PROPOSTA DE UM PROCESSO DE TESTE DE *SOFTWARE* PARA EMPRESAS DE PEQUENO PORTE

Proposal of a process software test for small business size

Pedro Sidnei Zanchett[1]
Larissa Caroline Flores[2]

Resumo: O presente artigo apresenta um estudo realizado por meio de pesquisa bibliográfica, com o objetivo de expor um processo de teste de *software* que se adéque a empresas de pequeno porte. É importante incorporar processos de teste durante aos processos de desenvolvimento a fim de garantir que o *software* tenha uma menor quantidade de falhas e seja entregue no prazo estipulado, pois, na prática da engenharia de *software*, existem muitos impedimentos pela falta de gestão na área de qualidade de *software*. Para isso, foram analisadas as características das principais metodologias de processo existentes e sugere-se o uso da metodologia ágil Scrum como uma alternativa.

Palavras-chave: Processo de testes de *software*. Qualidade de *software*. Métodos ágeis. Scrum. Pequenas empresas.

Abstract: This article presents a study by means of literature, in order to expose a software testing process that fits the small businesses. It is important to incorporate testing processes during the development process to ensure that the software has fewer flaws and is delivered with quality within the stipulated time, as in the practice of software engineering there are many impediments by the lack of management in the area of quality software. For this the characteristics were analyzed the main existing process methodologies and we suggest using the Scrum agile methodology as an alternative.

Keywords: Process test software. Quality software. Agile methods. Scrum. Small business.

Introdução

Existe uma demanda crescente de utilização de *software* nas mais variadas áreas e, portanto, desenvolver produtos de qualidade que atendam o mercado cada vez mais exigente tem sido fundamental para as empresas se posicionarem de forma competitiva. Com isso, as organizações começaram a focar em treinamentos dos profissionais e melhorias no ambiente de trabalho. Para avançar na qualidade, começaram a ser observados os processos de desenvolvimento de *software* utilizados na empresa.

Com foco nos processos, começaram a surgir nas empresas novas frentes de trabalho, buscando incorporar tendências e metodologias capazes de obter o melhor aproveitamento dos profissionais e resultar num produto mais padronizado e com melhores índices de qualidade.

Para obter uma alta qualidade dos produtos de *software*, diversos fatores influenciam, como o escopo do produto, a disponibilidade de tempo para execução do projeto, grau de conhecimento da equipe técnica, as dificuldades do negócio, entre outros. Com isso, para obter produtos mais confiáveis, seguros, com melhor desempenho e que atendam às necessidades dos usuários é importante que a produção seja regida por padrões, métodos e processos que possuam uma estruturação completa.

Engenharia de *software* é uma disciplina de engenharia cujo foco está em todos os aspectos da produção de *software*, desde os estágios iniciais da especificação do sistema até sua manutenção, quando o sistema já está sendo usado (SOMMERVILLE, 2011).

[1] Mestre em Engenharia e Gestão do Conhecimento pela Universidade Federal de Santa Catarina. Docente no Centro Universitário Leonardo Da Vinci – UNIASSELVI e Centro Universitário de Brusque - UNIFEBE. E-mail: pedrozanchett@gmail.com
[2] Especialista em Engenharia de Software pela Pós-graduação Centro Universitário Leonardo Da Vinci – UNIASSELVI. E-mail: larissacarol@gmail.com

Nesse sentido, a engenharia de *software* possui grandes contribuições a fazer, incluindo o foco do projeto desenvolvido: teste de *software*.

Uma possível definição mais abrangente e completa para qualidade de *software* seria a proposta por Bartié (2002, p. 16): ‘‘Qualidade de software é um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos’’.

Existem várias micro e pequenas empresas que ainda não têm definido um processo de teste formal e também faltam subsídios para implantação de um processo que atenda às necessidades e consiga executar as atividades de forma correta. Normalmente, os testes, quando incluídos no processo de desenvolvimento de *software* de micro e pequenas empresas, são executados por desenvolvedores ou analistas de sistemas. No entanto, os profissionais de desenvolvimento não possuem conhecimento sobre as técnicas de teste e, portanto, não podem se beneficiar com a aplicação dos critérios mais adequados ao contexto da organização e das características do *software* que está sendo desenvolvido.

Com relação a estas ideias, o Manual da FAPEG (2013) destaca que:

> Apesar de existirem inúmeras técnicas, políticas e metodologias na área de teste de software, no âmbito das micro e pequenas empresas existe uma deficiência perceptível em formas específicas para tais, em especial, no segmento de processos de teste de software e ferramentas que o apoiem de forma integrada.

Segundo Sartori (2005), as pequenas empresas têm características peculiares e distintas das grandes: geralmente desenvolvem *softwares* menores e menos complexos; não dispõem de muitos recursos financeiros; evitam ferramentas caras, sofisticadas e com procedimentos complexos; seus processos e métodos são únicos.

