UNIVERSIDADE DO VALE DO RIO DOS SINOS – UNISINOS

UNIDADE ACADÊMICA DE EDUCAÇÃO CONTINUADA

ESPECIALIZAÇÃO EM QUALIDADE DE SOFTWARE

Aniela Cole

AVALIAÇÃO DE ADERÊNCIA DE PRÁTICAS DO AGILE TESTING EM TIMES QUE UTILIZAM SCRUM

São Leopoldo

2015

UNIVERSIDADE DO VALE DO RIO DOS SINOS – UNISINOS

UNIDADE ACADÊMICA DE EDUCAÇÃO CONTINUADA

ESPECIALIZAÇÃO EM QUALIDADE DE SOFTWARE

Aniela Cole

AVALIAÇÃO DE ADERÊNCIA DE PRÁTICAS DO AGILE TESTING EM TIMES QUE UTILIZAM SCRUM

Trabalho de Conclusão de Curso apresentado como requisito parcial para a obtenção do título de Especialista em Qualidade de Software, pelo curso de Pós-Graduação Lato Sensu em Qualidade de Software da Universidade do Vale do Rio dos Sinos – UNISINOS.
Orientador: Prof. Esp. Priscila Coelho Blauth

São Leopoldo

2015

AGRADECIMENTOS

Um agradecimento especial àquele que está sempre ao meu lado, por seu apoio, por sua compreensão e incentivo.
Aos meus pais, familiares e amigos que compreenderam a minha ausência.
A professora Priscila Blauth, por sua atenção, orientações e ensinamentos.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

CI – *Continuous Integration* (Integração Contínua)

PO – *Product Owner* (Dono do Produto)

SM – *Scrum Master*

TDD – *Test Driven Development* (Desenvolvimento Dirigido a Testes)

SUMÁRIO

Avaliação de Aderência de Práticas do *Agile Testing* em times que Utilizam *Scrum*

Aniela Cole[1]

[1]Universidade do Vale do Rio do Sinos (Unisinos). Qualidade de Software. São Leopoldo – RS – Brasil

aniela.ads@gmail.com

Abstract. *Based on studies of agility, Scrum and Agile Testing have been identified specific aspects of testing in agile environments, including types and test techniques that should be performed by an agile team, as well as its form of more efficient execution. Based on this survey, it has been developed a case study on a company's information technology sector, with the objective of mapping the testing types and techniques which are used by this company. From the analysis it was possible to identify which types and testing techniques are being performed as suggested by the practice Agile Testing and what can be improved. This article aims to evaluate the adherence of a company team in question in relation to the practice Agile Testing.*

Resumo. *Baseando-se em estudos sobre Agilidade, Scrum e Agile Testing, foram identificados aspectos específicos de testes em ambientes ágeis, dentre eles, tipos e técnicas de testes que devem ser realizadas por um time ágil, bem como sua forma de execução mais eficiente. Com base neste levantamento, elaborou-se um estudo de caso em uma empresa do setor de tecnologia da informação, a fim de mapear os tipos e técnicas de testes que são utilizados e sua forma de execução. A partir da análise realizada foi possível identificar quais tipos e técnicas de testes estão sendo executadas da forma sugerida pela prática Agile Testing e quais podem ser melhoradas. Este artigo tem como objetivo avaliar a aderência de um time da empresa em questão em relação à prática Agile Testing.*

Palavras chave: *Agilidade, Scrum, Integração Contínua, Testes de Software, Agile Testing.*

1. Introdução

Nos últimos anos, o software se tornou um componente vital nos negócios. O sucesso de uma organização cada vez mais depende da utilização do software como um diferencial competitivo. Nesse sentido, o principal desafio na área de engenharia de software nas últimas duas décadas tem sido o estudo sobre a melhoria da qualidade e a redução de custo do software produzido [Pressman 2011].

À medida que os sistemas de informação se tornam mais estratégicos, as técnicas de desenvolvimento de software se adaptaram para que fosse possível responder rapidamente às mudanças do mercado. Desta forma, existe uma diversidade de

metodologias e descrições de processos, métodos e notações de modelagem, ferramentas e tecnologias obsoletas. Cada uma atingiu o seu ápice e foi ofuscada por novas metodologias, que mantinham princípios melhores [Pressman 2014].

Neste cenário, surgiram os métodos ágeis. No contexto de engenharia de software, agilidade tornou-se a forma de se descrever um moderno processo de software. Uma equipe ágil é aquela rápida e capaz de responder apropriadamente a mudança e, mudanças têm muito a ver com desenvolvimento de software, sejam no software que está sendo criado, nos membros da equipe, sejam em virtude das novas tecnologias, ou seja, mudanças de todos os tipos poderão ter um impacto no produto que está sendo desenvolvido. Uma equipe ágil reconhece que o software é desenvolvido por indivíduos, com trabalho em equipes, e que as habilidades dessas pessoas e suas capacidades em colaborar estão no cerne do sucesso do projeto [Jacobson 2002].

Dentre os métodos ágeis mais utilizados, o *Scrum* tem destaque. Seus princípios são consistentes com o manifesto ágil e são usados para orientar as atividades de desenvolvimento dentro de um framework. O *Scrum* enfatiza o uso de um conjunto de técnicas que provaram ser eficientes para projetos complexos e requisitos mutáveis [Schwaber e Sutherland 2013].

Para que seja possível, responder a mudanças é necessário ter um ambiente de desenvolvimento que favoreça o *feedback* contínuo, ou seja, que o impacto de cada alteração seja percebido instantaneamente por toda equipe. Para tanto, é recomendado que este ambiente possuísse integração contínua, onde serão executados todos os testes desenvolvidos do projeto. Os resultados dos testes serviriam como uma forma de *feedback* instantâneo, no qual o desenvolvedor pode detectar em pouco tempo se a funcionalidade desenvolvida ainda precisa ser corrigida ou se gerou não conformidades em outras funcionalidades [Duval 2007].

Para que o impacto de cada alteração possa ser identificado rapidamente pelo time de desenvolvimento, foi criado o conceito de *Agile Testing*, que se trata de um conjunto de práticas de testes, já conhecidas pelo mercado, aplicadas de forma que possam se complementar ao máximo. Tais práticas podem ser utilizadas por profissionais de teste e desenvolvedores, e visam atender ao manifesto ágil aplicando técnicas de testes de software que possibilitam a redução no tempo de execução de testes, além de aumentar a cobertura de cenários e possibilidades [Crispin e Gregory 2009].

1.1. Objetivos

Este trabalho tem como objetivo geral realizar uma avaliação de aderência de práticas do *Agile Testing*, em um time de desenvolvimento que utiliza *Scrum*, em uma empresa do setor de tecnologia da informação (TI). Para que seja possível atingir o objetivo geral foram definidos os seguintes objetivos específicos:

- Aprofundar conhecimentos em Metodologias Ágeis com foco no *Scrum*, que é a Metodologia Ágil utilizada pela empresa em que será realizado o estudo de caso. O objetivo será entender seus eventos, artefatos e cerimonias, para que seja possível mapear o processo de desenvolvimento.
- Aprofundar conhecimentos sobre *Agile Testing*, identificando técnicas e tipos de testes que podem ser aplicadas em projetos ágeis de acordo com os Quadrantes do *Agile Testing*;

- Realizar uma avaliação da aderência de práticas do *Agile Testing* em um time de desenvolvimento.
- Realizar uma análise crítica das informações levantadas.

1.2. Justificativa

Segundo [Crispin e Gregory 2009], o Desenvolvimento Ágil não é a única maneira de entregar software com sucesso, no entanto todas as equipes que entregaram software com sucesso tiveram várias semelhanças. Nestas equipes, os programadores escreviam testes unitários e de integração, garantindo um percentual considerável de cobertura. As equipes de desenvolvimento estavam disciplinadas no uso de controle de código fonte e integração de código. Analistas de Testes estavam envolvidos desde o início do ciclo de desenvolvimento e, possuíam tempo e recursos para fazer um trabalho adequado automatizando todas as formas necessárias de testes.

O desenvolvimento ágil possui valores, princípios e práticas fundamentais que permitem que as pessoas façam o seu melhor trabalho, mas é preciso uma serie de fatores para que os testes sejam realizados da forma correta. Muitas vezes não há equilíbrio entre os testes que são realizados, por exemplo: um time pode realizar somente teste manual, e não implementar nenhum tipo de teste automatizado [Crispin e Gregory 2009]. Neste caso, apenas utilizar uma metodologia ágil não garantirá a qualidade do projeto. Utilizar a prática *Agile Testing* complementará as características das metodologias ágeis utilizadas por um time ou uma empresa, a fim de garantir que os testes sejam feitos na proporção e momento corretos.

1.3. Delimitação do tema

O estudo de caso será limitado apenas a avaliação de aderência de práticas do *Agile Testing*, em projetos que utilizam *Scrum* e, a uma população alvo pequena, devido à limitação de tempo para a realização deste trabalho. Sendo assim, as conclusões obtidas neste estudo não poderão ser generalizadas e apenas contribuirão para direcionar estudos posteriores.

1.4. Organização do trabalho

Este trabalho é composto por cinco capítulos, que estão organizados da seguinte maneira:

- O capítulo 1 tem como objetivo apresentar a motivação para a execução deste trabalho, a organização do mesmo e os objetivos que se pretende alcançar durante a sua execução;
- O capítulo 2 apresenta a fundamentação teórica para o desenvolvimento deste trabalho;
- O capítulo 3 apresenta a metodologia com o delineamento da pesquisa, com a definição da população alvo, as técnicas de coleta e análise de dados, as limitações do método de estudo e as etapas desenvolvidas.
- O capítulo 4 apresenta um estudo de caso realizado, a análise e coleta de dados levantados e a análise crítica realizada a partir do levantamento de dados;
- Por fim, o capítulo 5, apresenta as conclusões obtidas através do estudo realizado.

2. Fundamentação Teórica

Desenvolver um software pode ser imprevisível e complicado, isso porque, por mais que a tecnologia e o negócio sejam de domínio da equipe, o software que será desenvolvido nunca foi construído antes, da mesma forma, com a mesma equipe, sob as mesmas circunstâncias. Desenvolver software pode ser considerado um processo empírico, que aceita a imprevisibilidade, tem mecanismos de ação corretiva e principalmente, que gera aprendizado através de experiências vividas [Agile 2001].

De acordo com suas características, uma empresa pode adotar como método de desenvolvimento, uma metodologia ágil e, além disso, um time ágil pode utilizar técnicas e práticas que apoiam execução do processo e garantem *feedback* instantâneo. Uma prática que pode ser adotada é o *Agile Testing*, que reúne tipos e técnicas de testes que visam atender ao Manifesto Ágil.

Portanto, neste capítulo serão abordados:

- ✓ O conceito de Agilidade, enfatizando o Manifesto Ágil, que possui relação com *Agile Testing*;
- ✓ A Metodologia Ágil *Scrum*, que é utilizada pela empresa onde será realizado o estudo de caso;
- ✓ Integração contínua;
- ✓ A prática *Agile Testing*, assim como seus tipos e técnicas de testes previstos.

2.1. Agilidade

Em 2001, um grupo de especialistas em processos de desenvolvimento de software, que compartilhava o objetivo de descobrir maneiras melhores de desenvolver *software*, criou e assinou o "Manifesto para Desenvolvimento Ágil de Software". Esta iniciativa deu origem ao movimento ágil, deu visibilidade para metodologias adaptativas que já existiam, e estimulou o surgimento de novas metodologias e técnicas [Agile 2001].

Através deste trabalho, este grupo passou a valorizar: Indivíduos e interações mais do que processos e ferramentas; Software em funcionamento mais do que documentação abrangente; Colaboração com o cliente mais do que negociação de contratos; Responder a mudanças mais que seguir um plano. O grupo ficou conhecido como Aliança Ágil e estabeleceu 12 princípios:

1. Satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado;
2. Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis se adaptam a mudanças que visam vantagem competitiva para o cliente;
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. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessários, e confie neles para que façam 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 equipes auto-organizáveis;
12. Em intervalos regulares, a equipe deve refletir sobre como se tornar mais eficaz, e então refina e ajusta seu comportamento para tal.

Nem todo modelo de processo ágil aplica esses 12 princípios atribuindo-lhes pesos iguais, e alguns modelos preferem ignorar ou pelo menos relevam a importância de um ou mais desses princípios. Entretanto, os princípios definem um espírito ágil mantido em cada um dos modelos [Pressman 2011].

Agilidade tornou-se a forma de descrever uma série de fatores que envolvem o time de desenvolvimento, entre eles: cultura, metodologia, valores, princípios, técnicas e práticas de desenvolvimento, - entre outros. Um time ágil, é aquele capaz de responder apropriadamente a mudanças que poderão ter um impacto no produto que está em desenvolvimento ou no projeto [Jacobson 2002].

Mas, agilidade vai além de resposta à mudança, abrangendo a filosofia de proposta ao manifesto ágil. Ela enfatiza a entrega rápida do software operacional e diminuem a importância dos artefatos intermediários; Assume o cliente como parte da equipe de desenvolvimento e trabalha para eliminar atitudes de "nós e eles", que continuam a invadir os projetos de softwares. Reconhece que o planejamento em um mundo incerto tem seus limites e que o plano de projeto deve ser flexível [Pressman 2014].

Segundo [Pressman 2014], agilidade pode ser aplicada em qualquer processo de software, entretanto, para obtê-la é necessário que o time:

- ✓ Possa adaptar e alinhar tarefas;
- ✓ Possa conduzir o planejamento compreendendo a fluidez de uma abordagem de desenvolvimento ágil;
- ✓ Possa eliminar tudo exceto os artefatos essenciais, conservando-os enxutos;
- ✓ Enfatize a estratégia de entrega incremental, conseguindo entregar ao cliente, de forma mais rápida possível, o software operacional.

Segundo [Pressman 2014], a principal diferença entre as metodologias ágeis e as metodologias tradicionais é que as metodologias ágeis são adaptativas, enquanto as tradicionais são prescritivas. Ou seja, metodologias ágeis se adaptam a novos fatores durante o desenvolvimento do projeto, em vez de tentar analisar previamente tudo o que pode ou não acontecer no decorrer do desenvolvimento. Essa análise prévia, praticada em metodologias prescritivas, é sempre difícil e apresenta alto custo, além de se tornar um problema quando for necessário fazer alterações nos planejamentos. Portanto, pode-se considerar que metodologias adaptativas conseguem trabalhar com mudanças de forma mais eficiente.

2.2. *Scrum*

O *Scrum* surgiu no início dos anos 90, como um framework estrutural, e vem sendo usado desde então para gerenciar o desenvolvimento de produtos complexos. Com

auxilio dele, pessoas podem tratar e resolver problemas complexos e adaptativos, enquanto produtiva e criativamente entregam produtos com o mais alto valor possível [Schwaber e Sutherland 2013].

O *Scrum* pode ser considerado leve, simples de entender e, em contrapartida, extremamente difícil de dominar. Uma característica do *Scrum* é demonstrar a eficácia das práticas de gerenciamento e desenvolvimento de produtos, permitindo que as pessoas que o utilizam possam melhorá-las. É importante ressaltar que o *Scrum* não é um processo ou técnica de desenvolvimento, mas sim um framework, que possibilita utilizar vários processos ou técnicas, enfatizando a melhoria contínua [Alliance 2012].

A figura abaixo (Figura 1) representa o funcionamento do *Scrum*, com seus Papeis, Eventos e Artefatos. Cada um deles será detalhado nas próximas sessões.

