UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DIOGO GIARETTA REALI

Análise e Experiência no Desenvolvimento Guiado Ao Comportamento

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Karin Becker

Porto Alegre
2015

AGRADECIMENTOS

A esta Universidade, seu corpo docente, direção e administração que possibilitaram o crescimento tanto profissional quanto pessoal, através do conhecimento, respeito e ética.

Ao meu professor orientador, Karin Becker, pelo empenho, dedicação e apoio para a elaboração deste trabalho.

A meus pais, namorada, familiares e amigos que me apoiaram e me fortaleceram durante essa caminhada.

Enfim, a todos aqueles que fizeram parte da minha formação, o meu muito obrigado.

RESUMO

A construção de um *Software* eficiente e de qualidade requer um processo de desenvolvimento claro e objetivo. Diferentes processos são utilizados atualmente, entre eles estão o Processo Unificado, altamente baseado em documentação, e aqueles baseados em métodos ágeis, como o *Behavior Driven Development* (BDD). Apesar de muito se comentar sobre as vantagens do uso de BDD, muito pouco ainda se conhece sobre os reais benefícios da utilização desses métodos. Esse trabalho se propõe a apresentar, através do estudo de caso do desenvolvimento de uma aplicação web, um *framework* de avaliação baseado em mensuração, uma investigação comparativa do desenvolvimento segundo cada uma das abordagens, bem como uma análise dos resultados obtidos.

Analysis and Experience of Behavior Driven Development

ABSTRACT

The construction of an efficient and qualified software requires a clear development process and goal. Different processes are currently used, which include the Unified Process , highly based on documentation, and those based on agile methods, such as Behavior Driven Development. While much has been said on the advantages of using BDD, very little is known about the real benefits of using these methods. This work intends to present a framework based on measurement evaluation, a comparative research of development according to each approach, and an analysis of the results, based on a case study of the development of a web application.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O sucesso do desenvolvimento de um sistema computacional e a qualidade do Software produzido estão fortemente relacionados com o uso de métodos que guiem o processo de desenvolvimento e que forneçam os subsídios necessários para a construção do software e para sua manutenção. Aliada a esse fato, está a crescente complexidade dos sistemas, que demanda entre outros, um conhecimento detalhado do domínio da aplicação. Essa dificuldade se inicia na identificação dos requisitos, pois como é sabido, o cliente dificilmente conhece suas reais necessidades e as tem de forma subjetiva e implícita. Isto faz com que os requisitos a serem elucidados mudem e/ou tomem forma ao longo do tempo. Uma vez elucidados, eles devem ser representados de uma maneira que facilite o entendimento por pessoas leigas em relação ao domínio da aplicação, mas engajados no desenvolvimento da aplicação. Nesse sentido, diferentes artefatos, como casos de uso, diagramas, modelos, histórias, têm sido propostos e discutidos em uma tentativa de melhorar a expressividade na representação dos requisitos do projeto e facilitar sua compreensão e validação.

Pesquisadores da área de engenharia de software têm se dedicado a definir métodos que facilitem o desenvolvimento de um software de qualidade, que respeite o prazo de entrega, que contemple todos os requisitos e que esteja de acordo com o orçamento aprovado. Sommerville (2011) afirma que "existe uma forte relação entre a qualidade do produto de software desenvolvido e a qualidade do processo de software". Assim, é necessário que o processo a ser utilizado no desenvolvimento de um software tenha determinadas características que garantam a qualidade do produto final, tais como facilidade de compreensão, visibilidade, facilidade de suporte, aceitabilidade, confiabilidade, robustez, facilidade de manutenção e rapidez.

Com esse objetivo, surgem então os primeiros métodos de desenvolvimento, hoje conhecidos como "pesados" (*heavy-weight),* os quais têm como uma de suas principais características o uso intenso de documentação (Khan e Qurashi e Khan, 2011). Devido à crescente preocupação em identificar as atividades envolvidas durante o desenvolvimento de um software, e organizá-las em processos compostos por um conjunto de artefatos com diferentes níveis de abstração para a representação, esses métodos, ao passar dos anos, criaram uma sobrecarga nas organizações, sem necessariamente, resolver os problemas originais de não atender os requisitos e respeitar os prazos de entrega ou custo. Entre os motivos, está a custosa produção de artefatos e, consequentemente, a alta carga de trabalho

necessário para gerenciá-los. Um exemplo clássico são os processos que compõem o modelo de processo Sequencial, caracterizando-se, como o próprio nome sugere, pela execução sequencial das tarefas e pela produção de extensa documentação. A inflexível divisão do projeto em fases distintas, fazem com que testes de validação sejam possíveis somente ao final do processo, o que torna as possíveis alterações necessárias, em particular aquelas relacionadas a requisitos, custosas, e em muitos casos, implicando em um grande retrabalho.

O popular Processo Unificado, que segue o modelo interativo incremental, também é classificado como pesado. Entre as principais características estão o uso extenso de documentação como forma de comunicação, a definição de disciplinas que organizam papéis e atividades, e o posicionamento destes em fases. No *Chaos Report de 2009* (The Standish Group, 2009), o nível de formalidade em processos de desenvolvimento é considerado como um dos grandes responsáveis pelo aumento de custos, sem ser um fator de sucesso para o desenvolvimento do projeto do software.

Métodos Ágeis, também denominados como "leves" (*light-weight)* (Fowler, 2005), têm sido apontados como uma alternativa aos pesados. Ambientes nos quais os requisitos não são estáveis, e onde é ainda menos possível prever quais serão as necessidades futuras, são o alvo comum de muitas empresas de desenvolvimento de software. Dessa forma, a busca por abordagens que proporcionem uma maior facilidade na adaptação de alterações de requisitos, bem como agilidade no processo de desenvolvimento, se tornam de grande importância para o sucesso destas. Entre as principais práticas de sucesso dos métodos ágeis, são apontadas a aproximação com o cliente, o planejamento constante e de curto prazo, reuniões diárias, uso de histórias, testes automatizados e integração contínua (Ambler, 2010).

A prática de teste automatizado relaciona a criação de testes com o objetivo de verificar de forma automática e recorrente trechos de código da aplicação para que se tenha segurança ao fazer alterações constantes no código, minimizando os efeitos colaterais. A automatização de testes permite, entre outras vantagens, a confiança necessária para desenvolver o software na forma de incrementos pequenos e para a constante refatoração com o intuito de agregar novas funcionalidades e/ou melhorar a qualidade do código (Crispin e Gregory , 2009).

Baseado no teste automatizado o *Extreme Programming* (XP), propõe o *Test Driven Development* (Desenvolvimento Orientado a Teste - TDD), que instrui o programador a escrever o código de produção somente depois que elaborar um caso de teste automatizado que falhe. A idéia básica é orientar as atividades de projeto e codificação pelos critérios

explícitos de correção, tal como representados pelos casos de testes automatizados (Erdogmus e Melnik e Jeffries, 2009). Pesquisas experimentais mostram evidências no aumento da qualidade do software (Janzen e Saiedian, 2008).

Decidiu-se então, estender este desenvolvimento guiado aos requisitos, através de uma técnica chamada de *Behavior Driven Development* (Desenvolvimento Guiado ao Comportamento - BDD) (Hammond e Umphress, 2012). BDD pode ser considerado a extensão do TDD com o objetivo de representar requisitos na forma de diferentes cenários que representam os critérios de aceitação.

Dessa forma, BDD se mostra como um método capaz de sobrepor as dificuldades de ambientes com requisitos dinâmicos, além de agilizar o processo de desenvolvimento. Essa técnica está fundamentada na captura dos requisitos de forma incremental, sendo estes representados na forma de diferentes histórias e cenários embutidos no próprio processo de desenvolvimento e nas ferramentas utilizadas (Solís e Wang, 2011).

Apesar de muito se comentar sobre as vantagens do uso de métodos guiados ao comportamento, muito pouco ainda se conhece sobre os reais benefícios da utilização desses métodos. Esse trabalho se propõe a apresentar um estudo de caso do desenvolvimento de uma aplicação web com o objetivo de comparar o desenvolvimento baseado em documentação e BDD, com o enfoque nas etapas de elucidação e validação dos requisitos. Mais especificamente, são propostos:

- um *framework* de avaliação baseado em mensuração desenvolvido com o apoio da abordagem Goal Question Metric (GQM) (Basili e Caldiera e Rombach, 1994);
- uma investigação comparativa de BDD e um processo orientado a documentação em um estudo de caso envolvendo o desenvolvimento de uma aplicação real;
- análise dos resultados obtidos, ressaltando as vantagens e desvantagens da utilização desses métodos.

No Capítulo 2 deste trabalho são apresentados a fundamentação teórica dos termos e conceitos utilizados. Nele cita-se, uma breve contextualização histórica, as definições relativas aos métodos ditos pesados, bem como aqueles referentes a métodos ágeis. O Capítulo 3 tem por objetivo descrever o estudo de caso, salientando os propósitos, o *framework* de avaliação, o experimento, bem como a sua execução. O próximo capítulo destina-se a demonstrar e analisar os resultados obtidos. O capítulo final tem por finalidade apresentar as conclusões e os trabalhos futuros relacionados a este trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem por objetivo apresentar uma fundamentação teórica dos conceitos relevantes e familiarizar o leitor com os termos significativos a este trabalho. O capítulo inicia com uma breve abordagem histórica. Posteriormente são definidos Modelos de Processos, o Processo Unificado, Modelos de Casos de Uso, Testes e Casos de Teste. Nas seções subsequentes são fundamentados os conceitos de Métodos Ágeis, *Extreme Programming* (XP), Histórias de Usuários, *Behavior Driven Development* (BDD), além da apresentação da ferramenta *JBehave*. Por fim, o leitor é introduzido à técnica de mensuração *Goal Question Metric* (GQM), bem como a algumas considerações finais.

2.1 Contexto Histórico

Apesar de o termo Crise de Software ser utilizado para referir-se aos desafios do desenvolvimento de *software* nos dias de hoje, é na década de 70 que ele foi cunhado. Nesse período a produção de *software* era feita de forma desorganizada, desestruturada e sem planejamento, não existindo documentação e métodos de desenvolvimento. Essas características culminavam no descumprimento do prazo, no alto custo do produto e, em muitos casos, na não satisfação das necessidades do cliente. Neste cenário surgiu a necessidade de processos estruturados, planejados e padronizadas para o desenvolvimento de *software*, visando atender as reais necessidades do cliente e tornar o produto compensador para os desenvolvedores (Presman, 2006). Surgem então, os primeiros métodos de desenvolvimento com a promessa de solucionar os problemas encontrados durante o desenvolvimento do produto. Mesmo utilizando técnicas avançadas de desenvolvimento e padrões consolidados de criação de *software*, características da Crise de Software perduram até hoje.

2.2 Modelos de Processo

Um processo de *software* é um conjunto de atividades e artefatos associados que tem como resultado um produto que reflete o uso de determinado método. Eles foram criados para

tornar a atividade de desenvolvimento menos caótica e visam organizar o desenvolvimento através do uso de técnicas e métodos (Sommerville, 2011).

Embora diferentes processos para o desenvolvimento de *software* sejam conhecidos pela literatura, existem atividades fundamentais comuns a todos eles. Sommerville (2011) destaca as seguintes atividades genéricas:

- Especificação: Tem por objetivo especificar quais serviços são necessários e as restrições que têm impacto no desenvolvimento e operação do sistema. Essa atividade envolve-se com um estudo de viabilidade, identificação, especificação e validação dos requisitos.
- Desenvolvimento: Realiza a conversão da especificação em um sistema executável. A atividade de análise e projeto tem a finalidade de conceber uma estrutura de *software* que atenda aos requisitos especificados através da elaboração de modelos, documentos, histórias, cenários e/ou diagramas. A implementação transforma essa estrutura em um programa executável.
- Verificação e Validação: Destina-se a mostrar que um sistema está em conformidade com a especificação e que atende as necessidades do cliente. Envolve processos de inspeção, revisão e verificação em todas as atividades do processo. Tipicamente, é intensificada após a implementação. Existem diferentes níveis de teste: de componente, de integração, de sistema e de aceitação.
- Evolução: É caracterizada pela necessidade do produto se adequar aos novos requisitos do cliente para continuar sendo útil, agregando novas funcionalidades e melhorias.

Um modelo de processo de *software* é uma representação abstrata de um processo de *software*. Cada modelo representa um processo sob determinada perspectiva, fornecendo somente informações parciais sobre o processo. Esses modelos genéricos, não são descrições definitivas de um processo, mas sim, abstrações que podem ser usadas para explicar diferentes abordagens para o desenvolvimento. Entre os modelos de processo de *software* mais conhecidos estão o sequencial, incremental-iterativo, prototipação, entre outros (Sommerville, 2011).

2.3 Processo Unificado

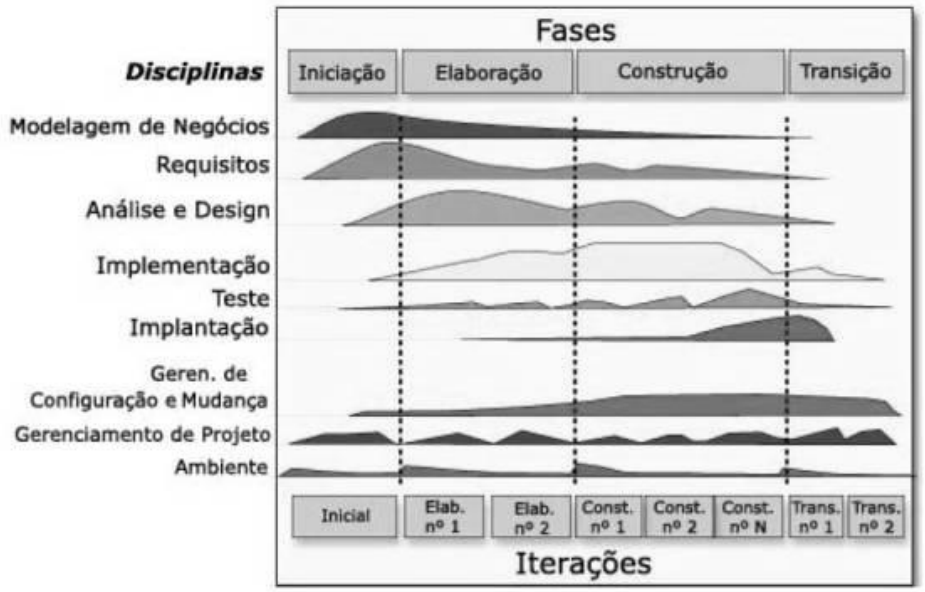
O Processo Unificado é um dos principais processos incrementais iterativos dentre os chamados métodos pesados e que, originalmente, visava a construção de sistemas orientados a objetos. Esse processo prega uma atitude de aceitar a mudança e a adaptação como fatores inevitáveis, progredindo por meio de uma série de ciclos estruturados de construção, realimentação e adaptação.

Originalmente criado pela *Rational® Software* e recentemente mantido pela *Object Management Group* (OMG), o *Rational Unified Process* (RUP) é o principal *framework* baseado no Processo Unificado utilizado comercialmente. Por ser um *framework* de desenvolvimento genérico e trabalhar em ciclos de desenvolvimento, o RUP tem como característica a habilidade de adequá-lo ao tamanho do projeto e a empresa desenvolvedora (Rational, 2001).

No RUP, um projeto de desenvolvimento é organizado em quatro fases: (Figura 2.1):

- Concepção: Essa fase tem como objetivo definir o escopo do projeto, mesmo que de forma genérica ou imprecisa, determinando de forma superficial os esforços e prazos necessários para a execução, bem como a viabilidade do projeto.
- Elaboração: Nessa fase é realizado o detalhamento dos requisitos, iniciando por aqueles que são de maior risco e valor arquitetural. Ao final dessa fase tem-se um detalhamento suficiente do produto para permitir um planejamento mais detalhado da fase de construção.
- Construção: Implementação dos elementos do projeto, tornando cada caso de uso operacional. É ainda nesta etapa que os testes são elaborados, executados ou intensificados.
- Transição: Fase na qual são realizados os testes finais e o produto é colocado à disposição dos usuários para testes beta e relatórios de *feedback* que podem levar a modificações. Informações de apoio como manuais e procedimentos para instalação, quando necessários, também são criados.

Figura 2.1 – Rational Unified Process



Fonte: Rational Unified Process - Best Practices for Software Development Teams (2001, Rational Software White Paper).

Altamente acoplado com a UML, esse processo define atividades, papéis, artefatos e fluxo de trabalho claros organizados em disciplinas (Fig. 2.1). De interesse desse trabalho, a disciplina de requisitos tem o objetivo de identificar os agentes que interagem com o sistema e desenvolver os casos de uso para modelar os requisitos do sistema, tendo como principal artefato o modelo de casos de uso, o qual será utilizado no restante do processo. Também de interesse, a disciplina de teste no RUP, como várias outras disciplinas, é um processo iterativo que envolve artefatos de outras disciplinas. Porém, é ao término da implementação que os testes são intensificados (Sommervile, 2011).

2.4 Modelo de Caso de Uso

Um modelo de caso de uso delimita o sistema no contexto de seu ambiente e estabelece quais entidades, ou seja, atores, deverão interagir com o sistema e quais serão as funcionalidades atendidas pelo mesmo. É fundamentalmente um conjunto de todos os casos de uso, incluindo diagramas e descrições textuais, sendo usado como fonte de informação essencial para as atividades de análise, projeto e teste (Larman, 2004).

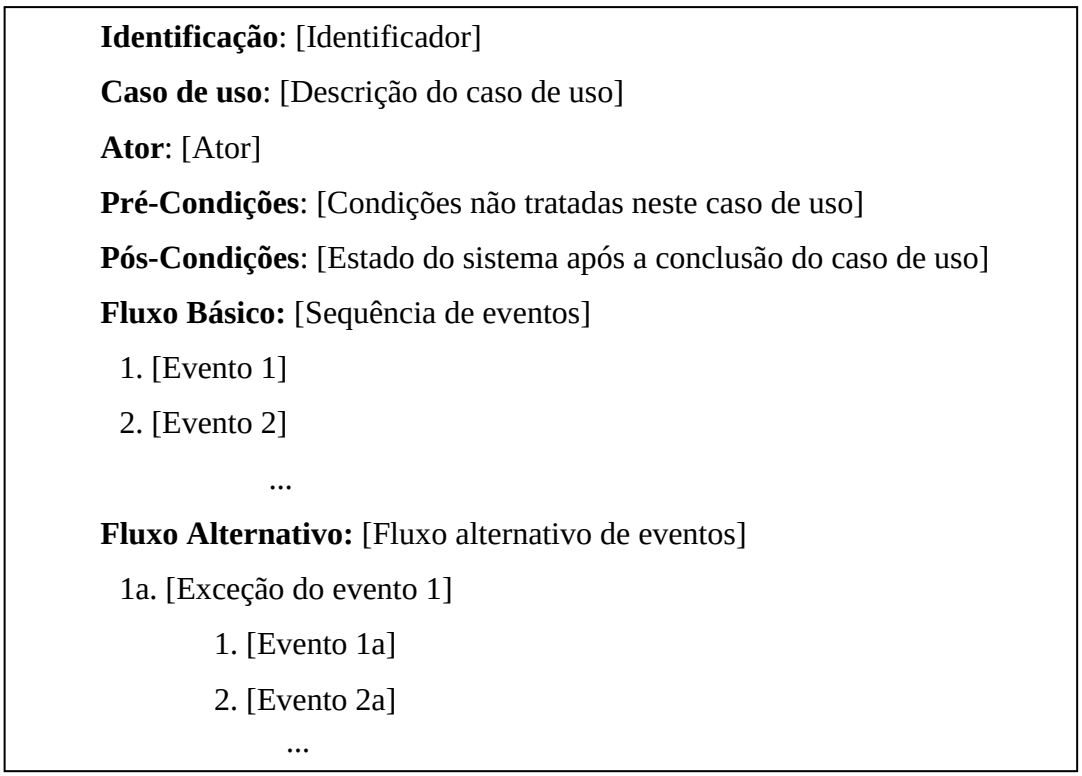
Casos de uso têm por finalidade expressar as funcionalidades do sistema do ponto de vista externo de um ou mais atores, ou seja, como o usuário e o sistema interagem para atingir

determinado objetivo. Um ator é uma entidade (usuário, dispositivo de *hardware* ou mesmo outro sistema) que interage com o sistema. Textualmente, um caso de uso descreve uma funcionalidade completa do sistema, através da definição de uma sequência de ações, incluindo suas variantes, que o sistema e/ou o ator devem executar com o objetivo de produzir como resultado algo de valor tangível para o ator (Larman, 2004).

O fluxo de eventos é composto pelo fluxo principal e por fluxos alternativos e especifica o comportamento de um caso de uso, descrevendo as etapas para execução de determinada funcionalidade. O fluxo principal considera os passos que o ator deve executar para realizar determinada funcionalidade. Os fluxos alternativos abordam o comportamento que desviam da sequência natural de execução, os quais poderão voltar a execução ao fluxo principal ou simplesmente finalizar o caso de uso.

A UML não define uma estrutura específica para o caso de uso textual. Dessa forma, é relevante que possua todas as informações necessárias para o correto desenvolvimento do sistema englobando os requisitos elucidados, bem como facilite a comunicação entre clientes e a equipe de desenvolvimento. Neste trabalho, será adotada uma extensão da notação proposta por Craig Larman, como ilustrado pela Figura 2.2

Figura 2.2 – Estrutura de um Caso de Uso



Fonte: Diogo Reali, 2015

Um caso de uso possui ainda a função de auxiliar a elaboração de casos de teste para o processo de validação do sistema. Desse modo, ele é o principal resultado da etapa de especificação de requisitos, sendo a entrada de várias outras disciplinas (Larman, 2004).

2.5 Testes e Casos de Teste

Teste é a principal atividade para validação do sistema (Sommerville, 2011). Para tanto, é necessária a construção de um plano de teste para guiar as atividades de teste ao longo do processo de desenvolvimento, além da especificação de um conjunto de casos de teste. Baseados na especificação dos requisitos e na estrutura da implementação, casos de teste definem o que deve ser testado, sob quais condições e qual é o resultado esperado.

Testes podem ser executados em diferentes pontos do ciclo de desenvolvimento. Dessa forma, são definidos os diferentes níveis de teste: unitário, de integração, de sistema e de aceitação. De interesse deste trabalho, o teste unitário tem por objetivo a verificação de uma unidade do *software*. Já o teste de sistema tem por finalidade verificar se o sistema desenvolvido satisfaz os requisitos especificados (Sommerville, 2011).

Quanto as técnicas, o teste de *software* pode ser dividido em dois grupos: teste de caixa branca e teste de caixa preta. O primeiro, baseado na análise do código fonte, possibilita que sejam testadas partes específicas da estrutura de um componente de *software*. Testes do segundo tipo, também chamados de testes funcionais, são derivados da especificação do *software*, e por esse motivo, podem ser criados logo que a especificação dos requisitos esteja disponível (Sommerville, 2011). Para este trabalho será dado enfoque nos testes caixa preta.

Para especificação de casos de teste funcionais, inicia-se decompondo a especificação de requisitos em características testáveis de forma independente. Após são eleitos representantes, ou seja, valores representativos para cada entrada, a fim de reunir um conjunto de combinações que garantam a cobertura do teste. Entre as técnicas de derivação de casos de teste citam-se o Particionamento por Equivalência, em que o domínio de entrada do *software* é dividido em classes de dados, e Análise do Valor Limite, complemento da técnica anterior, que visa testar os valores que estão nas fronteiras do domínio da entrada (Pezzè e Young, 2008).

2.6 Métodos Ágeis

Publicado em fevereiro de 2001, o Manifesto Ágil tem por lema “descobrir maneiras melhores de desenvolver *software*, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo”. Nesse manifesto são trazidos os valores de que: (Manifesto Ágil, 2001)

- indivíduos e interações são mais importantes que processos e ferramentas;
- *software* em funcionamento é mais importante que documentação completa e detalhada;
- colaboração com o cliente é mais importante do que negociação de contratos;
- adaptação a mudanças é mais importante do que seguir o plano inicial.

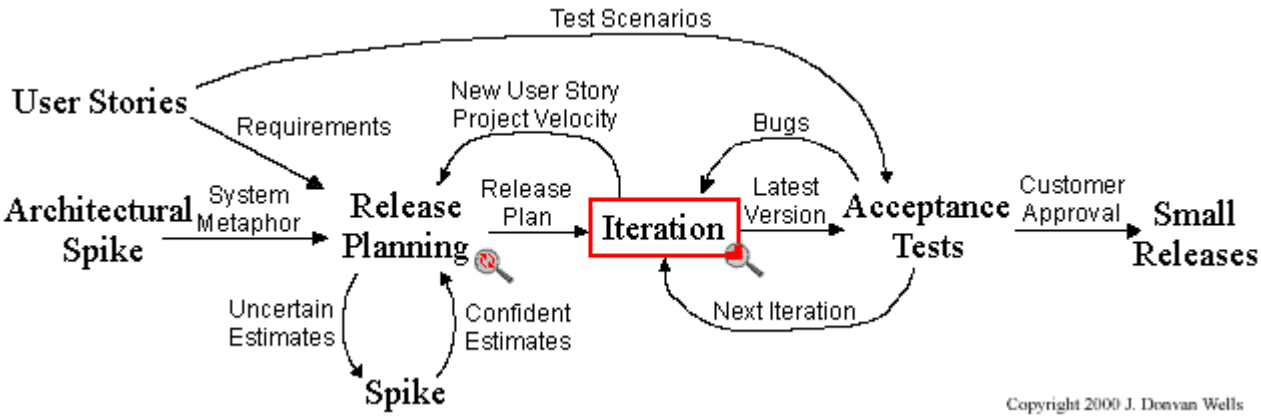
Métodos Ágeis procuram desenvolver *software* com o enfoque nas pessoas (clientes e desenvolvedores), em iterações curtas que possibilitem entregas constantes, acomodação de mudanças nos requisitos em todas as etapas do desenvolvimento, *software* operacional como demonstração de progresso e leveza no processo. Entre alguns dos contrastes com os métodos pesados, pode-se citar o uso de histórias como *placeholder* de conversas entre as partes envolvidas, geração de documentos somente quando agregam valor, enfoque na comunicação interpessoal e ciclos de desenvolvimento bastante curtos, entre duas e quatro semanas, que encerram sempre com a aceitação do incremento pelo cliente. Entre os principais representantes dos métodos ágeis estão o *Extreme Programming* (XP) (Don Wells, 2009) e *SCRUM* (Sutherland e Schwaber, 1995).

2.6.1 Extreme Programming (XP)

O método *Extreme Programming* propõe um conjunto de práticas cujo objetivo é aumentar a previsibilidade e redução de riscos através de ciclos curtos, princípio do cliente presente, a simplicidade e melhorias constantes de código para facilitar a mudança. Entre as principais práticas, de interesse deste trabalho, estão o uso de histórias como forma de representação de requisitos, a integração contínua através de versões pequenas, e o uso de testes automatizados. Além de aumentar a confiança do *software* a partir da execução de verificações automáticas das funcionalidades do sistema buscando possíveis efeitos colaterais oriundos das recentes modificações, a prática de testes automatizados é o mecanismo base do princípio da integração contínua e a origem do *Test Driven Development*.

A Figura 2.3 mostra o ciclo de vida de um projeto utilizando o processo de desenvolvimento XP. Nesse trabalho, serão abordados as histórias de usuário, os testes de aceitação juntamente com os cenários de testes.

Figura 2.3 – Extreme Programming Project



Fonte: http://www.extremeprogramming.org/map/project.html (2000, Don Wells)

2.6.2 Histórias de Usuário

Histórias de usuário expressam uma funcionalidade útil da perspectiva de um usuário ou cliente, delimitando o contexto e descrevendo a interação entre o usuário e o sistema. Elas devem seguir o princípio CCC: Cartão, Conversa e Confirmação. Cartões representam requisitos, ao invés de documentá-los, devendo ser curtos e adaptáveis, e conter estimativas e anotações do requisito (Davies, 2009). A conversa tem por objetivo obter detalhes a respeito da história, podendo resultar em outras histórias. A confirmação tem por finalidade capturar as expectativas do cliente para que seja possível a elaboração de testes de aceitação.

Para este trabalho será utilizado *template* de histórias de usuário proposto por Cohn (2004) e ilustrado na Figura 2.4.

Figura 2.4 – Estrutura de uma História de Usuário

[Título da história] (Breve descrição da história)
As a [Papel]
I want a [Funcionalidade]
So that I can get [Benefício]

Fonte: Diogo Reali, 2015

Entre as vantagens do uso de histórias, destacam-se a capacidade de mudarem o foco da escrita para conversas, enfatizar os objetivos e não os atributos do sistema, estimular o desenvolvimento incremental e iterativo, além de serem compreendidas tanto pela equipe de desenvolvimento quanto pelo cliente (Cohn, 2004).

2.6.3 Test Driven Development (TDD)

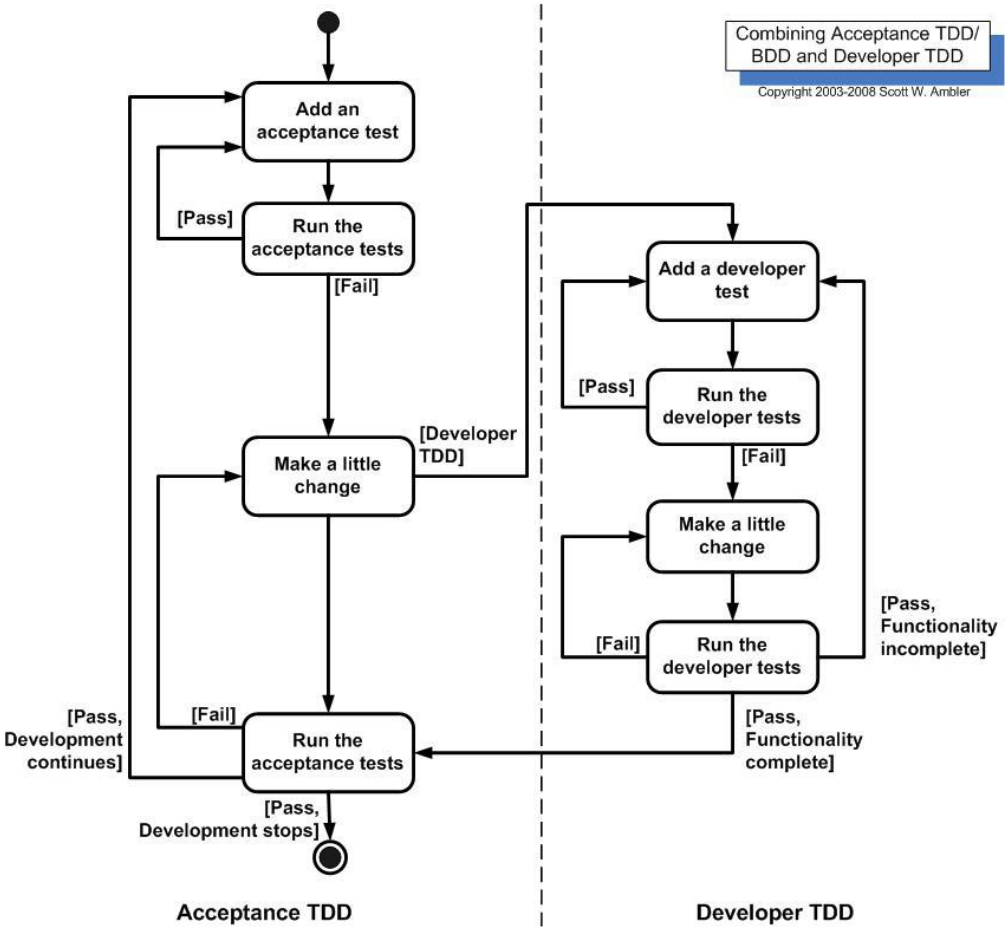
TDD é uma técnica de desenvolvimento de software incremental, cuja principal ideia é atrelar os testes automatizados às atividades de projeto e implementação, servindo como guia para o desenvolvimento. Nessa prática o programador é instruído a escrever uma linha de código do programa somente após elaborar um caso de teste automatizado que falhe. Assim o desenvolvimento pode ser visto como uma sequência de cinco passos:

1) Escrever um novo caso de teste (código de teste);
2) Executar todos os casos de testes e ver o novo teste falhar;
3) Escrever somente o código de produção necessário para fazer passar o caso de teste;
4) Re-executar os casos de teste e ver que todos passaram;
5) Refatorar o código.

Apesar da dificuldade de se obter evidências concretas dos benefícios de TDD, principalmente aquelas relativas a produtividade, pode-se citar entre as vantagens a redução da taxa de falhas, o aumento da qualidade do código, segurança e facilidade para realizar a refatoração do código e correção de falhas, uma vez que, ao re-executar todos os casos de teste a cada inserção de uma funcionalidade e/ou refatoração, e estes passarem, garante-se que o *software* não sofreu alteração do ponto de vista funcional e continua correto (Erdogmus e Melnik e Jeffries, 2011).

Na prática porém, é difícil de entender o processo de desenvolvimento segundo TDD. Entre alguns dos motivos, está o alto acoplamento desse processo com um plano de testes de aceitação que deve ser definido antes da codificação. Essa relação é ilustrada na Figura 2.5.

Figura 2.5 – Combinação de Testes de Aceitação e TDD



Fonte: Scott W. Ambler, 2008

A partir de então, surge a ideia de integrar na prática de testes automatizados o comportamento externo desejado do sistema, fazendo com que não sejam somente testes de validação altamente acoplados à implementação, mas sim, a própria especificação dos requisitos. Essa prática é conhecida como *Behavior Driven Development*.

2.7 Behavior Driven Development (BDD)

Proposto inicialmente em 2003 por Dan North, *Behavior Driven Development* (BDD) é conhecido como a evolução do TDD. Essa técnica está focada em pequenas especificações do comportamento do sistema, de uma maneira que elas possam ser automatizadas. O entendimento de BDD está longe de ser claro e unânime, não tendo na literatura uma definição bem aceita (North, 2008). Pode-se dizer que o principal objetivo de BDD é garantir automaticamente que todos os requisitos funcionais sejam tratados adequadamente pela implementação, por meio da ligação da descrição textual dos requisitos com testes automatizados. Os testes são claramente escritos e facilmente compreensíveis, através do uso de uma linguagem ubíqua que contém um vocabulário compartilhado entre as partes envolvidas no projeto, criando um entendimento comum do domínio (Solís e Wang, 2011).

Em BDD a atividade de especificação inicia com a identificação do comportamento esperado do sistema. O conjunto de requisitos pode ser obtido, por exemplo, através de discussões entre clientes e desenvolvedores sobre os resultados esperados do sistema. Estes requisitos são descritos posteriormente por histórias de usuários. As instâncias específicas de uma história são chamadas de cenários, ou seja, elas descrevem o comportamento do sistema em um estado específico quando um evento acontece (Solís e Wang, 2011). Na descrição de histórias e cenários, BDD faz uso de *templates* pré-definidos. Nesse trabalho optou-se por adotar o *template* de cenário mostrado pela Figura 2.6:

Figura 2.6 – Estrutura de um Cenário

Scenario 1: [Título do cenário]
Given [Contexto]
And [Outros contextos] ….
When [Evento]
Then [Resultado]
Scenario 2: [Título do cenário] ….

Fonte: Diogo Reali, 2015

O contexto de um cenário descreve o estado da aplicação antes do início da execução do comportamento especificado pelo cenário. O evento descreve a ação que inicia a execução

do cenário. Finalmente, o resultado descreve as alterações esperadas após a execução do cenário (Martin Fowler, 2013).

Testes de aceitação em BDD são especificações do comportamento do sistema, que têm por objetivo verificar as interações dos objetos e não somente seus estados. Os cenários derivados das histórias são traduzidos em testes automatizados que guiarão o desenvolvimento. Em outras palavras, BDD permite ter cenários de texto simples executáveis.

2.7.1 JBehave

Entre os diversos frameworks de desenvolvimento voltados ao BDD, está o *JBehave*[1]. Muitas de suas ideias têm aparecido em outras ferramentas, como é o caso do *RSpec*[2]. No *JBehave*, uma história de usuário é um arquivo de texto contendo um conjunto de cenários. Utilizando-se de *JUnit4*[3] para a execução dos testes automatizados, este *framework* possibilita a utilização de parâmetros para a validação dos cenários, além de proporcionar uma visualização fácil e limpa dos resultados da execução, mostrando os cenários que falharam, os que obtiveram sucesso, e aqueles que ainda não foram implementados (North, 2008).

A estrutura textual de uma história de usuário e cenários utilizados pela ferramenta *JBehave* é a mesma da adotada por esse trabalho (Figura 2.4 e Figura 2.6). Para a execução dos testes, a ferramenta faz uso de anotações *@Given*, *@When* ou *@Then* para indicar a que cláusula do cenário o teste automatizado anotado se refere.

2.8 Goal Question Metric (GQM)

O desenvolvimento de *software* requer um mecanismo de medida que auxilie no planejamento do projeto, determine os pontos fortes e fracos do processo e que permita avaliar as técnicas utilizadas. Proposto no final da década de 80, *Goal Question Metric* (GQM) assume que só há propósito em medir se uma organização, em primeiro lugar, definir seus objetivos, mapeá-los em termos de dados que traduzem estes objetivos em nível

[1] jbehave.org/

[2] http://rspec.info/

[3] junit.org/

operacional e, por fim, prover um arcabouço para interpretação dos dados de acordo com os objetivos. Dessa forma, o modelo de resultados possui três níveis (Basili e Caldiera e Rombach, 1994):

- Conceitual (Goal): Define os principais objetivos da medição. Um objetivo é definido para um objeto com um propósito específico, segundo um enfoque de qualidade e um ponto de vista, relativo a um ambiente específico.
- Operacional (Question): De acordo com os objetivos, definem-se perguntas cujas respostas dirão se os objetivos estão sendo alcançados. Questões tentam caracterizar o objeto da medição de acordo com o aspecto escolhido de qualidade e ponto de vista.
- Quantitativo (Metric): Definem-se as métricas que possam responder as questões, podendo ser medidas a partir de dados objetivos ou subjetivos.

A Figura 2.7 mostra um exemplo de aplicação de GQM, onde o objetivo é diminuir o custo de desenvolvimento para realizar uma alteração no sistema durante a fase de manutenção, e decidir quais métodos se adequam melhor ao processo.

Figura 2.7 – Exemplo de uma Aplicação de GQM

Goal	Diminuir o custo para alteração do sistema sob o ponto de vista do gerente de projeto.
Question Q1	Qual é o tempo atual para a modificação de uma funcionalidade?
Metrics M1	Número de ciclos médios.
M2	Desvio padrão.
M3	Porcentagem de casos acima do limite.
Question Q2	A performance do processo aumentou?
Metrics M4	Número de ciclos atuais / Patamar médio de ciclos

A definição de um *framework* de avaliação baseado em GQM é proposto na Seção 3.2, sendo utilizado para a análise das técnicas de desenvolvimento propostas pelo trabalho.

2.9 Considerações Finais

Este capítulo apresentou uma fundamentação teórica dos conceitos relevantes para a compreensão deste trabalho. A utilização desses métodos pelo experimento proposto, bem como suas adaptações, são descritos no Capítulo 3 deste trabalho.

3 ESTUDO DE CASO

Nesse capítulo é abordado o estudo de caso do desenvolvimento de uma aplicação *web* com o intuito de comparar a implementação baseada no Processo Unificado com os métodos voltados a BDD. Na primeira seção são delimitados os objetivos do estudo. Na seção subsequente é definido um *framework* baseado na mensuração do desenvolvimento segundo cada um dos métodos utilizados. Por fim, o experimento desenvolvido e sua execução são descritos.

3.1 Propósitos

Muito se discute sobre as vantagens do uso de BDD, porém pouco ainda se sabe sobre os reais benefícios de sua utilização. Entre as dificuldades estão a falta de uma definição formal amplamente aceita pela comunidade científica, o relacionamento das práticas utilizadas com ferramentas específicas, culminando na falta de evidências concretas das vantagens e desvantagens do uso de métodos guiados ao comportamento.

Como uma contribuição a este problema, esse trabalho objetiva realizar uma análise comparativa entre o desenvolvimento inspirado no Processo Unificado e BDD. Este trabalho concentra-se na forma de representação dos requisitos e critérios de aceitação em cada um dos casos, e no esforço para a produção do *software* que atenda esses critérios. Mais precisamente, busca-se avaliar qual processo exige menor esforço sob o ponto de vista do desenvolvedor, no que diz respeito ao desenvolvimento de uma nova funcionalidade, bem como no retrabalho necessário para correção de falhas encontradas no sistema. Além disso, objetiva-se comparar a qualidade do *software* entregue a partir da implementação segundo cada um dos métodos, ressaltando o número de falhas encontradas e a satisfação do cliente.

