CEVERINO, Aparecida. NASCIMENTO, Fernando Paes. **Utilização da técnica de desenvolvimento orientado por comportamento (bdd) no levantamento de requisitos**. Revista Interdisciplinar Científica Aplicada, Blumenau, v.10, n.3, p.40-51, TRIII 2016. ISSN 1980-7031.

UTILIZAÇÃO DA TÉCNICA DE DESENVOLVIMENTO ORIENTADO POR COMPORTAMENTO (BDD) NO LEVANTAMENTO DE REQUISITOS

Aparecida Cezerino[1], Fernando Paes Nascimento[2]

RESUMO:
Empresas e profissionais têm buscado cada vez mais técnicas e métodos de desenvolvimento que os auxiliem em uma maior efetividade no levantamento de requisitos, na comunicação e envolvimento do cliente e na qualidade da entrega do produto de software. Uma das principais causas de software com baixa qualidade ou que não atende as necessidades do cliente, se deve a falta de efetividade no processo de levantamento do requisito. A técnica BDD (do inglês, Behavior Driven Development, ou em português, "desenvolvimento orientado a comportamento") é baseada em usuários, buscando agregar valor de negócio, através de um vocabulário comum, possibilitando comum entendimento entre TI(Tecnologia da Informação) e Negócio. Cria-se comportamentos baseados no usuário, facilitando a compreensão de todos os envolvidos. Diante disso, este artigo é motivado a demonstrar um formato de levantamento de requisitos alternativo aos métodos tradicionais.

Palavras-chave: Behavior Driven Development. Levantamento de Requisitos. Linguagem Ubíqua.

ABSTRACT:
Businesses and professionals have increasingly sought technical and development of methods to assist them in greater effectiveness in requirements gathering, communication and customer engagement and quality of delivery of the software product. One of the main causes of software with low quality or that does not meet customer needs, is due to lack of effectiveness in requirement of the survey process. The BDD technique (English, Behavior Driven Development, or Portuguese, "desenvolvimento orientado a comportamento") is based on users, seeking to add business value through a common vocabulary, enabling common understanding between IT (Information Technology) and Business. Is created based on user behavior, facilitating the understanding of everyone involved. Therefore, this article is motivated to show a survey of alternative format requirements to traditional methods.

Keywords: Behavior Driven Development. Requirements gathering. Ubiquitous language.

1 INTRODUÇÃO

De acordo com Pressman (2006, p. 117), construir um software é algo tão desafiador, criativo, gostoso e compulsivo, que leva muitos desenvolvedores a ignorar as fases iniciais de análise e começarem logo o desenvolvimento, alegando que ele se tornará mais claro à medida que for sendo desenvolvido. Porém, Mello (2010), expõe que o levantamento de requisitos é umas das partes mais importantes do processo que resultará no desenvolvimento de um sistema.

[1] Pós-graduando em Engenharia de Software pelo Instituto Superior Tupy – SOCIESC. E-mail: cida.cezerino@gmail.com.
[2] Professor nos cursos de graduação e pós-graduação do Instituto Superior Tupy - SOCIESC. E-mail: executivo.nbusiness@gmail.com

Segundo Leffingwell, (1997), os requisitos estão associados aos principais problemas do desenvolvimento de software e 40 a 60% dos problemas encontrados nos projetos são consequências de falhas no levantamento de requisitos. Apenas 39% dos projetos de software são concluídos no prazo e no orçamento, e mesmo quando é concluído apresenta, em média, apenas 42% das funções e características propostas originariamente [Standish CHAOS Report 2013].

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. Seguindo esse desejo, um grupo de 17 desenvolvedores se reuniu para discutir formas de melhorar o desempenho de seus projetos. Embora cada envolvido tivesse suas próprias práticas e teorias sobre como fazer um projeto de software ter sucesso, cada qual com as suas particularidades, todos concordavam que, em suas experiências prévias, um pequeno conjunto de princípios sempre parecia ter sido respeitado quando os projetos davam certo. (Lacerda, 2012).

Esse encontro deu origem ao manifesto ágil, publicado em 2001. Os métodos ágeis baseiam-se no desenvolvimento incremental de software e são mais adequados para casos em que os requisitos mudam frequentemente durante o desenvolvimento. Estes métodos visam diminuir burocracias no processo, evitando trabalho que tem valor a longo prazo apenas e eliminando documentações que provavelmente nunca serão utilizadas (Sampaio, 2014).

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 continua (Ambler, 2010).

A automatização de testes permite 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).

Seguindo esse conceito o Extreme Programming(XP), propõe o Test driven development (Desenvolvimento Orientado a Teste- TDD). Conforme Aniche (2014), o TDD é uma técnica de desenvolvimento de software que segue a seguinte mecânica: escrever um teste que falha, fazê-lo passar da maneira mais simples possível e, por fim, refatorar o código. Ao praticar TDD, o desenvolvedor antes de começar a realizar a codificação de uma funcionalidade, explicita os objetivos da funcionalidade. Isto é realizado por meio de testes automatizados. O teste nada mais é do que um trecho de código que deixa claro o que outro determinado trecho de código deve fazer.

North (2012) afirma que o BDD (Behavior-Driven Development) é o mesmo que TDD, porém voltado para uma audiência maior: testadores, analistas, gerentes de projeto. O BDD se aplicado em equipes formadas exclusivamente por programadores será exatamente igual ao TDD,

porém se isto não é verdade, então o BDD se torna uma coisa diferente, maior: tona-se a comunicação entre todas essas partes interessadas para criar uma única visão coerente e para a qual as entregas são voltadas.

O presente artigo tem como objetivo principal estudar e avaliar os benefícios da aplicação da técnica BDD no levantamento de requisitos.

2 LEVANTAMENTO DE REQUISITOS COM USER STORIES (HISTÓRIAS DE USUÁRIO)

Conforme Maiden e Alexander (2004), as User Stories são a prática mais popular de captura de requisitos nas metodologias ágeis. Elas são pequenos cenários do sistema que ajudam os stakeholders a partilhar comportamentos esperados do sistema a desenvolver. Visam obter qual o requisito a implementar, o porquê da sua existência e qual o business value (valor de negócio) associado à sua implementação. São escritas com uma linguagem próxima do entendimento dos stakeholders (parte interessada), visando uma comunicação eficaz com a equipe de desenvolvimento. Servem como guia para a equipe de desenvolvimento, guiando futuras conversas e gerando discussões.

Cada User Story, ao ser criada, tem de ter presente quatro propriedades: (1) tem de ser descritiva, i.e., ser possível extrair uma expectativa do stakeholder ou uma funcionalidade do sistema; (2) tem de ser possível estimar o esforço necessário para a sua realização; (3) cada User Story tem de ter a capacidade de ser testada com o intuito de verificar se está de acordo com as expectativas comportamentais dos stakeholders e por último, (4) cada User Story é alvo de priorização por parte dos stakeholders para que a equipe de desenvolvimento consiga focar nas User Stories prioritárias antecipadamente.

Conforme Sampaio (2014), não existe uma formalização para a documentação de histórias e é comum que seu registro seja descartado depois de sua implementação, porém, a prática de BDD tem alcançado grande popularidade entre os times ágeis, e realizando os testes de aceitação de maneira automatizada as histórias acabam sendo documentadas de uma forma em que seu registro é mantido.

3 BDD -DESENVOLVIMENTO ORIENTADO POR COMPORTAMENTO (BEHAVIOUR DRIVEN DEVELOPMENT)

Dan North (2006) idealizou o BDD enquanto ministrava cursos de TDD para alunos desenvolvedores. Ao ministrar estes cursos ele percebeu, no entanto, que havia uma série de

dúvidas frequentes de seus alunos. Como desenvolvedores, eles queriam saber por onde começar, o que testar, o que não testar, e como entender porque seus testes deveriam falhar num primeiro momento. Ao buscar estas respostas, foi que Dan North deu origem ao BDD. Ele começou a utilizar a palavra “comportamento” no lugar de “teste” e verificou que essa palavra se encaixava perfeitamente com o cenário. Com a ajuda de seu colega Chris Matts, os dois reformularam algumas boas práticas já existentes, até ascender ao nível de uma nova técnica, que por fim respondia as frequentes dúvidas dos alunos de TDD.

Para estabelecer o teor da técnica BDD, Chris Matts e Dan North escreveram:

"Se conseguíssemos desenvolver um vocabulário consistente para analistas, testadores, desenvolvedores e para o negócio, então nós estaríamos no caminho certo para eliminar um pouco da ambiguidade e dos problemas de comunicação que ocorrem quando pessoas técnicas falam com pessoas de negócio."

Diante disso, o foco do BDD é estimular a colaboração entre todos os participantes do projeto (desenvolvedores, pessoas da qualidade, pessoas de negócio e cliente), estabelecendo uma linguagem única entre os envolvidos. Tanto cliente, como analista de negócio, quanto desenvolvimento e testes, devem falar sobre o sistema de uma mesma forma (North, 2016).

3.1 Os três princípios do BDD

Conforme Caroli (2014), os três princípios do BDD são:

1. **Nada além do suficiente:** pensar e planejar o design da aplicação em longo prazo não é benéfico no que se refere a valor para o negócio. Não se deve fazer menos do que o necessário para começar, mas qualquer coisa além disso é desgaste desnecessário.
2. **Entregar valor aos stakeholders:** se você estiver trabalhando em algo que não entrega valor e nem auxilia na habilidade de entregar valor, pare de fazê-lo agora mesmo.
3. **Tudo se baseia em comportamento:** tanto em nível de código como de especificações da aplicação, pode-se usar o mesmo pensamento e a mesma linguística para descrever comportamento, independente do nível de granularidade.

4 BDD TRAZ UMA LINGUAGEM ÚNICA PARA ANÁLISE

No ano de 2003, Eric Evans publicou seu livro mais vendido Domain-Driven Design. No livro, ele descreve o conceito de modelar um sistema utilizando uma linguagem única baseada no modelo de negócios, desta forma o vocabulário de negócio permeia a base de código do sistema.

Essa linguagem é utilizada para representar modelos e documentações, de forma que estes sejam compreendidos pelo cliente, analista, projetista, desenhista, testador, gerente, etc.

A linguagem ubíqua está presente em todas as partes do desenvolvimento e é utilizada por todos os envolvidos com o objetivo de eliminar a necessidade de traduções. As traduções fazem

com que termos e detalhes sejam perdidos, utilizando a mesma linguagem isso não acontece, porém, para a utilização da mesma linguagem por todos, muitas vezes é necessário um aprendizado (Prikladnick et al., 2014).

4.1 Gherkin

A adoção efetiva do BDD implica na utilização de um vocabulário consistente por parte do "negócio". De acordo com a Comunidade Open Source de Cucumber no GitHub (2013), o Gherkin é uma Linguagem de Domínio Específico que permite a descrição do comportamento do software sem o detalhamento de como este comportamento deve ser implementado. Esta linguagem serve para dois propósitos: documentação e automação de testes.

Conforme Vieira(2014) um arquivo Gherkin é semelhante a um Caso de Uso, e contém:

- Título da funcionalidade.
- Descrição da funcionalidade.
- Cenários, compostos por uma sequencia de passos, que descrevem uma interação entre um usuário e o sistema.

Estes arquivos usam a extensão *.feature* e são salvos em texto plano, isto significa que eles podem ser lidos e editados por ferramentas simples. A estrutura de um arquivo Gherkin é estabelecida pelas seguintes palavras-chave (com seus equivalentes em português):

- *Feature*, Funcionalidade.
- *Background*, Contexto.
- *Scenario*, Cenário.
- *Given*, Dado.
- *When*, Quando.
- *Then*, Então.
- *And*, E.
- *But*, Mas.
- Comentários.
- *Background*.
- *Scenario Outline*, Esquema do Cenário.
- *Examples*, Exemplos.

Para outro idioma basta utilizar as palavras equivalentes na linguagem desejada.

A seguir serão discutidos os principais elementos utilizados no Gherkin de acordo com Wynne e Hellesoy (2012).

> **Feature**: cada arquivo *Gherkin* começa com a palavra chave *Feature*. O texto imediatamente após a palavra *feature*, na mesma linha, é o nome da funcionalidade e as linhas subsequentes são a descrição da mesma. É possível utilizar qualquer palavra na descrição desde que não se use no início de nenhuma linha as palavras *Scenario*, *Background* ou *Scenario Outline*. A descrição pode ter múltiplas linhas. É um ótimo lugar para colocar detalhes sobre quem vai utilizar a funcionalidade, e por qual motivo. Em

Gherkin, uma *feature* deve ser seguida por um dos seguintes termos: *Scenario*, *Background* ou *Scenario Outline*

Scenario: Para expressar realmente o comportamento desejado, cada *feature* contém vários cenários. Cada cenário é um exemplo concreto de como o sistema deve comportar-se em determinada situação. Somando o comportamento definido por todos os cenários temos o comportamento esperado da *feature*. Ao escrever um cenário devemos ter em mente que cada cenário deve ter sentido e permite ser executado de forma independente de outros cenários. Todo cenário segue o seguinte padrão: obter o sistema a um estado particular; estimular o sistema; examinar o novo estado. No Gherkin são utilizadas as seguintes palavras chave para identificar esse comportamento dos cenários: *Given, Then, When, And,* e *But*.

Comentários: além das descrições de cada *feature*, dos cenários e dos passos, também podem ser colocados comentários em qualquer parte do documento, desde que iniciem com "#" e sejam a única coisa presente na linha onde foram colocados. As descrições devem ser utilizadas para colocar informações diretamente relevantes para os clientes e os comentários para informações
técnicas direcionadas ao time de desenvolvimento. É possível colocar os detalhes técnicos nas descrições, porém, para tal, deve ser certificado de que o cliente se sente confortável com essas informações.

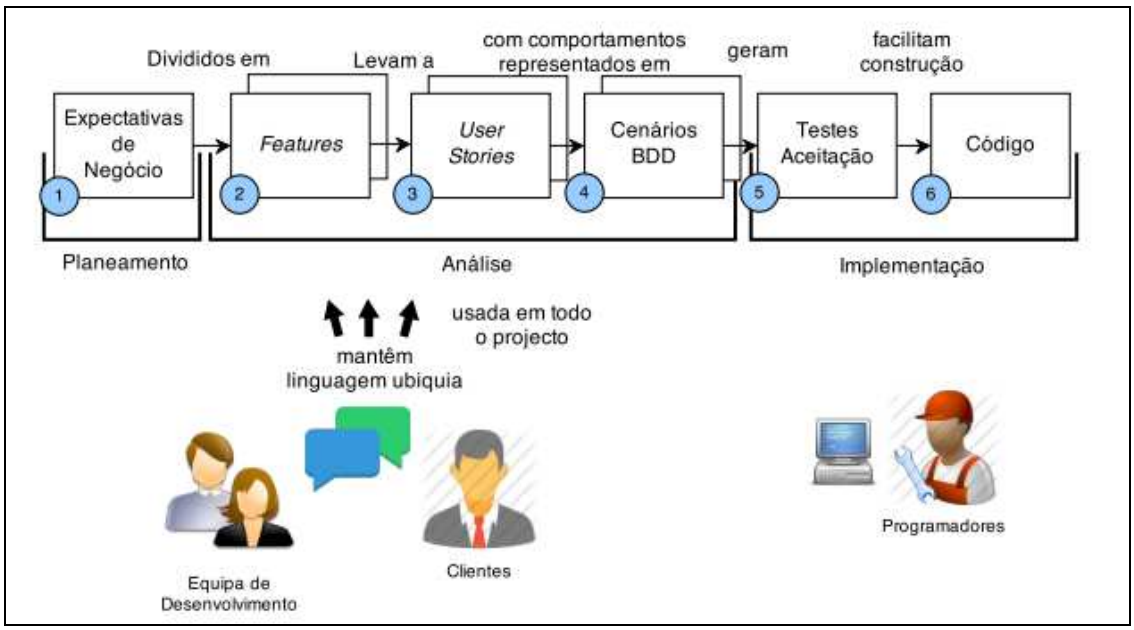
Background: é uma sessão da *feature* destinada a colocar informações necessárias a todos os cenários. É utilizada para evitar a repetição de informações nos cenários.

Scenario Outline: é utilizado para quando existem cenários que têm a mesma estrutura e o mesmo propósito, porém possuem entradas e saídas diferentes. Dessa forma é criado um cenário genérico e colocado entre os símbolos "< >" palavras que identifiquem as entradas e saídas e em seguida é colocada a seção "*Examples*" onde são colocadas em formato de tabela as entradas e saídas que devem ser testadas e as colunas são identificadas de acordo com as palavras entre "< >" apresentadas no *Scenario Outline*.

5 PROCESSO DO DESENVOLVIMENTO ORIENTADO POR COMPORTAMENTO-BDD

De acordo com SILVA(2014) o processo do BDD pode ser composto por seis passos, como mostra a Figura 1. É um processo para ser inserido numa metodologia ágil, logo, é iterativo. Após o sexto passo, são validados os artefatos produzidos com os *stakeholders* e o ciclo é reiniciado onde os artefatos anteriormente produzidos podem sofrer alterações.

Figura 1- O Processo do BDD



Fonte: Silva (2014)

A figura 1 demonstra que a prática do BDD inicia-se com a comunicação entre a equipe de desenvolvimento e os *stakeholders*. Dessa comunicação, na fase de planejamento, são definidas expectativas de negócio, que é a primeira caixa do processo. Os comportamentos são derivados dos objetivos de negócio que se pretende produzir. Na fase de análise, os objetivos de negócio são decompostos em funcionalidades (*features*) e cada uma das funcionalidades está associada a uma ou mais *User Stories*, a segunda e terceira caixa do processo respectivamente. Cada *User Story* é instanciada com múltiplos cenários BDD, onde em cada um é delineado o comportamento esperado para aquela situação. O cenário é construído com recurso à linguagem do projeto e é escrito pelos *stakeholders* ou em conjunto com a equipe de desenvolvimento. Os cenários BDD têm um *template*, também escrito por uma linguagem ubíqua independente de domínio, de forma a ser utilizado em qualquer situação. A Figura 2 mostra um *template* de cenário BDD (Silva, 2014).

Figura 2- *Template* de Cenário BDD

Título: (Título para descrever *story*)
Como um (papel)
Eu quero (funcionalidade)
Para que (benefício)

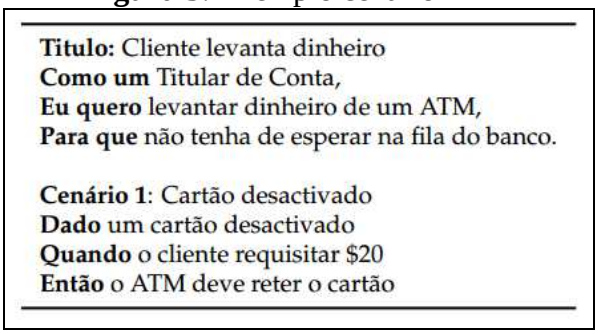
Cenário 1 (título)
Dado (contexto)
E (outro contexto)
Quando (evento)
Então (resultado)

Fonte: North (2006)

Um cenário BDD é composto por uma parte inicial que representa uma *User Story*. A segunda parte do cenário descreve o comportamento em si. **Dado** um contexto inicial, descrito por

várias características, **quando**, eventualmente, ocorrer uma mudança no domínio, **então** obtém-se o estado final pretendido. O objetivo de um cenário BDD é descrever como o sistema implementa uma funcionalidade e como se deve comportar dado um contexto e a ocorrência de um certo evento. A figura 3 mostra um exemplo de cenário BDD (Silva, 2014).

Figura 3: Exemplo cenário BDD



Fonte: North (2006)

Após a construção de cenários comportamentais, os mesmos serão transformados em testes de aceitação. A transformação é feita através de regras de mapeamento, para que o código seja escrito com a linguagem do cenário. O nome dos métodos e classes devem respeitar os nomes escritos no cenário BDD. Assim, a manutenção do sistema fica facilitada visto que tem nomes explícitos e associados ao seu comportamento esperado. Com a construção automática de testes de aceitação, a fase de implementação tem um início mais robusto, visto que os programadores já têm como base cenários de comportamento validados por parte dos clientes (Silva, 2014).

6 COMPARATIVO DO LEVANTAMENTO DE REQUISITOS USANDO MÉTODOS TRADICIONAIS VERSUS BDD

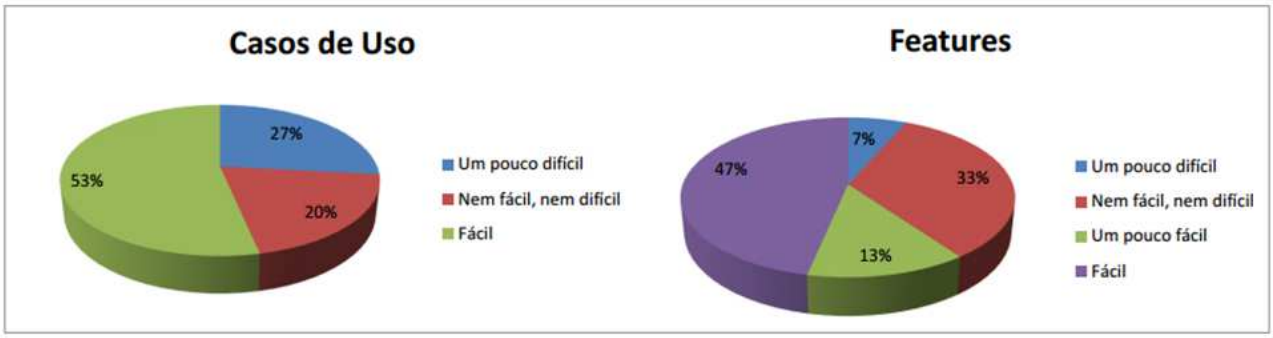
Sampaio (2014) desenvolveu um trabalho no qual teve como objetivo avaliar de forma comparativa o detalhamento de requisitos através do uso de casos de uso e de documentação em linguagem ubíqua com Gherkin.

Sampaio, realizou uma pesquisa que avaliou a compreensão do requisito, a facilidade de entendimento da documentação, a preferência pela forma de documentar e a capacidade de identificação de defeitos.

Na comparação sobre a facilidade de entendimento da documentação, 27% dos participantes afirmou achar os casos de uso um pouco difíceis, ao passo que apenas 7% achou as features um pouco difíceis. Dessa maneira, mesmo sendo o primeiro contato de todos os participantes com a documentação em formato de features, poucos a apontaram dificuldades na compreensão. A Figura

4 e a Figura 5 mostram a distribuição percentual da opinião dos participantes sobre a facilidade de entendimento de cada documentação.

Figura 4 - Facilidade de Entendimento dos *Features* e Casos de Uso



Fonte: Sampaio (2014)

Através desta pesquisa Sampaio pode concluir ser viável a utilização de *features* para o detalhamento de requisitos, porém, sua pesquisa possui algumas limitações que devem ser analisadas a fim de realizar trabalhos futuros para dar continuidade ao estudo. Abaixo é listado algumas das limitações de seu trabalho:

- O número de participantes foi pequeno, apenas 15 participaram até o final da pesquisa;
- Nenhum dos participantes estava familiarizado com o estilo de documentação apresentado pelas *features*, nem com a escrita em Gherkin;
- Não foi possível avaliar a identificação de defeitos de forma relevante para afirmar qual das documentações seria mais adequada a este propósito;
- Não foi considerado se os participantes trabalham ou já trabalharam com metodologias ágeis;

Recentemente Reali (2015), também desenvolveu um trabalho que 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. 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.

Conforme tabela 1 pode-se verificar qual processo necessita de um menor esforço para o desenvolvimento de uma funcionalidade.

Tabela 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: Reali (2015)

Conforme Reali (2015), estes resultados nos permitem concluir que, neste estudo de caso, o BDD implicou um menor esforço médio para o desenvolvimento de uma funcionalidade. Cabe destacar que BDD inclui neste esforço a total automatização dos teses, 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.

A tabela 2 mostra as funcionalidades em que foram encontrados os problemas.

Tabela 2- Funcionalidades Entregues que Apresentam 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: Reali (2015)

Reali (2015), conclui que os resultados obtidos indicam que o processo de desenvolvimento segundo BDD, se comparado aquele 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.

7 CONCLUSÃO

Conclui-se este artigo com uma frase de Edward V. Berard: "Caminhar sobre a água e desenvolver software a partir de uma especificação de requisitos é fácil se ambos estão congelados".

Assim, sendo a única certeza que se tem em desenvolvimento de software é que requisitos mudam e que o cliente raramente sabe expressar o que deseja para um software. Por este motivo a técnica de elicitação de requisitos é tão importante, pois conforme evidenciado, um resultado assertivo deste processo é fundamental para o desenvolvimento de um sistema. Uma necessidade mal compreendida acarretará em retrabalho futuro, e fere a satisfação do cliente com o produto.

O BDD é um processo que se baseia essencialmente em três princípios: descrever o comportamento, descobrir o comportamento que acrescenta valor ao negócio e utilizar histórias de usuário e cenários para se referir ao que é esperado do sistema. Com o conteúdo relatado neste artigo, quis destacar na técnica BDD, a vital importância da formalização na comunicação com o cliente e equipe, removendo ambiguidades, através de uma linguagem compreensiva, muito próxima da linguagem natural.

Embora seja uma técnica de desenvolvimento nova, BDD se mostra eficiente em comparação a outras técnicas tradicionais já conhecidas.

REFERÊNCIAS

ANICHE, M. “**Test-Driven Development**”. Disponível em: <http://tdd.caelum.com.br/>. Acessado em: 02 abr. 2016.

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

CAROLI, P. **Antologia Brasil: Histórias de Aprendizado e Inovação.** São Paulo: Casa do Código, 2014.

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

GitHub (2013) **“Gherkin”,** Disponível em: <https://github.com/cucumber/cucumber/wiki/Gherkin> Acessado em 10 de Abr. 2016.

LACERDA, Henrique. **Metodologia Ágil,** Disponível em: < http://www.henriquelacerda.com.br/tag/xp/>. Acessado em: 10 de Abr. 2016.

LEFFINGWELL, D. **Calculating Your Return on Investment from More Effective Requirements Management,** Disponível em<ftp://public.dhe.ibm.com/software/rational/web/whitepapers/2003/roi1.pdf>. Acesso em: 02 abr. 2016.

CEVERINO, Aparecida. NASCIMENTO, Fernando Paes. **Utilização da técnica de desenvolvimento orientado por comportamento (bdd) no levantamento de requisitos**. Revista Interdisciplinar Científica Aplicada, Blumenau, v.10, n.3, p.40-51, TRIII 2016. ISSN 1980-7031.

MELLO, L.C.S. **Levantamento de Requisitos**, Mato Grosso, nov. 2010. Disponível em: <http://www.ice.edu.br/TNX/encontrocomputacao/artigos-internos/aluno_leandro_cicero_levantamento_de_requisitos.pdf>. Acesso em: 02 abr. 2016.

MAIDEN, Neil Editeur. ALEXANDER, Ian Editeur. **Scenarios, stories, use cases : through the systems development life-cycle.** Disponível em: < http://opac.inria.fr/record=b1105823.>. Acesso em: 10 de Abr. 2016.

NORTH, D. (2006) **Introducing BDD.** Disponível em: <http://dannorth.net/introducing-bdd/>. Acesso em: 02 Abr. 2016.

NORTH, D. (2012) **BDD is like TDD if....** Disponível em: <http://dannorth.net/2012/05/31/bdd-is-like-tdd-if/>. Acesso em: 02 Abr. 2016.

PRESSMAN, Roger S. **Engenharia de software. 6. ed.** São Paulo: McGraw-Hill, 2006.

PRIKLADNICKI, R., Willi, R., Milani, F. (2014), **Métodos Ágeis para Desenvolvimento de Software**, Bookman, 1ª edição.

REALI, Diogo G. **Análise e Experiência no Desenvolvimento Guiado ao Comportamento.** Disponível em: <http://www.lume.ufrgs.br/bitstream/handle/10183/126080/000972339.pdf?sequence=1> Acessado em: 10 de Abr. 2016.

SAMPAIO, Rafaela da Fonseca. **Avaliação do Uso de Linguagem Ubíqua no Detalhamento de Requisitos Utilizando Gherkin.** 2014. 89 f. Trabalho de Conclusão de Curso (Sistemas de Informação)- Universidade Federal do Estado do Rio de Janeiro (UNIRIO), Rio de Janeiro, 2014.
SILVA, António Pedro Ferreira. **Uma Abordagem Ágil para Transformar Modelos Cognitivos em Modelos Comportamentais e de Domínio.** 2014**.** 219 f. Dissertação (Mestrado em Engenharia Informática)- Universidade Nova de Lisboa, Portugal, 2014.

SILVA, Marcio Andrade. **A importância do levantamento de requisitos no sucesso dos projetos de software**, Disponível em: <http://www.linhadecodigo.com.br/artigo/1685/a-importancia-do-levantamento-de-requisitos-no-sucesso-dos-projetos-de-software.aspx>. Acesso em: 10 Abr. 2016.

THE STANDISH GROUP. **Chaos Manifesto 2013**, Disponível em: <https://www.versionone.com/assets/img/files/CHAOSManifesto2013.pdf>. Acesso em: 10 Abr.. 2016.

VIEIRA, Daniela. **BDD com Cucumber – Parte 1**, Disponível em: <http://agiletesters.com.br/topic/20/bdd-com-cucumber-parte-1> Acesso em: 10 de Abr. 2016.

WYNNE, M., HELLESOY, A. (2012), **The Cucumber Book**, LLC, 1st edition.