Conforme Habra (apud BERNI, 2010), faltam recursos nas micro e pequenas empresas (MPE), tanto humanos quanto financeiros, atrelados ao processo de produção. Estas empresas têm, por definição, pequenas equipes e as pessoas envolvidas são pressionadas por prazos apertados para finalização das tarefas a elas atribuídas, há dificuldades em definir um processo padrão de desenvolvimento de *software* para ser seguido por todos, a partir de modelos de processo prescritivos, uma vez que esses têm como característica a forte ênfase em controle e documentação, o que pode burocratizar a empresa em demasia, engessando-a.

Em um ambiente de desenvolvimento de *software* em que os requisitos sofrem constante modificação, em que velocidade de resposta e entrega de produtos é o diferencial competitivo, um modelo de processo adaptável e flexível torna-se mais adequado às pequenas empresas de *software*, pois permite um acompanhamento e controle sem gerar grandes mudanças nas rotinas diárias da empresa.

> O modelo cascata, algumas vezes chamado de ciclo de vida clássico, sugere uma abordagem sequencial e sistemática para o desenvolvimento de software, começando com o levantamento de necessidades por parte do cliente, avançando pelas fases de planejamento, modelagem, construção, emprego e culminando no suporte contínuo do software concluído (PRESSMAN, 2011 p. 59).

Pressman (2011) acrescenta que existe uma variação do modelo cascata, chamado de modelo V, o qual descreve a relação entre ações de garantia da qualidade e as ações referentes à comunicação, à modelagem e a atividades de construção.

> Os métodos ágeis são métodos de desenvolvimento incremental em que os incrementos são pequenos e, normalmente, as novas versões do sistema são criadas e dispo-

> nibilizadas aos clientes a cada duas ou três semanas. Elas envolvem os clientes no processo de desenvolvimento para obter um feedback rápido sobre a evolução dos requisitos. Assim, minimiza-se a documentação, pois se utiliza mais a comunicação informal do que reuniões formais com documentos escritos (SOMMERVILLE, 2011 p. 39).

Sommerville (2011) afirma ainda que os processos de desenvolvimento de *software* que pretendem especificar por completo os requisitos para depois projetar, construir e testar não estão adaptados a um desenvolvimento rápido, ou seja, como normalmente durante o processo ocorrem mudanças nos requisitos, um processo convencional em cascata ou os que são baseados em especificações costumam ser demorados e o *software* é entregue ao cliente depois do prazo.

Qualidade de *software*

Princípios

Qualidade de *software* está relacionada a entregar ao cliente o produto final que satisfaça suas expectativas, dentro daquilo que foi acordado inicialmente por meio dos requisitos do projeto. Nesse contexto, qualidade de *software* objetiva garantir essa qualidade pela definição de processos de desenvolvimento (ENGHOLM JR, 2010).

Ainda nesse mesmo contexto, Pressman (2011) afirma que, no desenvolvimento de *software*, especificar a qualidade de um projeto compreende atender funções e características especificadas nos "requisitos desenvolvidos". O foco está no grau em que a implementação resulte no atendimento das necessidades e nas metas de desempenho.

Pode-se encontrar nas literaturas diversas definições para requisito de *software*.

• Requisitos de um sistema são descrições dos serviços que devem ser fornecidos por esse sistema e as suas restrições operacionais (SOMMERVILLE, 2011).

• Um requisito de um sistema é uma característica do sistema ou a descrição de algo que o sistema é capaz de realizar para atingir seus objetivos (PFLEEGER, 2004).

Baseado nessas definições, pode-se dizer que os requisitos de um sistema incluem especificações que o sistema deve prover, as propriedades que deve possuir e restrições que devem ser observadas no processo de desenvolvimento. Observa-se que as definições apontam para a existência de diferentes tipos de requisitos. Pode ser verificado que uma definição que é aceita quanto ao tipo de informação documentada por um requisito é o que distingue os requisitos funcionais e os nãos funcionais.

Conforme Pfleeger (2004), um requisito funcional descreve uma interação entre o sistema e seu ambiente. Por exemplo, para se determinar os requisitos funcionais, decidimos quais estados são aceitáveis para o sistema. Além disso, os requisitos funcionais descrevem como o sistema deve se comportar, considerando certo estímulo.

Nesse sentido, Sommerville (2011) afirma que requisitos funcionais são declarações de serviços que o sistema deve prover, descrevendo o que o sistema deve fazer. Os requisitos não-funcionais, em vez de informar o que o sistema fará, colocam restrições no sistema. Isto é, os requisitos não-funcionais ou restrições descrevem uma restrição no sistema que limita nossas opções para criar uma solução para o problema (PFLEEGER, 2004).

Baseado nesse conceito, percebe-se que a qualidade deve ser compreendida e embutida no processo de desenvolvimento desde a documentação e implementação, resultando dessa maneira em um *software* que possua uma maior estabilidade.

Sommerville (2011) sugere uma estrutura preliminar para um plano de qualidade, a qual inclui:

1. Introdução ao produto: descrição do produto, qual nicho de mercado pretendido e quais são as expectativas de qualidade do produto.
2. