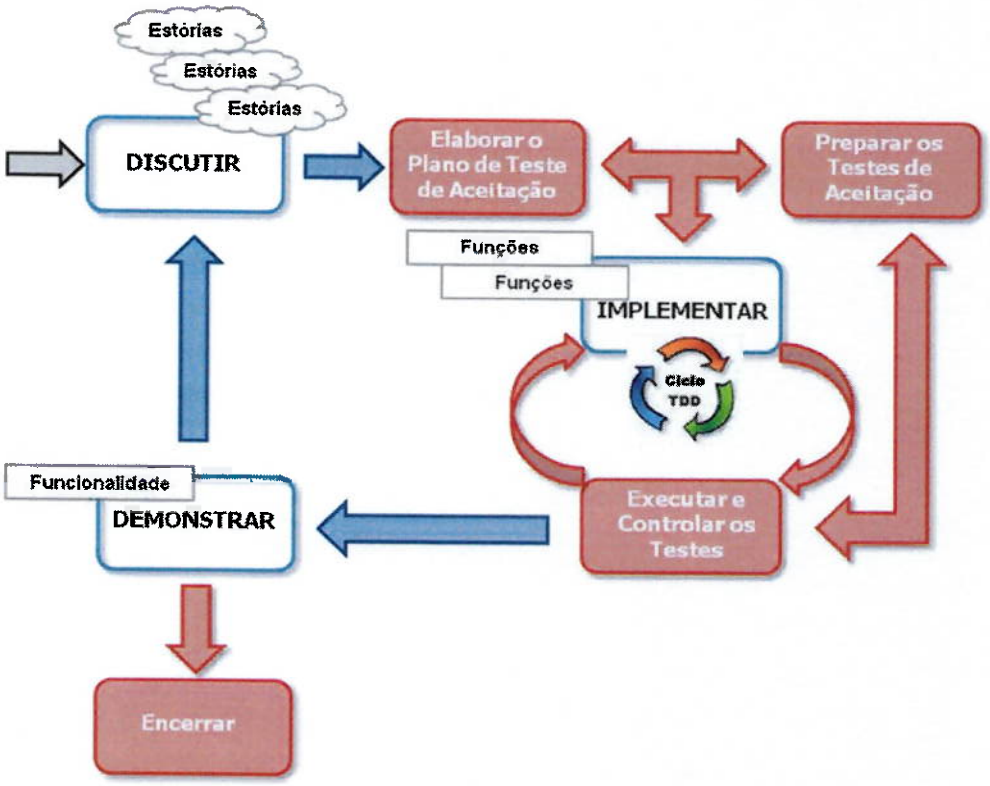

Figura 12 - Proposta de Adaptação do ATDD (adaptado de HENDRICKSON, 2008, p.3.)

Esta proposta de adaptação do ATDD substitui a etapa de automação dos testes de aceitação por um conjunto de etapas que forneçam subsídios à execução manual destes testes. Com isso, a sobrecarga de atribuições sobre o desenvolvedor é minimizada, uma vez que ele não tem a necessidade de desenvolver e manter um código de teste para validar o comportamento em nível de teste de aceitação de toda e qualquer funcionalidade implementada no código da aplicação. Desta forma, todo o esforço do desenvolvedor é concentrado em implementar a funcionalidade de valor ao cliente e as validações que lhe competem ficam direcionadas, a princípio, aos testes de unidade e integração destas unidades de acordo com a técnica do TDD.

A documentação proveniente desta proposta não deve burocratizar o processo de desenvolvimento do software. Esta documentação deve ser mínima e suficiente para especificar os testes de aceitação cujo objetivo é esclarecer os requisitos e objetivos de negócios. Logo, informações fora deste contexto são irrelevantes e não devem estar presentes nos produtos de testes.

No processo de testes de aceitação existem etapas essenciais que devem ser consideradas, não importa a complexidade do projeto de desenvolvimento de software. As etapas fundamentais dos testes de aceitação que compreendem a proposta de adaptação do ATDD são apresentadas individualmente a seguir, assim como os insumos e produtos das respectivas etapas.

4.2.1 Discutir

A primeira etapa do processo mantém as características descritas no capítulo anterior, onde durante a reunião de planejamento da iteração são discutidas as estórias dos usuários. Neste encontro é necessária a presença dos *stakeholders* do projeto que possuem informações relevantes sobre os requisitos e aspectos do sistema.

Os clientes expressam suas necessidades e expectativas através de exemplos concretos de funções, ilustrando como o sistema deve funcionar para cada situação, e a equipe de desenvolvimento e teste, em particular os

desenvolvedores e os analistas de testes, fazem perguntas para esclarecer e alinhar os entendimentos, tornando um meio de intenso aprendizado e discussão. As equipes de negócio e projeto, também participam desta discussão.

As estórias do usuário são revisadas, aprimoradas e desdobradas em estórias menores, o que permite torná-las independentes e simplificar a atuação dos desenvolvedores.

Os cenários de testes e critérios de aceitação são definidos para cada estória e as informações ficam documentadas em cartões de estória, onde o cliente prioriza as estórias pelo seu valor de negócio, incluindo o risco do negócio. O risco técnico é estimado pela equipe de desenvolvimento.

Esta etapa do processo é responsável por fornecer os insumos e produtos relacionados a seguir e ilustrado na Figura 13.

INSUMOS	PRODUTOS
• Estórias de Usuário	• Estórias Revisadas • Cenários de Testes • Critérios de Aceitação

Figura 13 - Etapa Discutir do ATDD Proposto

Nesta etapa do processo os seguintes pápeis estão envolvidos.

PAPEIS ENVOLVIDOS
• Gerente de Negócio • Analista de Negócio • Usuário Final / Cliente • Líder de Teste • Analista de Teste • Lider de Desenvolvimento • Analista de Desenvolvimento • Patrocinador do Projeto • Gerente do Projeto

4.2.2 Elaborar o Plano de Teste de Aceitação

A segunda etapa do processo tem alguma semelhança com o processo de ATDD descrito por Koskela (2007), onde ele sugere que nesta etapa os testes de aceitação devem ser escritos pelo cliente com a colaboração dos testadores e desenvolvedores. Porém, como citado na etapa anterior (Discutir), os aspectos relevantes aos testes de aceitação foram previamente esboçados para cada estória do usuário. Logo, a proposta desta etapa do processo é documentar e organizar os produtos de trabalho da etapa anterior (as estórias, cenários de testes e critérios de aceitação) em um Plano de Teste de Aceitação.

O intuíto é organizar os artefatos gerados inicialmente de modo que possam ser melhorados continuamente durante o processo de desenvolvimento do sistema, e forneça uma documentação formal especificando os requisitos e aspectos de negócios ao término da implementação, além de gerenciar as necessidades e expectativas dos clientes.

Esta etapa do processo é responsável por fornecer os insumos e produtos relacionados a seguir e ilustrado na Figura 14.

INSUMOS	PRODUTOS
• Estórias Revisadas • Cenários de Testes • Critérios de Aceitação	• Plano de Teste de Aceitação (Inicial) • *Sign-off* do Plano de Teste de Aceitação

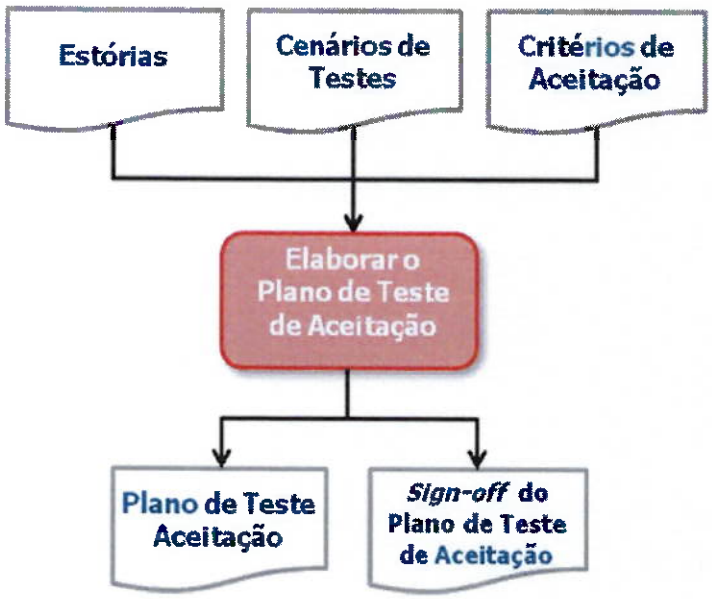

Figura 14 - Etapa Elaborar o Plano de Teste de Aceitação do ATDD Proposto

Um breve descritivo dos produtos fornecidos por esta etapa é descrito a seguir.

4.2.2.1 Plano de Teste de Aceitação

O Plano de Teste de Aceitação não deve burocratizar o processo de desenvolvimento do produto de software. Logo, o conteúdo deste documento deve privilegiar os artefatos dos testes de aceitação, esclarecendo os requisitos e objetivos de negócios, e os aspectos relevantes aos testes de aceitação. Sendo assim, não deve ser um documento extenso e conter informações irrelevantes para a validação das funções do sistema.

A responsabilidade pela elaboração e manutenção do Plano de Teste de Aceitação é da Equipe de Teste de Aceitação, cuja atribuição é do Líder de Teste, que juntamente com as equipes de Negócios e Desenvolvimento devem verificar e ratificar os entendimentos obtidos inicialmente.

O Plano de Teste de Aceitação deve ser concebido em um curto espaço de tempo. Portanto, algumas horas ou poucos dias, de acordo com dimensão

do projeto, devem ser consumidos na concepção da primeira versão deste documento, o qual é nomeado como Plano de Teste de Aceitação inicial. Isto porque este documento deverá ser aprimorado continuamente durante as iterações.

4.2.2.2 *Sign-off* do Plano de Teste de Aceitação

Após elaborar o Plano de Teste de Aceitação inicial os *stakeholders* devem se reunir para validar e criticar o documento. Se alguma divergência for identificada, a ocorrência é discutida até que todos tenham o mesmo entendimento e as expectativas sejam alinhadas. Em seguida, o *sign-off* do Plano de Teste de Aceitação inicial deve ser concedido pelos líderes das equipes do projeto, e a comunicação deste documento deve contemplar todos os *stakeholders* do projeto.

Para assegurar a uniformidade e sincronismo da informação é essencial que toda e qualquer mudança de requisito seja contemplada prontamente no Plano de Teste de Aceitação e comunicada a todos os *stakeholders* do projeto.

Nesta etapa do processo os seguintes pápeis estão envolvidos.

PAPEIS ENVOLVIDOS
• Gerente de Negócio • Analista de Negócio • Usuário Final / Cliente • Líder de Teste • Analista de Teste • Lider de Desenvolvimento • Analista de Desenvolvimento • Patrocinador do Projeto • Gerente do Projeto

Após a concepção do Plano de Teste de Aceitação inicial, duas etapas são realizadas simultaneamente, a implementação da funcionalidade, com o uso do TDD, e a preparação dos artefatos de testes (casos de testes, *scripts* de

teste, dados de testes, etc.) que serão utilizados na execução manual dos testes de aceitação.

Os detalhes dessas etapas são descritos a seguir.

4.2.3 Preparar os Testes de Aceitação

Em projeto de desenvolvimento de software, a preparação dos testes está inserida no contexto do ambiente de teste, que representa toda a infraestrutura onde o teste será executado, o que compreende as configurações de hardware, software, ferramentas, suprimentos, rede, equipe envolvida, aspectos organizacionais e documentação.

O escopo desta proposta, no que diz respeito ao ambiente de testes, tem seu foco na documentação dos testes de aceitação e equipe envolvida nas atividades de testes. Logo, os demais aspectos de ambiente não serão detalhados.

O objetivo fundamental desta etapa do processo é preparar os artefatos de testes de aceitação com a finalidade produzir subsídios a realização manual dos testes de aceitação em condições conhecidas e controladas.

O responsável por elaborar os testes de aceitação é o cliente, como preza o ATDD. No entanto, o analista de teste deve estar próximo do cliente e apoiá-lo permanentemente na preparação dos artefatos de testes. Para promover agilidade na preparação dos artefatos de testes, o analista de teste também poderá escrever os testes de aceitação, desde que tenha o aval do cliente. Além disso, todo artefato produzido pelo analista de teste deve, obrigatoriamente, ser validado pelo cliente antes de considerá-lo concluído.

Esta etapa do processo é responsável por fornecer os insumos e produtos relacionados a seguir e ilustrado na Figura 15.

INSUMOS	PRODUTOS
• Plano de Teste de Aceitação (Inicial / Signed)	• Casos de Testes • Dados de Testes • *Scripts* de Testes • Plano de Teste de Aceitação (Atualizado)

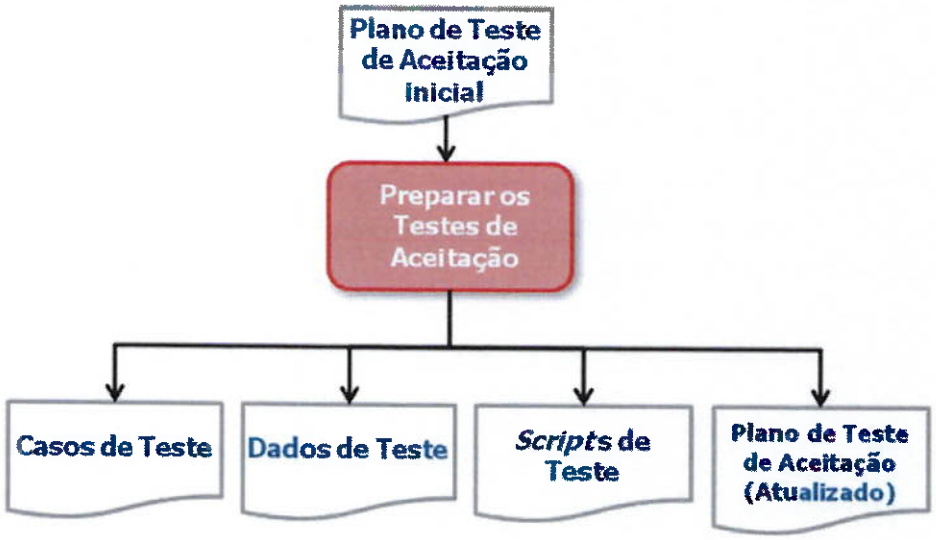

Figura 15 - Etapa Preparar os Testes de Aceitação do ATDD Proposto

Um breve descritivo dos produtos de testes fornecidos por esta etapa é descrito a seguir.

4.2.3.1 Casos de Testes

O Plano de Teste de Aceitação inicial, que contempla a prévia das estórias de usuário, cenários de testes e critérios de aceitação, fornece a base de conhecimento para produzir os testes.

Então, é preciso definir um conjunto específico de entrada de dados, condições de execução e resultados esperados, produzindo o artefato conhecido como Casos de Teste, cuja finalidade é avaliar os requisitos especificados do sistema.

No geral, a elaboração dos Casos de Teste compreende as etapas ilustradas na Figura 16.

Figura 16 - Etapas de Elaboração dos Casos de Testes

Os Casos de Testes devem verificar não somente as condições válidas de execução, como também as condições inválidas. Por esta razão é fundamental a colaboração entre os diversos *stakeholders* do projeto, pois cada área de conhecimento tem uma perspectiva particular do sistema. Por exemplo, a equipe de teste tem uma visão destrutiva do sistema, em busca de defeitos, enquanto a equipe de desenvolvimento tem uma visão construtiva, em implementar as funções do sistema, já a equipe de negócios tem um amplo conhecimento dos aspectos de negócio e a sua respectiva operação.

4.2.3.2 Dados de Teste

Preparar os dados de teste pode ser um processo não trivial, uma vez que a realização dos testes de aceitação exigem dados reais de produção ou dados que simulam rigorosamente a operação de produção, para que a probabilidade de incidentes no software quando implantado seja reduzido consideravelmente.

O uso de dados "reais" pode significar um risco maior de distribuição acidental de informações confidenciais. Por exemplo, uma informação impressa, enviada por email, transferida em uma rede pública ou em toda a área de trabalho, etc. Para minimizar este risco, é necessário certificar que todos os *stakeholders*, principalmente, os indivíduos envolvidos nos testes estão orientados e em alerta para com esses riscos. Outras formas de manter a segurança dos dados de testes de aceitação incluem:

- Estabelecer uma rede fisicamente separada, assim como impressoras.
- Codificar ou modificar dados confidenciais como nomes e endereços, tornando-os irreconhecíveis.
- Estabelecer servidor e diretórios específicos para os testes.
- Estabelecer contas de e-mail específico à equipe de teste.
- Limpar regularmente os diretórios com informações sigilosas.
- Retalhar regularmente os relatórios impressos.

4.2.3.3 *Scripts* de Teste

Uma vez que os Dados de Testes e Casos de Teste estão preparados é preciso identificar a necessidade de agrupá-los conforme o seu relacionamento de dependência, e preparar os *Scripts* de Teste, também conhecido como Roteiros de Teste, que descrevem os procedimentos e dados de testes necessários para a execução dos testes de aceitação. O nível de detalhamento dos *Scripts* de Teste tem influência direta com o perfil de quem os produziu, assim a colaboração do cliente é um parâmetro importante para determinar o quão detalhados deverão ser os procedimentos de teste.

Geralmente, cada *Script* de Teste equivale a um processo de negócio e representa uma seqüência lógica, incluindo os caminhos normais e alternativos, e as exceções. Além de fornecer instruções detalhadas dos objetivos de cada coleção de Casos de Teste e configuração do sistema em uso.

4.2.3.4 Plano de Teste de Aceitação Atualizado

Durante o desenvolvimento da etapa de Preparar os Testes de Aceitação, a implementação da funcionalidade está ocorrendo em paralelo. Uma vez que

o desenvolvedor não atua na implementação do código de teste para um *framework*, pois esta proposta não faz uso da automação dos testes, e dedica-se exclusivamente na implementação da funcionalidade do produto de software, é importante que a comunicação seja constante entre as equipes para alinhar quaisquer alterações nos cenários de testes ou critérios de aceitação, ou ainda a identificação de uma condição não prevista inicialmente e que o sistema deverá lidar.

Desta forma, os Gerentes de Projeto e Negócios, e os Líderes de Desenvolvimento e Testes são responsáveis por comunicar uns aos outros sobre toda e qualquer alteração ou previsibilidade de alteração dos cenários de testes e/ou critérios de aceitação.

Além disso, a medida que os artefatos de testes são produzidos, o Líder de Teste pode disponibilizá-los à equipe de desenvolvimento para certificar que eles estão no caminho certo e, principalmente, para motivá-los a validar antecipadamente ("pré-homologação") se a funcionalidade desenvolvida está de acordo com as expectativas do cliente. Esta prática garante a integridade da funcionalidade e minimiza imprevistos desagradáveis na execução dos testes de aceitação pelo cliente.

Ao término desta etapa, o Plano de Teste de Aceitação deve ser atualizado com os produtos de testes previstos nesta etapa para a execução dos testes de aceitação pertinentes à primeira iteração. Este processo deve se repetir continuamente durante o projeto de desenvolvimento do software conforme a introdução de novas necessidades de negócio (estórias / requisitos). Logo, a etapa de Preparar Testes de Aceitação deverá permanecer ativa e fornecer os produtos de testes com qualidade até que o projeto seja concluído.

4.2.4 Implementar

O Plano de Teste de Aceitação inicial, que contempla a prévia das estórias de usuário, cenários de testes e critérios de aceitação, fornece a base

de conhecimento para a implementação das estórias priorizadas pelo cliente para cada iteração.

O ATDD não determina uma técnica de implementação para a funcionalidade, mas considera como uma melhor prática o uso do TDD.

A equipe de desenvolvimento, em particular, os desenvolvedores são os responsáveis por implementar a funcionalidade, e nesta proposta a principal atribuição dos desenvolvedores é implementar o código da aplicação, uma vez que não há a necessidade de implementar e manter os códigos de testes.

No entanto, uma questão que pode vir à tona é se a ausência do desenvolvedor na implementação do código de teste pode tornar vulnerável o seu entendimento sobre os testes e, por conseqüência, as expectativas do cliente sobre o comportamento do sistema. Para minimizar esta possível lacuna, o desenvolvedor deve ter acesso freqüente ao cliente, que deve ser consultado para esclarecer as eventuais questões sobre como as funções (estórias) devem se comportar. Além disso, esta proposta recomenda que os artefatos de testes produzidos durante a etapa Preparar os Testes de Aceitação sejam disponibilizados, sempre que possível, previamente à equipe de desenvolvimento, de acordo com a evolução da preparação dos artefatos de testes. Isto significa que, enquanto as etapas Implementar e Preparar os Testes de Aceitação ocorrem em paralelo, conforme a equipe de teste de aceitação for finalizando um conjunto de artefatos de testes, ela deverá disponibilizá-lo à equipe de desenvolvimento de modo que possa lhe oferecer algum valor. Com isto, não é necessário aguardar a conclusão dos artefatos de testes previstos para a iteração, ou seja, os desenvolvedores poderão ter acesso aos artefatos de testes parciais que irão compor o *baseline* oficial dos testes de aceitação.

O objetivo de antecipar os artefatos de testes de aceitação aos desenvolvedores é comunicar e ratificar os entendimentos sobre os requisitos de negócio e, consequentemente, manter atualizado todos os *stakeholders* envolvidos com os testes. Além disso, esta ação deve encorajar os desenvolvedores a validar as funções, em nível de teste de aceitação, antes de entregar a funcionalidade para a validação do cliente. Com isso, é possível

minimizar a probabilidade de defeitos ou qualquer evento indesejável durante os testes de aceitação oficial pelo cliente.

Esta etapa do processo é responsável por fornecer os insumos e produtos relacionados a seguir e ilustrado na Figura 17.

INSUMOS	PRODUTOS
• Plano de Teste de Aceitação (inicial) • Artefatos Preliminares de Testes	• Funções previstas na iteração • Evidências dos testes de unidade

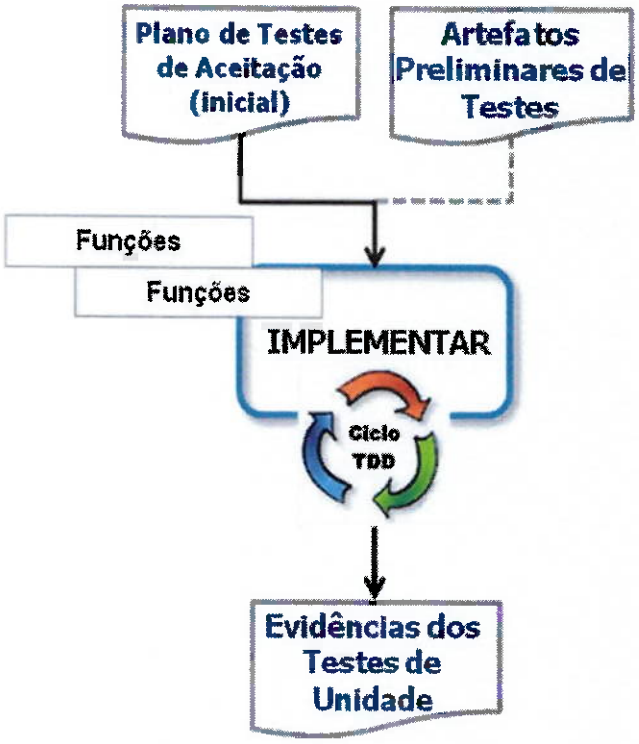

Figura 17 - Etapa Implementar do ATDD Proposto

4.2.4.1 Evidências dos Testes de Unidade

Os testes de unidade focalizam na menor parte do software, testando suas unidades individuais: componentes, objetos e funções.

Os testes de unidade são realizados pelos desenvolvedores que implementam os testes para as suas próprias rotinas.

As evidências dos testes de unidade devem ser documentadas para todos os testes, sejam eles bem sucedidos ou não. Assim, os desenvolvedores proveem subsídios para promover as correções necessárias no código da aplicação. Além, de garantir, através das evidências, a integridade das funções implementadas.

Nesta etapa do processo os seguintes pápeis estão envolvidos.

PAPEIS ENVOLVIDOS
• Lider de Desenvolvimento
• Analista de Desenvolvimento

4.2.5 Executar e Controlar os Testes

A etapa Executar e Controlar os testes de aceitação devem ocorrer imediatamente após a implementação das funções previstas para a iteração. Mas, é importante ressaltar que esta etapa deve iniciar somente após a realização dos testes que antecedem o nível de teste de aceitação, ou seja, os testes de unidade, integração e sistema devem ter sido previamente realizados. Contudo, o sistema deve estar pronto para ser testado adequadamente no nível de teste de aceitação.

O sucesso da etapa de execução e controle dos testes dependerá de tudo o que foi realizado anteriormente, portanto, a qualidade dos artefatos de testes produzidos será determinante para o cumprimento desta etapa.

O cliente é responsável pela a execução manual dos testes de aceitação, e pode ser representado pelos usuários finais do sistema para esta atividade. Além disso, o analista de teste deverá presenciar a execução dos testes e oferecer apóio às eventuais ocorrências, como no relato dos defeitos pelos usuários finais e reportar de imediato ao Líder de Teste sobre os defeitos de severidade crítica que podem comprometer a continuidade dos testes, e assim, as devidas ações serem tomadas. Isto porque os defeitos que inibem a continuidade dos testes, o Líder de Teste (sob orientação do Gerente de

Projeto) poderá suspender os testes de aceitação até que o defeito em questão seja solucionado. Caso a solução não seja imediata, pode haver razões para o cancelamento de quaisquer outros testes. Os critérios para a suspensão ou cancelamento dos testes de aceitação devem ser documentados no Plano de Teste de Aceitação.

Os testes de aceitação devem ser realizados em conformidade com os Casos de Teste e *Scripts* de Teste previamente especificados, e os resultados apresentados pelo sistema são comparados com os resultados esperados (expectativa do cliente), então é atribuída uma severidade a esta ocorrência que deve ser reportada através de relatório aos *stakeholders* do projeto.

O controle desta etapa do processo é essencial para garantir que os testes planejados sejam executados corretamente e seus resultados possam ser registrados e monitorados constantemente. Possibilitando comunicar o progresso dos testes de aceitação e o estado dos defeitos pendentes. Além disso, o controle efetivo permite obter um maior conhecimento do processo de testes para aprimorá-lo continuamente.

O líder de Teste é responsável por monitorar permanentemente esta etapa do processo e comunicar o progresso e o estado dos testes de aceitação aos *stakeholders*. Portanto, é altamente recomendável o uso de ferramentas de gestão de defeitos cuja finalidade é controlar o ciclo de vida de um defeito. Esta ferramenta de gestão de defeitos pode ser manual, uma simples planilha, ou automatizada. Certamente, o uso de uma ferramenta de gestão de defeitos automatizada é mais apropriado, pois oferece uma base comum para a entrada de informações, além de fomentar a integração entre as equipes de desenvolvimento e testes. Além disso, por meio dos relatórios gerenciais e métricas obtidas por essas ferramentas, é possível promover a melhoria contínua do processo de testes.

Há uma variedade de ferramentas de gestão de defeitos, inclusive gratuita e *open source*, por exemplo, o Mantis (http://www.mantisbt.org/) e Bugzilla (http://www.bugzilla.org/), que além dos benefícios citados permite customizar soluções de acordo com as necessidades do projeto.

Importante, a proposta de adaptação do ATDD refere-se à execução manual dos testes, o que não significa que os controles e documentos de testes devam ser realizados de forma manual.

Esta etapa do processo é responsável por fornecer os insumos e produtos, conforme relacionado a seguir e ilustrado na Figura 18.

INSUMOS	PRODUTOS
• Casos de Testes • Dados de Testes • Scripts de Testes • Plano de Teste de Aceitação (Atualizado)	• Log de Testes • Relatório de Ocorrências • Resumo dos Testes

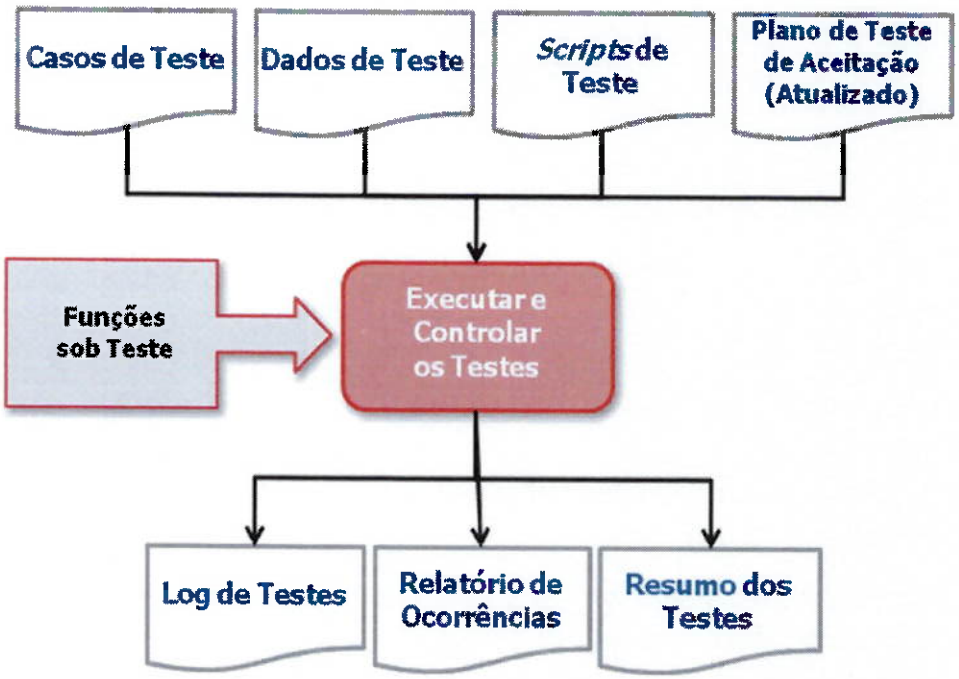

Figura 18 - Etapa Executar e Controlar os Testes do ATDD Proposto

Um breve descritivo dos produtos desta etapa é apresentado a seguir (BASTOS et al, 2007. p.211 - p.219).

4.2.5.1 *Log* de Testes

É importante estabelecer um registro de *log* durante a execução dos testes, isto é, um registro cronológico de todos os eventos significativos da execução dos testes. Além disso, o *log* de teste oferece uma ferramenta de apoio aos desenvolvedores para investigar as causas de defeitos identificados durante a execução dos testes.

O *log* de teste pode conter informações tais como:

- Identificador
- Descrição
  - Entradas das atividades e eventos
  - Descrição da execução (identificar o que foi executado)
  - Resultados (mensagens, requisições operacionais, etc.)
  - Informações sobre o ambiente (configuração, responsável pela execução do teste, etc.)
  - Eventos anormais (relacionar com o Relatório de Ocorrências)

4.2.5.2 Relatório de Ocorrências

O propósito do Relatório de Ocorrências é documentar qualquer evento inesperado identificado durante a execução dos testes que exija algum tipo de investigação e, posteriormente, poderá ser reconhecido como defeito. Este relatório também é conhecido como Relatório de Incidentes de Testes ou Defeitos.

Toda ocorrência deverá ter um Caso de Teste associado. Caso não exista, um Caso de Teste deverá ser escrito, e atualizado no Plano de Teste de Aceitação.

Um defeito que só pode ser solucionado através da adequação de um requisito de negócio também deve ser registrado e classificado para eventual controle das informações.

O Relatório de Ocorrências pode ser composto das seguintes informações:

- Identificador
- Sumário (Breve descrição da ocorrência)
- Descrição da ocorrência (defeito)
  - Entradas
  - Resultados esperados
  - Resultados encontrados
  - Anomalias
  - Data e hora
  - Passo do procedimento (referência ao *Script de Teste*)
  - Tentativas de repetição
  - Testadores envolvidos
  - Observações
- Impacto
  - Representa a severidade da ocorrência na continuidade do teste. Uma escala simples e praticada é: Crítica, Grave, Média e Leve.

Durante a execução dos testes de aceitação, o cliente ou usuário final do sistema, deve identificar os eventos inesperados, classificar a sua severidade de acordo com a sua percepção e relatar a ocorrência para a análise prévia do Líder ou Analista de Teste e, posteriormente, para o reconhecimento do defeito pelos desenvolvedores.

4.2.5.3 Resumo dos Testes

O propósito do Resumo dos Testes é fornecer um sumário dos resultados alcançados nos testes de aceitação tendo em vista um processo de melhoria dos testes. Este relatório deve cobrir todo o Plano de Teste de Aceitação e deve produzir alguns dos indicadores históricos do projeto.

O Resumo dos Testes pode conter os seguintes campos:

- **Identificador**

- **Resumo**

  Relacionar as funções testadas, as versões de software e documentação, e as características relevantes de ambiente de teste.

- **Variações**

  Registrar as discrepâncias em relação ao planejamento inicial (Plano de Teste de Aceitação) e descrever as razões que levaram a essa variação.

- **Avaliação Funcional**

  Identificar as funções que não foram suficientemente testadas.

- **Resumo dos Resultados**

  Número de Casos de Teste executados, defeitos identificados, tempo de execução para cada Caso de Teste, etc.

  - **Ocorrências Resolvidas**
  - **Ocorrências Não-Resolvidas**

- **Avaliação**

  Avaliação final do trabalho, apresentado as suas limitações, os possíveis riscos provenientes de falhas, etc.

- **Resumo de Atividades**

  Informações administrativas, como custo do teste (Hora x Homem), equipamentos, indicadores de testes para estimativas futuras, etc.

- **Aprovações**

  Especificar os responsáveis pela aprovação do relatório.

Nesta etapa do processo os seguintes pápeis estão envolvidos.

PAPEIS ENVOLVIDOS
• Usuário Final / Cliente • Analista de Teste • Líder de Teste

4.2.6 Demonstrar

A etapa Demonstrar deve determinar se os requisitos de negócios e os critérios de aceitação previstos para a iteração estão plenamente satisfatórios. Para isto, a Equipe de Testes de Aceitação, Líder e/ou Analista de Testes, deve apresentar ao Analista de Negócios todos os produtos da etapa Executar e Controlar dos testes (Resumo dos Testes, Relatório de Ocorrências e os Logs de Teste) para a apreciação e análise de conformidade aos requisitos e critérios de aceitação.

Além de verificar todas as evidências dos testes, o Analista de Negócios pode realizar alguns testes pontuais para validar as funções que representam maiores riscos ao negócio, além de realizar testes exploratórios para verificar se existem falhas nos critérios de aceitação e riscos que não foram identificados durante o processo.

Uma vez que todos os critérios de aceitação foram atendidos para as funções (estórias) previstas na iteração, se ainda houver novas estórias e iterações pendentes, deve-se iniciar a etapa Discutir novamente e o processo deve se repetir até que todas as estórias sejam implementadas e formalmente aceita pelo Analista de Negócios. Caso não existam novas estórias pendentes, o Analista de Negócios recomenda a aceitação formal do sistema ao Gerente de Negócios, que avalia o Resumo dos Testes, se existem defeitos críticos ao negócio e, posteriormente, deve solicitar a aceitação formal à equipe de Gestão do Projeto, conforme descrito na seção seguinte.

Esta etapa do processo é responsável por fornecer os insumos e produtos, conforme relacionado a seguir e ilustrado na Figura 19.

INSUMOS	PRODUTOS
• Log de Testes • Relatório de Ocorrências • Resumo dos Testes	• Aceitação Parcial • Artefatos e Registros dos Testes

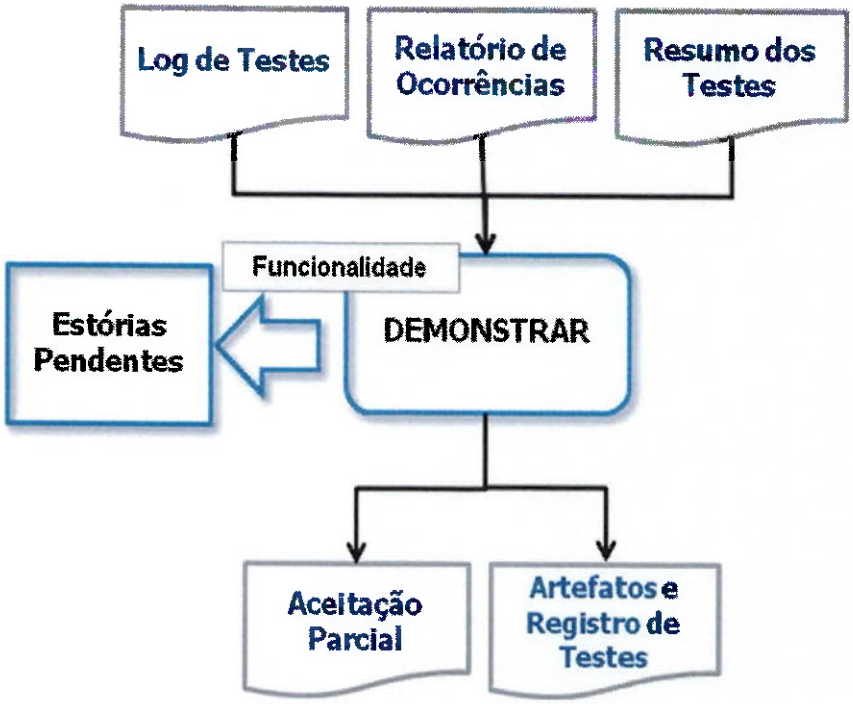

Figura 19 – Etapa Demonstrar do ATDD Proposto

Nesta etapa do processo os seguintes pápeis estão envolvidos.

PAPEIS ENVOLVIDOS
• Analista de Teste • Líder de Teste • Analista de Negócios • Gerente de Negócios

4.2.7 Encerrar

Uma vez que todos os requisitos de negócios e os critérios de aceitação estão plenamente satisfatórios e não existem novas estórias pendentes, o Gerente de Negócio deve recomendar a aceitação formal do sistema à equipe de Gestão do Projeto.

A aceitação do sistema é possível mesmo se houver defeitos pendentes que não ofereçam riscos ao negócio. O sistema pode ser aceito com a qualificação que esses defeitos não são críticos e serão corrigidos posteriormente. Desta forma, a aceitação do sistema pode ser obtida com o consenso comum entre a gerência do projeto.

A decisão de permitir uma aceitação qualificada e determinar que um defeito não represente risco ao negócio é uma prerrogativa do Gerente de Negócios, que é o proprietário do sistema. No entanto, com o consenso de todos os *stakeholders* que o sistema está apto para a implantação, a aceitação formal é concedida pelos líderes / gerentes das equipes do projeto e o processo de teste de aceitação é concluído.

Esta etapa do processo é responsável por fornecer os insumos e produtos, conforme relacionado a seguir e ilustrado na Figura 20.

INSUMOS	PRODUTOS
• Log de Testes • Relatório de Ocorrências • Resumo dos Testes	• Aceitação Formal do Sistema • Artefatos e Registros dos Testes

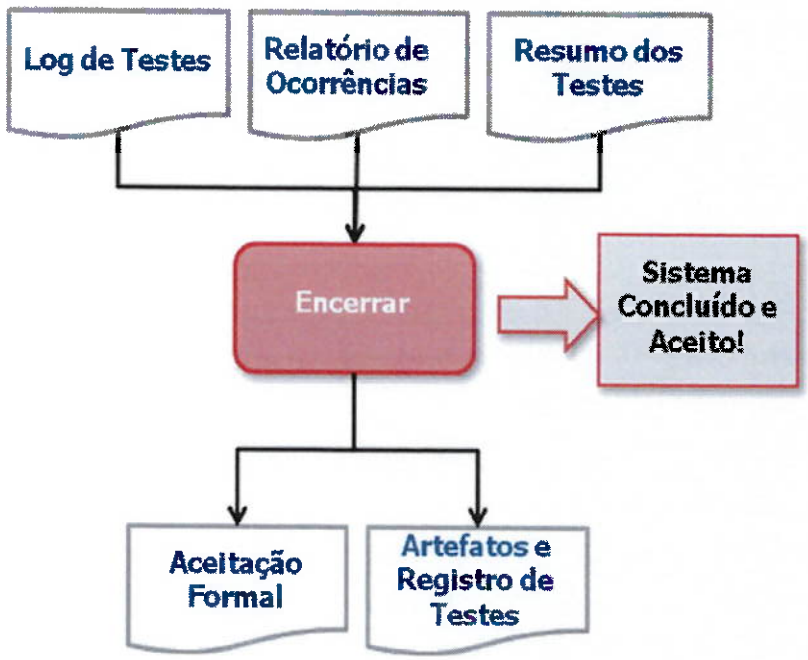

Figura 20 – Etapa Encerrar do ATDD Proposto

Um breve descritivo dos produtos desta etapa é apresentado a seguir.