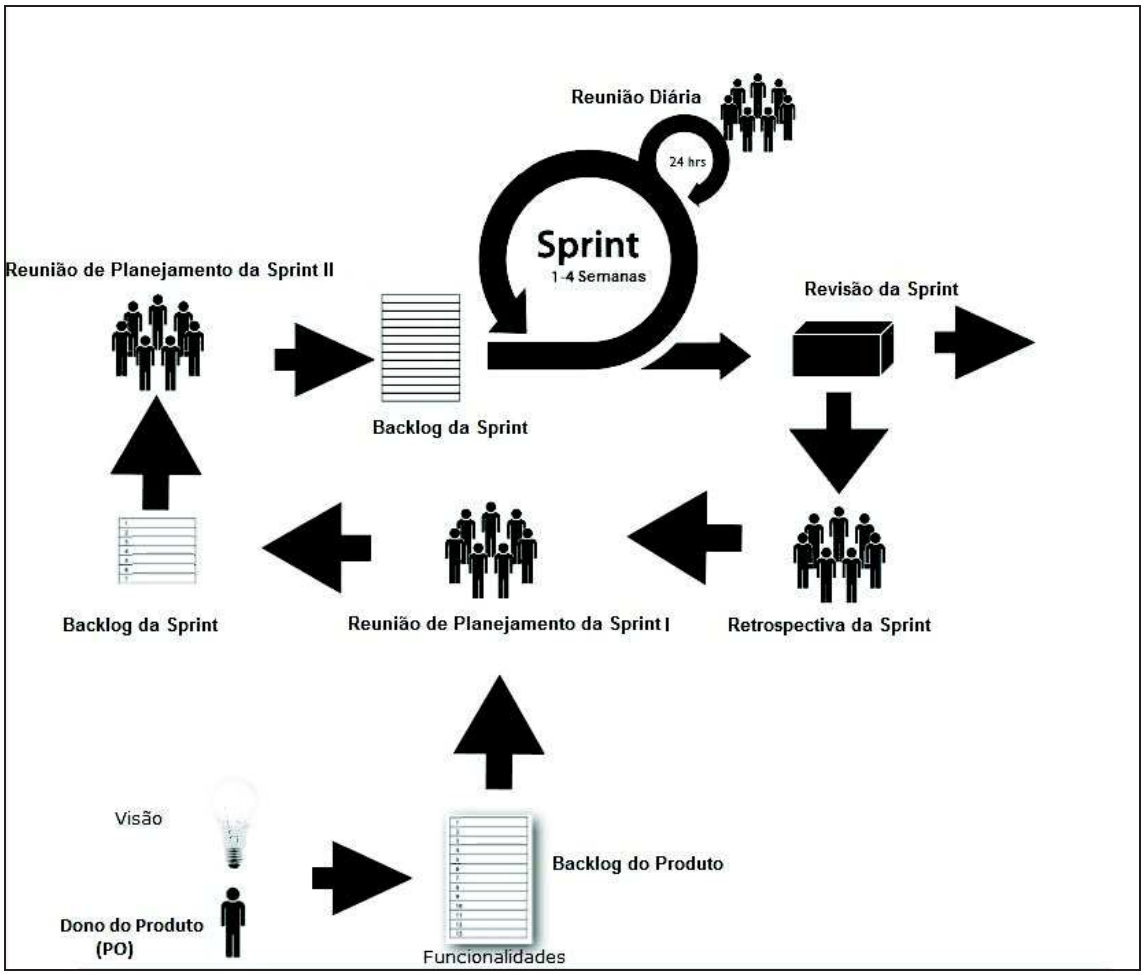

Figura 1: *Framework Scrum*

Fonte: [Sutherland 2014] – Traduzida pelo autor.

2.2.1 Time *Scrum*

Times *Scrum* são considerados multifuncionais, isso porque possuem todas as competências necessárias para completar o trabalho sem depender de pessoas que não

fazem parte do time [Sutherland 2014]. Para que isso seja possível, o *Scrum* possui os seguintes papéis:

- ✓ **Dono do Produto (*Product Owner* - PO):** É responsável por gerenciar o Backlog do Produto, podendo representar os interesses de um grupo de pessoas [Brito 2012].
- ✓ **Time de desenvolvimento:** O time de desenvolvimento consiste dos profissionais que realizam o trabalho para entregar um incremento de produto ao término de cada iteração. Pode ser composto por Desenvolvedores, Analistas de Sistemas e Analistas de Testes [Schwaber e Sutherland 2013].
- ✓ **Scrum Master (SM):** É o agente de mudanças dentro do *Sprint*, tem a responsabilidade de difundir o Scrum dentro da organização, garantir que o Scrum seja entendido e que haja aderência a seus valores, atuando como um líder servidor para a equipe, e é também o facilitador dos eventos do Scrum [Brito 2012].

2.2.2. Eventos Scrum

O Scrum utiliza eventos prescritos especificamente projetados para permitir uma transparência e inspeção criteriosa [Sutherland 2014]. São eles:

- ✓ ***Sprint*:** é um evento com tempo pré-definido de um mês ou menos, durante o qual um incremento de produto utilizável é criado. Não existem intervalos entre *Sprint*s, uma nova *Sprint* começa imediatamente após a conclusão da anterior. A *Sprint* agrupa alguns outros eventos do Scrum, dessa forma, nenhuma mudança que afete a meta do *Sprint* é realizada, equipe e qualidade são constantes, o escopo pode ser negociado, pois podem mudar à medida que os requisitos são mais bem entendidos [Brito 2012].
- ✓ **Planejamento da *Sprint*:** A reunião de planejamento da *Sprint* pode ser dividida em duas partes:
  - Reunião de Planejamento da *Sprint* Um (SP1): onde o PO apresenta os requisitos do Backlog do Produto e, o time colabora no dimensionamento dos requisitos selecionados para compor o *Backlog* da *Sprint*;
  - Reunião de Planejamento da *Sprint* Dois (SP2): onde o time define como construirá o incremento de produto [Brito 2012].
- ✓ **Reunião Diária:** É uma reunião realizada diariamente em um horário definido, e com duração de 15 minutos entre os membros do time, com a finalidade de sincronizar as atividades e de criar um plano para o próximo dia de trabalho. Durante a reunião os membros do Time de Desenvolvimento esclarecem:
  - O que eu fiz ontem que ajudou o Time a atender a meta da *Sprint*?
  - O que eu farei hoje para ajudar o Time a atender a meta da *Sprint*?
  - Eu vejo algum obstáculo que impeça a mim ou o Time no atendimento da meta da *Sprint*?
- ✓ **Revisão da *Sprint*:** Realizada ao final de cada *Sprint*, tem a finalidade de inspecionar o incremento e adaptar o Backlog do Produto. O time apresenta o incremento de produto e obtém – ou não – a aprovação do PO. O resultado da Revisão da *Sprint* é um Backlog do Produto revisado, que define os prováveis itens do Backlog do Produto para o *Sprint* seguinte. O Backlog do produto pode também ser ajustado para satisfazer a novas oportunidades [Brito 2012].

- ✓ **Retrospectiva da *Sprint*:** Ocorre logo após a reunião de revisão da *Sprint*, é uma oportunidade para o time identificar e ordenar os itens principais que potencialmente podem melhorar e criar um plano para implementação de melhorias [Brito 2012].

2.2.3. Artefatos Scrum

Artefatos Scrum representam o trabalho ou o valor de várias maneiras que são úteis no fornecimento de transparência e oportunidades para a inspeção e adaptação. Artefatos definidos pelo Scrum são especificamente concebidos para maximizar a transparência de informações-chave necessárias para garantir que equipes Scrum sejam bem-sucedidos na concretização de um incremento [Brito 2012].

- ✓ **Backlog do Produto:** Consiste em uma lista ordenada com tudo o que possa ser necessário no produto e é a única fonte de requisitos do mesmo [Brito 2012];
- ✓ **Backlog da *Sprint*:** É uma porção de requisitos do Product Backlog, priorizados para serem desenvolvidos dentro da *Sprint* [Brito 2012];
- ✓ **Incremento:** Consiste no resultado funcional da implementação dos itens do Backlog da *Sprint*, com maturidade para ser publicada em produção [Brito 2012].

2.4. Integração Contínua

Integração Contínua é uma prática de desenvolvimento de software onde os membros de um time integram seu trabalho frequentemente, geralmente cada pessoa integra pelo menos diariamente – podendo haver múltiplas integrações por dia. Cada integração é verificada por um build automatizado (incluindo testes) para detectar erros de integração o mais rápido possível. Segundo [Fowler 2006] o ambiente de Integração Contínua pode reduzir os riscos comuns do projeto, aumentando a confiança e melhorando comunicação.

Para que seja possível compreender o conceito de Integração Contínua, é necessário compreendermos o que é um build. Esta, consiste em uma compilação, execução de testes, inspeção de código, implantação em um ambiente, entre outras coisas. A Figura 2 ilustra os passos típicos de um cenário de Integração:

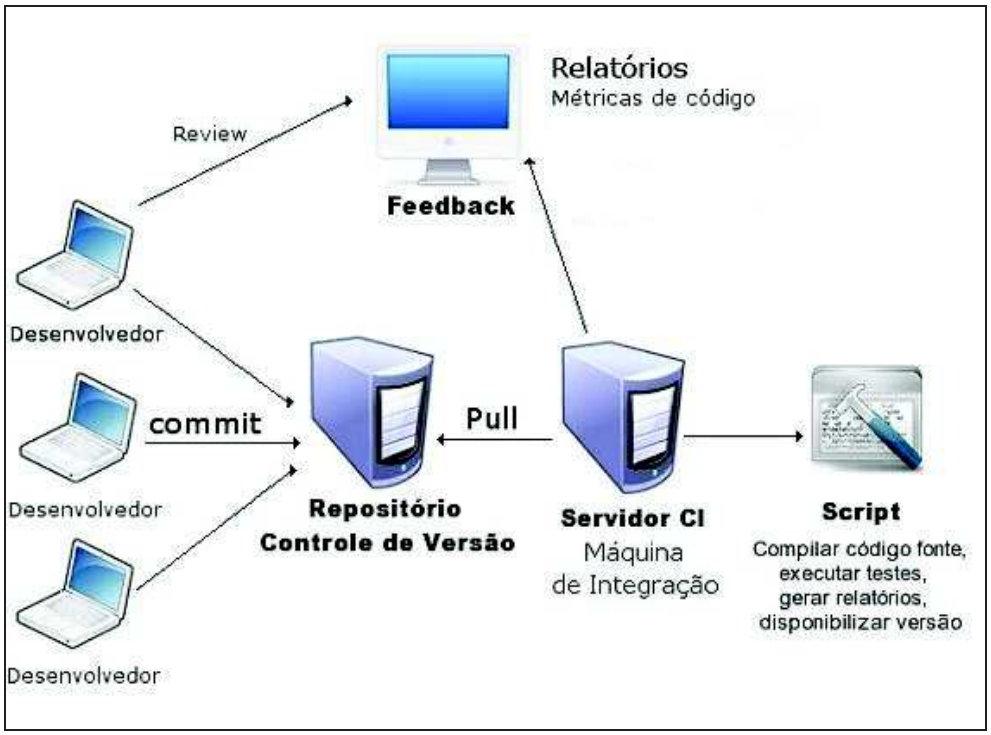

Figura 2: Passos de um ambiente de Integração Contínua.

Fonte: [Moreira 2013].

Observe que na figura, o desenvolvedor realiza um *commit* do seu código para um repositório de controle de versão. Depois do *commit*, o servidor de Integração Contínua detecta as alterações que foram realizadas, realiza uma cópia da versão atual e executa o build, com as integrações, testes, entre outros. O servidor de Integração Continua gera o feedback do resultado do build, por e-mail, para os interessados.

Uma vez que um desenvolvedor (entende-se por qualquer pessoa que faz uma alteração no código do projeto) conclui sua tarefa e então submete as suas alterações ao repositório de controle de versão, um build integrado ocorre com as mudanças aplicadas ao repositório de controle de versão [Fowler 2006]. Para ter um ambiente com Integração Contínua, é necessário utilizar um repositório de controle de versão. A finalidade de um repositório de controle de versão é gerenciar mudanças para código fonte e outros ativos de software (como documentação) usando um repositório de acesso controlado [Humble e Farley 2010].

Existem alguns termos e conceitos básicos de um repositório de versão:

- Repositório: É o local onde estão contidos todos os arquivos do projeto;
- *Commit*: Ato de envio das modificações realizadas localmente ao repositório;

Um dos objetivos fundamentais da Integração Contínua é produzir resposta sobre uma compilação de integração, permitindo saber o quanto antes possível se houve um problema com a compilação. Um mecanismo de resposta que pode ser utilizado é um e-mail que pode ser enviado para os interessados no Plano (*Script Build*), assim, ao receber imediatamente essas informações, o time pode atuar rapidamente no problema [Fowler 2006].

2.5. Teste de Software

Software está cada vez mais presente em diversas atividades que antigamente eram executadas manualmente, seja no controle de transações bancárias, até no controle do tráfego aéreo. Independente da área onde o software está atuando, um defeito no código, que dará origem a uma falha, sempre apresenta prejuízo à empresa, seja financeiro, de imagem, de credibilidade, de confiança e até mesmo, de vidas. Investimento em testes e documentação podem reduzir os riscos decorrentes de falhas, se os defeitos forem encontrados em tempo hábil para que possam ser corrigidos sem causarem impacto ao cliente [Bastos, Cristalli, Moreira e Rios 2012]. Algumas atividades de teste contemplam:

- ✓ **Planejamento e controle do teste:** Planejamento consiste em definir os objetivos do teste e especificar as atividades de forma a alcançá-los. O planejamento de teste considera o retorno das informações das atividades de monitoração e controle. Controle consiste em comparar o andamento das atividades de teste e comparar com o planejado, também envolve a tomada de decisões para ações necessárias, de modo a atingir o planejado [Bastos, Cristalli, Moreira e Rios 2012].
- ✓ **Análise e modelagem:** Nesta atividade os objetivos de teste definidos no Planejamento são transformados em condições e modelos de teste. Nela ode ser necessário revisar a base de testes, avaliar a testabilidade dos requisitos e do sistema, identificar e priorizar as condições ou requisitos de teste, projetar e priorizar os casos de teste, analisar a necessidade de dados, planejar a preparação o ambiente, criar uma rastreabilidade bidirecional entre os requisitos e os casos de teste [Bastos, Cristalli, Moreira e Rios 2012].
- ✓ **Implementação e execução dos testes:** Nesta atividade são implementados e priorizados os casos de teste analisados, são desenvolvidos e priorizados os procedimentos de teste e são criadas suítes de teste a partir dos casos de teste implementados. Também, deve ser verificado se o ambiente está preparado corretamente e se a rastreabilidade bidirecional esta atualizada. Esta atividade também contempla a execução dos testes, que pode ser feita manualmente ou utilizando ferramenta especifica. Os resultados obtidos são comparados com os esperados, e as divergências são reportadas para serem analisadas, corrigidas, e se necessário, testadas novamente [Bastos, Cristalli, Moreira e Rios 2012].
- ✓ **Atividades de encerramento de teste:** Nesta atividade são checados os entregáveis planejados, os relatórios de incidentes são fechados, é feita a documentação do aceite do sistema, é arquivado o ambiente de teste a infraestrutura para reuso, são levantadas e analisadas as lições aprendidas para futuras versões [Bastos, Cristalli, Moreira e Rios 2012].

De acordo com atividades descritas, pode-se concluir que testar da maneira tradicional pode ser ineficaz em um contexto ágil, isso porque as atividades de teste estão relacionadas às atividades de desenvolvimento e, portanto, testes variam para diferentes ciclos de vida [ISTQB 2014]. Testadores devem compreender as diferenças entre os testes em modelos de ciclo de vida tradicionais (por exemplo, o modelo V ou RUP) e os ciclos de vida Agile, a fim de trabalhar de forma eficaz [ISTQB 2014]. Os modelos ágeis diferem em termos da forma como as atividades de teste e desenvolvimento são integradas, os produtos do projeto de trabalho, os critérios de

nomes de entrada e saída utilizados para vários níveis de testes, o uso de ferramentas, e como teste independente pode ser efetivamente utilizado [ISTQB 2014].

2.6. *Agile Testing*

Em projetos que utilizam métodos ágeis é preciso ter um ambiente de desenvolvimento que favoreça o feedback contínuo, ou seja, que o impacto de cada alteração seja percebido instantaneamente por toda equipe. Para que isso seja possível é recomendado que este ambiente possuísse integração contínua, onde serão executados todos os testes desenvolvidos do projeto. Com os resultados destes testes o time pode detectar em pouco tempo se a funcionalidade desenvolvida ainda precisa ser corrigida ou se gerou não conformidades em outras funcionalidades [Duval 2007].

Para que garantir que os testes sejam realizados da forma e na proporção corretas, foi criado o conceito *Agile Testing*. *Agile Testing* consiste na aplicação de um conjunto de tipos e técnicas de testes, já conhecidas pelo mercado, combinadas de forma que possam se complementar ao máximo, visando atender ao manifesto ágil. O principal objetivo é combinar as técnicas de testes de software para que seja possível aperfeiçoar o tempo de execução de testes e garantir a maior cobertura de testes [Crispin e Gregory 2009].

