APLICAÇÃO DE TESTE DE SOFTWARE ÁGIL NA REGIÃO DO CARIRI - UM COMPARATIVO ENTRE METODOLOGIAS ÁGEIS NO DESENVOLVIMENTO DE SOFTWARE

Antônio de Pádua Pereira Carvalho / José Wilson Gonçalves
Wellington Feitoza Gonçalves

RESUMO

Este trabalho faz um estudo sob a ótica dos desenvolvedores de Software da Região do Cariri cearense, em relação a boas práticas do desenvolvimento de Sistemas utilizando a Engenharia de Software. Será analisado teste de software, o qual faz abordagem das dimensões do teste (Como, o Que e Quando testar), o ciclo de vida, os métodos ágeis como o scrum e xp com suas respectivas práticas, papéis e responsabilidades de uma equipe de testes em relação ao desenvolvimento de software, o processo de testes como o planejamento e controle, elencando resoluções para pontos cruciais do projeto, elevando assim a qualidade do produto final, o método quantitativo será abordado como pesquisa e levantamento de dados utilizando questionário e entrevista com profissionais da área do desenvolvimento de software da região do Cariri.

ABSTRACT

This work makes a study from the perspective of the Software developers of the Cariri region of Ceará, in relation to good practices of the Systems development using Software Engineering. Software testing will be analyzed, which addresses the test dimensions (How, What and When to test), the life cycle, agile methods such as scrum and xp with their respective practices, roles and responsibilities of a testing team in relation to software development, the testing process such as planning and control, listing resolutions for crucial points of the project thus raising the quality of the final product, the quantitative method will be approached as research and data collection using questionnaire and interview with professionals from the area of software development in the Cariri region.

1 INTRODUÇÃO

É notório que as empresas estão convertendo suas aplicações corporativas em sistemas orientados a serviços para criar serviços compostos onde estes podem gerar uma aplicação web sendo desenvolvida a partir de uma composição de serviços. A composição pode ser utilizada para incorporar regras de negócio, onde este terá uma funcionalidade mais extensa.

As indagações a serem contestadas nos processos ágeis tem como objetivo uma análise mais abrangente e extensa sobre o módulo onde a agilidade é mais do que uma resposta efetiva às mudanças e as alterações ocorrentes e pode ser aplicada em qualquer processo do ciclo de vida dos sistemas.

Este artigo explicará a interação de testes dos processos ágeis no desenvolvimento, diante das mudanças existentes no mercado. Os principais benefícios de utilizar as boas práticas da engenharia de software no desenvolvimento e nos testes onde apresentaremos os resultados do estudo nas seções seguintes.

2. REFERENCIAL TEÓRICO

2.1 MÉTODOS ÁGEIS

A engenharia de software é um guia de boas práticas para o desenvolvimento de sistemas, e tem mais importância do que a programação em si. É uma área onde o foco está na produção de sistemas em todas as suas fases do projeto.

De acordo com Pressman(2001), a agilidade é a palavra da atualidade, equipes ágeis, desenvolvimento ágil, teste ágil, onde equipes dão resposta adequada a mudanças durantes as fases do desenvolvimento e encaram de forma simples o manifesto ágil. Onde o desenvolvimento ágil é um conjunto de metodologias que dá suporte a projetos relacionados a engenharia de software.

Para Sommerville (2011) os testes em metodologias ágeis seguem padrões diferentes dos métodos tradicionais, pelo fato de não existir documentação extensa que possa servir como embasamento.

O teste de software serve para determinar se o sistema atingiu suas especificações e se funciona como foi projetado. Nesta fase é onde se encontram as falhas para serem corrigidas, visando encontrar o máximo de falhas possíveis e que possam ser corrigidos pela equipe de forma ágil antes da entrega final. (PRESSMAN, 2011)

Segundo Pressman, o teste de caixa preta tem como finalidade testar os seguintes categorias: Funções, erros de interface, erro de estrutura de dados ou acesso a banco de dados externos, erros de validação ou de execução. A entrega é a última parte da fase dos testes e deve estar finalizada e documentada de acordo com o que foi planejado com o cliente no projeto.

2.2 EXTREME PROGRAMMING (XP)

Princípios: (XP) Extreme Programming

Extreme Programming foi elaborado e criado por Kent Beck na década 90 (1997), em um projeto na Chrysler, onde este é um pacote de práticas simplificadas de desenvolvimento de software. XP é uma metodologia de desenvolvimento de software ágil, leve, pratica, não ditada, que tem um conjunto de fundamentos fundamentados, sendo que o XP pode ser utilizado por equipes de desenvolvimento e desenvolvedores medianos e não apenas por equipes muito experientes, porém muitos contestam sua real utilidade, por pouca compreensão ou assimilação de seus artefatos, deixando de lado o resto, documentações, organização, etc.

O objetivo, o propósito principal do XP e aplicar um conjunto de boas práticas da Engenharia de Software, buscando sempre as aplicar na sua forma máxima e extrema. O paradigma com o XP é mudado, onde não tem o medo da mudança, visto que o erra acontece com um baixo custo. Nos modelos tradicionais quanto mais tarde a mudanças, maiores são os custos no projeto, sendo estas evitadas de se fazerem na maioria das vezes, o XP diz que pode sim haver estas mudanças, sem receio, sobretudo quando as suas práticas são aplicadas arisca.

Práticas do Extreme Programming

O XP porta um grupo lógico de métodos de engenharia de software, que reforçam os valores e agrega dinâmicas de equipe. No XP temos um grupo de

quatro valores: comunicação, simplicidade, feedback e coragem, que a partir desses valores são geradas doze práticas: jogo do planejamento, relesases pequenos, metáfora, projeto simples, testes constantes, refatoramento, programação em pares, propriedade coletiva de código, iteração contínua, semana de 40 horas, cliente no local, padrões de codificação.

Abaixo, o que são cada uma das práticas do XP.

2.1.1 O jogo do planejamento

O planejamento de uma release e das iterações são realizados com base nos casos de uso simplificados e tem a colaboração dos desenvolvedores da equipe, cliente e são divididos em dois papéis:

- Negócio: Especialistas da equipe que entendem sobre o negócio e estabelecem prioridades para estabelecer as funcionalidades a serem entregues.
- Técnico: Pessoas que irão desenvolver as funcionalidades descritas e estimam qual o esforço e riscos envolvidos e comunicam ao pessoal de negócios.

2.1.2 Releases pequenos

Release são implantadas no cliente, isto é, são partes pequenas, que já estão prontas para serem usadas pelo cliente, uma funcionalidade denominada mais importante e desenvolvida mais rápido, pois o cliente tem mais necessidade em receber, estes releases são constantes (a cada 2, 3 meses) e desenvolvidas ao longo de uma interação sempre agregando valor ao cliente, para que não afete os prazos de desenvolvimento. Os fatores prazo e funcionalidades também podem serem seguidos, são analisados o que é mais importante, necessário e útil a se desenvolver a princípio para o cliente, então a release é entregue apenas com determinadas funcionalidades, estas mais importantes e depois ao passar do tempo são adicionadas novas funcionalidades.

Entretanto, algumas organizações que trabalham com a metodologia XP não querem entregas a curto prazo e frequentes, visto que na maioria das vezes antes do sistema ser entregue, módulo precisam dar algum tipo de treinamento, quando isso acontece as organizações deixam de seguir as boas práticas do XP.

2.1.3 Metáfora

Esta prática, oferece uma visão geral do sistema que pode ser compartilhada pelos stakeholders do projeto em questão, a ideia é aproximar o sistema que se encontra em desenvolvimento e o sistema, que não necessariamente é um software.

2.1.4 Projeto simples

Esta prática tem a finalidade de se concentrar em soluções simples e bem definidas para os problemas de hoje e que não se deve perder tempo investindo em soluções genéricas que demandem atender a funcionalidades futuras.

2.1.5 Testes constantes

Neste método os testes são realizados antes da programação, onde existem dois tipos de testes, são eles:

- Teste de unidade: São realizados para verificar tudo que possa dar errado.
- Teste unitário: São automatizados e verifica se o sistema passa em todos os testes.

Os testes são um mecanismo para assegurar que o sistema sempre funciona livre de erros e servem para dar um feedback aos stakeholders sobre as falhas encontradas.

2.1.6 Refatoramento

São melhorias constantes no projeto de software para garantir a capacidade do sistema se adaptar a mudanças, consistem em aplicar uma série de rotinas para melhorar o software tornando mais simples sem alterar sua funcionalidade.

2.1.7 Programação em pares

O desenvolvimento do código fonte nesta prática é escrito por uma dupla de programadores, onde estes possuem papéis distintos, sentados lado-a-lado e olhando para o computador. Um dos programadores será responsável pela codificação e pensará na lógica de programação já o outro parceiro observa o código e busca estrategicamente como melhorá-lo e simplificá-lo além de encontrar possíveis falhas. As duplas são trocadas frequentemente assim como os papéis com a finalidade de todos terem conhecimento sobre todas as partes do sistema.

2.1.8 Propriedade coletiva de código

A programação em pares incentiva a equipe a trabalhar mais próxima buscando atingir o melhor resultado esperado. A propriedade coletiva faz a equipe faz a equipe trabalhar unida buscando as melhores práticas no desenvolvimento onde todas as atividades são realizadas de forma controlada e pode ser facilitada com o uso de ferramentas de controle de versão.

2.1.9 Iteração contínua

As funcionalidades são implementadas e integradas várias vezes ao dia, uma forma simples de fazer isso é ter um terminal dedicado a integração e quando este equipamento estiver livre um par com o código a integrar ocupa este terminal e carrega a versão corrente do sistema, carrega as alterações que fizeram e rodam os testes até que eles estejam funcionando corretamente.

2.1.10 Semana de 40 horas

Este método não obriga equipes a trabalharem somente as 40 horas semanais em seus projetos, contudo o autor Beck, diz que não se deve trabalhar além de duas horas seguidas, para que o cansaço e a insatisfação não gerem em queda de qualidade do código.

2.1.11 Cliente no local

Em um projeto XP orienta que deve ter na equipe uma pessoa (usuário do sistema em desenvolvimento) para acompanhar e trabalhar junta com a equipe para responder as perguntas e dúvidas.

2.1.12 Padrões de codificação

O XP orienta a propriedade coletiva de código, então, é importante que se tenha padrões de codificação com o objetivo de que todos programem da mesma forma facilitando o entendimento e alterações no código.

2.2 SCRUM

O Scrum de início foi criado com o intuito de gerenciar projetos de fabricação de automóveis e de produtos de consumo, por Takeuchi e Nonaka no artigo “The new product development game”, em janeiro-fevereiro de 1986, pela Universidade de Harvard. Foi a partir deste artigo, que Takeuchi e Nokada perceberam que os

projetos realizados com equipes menores e multidisciplinares traziam melhores resultados, levando-os a associarem isto a formação Scrum do Rugby. Foi no ano de 1995, que o Scrum teve sua definição formalizada por Ken Schwaber, que fez um trabalho de consolidação para um método de desenvolvimento de software a nível mundial.

Este framework não é definido como uma técnica específica para o desenvolvimento de software durante a etapa de implementação, ele se intensifica em fazer a descrição de como os membros da equipe devem trabalhar para a produção de um sistema flexível, em um ambiente de constantes mudanças. O Scrum tem como ideia central o desenvolvimento de sistemas que envolvem inúmeras variáveis, sejam elas ambientais ou técnicas, onde estas possuem uma ampla perspectiva de mudança durante a execução do projeto. Os requisitos, prazos, recursos e tecnologias, são características que fazem com que o desenvolvimento do sistema de software se torne uma tarefa complexa e imprevisível, fazendo-se necessário o uso de um processo que seja flexível e que tenha a capacidade de responder às mudanças.

2.2.1 Elementos de apoio

Filho em seus estudos descreve os elementos de apoio do Scrum, e afirma que os únicos elementos produzidos pela equipe para seguir a práticas de Scrum são cartões com as funcionalidades e gráficos de acompanhamento. Os cartões são agrupados e formam o Backlog do Produto e outros backlogs. Já os gráficos, são atualizados com frequência para fazerem a reflexão do andamento do projeto. Estes estão disponíveis a seguir:

- Backlog do Produto: É responsável por fazer a listagem de todos os cartões de funcionalidades que o produto deve possuir e que ainda não foram desenvolvidas;
- Backlog Selecionado: É um subconjunto de funcionalidades que foram escolhidas pelo cliente a partir do backlog do produto para ser implementado no sprint atual e que não pode ser modificado durante o sprint;
- Backlog do Sprint: Faz uma listagem priorizada, que é obtida a partir da quebra dos cartões do backlog selecionado em tarefas menores;

- Backlog de Impedimentos: Faz uma listagem dos obstáculos que são identificados pela equipe e que não fazem parte do contexto do desenvolvimento;
- Gráficos de Acompanhamento: São gráficos que tem a finalidade de medir a quantidade de trabalho restante (burndown charts) e que são os preferidos em Scrum. Recomenda-se fazê-los para várias esferas do projeto: para o produto, para a release e para o sprint.

2.2.2 Papéis e responsabilidades

Os papéis no Scrum possuem tarefas e propósitos diferenciados durante o processo, sendo que suas práticas de acordo com Franco, são apresentadas a seguir:

- Cliente: Tem participação nas tarefas que se relacionam à definição da lista de funcionalidade do software para que seja desenvolvido ou melhorado, para elaborar os requisitos e restrições do produto final desejado.
- Gerente: Se torna o encarregado pela tomada das decisões finais, e utiliza as informações oculares que são disponibilizadas através de gráficos pelos padrões e convenções que devem ser seguidas no projeto. É o principal responsável por fazer os acordos relacionados aos objetivos e requisitos do projeto com os seus clientes.
- Equipe Scrum: Possui no projeto a autoridade na tomada de decisões e realiza as ações necessárias quanto a organização para poder atingir os objetivos preestabelecidos. É envolvida, na estimativa de esforço, na criação e revisão da lista de funcionalidade do produto, e vem a sugerir obstáculos que se façam necessários serem removidos do projeto.
- Scrum Master: Responsabiliza-se pela garantia de que o projeto seja conduzido das práticas, valores e regras que estão definidas no Scrum, onde o principal objetivo é que o progresso do projeto esteja em comum acordo com o que os clientes esperam. O Scrum Master faz a interação da Equipe Scrum, com os Clientes e o Gerente durante a execução do projeto. É também de sua responsabilidade a remoção e alteração de qualquer obstáculo que apareça ao longo do projeto, garantindo que a equipe trabalhe da forma mais produtiva possível.

- Responsável pelo Produto: É o responsável direto pelo projeto, gerenciamento, controle e por tornar visível a lista de funcionalidade do produto. É selecionado pelo Scrum Master, Clientes e Gerente. Tem responsabilidade na tomada das decisões finais referentes às tarefas necessárias para que ocorra a transformação da lista de funcionalidades no produto final, participando na estimativa do esforço de desenvolvimento necessário e por fim, se responsabiliza por fazer o detalhamento das informações que se referem à lista de funcionalidade que é utilizada pela Equipe Scrum.

2.2.3 Entregas contínuas

O modelo Scrum propõe uma metodologia que aplica um sistema de entregas contínuas. Nesta metodologia com os backlogs definidos (que são os requisitos funcionais do sistema), um sprint programado (tempo predeterminado no qual será dividido o trabalho para efetuação de uma entrega, e tem como prazo padrão duas a quatro semanas), são realizadas reuniões diárias com duração aproximada de 10 minutos, como forma de acompanhar o projeto e verificar se o mesmo está de acordo com o seu planejamento. Finalizado cada sprint, é realizada uma nova reunião de retrospectiva e planejamento do próximo Sprint.

2.2.4 Planejando as interações (entregas)

Para se planejar uma interação (uma entrega) na metodologia Scrum se faz necessário realizar uma sequência de atividades, que de acordo com Araújo e Galina são:

- Fazer a discussão de uma estória;
- Fazer a decomposição da estória discutida em tarefas;
- Fazer com que o desenvolvedor aceite a responsabilidade por cada tarefa;
- As estórias terem sido discutidas e todas as tarefas aceitas, os desenvolvedores individualmente fazem uma estimativa das tarefas aceitas para que possam se comprometer apenas com aquilo que realmente podem cumprir.

2.2.5 Desempenho da equipe e escopo de utilização

A definição do desempenho da equipe terá como base o histórico de entregas e conclusões de projetos anteriores, ações que irão definir o quanto em pontos a equipe consegue atingir. Através de um sistema de pontuação cada desenvolvedor atinge em média 20 pontos por Sprint e tem duas semanas de duração. Alguns fatores podem afetar o desempenho, os quais devem ser trazidos para que seja discutido na reunião de retrospectiva e planejamento. Este momento para fazer uma abordagem seguida de uma apresentação dos problemas e dificuldades encontrados, para que em seguida se buscar as resoluções e as melhorias no processo de desenvolvimento de software e consequentemente atingir o seu objetivo com relação a velocidade de conclusão dos próximos sprints.

A metodologia Scrum é destinada a pequenas equipes com menos de dez pessoas. De acordo com Schwaber e Beedle a equipe deve ser composta por cinco a nove integrantes, no caso de ter mais pessoas envolvidas no projeto é preferível que se formem múltiplas Equipes Scrum.

3 TESTE DE SOFTWARE

Os testes correspondem a uma etapa de muita importância no processo de desenvolvimento de software, pois validam se a aplicação está operando corretamente e estão atendendo aos requisitos especificados. Existem diversas técnicas que podem ser utilizadas em diferentes etapas e de maneiras diferentes para validar os pontos principais do software.

Pressman (2002) define qualidade como a satisfação de requisitos funcionais e de desempenho explicitamente declarados, normas de desenvolvimento explicitamente documentadas e características implícitas que são esperadas em todo software desenvolvido profissionalmente. O conceito de qualidade é variável, para a equipe de desenvolvimento de software um produto possui qualidade quando se comporta conforme está descrito no requisitos, já para o cliente um produto terá qualidade quando este atender suas necessidades. Devido ao aumento da exigência por qualidade, várias normas, metodologias ou modelos (ágile RUP, por exemplo) e órgão reguladores como CMMI, MPS.Br e ISO foram criado segundo o glossário de termos padrões da International Software Testing Qualifications Board (ISTQB)

temos: Teste é o processo que consiste em todas as atividades do ciclo de vida, tanto estáticas quanto dinâmicas, voltadas para o planejamento, preparação e avaliação de produtos de software e produtos de trabalho relacionados a fim de determinar se elas satisfazem os requisitos especificados e demonstrar que estão aptas para sua finalidade e para a detecção de defeitos.