Nesse trabalho, optou-se por realizar a análise comparativa entre esses dois processos segundo uma abordagem quantitativa. Por esse motivo, é proposto um *framework* com base em GQM para a mensuração do desenvolvimento segundo cada um desses métodos. A análise dar-se-á a partir do desenvolvimento de uma aplicação web de caráter comercial segundo os dois processos. Por se tratar de uma única aplicação, esta será dividida em partes equivalentes, cujo o desenvolvimento de cada uma delas seguirá um desses processos.

Com base nos resultados mensurados através do *framework,* será feita uma análise explicitando as vantagens, bem como as desvantagens da utilização de cada um dos dois métodos. Tal abordagem permite que uma equipe de desenvolvimento possa optar pelo processo que melhor se adeque as suas necessidades e, consequentemente, otimize a qualidade das entregas e os custos.

3.2 Experimento

O experimento consiste no desenvolvimento de uma aplicação *web* que objetiva o controle de estoque de uma loja de comércio de pneus e peças para carros. O desenvolvimento da aplicação foi dividido em módulos, os quais foram decompostos em funcionalidades que representam as quatro operações básicas de um sistema: consulta, inserção, alteração e exclusão. Para que seja possível a comparação entre o desenvolvimento segundo cada processo, agrupou-se essas funcionalidades em dois conjuntos distintos de mesmo porte os quais representam o desenvolvimento segundo cada um dos processos.

Com o objetivo de coletar os dados necessários para a construção das métricas e responder as questões propostas pelo *framework* de avaliação, realizou-se a mensuração dos esforços necessários para cada uma das atividades envolvidas no processo. Os resultados obtidos, bem como sua análise serão abordados no Capítulo 4.

3.2.1 Aplicação

A aplicação destina-se auxiliar o proprietário e funcionários de uma loja do ramo de autopeças nas operações relacionadas a compra e venda de peças. O *software* proposto tem como principais objetivos:

a) Otimizar o controle e a utilização do estoque;
b) Permitir o cadastro de fornecedores e clientes facilitando a recuperação das informações cadastrais;
c) Detalhamento das vendas e compras efetuadas e dos produtos envolvidos.

Para que as necessidades do cliente sejam satisfeitas, a aplicação é composta por cinco módulos relacionados a diferentes dados e operações. São eles:

- Clientes: Concentra a base cadastral dos clientes do estabelecimento, contendo entre outras informações, o nome, CPF/CNPJ, endereço, telefone e e-mail do cliente. Permite as operações inclusão, alteração, exclusão e detalhamento dos dados cadastrais. Possibilita também a pesquisa por clientes a partir do nome e CPF/CNPJ. Associa ainda uma lista de veículos pertencentes a cada cliente.
- Fornecedores: Bastante semelhante ao módulo de clientes, este contém as informações referentes aos fornecedores do estabelecimento. Permite as operações de inclusão, alteração, exclusão, detalhamento e pesquisa.
- Produtos: Este módulo tem por objetivo armazenar as informações referentes aos produtos comercializados pelo estabelecimento. Contém entre outras informações, o código de referência, a descrição, o fabricante e o total em estoque do produto. Além das operações de inclusão, alteração, detalhamento e exclusão, permite a pesquisa por referência, descrição e fabricante.
- Compras: Contém as informações referentes às compras realizadas, relacionando cada fornecedor a uma lista de produtos comprados, com as respectivas quantidades e valores. Permite somente as operações de inclusão, detalhamento e pesquisa. As consultas podem ser baseadas no fornecedor e no dia em que se realizou a compra.
- Vendas: Responsável por manter as informações referentes as vendas realizadas, associando a um cliente e veículo uma lista dos produtos vendidos. Permite as operações de inclusão, detalhamento e pesquisa. As pesquisas podem ser baseadas nas informações de cliente e data da venda.

No que diz respeito a arquitetura de implementação, a aplicação foi desenvolvida em linguagem de programação orientada a objeto, precisamente *Java*[4] e baseada no padrão *MVC*[5] (*Model-View-Controller*). Utiliza-se de *JSF2*[6] e *Primefaces*[7] para a *interface*, e

[4] https://www.java.com/pt_BR/

[5] Reenskaug, 1978

[6] https://javaserverfaces.java.net/

[7] www.primefaces.org/

JPA[8]/*Hibernate*[9] para persistência em banco de dados objeto relacional *Postgres*[10]. O ambiente de desenvolvimento foi o *Eclipse Luna*[11]

3.2.2 Processos Adotados

Para o desenvolvimento da aplicação proposta foram escolhidos dois processos nos quais será dado enfoque às atividades de expressão dos requisitos e dos critérios de aceitação, e de como estes orientam a produção do *software* de qualidade. As definições dos métodos utilizados por cada um dos processos, bem como a exemplificação do seu uso são abordados a seguir.

A identificação dos requisitos foi uma atividade comum aos dois processos como forma do compreensão da aplicação. Por esse motivo, ela não será abordada em cada um dos processos. Esse trabalho reconhece que idealmente as interações com o cliente deveriam ser diferentes em cada um dos processos devido ao desenvolvimento cíclico, porém não foi possível devido a difícil disponibilidade do cliente para os contatos posteriores.

A atividade de elucidação dos requisitos baseou-se em uma entrevista com o cliente e na observação das tarefas executadas pelo mesmo. Pontos importantes como o objetivo do sistema, funcionalidades desejadas, quais são os dados importantes para serem manipulados, bem como as dificuldades do modelo do negócio atual foram abordados.

Apesar dos processos adotados serem idealmente incrementais, não foi possível a demonstração de cada incremento para o cliente, ao contrário da expectativa inicial quando este estudo de caso foi proposto. Assim, a demonstração da aplicação ocorreu apenas ao término do desenvolvimento de todas as funcionalidades propostas. Nesse momento, a partir da demonstração e do esclarecimento com o cliente, foram identificadas as funcionalidades que não satisfaziam as suas reais necessidades e que por esse motivo deveriam ser corrigidas.

[8] http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

[9] www.hibernate.org/

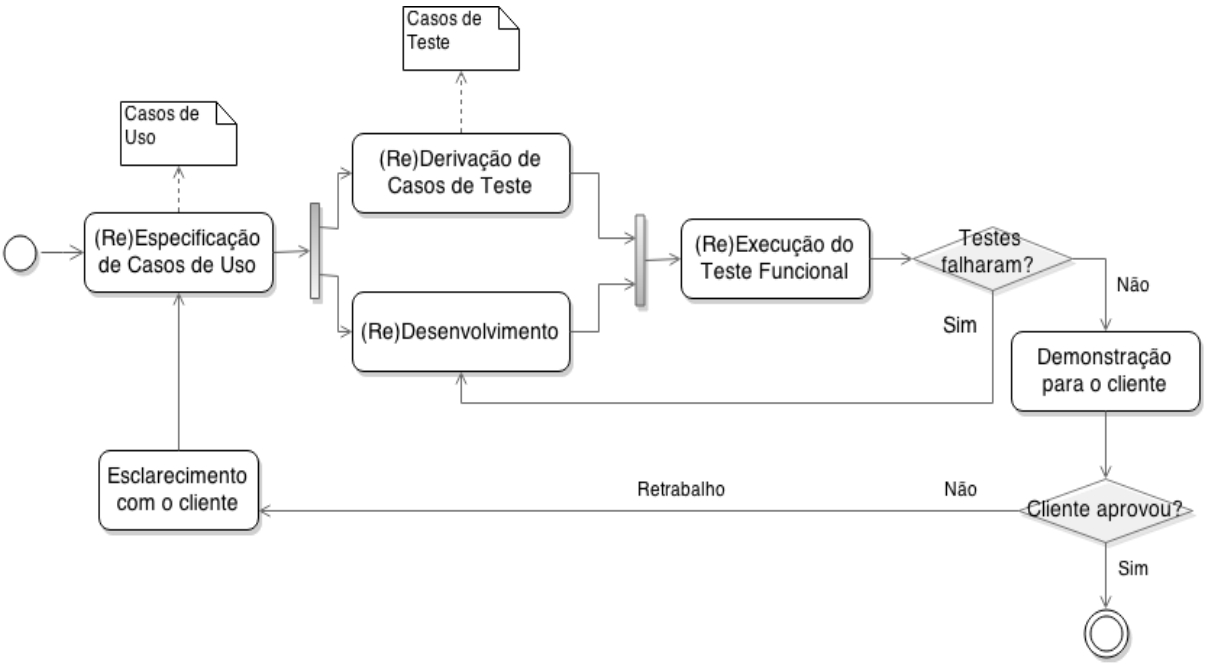
[10] www.postgresql.org/

[11] https://eclipse.org/

3.2.2.1 Processo Inspirado no Processo Unificado

O processo adotado, inspirado no Processo Unificado, é composto pela especificação de casos de uso, implementação, derivação de casos de teste e execução de testes funcionais. O processo utilizado é ilustrado pela Figura 3.1, e detalhado a seguir. Salienta-se que o processo é significativamente mais simples que a proposta completa do Processo Unificado e que, embora ele defina papeis diferentes para as disciplinas de Requisitos e Teste, no presente trabalho essas disciplinas foram executadas sempre pela mesma pessoa. Essas adaptações refletem o contexto típico de desenvolvimento de *software* em empresas de pequeno porte.

Figura 3.1 – Processo Utilizado baseado no Processo Unificado



Fonte: Diogo Reali, 2015

a) Especificação dos Casos de Uso: Uma vez que os requisitos foram identificados, o processo inicia-se com a elaboração de um modelo de casos de uso. Após a identificação dos atores e casos de uso, estes são descritos textualmente através da elaboração de casos de uso demonstrados na Seção 2.4. Cada descrição de casos de uso contém pré e pós condições da funcionalidade em questão e o fluxo de execução para determinada tarefa. Um exemplo de caso de uso utilizado durante o desenvolvimento da aplicação que objetiva o cadastro de um produto é apresentado no Apêndice A. Este artefato guiará a atividade de

implementação e de validação do sistema através da derivação de casos de teste. Nesse trabalho não ocorreu a validação com o cliente da documentação gerada.

b) Desenvolvimento: Após a conclusão da especificação de todos os requisitos para as funcionalidades desejadas, inicia-se a implementação do *software* com base nas informações descritas nos casos de uso. A execução dessa atividade objetiva tornar a tarefa descrita pelo caso de uso funcional e contemplar todos os requisitos presentes nesse artefato. Nesse processo, o desenvolvimento não é atrelado a elaboração de nenhum tipo de teste automatizado, e por esse motivo será chamado de desenvolvimento puro. A validação dos requisitos será de responsabilidade somente do teste funcional, sendo realizada ao término do desenvolvimento.

c) Derivação dos Casos de Teste Funcionais: Paralelamente à implementação, é realizada a derivação de casos de teste a partir da especificação dos casos de uso. Baseados nas técnicas de Particionamento por Equivalência e Análise por Valor Limite, são elaborados documentos que objetivam guiar os testes funcionais, descrevendo a entrada de dados para cada um dos casos de uso e o comportamento esperado do sistema. O Apêndice B mostra um exemplo de documento de casos de teste derivados a partir do caso de uso abordado pelo Apêndice A.

d) Execução dos Testes Funcionais: Ao término da implementação e da elaboração dos casos de teste é possível a execução dos testes funcionais da aplicação. Nessa atividade, os testes são executados através da manipulação do sistema segundo o documento de casos de teste. Ou seja, são realizadas operações e entradas de dados na aplicação a fim de comparar o comportamento definido pelo caso de teste com aquele observado na execução do sistema. Caso a totalidade de casos de teste de determinada funcionalidade obtenha sucesso, ou seja, o comportamento esperado é igual ao observado, diz-se que a implementação satisfaz os requisitos elencados por este caso de uso e o desenvolvimento deste é dado por encerrado. Se um ou mais testes falharem, o fluxo do processo retorna para a atividade de desenvolvimento.

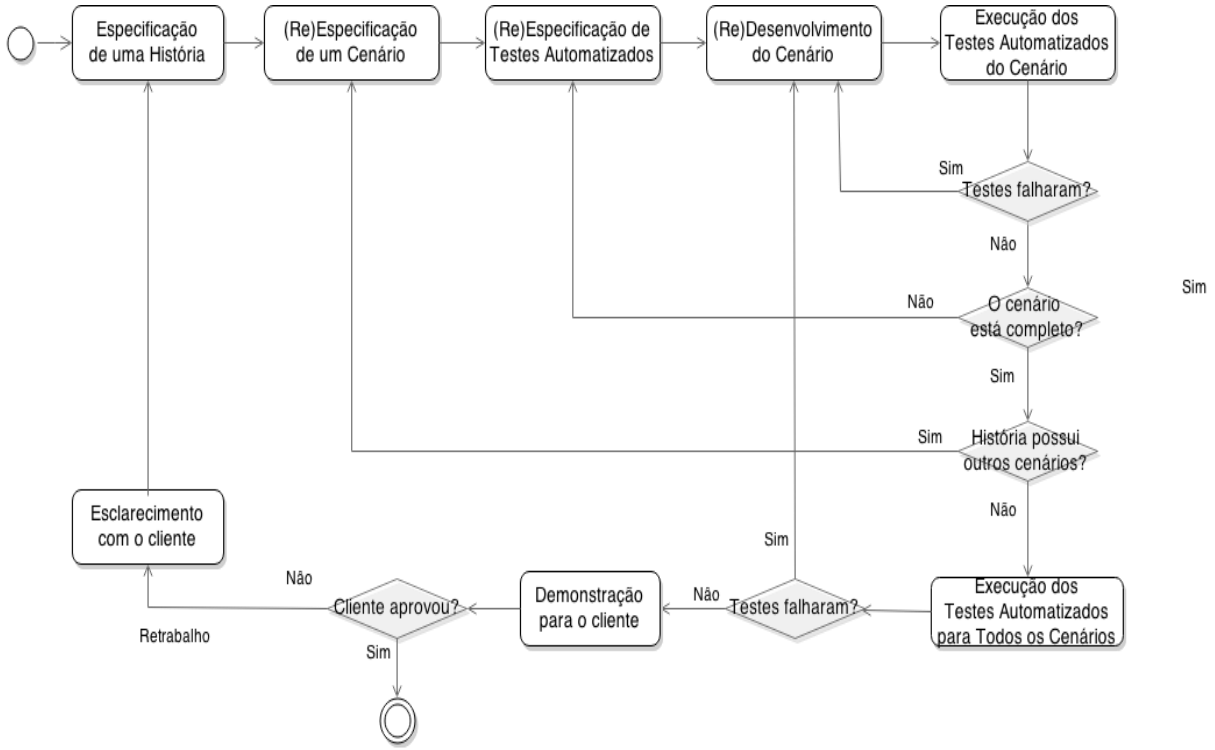
e) Correção: O retrabalho é o conjunto de atividades necessárias para que a aplicação satisfaça as necessidades do cliente para as funcionalidades entregues que não alcançaram os objetivos esperados por ele. Dessa forma, inicia-se com o

esclarecimento da funcionalidade com o cliente, a re-especificação do caso de uso, a reformulação de casos de teste e o redesenvolvimento. Por fim, a re-execução dos testes funcionais em sua plenitude é obrigatória para a garantia de que todos os requisitos, mesmo aqueles que já obtiveram sucesso nos testes anteriores, estejam em conformidade com o caso de uso.

