UNIVERSIDADE FEDERAL DE OURO PRETO
ESCOLA DE MINAS
COLEGIADO DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO - CECAU

WILKER HUDSON DE PAULA

QUALIDADE DE SOFTWARE E DESENVOLVIMENTO DIRIGIDO POR COMPORTAMENTO - BDD: UM ESTUDO DE CASO

MONOGRAFIA DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Ouro Preto, 2019

WILKER HUDSON DE PAULA

QUALIDADE DE SOFTWARE E DESENVOLVIMENTO DIRIGIDO POR COMPORTAMENTO - BDD: UM ESTUDO DE CASO

Monografia apresentada ao curso Engenharia de Controle e Automação da Escola de Minas, da Universidade Federal de Ouro Preto, como requisito parcial para a obtenção do título de bacharel em Engenharia de Controle e Automação.

Orientadora: Prof. Dr. Karla B. P. Palmieri

Ouro Preto
Escola de Minas – UFOP
Dezembro/2019

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

P324q Paula, Wilker Hudson De .
Qualidade de Software e Desenvolvimento Dirigido Por Comportamento - BDD [manuscrito]: um estudo de caso. / Wilker Hudson De Paula. - 2019.
60 f.: il.: color..

Orientadora: Profa. Dra. Karla Boaventura Pimenta Palmieri.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Escola de Minas.

1. Engenharia de Software. 2. Software - Qualidade. 3. Behavior Driven Development (BDD). 4. Desenvolvimento ágil de software. I. Palmieri, Karla Boaventura Pimenta. II. Universidade Federal de Ouro Preto. III. Título.

CDU 681.5

Bibliotecário(a) Responsável: Maristela Sanches Lima Mesquita - CRB:1716

MINISTÉRIO DA EDUCAÇÃO
Universidade Federal de Ouro Preto
Escola de Minas
Colegiado do Curso de Engenharia de Controle e Automação - CECAU

ATA DE DEFESA

Aos 16 dias do mês de Dezembro de 2019, às 13:00h, na Escola de Minas – UFOP, reuniu-se a Comissão Avaliadora designada para julgar a Monografia do graduando **WILKER HUDSON DE PAULA** do curso de Engenharia de Controle e Automação intitulada **QUALIDADE DE *SOFTWARE* E DESENVOLVIMENTO DIRIGIDO POR COMPORTAMENTO - BBD: UM ESTUDO DE CASO**, sob orientação da Profa. Dra. Karla Boaventura Pimenta Palmieri, sendo a referida Comissão composta pelo BSc Fernando dos Santos Alves Fernandes e pelo Prof. Engenheiro Gradimilo Cândido de Jesus.

A Comissão Avaliadora resolveu considerar o trabalho aprovado atendendo às exigências para defesa e recomendações da banca examinadora.

Ouro Preto, 16 de Dezembro de 2019.

Karla Palmieri

Profa. Dra. Karla Boaventura Pimenta Palmieri – Orientadora

Alves Fernandes

BSc. Fernando dos Santos Alves Fernandes – Convidado

Prof. Engenheiro Gradimilo Cândido de Jesus – Convidado

Wilker Hudson de Paula

Wilker Hudson de Paula - Aluno

AGRADECIMENTOS

Agradeço a Deus e aos guias espirituais por me ajudarem e iluminarem meus caminhos até aqui, dando-me forças para sempre seguir em frente.

À UFOP, FG, LAP e aos professores pelo ensino de excelência.

À Prof. Dr. Karla, que aceitou ser minha orientadora e me deixou livre para explorar todos os campos que eu quis. Obrigado pela orientação e por sempre confiar em mim.

Dedico este trabalho de conclusão de curso aos meus pais, José Geraldo e Maria José, à minha irmã, Kênia, por sempre me apoiarem e me amarem incondicionalmente, e a todos os meus amigos que, direta ou indiretamente, me ajudaram a caminhar até aqui.

Aos colegas de trabalho que me ajudaram na parte de aplicação técnica desta monografia.

A todos que, de alguma forma, colaboraram para a realização deste trabalho.

« Boire pour la soif
Je ne sais pas ce qui de nous deux restera
Tu dis mais je ne regarde pas
Je n'ai jamais vu la mer […]
Et la pluie qui revient, dans nos voix
Pas une chanson où je ne pense à toi
Dans ce monde inhabitable
Il vaut mieux danser sur les tables
A port coton qu'on se revoit »
ZAZ – Port Coton

RESUMO

Esta monografia apresenta os resultados e conclusões do acompanhamento da aplicação da técnica de BDD - Desenvolvimento Dirigido por Comportamento - e de histórias de usuários (User Stories) como melhoria de processo em um projeto que utiliza Kanban. Foi utilizada a pesquisa-ação como método de pesquisa, coleta de dados via questionários com os membros do projeto e pela observação, além da avaliação da análise de conteúdo. Após realizada a análise dos dados, os resultados mostraram o quanto foi positiva a melhoria no desenvolvimento ágil de *software* utilizando o BDD.

Palavras-chave: Engenharia de *Software*. Qualidade de *Software*. BDD. Métodos Ágeis.

ABSTRACT

This monograph presents the results and conclusions of monitoring the application of the technique of BDD (Behavior Driven Development) and User Stories, as process improvement in a project that is using Kanban. The research-action was used as a method of research, data collection via questionnaires answered by project members and by observation, in addition to the evaluation of content analysis. Through data analysis, the results showed how positive was improving in agile software development using BDD.

Keywords: Software Engineering. Software quality. BDD. Agile Methods.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS

BDD – *Behavior Driven Development*

TDD – *Test-Driven Development*

IEEE – Instituto de Engenheiros Eletricistas e Eletrônicos

ISO – *International Organization Standardization*

IEC – *International Electrotechnical Comission*

QA – *Quality Assurance*

PO – *Product Owner*

US – *User Stories*

DEV – Desenvolvedor(es)

SRE - *Site Reliability Engineer*

UX – *User Experience*

UI – *User Interface*

SUMÁRIO

1 INTRODUÇÃO

1.1 CONSIDERAÇÕES INICIAIS

A definição de qualidade, em primeiro momento, pode parecer algo trivial e de fácil definição, porém, se analisada de forma intrínseca, é possível perceber que o conceito de qualidade é algo variável. Por isso, este estudo se baseia em entender como, de fato, a qualidade de *software* e o BDD (*Behavior Driven Development*) se correlacionam.

Considera-se que investir em qualidade do produto e na satisfação do cliente é um dos principais fatores para que o produto seja testado antes que o entregue como peça final para o mercado, independentemente de o *software*, por exemplo, tenha uma ou várias funções pré-definidas, como em sistemas embarcados ou aplicações com uma destinação mais ampla e não tão genérica.

Com o BDD, o processo de teste se inicia bem antes da construção do código fonte em si. Em desenvolvimento orientado por comportamento (BDD), os requisitos para um componente de *software* servem de base para a criação de uma série de casos de teste que exercitam a interface, tentando encontrar erros nas estruturas de dados e funcionalidade fornecida pelo componente.

Os projetos de sistemas trazem em sua bagagem uma grande multidisciplinaridade, levando quesito qualidade para um patamar de destaque durante a sua projeção, execução e finalização. Com as metodologias ágeis (*Scrum, Kanban,* etc.) em ascensão, o BDD ficou em evidência na comunidade acadêmica e industrial.

Em um panorama em que geralmente a complexidade do *software* tende a ser cada vez mais ascendente e os prazos diminuem de maneira proporcionalmente inversa, deve-se encontrar uma maneira de reduzir os impactos para que a qualidade de *software* se mantenha tão ótima quanto pensada inicialmente, mesmo com todos os desafios que a construção de um *software* exige.

Assim, foi nesse contexto que o BDD surgiu, através de uma crítica de Dan North ao TDD (Desenvolvimento orientado a testes), no qual ele visava otimizar o conceito de ‘verificação e validação’, já aplicado, e tornar mais eficiente a construção de cenários a serem testados e/ou desenvolvidos.

1.2 OBJETIVOS

Este estudo visa alcançar o objetivo geral e os objetivos específicos relacionados abaixo.

1.2.1 Objetivo geral

Aprofundar os conhecimentos teóricos sobre qualidade de *software* e o uso do BDD em construção de *software* desde o planejamento até a sua saída como produto final.

1.2.2 Objetivos específicos

- Abordar o conceito de qualidade de *software*;
- Entender o conceito de BDD;
- Analisar, de forma teórica, como o BDD ajuda no processo de criação de programas;
- Analisar os resultados da aplicação de BDD na confecção de *software* em uma *startup* de TI.

1.3 MOTIVAÇÃO

> Apesar de gerentes e profissionais envolvidos com a área técnica reconhecerem a necessidade de uma abordagem mais disciplinada em relação ao *software*, eles continuam a discutir a maneira como essa disciplina deve ser aplicada. Muitos indivíduos e empresas desenvolvem *software* de forma desordenada, mesmo ao construírem sistemas dirigidos às mais avançadas tecnologias. Grande parte de profissionais e estudantes não estão cientes dos métodos modernos. E, como consequência, a qualidade do *software* que produzem é afetada. Além disso, a discussão e a controvérsia sobre a real natureza da abordagem de engenharia de *software* continuam. A engenharia de *software* é um estudo repleto de contrastes. A postura mudou, progressos foram feitos, porém, falta muito para essa disciplina atingir a maturidade (PRESSMAN, 2011)

De encontro com essa abordagem, acredita-se que a discussão a ser promovida neste trabalho pode gerar uma reflexão a respeito dos aspectos críticos que envolvem a qualidade de *software* na prática e, assim, contribuir para as pesquisas futuras, bem como para o aumento da literatura da área.

1.4 JUSTIFICATIVA

A empresa, em que este estudo foi aplicado, atua nesse segmento há 5 anos e é pioneira no assunto de monitoramento web, possuindo um amplo reconhecimento nacional na área de *marketing*. Desse modo, é preciso que exista uma administração eficaz de seu controle de qualidade e desenvolvimento de *software*, para que haja sempre a satisfação do cliente.

A implantação do sistema Kanban dentro de uma *startup* fabricante de *software* pode auxiliar a empresa nessas questões, pois é uma metodologia de método ágil que tem como objetivo o controle preciso de tempo e organização de novos *realeases*, de produção e de controle da qualidade extremamente precisa. Além disso, ele também possui um baixo custo, uma vez que se utiliza de cartões que permitem o controle visual da posição de estágio de qualquer nova *feature* a ser desenvolvida e/ou implementada.

1.5 ESTRUTURA DO TRABALHO

O trabalho está dividido em seis capítulos. No Capítulo 2, tem-se uma breve revisão da literatura a respeito de Qualidade de *Software*, Engenharia de *Software*, Testes de *Software*, *Test Driven Development, Behavior Driven Development* e Métodos ágeis. No Capítulo 3, apresenta-se os processos metodológicos da pesquisa utilizada para o desenvolvimento deste trabalho. O capítulo 4 traz o estudo de caso aplicado. No capítulo 5, a apresentação e os resultados obtidos com o estudo. E no Capítulo 6, tem-se a conclusão resultante desta monografia.