A pouca importância aos testes extrapola as salas de aula e se reflete diretamente nas empresas de desenvolvimento de software, que consideram os testes como uma atividade secundária. Essa visão errônea, aliada ao fato de que os testes consomem tempo e recursos para sua execução, fazem com que os testes sejam suprimidos, principalmente quando se visa cumprir um prazo de entrega mal planejado. Porém, softwares mal testados provocam prejuízos enormes às organizações (BARTIÉ, 2002).

Segundo Kitchenham e Charters (2007) uma revisão sistemática deve ser realizada seguindo as etapas de planejamento da revisão, condução da revisão e documentação da revisão, podendo estes passos ser observados também para um mapeamento sistemático. Assim, as seções seguintes detalham cada etapa realizada.

3.1 TESTES ÁGEIS DE SOFTWARE: CONCEITOS E PRÁTICA

Para que sejam implementadas as técnicas ágeis em um ambiente de testes, se faz necessário mudar os costumes das equipes que estão acostumadas a trabalhar. Entre as mudanças necessárias, uma delas é a introdução ao conceito de colaboração com o cliente, descartando a utilização de uma documentação mais abrangente, para que este quesito passe a fazer parte do manifesto ágil. É a partir dessa técnica que passamos a fazer um trabalho mais próximo, mais eficiente e eficaz com o cliente, uma vez que a sua participação no processo será a peça fundamental para que a técnica ágil de testes possa ser utilizada. Na ocasião em que se fala em ágil, estamos falando que a equipe é infectada por testes e consequentemente o sistema é implantado para ser experimentado desde a iniciação do projeto e não apenas ao final da sua codificação.

Os testes de maneira ágil devem seguir a formulação de pequenas ações, para que a informação de retorno sobre a forma como está sendo construído o

sistema e os erros nele existentes possam ser rapidamente apresentados e reparados. A interação ao ser terminada deve estar com todos os testes concluídos com sucesso.

Quando bem aplicado, o processo de Teste Ágil torna-se muito produtivo, gera indicadores e encontra uma grande sucessão de defeitos nos resultados obtidos. O uso de uma ferramenta de gerenciamento torna-se indispensável, sendo recomendado a utilização do TestLink pelo custo benefício da ferramenta, que gera possibilidade da criação do Plano , dos casos de teste, dos papéis envolvidos, da execução de teste, do resultado e relatórios finais. O Teste Ágil pode tornar-se uma excelente opção para as empresas de TI de pequeno e médio porte ou ainda em pequenos projetos específico de uma empresa de com esse porte maior.

3.2 TEST DRIVENDEVELOPMENT (TDD)

Segundo Gomes(2013), o Test DrivenDevelopment (TDD) é um processo de grande importância onde o desenvolvedor começa a implantação de um novo serviço pelo teste onde este deve se manter simples e de fácil entendimento visando a qualidade do código fonte.

O conceito de TTD é uma forma de testar primeiro, antes de ver o software em sua versão final verificando os erros de lógica para que o software seja entregue de acordo com seus requisitos. (Lopes, 2012)

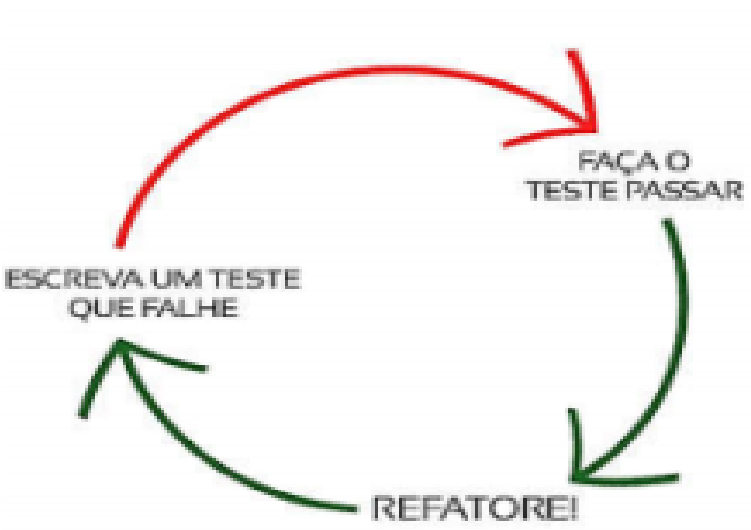
> [...] o teste não é uma tarefa trivial. A dificuldade em testar software é caracterizada por alguns pontos importantes como: o teste de software é um processo caro; existe uma falta de conhecimento sobre a relação custo/benefício do teste; há falta de profissionais especializados na área de teste; existem dificuldades em implantar um processo de teste; há o desconhecimento de um procedimento de teste adequado; há o desconhecimento de técnicas de teste adequadas; há o desconhecimento sobre como planejar a atividade de teste; e finalmente, a preocupação com a atividade de teste somente na fase final do projeto (CRESPO etal, 2004).