3.2.2.2 BDD

O processo adotado de desenvolvimento guiado pelo comportamento é composto por entrevista com o cliente, a especificação de histórias, cenários, testes automatizados, desenvolvimento e execução dos testes. O ciclo de desenvolvimento de uma história é constituído pelas atividades de especificação de um cenário, desenvolvimento com teste automatizado do cenário e execução dos testes. As atividades envolvidas são mostradas pela Figura 3.2, e descritas a seguir.

Figura 3.2 – Processo Utilizado segundo BDD



Fonte: Diogo Reali, 2015

a) Especificação da História: Uma vez identificados os requisitos, estes são expressos através de histórias descritas no formato discutido na Seção 2.6.2. A atividade de especificação das histórias tem por finalidade definir uma funcionalidade desejada, o usuário envolvido, e o benefício que o desenvolvimento desta *feature* traz ao usuário. O Apêndice C demonstra uma história para a necessidade do cliente de poder alterar os produtos cadastrados.

b) Especificação dos Cenários: Os cenários derivados a partir de uma história, descrevem o comportamento do sistema em um estado específico quando um evento acontece. No ciclo de desenvolvimento adotado por esse processo, especifica-se somente um cenário por iteração. O Apêndice D mostra todos os cenários derivados da história abordada anteriormente ao final do processo de desenvolvimento.

c) Elaboração dos Testes Automatizados: Como o desenvolvimento continua de forma integrada com TDD, o cenário guia a elaboração dos diferentes casos de testes automatizados (i.e. código de teste), os quais, juntos, guiarão o processo de desenvolvimento. Inicialmente, cada teste definido deve falhar por não ainda não possuir o código de produção. Além disso, os testes automatizados são importantes para a contínua integração de novas funcionalidades ainda não elucidadas, bem como para a validação de requisitos. A implementação dos testes automatizados para o cenário especificado anteriormente é mostrada no Apêndice E.

d) Desenvolvimento: Consiste no desenvolvimento de código de produção correspondente aos casos de teste especificados. Ele objetiva modificar o estado da execução dos testes automatizados relativos ao cenário em questão, fazendo-os passar de um estado de falha para o de sucesso. Dessa forma, o desenvolvimento só é dado por encerrado quando todos os testes automatizados definidos para o cenário possuírem um código de produção que obtenha sucesso para o código teste. Enquanto isso não ocorrer, o ciclo de desenvolvimento e execução dos testes automatizados é repetido.

e) Execução dos Testes Automatizados do Cenário: São executados os testes automatizados que guiaram o processo de desenvolvimento do cenário. Até que todos os casos de teste do cenário não possuam um código de produção

que os satisfaça, o fluxo do processo retorna para a atividade de desenvolvimento.

f) Execução dos Testes Automatizados para Todos os Cenários: No momento em que todos os testes automatizados do cenário obtiverem sucesso e que não existirem outros cenários para serem desenvolvidos, pode-se então executar os testes automatizados para todos os cenários da aplicação. Caso algum deles falhe, principalmente devido ao conflito para a integração das funcionalidades implementadas recentemente com as existentes, o fluxo de execução do processo retorna para a atividade de desenvolvimento. Caso contrário, um novo ciclo de desenvolvimento é iniciado a partir da especificação de uma nova história de usuário.

g) Correção: No processo guiado por comportamento adotado por esse trabalho, chama-se de retrabalho o conjunto de atividades envolvidas para a alteração das funcionalidades da aplicação entregues a fim de satisfazer as necessidades do cliente. Assim, inicia-se com o esclarecimento da funcionalidade com o cliente, a re-especificação da história e cenários, a reformulação dos testes automatizados, redesenvolvimento e a re-execução dos testes automatizados de determinado cenário. A re-execução de todos os testes automatizados e a obtenção do sucesso para todos é obrigatória para a garantia de que todos os requisitos estejam em conformidade com a especificação. Esse processo é ilustrado pela Figura 3.2.

3.3 Framework de Avaliação

Para que seja possível a comparação entre as duas abordagens de desenvolvimento, foi elaborado um *framework* de avaliação para mensurar os resultados de interesse deste trabalho. De forma que, o *framework* permite refletir sobre as vantagens e desvantagens de se utilizar cada um desses processos.

Elaborado com base em GQM, o *framework* tem a finalidade de oferecer dados objetivos referentes aos aspectos de desenvolvimento do *software* segundo o Processo Unificado e BDD, indicando qual deles exige menor esforço de desenvolvimento sob o ponto de vista do desenvolvedor. Para que o objetivo seja alcançado, é necessária a elaboração de

questões que ressaltem as características de cada processo, tais como o esforço médio para a entrega de uma funcionalidade e/ou para correção de uma entrega, o número de erros encontrados e de funcionalidades que tiveram retrabalho. Apesar de também ser de interesse desse trabalho, não foi possível realizar a mensuração das quantidades de *bugs* encontrados nas funcionalidades entregues devido ao cliente não ter disponibilidade para utilizar a aplicação de maneira mais intensa. Nesse trabalho, entende-se como esforço o número de horas trabalhadas em determinada atividade. Considera-se como funcionalidade, uma operação de inserção, alteração, exclusão ou consulta observadas do ponto de vista do usuário.

A fim de responder as questões propostas e realizar a comparação, é preciso que um conjunto de métricas seja definido. O *framework* contém métricas diretas, coletadas com base em observações, e indiretas, calculadas a partir de outras métricas. Além disso, medidas subjetivas como a satisfação do usuário também foram consideradas.

O *framework* de avaliação é exposto a seguir:

Goal	Analisar qual processo exige menor esforço de desenvolvimento do *software* sob o ponto de vista do desenvolvedor

Question	**Q1**	Qual o esforço médio para entrega de uma funcionalidade usando BDD?
Metrics	**M1**	**Número de funcionalidades (inserção, remoção, alteração, consulta)**
	R1	**Esforço médio de desenvolvimento com BDD:** $(\sum Bi) / M1$
	B_i	**Esforço de Desenvolvimento por funcionalidade com BDD:** Esforço de elucidação de requisitos + esforço de especificação de histórias + esforço de especificação dos cenários + esforço de desenvolvimento com teste automatizado + BT_i
	BT_i	**Esforço de Retrabalho com BDD:** Esforço de esclarecimento com o cliente + esforço de re-especificação da história + esforço de alteração e/ou criação de cenários + esforço de (re)desenvolvimento

Question	**Q2**	Qual o esforço médio para entrega de uma funcionalidade usando o Processo Unificado?
Metrics	**R2**	**Esforço médio de desenvolvimento com Processo Unificado: (**$\sum$ Di) / M1
	D_i	**Esforço de Desenvolvimento por funcionalidade com Processo Unificado:** Esforço de elucidação de requisitos + esforço de especificação do caso de uso + esforço de especificação dos casos de teste + esforço de desenvolvimento (puro) + esforço de execução do teste funcional + DT_i
	DT_i	**Esforço de Retrabalho com Processo Unificado:** Esforço de esclarecimento com o cliente + esforço de re-especificação do caso de uso + esforço de re-especificação dos casos de teste + tempo de (re)desenvolvimento + esforço de re-execução do teste funcional

Question	**Q3**	Quantas funcionalidades entregues tiveram retrabalho usando BDD?
Metrics	**NB**	Quantidade de funcionalidades entregues que tiveram retrabalho com BDD

Question	**Q4**	Quantas funcionalidades entregues tiveram retrabalho usando o Processo Unificado?
Metrics	**NU**	Quantidade de funcionalidades entregues que tiveram retrabalho com Processo Unificado

Question	**Q5**	Qual é o esforço médio para uma correção de uma funcionalidade usando BDD?
Metrics	**R5**	($\sum$ BTi) / NB

Question	**Q6**	Qual é o esforço médio para uma correção de uma funcionalidade usando o Processo Unificado?
Metrics	**R6**	($\sum$ DTi) / NU

Question	**Q7**	Qual é a proporção do esforço para correção de uma funcionalidade entregue sobre o esforço total usando BDD?
Metrics	**R7**	($\sum$ BT_i) / ($\sum$ B_i)

Question **Q8**	Qual é a proporção do esforço para correção de uma funcionalidade entregue sobre o esforço total usando Processo Unificado?
Metrics **R8**	$(\sum DT_i) / (\sum D_i)$

Question **Q9**	Quantos erros em funcionalidades entregues usando BDD foram encontrados?
Metrics **R9**	Quantidade de erros encontrados em funcionalidades entregues usando BDD

Question **Q10**	Quantos erros em funcionalidades entregues usando o Processo Unificado foram encontrados?
Metrics **R10**	Quantidade de erros encontrados em funcionalidades entregues usando o Processo Unificado

Question **Q11**	As funcionalidades entregues usando BDD são satisfatórias para o cliente?
Metrics **R11**	Satisfação do usuário (nota 0-10)

Question **Q12**	As funcionalidades entregues usando o Processo Unificado são satisfatórias para o cliente?
Metrics **R12**	Satisfação do usuário (nota 0-10)

3.4 Execução

A execução do experimento teve início a partir do contato com o cliente para a apresentação da proposta de uma aplicação *web* para auxiliar na gerência da loja. Uma vez que o cliente aceitou participar, realizou-se uma entrevista com o mesmo, bem como a observação das atividades executadas a fim de entender suas reais necessidades.

Para que seja a possível a comparação entre o Processo Unificado e BDD dividiu-se o conjunto de funcionalidades desejadas pelo cliente em dois grupos, de forma que, o número de funcionalidades pertencentes a cada um os conjuntos fosse o mesmo. Por se tratar de operações semelhantes, a complexidade dois conjuntos definidos é aproximadamente a mesma. O conjunto de funcionalidades definidos para cada um dos processos são elencados a na Tabela 3.3.

Tabela 3.3 – Conjunto de Funcionalidades para cada um dos Processos de Desenvolvimento

Conjunto de Funcionalidades Usando o Processo Unificado	Conjunto de Funcionalidades usando BDD
Cadastrar Produto	Cadastrar Fornecedor
Alterar Fornecedor	Alterar Produto
Consultar Produto	Consultar Fornecedor
Excluir Fornecedor	Excluir Produto
Cadastrar Cliente	Alterar Cliente
Excluir Cliente	Consultar Cliente
Associar Veículo	Remover Veículo
Cadastrar Compra	Cadastrar Venda
Consultar Venda	Consultar Compra
Detalhar Venda	Detalhar Compra

Paralelamente à definição dos dois conjuntos, especificou-se a arquitetura da aplicação web. Realizaram-se alguns testes antes de iniciar o desenvolvimento, com o objetivo de descartar a influência do aprendizado da tecnologia utilizada na comparação entre os dois processos, bem como garantir que a arquitetura suporte as funcionalidades desejadas. Para BDD, realizou-se a elaboração de uma história de usuário, seus cenários, bem como testes automatizados a partir da ferramenta *JBehave*. Para o Processo Unificado, efetuou-se a elaboração de um documento de caso de uso e seus respectivos casos de teste. Em ambos os processos, essas atividades foram discutidas e refeitas até que houvesse o alinhamento do que seria adotado em cada um dos processos.

Uma vez que os dois conjuntos de funcionalidades foram delimitados e a arquitetura da aplicação foi definida, pôde-se iniciar o processo de desenvolvimento de cada conjunto. Realizou-se o desenvolvimento de cada uma das funcionalidades de maneira intercalada entre os dois processos, além disso todas as atividades foram executadas pelo mesma pessoa. Durante o processo, os esforços necessários para a execução de cada uma das atividades que compõem cada um dos processos de desenvolvimento foram medidos e anotados a fim de compor a base de dados que irá permitir a comparação entre os dois processos. Os resultados obtidos e sua análise serão mostrados no capítulo seguinte.

4 RESULTADOS

Este capítulo destina-se a apresentar os resultados obtidos a partir do experimento proposto por esse trabalho. No decorrer deste capítulo, é realizada também uma análise comparativa entre os dois processos utilizando-se das métricas coletadas.

4.1 Esforço Total de Desenvolvimento

As duas primeiras perguntas, Q1 e Q2, do *framework* de avaliação da Seção 3.3 buscam entender qual processo necessita de um menor esforço para o desenvolvimento de uma funcionalidade. Estes dados são ilustrados pela Tabela 4.1.