2.6.1. Diferenças entre Teste Tradicional e *Agile Testing*

A figura a seguir (Figura 3) ajuda a compreender as diferenças entre testes no desenvolvimento tradicional, usando como exemplo a metodologia Cascata, e testes no desenvolvimento ágil.

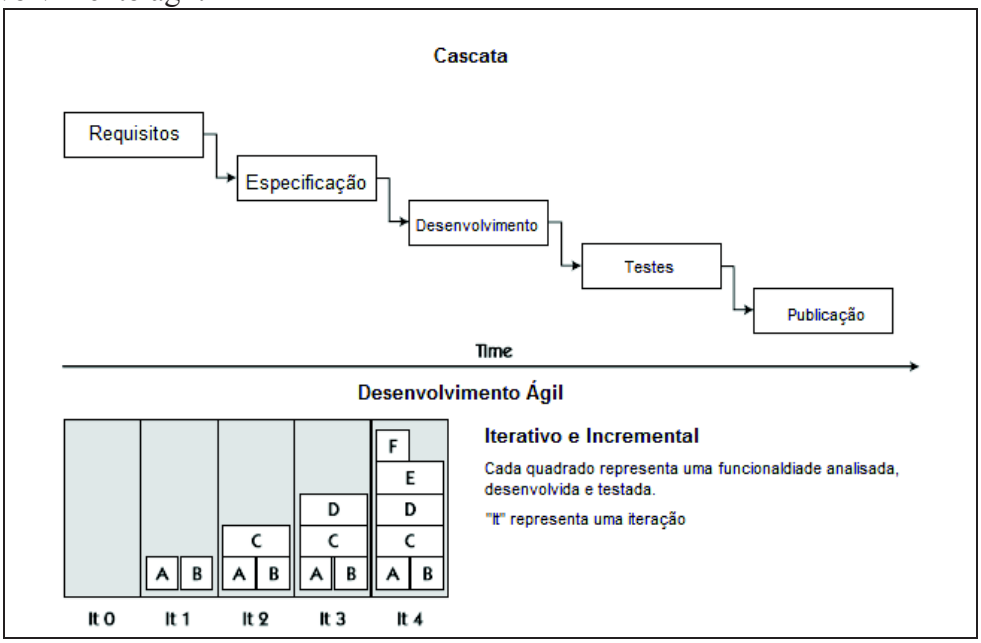

Figura 3: Testes no Desenvolvimento Tradicional versus Desenvolvimento Ágil

Fonte: [Crispin e Gregory 2009] – Traduzido pelo autor.

No diagrama de abordagem gradual, é evidente que o teste acontece no final, antes da publicação. O diagrama é idealista, porque dá a impressão de que há tanto tempo para o teste como existe para a codificação, o que em muitos projetos, não é a

realidade. O teste pode ser prejudicado porque a codificação leva mais tempo do que o esperado, e porque as equipes entregam um projeto pronto somente no final [Crispin e Gregory 2009]. É comum no desenvolvimento tradicional, os testadores receberem uma versão com todas as alterações do projeto e a partir de aí começarem suas verificações.

Por outro lado, o Ágil é iterativo e incremental, portanto os testadores podem testar cada incremento de codificação, assim que é desenvolvido. O time analisa, desenvolve e testa uma funcionalidade ou parte dela apenas, e segue fazendo isso até que a solicitação do cliente seja atendida. O time deve terminar a codificação e testes no mesmo momento, porque uma funcionalidade não é "pronta" até que tenha sido testada [Schwaber e Sutherland 2013].

Existem muitas abordagens para se desenvolver um projeto ágil. Um time pode ser dedicado a um único projeto ou um projeto pode ser divido em várias em times. Independentemente de como um projeto é iniciado, é necessário que todo time, e isso inclui o testador, tenha entendimento da visão deste projeto, das principais funcionalidades, do problema que este projeto resolve e da necessidade do cliente [Crispin e Gregory 2009].

No desenvolvimento tradicional, é comum a criação de um plano e de estratégias de testes baseando-se em um documento de requisitos que foram criados por analistas de negócios antes sequer de alguém ter pensado em escrever uma linha de código [Bastos, Cristalli, Moreira e Rios 2012]. Por outro lado, no Ágil, um membro do time terá que pensar nos testes que ilustram os requisitos antes da codificação começar. Isso requer colaboração entre um analista de negócios e um membro do time, que pode ser um analista de testes, analista de sistemas ou desenvolvedor [Crispin e Gregory 2009].

Para cada funcionalidade, devem ser elaborados cenários de teste funcionais detalhados, de preferência com base em exemplos fornecidos por analistas de negócio, com critérios de aceitação. Em um projeto ágil, o planejamento e estratégia de testes podem ser abordados de maneira diferente em cada projeto, equipe e, às vezes, iteração.

2.6.2. Funções de um Analista de Testes em um Time Ágil

Uma das maiores diferenças do desenvolvimento ágil em relação ao desenvolvimento tradicional é a abordagem de time. No desenvolvimento ágil, não são apenas os analistas de testes que se sentem responsáveis pela qualidade, todos os membros do time devem priorizar produzir software de alta qualidade em um período de tempo que maximiza o seu valor para o negócio. Todos os membros de um time ágil ficam "infectados pelo teste", e isso inclui desenvolver testes a partir do nível de unidade [Schwaber e Sutherland 2013].

Enquanto no desenvolvimento utilizando uma metodologia tradicional, os testes são realizados por uma equipe de testes independente, no ágil, os analistas de testes devem trabalhar com o time do projeto. Uma das premissas do *Agile Testing* é que os analistas de testes sejam membros do time de desenvolvimento, e as tarefas de testes sejam tratadas com a mesma atenção que outras tarefas.

Times de projetos ágeis são geralmente considerados multifuncionais, pois possuem membros de diversas origens. A diferença entre uma equipe multifuncional tradicional e um time ágil é a abordagem de time. No ágil, os membros não estão apenas representando suas atribuições, mas estão se tornando parte do time durante o tempo que o time ou o projeto existem, conforme o que está sendo demonstrado na Figura 4.

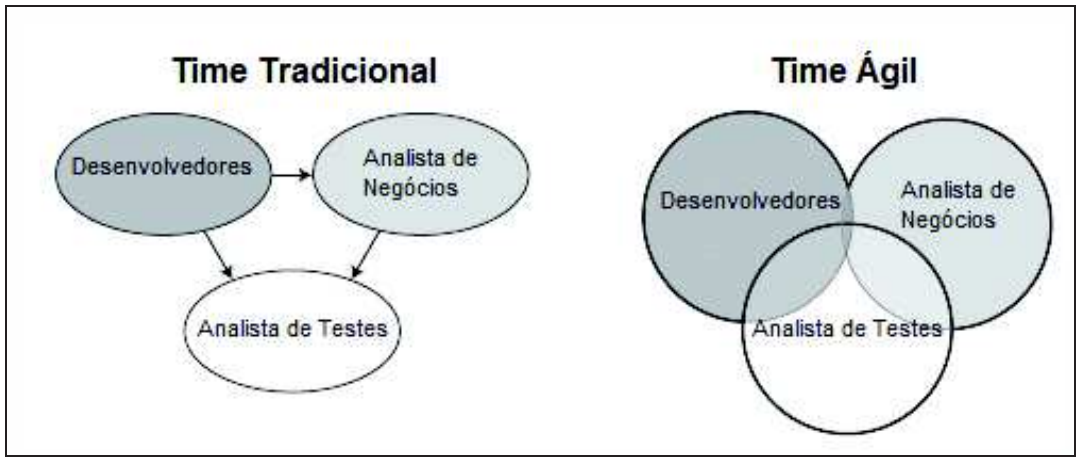

Figura 4: Conceito de time no Desenvolvimento Tradicional versus Desenvolvimento Ágil

Fonte: [Crispin e Gregory 2009] – traduzida pelo autor.

Um analista de testes, em um time ágil, pode ajudar a resolver problemas, seja fornecendo maneiras de reduzir o tempo de resposta de alterações ou auxiliando na priorização de não conformidades. Segundo [Crispin e Gregory 2009], como um membro de um time ágil, o analista de testes terá que se adaptar às necessidades do time, que podem ser, por exemplo:

- ✓ Realizar testes exploratórios manuais para encontrar erros importantes, de fluxos que podem não estar definidos nos cenários do plano de teste;
- ✓ Trabalhar em par com desenvolvedores para escrever testes: unitários, de integração e automatizados;
- ✓ Fazer testes manuais para verificar as funcionalidades de uma publicação;
- ✓ Participar das cerimônias;
- ✓ Executar testes automatizados;
- ✓ Fazer testes de performance, carga e stress;
- ✓ Escrever documentação funcional, entre outras atividades.

Segundo [Sutherland 2014], um time ágil deve assumir que cada membro da equipe tem o mesmo valor: Cada membro do time tem o mesmo valor para o time. Se analistas de testes ou qualquer outro membro do time se sentem excluídos ou menos valorizados, a abordagem de time está condenada. Os analistas de testes devem ser convidados para todas as reuniões do time, mesmo que sejam técnicas, por exemplo;

2.6.3. Quadrantes do *Agile Testing*

Agile Testing é uma prática que segue os princípios ágeis para entregar ao usuário um produto com maior qualidade. Esta prática agrupa técnicas de testes conhecidas a fim de que se complementem ao máximo, assim, testes deixam de ser uma forma de evitar a entrega de um sistema com não conformidades, e passam a fazer parte de uma política para desenvolver um bom produto desde o início do projeto.

Para que isso seja possível, as práticas de testes foram relacionadas de acordo com seu contexto de aplicação. Em um eixo, estão divididos testes que "Suportam o Time" e os testes que "criticam o Produto". O outro eixo testes Voltados a Negócios e testes voltados para Tecnologia:

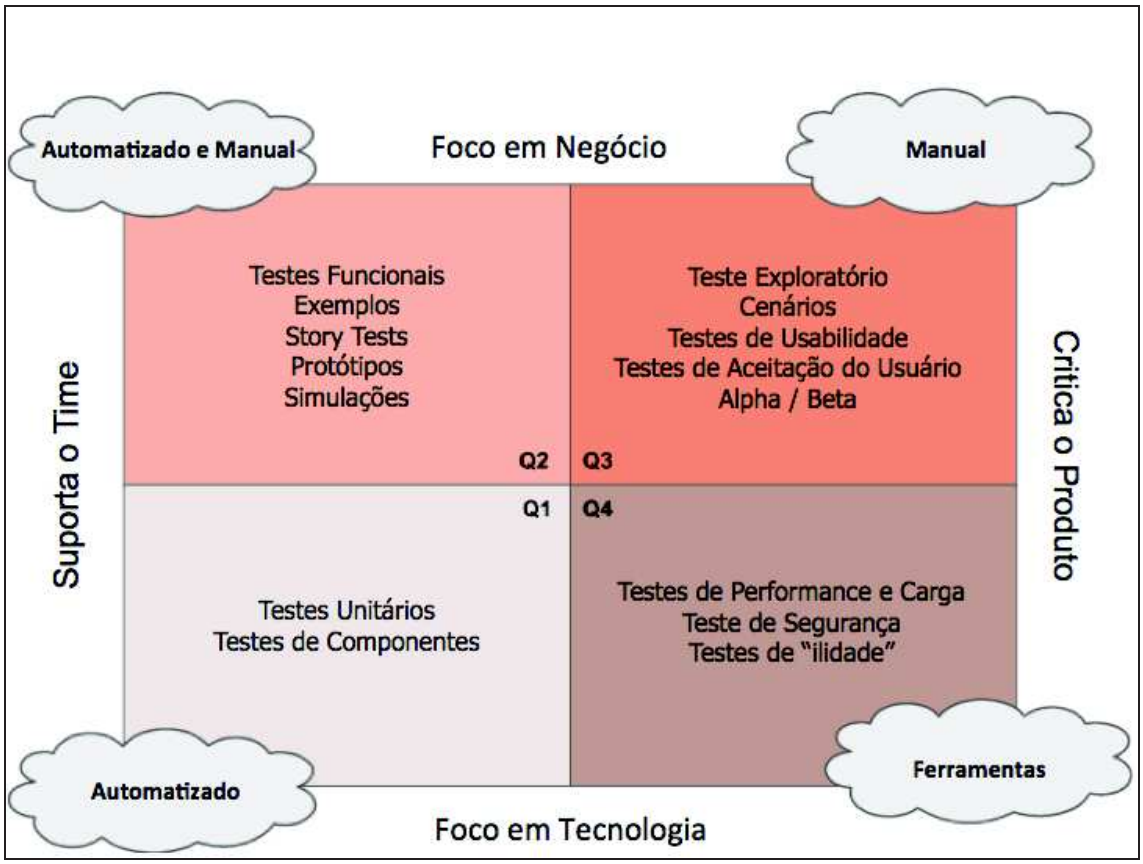

Figura 5: Quadrantes do *Agile Testing*

Fonte: [Nogueira 2013]

Os quadrantes da esquerda devem incluir testes que auxiliam o time no desenvolvimento da solução ou produto. Os testes realizados nos quadrantes Q1 e Q2 estão relacionados com especificação de requisitos e definição da solução [Crispin e Gregory 2009]. Já os quadrantes Q3 e Q4 devem incluir testes que verifiquem se a solução foi desenvolvida da forma correta, atendendo aos requisitos funcionais e não funcionais, envolvendo testes que revisam o produto de forma construtiva, a fim de melhorá-lo [Crispin e Gregory 2009].

Nem sempre a execução de testes segue a ordem numérica dos quadrantes, ou seja, existem algumas situações em que testes do Q2 podem ser feitos antes dos testes do Q1. Segundo [Crispin 2011], é comum projetos começarem aplicando testes do Q2 para entender o problema do cliente e criando cenários para validar uma solução. Isso poque, neste quadrante, são elaborados os exemplos que podem se transformam em especificações, e estas, irão impulsionar o desenvolvimento.

2.6.3.1. Q1 – Testes que utilizam tecnologia para apoiar o time

No Quadrante Um (Q1), estão os testes unitários e testes de componentes, que visam garantir a qualidade, ajudando o desenvolvedor a entender exatamente o que o código precisa fazer, a se concentrar na funcionalidade que está sendo entregue a fim de adotar a abordagem de desenvolvimento mais elegante, simples e de manutenível. Uma

característica ressaltada pelo *Agile Testing*, é que a responsabilidade de escrever estes testes geralmente é do desenvolvedor [Crispin 2011].

O teste de componente verifica o funcionamento de componentes testáveis separadamente, como por exemplo, módulos, programas, objetos, classes e até mesmo métodos, que são testáveis separadamente. Por essa característica, pode ser feito isolado do resto do sistema, dependendo do contexto do ciclo de desenvolvimento e do sistema [Crispin e Gregory 2009]. Teste de componente pode incluir testes funcionais, não funcionais, além de teste estrutural (cobertura de código). Os testes de unidade verificam a funcionalidade de um pequeno subconjunto do sistema, tais como um objeto ou método. Um conjunto de testes de unidade pode compor um teste de componente [Meszaros 2007].

Uma das principais técnicas aplicadas neste quadrante é o Desenvolvimento Orientado a Testes, conhecido pela sigla TDD (*Test-Driven Development*). De acordo com [Beck 2002], TDD é uma abordagem evolutiva na qual o desenvolvedor escreve o teste antes de escrever o código funcional necessário para satisfazer aquele teste. O TDD possibilita uma forma de refletir sobre a modelagem antes de escrever código funcional, e também possibilita obter um código fonte bem testado.

Basicamente, o primeiro passo no TDD é escrever os testes unitários, que validam a funcionalidade que está sendo desenvolvida. Após isso, os testes são executados e, falham. Então, são escritos os métodos que farão com que o teste passe. E então, os métodos e os testes são refatorados, ou seja, melhorados [Beck 2002]. A Figura 6 ilustra o mantra do TDD.

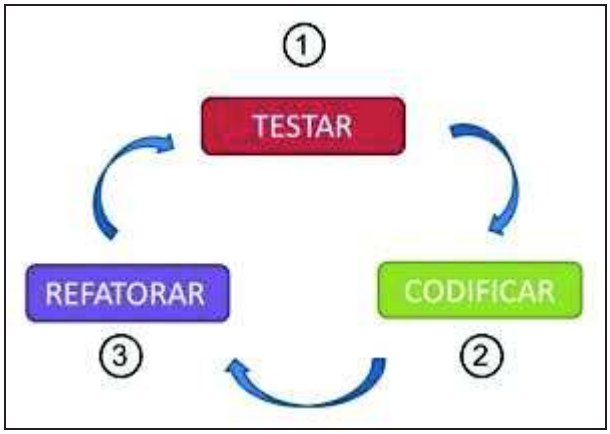

Figura 6: Mantra do TDD

Fonte: [Beck 2002] – Traduzido pelo autor.

Com TDD, possíveis erros em nível de unidade são identificados e corrigidos pelo próprio desenvolvedor. Por isso, este tipo de teste é tão importante no desenvolvimento ágil. Ao construir o código em pequenos incrementos, o desenvolvedor tem a chance de pensar através da necessidade do cliente. E, quando o teste é feito antes da codificação, o código já nasce naturalmente testado. Por isso, no Q1, as atividades são destinadas a produzir software com a mais alta qualidade interna possível.

Escrever testes de Unidade e de Componente, usando o TDD, é extremante importante para o time de desenvolvimento porque estes testes permitem:

1) **Que o time seja mais produtivo:** Produtividade é um efeito colateral de longo prazo da produção de código com qualidade. Testes unitários podem avisar o time caso exista alguma falha, e o erro pode ser encontrado e corrigido rapidamente. Testes de unidade permitem que os desenvolvedores possam refatorar o código com frequência, mantendo-o em funcionamento [Beck 2002]. Segundo [Beck 2002], quando o teste de unidade é negligenciado, um Analista de Testes pode gastar todo seu tempo de testes encontrando falhas no fluxo básico, sem ter tempo para testar cenários mais complexos.

2) **Reduzir o número de erros básicos:** As principais práticas relacionadas com os testes desenvolvidos por desenvolvedores facilitam o trabalho dos Analistas de Testes. Escrevê-los, significa primeiro que os desenvolvedores estão produzindo um código testável. E, esta infraestrutura ajudará ao time quando forem escritos os testes voltados para o negócio, e aqueles que criticam o produto. Todo o time está continuamente pensando em maneiras de melhorar a criação do software [Beck 2002].

3) Projetar o software pensando em testes: Uma das vantagens do TDD é que o código expressa intenção de fazer os testes passarem. O time deve pensar sobre como desenvolver e executar os testes automatizados para cada funcionalidade. Sendo assim, o time inteiro está comprometido com os testes e em como pode ser criada uma arquitetura que facilite o trabalho de automação [Beck 2002].

4) *Feedback* Instantâneo: O maior valor de testes de unidade está na velocidade de resposta, que pode ser a cada construção do projeto, ou a cada execução da Integração Continua [Beck 2002].

2.6.3.2. Q2 – Testes de negócio que apoiam o time

Conforme [Crispin 2011], os primeiros testes realizados em uma funcionalidade, geralmente são os testes do Q2, por serem os que auxiliam no entendimento da necessidade do cliente, que visam verificar se a solução proposta atende aos requisitos de negócio, e ajudam a fornecer detalhes suficientes para orientar a codificação.

Estes testes devem expressar necessidades do cliente baseadas em exemplos ou cenários, usar uma linguagem que o cliente e o time possam entender. No Q2 são realizados Testes Funcionais, Exemplos, Historias de Usuários e Simulações. Estes testes podem ser executados por qualquer membro do time de desenvolvimento, e devem ser executados de formas manuais e automatizados [Crispin e Gregory 2009].

Testes funcionais são aqueles que verificam as funcionalidades que um sistema, subsistema ou componente devem realizar. Visam testar por completo as funcionalidades que representam o que o sistema faz [ISTQB 2014]. **Exemplos** podem ser dados fornecidos pelo cliente, que servem de entrada e saída para um teste [ISTQB 2014]. **Simulações** podem contemplar dados do ambiente de produção, e seu objetivo é simular um usuário real, operando o *software* [ISTQB 2014].

História de Usuário é uma descrição concisa de uma necessidade do usuário do produto (ou seja, de um "requisito") sob o ponto de vista desse usuário. A história de usuário busca descrever essa necessidade de uma forma simples e leve. Um dos princípios por trás das Historias de Usuários é o de que o produto poderia ser integralmente representado por meio das necessidades de seus usuários. Existem histórias que são mais complexas que as demais, e por consequência, levam mais tempo para serem desenvolvidas, elas são chamadas de **épicos**. Neste caso, o time de

desenvolvimento pode solicitar ao cliente quebrá-las em histórias menores [Smart 2014].

Uma técnica que pode ser utilizada para automatizar testes de história de usuários é o BDD. No BDD, os testes basicamente são compostos em duas partes: a definição da funcionalidade a ser implementada (História de Usuário) e os cenários de uso que validarão esta funcionalidade. BDD pode ser traduzido para Desenvolvimento Dirigido ao Comportamento, isso porque ele colabora para que o desenvolvimento objetive a entrega de valor, através da formação de um vocabulário comum, reduzindo a distância entre o Cliente e o time de desenvolvimento [Cohn 2004].

Segundo [Cohn 2004] o BDD está fundamentado em três princípios:

1. O cliente (Negocio) e time de desenvolvimento (Tecnologia) devem "falar" sobre um sistema da mesma forma;

2. Qualquer sistema deveria ter um valor identificável e verificável para o "negócio";

3. Análise, design e planejamento precoce tem, sempre, retorno significativo.

BDD se apoia no uso de um vocabulário pequeno e bem específico, dessa forma facilita a comunicação entre os interessados, e garante que estejam alinhados. Como o BDD é guiado pelo comportamento do negócio, uma maneira de perceber o benefício produzido é pela interface gráfica, produzindo um feedback rápido para saber se os requisitos descritos através dos comportamentos estão funcionais [Smart 2014].

A linguagem de negócio usada em BDD é extraída das histórias ou especificações fornecidas pelo cliente durante o levantamento dos requisitos. Estas histórias e cenários podem ser automatizadas pelo time de desenvolvimento utilizando um *framework* de BDD. Segundo [Smart 2014] algumas vantagens em usar BDD são:

- **Comunicação entre equipes:** o BDD possibilita a todos interessados em projeto falarem utilizando a mesma linguagem – uma linguagem de negócio, e isso possibilita que a comunicação seja mais eficiente;
- **Compartilhamento de conhecimento**: todo fluxo de negócio está documentado e, a documentação é executável;
- **Documentação dinâmica**: com documentação executável, sua manutenção torna-se parte do processo de desenvolvimento, isso porque sempre que um cenário apresenta erro é necessária uma ação, pois pode ser que uma regra tenha sido alterada, ou realmente seja uma falha de programação;
- **Visão do todo**: BDD sugere que o time de desenvolvimento escreva 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.

A imagem a seguir demonstra uma história BDD com cenários descritos:

```
Narrativa:
Como um cliente,
Desejo retirar dinheiro de um caixa eletrônico,
Para obter maior agilidade no meu atendimento e para que eu não precise entrar na fila no banco

Cenário: Saquar dinheiro de conta que possui saldo

Dado que a conta possui saldo
E o cartão é válido
E o terminal possui dinheiro
Quando o cliente solicita um saque
Então o dinheiro será dispensado
E o valor sacado será debitado da conta
E o cartão será liberado
```

Figura 7: Exemplo de história de usuário e cenários BDD

Fonte: [North 2006] – Traduzido pelo autor.

2.6.3.3. Q3 – Testes voltados para negócio que apoiam o time

Estes testes visam criticar ou avaliar um produto manipulando o *software* tentando recriar as experiências reais dos usuários finais. Utilizam as percepções que os Analistas de Testes ou usuários finais têm da forma como o *software* se comporta e, portanto, podem ser realizados somente quando uma funcionalidade esteja concluída. Por isso, compreender os diferentes cenários de negócios e fluxos de trabalho ajuda a tornar estes testes mais eficientes, e torna a experiência mais realista [Crispin e Gregory 2009].

Devido a esta característica, é difícil automatizar testes voltados para o negócio que criticam o produto, porque tais testes necessitam da inteligência humana, experiência e instinto. Grande parte dos testes discutidos no Q3 é manual, mas os mesmos não serão suficientes para produzir software de alta qualidade. Ao chegar no Q3, é necessário que os testes dos Q1 e Q2 estejam automatizados. No Q3 estão os cenários de testes, testes exploratórios e testes de usabilidade [Crispin e Gregory 2009].

Cenários de Testes

Podem testar o sistema de ponta a ponta, na perspectiva de um usuário final. Ter domínio da vida real é fundamental para a criação de cenários precisos, desta forma, os usuários finais podem ajudar a definir cenários e fluxos de trabalho que podem imitar o seu comportamento. Para testar diferentes cenários, os dados utilizados e o fluxo precisam ser realistas. Neste caso, o cliente pode fornecer uma amostra, de dados para o teste. Os cenários visam garantir o correto funcionamento dos requisitos funcionais e não funcionais do sistema. Os requisitos podem estar como um texto, ou representados visualmente em diagramas. Diagramas de fluxo podem ajudar a definir cenários e pensar através de um problema complexo [Crispin e Gregory 2009].

Teste Exploratório

Teste exploratório é uma abordagem importante para testes no mundo ágil [Crispin e Gregory 2009]. Como uma ferramenta de investigação, é um complemento fundamental para os testes de história e para a suíte de regressão automatizada. É uma abordagem sofisticada e pensativa, por não possuir um roteiro, permite ir além das variações óbvias que já foram testadas [Crispin e Gregory 2009].

Testes de Usabilidade

Testes de usabilidade criticam o produto sob a perspectiva do usuário final. Para isso, existem algumas técnicas que ajudam o time a pensar nas necessidades do cliente e nos testes que podem ser executados. Uma delas, aplicada comumente em projetos ágeis, é a utilização de Testes de Persona. Personas são usuários típicos do sistema, que representam as necessidades de grupos de usuários, em termos de seus objetivos e características pessoais [Crispin e Gregory 2009]. Personas identificam o usuário, suas motivações, expectativas e metas, responsáveis pela condução do seu comportamento no sistema, e trazem os usuários para a vida, dando-lhes nomes, personalidades e muitas vezes uma foto. Embora as personas sejam fictícias, elas são baseadas no conhecimento de usuários reais.

Teste de Aceitação

Teste de aceitação geralmente é feito pelo cliente, Dono do Produto (PO) e usuários finais do software, sendo assim, o objetivo deste tipo de teste não é procurar defeitos, mas sim estabelecer a confiança [Crispin e Gregory 2009]. Teste de aceite pode avaliar a disponibilidade do sistema para entrar em produção, apesar de não ser necessariamente o último nível de teste. Por exemplo, teste de integração em larga escala pode vir após o teste de aceite de um sistema. De acordo com [Crispin e Gregory 2009] existem diferentes formas de executar testes do nível de aceite:

1. **Teste de Aceite de Usuário:** Normalmente verifica se o sistema está apropriado para o uso por um usuário com perfil de negócio.
2. **Alfa / Beta Teste:** São feitos pelos clientes do software, onde o Alpha teste é executado no ambiente do cliente, e Beta teste é feito em um ambiente disponibilizado pela empresa que está desenvolvendo.

2.6.3.4. Q4 – Testes que criticam os produtos voltados para tecnologia

Os testes voltados para a tecnologia que criticam o produto visam verificar requisitos não funcionais, preocupando-se com deficiências no produto a partir de um ponto de vista técnico. Em vez de usar a linguagem de domínio do negócio, descrevem os requisitos usando um vocabulário de domínio de programação. Os requisitos não funcionais incluem questões de configuração, segurança, desempenho, gerenciamento de memória, confiabilidade, interoperabilidade, escalabilidade, a recuperação, e até mesmo a conversão de dados.

Para que os testes do Q4 possam gerar valor para o time e para produto, é necessário ter um resultado esperado claramente definido [Ghirghiu 2005]. Também que se deve manter um registro com o resultado de cada teste. Este registro deve ser atualizado constantemente, para que os resultados tenham um histórico e versões possam ser comparadas. Outro ponto importante, é que o ambiente onde os testes serão realizados deve ser equivalente a um ambiente de produção [Ghirghiu 2005].

No Q4, [Crispin e Gregory 2009] defiram: teste de segurança, testes de performance, teste de carga e testes de “ilidade”, que incluem testes de: confiabilidade, interoperabilidade, escalabilidade, a recuperação, e até mesmo a conversão de dados. Os testes do Q4, listados abaixo, geralmente são realizados pelo Analista de Testes.

Teste de Segurança

Testes de segurança podem ser elaborados a partir de duas perspectivas: de dentro para fora (teste caixa-branca) e de fora para dentro (teste de caixa-preta) [Crispin e Gregory 2009]. Se a abordagem escolhida for caixa-branca, o código fonte será analisado estaticamente com uma variedade de ferramentas que tentam descobrir erros

de codificação comuns, que podem fazer a aplicação vulnerável a ataques. Já se a abordagem escolhida for caixa-preta uma técnica que pode ser utilizada é baseada na injeção de falhas, onde os testadores podem tentar fazer a aplicação falhar, utilizando alguns recursos como injeção SQL e mudanças bytes aleatórios na entrada arquivos, por exemplo.

Teste de Performance e Carga

Testes de performance geralmente são utilizados para identificar os pontos de gargalo de software, ou para estabelecer uma linha de base para futuros testes. Também, são utilizados para garantir a conformidade com os requisitos de desempenho, sendo assim, podem ajudar os interessados a tomar decisões relacionadas com a qualidade geral do aplicativo que está sendo testado [Crispin e Gregory 2009]. O teste de carga avalia o comportamento do software à medida que o número de usuários, acessando simultaneamente, aumenta.

2.6.4. Princípios do *Agile Testing*

Segundo [Hendrickson 2008], o *Agile Testing* estabelece nove princípios, que são:

1. Um processo de desenvolvimento que inclua testes e que resulta em um produto melhor;
2. Os testes não podem ser vistos apenas como uma fase do projeto;
3. Todos testam, não apenas o Analista em testes;
4. Uma funcionalidade só é considerada pronta se forem realizados os testes necessários;
5. Diminua o tempo entre a implementação de uma história e a validação com o usuário;
6. Erros encontrados devem ser resolvidos assim que possível;
7. Testes descobrem requisitos não explicitados;
8. Reduza a documentação de teste;
9. Adote, sempre que possível, a técnica de TDD.

3. Metodologia

Neste capítulo é apresentado o delineamento da pesquisa bem como a definição da população-alvo e as técnicas para coleta e análise de dados. Além disso, são destacadas as limitações e as etapas desenvolvidas neste trabalho.

3.1. Delineamento da Pesquisa

A pesquisa será realizada em duas etapas: levantamento bibliográfico e estudo de caso. O levantamento bibliográfico fará uma abordagem sobre Metodologias Ágeis, em seguida serão detalhadas algumas características do *Scrum* e a prática *Agile Testing*. O estudo de caso mapeará o processo de desenvolvimento do time onde este será aplicado, e com base no estudo bibliográfico realizado, será avaliado se as atividades de testes executadas são aderentes a prática *Agile Testing*. O objetivo será identificar se os testes são realizados no momento certo, de acordo com o processo de desenvolvimento, e, se o time em questão está aderente aos princípios do *Agile Testing*.

A utilização do método de pesquisa estudo de caso deve-se ao fato que o pesquisador tem pouco controle sobre os eventos, e o foco se encontra em fenômenos contemporâneos inseridos em algum contexto da vida real, onde se deseja saber “como”

e “por que”. O estudo de caso será realizado com um grupo de profissionais de uma empresa do setor de TI, com o objetivo de avaliar a aderência da prática *Agile Testing*.

3.2. Definição da População

A população alvo do estudo de caso será um time de desenvolvimento de uma empresa de TI na qual a autora da pesquisa atua como Analista de Testes. Este grupo de pessoas será composto por aproximadamente 30 profissionais, sendo eles, Analistas de Testes, Analista de Sistemas e Desenvolvedores.

A escolha desta população alvo se deu pela técnica de amostragem por conveniência. Nesta técnica, o pesquisador seleciona os elementos dos quais tem acesso, admitindo que estes possam representar a população de alguma forma. A grande vantagem da amostragem por conveniência ou acessibilidade é o seu baixo custo e rapidez [Valentim 2005].

3.3. Técnica de Coleta de Dados

A coleta de dados será feita através de revisão de artefatos no repositório e observação do processo de desenvolvimento.

3.4. Técnica de Análise de Dados

A partir dos dados coletados, será realizada uma análise quantitativa baseada nos graus de implementação definidos pelo Guia de Avaliação do MPS-BR. O mesmo, será utilizado porque possui uma estrutura que irá auxiliar no cálculo de aderência [Softex 2013].

3.5. Limitações do Método de Estudo

O estudo de caso será limitado a apenas a aplicação do *Agile Testing* em projetos que utilizam *Scrum* e a uma população alvo de pequeno porte, devido à limitação de tempo para a realização deste trabalho. Sendo assim, as conclusões obtidas neste estudo não poderão ser generalizadas, e apenas contribuirão para direcionar estudos posteriores.

3.6. Etapas a Serem Desenvolvidas

As etapas que serão desenvolvidas neste artigo são:

1. Caracterização da amostra onde será realizado o estudo de caso;
2. Definição dos repositórios avaliados;
3. Mapeamento do processo de desenvolvimento
4. Mapeamento de testes realizados em cada etapa do processo de desenvolvimento;
5. Análise de testes realizados em cada quadrante do *Agile Testing*.
6. Análise crítica do resultado obtido.

4. Aderência de práticas do *Agile Testing* em projetos que utilizam *Scrum*

Segundo [Crispin e Gregory 2009], utilizar uma Metodologia Ágil não vai garantir por completo a qualidade de um projeto, mas se a Metodologia Ágil for utilizada sendo combinadas com práticas do *Agile Testing*, suas características serão completadas a fim de garantir que os testes sejam feitos na proporção e momento corretos.

Com base no referencial teórico deste trabalho, *Agile Testing* será considerado um modelo de referência de testes em times que utilizam Metodologias Ágeis, e com base neste modelo, será avaliado o processo de desenvolvimento da empresa objeto de

estudo, a fim de verificar o quão aderente a empresa está em relação as práticas sugeridas pelo modelo *Agile Testing*.

4.1. Caracterização da amostra

Este trabalho foi realizado em uma empresa do setor de tecnologia da informação que desenvolve soluções para toda a cadeia de suprimentos, possui milhares de empresas conectadas à sua malha em todo o mundo. É a única do segmento de software que oferece soluções de ponta a ponta, desde a produção até o consumidor final, passando pela distribuição e logística. A empresa possui mais de 100 varejos homologados, entre eles, os 10 maiores do Brasil e do mundo. A empresa em questão sincroniza indústrias, distribuidores, operadores logísticos, varejos, instituições financeiras e Governo.

O estudo de caso será aplicado em um time de desenvolvimento, que desenvolve e matem soluções customizadas para clientes da empresa em questão, composto por:

- 01 coordenador;
- 01 facilitador;
- 02 analistas de negócio;
- 03 gerentes de projeto;
- 06 analistas de sistemas;
- 07 analistas de testes;
- 15 desenvolvedores.

O time de desenvolvimento objeto de estudo utiliza como metodologia de desenvolvimento o Scrum. Cada time *Scrum*, chamado de célula, é dedicado a projetos específicos, independentes entre si. Cada célula possui um Analista de Sistemas, um ou mais Analista de Testes e um ou mais Desenvolvedores. Para este estudo de caso, será avaliado um projeto em cada célula, totalizando em seis projetos.

4.2. Repositórios avaliados

Nesta sessão, serão listadas as ferramentas e *frameworks* utilizados pelo time de desenvolvimento que possuem relação com atividades de testes. As informações destes repositórios irão basear o estudo de caso.

4.2.1. *Confluence*

Confluence é uma ferramenta colaborativa que tem como objetivo facilitar a comunicação e documentação de um projeto. Tal ferramenta possibilita a documentação de funcionalidades e testes.

4.2.2. JIRA

JIRA é uma ferramenta para planejar, rastrear e acompanhar o trabalho que está sendo realizado em cada projeto, e que permite a extração de indicadores e métricas. O JIRA pode ser utilizado para organizar *Sprints* e releases, registrar atividades, não conformidades, melhorias, tarefas, atribuir trabalhos, estimar e registrar o tempo já utilizado. No JIRA, uma atividade é chamada de *Issue*, e na empresa objeto deste estudo, ela pode ser categorizada como Melhoria, Nova Funcionalidade, Tarefa, Não Conformidade, entre outros.

4.2.3. SVN

O *Subversion*, ou simplesmente SVN, é uma ferramenta de controle de versão que permite, além do desenvolvimento colaborativo a partir de um repositório único, merge de conteúdo, armazenamento de logs e geração de estatísticas diversas. Atuando como a máquina do tempo do desenvolvedor, ferramentas com o SVN permitem retornar o código a um estado anterior, facilitando a análise das implementações realizadas e a mesclarem com implementações distintas de períodos diferentes para a criação de uma única versão.

4.2.4. Bamboo

Bamboo é um servidor de integração contínua (CI) que pode ser usado para automatizar a liberação de um *software*, executar testes automatizados e realizar publicações em diversos ambientes.

4.3. Processo de desenvolvimento

O Time Desenvolvimento PE possui autonomia para assumir novos projetos ou novas customizações nos projetos existentes, isso porque possui profissionais que atuam desde a área comercial até a homologação junto ao cliente final. Por esta característica, o fluxo de trabalho deste time é um pouco diversificado do fluxo padrão da empresa.

Para que seja possível avaliar a aderência da empresa objeto desde estudo em relação à prática do *Agile Testing*, foi necessário mapear as atividades do processo de desenvolvimento. Para isso, utilizou-se como técnica de coleta de dados, análise nos repositórios já mencionados e observação.

O processo de desenvolvimento pode ser iniciado, através de solicitações do cliente em forma de: melhorias, novas funcionalidades ou não conformidades, identificadas em ambiente de produção. A figura 8 representa o processo de desenvolvimento do time objeto de estudo quando o cliente solicita uma melhoria ou nova funcionalidade. Neste caso, o resultado final do processo de desenvolvimento pode ser uma versão entregue ao cliente.

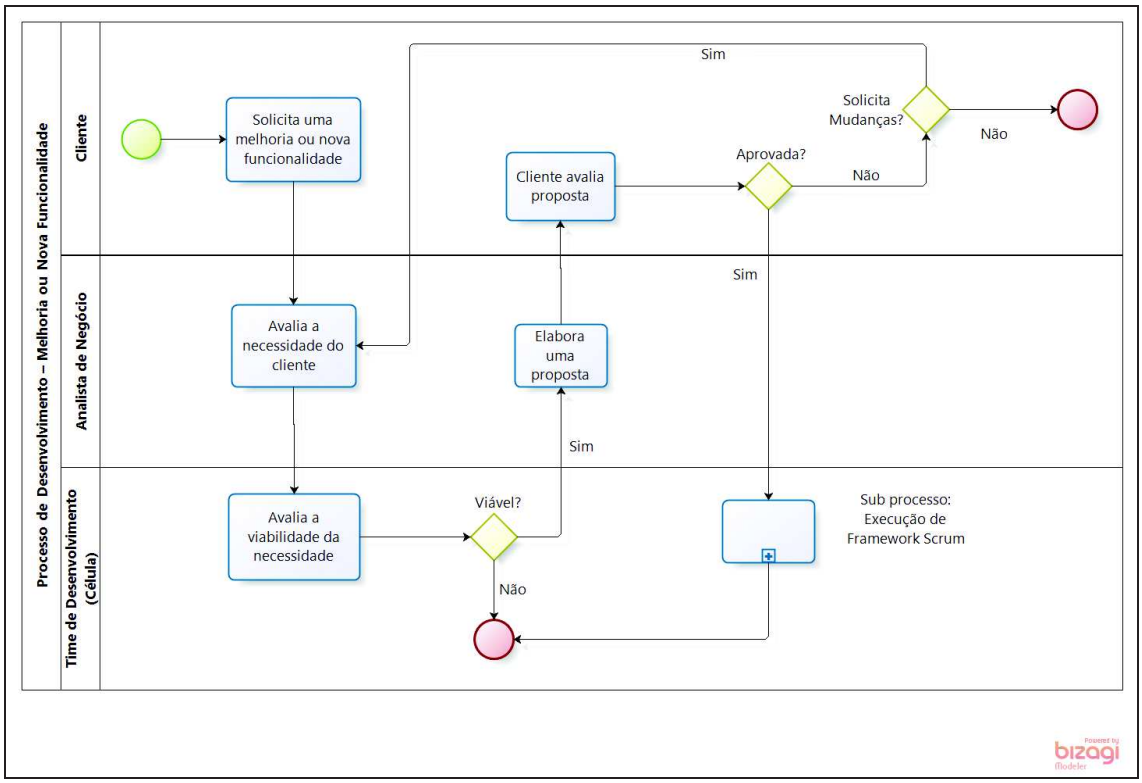

Figura 8: Processo de Desenvolvimento – Melhoria ou Nova Funcionalidade

Fonte: Elaborado pelo autor

Na empresa em questão, toda necessidade do cliente é avaliada por Analistas de Negócios e pelo time de desenvolvimento. A reponsabilidade do Analista de Negócios é entender a necessidade do cliente, para fornecer detalhes ao Time de Desenvolvimento (Célula). O time, por sua vez, avaliará se é possível atender à necessidade, e se as alterações que devem ser realizadas serão viáveis para o projeto. Cabe ao time neste momento, sugerir soluções que atendam a necessidade do cliente e que acrescentem valor ao projeto. Para entender o escopo, o time de desenvolvimento PE levanta cenários e exemplos que ajudam a identificar uma solução, e validá-la posteriormente. Todos os membros do time podem sugerir melhorias em uma solução.

Após o time desenhar uma solução para a necessidade do cliente, cabe ao Analista de Negócio, elaborar uma proposta contendo a explicação da solução, cenários e exemplos que o time levantou. Essa proposta é enviada ao cliente para que o mesmo possa avaliá-la, e aprová-la, caso esteja de acordo com suas expectativas. Com a proposta assinada, inicia-se a execução do "Subprocesso: Execução do *Framework Scrum*", que será detalhado a seguir, na Figura 9.

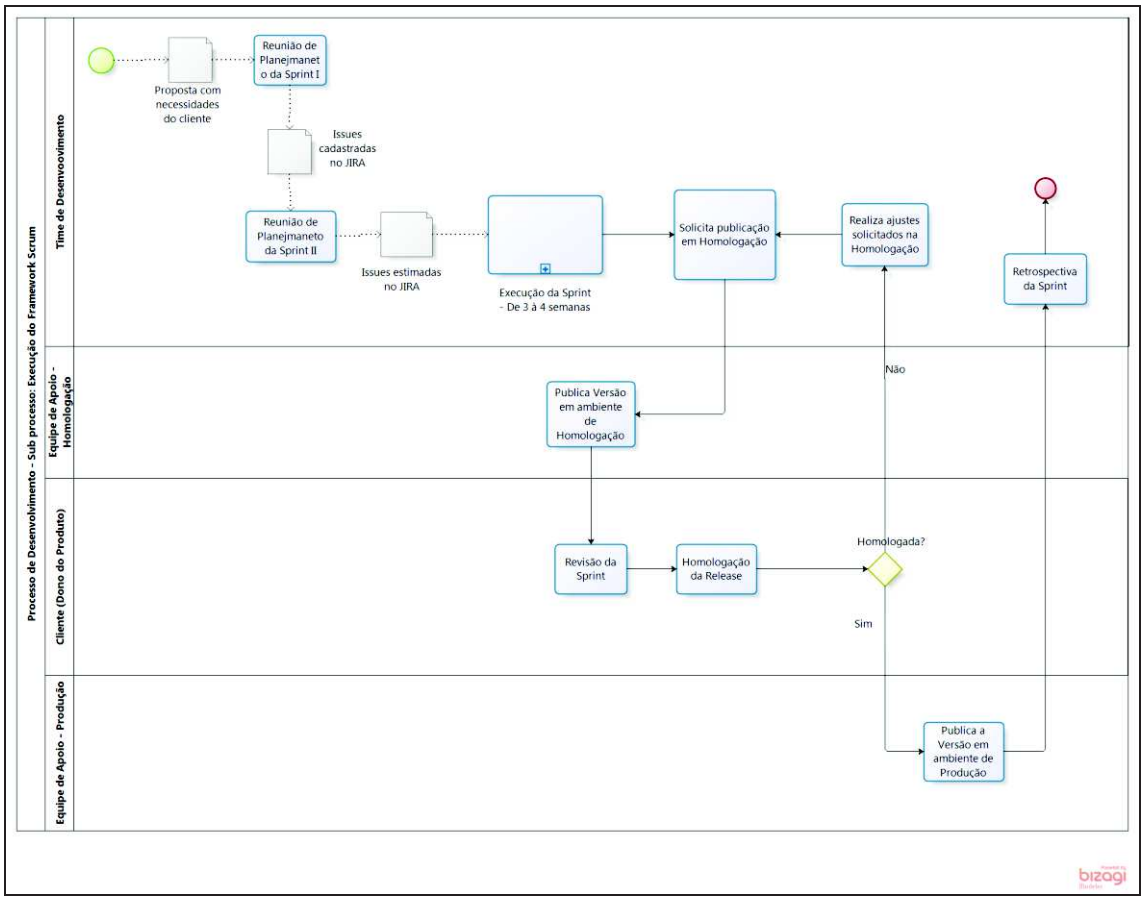

Figura 9: Subprocesso: Execução do *Framework Scrum*

Fonte: Elaborado pelo autor

Conforme pode ser observado na imagem acima, o artefato de entrada para execução do *Scrum* é uma proposta, cuja constituição está no Backlog do Produto, naquele momento. A primeira atividade realizada pelo time é a Reunião de Planejamento da *Sprint* I, onde são discutidas as necessidades do cliente, e a solução contida na proposta. Para que todos os membros do time tenham o mesmo entendimento, são levantados cenários e exemplos. Nesta cerimônia, o time define quais atividades do Backlog do Produto que serão desenvolvidas na *Sprint* que se iniciará. A Reunião de Planejamento da *Sprint* II é onde o time discute a melhoria ou nova funcionalidade, quebrando-a em *Issues*, adiciona critérios de aceitação e realiza uma estimativa do esforço que será necessário.

No sub - processo "Execução da *Sprint*" são realizadas as atividades de Análise, Desenvolvimento, Testes e Documentação. Na prática, o time realiza a análise as funcionalidades tecnicamente, avaliando que regras negócio serão adicionadas e alteradas e, o impacto delas nas demais funcionalidades. O time codifica as funcionalidades e os testes unitários. E o time realiza teses funcionais e documenta as regras de negocio adicionadas e alteradas. A Figura 10, irá detalhar estas atividades.

O mapeamento do processo de uma *Sprint* considera como entrada uma ou mais *issues* no JIRA, que na empresa em questão, são descritas na forma de História de

Usuário. Conforme o referencial teórico deste trabalho, uma história pode conter uma ou mais regras de negócio e pode conter um ou mais cenários.

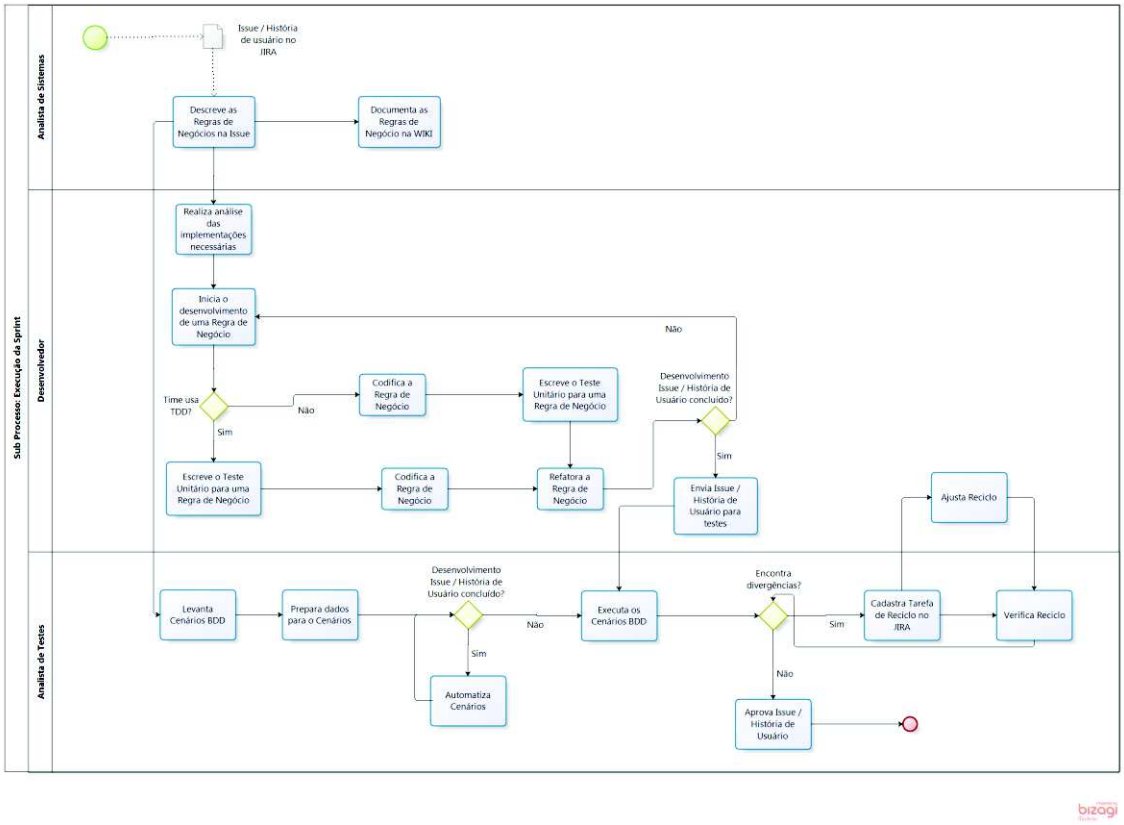

Figura 10: Sub Processo *Sprint*

Fonte: Elaborado pelo autor

Um fluxo de trabalho pode ser iniciado também através de uma Não Conformidade identificada em ambiente de produção. Caso a Não Conformidade não gere grandes impactos ao cliente, ela poderá ser resolvida na próxima versão planejada, então entraria normalmente no Processo representado pela Figura 9. Por outro lado, caso a Não Conformidade seja crítica e esteja prejudicando as operações do cliente, será gerada uma versão emergencial, denominada de *Patch.* Para um *Patch*, o processo de desenvolvimento é diferenciado, conforme apresentado na Figura 10.

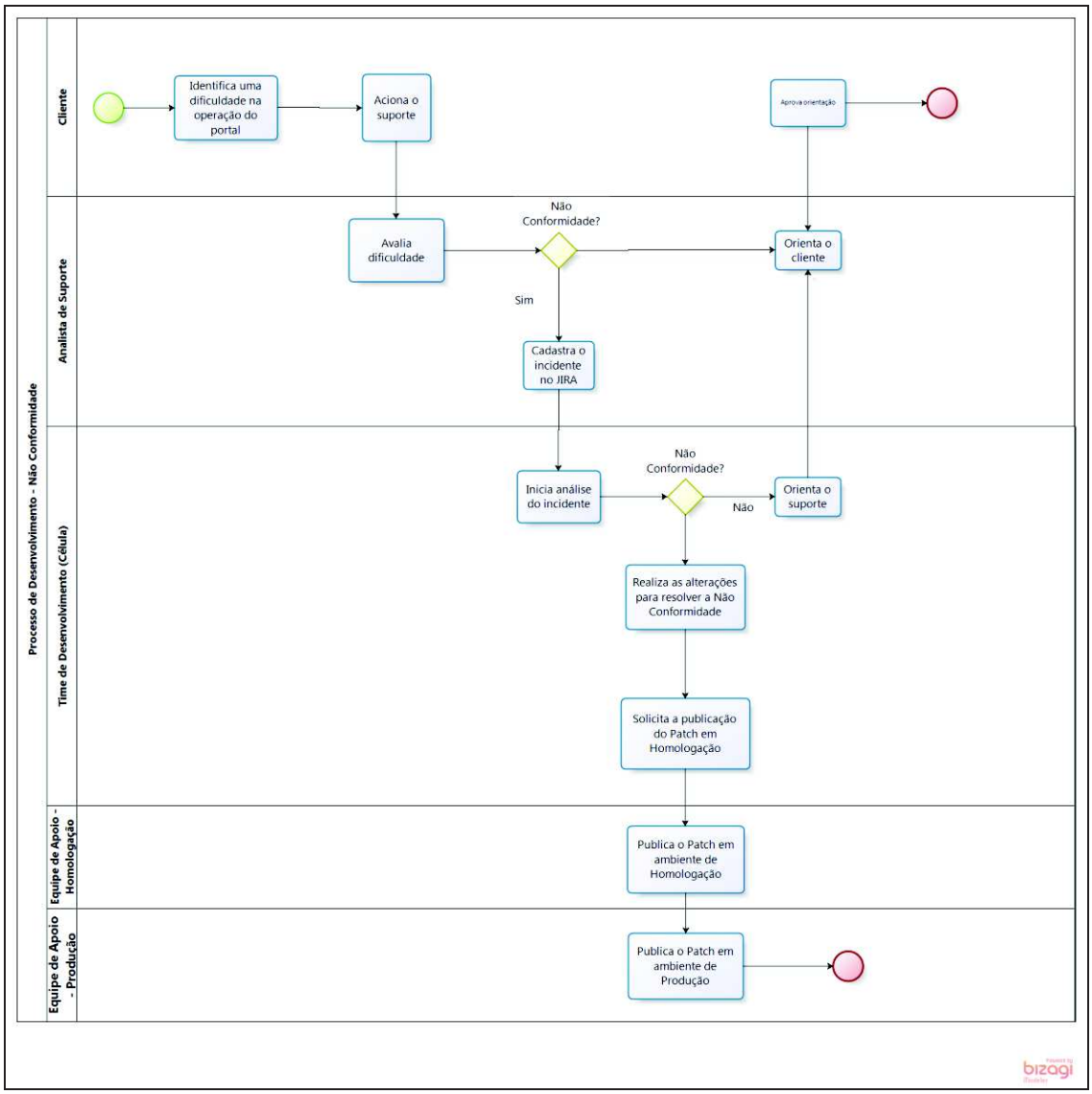

Figura 11: Processo de Desenvolvimento – Não Conformidade

Fonte: Elaborado pelo autor

O processo para geração de um *Patch* é uma exceção. Ocorre somente quando o cliente identifica não conformidades de grande impacto. Se o time estiver executando uma *Sprint*, e o cliente identificar um problema que necessite de um *Patch*, a *Sprint* será parada e o time dará foco total no *Patch*.

4.4. Análise de testes realizados em cada etapa de desenvolvimento por quadrante

Para que seja possível avaliar a aderência da empresa objeto deste estudo em relação à prática *Agile Testing*, serão analisados alguns repositórios a fim de identificar em que etapa do processo de desenvolvimento são realizados os testes agrupados por Quadrante. Os repositórios que serão avaliados são:

- Confluence:

- Serão avaliados os documentos de proposta a fim de verificar se são levantados exemplos e cenários com o cliente;
- Serão avaliados os documentos de testes a fim de verificar se contemplam historias de usuários e cenários;

- JIRA: Serão verificadas se as Melhorias e Novas Funcionalidades possuem Histórias de Usuários e Critério de Aceitação;
- SVN: Serão verificados os testes automatizados que são desenvolvidos;
- BAMBOO: Será verificado se os testes automatizados (testes unitários e funcionais) são executados na Integração Contínua.

A partir da análise nos repositórios listados acima e da observação realizada, foi possível identificar no processo de desenvolvimento de Melhoria e Nova Funcionalidade, em que etapa são realizados testes de cada quadrante. A figura 12 demostra o resultado da análise realizada.

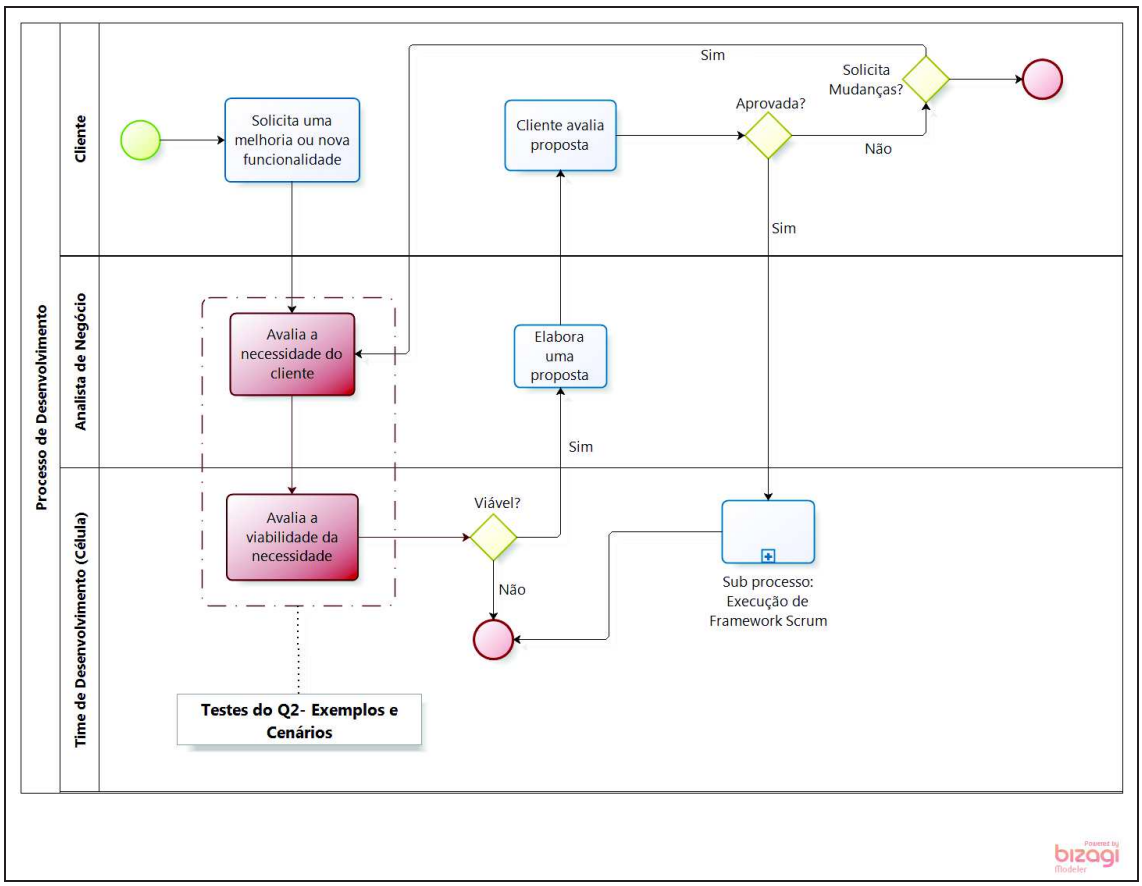

Figura 12: Processo de Desenvolvimento – Melhoria ou Nova Funcionalidade – Análise de testes realizados em cada etapa por quadrante.

Fonte: Elaborado pelo autor

De acordo com a Figura 12, na etapa onde o Analista de Negócios avalia a necessidade do cliente, ele levanta exemplos de situações reais do mesmo para entender a dificuldade e o problema que será resolvido. Esta atividade está de acordo com a definição da prática de **Exemplos**, definida pelo [ISTQB 2014], no referencial teórico

deste trabalho, onde dados fornecidos pelo cliente que podem servir de entrada e saída para um teste. O teste deste estudo de caso é de validação de uma dificuldade.

Na etapa onde o time de desenvolvimento (célula) avalia a viabilidade da necessidade do cliente são escritas **histórias de usuário,** elas possuem a narrativa respeitando o **padrão definido** no referencial teórico deste trabalho. A Figura 13 apresenta uma história de usuário escrita na empresa objeto deste estudo. As informações de <Empresa> e < Aplicação> não podem ser expostas devido à política de segurança de informação da empresa em questão.

```
Narrativa:
Como um usuário fornecedor da empresa <Empresa>
Desejo acessar a aplicação <Aplicação> e visualizar o relatório de Contas a Receber
Para obter o valor exato que irei receber da empresa <Empresa> a qual sou fornecedor

Cenário: Verificar visualização do relatório de Contas a Receber
Dado que ao clicar no menu Relatórios > Contas a Receber
Quando acessar a página Relatórios Contas a Receber
Então irei preencher o campo data inicial com o valor 01/06/2015
Então irei preencher o campo data final com o valor 30/06/2015
Então irei clicar no botão com descrição Pesquisar
Então serão exibidos os resultados da pesquisa
```

Figura 13: Narrativa de História de Usuário escrita com cenário

Na avaliação da viabilidade de necessidade do cliente, são também descritos cenários **BDD** para a Narrativa que está sendo avaliada, conforme Figura 13. Note que os testes do Q2 são aplicados antes de se ter uma proposta assinada com o cliente, isso porque a empresa acredita que levantar cenários é a melhor forma de validar o entendimento e a viabilidade de uma necessidade do cliente. Muitas vezes, os cenários de teste são levantados, a proposta é enviada ao cliente – incluindo os cenários – e, o mesmo identifica outras necessidades justamente avaliando os cenários que foram enviados.

Para que seja possível avaliar a aderência da empresa em questão em relação à prática *Agile Testing*, foram avaliados, também, os testes realizados durante a Execução do Framework *Scrum*. O resultado será exibido na Figura 14:

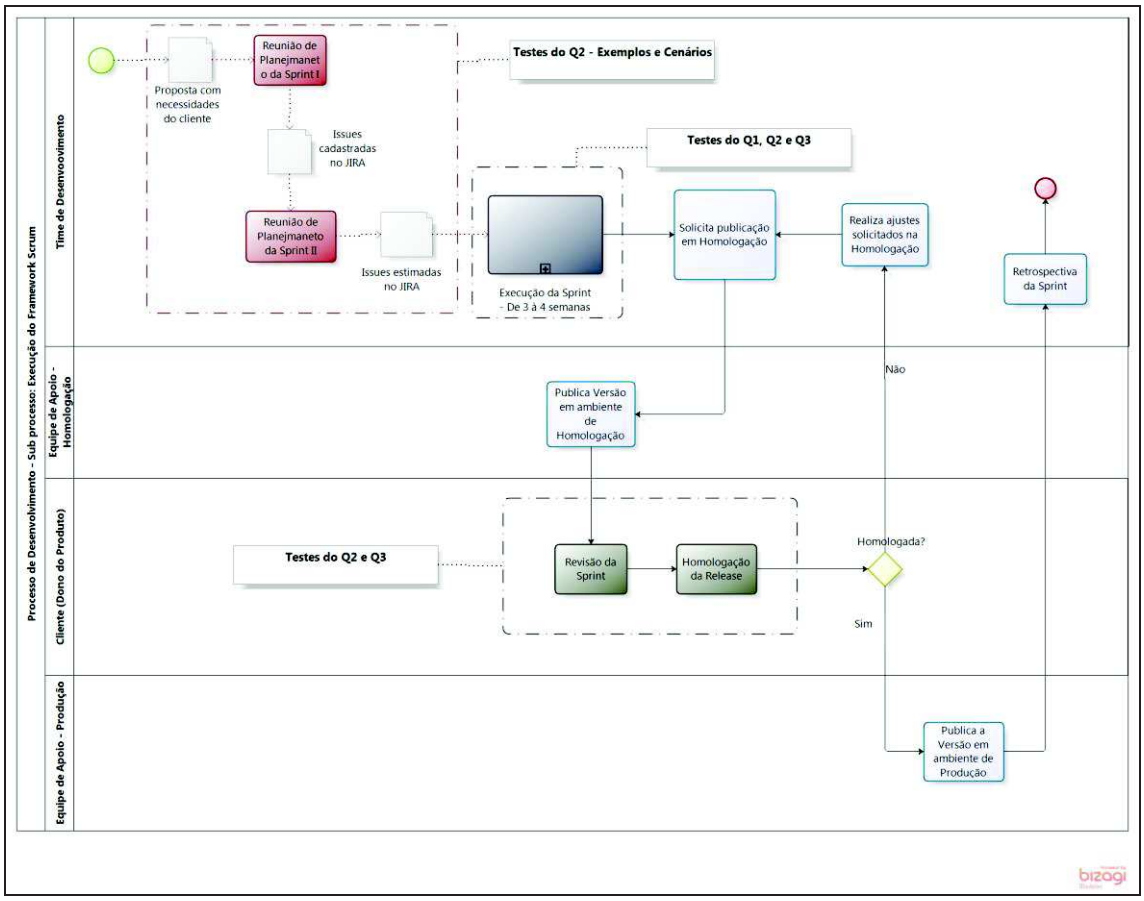

Figura 14: Execução do *Framework Scrum* – Análise de testes realizados em cada etapa por quadrante.