4.2.7.1 Artefatos e Registro de Testes

Os artefatos produzidos no processo de teste, tais como, Casos de Teste, *Scripts* de Teste, Dados de Teste, Log de Teste, Relatório de Ocorrências e Resumo dos Testes devem ser mantidos para testar as versões posteriores do sistema. O Líder de Teste de Aceitação deve entregar formalmente esses artefatos e registros de teste para a custódia do Gerente de Negócios.

É importante manter todos os registros dos testes ao menos durante o período de garantia do produto de software.

Nesta etapa do processo os seguintes pápeis estão envolvidos.

PAPEIS ENVOLVIDOS
• Gerente de Negócio • Gerente do Projeto • Patrocinador do Projeto • Líder de Teste • Lider de Desenvolvimento

4.3 Implementar *versus* Executar e Controlar os Testes

A interação das etapas Implementar e Executar e Controlar os Testes é responsável por garantir a integridade das funções implementadas, que devem satisfazer os requisitos de negócio do cliente, e deve assegurar que os artefatos de testes estão em conformidade com tais exigências.

A Figura 21 ilustra esta interação.

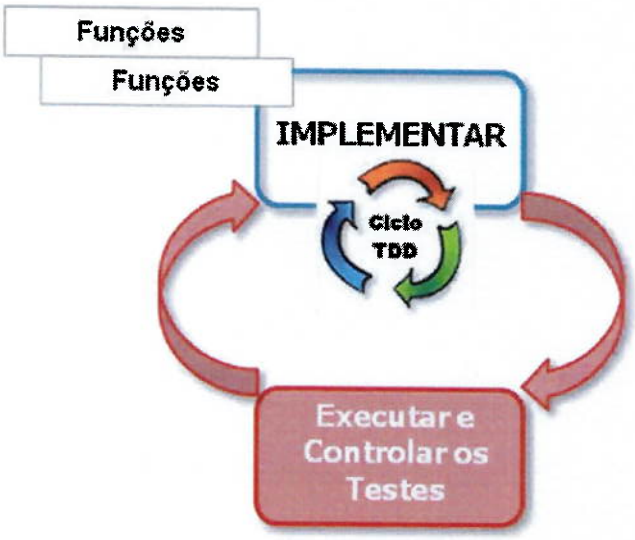

Figura 21 - Interação das etapas Implementar *versus* Executar e Controlar os Testes da do ATDD Proposto

Esta interação oferece a perspectiva de quem executa os testes de aceitação (usuários finais / clientes com o apoio da equipe de testes), cuja finalidade é identificar os defeitos e relatá-los, e a perspectiva de quem deve reconhecer os defeitos, promover as correções e liberar uma nova versão de software (equipe de desenvolvimento).

Quando as estórias de usuário previstas para a iteração são implementadas, assim como os respectivos artefatos de testes preparados, deve-se iniciar a execução manual dos testes de aceitação, conforme ilustrado na Figura 22.

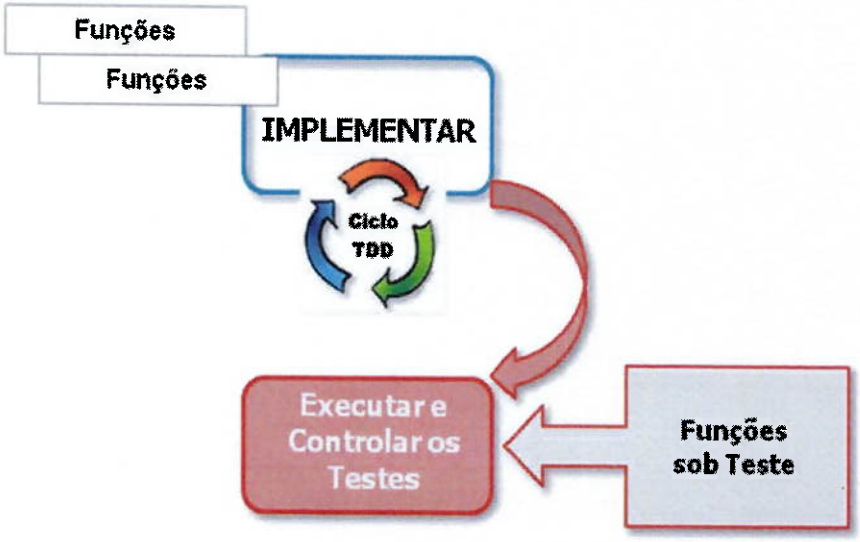

Figura 22 – Perspectiva da Execução dos Testes do ATDD Proposto

Durante a execução dos testes de aceitação quando um evento inesperado é identificado é necessário relatá-lo para que os desenvolvedores possam reconhecer o evento como um defeito válido ou não.

A Figura 23 ilustra as etapas que compreendem a identificação de um defeito.

Figura 23 - Identificação de defeitos (BASTOS et al, 2007. p.193)

A seguir há uma breve descrição das etapas que compreendem a identificação de um defeito. (BASTOS et al, 2007. p.193 - p.196).

- **Identificar o Defeito:** A identificação do defeito ocorre através da execução dos *Scripts* e Casos de Testes, que devem garantir que as funções do sistema estão plenamente aderentes aos requisitos do cliente.

- **Relatar o Defeito:** Ao relatar um defeito algumas características relevantes devem ser consideradas para assegurar a sua legibilidade:

    - **Resumir:** Relatar claramente e de forma sucinta.

    - **Precisar:** Deve-se entender por completo o comportamento do defeito e a sua abrangência.

    - **Neutralizar:** Informar apenas os fatos, omitir emoções.

    - **Reproduzir:** Ao identificar um defeito, a primeira tarefa é verificar quais foram os passos prévios para a detecção do defeito.

Reportar como reproduzi-lo é essencial para apoiar os desenvolvedores durante a correção do defeito e, posteriormente, para o usuário validar as correções.

- **Determinar Impacto:** Deve-se determinar qual a severidade (impacto) do defeito para o funcionamento da aplicação. Esta classificação é feita pelo usuário que identificou o defeito e basea-se na sua percepção.

- **Evidenciar:** Devem-se fornecer as evidências da existência de um defeito. A evidência pode ser um arquivo, imagem, vídeo, etc. É recomendável fornecer também o Caso de Teste e o *Script* de Teste utilizado na identificação do defeito. Evidenciar o defeito oferece credibilidade ao seu relato.

O relato de defeitos pode ocorrer automaticamente quando utilizadas ferramentas de gestão de defeitos, o que promove maior agilidade no reconhecimento do defeito. Independente do uso de ferramentas é fundamental que os defeitos sejam relatados periodicamente durante a iteração.

- **Reconhecer o Defeito:** Após receber o defeito e as evidências de sua presença, o desenvolvedor deve reconhecer o defeito como válido ou não. O desenvolvedor sempre consultará a documentação do sistema para reconhecer o defeito e deverá considerá-lo se o sistema apresentar um comportamento diferente do especificado nas estórias ou requisitos. Sempre que houver divergências se um defeito é válido ou não, deve-se direcionar a decisão para o proprietário do sistema (cliente) que definirá se o defeito é válido ou não.

Caso o defeito relatado não seja reconhecido como defeito do sistema, mas sim, um defeito do artefato de testes e/ou requisito, a ocorrência deve ser reclassificada conforme a origem do defeito e, posteriormente, o Plano de Teste de Aceitação, assim como os demais artefatos de testes deverão ser

atualizados com as definições estabelecidas em comum acordo com os *stakeholders* do projeto.

Quando o defeito é reconhecido como do sistema, ele deve ser solucionado de acordo com a prioridade de correção. É recomendável que o desenvolvedor consulte o cliente para que ambos definam a prioridade de correção. Tipicamente, a escala de prioridade é definida como: Imediata, Alta, Normal e Baixa.

Após corrigir o defeito, os desenvolvedores esclarecem a natureza da correção, através do documento conhecido como *release notes*, e estabelecem quando a correção será liberada para os testes de aceitação.

Quando uma nova versão do sistema é liberada para os testes de aceitação, a prioridade na reexecução dos testes deve ser os defeitos que apresentaram falha durante a primeira execução dos testes, conhecido também como primeiro ciclo de testes. Caso todos os defeitos tenham sido corrigidos, os usuários finais devem executar os demais Casos e *Scripts* de Testes para assegurar que as correções não introduziram falhas que inicialmente não existiam.

A Figura 24 ilustra a perspectiva da equipe de desenvolvimento (reconhecer o defeito, solucioná-lo e liberar nova versão de software) com a interação entre as etapas Implementar e Executar e Controlar os Testes.

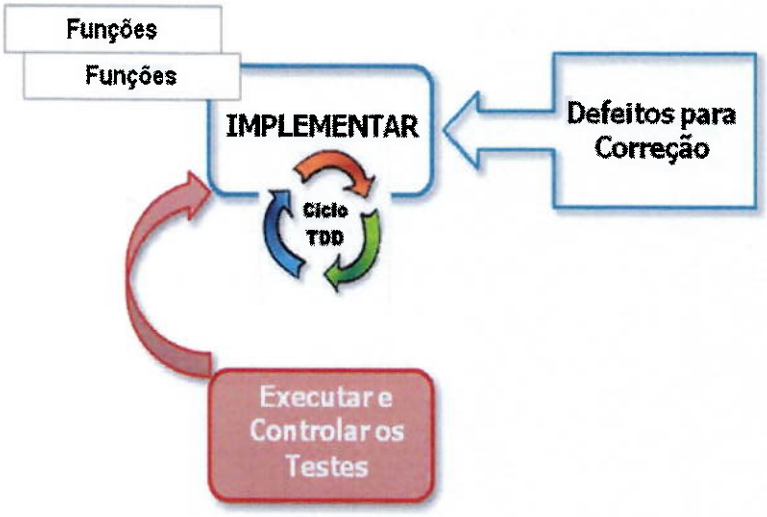

Figura 24 – Perspectiva da Equipe de Desenvolvimento para o ATDD Proposto

Ao término da realização de todos os testes previstos para a iteração, se nenhum novo defeito foi identificado, os registros de testes, assim como as funções previstas na iteração devem ser submetidas à aceitação do Analista e Gerente de Negócios, conforme descrito na etapa Demonstrar.

Nesta interação entre as etapas Implementar e Executar e Controlar os Testes os seguintes pápeis estão envolvidos.