Tabela 4.1 – Esforço Total para o Desenvolvimento de uma Funcionalidade Segundo cada Processo

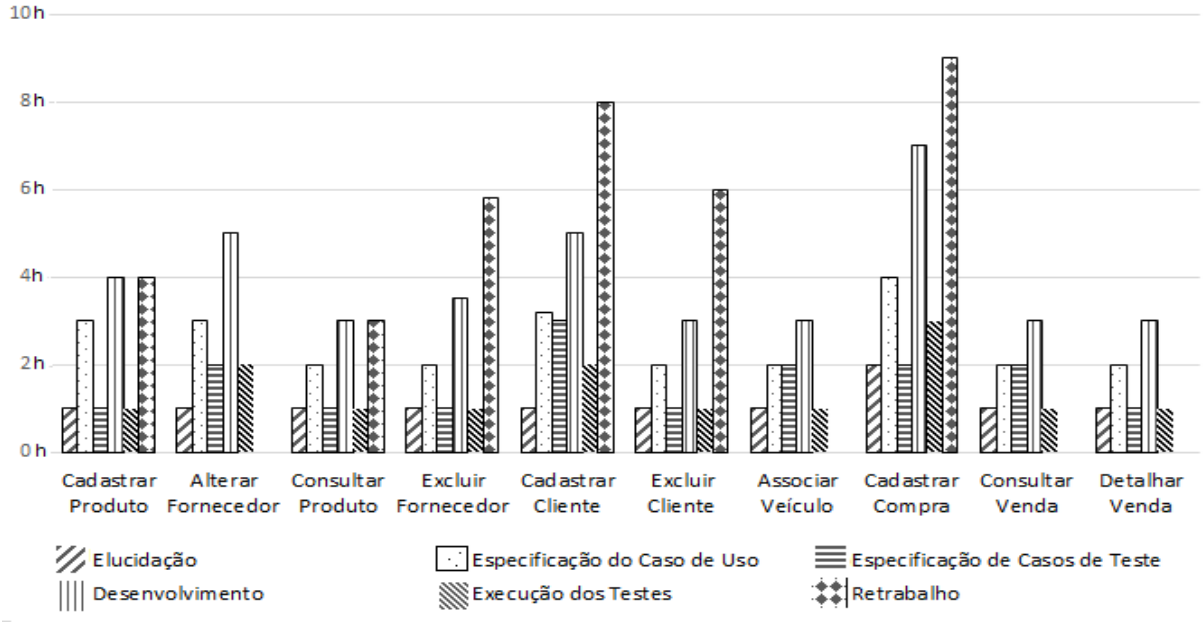
Processo Unificado		**BDD**	
Funcionalidade	**Esforço (h)**	**Funcionalidade**	**Esforço (h)**
Cadastrar Produto	**14**	Cadastrar Fornecedor	**11,5**
Alterar Fornecedor	**13**	Alterar Produto	**9,5**
Consultar Produto	**11**	Consultar Fornecedor	**7,5**
Excluir Fornecedor	**14,3**	Excluir Produto	**18,5**
Cadastrar Cliente	**22,2**	Alterar Cliente	**12**
Excluir Cliente	**14**	Consultar Cliente	**12**
Associar Veículo	**9**	Remover Carro	**6,5**
Cadastrar Compra	**27**	Cadastrar Venda	**32,5**
Consultar Venda	**9**	Consultar Compra	**10**
Detalhar Venda	**8**	Consultar Itens Compra	**9**
Média	**14,15**	**Média**	**12,9**

Fonte: Diogo Reali, 2015

Estes resultados nos permitem concluir que, neste estudo de caso, BDD implicou um menor esforço médio para o desenvolvimento de uma funcionalidade. Cabe destacar que BDD inclui neste esforço a total automatização dos testes, o que proporciona segurança para a refatoração de código e para a integração de novas funcionalidades. No processo inspirado no Processo Unificado o desenvolvimento é puro e não inclui a automatização dos testes.

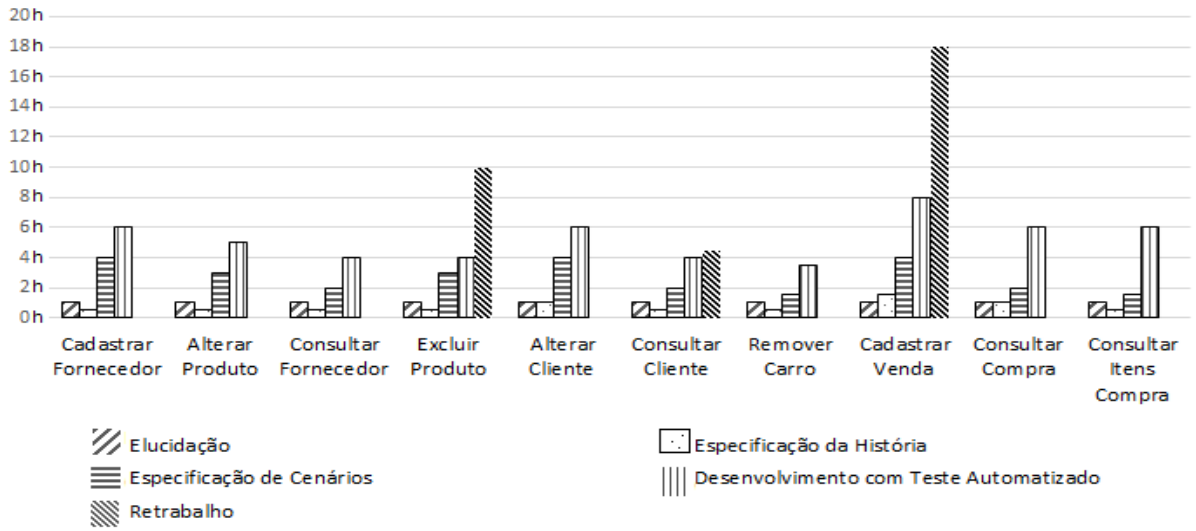
Os gráficos das Figuras 4.2 e 4.3 detalham a distribuição desses esforços nos diversos tipos de atividades que compõem cada um dos processos para a entrega de uma funcionalidade para o cliente. O Apêndice F e o Apêndice G exibem estes resultados de forma detalhada.

Figura 4.2 – Esforços por Tipo de Atividade para o Desenvolvimento de uma Funcionalidade Baseado no Processo Unificado



Fonte: Diogo Reali, 2015

Figura 4.3 – Esforços por Tipo de Atividade para o Desenvolvimento de uma Funcionalidade Segundo BDD



Fonte: Diogo Reali, 2015

É possível notar para o Processo Unificado que o esforço de especificação de requisitos só não é maior que o esforço de desenvolvimento em todas as funcionalidades. Se somados os esforços das atividades de especificação do caso de uso e especificação dos casos de teste representam 29,32% do esforço total, enquanto a atividade de desenvolvimento representa apenas 27,91%. Essa característica demonstra um significativo esforço despendido para a produção de documentação durante este processo, e explica o porquê desses métodos serem chamados de pesados. Também é possível observar um grande esforço dedicado a retrabalho.

No processo de desenvolvimento segundo BDD esse aspecto não acontece. Neste processo, a atividade de desenvolvimento é a que concentra o maior esforço, seguido pela especificação de cenários. Esse atributo reflete o princípio do Manifesto Ágil de que *software* funcional é mais importante do que documentação completa e detalhada. Nota-se também que ocorreu um menor esforço em retrabalho.

Ao comparar os esforços de desenvolvimento considerando o teste automatizado entre os dois processos, percebe-se que BDD concentra 40% dos esforços totais para a atividade de desenvolvimento a qual inclui testes automatizados, o que é consideravelmente maior que o esforço de desenvolvimento despendido no Processo Unificado (27,91%). Neste trabalho, não foi possível coletar de forma independente os esforços de desenvolvimento separados por código de produção e código de testes, uma vez que os dois estão fortemente integrados.

4.2 Correção da Aplicação

Como mencionado anteriormente, ocorreu a demonstração ao cliente do sistema completo e este apontou divergências quanto a sua expectativa e o sistema entregue. Por esse motivo, necessitou-se da atividade de retrabalho para a correção da aplicação, como pode ser observado pelas Figuras 4.2 e 4.3. Em relação as questões Q3 e Q4, destaca-se que a quantidade de funcionalidades que envolveram retrabalho foi menor no BDD em relação ao Processo Unificado (NB=3 e NU=6). A Tabela 4.4 mostra as funcionalidades em que foram encontrados os problemas.

Tabela 4.4 – Funcionalidades Entregues que Apresentaram Problemas

Processo Unificado		**BDD**	
Funcionalidade	**Erro**	**Funcionalidade**	**Erro**
Cadastrar Produto	Permitir o cadastro de produtos com estoque	Excluir Produto	Excluir somente produtos sem vendas ou compras vinculadas
Consultar Produto	Remover o campo Estoque Min.	Cadastrar Venda	Validar a quantidade disponível em estoque
Excluir Fornecedor	Excluir somente fornecedores sem compras vinculadas	Cadastrar Venda	Não permitir o cadastro de vendas sem itens
Cadastrar Cliente	Permitir o cadastro de clientes do tipo empresa	Cadastrar Venda	Não possibilitar alterar o cadastro da venda
Excluir Cliente	Excluir somente clientes sem vendas vinculadas	Cadastrar Venda	Associar o carro do cliente a venda
Cadastrar Compra	Não possibilitar alterar o cadastro da compra	Consultar Cliente	Exibir o campo CPF/CNPJ
Cadastrar Compra	Não permitir o cadastro de compras sem itens		
Total: 6	**Total: 7**	**Total: 3**	**Total: 6**

Fonte: Diogo Reali, 2015

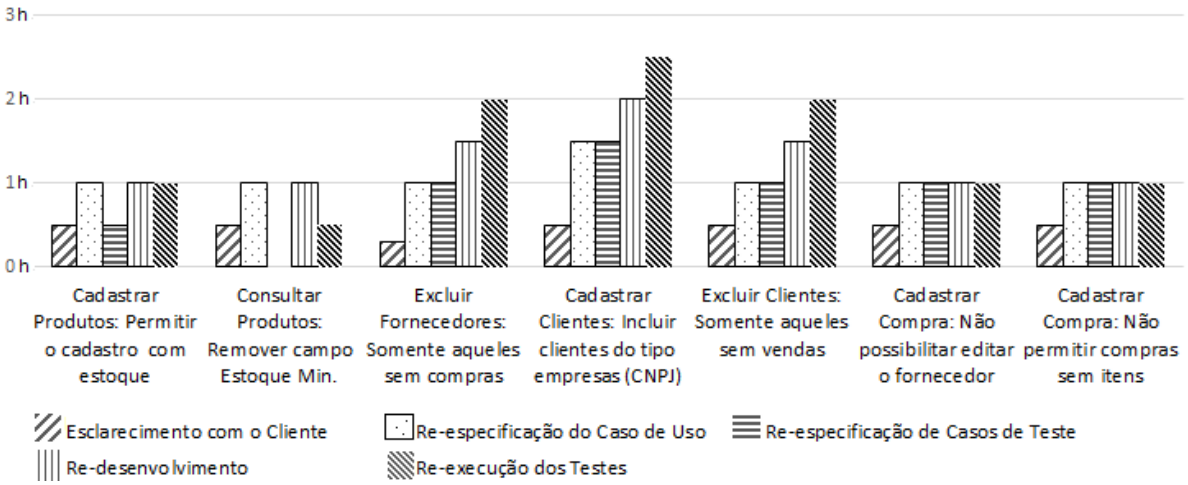
Nota-se o maior número de erros em funcionalidades desenvolvidas segundo o Processo Unificado (R10 = 7) se comparado a BDD (R9 = 6) , sendo que neste último caso os erros estiveram concentrados em menos funcionalidades. Não há subsídios neste estudo de caso suficientes para apontar o motivo desses resultados, porém, acredita-se que estes sejam mera casualidade. Acredita-se também, que se a demonstração para o cliente fosse realizada de maneira incremental, talvez maiores evidências sobre as vantagens de BDD fossem obtidas.

4.3 Esforço de Retrabalho

Entre as alterações desejadas pelo cliente, pode-se citar a necessidade de cadastrar um produto com um estoque inicial e de vincular um carro a venda. Além disso, entre outras correções, a aplicação não deveria permitir a exclusão de clientes, fornecedores e produtos que tenham vendas e/ou compras relacionadas, bem como que fossem realizadas compras e vendas vazias, ou seja, sem produtos associados.

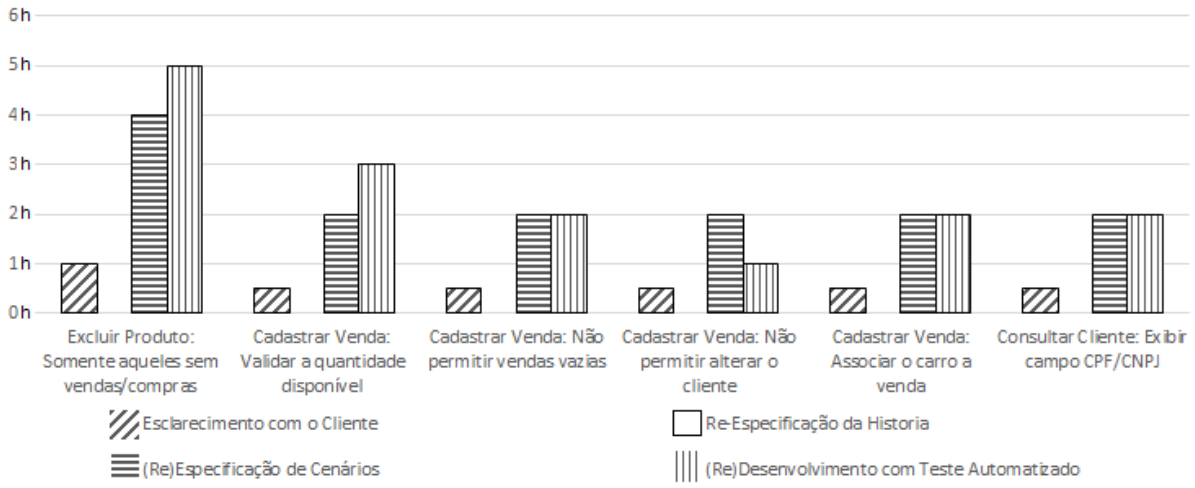
Considera-se uma correção, a modificação de uma funcionalidade entregue a fim de atender uma expectativa do cliente. A discriminação dos esforços de cada uma das atividades necessárias para a realização de uma correção em uma funcionalidade entregue com base no Processo Unificado e BDD são detalhados nas Figuras 4.5 e 4.6, respectivamente. Nestes gráficos, os somatórios dos esforços das atividades para todas as correções de uma funcionalidade representam o esforço total de retrabalho da funcionalidade mostradas nas Figuras 4.2 e 4.3. O Apêndice H e o Apêndice I mostram esses resultados de maneira detalhada.