Contudo o autor Aniche (2012) orienta que o ciclo de vida do desenvolvimento guiado por teste (figura 1) consiste em:

- Escrever um teste unitário para a funcionalidade em questão;
- Falha no teste;

- Desenvolve código mais enxuto e simple;
- O teste é executado;
- A refatoração é executada para excluir a duplicidade.

Figura 1: Ciclo vermelho - verde- refatora



Fonte: ANICHE(2012).

Para os autores Seshadri e Green (2014) eles compreendem os princípios de TTD como código enxuto, e só é implementado quando ocorre alguma falha, o mínimo de código e quando todos os testes passarem o ciclo deve ser reiniciado.

3.3 ATDD

Para Nadalete(2010) os testes de aceitação são desenvolvidos de maneira colaborativa e descritos em uma linguagem simples. Onde toda equipe compartilha o mesmo entendimento do que deve ser feito, essa nova abordagem incorpora benefícios do TDD com critérios do cliente.

O ATDD tem com finalidade documentar os critérios de aceitação para as features em desenvolvimento e tende a prover um feedback sobre o andamento da conclusão da tarefa da equipe de desenvolvimento se encontra. (Nadalete, 2010)

O desenvolvimento de software orientado por testes é entendido pelas seguintes etapas:

- Discutir: Reunião colaborativa da equipe sobre as possíveis restrições, assunções, premissas, expectativas para os critérios de aceitação.
- Refinar: Trata do critério de aceitação junto e concreto de cenário de uso.
- Desenvolver (Develop): Trata da transformação dos testes de aceitação informando o comportamento esperado do software em teste.

4 METODOLOGIA

Segundo Yin (1994), o método quantitativo é o uso de instrumental estatístico, de dados numéricos. Já o método qualitativo se caracteriza pela qualificação dos dados coletados, durante a análise do problema.

Esta pesquisa tem o propósito de fazer um estudo exploratório e descritivo do cenário e contexto de Testes de Softwares Ágil, onde o alvo da investigação são profissionais da área de desenvolvimento de software, procurando enfatizar as vantagens e desvantagem dos testes em metodologias ágeis. O instrumento utilizado para a coleta de dados foi um questionário com questões abertas, onde foram aplicadas a profissionais da área da tecnologia.

5 RESULTADO

ENTREVISTADO – A.

O entrevistado A, possui Mestrado em Engenharia de Software, com mais de 5 anos de experiência no mercado de desenvolvimento de software, atenta que os testes de softwares usados nas metodologias ágeis funcionam realmente e que além de validar a implementação, também minimiza erros em todas as etapas do projeto, porém a dificuldades que são encontradas no processo dos testes, principalmente nas fases iniciais do processo de desenvolvimento, onde está concentrada a maior incidência de erros, como por exemplo, as ferramentas de automatização de teste e também servidores dedicados para testar uma aplicação por exemplo que envolve uma série de integrações, é algo caro. Cita também que o Scrum é a metodologia ágil onde os testes são melhores aplicados e tem maior nível de sucesso, sendo que os testes unitários são indispensáveis independente da metodologia aplicada.

ENTREVISTADO – B.

O entrevistado B, possui Graduação em Análise e Desenvolvimento de Sistemas, com mais de 5 anos de experiência no mercado de desenvolvimento de software, contempla que os testes funcionam sim nas metodologias ágeis e que diminuem os riscos de futuros bugs e outras reclamações de erros por parte do usuário final. Tem como melhor metodologia o Scrum, por ter maior equipe e mais elementos especializados, porém o tempo da Sprint pode ser curto para a realização dos testes, principalmente nas fases iniciais do processo de desenvolvimento, onde os erros, bugs tem maior ocorrência, porém os testes unitários e de aceitação do cliente são indispensáveis. Uma das principais vantagens encontradas na realização de testes de software nas metodologias ágeis e evitar retrabalho com bugs encontrados após a entrega do produto ao cliente final.

ENTREVISTADO – C.

O entrevistado C, possui Graduação em Administração, com mais de 5 anos de experiência no mercado de desenvolvimento de software, considera como promissores os testes nas metodologias ágeis, pois dão mais velocidade e eficiência na produção do software, sendo esses mais precisos. Referência que o acontecimento grandessíssimo de erros está nas fases finais do processo, e a comunicação como sendo a principal dificuldade encontradas na realização dos testes, porém com essa dificuldade quebrada a velocidade e a eficiência dos testes e umas das mais importantes vantagens encontradas na realização de testes de software ágil.

ENTREVISTADO – D.

O entrevistado D, Mestrando de Engenharia de Software, com mais de 10 anos de experiência no mercado de desenvolvimento de software, diz que os testes funcionam sim nas metodologias ágeis e acredita que é necessário realizar os devidos testes, para evitar possíveis bugs no sistema, ou um excesso deles. O Scrum se aplica melhor como metodologia ágil devido as Sprints, entregas rápidas, com testes funcionais que são indispensáveis sobretudo nas fases iniciais do

processo de desenvolvimento para evitar problemas de indisponibilidade do sistema, onde tem como maior dificuldade encontrada, registros de relatos dos usuários.

ENTREVISTADO – E.

O entrevistado E, possui Graduação em Sistema de Informação, com mais de 5 anos de experiência no mercado de desenvolvimento de software, considera que os testes unitários e end-to-end são indispensáveis no decorrer de todo o processo, inclusive nas fases iniciais do processo, onde ocorrem o maior número de erros, bugs, muitas vezes por ser dado pouca importância a estes. Cita que o XP tem maior nível de sucesso nos testes, proporcionando qualidade para o produto, visto que os testes de software usados nas metodologias ágeis tem bom resultado, já o Scrum é para gerenciar projetos.

6 ANÁLISE DOS RESULTADOS

De acordo com os entrevistados a metodologia ágil é importante principalmente em projetos de curto prazo porém é indispensável o teste nas fases definida pois é a partir daí que sabemos de fato se o produtos está pronto para o cliente utilizar. Sabemos que existe algumas limitações para as empresas desenvolvedora de software seguir uma metodologia pois cultura organizacional da maioria das empresas pequena e sem processo definido e muito menos o conhecimento sobre metodologia ágil. Podemos também citar como dificuldade a conhecimento a ferramenta para teste. Mas para que o produto seja entregue com qualidade ao cliente é necessário passar por todo o processo da metodologia ágil.

7 CONSIDERAÇÕES FINAIS

Pode-se perceber a partir da análise realizada sobre os estudos e métodos de testes de software ágeis, que os testes de software em metodologias ágeis atuam sim, de forma eficaz, com técnicas que são ajustadas, logo o que se precisa saber é como aplicar as técnicas de testes e como colaborar com o Time já que o ambiente ágil é altamente colaborativo. É evidente que os obstáculos relevantes são manter o alinhamento entre as tecnologias e frameworks utilizadas no desenvolvimento, já que a aplicação de testes deve ter o mesmo horizonte de referência, isto influencia

no esforço durante a execução dos testes e em uma interpretação mais clara dos resultados.

No entanto, quando se utiliza métodos ágeis não se espera que uma única equipe se responsabilize pela qualidade final das entregas, mas sim que todos na equipe tenham sua colaboração no controle dessa qualidade. Compartilhando responsabilidade, logo você não tem um processo de teste, você tem procedimentos de testes executados em várias atividades, processos do desenvolvimento do software. Ainda segundo autores autores, os mesmos apelam para o uso do Acceptance Test-Driven Development (ATDD), ou "Desenvolvimento Orientado a Testes de Aceitação" juntamente com o Scrum, sendo que o (ATDD) é uma prática de obtenção de requisitos de forma colaborativa utilizada por equipes ágeis, onde exemplos objetivos e testes automatizados são utilizados para especificar os requisitos, fazendo-se mais claros, com o objetivo de criar especificações executáveis, estes são feitos em sessões de criação do backlog do produto, onde a participação do Product Owner, da equipe, além dos demais interessados. Teste é adequado ao projeto ou produto, a qualidade que ele proporciona pode ser requerida ou não, logo, não há um teste "indispensável", mas qual teste que permita pegar o defeito antes do cliente se faz necessário, e contudo resolvê-lo.