2 CONCEITOS GERAIS E REVISÃO DA LITERATURA

Neste capítulo, serão apresentados os resumos dos conceitos necessários para o entendimento deste trabalho: Engenharia de *Software,* Qualidade de *Software*, Métodos ágeis, Testes de *Software* e *Behavior Driven Development*.

2.1 ENGENHARIA DE *SOFTWARE*

A primeira definição do termo “engenharia de *software*” se deu por Friedrich Ludwig Bauer, em 1968, como “a criação e a utilização de sólidos princípios de engenharia a fim de obter software de maneira econômica, que seja confiável e que trabalhe em máquinas reais". Esse termo foi popularizado por Margaret Hamilton, famosa cientista da computação que participou do projeto *Apollo 11* da NASA, naquela época, ciência da computação e engenharia de *software* não eram ainda disciplinas regulares nas faculdades. Hamilton, então, associou o termo “engenharia” para ganhar mais reconhecimento no cargo que exercia dentro da NASA e do MIT.

De acordo com o Dicionário Aurélio Eletrônico V.2.0, engenharia define-se como:

> Arte de aplicar conhecimentos científicos e empíricos e certas habilitações específicas à criação de estruturas, dispositivos e processos que se utilizam para converter recursos naturais em formas adequadas ao atendimento das necessidades humanas.

A engenharia de *software* é a área da computação que foca nos aspectos mais práticos da produção de um sistema de *software*. Os fundamentos técnicos e científicos desta área envolvem o uso de modelos abstratos e precisos que permitem ao engenheiro especificar, projetar, implementar e manter sistemas de *software*, avaliando e garantindo sua qualidade.

De acordo com Pressman (2001), a engenharia de *software* carrega consigo três elementos fundamentais:

- **Métodos –** Provêm uma base técnica para se construir o *software*. Eles incluem um amplo conjunto de tarefas, que abrange análise de requisitos, projeto, construção de programas, teste e manutenção;

- **Ferramentas –** O principal objetivo é proporcionar apoio automatizado ou semiautomatizado para as atividades dos processos. É um sistema de suporte ao desenvolvimento de *software* que abrange toda ferramenta baseada em computadores que auxiliam atividades de engenharia de *software*, desde análise de requisitos e modelagem até programação e testes.
- **Processos –** Constituem a interação dos métodos, ferramentas e pessoas. Mantêm unidas todas as camadas de tecnologia e permite o desenvolvimento racional e oportuno de sistemas para computadores.

Obter qualidade nos processos e produtos de *software* não é uma tarefa fácil. O processo interno já formado em um ambiente organizacional alinhado com a resistência à mudança pode dificultar muito alcançar os objetivos de qualidade inicialmente traçados. No entanto, nada é mais decepcionante do que produzir um *software* que não satisfaça as necessidades dos clientes (Maldonado, 2001).

2.2 QUALIDADE DE *SOFTWARE*

Este tópico não tem como objetivo de apresentar todos os pontos teóricos sobre qualidade de *software*, mas apenas os aspectos mais relevantes para este presente trabalho.

Como destaca Herman G. Weinberg: “A qualidade é relativa. O que é qualidade para uma pessoa pode ser falta de qualidade para outra”.

De acordo com o IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos), a qualidade se define como “Grau de conformidade de um sistema, componente ou processo com os respectivos requisitos”, ou, também, como “Grau de conformidade de um sistema, componente ou processo com as necessidades e expectativas de clientes ou usuários”.

A norma NBR ISO 9000:2005 define qualidade como sendo o grau no qual um conjunto de características inerentes satisfaz aos requisitos, ou seja, é possível afirmar que se algum produto ou serviço atende aos requisitos especificados, este mesmo produto ou serviço possui a qualidade desejada.

A engenharia de *software* consulta dois órgãos internacionais de padronização de normas: a *International Organization Standardization* - ISO e a *International Electrotechnical Comission* - IEC. Ambas são conhecidas como a norma internacional

ISO/IEC, na qual a qualidade de *software* é definida como “A totalidade de características de um produto de *software* que lhe confere a capacidade de satisfazer necessidades explícitas e implícitas.”

Conforme Sommerville (2007), qualidade é um conceito complexo e que não é diretamente comparável com a qualidade na manufatura. Visto que, na manufatura, tem-se que o entendimento de qualidade se dá quando o produto desenvolvido atende às suas especificações. (CROSBY, 1979).

Para Campos (1992), a qualidade é, fundamentalmente, a adequação ao uso; ou, ainda, a totalidade das características de um produto ou serviço que se relacionam com a sua capacidade de atender às necessidades de um cliente. A qualidade na prestação de serviços, matéria prima e entrega final do produto são os principais fatores que atraem clientes para empresas de *software* quando se considera a aquisição de um produto ou serviço que se deseja. Uma vez entendidas e alinhadas as necessidades do cliente, pode-se traçar o plano de metas e execução de cenários de testes que serão necessários.

De acordo com Pressman (2011), uma das maneiras de se garantir que o trabalho foi realizado de forma correta e coerente, é observar a qualidade por meio da verificação dos resultados das atividades de controle de qualidade, efetuando a verificação de erros antes da entrega e de defeitos que acabaram passando despercebidos e direcionados para produção.

A figura 1 apresenta a categorização dos princípios que influenciam a qualidade de *software*, de acordo com Pressman (2011, p 361), onde apresenta-se os fatores de qualidade de McCall, Richards e Walters (1977):

- Correção: o quanto um programa satisfaz e cumpre os objetivos solicitados pelo cliente;
- Confiabilidade: o quanto se pode esperar que um programa realize a função pretendida com a precisão exigida;
- Eficiência: a quantidade de recursos computacionais e de código exigida para que um programa execute sua função;
- Integridade: o quanto o acesso ao *software* ou dados por pessoas não autorizadas pode ser controlado;

- Usabilidade: o quanto de esforço é necessário para aprender, operar, preparar a entrada de dados e interpretar as saídas produzidas pelo sistema;
- Facilidade de manutenção: o quanto de esforço é necessário para localizar e eliminar problemas um programa;
- Flexibilidade: o quanto de esforço é necessário para modificar um programa em operação;
- Testabilidade: o quanto de esforço é necessário para testar um programa de modo a garantir que ele desempenhe a função determinada;
- Portabilidade: o quanto de esforço é necessário para transferir o programa de um ambiente de *hardware* e/ou *software* para outro;
- Reusabilidade: o quanto um programa (ou partes dele) pode(m) ser reutilizado (as) em outros programas;
- Interoperabilidade: o quanto de esforço é necessário para incorporar um sistema a outro.

Figura 1. Fatores de qualidade de *software* de McCal

Fonte: Pressmann, R. Engenharia de *Software* - Uma Abordagem Profissional (2016). Capítulo 19, p. 416

2.3 MÉTODOS ÁGEIS

No ano 2000, um grupo de renomados consultores *experts* em *Extreme Programming* (XP) reuniram-se em Oregan, Estados Unidos da América (EUA), para discutirem, sob a ótica de cada um, sobre as muitas questões que envolviam o

processo de desenvolvimento com XP. Neste evento, foi debatido como o XP e os processos conhecidos anteriormente como métodos leves (*Lightweight Methods*) se correlacionavam. Os métodos leves se guiavam em uma relação antônima com os métodos pesados que tinham como uma das principais características a formalização exacerbada das suas documentações e regulamentações, sendo influenciadas pelo modelo em cascata, que é burocrático.

Em decorrência de tal reunião, eles concluíram que o XP era melhor como um método específico, no entanto, asseguraram que havia um espaço comum entre o XP e os métodos leves. Um dos participantes desta reunião, Robert Cecil Martin, decidiu organizar um outro encontro com pessoas interessadas no tema de métodos leves. Várias pessoas foram contatadas, sendo elas, interessadas ou atuantes no meio. Em fevereiro de 2001, uma reunião realizada em Utah (EUA) marcou o surgimento e propagação do paradigma de desenvolvimento de *softwares* ágeis, mais tarde conhecido como manifesto ágil.

O Manifesto Ágil enumera pontos e boas práticas para ajudar com a melhoria nos processos de desenvolvimento. Ele vem ao encontro da valorização do trabalho do indivíduo para com o desenvolvimento (BECK, Kent et al):

- **Indivíduos e interação entre eles** mais que processos e ferramentas;
- ***Software* em funcionamento** mais que documentação abrangente;
- **Colaboração com o cliente** mais que negociação de contratos;
- **Responder a mudanças** mais que seguir um plano.

É atribuído mais valor aos itens em negrito, porém, não é negado o valor dos itens restantes.

Os doze princípios do manifesto ágil são:

1. Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de *software* com valor agregado;
2. Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas;
3. Entregar frequentemente *software* funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo;

4. Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto;
5. Construir projetos em torno de indivíduos motivados. Dando a eles o ambiente e o suporte necessário, e confiando neles para fazer o trabalho;
6. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face;
7. *Software* funcionando é a medida primária de progresso;
8. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente;
9. Contínua atenção a excelência técnica e bom design aumenta a agilidade;
10. Simplicidade: a arte de maximizar a quantidade de trabalho não realizado é essencial;
11. As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis;
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo;

Pressman (2011) alega que “se os membros da equipe de *software* devem orientar as características do processo que é aplicado para construir *software*, deve existir um certo número de traços-chaves entre as pessoas de uma equipe ágil e a equipe em si.”

Segundo Bernardo (2017), cada método ágil existente hoje carrega consigo os valores e princípios arraigados no manifesto ágil; por isso, métodos como SCRUM, KANBAN e XP são denominados ágeis. Os melhores métodos para o desenvolvimento de *software* devem ser encorajados e os processos ágeis são cabíveis desse encorajamento, já que em sua totalidade conseguem agregar o maior conjunto de benefícios na construção de um sistema.

2.3.1 Kanban

O Kanban é uma metodologia focada na utilização de *cards* que indicam o progresso de um projeto e é utilizada para melhor visualização de processos como um todo. Essa metodologia começou com o sistema Toyota de produção que começou

a se destacar nos meados dos anos 1970, em um período de crise e recessão econômica, quando o mercado americano via muitas empresas com seus lucros caindo.

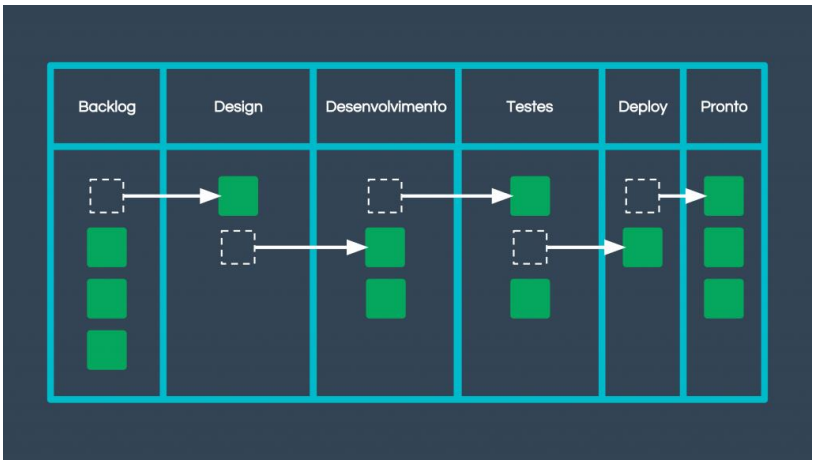
A Toyota, uma empresa japonesa de automóveis, àquela altura, apresentava bons lucros e vendas favoráveis, despertando a curiosidade do mercado americano. O sucesso na implantação do sistema Kanban na produção foi resultado obtido através da experiência adquirida da análise da produtividade industrial no Japão, com o objetivo de solucionar problemas que levaram as empresas e o país para uma grave crise no início da década de 1980 (RIBEIRO, 1999).

Tal desempenho excelente se deve ao sistema de produção enxuto (ou *just-in-time*[1]) desenvolvido pela Toyota com início na década de 1950. O *just-in-time* (JIT) pode ser definido como ter a “peça certa no momento certo e na quantidade certa, e podemos acrescentar no lugar certo” (ROTHER e SHOOK, 1998).

Quando se trata de melhoria da eficiência de produção em uma empresa, uma série de metodologias e ferramentas vem à mente. Uma das mais conhecidas e utilizadas atualmente é o método Kanban, que, em tradução literal, significa cartão (*cards)* ou sinalização. Por se tratar de um método em que seu objetivo é ser extremamente visual e intuitivo, várias empresas recorrem a ele para agilizar a entrega de seus projetos.

A figura 2 exemplifica como são utilizados os *cards* no sistema Kanban.

Figura 2. Estágios do Kanban e cartões



[1] *Just in time* (JIT) significa “no momento certo”. É um modelo japonês que procura eliminar estoques e agilizar a produção.

Fonte: TARGET TEAL, 2018.

O Kanban apresenta cinco regras, propostas por Villar et.al (p. 249, 2008):

1. O processo subseqüente (cliente) deve retirar do processo precedente (fornecedor) os itens necessários apenas nas quantidades e no tempo necessário. Em outras palavras, não se pode retirar nenhum tipo de item sem o correspondente Kanban que a ele é associado e as quantidades estabelecidas no Kanban têm que ser rigorosamente obedecidas.
2. O processo precedente (fornecedor) deve produzir seus itens somente nas quantidades requisitadas pelo processo subseqüente. Isto significa dizer que qualquer produção superior ao estabelecido nos kanbans é terminantemente proibida e que quando vários tipos de itens são produzidos há que se obedecer à seqüência original dos kanbans.
3. Produtos com defeitos não devem ser liberados para o processo subseqüente. Esta regra estabelece um pré-requisito para o funcionamento do sistema: trabalhar com qualidade.
4. O número de kanbans no sistema deve ser minimizado, objetivando manter as mínimas quantidades em estoque de itens em processo. A ideia é motivar os responsáveis pela produção na busca permanente de soluções que reduzam os inventários dos itens.
5. O sistema Kanban é usado para adaptar-se às pequenas flutuações na demanda. Os sistemas tradicionais não permitem uma reação rápida nas reprogramações, mesmo se tratando de pequenas flutuações.

2.4 TESTE DE *SOFTWARE*

Das várias definições de teste de *software*, pode-se destacar as seguintes:

- O processo de operação de um sistema ou componente em específicas condições, observando ou registrando os resultados, e fazendo uma avaliação em alguns aspectos do sistema ou componente (IEEE).

- É o processo de executar um programa ou sistema com a intenção de encontrar defeitos (Glen Meyers).
- Qualquer atividade que a partir da avaliação de um atributo ou capacidade de um sistema seja possível determinar se ele alcança os resultados desejados (Bill Hetzel).

De acordo com James Bach (1999),

> Teste de *software* é qualquer atividade que visa avaliar um atributo ou capacidade de um programa ou sistema e determinar se ele atende aos resultados necessários. [...] A dificuldade no teste de *software* decorre da complexidade do *software*: não podemos testar completamente um programa com complexidade moderada. Testar é mais do que apenas depuração. O objetivo dos testes pode ser garantia de qualidade, verificação e validação ou estimativa de confiabilidade.

Para o desenvolvimento de um produto, todo o cuidado é necessário. Isso porque dificilmente o mercado deixará um problema passar despercebido; falhas de funcionamento não são apenas notadas, como irão gerar reclamações e incidentes para outras áreas, podem ser publicadas e escancaradas, causando danos irreparáveis à imagem da empresa, perda de cliente e receita.

Uma solução é ter uma alternativa para apontar as falhas antes do lançamento de um produto, permitindo sua correção antecipada e evitando o desgaste junto aos usuários. É justamente nesse contexto que se conceitua o *Quality Assurance* (QA), que é o profissional responsável pela qualidade do produto, bem como pelas rotinas de testes.

QA faz referência a um profissional ou à uma equipe cuja função é garantir a qualidade no desenvolvimento de um produto ou serviço (GAEA, 2015). A qualidade de um produto ou serviço é fundamental para uma empresa, dado que estão em jogo a imagem da marca, a satisfação do cliente, oportunidades de negócios e a credibilidade da companhia. Por isso, o *Quality Assurance* é tão importante.

Em relação aos testes, pode-se ter teste manual, em que o testador testa manualmente o *software* ou parte dele e pode-se haver testes automatizados.

2.4.1 Testes Automatizados

Automatizar não significa substituir por completo os testes manuais, isso porque, em geral, estudos apontam que testes manuais podem encontrar mais

defeitos que os testes automatizados, pois, quando se interage diretamente com o sistema, sem a interface de um programa que automatiza todo o teste, consegue-se pensar em uma solução que, até então, não havia sido pensada apenas fazendo a leitura dos requisitos. Essa é uma das razões pela qual é preciso sabedoria e maturidade na área para se automatizar testes e ter consciência do planejamento e etapas a se seguir. É necessário que se tenha malícia para imaginar caminhos alternativos em situações que ainda não haviam sido previstas. A automação torna-se uma opção quando se tem certeza de uma certa instabilidade no *software*.

Dessa maneira, para entender o tempo que irá se gastar para criar um teste automatizado, é preciso, primeiro, compreender seu nível de complexidade, se seu intuito é a manutenção e, também, entender se existe a viabilidade de automação para os casos que a cada *release* o teste deve ser recriado.

Como escrever bons testes:

- criatividade;
- levar em consideração a experiência nos testes manuais;
- planejamento;
- o teste codificado deve ter a mesma atenção que código de sistema;
- o código precisa ser simples e legível;
- precisa ter manutenção;
- lembrar-se que os testes são passíveis de erros;
- abordar testes positivos e negativos;
- fazer com que o usuário se abra contigo para que possa criar mais situações de teste;
- os testes devem ser descritivos igual a um caso de teste;
- diminuir o gargalo de defeitos implica em ganho nas próximas versões;
- testar muito em pouco tempo.

Mais do que saber automatizar, o essencial é saber quando automatizar um teste, tendo em vista que com um teste manual consegue-se obter, em alguns casos, melhores resultados. Não existe regra para resolução de testes, é necessário que o profissional tenha sempre em mente o que é melhor para o seu sistema e o que ele quer extrair dele. Lembrando que o cliente sempre consegue descobrir uma brecha

no sistema; dado isto, pensar em muitos cenários de testes não é sempre uma tarefa trivial.

2.4.2 Testes de *Software* e Métodos Ágeis

Pensando na qualidade inserida no modelo ágil, Brian Marick (2003) apresenta os quadrantes que enfatizam a automação. O modelo de teste ágil foi pensado não apenas como uma ferramenta para detectar defeitos, nele também é analisado seu papel de especificação do comportamento do sistema e suporte à equipe de desenvolvimento; requisitos, testes e exemplos são aspectos diferentes das mesmas especificações.

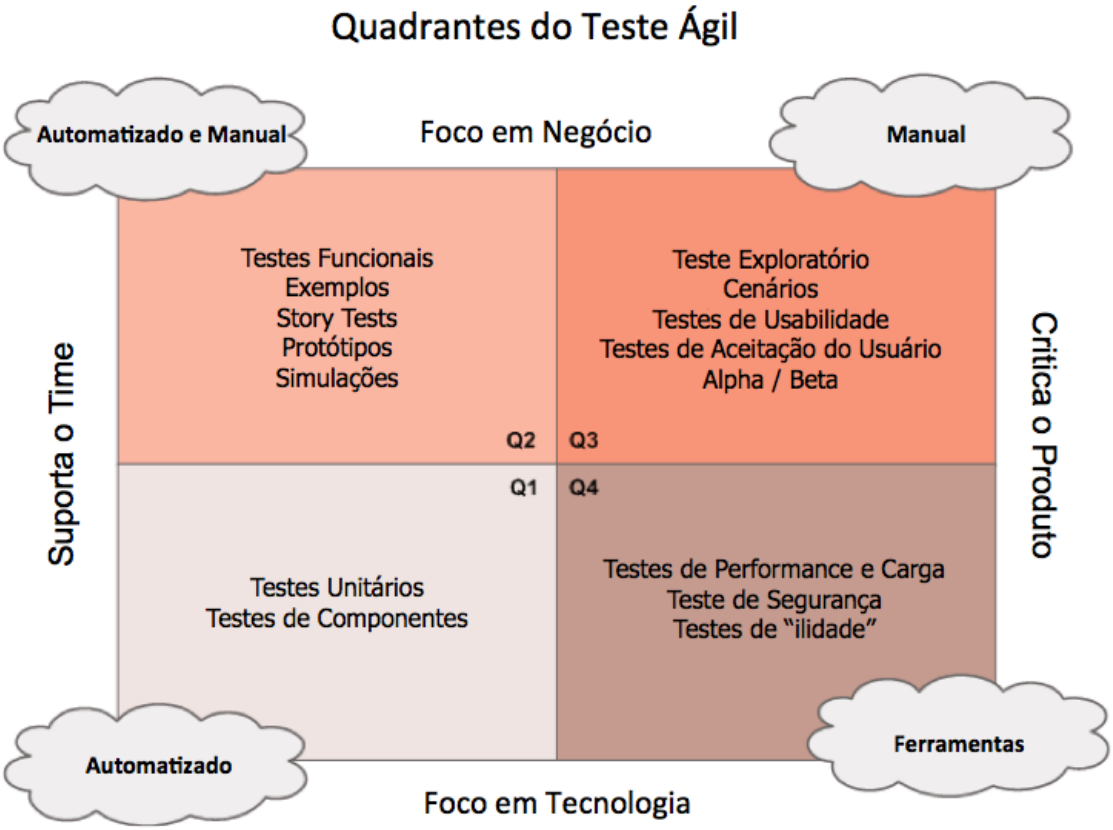
Dentre as razões citadas por Lisa Crispin e Janet Gregory (2016) estão:

- Teste manual é demorado;
- reduzir as tarefas de teste propensas a erros;
- liberar tempo para fazer o trabalho da melhor forma;
- prover segurança na mudança do código, pois o teste irá avisar se quebrou algo;
- prover *feedback* cedo e frequente;
- os testes provem uma excelente documentação;

De acordo com Crispin e Gregory (2009), no desenvolvimento ágil, o *tester* passa por uma transformação que fará com que ele trabalhe proativamente e com uma maior interação com os desenvolvedores e analistas de negócios.

Na figura 3, é apresentado o Quadrante de Teste Ágil, ele foi criado por Brian Marick, que apresentou vários termos para as mais diferentes categorias de testes, aumentando a participação e, por conseguinte, a qualidade do profissional responsável pelo teste.