PAPEIS ENVOLVIDOS
• Usuário Final / Cliente • Líder de Teste • Analista de Teste • Líder de Desenvolvimento • Analista de Desenvolvimento

4.4 Considerações do Capítulo

A proposta de adaptação do ATDD descrita neste capítulo tem como finalidade encorajar e privilegiar o uso dos testes de aceitação no contexto do desenvolvimento ágil de software que, geralmente, faz uso apenas dos testes de unidade e suas respectivas integrações, conforme previsto na técnica do TDD, como testes definitivos para garantir a qualidade do produto de software. Além disso, a substituição dos testes de aceitação executáveis pelo planejamento e organização dos testes de modo que seja possível a execução manual dos testes de aceitação oferece uma alternativa para que o teste de aceitação não seja omitido ou negligenciado dado à complexidade em automatizar os testes de aceitação.

Esta proposta preza pela colaboração e sinergia entre os *stakeholders* do projeto de software, onde os interesses do cliente devem permear todas as etapas do processo de desenvolvimento do software, visando satisfazer suas necessidades e expectativas.

Os artefatos mencionados no ATDD proposto representam os produtos de testes mínimos para que o processo de teste funcione satisfatoriamente. Além disso, embora a execução dos testes tenha o cunho manual, não significa que a elaboração, o controle e a manutenção dos testes devam ser realizados manualmente. Existem ferramentas de gestão de testes gratuitas e *open source* que oferecem suporte aos documentos e as etapas citadas nesta proposta. Desta forma, é possível garantir com menor esforço uma gestão eficaz dos processos de testes descritos neste capítulo.

O capítulo seguinte apresenta algumas diretrizes para o uso do ATDD proposto, permeando todas as etapas descritas neste capítulo.

5. DIRETRIZES PARA O USO DO ATDD PROPOSTO

A proposta de adaptação do ATDD acresce quatro etapas ao processo original de Hendrickson (2008), conforme exposto em detalhes no capítulo anterior. Este acréscimo tem como finalidade encorajar e privilegiar o uso dos testes de aceitação no projeto de desenvolvimento de software.

Neste capítulo, a título de ilustração, sugerem-se algumas diretrizes para o uso do ATDD proposto, promovendo algumas recomendações para cada uma das etapas que compreende esta proposta. A finalidade destas diretrizes é apoiar as equipes do projeto de software a desempenhar satisfatoriamente algumas das atividades de engenharia de software previstas nesta proposta.

Além disso, alguns aspectos do uso do ATDD proposto no projeto de software são inferidos e descritos neste capítulo.

5.1 Discutir

Nesta etapa do processo, o cliente deve descrever através de exemplos concretos de funções suas necessidades e expectativas sobre o sistema, ilustrando como o sistema deve funcionar. Para assegurar um melhor entendimento, a equipe de teste e desenvolvimento deve discutir com o cliente sobre as funções e características do sistema para elicitar os critérios de aceitação do sistema.

Este evento promove uma intensa discussão, estimula questionamentos e promove o aprendizado. Permite também levantar e documentar as informações dos problemas atuais, possíveis oportunidades e riscos.

Para desenvolver esta etapa adequadamente, algumas diretrizes devem ser observadas:

- Reunir os *stakeholders* em um local apropriado, quieto e sem interrupções.

- Programar um tempo hábil para que todas as questões sejam discutidas. Caso seja necessário, este encontro deve ocorrer em mais de um momento.

- É imprescindível a participação de todas as equipes do projeto. Ou seja, as equipes de: negócios, testes, desenvolvimento e gestão de projeto.

- O papel do cliente deve ser representado por indivíduos com amplo domínio do negócio.

- Usar técnicas visuais para melhorar a comunicação e o entendimento.

- Evitar o uso de termos técnicos.

- Evitar entrar no contexto de potenciais soluções.

- Evitar generalizações por parte do cliente (obter informações específicas).

- Evitar conduzir a discussão com a tentativa de persuadir o cliente.

- Fazer perguntas ou comentários que encoraje o cliente a falar mais sobre o item em discussão.

- Questionar o cliente se a estória em discussão tem dependências, de modo a agrupar as estórias comuns do sistema, formando assim um escopo conciso.

- Questionar os possíveis cenários da estória em discussão (fluxo principal, alternativos, exceções, etc.).

- Registrar todas as informações relevantes da discussão (cenários de testes, critérios de aceitação, etc.).

Uma vez que são colocadas as perguntas certas durante esta discussão, o cliente deve pensar com cautela e mais critério sobre suas necessidades, e o que pode parecer uma simples estória pode ser desdobrada em mais de uma estória.

O desafio desta etapa do processo está em extrair do cliente todas as informações relevantes do negócio, esclarecer e alinhar com todos os *stakeholders* as necessidades e expectativas sobre o que o sistema deve e não deve fazer.

5.2 Elaborar o Plano de Teste de Aceitação

Nesta etapa o Plano de Teste de Aceitação é elaborado com o objetivo de documentar e organizar os produtos de trabalho da etapa anterior (Discutir). Este documento é elaborado colaborativamente entre as equipes de negócio, teste e desenvolvimento, com o apoio da equipe de gestão de projetos.

Além de documentar, esta etapa deve ratificar os entendimentos e alinhar as expectativas discutidas inicialmente.

Para desenvolver esta etapa adequadamente, algumas diretrizes devem ser observadas:

- O tempo para documentar os produtos de trabalho da etapa discussão deve ser de algumas horas ou poucos dias (por iteração).

    - O Plano de Teste de Aceitação é insumo para as etapas seguintes do processo, logo documentá-lo não deve consumir um tempo que venha a ser prejudicial à evolução do projeto de desenvolvimento de software.

- O Plano de Teste de Aceitação não deve ser extenso.
    - Este documento deve conter apenas informações relevantes ao negócio e especificar os aspectos inerentes aos testes de aceitação.
- Garantir o sincronismo do Plano de Teste de Aceitação.
    - Quaisquer manutenções nos requisitos, independente do motivo, devem ser contempladas prontamente no Plano de Teste de Aceitação.
- Comunicar as atualizações através do Plano de Teste de Aceitação.
    - A comunicação face a face é a maneira mais eficiente de transmitir informações entre os *stakeholders*, e deve ser priorizada no projeto de desenvolvimento. Logo, a documentação não deve ser o único meio de comunicação de mudanças. Documentar os aspectos de negócio e seus respectivos testes através do Plano de Teste de Aceitação assegura que ao término do projeto tenha-se um documento formal atualizado do sistema.

O desafio desta etapa do processo é garantir que o conhecimento não fique tácito e minimizar o risco de que informações relevantes caiam no esquecimento.

5.3 Preparar os Testes de Aceitação

Nesta etapa os artefatos de testes são preparados para a realização dos testes de aceitação. A produção destes artefatos é de responsabilidade da equipe de testes com a participação efetiva do cliente, que deve escrever os testes com o respectivo apoio técnico.

Ferramentas de apoio à construção destes artefatos de testes permitem maior eficácia na criação e controle destes artefatos. Existem ferramentas comercias e *open source*, que podem assegurar uma boa produtividade a esta etapa do processo. Alguns exemplos de ferramentas *open source* são: o Testlink (teamst.org) e o Testopia (mozilla.org/projects/testopia).

Para desenvolver esta etapa adequadamente, algumas diretrizes devem ser observadas:

- Os artefatos de testes devem ser objetivo e bem escritos de modo que qualquer usuário final do sistema possa compreendê-los e utilizá-los sem dificuldades.

- Um Caso de Teste objetivo é aquele que valida a uma única função. Assim, os testadores conseguem focar melhor na idéia principal do teste (alvo do teste).

- Um Caso de Teste deve ser auto-suficiente e conter toda a informação necessária para executá-lo.

- Evitar Casos de Teste exaustivos, com um número muito grande de passos, pois além de tomar muito tempo durante sua execução, tendem a causar dispersão durante a realização do teste, perdendo o foco principal do teste.

- Identificar e classificar os Casos de Teste mais relevantes para o negócio que devem contemplar o *baseline* do *Smoke Test*.

- Um *Script* de Teste deve contemplar Casos de Testes eficientes, com a maior cobertura possível.

- Revisar os *Scripts* de Teste produzidos, com a finalidade de detectar falha de compreensão ou irrelevância nos Casos de Teste.

- Garantir a confidencialidade dos dados reais de produção.

- Na ausência de dados reais de produção, utilizar dados que simulem rigorosamente a operação de produção.

- Quando possível, disponibilizar previamente a equipe de desenvolvimento os artefatos de testes produzidos e validados pelo cliente.

- Comunicar aos *stakeholders* a alteração ou previsibilidade de alteração dos cenários de testes e/ou critérios de aceitação.

- Garantir o sincronismo dos artefatos de testes (Casos, *Scripts*, Dados de teste, etc.).

O desafio desta etapa do processo é garantir a participação efetiva do cliente na elaboração dos artefatos de teste.

5.4 Implementar

Nesta etapa as funções priorizadas pelo cliente para a iteração são implementadas pela equipe de desenvolvimento. Assim como o ATDD, esta proposta de adaptação do ATDD não determina como implementar a função, mas considera o uso do TDD a melhor prática.

Para desenvolver esta etapa adequadamente, algumas diretrizes são relacionadas a seguir (ANICHE E GEROSA, 2010. p.473; p.475.) e (CAUSEVIC, et. al, 2011. p.341.):

- Empregar desenvolvedores com experiência e amplo conhecimento do TDD, e as ferramentas específicas de apoio a esta técnica.

- Prezar o mantra do TDD (não negligenciar nenhuma das etapas do TDD).

- Iniciar com um teste simples e aprimorá-lo durante a implementação.
- Implementar com simplicidade a função.
- Refatorar constantemente o código da aplicação e o código de teste.
- Executar toda suíte de teste a cada correção realizada.

Além das diretrizes inerentes a técnica do TDD, o ATDD proposto prevê algumas diretrizes para esta etapa do processo:

- O desenvolvedor deve ter acesso freqüente ao cliente para esclarecer dúvidas sobre suas necessidades e expectativas.
- Os artefatos dos testes de aceitação podem ser disponibilizados previamente aos desenvolvedores para certificar a conformidade da função implementada.
- Os registros dos testes de unidade (evidências), incluindo os testes falhos, podem prover informações sobre a qualidade e as limitações técnicas da equipe de desenvolvimento. Desta forma, é possível atuar na capacitação dos profissionais de desenvolvimento com treinamentos específicos.