8 REFERÊNCIAS BIBLIOGRÁFICAS

- [Agile Manifesto (2004)] Disponível em http://agilemanifesto.org/, acessado em 11 de Setembro de 2019.
- ANICHE, Mauricio. Test DrivenDevelopment - Teste e Design no Mundo Real. 1. ed. São Paulo: Casa do Código, 2012.
- ARAUJO, R. C., GALINA, T. C.: Análise de Escopo e Planejamento no Desenvolvimento de Software, sob a Perspectiva Ágil. Universidade do Vale do Rio dos Sinos (Unisinos).
- BARTIÉ, Alexandre. Garantia de Qualidade de Software: adquirindo maturidade organizacional. Rio de Janeiro: Editora Campus, 2002.
- [Beck (1999)] Beck, K. Programação Extrema Expli-cada. Bookman, 1999.
- [Brooks (1987)] Brooks, F. No Silver Bullet: Essence and Accidents of Software Engineering. Proc. IFIP, IEEE CS Press, pp. 1069-1076; reprinted in IEEE Computer, pp. 10-19, Apr. 1987.
- [Charette, R., (2001)] Charette, R. “Fair Fight? Ag-ile Versus Heavy Methodologies”, Cutter Consor-tium E-project Management Advisory Service, 2,13, 2001.
- CRESPO, Adalberto Nobiato. et al. Uma Metodologia para Teste de Software no Contexto da Melhoria de Processo. Artigo, São Paulo: [2004?]. Disponível em:<http://www.testexpert.com.br/files/Uma%20Metodologia%20para%20Teste%20de%20Software%20no%20Contexto%20da%20Melhoria%20de%20Processo.PDF>.
- [Cockburn et al., (2001)] Cockburn, A. e Highsmith, J. “Agile Software Development: The Business of Innovation”, IEEE Computer, Sept., pp. 120-122, 2001.
- EXTREME Programming (XP). [S. l.], 2009. Disponível em: https://www.cin.ufpe.br/~gamr/FAFICA/Desenvolvimento%20de%20sistemas/XP.pdf. Acesso em: 22 fev. 2020.
- [Gilb (1999)] Gilb, T. Principles of Software Engineer-ing Management. Addison-Wesley, 1988.
- GOMES, André Faria. Agile - Desenvolvimento de software com entregas frequentes e foco no valor de negócio. 1. ed. São Paulo: Casa do Código, 2013.

- [Highsmith et al., (2000)] Highsmith, J. Orr, K. Cock-burn, A. Extreme Programming. E-Business Ap-plication Delivery, Feb., pp. 4-17, 2000.
- ISO/IEC 12207, International Organization for Standardization/ International Electrotechnical Comission. ISO/IEC 12207 Systems and software engineering–Software life cycle processes, Geneve: ISO, 2008.
- LOPES, Camilo. Test Driven Development naPrática. 1. ed. Rio de Janeiro: Ciência Moderna, 2012.
- NADALETE, Lucas Gonçalves. Desenvolvimento de Software Orientado a Aspectos. São Paulo,edição 1.2010.
- [Pressman (2001)] Pressman, R. Engenharia de Soft-ware. McGraw-Hill, 2001.
- REBELO, Paulo. Acceptance Test-Driven Development (ATDD), passo a passo. InfoQ. Disponível em: .
- SHYAM, Seshadri; GREEN, Brad. Desenvolvendo com AngularJS: Aumento de produtividade com aplicações web estruturadas. São Paulo: Novatec, 2014.
- SOMMERVILLE, I. (2007) Engenharia de Software. 8° edição. Pearson Education.
- T. Dyba?, T. Dingsøyr. Empirical studies of agile software development: A systematic review. Information and Software Technology 50 (2008) 833–859
- Torgeir Dingsøyr, Et al. A decade of agile methodologies: Towards explaining agile software development. The Journal of Systems and Software 85 (2012) 1213–1221
- YIN, R. K. Estudo de caso: planejamento e métodos. 3. ed. Porto Alegre: Bookman, 2005.

APÊNDICE - I

Questionário Teste de Software Ágil

1. Nome do entrevistado? Local (cidade)?

*

2. Formação do profissional?

O profissional atua no mercado há quantos anos?

*

De 1 a 5 anos

De 5 a 10 anos

Mais de 10 anos

3. Na sua opinião profissional, os testes de software usados nas metodologias ágeis funciona realmente? Justifique.

*

4. Quais as principais dificuldades encontradas na realização de testes de software nas metodologias ágeis?

*

5. Quais as principais vantagens encontradas na realização de testes de software nas metodologias ágeis?

*

6. Onde está concentrada a maior incidência dos erros nos testes de software ágil?

*

nas fases iniciais do processo de desenvolvimento

No meio do processo

nas fases finais do processo

7. Na sua opinião, qual dessas duas metodologia ágeis os testes são melhores aplicados e tem maior nível de sucesso, Scrum ou XP? Justifique.

Question Type

8. Na sua opinião, a metodologia sendo ágil ou não, quais testes são indispensáveis no processo de desenvolvimento do software?