Figura 3: Quadrantes de Marick



Fonte: Marick, B. 2003.

A ordem de numeração dos quadrantes pouco tem a ver com o momento em que cada teste será executado. O momento para cada tipo de teste relaciona-se com os objetivos e riscos que estão associados ao projeto.

De acordo com Crispin e Gregory (2009), cada quadrante representa:

Quadrante 1:

O Quadrante 1, Q1, representa o desenvolvimento orientado a testes, a principal prática de desenvolvimento ágil, o TDD. Neste quadrante, tem-se duas práticas: teste de unidade, que valida um pequeno subconjunto da aplicação como objetos e/ou métodos, e testes de componente, que valida partes maiores da aplicação como um grupo de classes que fornecem algum serviço. Com a técnica de TDD, o desenvolvedor pode desenvolver uma funcionalidade sem se preocupar em, posteriormente, modificar parte da aplicação, auxiliando nas decisões de arquitetura

e design. Os testes desenvolvidos são geralmente executados dentro de um comportamento de Integração contínua para prover, em um curto tempo, um *feedback* da qualidade do código. Neste quadrante, não há interação com o cliente.

Quadrante 2:

Este quadrante, Q2, também suporta o time de desenvolvimento, continua guiando o desenvolvimento, mas de uma maneira mais alto-nível, focando mais em testes que o cliente entenda, no qual este define a qualidade externa de que ele precisa. Neste quadrante, o cliente define exemplos que serão usados para um entendimento maior do funcionamento da aplicação e são escritos de forma com que o cliente ou papéis ligados a negócio entendam. Todo o teste executado nesta diagonal tem um foco no funcionamento do produto e alguns deles podem, até mesmo, ter uma pequena duplicação com alguns testes criados no Quadrante 1. Testes focados no negócio também podem ser automatizados e, usualmente, a técnica de BDD é utilizada na escrita e execução automatizada destes exemplos.

Quadrante 3:[2]

Os testes do quadrante 3, Q3, são utilizados para criticar o produto. Este quadrante serve para identificar se o *software* desenvolvido está dentro das expectativas ou não. Para isso, tenta-se simular o comportamento do utilizador final. Em certos casos, estes testes podem ser automatizados, mas em outros apenas a intervenção humana poderá ditar a qualidade do produto.

Quadrante 4:[2]

No Q4, os testes são mais técnicos e criticam o produto em termos de desempenho, carga e segurança. Atualmente, negligenciar aspectos como desempenho pode tirar a vantagem competitiva de um cliente. Geralmente, já se conhece aspectos relacionados a desempenho e à segurança quando se refina

[2] Os quadrantes 3 e 4 são conhecidos como testes que criticam o produto. O termo “criticam”, aqui apresentado, não é apenas de forma deletéria, mas também relacionados a melhoria. O foco é saber como o time irá aprender a melhorar o produto, escrevendo, se necessário, mais critérios e exemplos.

algumas histórias de usuários. As técnicas aplicadas a desempenho, carga e segurança vão desde os níveis mais baixos, como a utilização de ferramentas que simulam diversos usuários simultaneamente.

2.5 TDD

O TDD (*Test-Driven Development* ou Desenvolvimento Orientado por Teste) é uma metodologia ágil de desenvolvimento de *software* desenvolvida por Kent Beck (2003) para produzir o que se chama de “código limpo que funciona”. É uma técnica que consiste em pequenas iterações na qual novos casos de testes são escritos, contemplando uma nova funcionalidade ou melhoria e, depois, o código necessário e suficiente para passar esse teste é implementado, então, o *software* é refatorado para contemplar as mudanças de forma que os testes continuem passando.

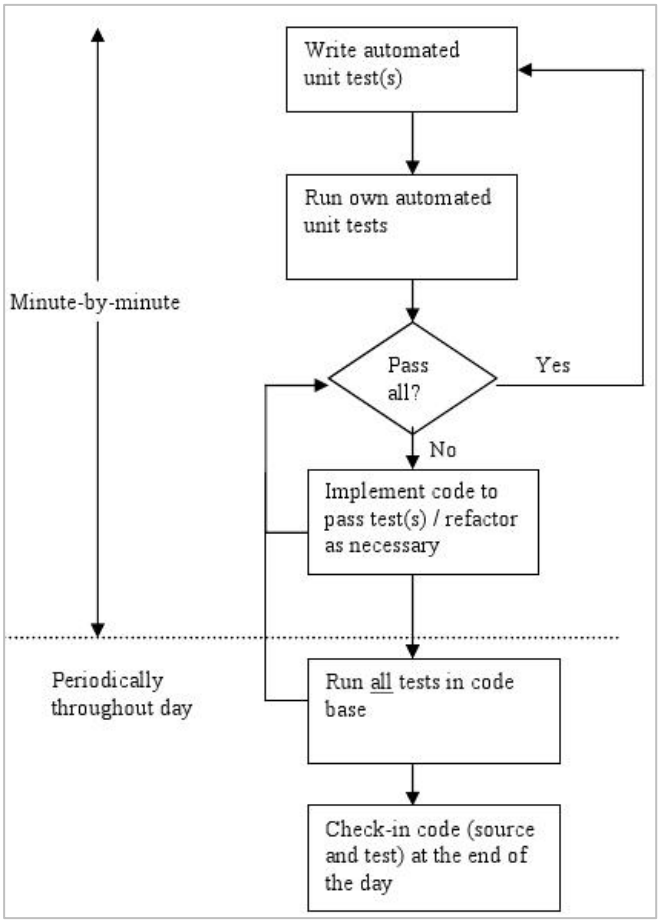
Apesar de se apresentar como uma técnica simples, ela ainda enfrenta grande dificuldade por parte de alguns desenvolvedores no que diz respeito à sua aplicação no dia a dia, não conseguindo guiar um projeto de TDD a partir de testes.

No TDD, os testes são pensados e escritos antes do código ser feito. Segundo Beck (2003), os seguintes passos são necessários para a utilização dessa técnica:

- Adicionar um novo teste;
- executar todos os testes;
- implementar o código necessário para passar no teste;
- executar todos os testes;
- refatorar o código.

A figura 4, de Bhat e Nagappan (2006), apresenta os passos do TDD.

Figura 4: Test Driven Development



Fonte: BHAT; NAGAPPAN, 2006, p.357.

Uma das grandes vantagens do TDD é o fato de se pensar no *design* antes de escrever, até mesmo, a primeira linha de código, tudo isso é pensado no momento em que se descreve o comportamento geral do sistema. É dessa forma que se nasceu outra metodologia ágil, na qual esse trabalho se baseia, o desenvolvimento orientado a comportamento (BDD). O BDD, com a sua evolução no tempo, passou a ser um processo que engloba desde a análise de requisitos até a escrita final do código fonte.

O problema com o TDD é a palavra “teste”. Quando se fala em teste, no sentido mais normativo da palavra, espera-se que o *software* já esteja em um processo de ser verificado e, como pode-se notar, o TDD é mais do que somente testes. North (2003) notou e criticou esse aspecto no TDD e, assim, criou o *Behavior-Driven Development* (BDD).

2.6 BDD

O BDD (Desenvolvimento Orientado por Comportamento) é uma técnica semelhante ao TDD em vários aspectos. O BDD transfere o foco dos testes de implementação para os comportamentos que o sistema expõe, ele é focado na colaboração entre desenvolvedores, QAs, analistas de negócios ou mesmo pessoas que não fazem parte da área técnica, visando integrar regras de negócios com linguagem de programação, focando o comportamento do *software*.

O foco em BDD é a linguagem e as interações usadas no processo de desenvolvimento de *software*. Desenvolvedores que se beneficiam destas técnicas escrevem os testes em sua língua nativa em combinação com a linguagem ubíqua (*Ubiquitous Language*[3]), isso permite que eles foquem no porquê o código deve ser criado ao invés de detalhes técnicos e, ainda, possibilita uma comunicação eficiente entre as equipes de desenvolvimento e testes.

A linguagem de negócio usada em BDD é extraída das estórias ou especificações fornecidas pelo cliente durante o levantamento dos requisitos. Alguns *frameworks* utilizam o conteúdo das estórias escritas em um arquivo de texto como cenários para os testes. Quando Dan North apresentou este conceito em 2003, ele sugeriu um padrão para escrita destes arquivos (DEVMEDIA, 2011). Uma das vantagens do BDD é o seu comportamento que se modifica menos vezes do que os testes, refletindo a necessidade de novas *features* do sistema.

Segundo Auvray (2009), o BDD (*Behavior Driven Development*) surge como uma resposta ao *Test Driven Development* (TDD). O BDD incentiva a colaboração de todos os que participam em um projeto de desenvolvimento de *software*, tais como: desenvolvedores, QA (*Quality Assurance*), PO (*Product Owner*), analistas não técnicos, analista de negócio e arquiteto de *softwares.*

O BDD surgiu por meio de práticas ágeis já estabelecidas e consolidadas no mercado e foi criado para torná-las mais acessíveis, com uma melhor aceitação e efetivação para equipes novas que trabalham com a linha de desenvolvimento ágil de *software*. Houve um hiato de tempo até que o BDD se tornasse maduro o suficiente

[3] Linguagem Ubíqua: é uma linguagem estruturada em torno do modelo de domínio e usada por todos os membros da equipe para conectar todas as suas atividades com o *software*.

para compreender um quadro mais amplo de análise ágil e de automação de testes de aceitação.

North (2003), ainda sugere que:

> A linguagem utilizada no BDD deve ser extraída nas especificações fornecidas pelo cliente durante o levantamento dos requisitos. Isso facilitará a comunicação e entendimento do time como um todo. O t*emplate* não pode ser engessado e nem artificial ao ponto de criar restrições aos analistas, mas estruturado suficiente de modo a quebrar a história em seus fragmentos constituintes e automatizá-las. (NORTH, 2003, p. 3)

Os critérios de aceitação foram descritos em termos de cenários que assumiram a seguinte forma:

- ***Given*** - Dado algum contexto inicial.
- ***When*** - Quando algum evento ocorrer.
- ***Then*** - Então verificar os resultados.

Para exemplificação, será usado o exemplo de um cliente realizando uma compra no cartão de crédito. Uma história pode ser assim:

+ **Título:** Cliente faz a compra +

Como um comprador,
Eu quero realizar a compra usando meu cartão de crédito,
Para que eu possa adquirir tal produto.

Existem vários cenários a considerar: a conta pode não ter crédito, o cartão pode estar bloqueado, a bandeira do cartão pode não ser aceita pela máquina, a máquina pode estar com defeito ou sem rede.

Usando o *template* dado acima, pode-se ter esse exemplo de cenário:

Critérios de aceitação (apresentado como cenário)

+ Cenário 1: O cartão possui limite +

Dado que há crédito no cartão

E a máquina não apresenta defeitos
Quando o cliente solicitar a comprar
Então garantir que a compra seja realizada
E garantir que o débito seja adicionado na conta do cliente

Pode-se usar o “E” para repetir o último comando múltiplas vezes.
A funcionalidade é escrita segundo o seguinte padrão:

Funcionalidade: [Nome]
Para [Valor ao Negócio]
Eu, como [Papel]
Desejo poder realizar [Funcionalidade]

Dessa forma, fica muito claro a todos quais são os objetivos que uma funcionalidade deseja atingir e é de muito fácil entendimento.

2.6.1 Vantagens do BDD

A vantagem de se usar o BDD em um projeto de TI é que pode-se começar por onde for mais confortável: pelo elemento mais básico do código que resolva o problema, a classe mais óbvia, o método mais fácil de fazer, o módulo com menos independências. O teste também evolui conforme o projeto progride. É necessário atenção às boas práticas: código limpo, orientação a objeto “de verdade”, sem “*Bad Smells*” ou “*eXtreme Go Horse Programming*”.

O principal problema na "testabilidade" do código é estar mal projetado, com alto acoplamento e várias responsabilidades. Portanto, código projetado para ser "testável" deve ser bem projetado, acima de tudo.

De acordo com Luz (2012), as vantagens do BDD são:

- **BDD utiliza linguagem natural para definir comportamentos da aplicação:** O uso da linguagem cotidiana facilita a comunicação dos membros do time. Desenvolvedores geralmente utilizam uma linguagem mais técnica e ao utilizar

menos esta abordagem a aproximação com pessoas de negócio e clientes se torna mais fácil.

- **Estimula participação do Cliente:** No BDD o próprio cliente pode definir comportamentos e escrever o que vai ser testado. Mesmo que não escreva, é ele quem valida esses comportamentos da aplicação. Também é mais fácil explicar, caso precise, como determinada funcionalidade está sendo testada. Logo, o cliente pode ter mais interesse em ficar ciente de mais esta etapa.
- **Artefatos semelhantes às práticas ágeis:** Uso de *user stories* e testes de aceitação é comum no BDD.
- **Favorece o desenvolvimento de forma evolutiva:** Podemos definir os sprints de acordo com os comportamentos definidos inicialmente.
- **Facilita a escrita de testes:** Apesar de ter uma filosofia parecida com o TDD, ao descrever comportamentos é mais confortável para o desenvolvedor escrever cenários e montar seus testes em cima disto, além de ser mais legível e de fácil entendimento, já que o nome dos métodos de teste dizem como a implementação deve se comportar.
- **Maior isolamento da aplicação:** Através da descrição de cenários e comportamentos, é mais simples identificar os componentes que envolvem determinada funcionalidade. Sem contar o fato de, ao melhorar o design, possivelmente diminui-se o acoplamento entre as classes.
- **Aumenta o entendimento do negócio por parte do time:** A implementação de comportamentos e cenários por parte dos desenvolvedores faz com que eles tenham um melhor entendimento das regras de negócio e do domínio da aplicação.
- **Aumenta a qualidade do *software*:** Por fim, ao realizar mais e melhores testes, tem-se menos erros e bugs, gerando um *software* com maior qualidade, deixando, então, o cliente mais satisfeito e alcançando seus objetivos de forma mais eficiente.

Segundo Anderle (2015, *apud* Soares I, 2011), também se têm as seguintes vantagens:

- **Comunicação entre equipes:** em algumas empresas de desenvolvimento de *software* nem sempre os desenvolvedores e testadores trabalham de uma forma unida, mas o BDD proporciona uma aproximação entre os membros da equipe em função dos testadores serem aptos a escreverem os cenários de teste para a equipe;
- **Compartilhamento de conhecimento:** tendo testadores e desenvolvedores trabalhando mais próximos, a tendência é que, com o passar do tempo, ocorra uma troca de conhecimento, fazendo com que se tenha uma equipe multifuncional;
- **Documentação dinâmica:** algumas equipes ágeis admitem que não documentam a aplicação em função de se tornar custosa. Usando frameworks de BDD, esta documentação será criada dinamicamente. Alguns geram relatórios em formato HTML, o que irá auxiliar uma consulta futura;
- **Visão do todo:** Fergus O'Connell, em sua obra *How to Run Successful High-Tech Project-Based Organizations* (Artech House, 1999), apresenta uma relação dos principais motivos que levam projetos de *software* ao fracasso. O primeiro deles é: "os objetivos do projeto não são bem definidos e compartilhados entre todos os envolvidos". Por este motivo, BDD sugere que os analistas/testadores escrevam os cenários antes mesmo dos testes serem implementados e, desta forma, os desenvolvedores terão uma visão geral do objetivo do projeto antes de codificá-lo.

  Das desvantagens, pode-se citar:

- Requer alto nível de engajamento e colaboração da equipe de negócio envolvida;
- BDD trabalha melhor em um contexto Ágil ou Interativo.

2.6.2 BDD x TDD

O modo como as empresas aplicam o TDD e o BDD na prática tende a variar muito. De acordo com Neto (2015), alguns dos fatores que afetam esta utilização são:

- **Porte dos Projetos –** Estudos mostram que o TDD obtém melhores resultados em projetos de grande porte [Nachiappan Nagappan et al, 2008]. Por este motivo, as empresas ou não aplicam TDD nos projetos pequenos ou "enxugam" o TDD para que ele possa ser aplicado nesses projetos.
- **Cultura da Empresa e Objetivos da Empresa –** Algumas empresas não possuem a cultura de testar o *software* desenvolvido por elas; o motivo para isso é bastante variado.
- **Experiência e Opinião de Colaboradores –** Não há consenso até mesmo entre especialistas na área. David Heinemeier, criador do "*Ruby on Rails*[4]", prega que o TDD está morto. Kent Beck vê potencial na técnica. Como colaboradores de empresas, essas visões afetam como eles trabalham e, consequentemente, na forma como os métodos são utilizados dentro das empresas.

Algumas das principais diferenças entre BDD e TDD são:

Quadro 1 – Diferenças entre BDD e TDD

TDD (Kent, 2003)	BDD (Dan North, 2004)
TDD é muito mais do que criar testes e ter qualidade no desenvolvimento de uma determinada funcionalidade;	Linguagem comum, onde todos podem se entender de uma forma melhor, aprofundar no entendimento do problema e aumentar a chance de construir algo correto, rede de segurança para efetuar mudanças;
Apesar de muitos acreditarem que a essência é codificar um sistema com 100% de cobertura dos testes;	Feedback rápido.
É a necessidade que um código deve atender;	Documentação "viva";

[4] Ruby on Rails é um *framework* livre que assevera aumentar velocidade e facilidade no desenvolvimento de sites orientados a banco de dados, uma vez que é possível criar aplicações com base em estruturas pré-definidas.

Escolhe-se um modelo como será atendida essa necessidade;	Reaproveitamento de código;
Garantia que o que for codificado atenda essa necessidade;	Mais fácil de saber onde começa e termina uma funcionalidade;
Está ligado ao projeto, ao design do código.	Entrega de valores ao cliente;

Fonte: Elaborado pelo autor.

2.6.3 Requisitos e *Users Stories* (Histórias de Usuários)

Num contexto em que se tem método ágil, obrigatoriamente, existe história de usuário. “Histórias de usuário” é a ferramenta mais utilizada para se levantar os requisitos ágeis. Independentemente do método ágil a ser aplicado, é quase sempre garantido que irá existir uma lista de histórias de usuário.

As *Users Stories* (US) surgiram no XP (*eXtreme Programming*), no final da década de 1980, o seu principal objetivo é representar o desejo do usuário por alguma funcionalidade. Além de representar um desejo, as histórias de usuário também devem ser de fácil priorização.

Casos de Uso e *User Stories* (US) são similares, como é apresentado por Fowler (2003):

> Ambos são utilizados para organizar requisitos. Porém, enquanto Casos de Uso descrevem ações de interação segundo uma narrativa impessoal entre o usuário e o sistema, *User Stories* focam nos objetivos do usuário e como o sistema alcança esses objetivos. (FOWLER, 2003, p. 312)

Muitas perguntas devem ser levantadas ao se escrever uma US, como por exemplo:

- *User Stories* são iguais Casos de Uso?
- Como descrevo minhas *User Stories*?
- Que tipo de informações pode se inserir nas *User Stories*?
- De quem e para quem são feitas?

As respostas para essas perguntas são amplamente discutidas nas reuniões de Kanban, juntamente com toda a equipe técnica envolvida no projeto.

US dividem os requisitos para que seja possível, e mais fácil, estimar o esforço até que se alcance tal objetivo. À priori, US são descrições simples que descrevem uma funcionalidade e é recomendável que sejam escritas segundo o ponto de vista do usuário, sendo elas curtas, simples e claras.

3 METODOLOGIA

Neste capítulo, é apresentado o estudo de caso sobre a utilização de práticas de usabilidade e testes automatizados em uma empresa de desenvolvimento de *software*, no qual será definido um guia de como essas práticas podem ser inseridas nesse contexto.

O objetivo global desse estudo é analisar como o BDD tem auxiliado o grupo de QAs de uma *startup martech*, no time de DevOps da mesma, e introduzir técnicas de usabilidade no desenvolvimento empírico a partir das práticas de BDD, tendo em vista a utilização de várias técnicas de usabilidade e de testes de *software*.

3.1 APRESENTAÇÃO DA EMPRESA

A empresa em que este estudo de caso foi aplicado está situada na cidade de Ouro Preto, no estado de Minas Gerais, Brasil. Devido a dados sigilosos de clientes e funcionários, nomes não serão usados explicitamente neste trabalho. Doravante, a empresa será identificada apenas como “empresa X” e os participantes da pesquisa serão identificados como QA1, QA2, QA3, etc.

A empresa X, deste estudo, é de desenvolvimento de *software* do setor de serviços (SAAS – *Software as a Service*), de grande porte (mais de 100 funcionários). A empresa utiliza o método ágil Kanban juntamente com o SCRUM para o auxílio das organizações de suas atividades e desenvolvimento dos *softwares* que a ela produz.

Dentro do escopo empresarial, ela se destaca no ramo de Marketing, possui cinco anos no mercado e é pioneira no Brasil no serviço que oferece, tendo grandes marcas, nacionais e internacionais, em seu catálogo de clientes.

Internamente, a empresa é subdividida em *squads*. Cada *squad* é responsável por uma área de desenvolvimento do produto, seja ele já lançado ou a ser lançado. Dentro de cada *squad*, geralmente, há:

- 1 PO (profissional que define os itens que compõem o *Product Backlog* e os prioriza nas Sprint Planning Meetings);
- 1 desenvolvedor *back-end* (esponsável por “dar vida” à interface. Trabalha com a parte da aplicação que interage diretamente com o usuário);
- 1 desenvolvedor *front-end* (trabalha na parte de “trás” da aplicação. Ele é o responsável, em termos gerais, pela implementação da regra de negócio);

- 1 *Agile Master* (é um facilitador, que garante ao time ágil o fornecimento de um ambiente propício para concluir com sucesso o projeto);
- 1 QA;
- 1 desenvolvedor de banco de dados (profissional responsável por atuar com administração de banco de dados, desenvolver melhorias, identificar e solucionar problemas);
- 1 SER (*Site Reliability Engineer* - o objetivo do SRE está em encontrar formas para aprimorar o design e a operação dos sistemas para fazê-los mais escaláveis, confiáveis e mais eficientes);
- Designer (UI/UX - UI *Design*, ou Design de Interface do Usuário, é o meio pela qual uma pessoa interage e controla um dispositivo, software ou aplicativo. UX *Design*, está relacionado com a experiência do usuário e com seus sentimentos).