O uso do TDD demanda esforço, o que tende a desviar os desenvolvedores das boas práticas previstas pela técnica. Logo, o desafio desta etapa do processo está em aplicar com perseverança e disciplina o TDD, com o foco na solução sustentável dos problemas.

5.5 Executar e Controlar os Testes

Após a implementação das funções previstas para a iteração, nesta etapa do processo é o momento de validar a aderência das funções com os requisitos, que estão explícitos através dos artefatos de testes.

O cliente é responsável pela a realização dos testes de aceitação, geralmente, representado pelos usuários finais do sistema, que devem contar com o apoio permanente da equipe de teste durante toda a execução dos testes.

Embora esta proposta preveja a execução dos testes manuais, é recomendável o uso de ferramenta de gestão de defeitos para controlar o ciclo de vida dos defeitos, desde o seu relato até o seu fechamento. Além disso, oferece também uma base comum para a entrada das informações e promove a integração entre a equipe de desenvolvimento e a equipe de testes.

Para desenvolver esta etapa adequadamente, algumas diretrizes devem ser observadas:

- Assegurar que os testes que antecedem o nível de aceitação (testes de unidade, integrado e sistema) foram realizados.

- A equipe de teste deve apoiar a execução dos testes.

- Estabelecer um nível de *log* (cronológico) para registrar todos os eventos significativos dos testes.

- Documentar através de um relatório de ocorrências/incidentes qualquer evento inesperado durante a execução dos testes.

- Todo e qualquer defeito identificado sem o uso de um artefato de teste deve ser documentado para evitar a sua presença em futuros testes.

- Defeito de severidade crítica, que impossibilite a continuidade dos testes, deve ser relatado de imediato ao Líder de Teste para que sejam tomadas as providências cabíveis (suspensão ou cancelamento dos testes).

- Controlar e monitorar o progresso e os resultados dos testes.

- Comunicar frequentemente o progresso e os resultados dos testes através de relatórios resumo.

- Assegurar a rastreabilidade dos artefatos de testes com os requisitos.

- Assegurar a integridade e o versionamento dos artefatos de testes (sincronismo com os requisitos).

O desafio desta etapa do processo é certificar a integridade do planejamento dos testes, assim como seus artefatos, e assegurar a qualidade do produto de software em conformidade com as expectativas e necessidades do cliente. Além disso, deve-se garantir que quaisquer desvios nestes aspectos sejam contornados adequadamente, sem comprometer o progresso dos testes e, consequentemente, o projeto de desenvolvimento do software.

O conhecimento adquirido nesta etapa do processo deve promover a melhoria contínua dos processos de testes.

5.6 Demonstrar

Nesta etapa do processo as evidências dos testes para a iteração são verificadas pelo Analista de Negócios para assegurar a aderência aos requisitos de negócios e os critérios de aceitação.

Para desenvolver esta etapa adequadamente, algumas diretrizes devem ser observadas:

- Utilizar como referência a última versão do Plano de Teste de Aceitação para validar o produto de software implementado.

- Verificar as evidências dos testes (Resumo dos Testes, Relatório de Ocorrências e os *Logs* de Teste).

- Realizar testes exploratórios no sistema para assegurar a qualidade do produto.

- Validar a criticidade dos defeitos presentes no software para o negócio.

    - Caso seja identificado algum defeito que represente alguma ameaça ao negócio, o sistema deve ser recusado e o defeito corrigido.

- Recomendar a aceitação formal do sistema à equipe de Gestão do Projeto.

Através da análise estática dos resultados provenientes dos testes, e também da análise dinâmica, com a aplicação de testes exploratórios, a equipe de negócio deve comprovar e ratificar a conformidade do software implementado com as expectativas e necessidades do cliente. Isto não significa que o software tenha que ser livre de defeitos, no entanto, deve ser suficientemente bom para o uso pretendido, o qual irá determinar o grau de confiança do produto de software.

5.7 Encerrar

Esta etapa do processo declara a aceitação formal do sistema pela gerência do projeto, permitindo também a aceitação qualificada do sistema. Esta aceitação qualificada não deve representar riscos ao negócio. Isto significa que apenas os defeitos de baixa severidade e sem impacto significativo a operação são passíveis a esta qualificação.

Para desenvolver esta etapa adequadamente, algumas recomendações são relacionadas a seguir:

- A aceitação qualificada é uma prerrogativa apenas dos Gerentes de Negócios e Projeto.

- A aceitação formal do sistema deve ser concedida pelos líderes / gerentes das equipes do projeto.

- Os registros de testes devem ser retidos durante a garantia do produto de software.

- Os artefatos de testes devem ser mantidos em um repositório para fins de reuso em versões posteriores do sistema.

Esta etapa do processo formaliza a conclusão dos testes de aceitação e habilita a promoção do sistema para a implantação no ambiente de produção.

5.8 Implementar *versus* Executar e Controlar os Testes

Esta interação entre as etapas, implementar e executar e controlar os testes, tem como finalidade reconhecer e rastrear as causas que originaram um defeito, após identificá-los na etapa de execução e controle, e solucionar este defeito provendo uma nova versão do sistema.

Para desenvolver esta etapa adequadamente, algumas diretrizes devem ser observadas:

- Fornecer aos desenvolverdores as evidências da existência do defeito (*logs*, artefatos com os procedimentos e dados do teste para a reprodução do defeito, etc.).

- Reclassificar o *status* do defeito quando ele não for reconhecido como falha do software, mas, falha dos artefatos de testes e/ou requisitos. Nesta condição devem-se prover as respectivas atualizações dos artefatos em comum acordo entre os *stakeholders*.

- Corrigir os defeitos de acordo com a prioridade de correção, definidas em comum acordo entre o desenvolvedor e o cliente.

- Documentar a natureza dos defeitos através de um *release notes*.

- Priorizar a reexecução dos testes que falharam após gerar uma nova versão do sistema.

- Utilizar o *Smoke Test* como atalho aos testes de regressão nos estágios avançados do desenvolvimento de software.

    - Após inúmeras iterações, o tempo para executar os testes regressivos de forma manual pode ser maior do que o tempo usado para testar as novas funções da iteração em questão. Desta forma, recomenda-se adotar o *Smoke Test* para validar as funções de negócio mais críticas das iterações anteriores e assegurar que estas funções continuam funcionando. Então, após realizar o *Smoke Test*, se nenhuma falha for identificada no sistema, deve-se prosseguir apenas com os testes planejados para a iteração em questão.

O desafio desta etapa do processo é minimizar o número de versões do sistema, que deve ser o menor possível. Gerar muitas versões do sistema, além de indicar a baixa qualidade no desenvolvimento do produto, promove um trabalho adicional das equipes do projeto, e isto pode comprometer os prazos do projeto, além da qualidade global do sistema.

Contudo, é fundamental assegurar a boa qualidade dos insumos e produtos de trabalho das etapas anteriores. Além disso, a colaboração e

comunicação são instrumentos determinantes para promover o sucesso desta etapa, assim como do projeto de software.

5.9 Resultados do ATDD Proposto no projeto de software

Embora a o ATDD proposto privilegie a execução manual dos testes de aceitação, esta proposta preza por preservar os resultados positivos do ATDD, dentre eles, a especificação através de exemplos concretos de funções do sistema, onde o cliente por meio de exemplos inteligíveis elucida como o sistema deve funcionar, promovendo uma interação entre os membros da equipe de projeto, que devem assumir uma postura de colaboração e comprometimento com o sucesso do projeto.

Considerando as afirmações apresentadas no ATDD proposto e suas respectivas diretrizes, é possível inferir alguns resultados provenientes desta proposta, conforme segue.

- **Foco na área de domínio**

  Recomenda-se que as áreas de conhecimento do projeto de software, representada, essencialmente, pelas equipes de negócio, teste e desenvolvimento, atuem com foco em suas competências de domínio. Isto não significa que as equipes não devam trabalhar colaborativamente, mas que os esforços sejam concentrados, principalmente, sob suas melhores habilidades e conhecimentos. Por exemplo, o desenvolvedor tem como foco implementar a função, os analistas de testes tem como foco apoiar o cliente na modelagem dos testes de aceitação, etc.

- **Documentação dos requisitos de negócio**

  Ao término do projeto de desenvolvimento de software, o Plano de Teste de Aceitação fornece a especificação dos testes de aceitação como

documentação dos requisitos de negócios, descritos em linguagem natural e de domínio do cliente, sem uso de tabelas e códigos de suporte aos testes. Esta legibilidade permite que qualquer indivíduo, sem habilidades específicas, possa compreender e fazer uso do documento.

- **Intuição *versus* Rotina**

O "paradoxo do pesticida" diz que quanto mais veneno se coloca nas pragas, mais resistentes elas ficam. No contexto do teste de software, isto significa que de tanto executar um teste automatizado, os defeitos que existiam não aparecem mais, o que faz com que as pessoas acreditem que não existam mais defeitos no código da aplicação. Porém, se fosse alterado algum comando nos testes automatizados, existe uma maior probabilidade de identificar novos defeitos.

Nos testes manuais esta rotina é minimizada, onde o testador, instintivamente, procura realizar o teste por um caminho diferente do realizado anteriormente, quando possível. Além disso, a validação dos procedimentos operacionais de forma manual permite simular situações reais do cotidiano, possibilitando o *feedback* do cliente.

- **Reuso dos artefatos de testes**

Os artefatos de testes produzidos durante o projeto de desenvolvimento do software devem ser armazenados para eventual reuso em projetos de software com características semelhantes. O reuso dos artefatos de testes pode minimizar o esforço na preparação dos testes de aceitação em projetos futuro.

Alguns fatores críticos ao uso do ATDD proposto também podem ser citados:

- **Redução de agilidade**

  Comparado com o tempo de execução de uma automação de testes, que permite executar os testes com muita agilidade e em horários flexíveis, por exemplo, durante toda a noite, o teste de aceitação realizado de forma manual acrescenta mais tempo à etapa de execução dos testes. Esta inconveniência, geralmente, não é um obstáculo por optar pelos testes manuais, pois, os testes de aceitação possuem um escopo mais abreviado. No entanto, automatizar os testes pode ser inviável, como já foi citado neste trabalho.

  Para otimizar o tempo de execução manual dos testes de aceitação, dependendo do escopo e situação do projeto de software, é recomendável fazer uso de um maior número de usuários finais do sistema para a realização dos testes de aceitação.