Fonte: Elaborado pelo autor

Na **Reunião de Planejamento da *Sprint* I e II**, o time discute as necessidades do cliente detalhadamente, analisando-as na perspectiva de negócio e de tecnologia. Nesta etapa, muitos **cenários de teste** são escritos com base nos **exemplos** que foram levantados na fase de proposta, para que todo time tenha entendimento do problema do cliente e da solução que será desenvolvida. Estes cenários de teste, não são cenários BDD, mas sim cenários de teste tradicionais, que visam testar o sistema na perspectiva de um usuário final. Portando, estas etapas contemplam testes do Q2 e Q3.

Na **Revisão da *Sprint***, os **exemplos** e **cenários de teste BDD** levantados para elaboração da proposta, são testados juntamente com o PO. Além disto, o PO poderá também homologar a versão, desta forma, fará seus testes de **aceitação** (que na empresa em questão incluem testes de Alpha / Beta) e de **usabilidade**, para que ele possa aprovar ou não a versão. Portanto, pode-se concluir que nesta cerimonia do *Scrum*, são realizados testes do Q2 e Q3.

De acordo com a análise dos dados coletados, a execução da *Sprint*, apresentada pela Figura 15, contempla testes do Q1, Q2 e Q3, demonstrando em que etapa do processo de desenvolvimento são executados testes de cada quadrante.

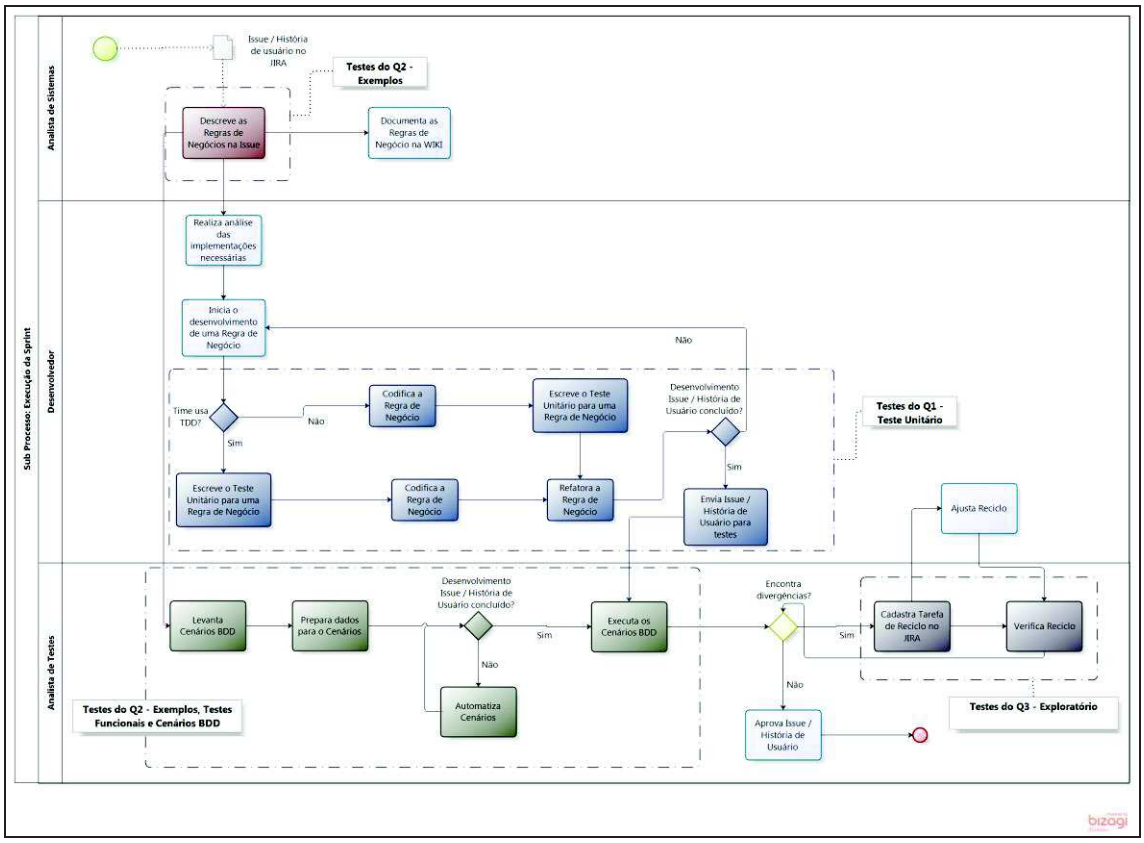

Figura 15: *Sprint* – Análise de testes realizados em cada etapa por quadrante.

Fonte: Elaborado pelo autor

Na atividade onde o Analista de Sistemas descreve as Regras de Negócio na Issue, ele utiliza os **exemplos** de situações reais do cliente, levantados nas etapas anteriores. Este tipo de teste está categorizado no Q2. Nesta atividade, os exemplos são utilizados como critério de entrada e saída para validar uma regra. Com exemplos fornecidos pelo cliente é possível saber se com determinados dados de entrada, aplicando determinada regra, a saída será conforme o desejado.

A próxima atividade que envolve testes está relacionada com o desenvolvimento. Durante essas atividades, o desenvolvedor escreve testes **unitários,** que estão categorizados no Q1, no momento que realizara a codificação das regras de negócios. Um ponto de atenção é que somente os testes unitários possuem cobertura medida, e que a prática TDD é opcional no desenvolvimento. Ou seja, algumas Células desenvolvem o teste unitário utilizando TDD e outras, escrevem os testes após a implementação dos métodos.

O Analista de Testes por sua vez, executa **Testes funcionais (Q2)**, mapeando **Cenários (Q3)** para verificar as funcionalidades de um sistema, subsistema ou componente, e para fazer testes **Exploratórios**. Para isso, os **exemplos (Q2)**, utilizados nos processos anteriores, são critérios de entrada e saída para um teste. O Analista de Testes também executa as **Histórias de Usuário (Q2)** e os **Cenários BDD (Q2)** que foram mapeados nna avaliação da viabilidade da necessidade do cliente, demonstrado anteriormente na Figura 12.

Eventualmente, o Analista de Teste faz testes de **performance** nos webservices das aplicações com o objetivo de verificação como a aplicação se comporta com grandes volumes de dados, mas não é mantido nenhum histórico deste teste e, os mesmos não são realizados em versões futuras. Sendo assim, esta prática não será considerada como aderente porque, segundo o referencial teórico deste trabalho, os testes de performance geralmente são utilizados para identificar os pontos de gargalo de software e devem servir para estabelecer uma linha de base para testes futuros. Desta forma, o time onde foi realizado o estudo de caso não realiza nenhum teste categorizado no Q4.

Além dos processos analisados acima, existe ainda o processo de geração de um patch contendo uma Não Conformidade relatada pelo cliente. Os testes realizados por quadrante serão neste processo serão demonstrados na Figura 15.

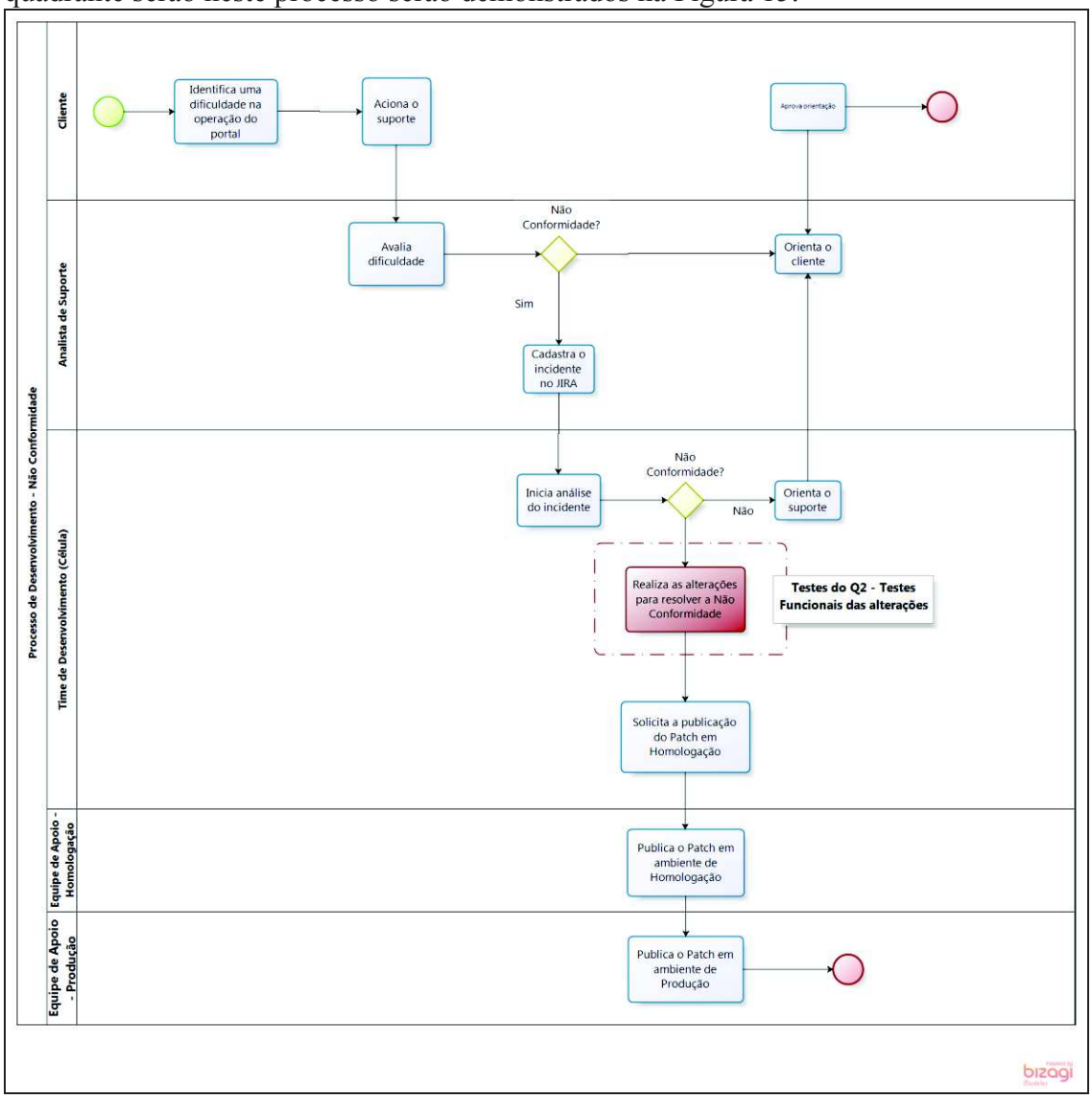

Figura 16: Processo de Desenvolvimento de Não Conformidade – Análise de testes realizados em cada etapa por quadrante.

Fonte: Elaborado pelo autor

Conforme pode ser observado na Figura 15, são realizados apenas **Testes Funcionais (Q2)** da funcionalidade que teve alterações. Conforme o referencial teórico deste trabalho, uma premissa para ter-se um software com qualidade é que se possua uma suíte de regressão automatizada, porém, conforme as Figuras 14 e 15 fica evidente que esta prática não é atendida na empresa em questão.

Baseando-se no levantamento realizado, é possível identificar que técnicas e tipos de testes, categorizados em cada quadrante, são realizados na empresa em que foi realizado o estudo de caso. Com base neste levantamento, foi montada a Tabela 1.

Quadrante	Tipo / Técnica de Testes	Forma de Execução
Q1	Testes Unitários	Automatizado
Q2	Testes Funcionais	Manual e Automatizado
	Exemplos	Manual
	Histórias de Usuários	Manual e Automatizado
Q3	Testes Exploratórios	Manual
	Cenários	Manual
	Usabilidade	Manual
	Testes de Aceitação	Manual

Tabela 1: Testes realizados pelo time de desenvolvimento por quadrante

Fonte: Elaborado pelo autor.

4.5. Análise crítica do resultado

Agile Testing é um conjunto de práticas de testes, já conhecidas pelo mercado, aplicadas de forma que possam se complementar ao máximo. Estas práticas podem ser utilizadas por profissionais de teste e desenvolvedores, e visam atender ao manifesto ágil, aplicando técnicas de testes de software que possibilitam a redução no tempo de execução de testes e ainda podem aumentar a cobertura de cenários e possibilidades.

Para realização do estudo de caso, assumiu-se que *Agile Testing* é um modelo que pode ser seguido e, como referência para sua aplicação existe a definição dos Quadrantes do *Agile Testing*, onde cada quadrante agrupa algumas técnicas que são indispensáveis para que o time possua um tempo de resposta praticamente instantâneo, que possibilite que o impacto de cada alteração seja percebido assim que a mesma for realizada. Os Quadrantes do *Agile Testing* definem:

1. Técnicas de testes que devem ser aplicadas;
2. A forma de execução de cada técnica de teste, ou seja, se é executada de forma manual, automatizada ou se executada com auxílio de ferramentas.

Baseando-se na sessão "2.6.3. Quadrantes do *Agile Testing*" é possível avaliar o quanto aderente é a empresa objeto do estudo de caso em relação ao *Agile Testing*. Para isso, serão utilizados os graus de implementação definidos pelo Guia de Avaliação do MPS-BR, que estão sendo listados na Tabela 2.

Grau de implementação	Caracterização
Totalmente implementado (T)	- O indicador direto está presente e é julgado adequado - Existe pelo menos uma afirmação confirmando a implementação - Não foi notado nenhum ponto fraco substancial na avaliação inicial ou na avaliação final.
Largamente implementado (L)	- O indicador direto está presente e é julgado adequado - Existe pelo menos uma afirmação confirmando a implementação - Foi notado um ou mais pontos fracos substanciais na avaliação inicial ou na avaliação final.
Parcialmente implementado (P)	- O indicador direto não está presente ou é julgado inadequado - Artefatos/afirmações sugerem que alguns aspectos do resultado esperado estão implementados - Foi notado um ou mais pontos fracos substanciais.
Não implementado (N)	- Qualquer situação diferente das acima
Não avaliado (NA)	- O projeto/trabalho não está na fase de desenvolvimento que permite atender ao resultado ou não faz parte do escopo do projeto atender ao resultado.
Fora do escopo (F)	- O resultado esperado está fora do escopo da avaliação, conforme documentado no plano da avaliação.

Tabela 2: Critérios de Avaliação

Fonte: [Softex 2013]

Para medir o percentual de adesão da empresa objeto deste estudo de caso em relação à prática *Agile Testing*, foi considerado que cada tipo ou técnica de testes tem o mesmo peso em relação aos demais. Isso porque [Crispin e Gregory 2009] não apresentam uma definição de quais testes devem ser tratados com maior ênfase. Logo, dividiu-se 100 (nota máxima, que indica que a empresa está completamente aderente) pelo total número de Tipo de Testes, que é 15. Logo, atribuiu-se um peso de 6,66 para cada Tipo de Testes.

Então, considerou-se que quando um tipo de teste for avaliado como:

- ✓ T – Representa 100% do peso de um Tipo / Técnica de teste, portanto sua pontuação é 6,66;
- ✓ L – Representa 66,66% do peso de um Tipo / Técnica de teste, portanto sua pontuação é 4,44;
- ✓ P – Representa 33,33% do peso de um Tipo / Técnica de teste, portanto sua pontuação é 2,22;
- ✓ N – Representa 0% do peso de um Tipo / Técnica de teste, portanto sua pontuação é sua pontuação é 0.

Utilizando a Figura 5: Quadrantes do *Agile Testing* (apresentada na sessão 2.6.3) como modelo de aplicação de testes e os Graus de Implementação definidos por [Softex

2013], pode-se realizar um comparativo com a Tabela 1 (acima), a fim de identificar se a empresa em questão está ou não aderente à prática *Agile Testing*.