A quantidade de profissionais dentro de cada *squad* pode variar para mais. Há apenas 1 QA que não trabalha dentro do esquema de *squads*.

3.2 DELINEAMENTO DA PESQUISA

Para o presente trabalho, foi utilizado o método de pesquisa qualitativa. Segundo Oliveira (2012, *apud* Lüdke e André, 1986, p. 13), dentro de uma vertente qualitativa, apresentamos dois tipos de se fazer pesquisa nessa área. De acordo isso, a pesquisa etnográfica e o estudo de caso “vêm ganhando crescente aceitação na área de educação, devido principalmente ao seu potencial para estudar as questões relacionadas à escola”. (LÜDKE, 1986, p. 13)

Consoante a Godoy (1995), o caráter qualitativo é dado quando se têm questões ou focos de interesse amplos, que vão se definindo à medida que o estudo se desenvolve, portanto, justifica-se, assim, a abordagem qualitativa. Ainda, segundo o autor, quando a preocupação for a compreensão da teia de ralações sociais e culturais que se estabelecem no interior das organizações, o trabalho qualitativo pode oferecer evidências interessantes e relevantes.

Esta pesquisa foi realizada na empresa X, onde se escolheu o técnicos do time de QA que trabalham com metodologias ágeis em seus projetos (Kanban/SCRUM), pois são eles que ficam com o cargo de testar e cobrar que o resto do time utilize, de

maneira coesa e clara, as US e os requisitos para que se garanta a qualidade final do produto, evitando o retrabalho da parte da equipe como um todo, tendo, então, como objetivo: identificar quais gargalhos encontram-se nos processos atuais da empresa X e propor uma melhoria em seu processo de desenvolvimento com o uso do BDD. Como resultado, os dados desta pesquisa serão comentados com relação ao êxito ou não dos objetivos propostos.

3.3 TÉCNICA PARA A COLETA DE DADOS

Para desenvolver este estudo, foi utilizada a técnica de observação do trabalho realizado pelo time e também foi elaborado um questionário formado por um conjunto de questões que são respondidas por escrito pelo pesquisado (GIL, 2002).

As questões foram criadas com base na literatura, tendo como referência o que os autores identificaram como sendo os benefícios do BDD (Item 2.6.1) dentro de um projeto, com o objetivo de obter respostas e analisar se estas responderão as questões ao qual o objetivo do trabalho se propõe.

De acordo com Gil (1994), é importante determinar o número adequado de perguntas, levando em consideração o possível interesse dos respondentes pelo tema (1994, p. 129).

Passo 1 – Fez-se um levantamento de quais *squads* dentro da empresa usam o Kanban e poderiam ser analisadas os gargalos e dificuldades que os QAs enfrentam, além de melhorias no processo, desde a aplicação do BDD.

Passo 2 – Foi realizado um primeiro contato pessoalmente com os integrantes do time pedindo autorização para a capitã do time de QA da empresa X.

Passo 3 – Com o consentimento do colaborador, de posse do questionário gerado em formato de formulário, estes foram enviados via Google Forms (Apêndice A). O colaborador, assim, poderia responder eletronicamente. Na maioria das questões, se fez necessário um questionamento usando o “Por quê?” tanto para quando a resposta fosse “Sim” quanto para quando a resposta fosse “Não”, sendo que todas as questões eram de caráter obrigatório, com respostas abertas.

Passo 4 - Após o envio do questionário aos colaboradores, foi estabelecido um prazo de retorno (1 semana).

Inicialmente, foram considerados todos os projetos da empresa da parte de produto e tecnologia, que possui, no mínimo, um QA em seu time, totalizando, então, 6 times, com 6 entrevistados. Apenas um dos *squads* não utiliza metodologia ágil, pois este trata de *bugs* que já se encontram em produção (já foram lançados para clientes), sendo uma característica ímpar deste *squad*, a resposta do QA envolvido também foi considerada para este estudo, porém com menos significância nas considerações finais.

As questões foram adaptadas do estudo “Introdução de BDD (*Behavior Driven Development*) como Melhoria de Processo no Desenvolvimento Ágil de *Software*”, de autoria de Angelita Anderle (2015).

O quadro 2 apresenta as questões respondidas e as suas referências teóricas.

Quadro 2 - Questionário sobre o uso do BDD em um projeto ágil

Questão:	**Referência Teórica:**
1 - Há quanto tempo você trabalha com desenvolvimento de *software* Ágil?	Elaborada por Anderle (2015): com o intuito de saber quanto tempo o colaborador possui de experiência com desenvolvimento ágil.
2 - Você vê valor no uso do BDD no seu projeto? Sim / Não. Por quê?	Smart (2015) - o BDD ajuda as equipes a concentrar seus esforços na identificação, compreensão e na construção de aplicações que agreguem valor para o negócio, e certificasse que estas aplicações são bem projetas e bem implementadas.
3 - No seu projeto a US (*user story*) é escrita em formato de BDD? Se sim, tendo as US escritas no formato BDD, você acredita que isso trouxe um ganho de qualidade no desenvolvimento do *software*?	Rocha (2013) - Evita-se erros de compreensão e interpretação das histórias de usuário; North (2003) - BDD traz uma “linguagem única” para análise;
4 - Você acredita que com o uso do BDD todos da equipe tiveram um melhor entendimento do que era requisitado? Sim / Não. Por quê?	Rocha (2013) - O que antes eram requisitos funcionais e requisitos não funcionais são transformados em

	comportamentos funcionais do *software* e critérios de aceitação; North (2003) - Isso facilitará a comunicação e entendimento do time como um todo.
5 - Você acredita que com o uso do BDD melhorou a comunicação entre os membros da equipe? Sim / Não. Por quê?	Smart (2015) – Comunicação (encoraja analistas de negócios, desenvolvedores de *software* e testadores a colaborar mais de perto). Soares I (2011) – Comunicação entre equipes.
6 - Você acredita que com o uso do BDD o compartilhamento do conhecimento entre os membros da equipe (por ex: QA, DEV, PO, Agile Master) melhorou? Sim / Não. Por quê?	Soares I (2011) - Compartilhamento de conhecimento. Rocha (2013) - As reuniões de planejamento, revisão e diárias tornam-se mais eficazes.
7 - Quais as vantagens e desvantagens que você vê com o uso do BDD?	Smart (2015) - *Releases* mais rápidas; North (2003) - Os testes são focados no que realmente tem valor para o usuário;
8 - Relacionado ao esforço do desenvolvimento, você acredita que houve menos retrabalho com o uso do BDD? Sim / Não. Por quê?	Smart (2015) - Redução do desperdício; Redução de custo.
9 - De uma maneira geral, após a introdução e a aplicação do BDD no projeto, você acredita que foi possível transformar os requisitos em um produto que agrega valor ao cliente?	North (2003) - BDD tenta ajudar os desenvolvedores no foco sobre o real valor a ser agregado ao cliente;

Fonte: Elaborado pelo autor.

3.4 TÉCNICAS DE ANÁLISE DOS DADOS

Após o envio do questionário, e ele ter sido respondido pelos entrevistados, foi realizada uma análise das respostas, que é apresentada no próximo capítulo, na qual é verificada a eficácia, ou não, da introdução do uso de BDD no processo de desenvolvimento de *software*.

Todas as respostas encontram-se no Apêndice B. Não foi realizada a análise da questão 1, uma vez que esta não é diretamente ligada à aplicação do método em si, mas sim da experiencia do entrevistado com o método ágil.

Como todas as questões são abertas, obtêm-se respostas dissertativas dos participantes. A técnica comumente usada nesse tipo de caso com respostas qualitativas é a análise de conteúdo.

Segundo Flick (2009):

> “Na síntese da análise de conteúdo, o material é parafraseado, o que indica que termos e paráfrases menos pertinentes que possuam significados iguais são omitidos. E ainda ressalta que esse tipo de análise é clássico para analisar materiais textuais sendo estes gerados através de entrevistas, mídia e outros”. (FLICK, 2009, p.107)

4 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

4.1 ANÁLISE DAS RESPOSTAS DO QUESTIONÁRIO DE PESQUISA

A análise dos dados do questionário foi realizada com o auxílio do Microsoft Office Excel. Nesta etapa, analisou-se e obteve-se os indicativos dos resultados coletados.

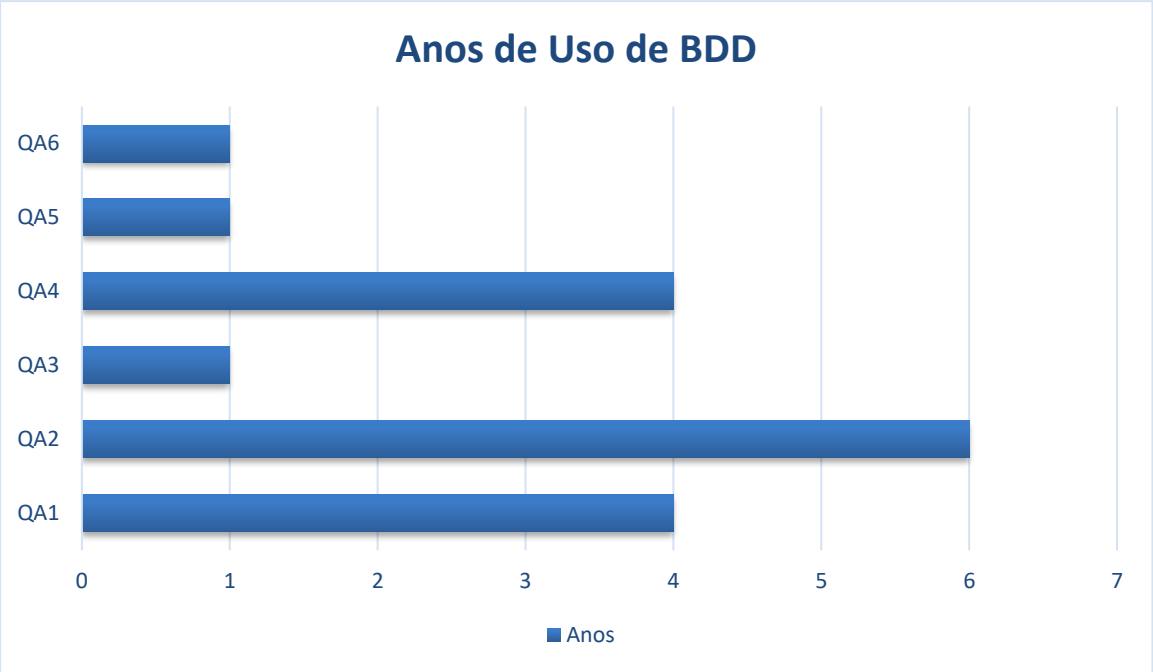
Logo abaixo, são exibidos os resultados obtidos com esta pesquisa. Obteve-se 100% dos questionários respondidos pelos QAs da empresa X, de todos os *squads*, totalizando 6 questionários respondidos.

1. *Há quanto tempo você trabalha com desenvolvimento de software Ágil?*
   - Identificou-se que a média de anos que os entrevistados trabalham com desenvolvimento ágil é de 2,8 anos; ainda, pode-se observar que metade dos entrevistados ainda são iniciantes na metodologia ágil.

Quadro 3 - Funções X Anos de trabalho com o desenvolvimento ágil

QA1	4 anos
QA2	6 anos
QA3	1 ano
QA4	4 anos
QA5	1 ano
QA6	1 ano

Fonte: Elaborado pelo autor.



Fonte: Elaborado pelo autor.

2. *Você vê valor no uso do BDD no seu projeto? Sim / Não. Por quê?*
   - Todos os participantes afirmaram que veem valor no uso do BDD no cotidiano do seu projeto, principalmente na parte de expansão do conhecimento entre os participantes do projeto.

3. *No seu projeto a US (User Story) é escrita em formato de BDD? Se sim, tendo as US escritas no formato BDD, você acredita que isso trouxe um ganho de qualidade no desenvolvimento do software?*
   - Todos os participantes responderam que sim, afirmando que quando a maioria das US são escritas, utilizando a metodologia ágil, auxiliam no entendimento do comportamento do *software*.

4. *Você acredita que com o uso do BDD todos da equipe tiveram um melhor entendimento do que era requisitado? Sim / Não. Por quê?*
   - Todos os participantes afirmaram que sim, destacaram, principalmente, sobre como o BDD auxilia na melhor escrita das tarefas a serem feitas e/ou implementadas no sistema.

5. *Você acredita que com o uso do BDD melhorou a comunicação entre os membros da equipe? Sim / Não. Por quê?*
   - Os participantes responderam que sim, em sua totalidade, destacando como o processo foi melhorado desde que o BDD passou a ser implantado na empresa, principalmente no passo de se entender quando há uma nova *feature* a ser implementada. Isso destaca como a equipe conseguiu uma melhor comunicação verbal e escrita utilizando o método ágil.

6. *Você acredita que com o uso do BDD o compartilhamento do conhecimento entre os membros da equipe (por ex: QA, DEV, PO, Scrum Master) melhorou? Sim / Não. Por quê?*
   - Todos responderam que sim, destacando que há um consenso na parte da escrita do BDD, já que não é uma linguagem estritamente técnica, possibilitando que todos da equipe possam propor melhorias e discutir sobre a funcionalidade nova.

7. *Quais as vantagens e desvantagens que você vê com o uso do BDD?*
   - Das vantagens e desvantagens, destacam-se, respectivamente:
     1. Foco na entrega de valor da funcionalidade que a empresa quer alcançar e não foco no MVPI (entrega individual).
     2. Mensurar a funcionalidade e não necessariamente a qualidade da entrega, a não ser que tenha um desenvolvimento orientado a teste.

8. *Relacionado ao esforço do desenvolvimento, você acredita que houve menos retrabalho com o uso do BDD? Sim / Não. Por quê?*
   - Houve consenso entre a maioria, apenas um QA que não concordou, dizem que em alguns momentos sim, em outros não, já que o BDD não é suficiente se não for implementado junto com outras ferramentas e outros mecanismos de facilitar o planejamento de desenvolvimento.

9. *De uma maneira geral, após a introdução e a aplicação do BDD no projeto, você acredita que foi possível transformar os requisitos em um produto que agrega valor ao cliente?*

- Houve unanimidade, todos os participantes concordaram com a afirmação, dizendo que é possível sim transformar os requisitos em um produto que agrega valor ao cliente.

5 CONSIDERAÇÕES FINAIS

Durante o desenvolvimento deste trabalho, estudou-se o desenvolvimento de *software* e suas características, focando nos métodos ágeis e nas práticas de *software* livre. Assim como a forma que esse desenvolvimento está ligado com práticas de BDD e como estas práticas (testes de aceitação) podem ser aplicadas ou não em um ambiente real de desenvolvimento que já se encontra estabelecido e engessado, além da aplicação de práticas de melhorias de usabilidade.

Ao término deste, pode-se observar que os objetivos gerais e específicos foram atingidos, uma vez que foi possível modificar o ambiente proposto pelo estudo e obter um bom resultado que foi capaz de apresentar argumentos, dando créditos ao uso do BDD como uma melhoria de processo.

Os aspectos da qualidade fazem parte do cotidiano do time e após a implantação do BDD pode-se observar que isto proporcionou uma melhora no processo de desenvolvimento ágil para os projetos. Essa percepção pode ser notada por meio das respostas obtidas, pois elas revelam que ocorreu uma melhora em vários sentidos relacionados ao entendimento das US.

Podemos destacar alguns pontos como:

- Todos conseguiram perceber valor no uso do BDD no projeto em que estavam trabalhando e identificam vantagens no seu uso;
- Aumento da qualidade em função de ter os cenários mais detalhados e podendo identificar o que era esperado;

A contribuição deste estudo para a área de qualidade de *software* destaca-se pela melhoria de processo usando o BDD e de como seus resultados foram positivos para o projeto.

À partir deste estudo de caso, verificou-se que as práticas de BDD são utilizadas constantemente durante o desenvolvimento empírico, porém existe uma certa dificuldade em aderir tanto práticas de testes, quanto práticas de usabilidade pela equipe de desenvolvimento, principalmente durante *sprints* de final de *release*, onde o prazo de entrega encontra-se mais perto, e a equipe tende a se preocupar mais com o desenvolvimento da funcionalidade em si. Porém quando essas práticas são utilizadas de forma frequente e como planejadas, verificou-se que o desenvolvimento dos testes de aceitação, assim como o uso de protótipos, pode

agilizar o processo de avaliações de usabilidade por meio de *checklists* e heurísticas, pois não existe a necessidade de uma funcionalidade está totalmente desenvolvida.

REFERÊNCIAS

ANDERLE, Angelita. **Introdução de BDD (Behavior Driven Development) como Melhoria de Processo no Desenvolvimento Ágil de Software.** UNISINOS. São Leopoldo. 2015.

Bach, J. (1999). **Reframing requirements analysis.** Computer, 32(2), 120–122.

Bhat, T., Nagappan, N**.: Evaluating the efficacy of test-driven development: industrial case studies.** In: ISESE 2006: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering, pp. 356–363. ACM, New York (2006).

BERNARDO, P. C. **Padrões de testes automatizados.** Dissertação (Mestrado) — Instituto de Matemática e Estatística – Universidade de São Paulo, 2017. Disponível em: <http://www.teses.usp.br/teses/disponiveis/45/45134/tde-02042012-120707/>. Acesso em 29 set 2019.

BECK, K. **Test Driven Development: By Example.** [S.l.]: Addison-Wesley Professional, 2003.

CAMPOS, Vicente Falconi. **Controle da Qualidade Total.** Rio de Janeiro: Editora Bloch, 3ª edição, 1992.

CRISPIN, Lisa; GREGORY Janet. **Agile Testing: A Practical Guide for Testers and Agile Teams**. 1. Ed. Addison-Wesley Professional. Boston, 2009.

CROSBY, Philip B. ***Quality is Free.*** Nova Iorque: New American Library, 1979.

DEVMEDIA. ***Qualidade de Software.*** 2011. Disponível em: <https://www.devmedia.com.br/qualidade-de-software/9408> Acesso em 1 out 2019.

FLICK, Uwe. **Introdução à Pesquisa Qualitativa**. 3.ed. São Paulo: Artmed, 2009.

FOWLER, M. ***Use Cases and Stories***. EUA: Thought Works. 2003.

GODOY, Arilda Schmidt. Introdução à Pesquisa Qualitativa e suas Possibilidades. RAE – **Revista de Administração de Empresas**. São Paulo, v.35, n2, p.57-63. 1995.

ISO/IEC 9241-11. **NBR ISO/IEC 9241-11: ERequisitos ergonômicos para o trabalho com dispositivos de interação visual Parte 11: Orientações sobre usabilidade.** [S.l.], 2003.

McCall, J.A., Richards, P.K. and Walters, G.F. (1977) **Factors in Software Quality.** RADC TR-77-369, Rome Air Development Center, Rome

OLIVEIRA, S. R. B. **“Conceitos Fundamentais de Qualidade de Software”.** 2012.

<http://www.ufpa.br/srbo/disciplinas/cbcc_cbsi_mestrado_qualidade/aulas/aula02.pdf > Acesso em 18 out. 2019.

PRESSMAN, R. S. **Engenharia de Software.** [S.l.]: MAKRON Books, 2005.

PRESSMAN, Roger S. **Engenharia de software: uma abordagem profissional**. 7. ed. Porto Alegre: Artmed, 2011.

RIBEIRO, Vera Masagão. **"Ensino fundamental de jovens e adultos: Idéias em torno do currículo"**. In: Seminário Internacional de Educação e Escolarização de Jovens e Adultos: Experiências internacionais Educação & Sociedade, ano XX, nº 68, Dezembro/99 201 [Trabalhos apresentados]. São Paulo: MEC/Ibeac, v. 1, 1998, pp. 225-233. Alfabetismo e atitudes: Pesquisa com jovens e adultos. São Paulo; Campinas: Ação Educativa/Papirus, 1999.

ROCHA, G. Fabio. **Scrum e BDD: o casamento perfeito.** 2013. Disponível em: <http://www.devmedia.com.br/scrum-e-bdd-o-casamento-perfeito/28174> Acesso em 12 nov. 2019.

ROTHER, M. & SHOOK, J. (1998)- ***Learning to See - Value Stream Mapping to Add Value and Eliminate Muda.*** The Lean Enterprise Institute, MA, USA.

SMART, John F. (*Foreword By Dan North*) **BDD in Action: Behavior Driven Development for the whole software lifecycle.** Manning Publications Co. NY. 2015.

SOARES, I. **Desenvolvimento orientado por comportamento (BDD) - Um novo olhar sobre o TDD.** Java Magazinne, Rio de Janeiro, v. I, n. 91, 2011. Disponível em: <http://www.devmedia.com.br/desenvolvimento-orientado-por-comportamento-bddartigo- java-magazine-91/21127> Acesso em 01 nov. 2019.

SOMMERVILLE, I. **Software Engineering.** [S.l.]: Addison Wesley Professional, 2007.
GIL, Antonio Carlos. **Métodos e Técnicas de Pesquisa Social.** 4.ed. São Paulo: Atlas, 1994.

The Institute of Electrical and Electronics Engineers. **IEEE Std 829: Standard for Software Test Documentation**. New York: IEEE Computer Society, 1998.

VILLAR, A. M.; SILVA, L. M. F. & NÓBREGA, M. M. **Planejamento, Programação e Controle da Produção.** Editora Universitária da UFPB, João Pessoa – PB, 2008.

Wikipedia **Behavior Driven Development**. 2015. Disponível em: <http://pt.wikipedia.org/wiki/Behavior_Driven_Development> Acesso em 14 out. 2019.