- **Redução de produtividade**

  Após muitas iterações do software, o tempo para executar os testes regressivos de forma manual pode ser maior do que o tempo usado para testar as novas funções implementadas para uma iteração atual.

  Uma maneira de minimizar esta perda de produtividade é realizar o *Smoke Test* como teste de regressão para validar somente as funções de negócio mais críticas das iterações anteriores. Esta solução pode representar riscos à qualidade do produto de software.

5.10 Considerações do Capítulo

As diretrizes expressas neste capítulo têm como finalidade apoiar os *stakeholders* do projeto de desenvolvimento de software a desempenhar satisfatoriamente cada uma das etapas que compreendem o ATDD proposto.

Estas recomendações norteadas pelos valores e princípios da filosofia ágil, que privilegia a comunicação direta e eficaz, a entrega freqüente de software funcionando, pessoas de negócio, testes e desenvolvimento interagindo e colaborando com experiências e conhecimentos, possibilitam produzir melhores resultados que agregue valor ao cliente.

Os aspectos positivos previstos pelo ATDD são preservados nesta proposta. Além disso, os resultados inferidos ao uso desta proposta são fundamentados no conhecimento acadêmico e cientifico das respectivas afirmações. Logo, não é possível predizer outros aspectos relevantes do uso desta proposta, uma vez que o foco deste trabalho não é a aplicação do ATDD proposto.

O uso isolado das diretrizes presentes neste capítulo não assegura o sucesso do ATDD proposto. Isto significa que é fundamental a adoção de uma atitude e comportamento positivo por parte dos *stakeholders* do projeto de software quanto a sua adoção, considerando a relevância dos testes de aceitação à qualidade do produto de software e satisfação do cliente.

6. CONSIDERAÇÕES FINAIS

Conforme discutido no decorrer deste trabalho, os métodos ágeis são um conjunto de métodos de desenvolvimento de software adequados às mudanças constantes. Diversos valores, princípios e práticas devem ser aplicados no projeto de desenvolvimento de software para assegurar o sucesso do projeto.

A atividade de testes está inserida neste contexto como um instrumento para garantir a qualidade do produto de software.

Este trabalho teve como finalidade caracterizar a atividade de teste de software, em particular, os testes de aceitação aplicado dentro do contexto dos métodos ágeis, abordando a técnica do *Acceptance Test Driven Development* (ATDD) para assegurar a clareza das necessidades de negócio do cliente, garantir a implementação adequada do produto de software e a satisfação do cliente.

Algumas diretrizes são apresentadas com a finalidade de apoiar os *stakeholders* do projeto de desenvolvimento de software a desempenhar satisfatoriamente o uso do ATDD proposto. Estas diretrizes norteadas pelos valores e princípios da filosofia ágil possibilitam produzir resultados de valor ao cliente.

Este capítulo apresenta também as contribuições do trabalho desenvolvido e as sugestões para trabalhos futuros.

6.1 Contribuições do Trabalho

Como contribuição acadêmica, este trabalho propôs uma adaptação de um processo de teste de aceitação de software no contexto dos métodos ágeis para estruturar e coordenar, principalmente, as equipes de teste e desenvolvimento. Além de recomendar algumas diretrizes para a adoção desta proposta.

Desta forma, procurou-se com este trabalho contribuir com a redução do número de defeitos identificados durante o projeto e a implementação do

software. Assim, procura-se reduzir os custos do desenvolvimento e a manutenção do produto de software.

Este trabalho apresentou algumas dificuldades dada a escassez de estudos descrevendo os aspectos inerentes ao processo do ATDD. A maioria dos trabalhos destaca o ATDD sob o ponto de vista da experimentação de ferramentas de automação dos testes, avaliando os desafios e aspectos da utilização destas ferramentas. Portanto, consolidar o conteúdo do presente trabalho e propor uma adaptação do ATDD representaram um grande desafio.

Contudo, este trabalho restringiu-se ao campo teórico e não há evidências dos resultados do ATDD proposto. Desta forma, é importante que trabalhos futuros explorem amplamente o ATDD proposto de modo a validar as afirmações presentes neste trabalho e permitir a identificação de melhorias ao processo.

6.2 Trabalhos Futuros

O ATDD proposto acresce novas etapas ao processo original de modo a viabilizar a execução manual dos testes de aceitação pelos usuários finais do sistema, em detrimento a automação dos testes. Portanto, trabalhos futuros podem ser desenvolvidos experimentando o ATDD proposto para consistir a aderência das novas etapas do processo. Esta experimentação pode ser realizada por diferentes grupos de desenvolvimento e testes de software, com níveis de conhecimentos distintos e sob o contexto de um projeto de software com características semelhantes.

Outra oportunidade de trabalho futuro é comparar o ATDD (original) e o ATDD proposto usando o mesmo projeto de desenvolvimento de software para mensurar precisamente as discrepâncias com relação a produtividade entre as etapas dos processos, as distinções na qualidade dos testes produzidos e na manutenibilidade desses testes. Além disso, verificar a defasagem no prazo, custo e qualidade do projeto de software utilizando ambas as abordagens.

Por fim, avaliar outros aspectos do uso do ATDD proposto não identificados no presente trabalho em função da não aplicação desta proposta.

REFERÊNCIAS

ABRAHAMSSON, P.; SALO, O.; RONKAINEN, J.; WARSTA, J. **Agile Software Development Methods – Review and Analysis.** Relatório Técnico 478, VTT PUBLICATIONS, 2002. 112p.

ANICHE, M.F., GEROSA, M.A. **Most Common Mistakes in Test-Driven Development Practice: Results from an Online Survey with Developers.** In ICST Workshops, 2010. p.469-478.

ASTELS, D. **Test Driven Development: A Practical Guide**. Upper Saddle River, New Jersey, USA, Prentice Hall, 2003. 562p.

BASTOS, A., RIOS, E., CRISTALLI,R., MOREIRA, T. **Base de Conhecimento em Teste de Software**, São Paulo, SP, Martins Fontes, 2007. 263p.

BECK, K. **Test Driven Development: By Example.** Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc., 2003. 220p.

BUGZILLA. Website: http://www.bugzilla.org/. Acessado em 18/01/2012.

CAUSEVIC, A., SUNDMARK, D., PUNNEKKAT, S. **Factors Limiting Industrial Adoption of Test Driven Development A Systematic Review**, 2011. p.337-346.

CUNNINGHAM, W. **FIT Documentation**, 2002. Disponível no website: http://fit.c2.com/wiki.cgi?FitDocumentation. Acessado em: 05/12/2011.

FITNESSE DOCUMENTATION ONLINE. Disponível no website: http://www.fitnesse.org/FitNesse.UserGuide. Acessado em: 20/05/2012.

FOWLER, M. **Refactoring: Improving the Design of Existing Code**. Addison Wesley Professional, 1999. 464p.

HANSSEN, G.; HAUGSET, B. **Automated Acceptance Testing Using Fit,** in Proc. HICSS, 2009. p.1-8.

HENDRICKSON, E. **Driving Development with Tests: ATDD and TDD**, Conference at Software Testing Australia and New Zealand (STANZ), 2008. p.1-9.

IEEEStd1012-1998. IEEE Standard for Software Verification and Validation. IEEE Computer Society, July 1998. 71p.

JEFFRIES, R.; MELNIK, G. **Guest editors' introduction: TDD-the art of fearless programming**, IEEE Software, v. 24, n. 3, 2007. p.24-30.

KOSKELA, L. **Test Driven: TDD and Acceptance TDD for Java Developers**, Manning Publications, 2007. 544p.

KRUCHTEN, P. **The Rational Unified Process: An Introduction,** 2nd Edition, 2000. 320p.

LARMAN, C; VODDE, B. **ATDD with Robot Framework**, 2010. p.1-15. Disponível em: http://code.google.com/p/robotframework/wiki/ATDDWithRobotFrameworkArticle Acessado em: 18/01/2012

MANTIS, Website: http://www.mantisbt.org/. Acessado em 18/01/2012.

MARTIN, R. C. **Professionalism and Test-Driven Development**. IEEE Software, v.24, n. 3, 2007. p.32-36.

MARTIN, R.C., MELNIK, G.: **Tests and Requirements, Requirements and Tests: A Möbius Strip**. IEEE Software, 2008. p.54-59.

MELNIK, G. **Empirical Analyses of Executable Acceptance Test Driven Development.** PhD Thesis. University of Calgary, 2007. 191p.

MELNIK, G.; READ, K.; MAURER, F. **Suitability of FIT User Acceptance Tests for Specifying Functional Requirements Developer Perspective**, in Proc. XP Agile Universe, 2004. p.60-72.

PARK, S., MAURER, F. **The Benefits and Challenges of Executable Acceptance Testing,** Workshop on Scrutinizing Agile, In Conjunction with ICSE, 2008. p.19-22.

PROJECT MANAGEMENT INSTITUTE (PMI), **A Guide to the Project Management Body of Knowledge**, 3rd Edition, 2004. 390p.

PUGH, K. **Lean-Agile Acceptance Test-Driven Development: Better Software Through Collaboration**, Addison-Wesley Professional, 2011. 345p.

PULEIO, M. **How not to do agile testing**. In: AGILE 2006 Conference, Minneapolis, Minnesota, USA: IEEE Computer Society, 2006. p.305-314.

READ, K.; MELNIK G.; MAURER F. **Student Experiences with Executable Acceptance Testing**, in Proc. AGILE, 2005. p.312-317.

RICCA F.; TORCHIANO, M.; PENTA, M.; CECCATO, M.; TONELLA, P. **Using acceptance tests as a support for clarifying requirements A series of experiments**, 2007. p.270-283.

SAUVÉ, J.; NETO, E.; NETO, O. **Experience Report Using EasyAccept to Drive Development of Software**, SAST 2007. p.79-84.

SINIAALTO, M. **Test-Driven Development: Empirical Body Of Evidence**, Volume: D.2.7, Publisher: Information Technology for European Advancement, 2006.15p.

TESTLINK. Website: http://teamst.org/. Acessado em: 20/04/2012.

TESTOPIA. Website: http://mozilla.org/projects/testopia/. Acessado em: 20/04/2012.

VERSION ONE SURVEY: **The State of Agile Development,** 2011.12p. Disponível no website: http://www.versionone.com/state_of_agile_development_survey/11/. Acessado em: 20/04/2012.