Quadrante	Tipo / Técnica de Testes	Forma de Execução (Esperada)	Grau de Implementação	Pontuação	Justificativa
Q1	Testes Unitários	Automatizado	T	6,66	Todos projetos avaliados possuem testes unitário sendo executado no Bamboo.
	Testes de Componentes	Automatizado	N	0	Nenhum projeto avaliado realiza testes de componente.
Q2	Testes Funcionais	Automatizado e Manual	L	4,44	Todas as células avaliadas possuem testes funcionais sendo executados de forma manual porém a minoria executa estes testes de forma automatizada.
	Exemplos	Automatizado e Manual	P	2,22	Todas as células avaliadas possuem testes de exemplos sendo executados de forma manual porém nenhuma executa estes testes de forma automatizada.
	Histórias de Usuários	Automatizado e Manual	L	4,44	Todas as células avaliadas possuem testes de histórias de usuário sendo executados de forma manual porém a minoria executa estes testes de forma automatizada.
	Protótipos	Automatizado e Manual	N	0	Nenhuma célula avaliada realiza testes de protótipos.
	Simulações	Automatizado e Manual	N	0	Nenhuma célula avaliada realiza testes de simulações.
Q3	Testes Exploratórios	Manual	T	6,66	Todas as células avaliadas possuem testes de exploratórios sendo realizados.
	Cenários	Manual	T	6,66	Todas as células avaliadas possuem testes de cenários sendo realizados.
	Testes de Usabilidade	Manual	T	6,66	Todas as células avaliadas possuem testes de usabilidade sendo realizados.
	Testes de Aceitação	Manual	T	6,66	Todas as células avaliadas possuem testes de aceitação sendo realizados.
	Alpha / Beta	Manual	T	6,66	Todas as células avaliadas possuem testes Alpha / Beta sendo realizados.
Q4	Performance e Carga	Ferramentas	N	0	Nenhuma célula avaliada realiza teste de performance e carga de acordo com o previsto pelo modelo Agile Testing
	Segurança	Ferramentas	N	0	Nenhuma célula avaliada realiza testes de segurança.
	Testes de "ilidade"	Ferramentas	N	0	Nenhuma célula avaliada realiza testes categorizados como "ilidade"

Tabela 3: Resultado da avaliação de aderência do *Agile Testing*.

Fonte: Elaborado pelo autor

Com base na avaliação realizada, o time de desenvolvimento da empresa avaliada neste estudo apresentou o seguinte resultado de aderência:

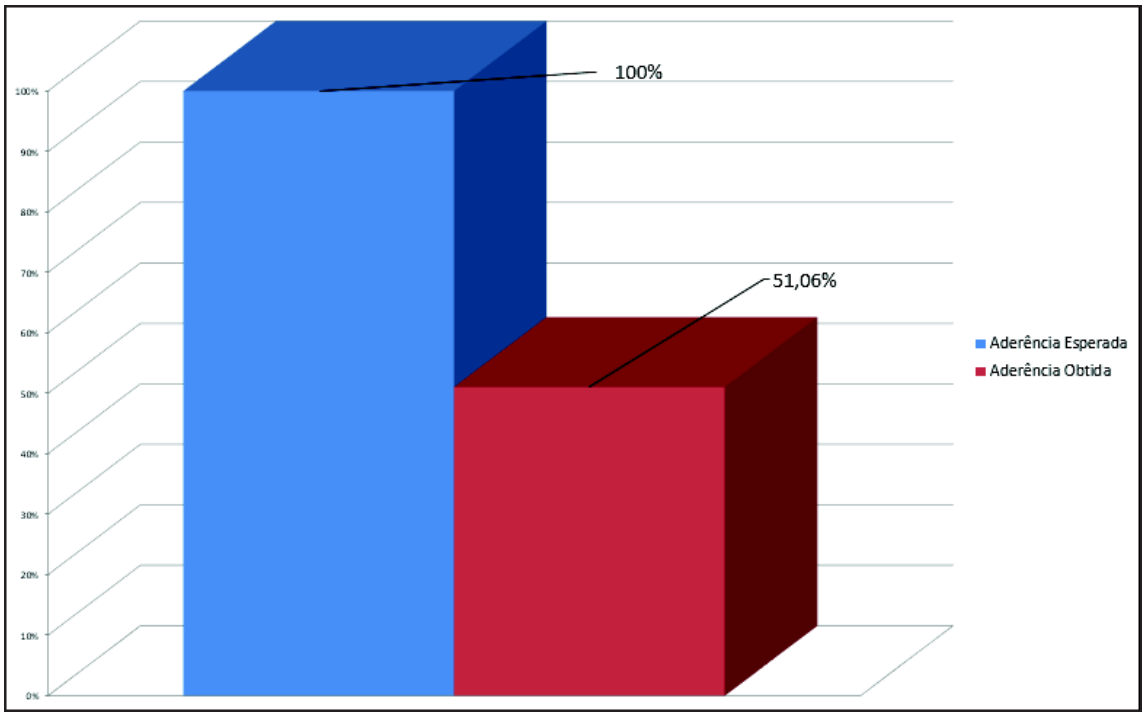

Figura 17: Aderência obtida em relação ao modelo *Agile Testing*

Fonte: Elaborado pelo autor

A Figura 18 apresenta os resultados da avaliação sendo categorizados por Quadrante do *Agile Testing*, assim é possível realizar uma análise crítica contemplando detalhes de cada quadrante.

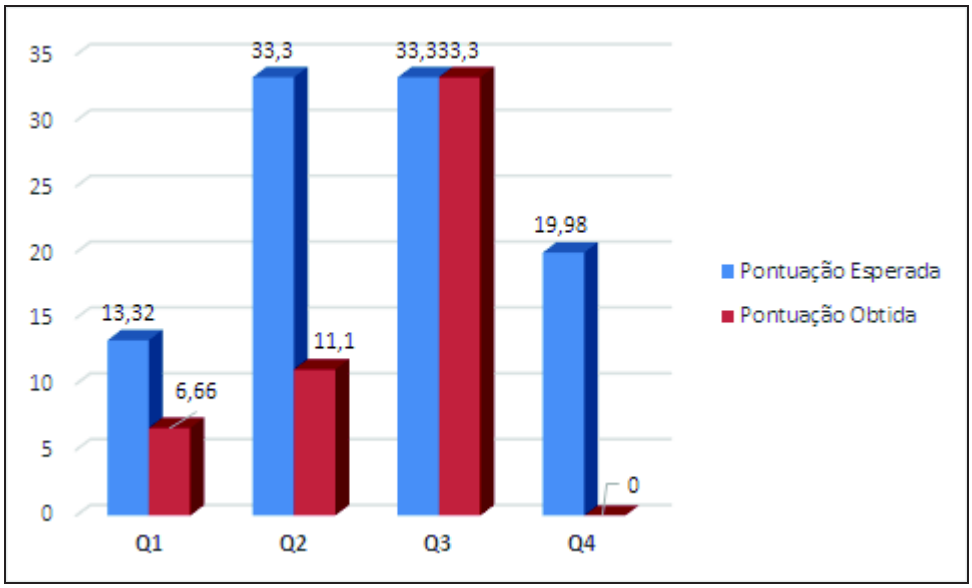

Figura 18: Resultado da aderência por quadrante do *Agile Testing*.

Fonte: Elaborado pelo autor

Analisando os resultados do levantamento realizado, é possível identificar que a empresa objeto do estudo de caso poderia explorar testes de componentes, categorizados no Q1, a fim de verificar os módulos de suas aplicações. Estes complementariam os testes unitários, que já são realizados, e poderiam aumentar a cobertura de código, aumentando o feedback e a segurança do time em relação às alterações. Outro ponto que pode ser considerado é que, segundo os Princípios do *Agile Testing* disponíveis na sessão 2.6.4, o time deve sempre que possível adotar a técnica TDD. E, de acordo com a Figura 15, disponível na sessão 4.4, a adoção do TDD é opcional.

Além disso, notou-se que os testes Funcionais e Histórias de Usuários do Q2 são executados de forma automatizada na minoria das células avaliadas. E, os testes de exemplo são executados somente de forma manual. Hoje, é total interesse da empresa que estes testes sejam executados cada vez mais de forma automatizada, sendo que a empresa possui um time interno dedicado à facilitação da automação destes tipos de Testes. Isso porque a empresa deseja obter os benefícios de ter um *feedback* instantâneo, aumentando assim a produtividade de seus times e confiança no trabalho desenvolvido.

Ainda em relação aos testes do Q2, os testes de protótipos e simulações não são realizados. Estes testes geralmente são mais próximos possíveis do uso do cliente, já que possuem como entrada ou saída dados fornecidos pelo cliente. Ainda, tais testes podem contemplar dados do ambiente de produção reproduzindo o comportamente do usuário final. Estes testes garantiriam os cenários básicos para serem validados, e agregariam na qualidade final dos projetos da empresa em questão.

Observou-se também que a empresa executa conforme o previsto todos os testes do Q3. Porém, conforme definição de [Crispin e Gregory 2009], disponível na sessão 2.6.3.3, os testes do Q3 são suficientes para produzir software de alta qualidade somente quando os testes do Q1 e Q2 estão automatizados. E, como foi apresentado, esta não é a realidade da empresa. Como resultado negativo da não automatizados de testes do Q1 e principalmente Q2, os Analistas de Testes usam o tempo que teriam para explorar cenários complexos para realizar testes simples, que verificam apenas o "caminho feliz".

Referindo-se agora aos testes do Q4, a empesa reconhece que precisa investir na execução de testes de performance e carga, isso porque houve diversas situações onde estes testes se mostraram determinantes para a qualidade no ponto de vista do cliente. Também existem iniciativas dentro da empresa para fomentar testes de Segurança. Isso indica que se a avaliação de aderência fosse realizada periodicamente, a empresa em questão poderia estar melhorando seu percentual de aderência.

5. Conclusão

Este trabalho teve como objetivo principal realizar uma avaliação de aderência de práticas do *Agile Testing* em um time que utiliza *Scrum*. Para que isso fosse possível, foi feito um estudo aprofundado sobre metodologias ágeis, com foco no *Scrum*, e sobre a prática *Agile Testing*. Com o embasamento teórico, foi possível realizar a avaliação de aderência de um time de desenvolvimento de uma empresa do setor de TI e elaborar uma análise crítica com base nos resultados obtidos.

Sendo assim, a partir deste trabalho, podemos concluir que o time de desenvolvimento da empresa avaliada não está completamente aderente às práticas sugeridas pelo modelo *Agile Testing*. Conforme as informações apresentadas, é possível

observar que existem alguns tipos e técnicas de testes que não são realizados, pelo time de desenvolvimento, da empresa em questão. Outros tipos e técnicas não são executados de acordo com o esperado pelo modelo *Agile Testing*.

Com as informações levantadas neste trabalho, caso seja do interesse da empresa, pode-se criar um planejamento com objetivo de atingir aderência total ao *Agile Testing*. Também é possível identificar pontos de melhoria em relação a testes já realizados, como por exemplo, automatizar os testes do Q2 que são realizados de forma manual. Sendo assim, para a empresa em questão, o levantamento realizado representa uma contribuição positiva, pois mostra que existem tipos e técnicas que testes que estão de acordo com o modelo e existem testes que poderiam ser utilizados de forma mais eficiente.

Outra contribuição deste trabalho, é o modelo de avaliação de aderência do *Agile Testing*, que foi construído combinando os Quadrantes do *Agile Testing* com os graus de implementação do Guia de Avaliação do MPS-BR. Este modelo potencialmente utilizado por qualquer interessado em medir o grau de implementação do *Agile Testing* no seu projeto, time ou empresa.

Como trabalho futuro, poderiam ser estudadas formas de medir cobertura de testes automatizados na empresa em que foi realizado o estudo de caso. Isso porque medir a cobertura poderia ajudar a garantir que os testes sejam constantemente melhorados, e evoluídos juntamente com o software. Ainda, a cobertura de testes poderia ser uma métrica para o time, que quando coletada regularmente, iria compor uma base histórica, fundamental para o time entender o que precisa ser melhorado.

Embora a empresa avaliada não seja completamente aderente ao *Agile Testing*, os tipos e práticas de testes aplicados mostram-se eficientes no dia-a-dia do time de desenvolvimento. Cabe à empresa avaliar o ganho que seria obtido com uma aderência completa à prática *Agile Testing* e, se esta prática iria contribuir significativamente no resultado do trabalho do time de desenvolvimento. É importante ressaltar que *Agile Testing* é um conceito novo no mercado que pode demonstrar um caminho a ser seguido a fim de obter, entre outras vantagens, maior agilidade na realização dos testes, porém é necessário que se estude de forma aprofundada todas suas implicações.

REFERÊNCIAS

Agile, A. (2001) "Manifesto para o desenvolvimento ágil de software", http://manifestoagil.com.br/principios.html, fevereiro.

Alliance, S. (2012) "*Scrum*: uma descrição", https://www.*Scrum*alliance.org/*Scrum*/media/*Scrum*AllianceMedia/Files%20e%20PDFs/Why%20Scrum/Core%20Scrum%20Translations/Core-Scrum-Portuguese.pdf, fevereiro.

Bastos, A. e Cristalli, R. e Moreira, T e Rios, E. (2012) "Base de Conhecimento Em Teste de Software" 3ª Edição.

Beck, K. (2000), Extreme Programming Explained: Embrace Change, 1ª Edição.

Beck, K. e Eres, C. (2004), Extreme Programming Explained: Embrace Change. 2ª Edição.

Beck, K. (2002), Test Driven Development: By Example, 1ª Edição.

Brito, E. (2012) "Diretrizes para avaliação de ferramentas de gestão de projetos utilizando metodologias ágeis", http://www.espweb.uem.br/wp/wp-content/uploads/2012/05/Emerson-Jos%C3%A9-Morgado-Brito.pdf, março.

Cohn, M. (2004), User Stories Applied, 1ª Edição.

Crispin, L. e Gregory, J. (2009), *Agile Testing*: a practical guide for testers e agile teams, 1ª Edição.

Crispin, L. (2009) "Using the *Agile Testing* Quadrants" http://lisacrispin.com/2011/11/08/using-the-agile-testing-quadrants/, Março.

Duval, P. M. (2007), Continuous Integration: Improving Software Quality e Reducing Risk, 7ª Edição.

Fowler, M. (2006) "Continuous Integration", http://martinfowler.com/articles/continuousIntegration.html, Março.

Humble J. e Farley D. (2010), Continuous Delivery: Reliable Software Releases through Build, Test, e Deployment Automation, 1ª Edição.

Ghirghiu, G. (2005) "*Agile Testing*", http://agiletesting.blogspot.com, Março.

ISTQB (2014) "Foundation Level Extension Syllabus Agile Tester", http://www.istqb.org/downloads/finish/52/138.html, Maio.

Jacobson, I. (2002) "A Resounding "Yes" to Agile Processes – But Also More", http://www.taika.com/techlib/files/agile-processes.pdf, Março.

Linz, T. (2014) "Testing in Scrum: A Guide for Software Quality Assurance in the Agile World", http://www.4qa.by/wp-content/uploads/persist/eurostar-conferences-ebooks-testing-in-scrum-tilo-linz.pdf, Maio.

Meszaros, G. (2007), XUnit Test Patterns: Refactoring Test Code, 1ª Edição.

Moreira, A. (2013) "Introdução: Integração Contínua", http://siep.ifpe.edu.br/eerson/blog/?page_id=1015, maio.

North, J. (2006) "Introducing BDD", http://dannorth.net/introducing-bdd/, Março.

Nogueira, E (2013) "Introdução do Quadrante de Teste Ágil", http://blog.adaptworks.com.br/2013/11/introducao-do-quadrante-de-teste-agil/, maio.

Pressman, R. S. e Maxim, Bruce. R. (2014), Software Engineering: a practitioner's approach, 8ª Edição.

Pressman, R. S. (2011). Engenharia de software: uma abordagem profissional, 7ª Edição.

Schwaber, K. e Sutherland, J. (2013) "Guia do Scrum - Um guia definitivo para o Scrum: As regras do jogo", http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf, março.

Smart, J. F. (2014), BDD in Action: Behavior-driven development for the whole software lifecycle, 1ª Edição.

Softex (2013) "MPS.BR - Guia de Avaliação", http://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_de-Avaliacao_2013.pdf, junho.

Sutherland J. (2014) "The Scrum Hebook", http://www.scruminc.com/wp-content/uploads/2014/07/The-Scrum-Hebook.pdf, Março.

Valentim, M. L. P. (2005), Métodos qualitativos de pesquisa em Ciência da Informação, 1ª Edição.