APÊNDICE A – Plataforma de Coleta de Dados

Formulário BDD

Descrição do formulário

Nome (pode ser só o primeiro nome, lembrando que nenhum dado pessoal será divulgado no trabalho final) *

Texto de resposta curta

Há quanto tempo você trabalha com desenvolvimento de software Ágil? *

Texto de resposta curta

Você vê valor no uso do BDD no seu projeto? Sim / Não. Por quê? *

Texto de resposta longa

No seu projeto a US (user story) é escrita em formato de BDD? Se sim, tendo as US escritas no formato BDD, você acredita que isso trouxe um ganho de qualidade no desenvolvimento do software? *

Texto de resposta longa

Você acredita que com o uso do BDD todos da equipe tiveram um melhor entendimento do que era requisitado? Sim / Não. Por quê? *

Texto de resposta longa

Você acredita que com o uso do BDD melhorou a comunicação entre os membros da equipe? Sim / Não. Por quê? *

Texto de resposta longa

Você acredita que com o uso do BDD o compartilhamento do conhecimento entre os membros da equipe (por ex: QA, DEV, PO, Agile Master) melhorou? Sim / Não. Por quê? *

Quais as vantagens e desvantagens que você vê com o uso do BDD? *

Texto de resposta longa

Relacionado ao esforço do desenvolvimento, você acredita que houve menos retrabalho com o uso do BDD? Sim / Não. Por quê? *

Texto de resposta longa

De uma maneira geral, após a introdução e a aplicação do BDD no projeto, você acredita que foi possível transformar os requisitos em um produto que agrega valor ao cliente? *

Texto de resposta longa

APÊNDICE B - Respostas dos colaboradores

2. Há quanto tempo você trabalha com desenvolvimento de software Ágil?

QA1 = “4 anos”

QA2 = “6 anos”

QA3 = “1 ano”

QA4 = “4 anos”

QA5 = “1 ano”

QA6 = “Pouco mais de um ano”.

3. Você vê valor no uso do BDD no seu projeto? Sim / Não. Por quê?

QA1 = “Demais”.

QA2 = “Muito”.

QA3 = “Sim” .

QA4 = “Sim, expandir os conhecimentos entre todos os envolvidos pra saber cada cenário e os desenvolvedores e QA conseguiam ter uma ideia mais clara das

necessidades do negócio, facilitando na compreensão do que realmente precisava ser desenvolvido e testada".

QA5 = "Sim, deixa o entendimento das tarefas mais simples, facilitando o desenvolvimento da mesma".
QA6 = "Sim, porque facilita o processo de reconhecimento entre a *feature* e o time de desenvolvimento, porque auxilia nos cenários de teste e porque ajuda na comunicação".

4. No seu projeto a US (*User Story*) é escrita em formato de BDD? Se sim, tendo as US escritas no formato BDD, você acredita que isso trouxe um ganho de qualidade no desenvolvimento do software?

QA1 = "Não".
QA2 = "Sim, quando a maioria das US são escritas utilizando a metodologia ágil auxilia no entendimento do comportamento do software. Além disso, facilita na hora de criar mais cenários de teste, uma vez que os critérios de aceite não obrigatoriamente cobre todas as condições."
QA3 = "Sim, melhorou demais depois que começamos a usar o bdd."

QA4 = "Sim, quando chega no desenvolvimento já entendeu o que é pra ser feito, sem nenhuma dúvida e nas escritas do BDD já sabe qual seria o comportamento real do cliente".
QA5 = "Sim, com certeza".
QA6 = "Há pouco tempo alguns times começaram a utilizar o BDD, desde que o time aderiu ao BDD notou-se que as USs estão mais bem descritas, principalmente porque anteriormente algumas USs eram aprovadas e desenvolvidas pelo time sem sequer ter critério de aceite. A melhoria na escrita das USs auxilia em todo o processo de desenvolvimento, desde o entendimento da feature até a criação de cenários para automatização de testes".

5. Você acredita que com o uso do BDD todos da equipe tiveram um melhor entendimento do que era requisitado? Sim / Não. Por quê?

QA1 = "Não participo de um time que usa o bdd, mas acredito que os times que usam têm este ganho, já que as especificações ficam mais claras."

QA2 = “Sim, pois melhora a comunicar entre todas as áreas envolvidas, onde todos falam a mesma língua, mas para áreas mais técnicas não necessita da escrita no formato Gherkin, uma vez que a alteração é um serviço ou uma parte específica do código, e não um comportamento ao navegar no sistema. (backend, frontend ou configuração de uma ferramenta de alerta) Nesse caso, é escrita uma task de forma mais técnica e objetiva, e pode ser escrita com o auxílio do desenvolvedor”.
QA 3 = “Sim, o uso do bdd deixa mais claro o que tem que ser feito e qualquer um pode entender, não precisa ser técnico, isso traz mais alinhamento para todo o time.”
QA4 = “Sim. Aproximar toda a equipe buscando colaboração e comunicação e entendimento para sanar quaisquer dúvidas antes do desenvolvimento, envolve integrante da equipe com mesmo conhecimento gerando um ciclo de vida saudável e maduro na construção de um software.”
QA5 = “Sim, porque com o bdd a explicação da tarefa fica simples e clara”.
QA6 = “Sim, tendo vista que a partir dos critérios de aceite as USs ficaram mais bem definidas, sendo melhor entendida pelo PO e gerando maiores questionamentos no toma na hora de pensar em seu desenvolvimento”.

6. Você acredita que com o uso do BDD melhorou a comunicação entre os membros da equipe? Sim / Não. Por quê?

QA1 = “Não participo de um time que usa o bdd, mas acredito que os times que usam têm este ganho, já que as especificações ficam mais claras.”
QA2 = “Sim, uma vez que no time tem áreas que não são de tecnologia. (Design, agilidade, produto)”.
QA3 = “Sim, como a escrita do bdd não é algo técnico, todos do time podem propor melhorias e discutir sobre a funcionalidade proposta”.
QA4 = “Sim, BDD faz compartilhar o conhecimento entre desenvolvedores e *testers* para desenvolver o software, um transmite conhecimento para o outro e torna a equipe mais coesa tecnicamente”.
QA5 = “Sim, porque com todo mundo entendendo o objetivo da tarefa a comunicação fica mais fácil entre os membros da equipe”.
QA6 = “Sim. A partir dos critérios de aceites nem definidos no formato do BDD o time começou a questionar mais e a discutir mais sobre a feature que tem como objetivo entregar valor”.

7. Você acredita que com o uso do BDD o compartilhamento do conhecimento entre os membros da equipe (por ex: QA, DEV, PO, Agile Master) melhorou? Sim / Não. Por quê?

QA1 = “Não participo de um time que usa o bdd, mas não acho que o bdd tenha grande impacto na questão de compartilhar conhecimento, pois não é o seu foco”.
QA2 = “Sim, pois facilita o entendimento do que é esperado e interfere/ melhora na qualidade do que é entregue para o cliente final, podendo aumentar a satisfação do cliente, uma vez que pode focar no design e na regra de negócios afim de entregar a real necessidade do cliente”.
QA3 = “Sim, com o bdd nós forçamos que o PO tenha um entendimento total do que deve ser feito, isso aumenta o conhecimento dele sobre o sistema, além disso DEV e QA tem que estarem alinhados para conseguirem fazer um bom trabalho”.
QA4 = “Sim”.
QA5 = “Sim, por ter um entendimento simples, o compartilhamento de conhecimento fica mais fácil”.
QA6 = “Sim, tendo em vista as discussões e trocas de ponto de vista e conhecimento acerca do que está sendo desenvolvido”.

8. Quais as vantagens e desvantagens que você vê com o uso do BDD?

QA1 = “O bdd torna as especificações de tarefas mais detalhadas e padronizadas, fazendo com que o entendimento da tarefa fique mais claro. Por outro lado, isto torna o processo de criação de tarefas mais trabalhoso, já que são exigidos muitos detalhes, como comportamento esperado para vários casos de uso, por exemplo”.
QA2 = “Vantagem: foco na entrega de valor da funcionalidade que a empresa quer alcançar e não foco no MVPI (entrega individual), maximizando a entrega de valor e minimiza a introdução de recurso desnecessário para o cliente, podendo lançar de forma mais rápida quando o objetivo é diminuir gargalos e reduzir a carga de teste manuais. Desvantagem: mensurar a funcionalidade e não necessariamente a qualidade da entrega, a não ser que tenha um desenvolvimento orientado a teste. (BDD/ TDD) Nem todas as ferramentas suportam esse tipo de trabalho, e todo o time deve estar comprometido com a prática e os pilares do BDD. (BDD não é apenas sobre teste)”.

QA3 = “Vantagens: alinhamento do time, padronização da escrita, facilidade do entendimento da funcionalidade. Desvantagem: tomar cuidado para que a escrita fique muito grande e isso se torne um gargalo no fluxo”.
QA4 = “Vantagens: Colaboração, conhecimento e comunicação. Não vejo nenhuma desvantagem”.
QA5 = “Vantagens: facilita o entendimento das tarefas, fácil de aplicar, padronização da estrutura das tarefas”.
QA6 = “o BDD é uma técnica ágil que auxilia muito no entendimento e na especificação de desenvolvimento de um produto, colabora para o alinhamento do time e auxilia na montagem de cenário de testes automáticos. Acredito que o BDD pode não auxiliar quando a sua aplicação não é eficaz e quando não existe um alinhamento e aceitação da técnica por meio do time comum todo. Talvez seja válido, junto com o BDD, implementar outros mecanismos que auxiliem no processo de desenvolvimento e ajudem a traçar planos e estratégias de como uma *feature* vai ser desenvolvida com qualidade e num tempo hábil”.

9. Relacionado ao esforço do desenvolvimento, você acredita que houve menos retrabalho com o uso do BDD? Sim / Não. Por quê?

QA1 = “Não posso responder pois não estou em um time que utiliza o bdd”.
QA2 = “Sim, pois evita retrabalho com constantes melhorias durante a sprint, uma vez que é de fato identificado o que deve ser testado e desenvolvido”.
QA3 = “Sim, evita retrabalho”.
QA4 = “Sim, o desenvolvedor já saber o que é pra ser feito e não vai gerar dúvida quando estiver sendo desenvolvido”.
QA5 = “Sim, porque com todo mundo entendendo o objetivo da tarefa melhor, menos pessoas cometem erros/retrabalho”.
QA6 = “Em alguns momentos sim, em outros não, justamente porque o BDD não é suficiente se não for implementado junto com outras ferramentas e outros mecanismos de facilitar o planejamento de desenvolvimento”.

10. De uma maneira geral, após a introdução e a aplicação do BDD no projeto, você acredita que foi possível transformar os requisitos em um produto que agrega valor ao cliente?

QA1 = “Não participo de um time que usa o bdd, mas acredito que sim, pois com especificações mais claras, é mais fácil garantir que requisitos estejam sendo

cumpridos. Garantindo que estes requisitos são cumpridos, é mais fácil garantir a qualidade do produto final”.
QA2 = “Sim, uma vez que as funcionalidades são quebradas em Épicos, que permite a visão do todo de forma macro do que é esperado no final da entrega do MVP, e em US que são as pequenas entregas dentro de valor dentro do todo, cada US é uma entrega de valor para o cliente. A maior dificuldade no início é entender o que é um MVP e que as US devem ser entregas independentes”.
QA3 = “Sim, além disso é possível usar o bdd como requisitos dos testes automáticos”.
QA4 = “Acaba agregando muita qualidade na entrega e entender a necessidade do cliente gerando maior valor na entrega”.
QA5 = “Sim”.
QA6 = “Sim”.