Figura 4.5 – Esforços de Retrabalho por Atividade para cada Correção em uma Funcionalidade Baseado no Processo Unificado



Fonte: Diogo Reali, 2015

Figura 4.6 – Esforços de Retrabalho por Atividade para cada Correção em uma Funcionalidade Segundo BDD

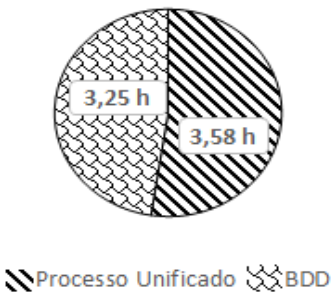


Fonte: Diogo Reali, 2015

A Figura 4.5 mostra que além do esforço necessário para a atividade de re-desenvolvimento para a correção da funcionalidade, o desenvolvimento segundo o Processo Unificado exigiu um grande esforço para atualização da documentação e principalmente para a re-execução de todos os testes da funcionalidade. Já em BDD, os esforços concentram-se nas atividades de re-especificação dos cenários para a correção e/ou especificação de novos cenários e na atividade de desenvolvimento com teste automatizado para a adequação da aplicação.

Em relação as perguntas Q5 e Q6 do *framework* da Seção 3.3, de maneira comparativa, a Figura 4.7 ressalta os esforços médios despendidos para a atividade de retrabalho para a correção de uma funcionalidade a partir de cada um dos processos de desenvolvimento. Percebe-se que o esforço médio necessário para a correção de uma funcionalidade é ligeiramente maior no Processo Unificado (R6 = 3,58h) em relação a BDD (R5 = 3,25h).

Figura 4.7 – Esforço Médio de Retrabalho por Funcionalidade



Fonte: Diogo Reali, 2015

Apesar de os esforços médios de retrabalho serem bastante semelhantes nos dois processos, é importante salientar que o esforço, quando consideradas somente as funcionalidades que necessitaram de retrabalho, é notavelmente maior em BDD do que em relação ao Processo Unificado. Esta característica, ao contradizer o princípio ágil de que o *software* produzido permite a fácil manutenção e incorporação de alterações, decorreu, neste trabalho, principalmente devido ao alto custo de manutenção dos cenários e testes automatizados pela ferramenta *JBehave* durante a atividade de retrabalho.

4.4 Proporção entre os Esforços de Desenvolvimento e de Retrabalho

As questões Q7 e Q8 do *framework* da Seção 3.3 referem-se a proporção entre os esforços totais de desenvolvimento e aquele despendido para a execução da atividade de retrabalho para cada um dos processos de desenvolvimento. Neste estudo de caso, obteve-se a proporção de 25,19% para BDD (R7) e de 25,30% para o Processo Unificado (R8), o que, por serem bastante próximos, não representam resultados conclusivos.

4.5 Satisfação do Usuário

O gráfico da Figura 4.8 diz respeito à qualidade das funcionalidades entregues com base na satisfação do cliente em relação ao conjunto de funcionalidades demonstradas a partir de cada um dos processos de desenvolvimento. Os resultados obtidos para as métricas R11 e R12 não possibilitaram uma conclusão significativa, uma vez que estas são iguais para ambos os processos.

Figura 4.8 – Satisfação do Cliente para cada Conjunto de Funcionalidade Entregue

Fonte: Diogo Reali, 2015

5 CONCLUSÃO E TRABALHOS FUTUROS

Esse trabalho teve como objetivo realizar uma investigação comparativa do desenvolvimento segundo BDD e uma adaptação do Processo Unificado em um estudo de caso envolvendo uma aplicação real. Para isto, desenvolveu-se, com o apoio da abordagem GQM, um *framework* de avaliação baseado em mensuração com a finalidade de possibilitar a comparação entre os dois processos.

O experimento proposto pretendia indicar qual o processo de desenvolvimento, o pesado baseado em documentação ou o ágil voltado ao comportamento, exige o menor esforço sob o ponto de vista do desenvolvedor. Os resultados obtidos indicam que o processo de desenvolvimento segundo BDD, se comparado àquele com base no Processo Unificado, necessita de um menor número de horas trabalhadas para a entrega de uma funcionalidade com complexidade semelhante. Além disso, o pequeno número de funcionalidades entregues que necessitaram de retrabalho demonstra a significativa capacidade de representação e de validação dos requisitos pelo processo segundo BDD.

Aponta-se como desvantagem de BDD a dificuldade, em muitos casos, de expressar o formato de campos e botões para determinado requisito. Essa particularidade acaba por exigir uma intensa comunicação entre os membros da equipe e cliente, o que frequentemente não é possível. Contrariamente, no Processo Unificado não se tem esse impedimento, uma vez que os requisitos estão expressos de maneira clara nos artefatos de casos de uso.

Em relação ao Processo Unificado utilizado, o prejuízo para o desenvolvimento está na exigência de um grande esforço para a atividade de execução dos testes, devido principalmente aos testes de regressão. De certo modo, essa propriedade explica a necessidade das grandes empresas do setor de desenvolvimento de *software* possuírem equipes responsáveis somente pela execução dos testes. Em BDD, opostamente, por possuir testes automatizados, essa atividade é fácil e quase instantânea, mas implica em um maior esforço para a atividade de desenvolvimento.

É importante salientar que os resultados expostos, bem como a análise comparativa entre os dois processos consideram o desenvolvimento de uma aplicação de pequeno porte por uma única pessoa. Dessa maneira, as conclusões apresentadas podem não refletir o cenário de desenvolvimento de uma grande empresa, e de aplicações com um alto nível de complexidade.Ressalta-se também a necessidade de adaptações em ambos os processos de desenvolvimento devido as condições não ideais de iteração com o cliente. Contudo, a

principal contribuição está no *framework* de avaliação proposto, o qual pode ser utilizado para a reprodução do experimento voltado a uma aplicação de grande porte.

Ao término desse trabalho, como trabalho futuro, fixa-se o ensejo de propor um "novo" processo de desenvolvimento que seja composto por aquelas atividades do Processo Unificado e segundo BDD que se acredita obterem uma maior eficácia, fazendo-se uso, de maneira entrelaçada, dos diversos artefatos utilizados por cada um dos processos. Rapidamente, pode-se citar, por exemplo, a utilização de um caso de uso modificado, que contenha apenas as definições de campos e botões, em conjunto com a elaboração de cenários para guiar o processo de desenvolvimento.

REFERÊNCIAS

KHAN, A. QURASHI, R. KHAN, U. **A Comprehensive Study of Commonly Practiced Heavy & Light Weight Software Methodologies**, International Journal of Computer Science Issues, 2011.

CRISPIN, L. GREGORY, J. **Agile Testing: A Practical Guide For Testers and Agile Teams.** Massachusetts, EUA. Pearson Education. 2009

ERDOGMUS, H. MELNIK, G. JEFFRIES, R. **Test-Driven Development,** Nova York, EUA. Encyclopedia of Software Engineering. 2009.

PRESMAN R. **Engenharia de Software.** McGraw-Hill. 2006.

SOMMERVILLE, I. **Engenharia de Software.** Pearson. 2011.

JANZEN, D. SAIEDIAN, H. **Does Test-Driven Development Really Improve Software Design Quality?** IEEE Software. 2008.

HAMMOND, S. UMPHRESS, D. **Test Driven Development: The State of the Practice**. ACM-SE ’12. 2012.

SOLÍS, C. WANG, X. **A Study of the Characteristics of Behaviour Driven Development**. Software Engineering and Advanced Applications. 2011.

RIMMER, C. **Introduction. Behaviour-Driven Development.** Disponível em <http://behaviour-driven.org/Introduction>. Acesso em 21.mai.2015

RATIONAL, S. **Rational Unified Process: Best Practices for Software Development Teams.** 2001. Disponível em <https://www.ibm.com/developerworks/rational/library/>. Acesso em 21.mai.2015

PEZZÈ, M. YOUNG, M. **Software Testing and Analysis: Process, Principles and Techniques**. John Wiley & Sons. 2008.

STANDISH, G. **Chaos Summary**, Massachusetts, EUA. 2009.

BASILI, V. CALDIERA, G. ROMBACH, H. **The GQM Approach.** Encyclopedia of Software Engineering. John Wiley & Sons. 1994.

SOLIGEN, R. BERGHOUT, E. **The Goal/Question/Metric Method: a pratical guide for quality improvement of software development**. McGraw-Hill, 1999.

SCHWABER, K. SUTHERLAND, J. **SCRUM.** 1995. Massachusetts, USA. Disponível em <https://www.scrum.org/>. Acesso em 21.mai.2015

WELLS, D. **Extreme Programming**, 2000.
Disponível em <http://www.extremeprogramming.org/>. Acesso em 21.mai.2015

AMBLER, S. **Agile modeling: effective practices for eXtreme Programming and the Unified Process.** John Wiley & Sons. 2010.

COHN, M. **User Stories Applied: For Agile Software Development.** Addison-Wesley Professional. 2004.
LARMAN, C. **Agile and Iterative Development: A Manager's Guide.** Pearson Education. Massachusetts, EUA. 2004

DAVIES, R. SEDLEY, L. **Agile Coaching.** Progmatic Bookshelf. 2009

NORTH, D. **Introducing BDD.** Disponível em <http://dannorth.net/category/bdd/>. Acesso em 21.mai.2015

FOWLER, M. **The New Methodology**, 2005.
Disponível em <http://www.martinfowler.com/articles/newMethodology.html>. Acesso em 21.mai.2015

BECK, K. BEEDLE, M. BENNEKUM, A. COCKBURN, A. CUNNINGHAM, W. FOWLER, M. GREENING, J. HIGHSMITH, J. HUNT, A. JEFFRIES, R. KERN, J. MARICK, B. MARTIN, R. MELLOR, S. SCHWABER, K. SUTHERLAND, J. THOMAS, D. **Manifesto for Agile Software Development**, 2001.
Diponível em <http://www.agilemanifesto.org/>. Acesso em 21.mai.2015

AGILE ALLIANCE, 2015. Disponível em < http://www.agilealliance.org/>. Acesso em 21.mai.2015

REENSKAUG, T. **MVC XEROX PARC 1978-79**, 1979.
Disponível em <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>. Acesso em 21.mai.2015

APÊNDICE A – Exemplo de um documento de Caso de Uso

UC02 – Cadastrar Produtos

1. DESCRIÇÃO

Este caso de uso permite incluir produtos.

2. PRÉ-CONDIÇÕES

Não se aplica.

3. FLUXO DE EVENTOS

Este caso de uso se inicia quando o ator necessita incluir produtos.

FLUXO BÁSICO – CADASTRAR PRODUTO

1. O ator aciona o procedimento de “Produtos”;
2. O ator aciona o procedimento de incluir um novo produto;
3. O sistema inicializa o valor de estoque do produto em 0.
4. O sistema solicita os dados do produto, conforme itens 5.1;
5. O ator informa os dados do produto solicitados, conforme os 5.1;
6. O ator solicita que os dados sejam salvos;
7. O sistema valida os dados do produto informados; [RN1] [RN2] [RN3] [RN4] FE1 FE2 FE3 FE4
8. O sistema grava o produto no banco de dados;
9. O sistema exibe a mensagem: “Inclusão realizada com sucesso”;
10. O ator repete passos 2 – 9 até indicar que terminou.
11. O ator finaliza o caso de uso;
12. O caso de uso é encerrado.

FLUXOS DE EXCEÇÃO

FE1. <u>Campos obrigatórios não informados</u>

No passo 6, do Fluxo Básico, o sistema identificou que um ou mais campos obrigatórios não foram informados.

1.1 O sistema exibe a mensagem: “Preencha os campos obrigatórios.”

1.2 O sistema retorna para o passo do Fluxo Básico onde o ator possa efetuar nova entrada;

FE2. Valor do campo Referência não é válido

No passo 6, do Fluxo Básico, o sistema identificou que o campo Referência não é um valor válido.

1.1 O sistema exibe a mensagem: “Já existe um produto com esta referência.”

1.2 O sistema retorna para o passo do Fluxo Básico onde o ator possa efetuar nova entrada;

FE3.Valor do campo Estoque Min. não é válido

No passo 6, do Fluxo Básico, o sistema identificou que o campo Estoque Mín. não é um valor válido numérico.

1.1 O sistema exibe a mensagem: “Estoque Mín.: Deve ser um número formado por um ou mais dígitos.”

1.2 O sistema retorna para o passo do Fluxo Básico onde o ator possa efetuar nova entrada;

FE4.Valor do campo Estoque não é válido

No passo 6, do Fluxo Básico, o sistema identificou que o campo Estoque não é um valor válido numérico.

1.1 O sistema exibe a mensagem: “Estoque : Deve ser um número formado por um ou mais dígitos.”

1.2 O sistema retorna para o passo do Fluxo Básico onde o ator possa efetuar nova entrada;

REGRAS DE NEGÓCIO

RN1. Validar campos obrigatórios

O sistema valida se os campos Referência, Produto e Fabricante estão preenchidos.

RN2. Validar campo Referência é um valor válido

O sistema valida se não existe algum outro registro de produto com os mesmo valor do campo Referência

RN3. Validar campo Estoque Mín. é um valor válido

O sistema valida se o campo Estoque Mín. é um valor numérico.

RN4. Validar campo Estoque é um valor válido

O sistema valida se o campo Estoque é um valor numérico.

4. PÓS-CONDIÇÃO

Não se aplica.

5. DEFINIÇÃO DOS CAMPOS E BOTÕES

5.1 INCLUSÃO DE PRODUTO

Campo	Obrigatório	Tipo	Observação
Referência	S	TX	
Descrição	S	TX	
Fabricante	S	TX	
Estoque Mín	N	TX	Somente valores numéricos.
Estoque	S	TX	Somente valores numéricos.
Botões			
Salvar	N/A	BT	Salva o produto
Excluir	N/A	BT	Inicia o cenário que permite excluir um produto. Disponível somente na alteração do produto
Cancelar	N/A	BT	Retorna para o cenário consultar produtos.

Tipo: **BT –** Botão; **CB –** ComboBox; **TX** - Texto. Obrigatório: **N/A -** Não se aplica

APÊNDICE B – Exemplo de um documento de Casos de Teste

TC01 – Cadastrar Produto

1. PRÉ-CONDIÇÃO

Os produtos que seguem já estão cadastrados.

Produto	Referência	Fabricante	Estoque Mínimo
A	refA	fabA	

2. CASOS DE TESTE

Os seguintes entradas de dados e respectivos resultados esperados.

Produto	Referência	Fabricante	Estoque Mín.	Estoque	Resultados Esperados
novoA	refNovoA	fabNovoA	10	0	“Inclusão realizada com sucesso.”
novoB	refNovoB	fabNovoB		5	“Inclusão realizada com sucesso.”
	refNovoC	fabNovoC		10	“Preencha os campos obrigatórios.”
novoC		fabNovoC		0	“Preencha os campos obrigatórios.”
novoC	refNovoC			15	“Preencha os campos obrigatórios.”
					“Preencha os campos obrigatórios.”
		fabNovoC		0	“Preencha os campos obrigatórios.”
	refNovoC			17	“Preencha os campos obrigatórios.”
novoC				0	“Preencha os campos obrigatórios.”
novoC	refNovoC	fabNovoC			“Preencha os campos obrigatórios.”
refOutroA	refA	fabOutroA	20	0	“Já existe um produto com esta referência.”
refOutroA	refA	fabOutroA		0	“Já existe um produto com esta referência.”
novoD	refNovoD	fabNovoD	teste	0	Estoque Mín.: ‘teste’ deve ser um número formado por um ou mais dígitos.
novoD	refNovoD	fabNovoD		teste	Estoque: ‘teste’ deve ser um número formado por um ou mais dígitos.

APÊNDICE C – Exemplo de uma História de Usuário

Title: Alterar Cadastro Produto

Narrative:
As a funcionário
I want poder alterar os dados de um produto
So that eu posso realizar a modificações pontuais mantendo o cadastro atualizado

APÊNDICE D – Exemplos de Cenários

Scenario: *Alteracao de um produto*
Given um produto A ja cadastrado e que necessite que seus dados sejam alterados
When o funcionario entrar na tela de listagem de produtos e clicar em editar o produtoA
And alterar um dos campos obrigatoros uma <referencia> unica, um <nome> e um <fabricante>
And alterar opcionalmente uma quantidade de <estoque>
And tentar salvar as alteracoes
Then todas as alteracoes devem ser salvas e a mensagem de Alteracao realizada com sucesso deve ser mostrada.

Examples:
|*referencia*|*nome*|*fabricante*|*estoque*|
|*ref2*|*nome2*|*fabricante*| *10*|

Scenario: *Alterar produto sem dados*
Given um produto A ja cadastrado e que necessite que seus dados sejam alterados
When o funcionario entrar na tela de listagem de produtos e clicar em editar o produtoA
And nao informar todos os campos obrigatorios, que sao <referencia> ,<nome> e <fabricante>
And tentar salvar as alteracoes
Then as alteracoes nao podem ser salvas e a mensagem Preencha os campos obrigatorios deve ser mostrada

Examples:
referencia	*nome*	*fabricante*	*estoque*
			10
	nome2	*fabricante*	*10*
ref2		*fabricante*	*10*
ref2	*nome2*		*10*

Scenario: *Alterar produto com referencia ja exista*
Given um produto A ja cadastrado e que necessite que seus dados sejam alterados
Given um produto B ja cadastrado com referencia <referencia>
When o funcionario entrar na tela de listagem de produtos e clicar em editar o produtA
And informar a referencia <referencia>
And tentar salvar as alteracoes
Then as alteracoes nao podem ser salvas e a mensagem Ja existe um produto com esta referencia deve ser mostrada.

Examples:
|*referencia*|*nome*|*fabricante*|*estoque*|
|*referenciaB*|*nome2*|*fabricante*| *10*|

APÊNDICE E – Exemplos de Testes automatizados

```
public class AlterarCadastroProduto{

	private ProdutoMB produtoMB = new ProdutoMB();
	private ProdutoFacade produtoFacade = new ProdutoFacade();
	//private Produto produto;
	private Produto produtoB;

	private Integer idProdutoAlterar;

	@BeforeStory
	public void inicializarBancoDados(){
		produtoMB.carregarInserir();
		produtoMB.getProduto().setFabricante("fabricante");
		produtoMB.getProduto().setNome("nome");
		produtoMB.getProduto().setReferencia("referenciaA");

		produtoMB.salvar();

		idProdutoAlterar = produtoMB.getProduto().getId();
	}

	@Given("um produto A ja cadastrado e que necessite que seus dados sejam alterados")
	public void givenUmProdutoAJaCadastradoEQueNecessiteQueSeusDadosSejamAlterados(){

	}

	@When("o funcionario entrar na tela de listagem de produtos e clicar em editar o
produto A")
	public void
whenOFuncionarioEntrarNaTelaDeListagemDeProdutosEClicarEmEditarOProdutoA(){
		produtoMB.carregarProduto(idProdutoAlterar);
	}

	@When("alterar um dos campos obrigatoros uma <referencia> unica, um <nome> e  um
<fabricante>")
	public void whenInformarObrigatoriamenteUmareferenciaUnicaUmnomeEUmfabricante(
	@Named("referencia") String referencia, @Named("nome") String nome,
	@Named("fabricante") String fabricante){
		produtoMB.getProduto().setReferencia(referencia);
		produtoMB.getProduto().setNome(nome);
		produtoMB.getProduto().setFabricante(fabricante);
	}

	@When("alterar opcionalmente uma quantidade de <estoque>")
	public void whenOpcionalmenteUmaQuantidadeDeestoque(@Named("estoque") Integer
estoque){
		produtoMB.getProduto().setEstoqueMin(estoque);
	}
```

```
	@When("tentar salvar as alteracoes")
	public void whenSalvarAsAlteracoes(){
		produtoMB.salvar();
	}

	@Then("todas as alteracoes devem ser salvas e a mensagem de Alteracao realizada com sucesso deve ser mostrada.")
	public void thenAMensagemDeInclusaoRealizadaComSucessoDeveSerMostrada(){
		Assert.assertEquals("Alteração realizada com sucesso.", produtoMB.getMsg());
		Assert.assertEquals(idProdutoAlterar, produtoMB.getProduto().getId());
	}


	@When("nao informar todos os campos obrigatorios, que sao <referencia> ,<nome> e <fabricante>")
	public void whenNãoInformarTodosOsCamposObrigatoriosQueSãoreferencianomeEfabricante(@Named("referencia") String referencia, @Named("nome") String nome, @Named("fabricante") String fabricante){
		produtoMB.getProduto().setReferencia(referencia);
		produtoMB.getProduto().setNome(nome);
		produtoMB.getProduto().setFabricante(fabricante);
	}

	@Then("as alteracoes nao podem ser salvas e a mensagem Preencha os campos obrigatorios deve ser mostrada")
	public void thenAsAlteracoesNaoPodemSerSalvasEAMensagemPreenchaOsCamposObrigatoriosDeveSerMostrada(){
		Assert.assertEquals("Preencha os campos obrigatórios", produtoMB.getMsg());
		if(produtoFacade.pesquisar(idProdutoAlterar).equals(produtoMB.getProduto())){
			Assert.fail();
		}
	}

	@Given("um produto B ja cadastrado com referencia <referencia>")
	public void givenUmProdutoBJaCadastradoComReferenciareferencia(@Named("referencia") String referencia){
		produtoMB.carregarInserir();
		produtoMB.getProduto().setFabricante("fabricante");
		produtoMB.getProduto().setNome("nome");
		produtoMB.getProduto().setReferencia(referencia);

		produtoMB.salvar();
		produtoB = produtoMB.getProduto();
	}


	@When("informar a referencia <referencia>")
	public void whenInformarAReferenciareferencia(@Named("referencia") String referencia){
		produtoMB.getProduto().setReferencia(referencia);
	}
```

```
	@Then("as alteracoes nao podem ser salvas e a mensagem Ja existe um produto com esta referencia deve ser mostrada.")
	public void thenAsAlteracoesNaoPodemSerSalvasEAMensagemJaExisteUmProdutoComEstaReferenciaDeveSerMostrada(){
		Assert.assertEquals("Já existe um produto com esta referência", produtoMB.getMsg());
		if(produtoFacade.pesquisar(idProdutoAlterar).equals(produtoMB.getProduto())){
			Assert.fail();
		}
	}

	@AfterStory
	public void limparBancoDados(){
		produtoFacade.excluir(produtoMB.getProduto());
		produtoFacade.excluir(produtoB);
	}

}
```

APÊNDICE F – Esforços por Tipo de Atividade para o Desenvolvimento de uma Funcionalidade Baseado no Processo Unificado

Funcionalidade	Elucidação	Especificação do Caso de Uso	Especificação de Casos de Teste	Desenvolvimento	Execução dos Testes	Retrabalho	Total
Cadastrar Produto	1	3	1	4	1	4	**14**
Alterar Fornecedor	1	3	2	5	2	0	**13**
Consultar Produto	1	2	1	3	1	3	**11**
Excluir Fornecedor	1	2	1	3,5	1	5,8	**14,3**
Cadastrar Cliente	1	3,2	3	5	2	8	**22,2**
Excluir Cliente	1	2	1	3	1	6	**14**
Associar Veículo	1	2	2	3	1	0	**9**
Cadastrar Compra	2	4	2	7	3	9	**27**
Consultar Venda	1	2	2	3	1	0	**9**
Detalhar Venda	1	2	1	3	1	0	**8**
						TOTAL	**141,5**

APÊNDICE G – Esforços por Tipo de Atividade para o Desenvolvimento de uma Funcionalidade segundo o BDD

Funcionalidade	Elucidação	Especificação da História	Especificação de Cenários	Desenvolvimento com Teste Automatizado	Retrabalho	Total
Cadastrar Fornecedor	1	0,5	4	6	0	**11,5**
Alterar Produto	1	0,5	3	5	0	**9,5**
Consultar Fornecedor	1	0,5	2	4	0	**7,5**
Excluir Produto	1	0,5	3	4	10	**18,5**
Alterar Cliente	1	1	4	6	0	**12**
Consultar Cliente	1	0,5	2	4	4,5	**12**
Remover Carro	1	0,5	1,5	3,5	0	**6,5**
Cadastrar Venda	1	1,5	4	8	18	**32,5**
Consultar Compra	1	1	2	6	0	**10**
Consultar Itens Compra	1	0,5	1,5	6	0	**9**
					TOTAL	**129**

APÊNDICE H – Esforços de Retrabalho por Tipo de Atividade para cada Correção em uma Funcionalidade Baseado no Processo Unificado

Correção	Esclarecimento com o Cliente	Re-especificação do Caso de Uso	Re-especificação de Casos de Teste	Re-desenvolvimento	Re-execução dos Testes	Total
Cadastro Produtos: Permitir o cadastro com estoque	0,5	1	0,5	1	1	**4**
Consulta Produtos: Remover campo Estoque Min.	0,5	1	0	1	0,5	**3**
Excluir Fornecedores: Somente aqueles sem compras	0,3	1	1	1,5	2	**5,8**
Cadastrar Clientes: Incluir clientes do tipo empresas (CNPJ)	0,5	1,5	1,5	2	2,5	**8**
Excluir Clientes: Somente aqueles sem vendas	0,5	1	1	1,5	2	**6**
Cadastrar Compra: Não possibilitar editar o fornecedor	0,5	1	1	1	1	**4,5**
Cadastrar Compra: Não permitir compras sem itens	0,5	1	1	1	1	**4,5**
					TOTAL	**35,8**

APÊNDICE I – Esforços de Retrabalho por Tipo de Atividade para cada Correção em uma Funcionalidade segundo o BDD

Correção	Esclarecimento com o Cliente	Re-Especificação da Historia	(Re)Especificação de Cenários	(Re)Desenvolvimento com Teste Automatizado	Total
Excluir Produto: Somente aqueles sem vendas/compras	1	0	4	5	**10**
Cadastrar Venda: Validar a quantidade disponível	0,5	0	2	3	**5,5**
Cadastrar Venda: Não permitir vendas vazias	0,5	0	2	2	**4,5**
Cadastrar Venda: Não permitir alterar o cliente	0,5	0	2	1	**3,5**
Cadastrar Venda: Associar o carro a venda	0,5	0	2	2	**4,5**
Consultar Cliente: Exibir campo CPF/CNPJ	0,5	0	2	2	**4,5**
				TOTAL	**32,5**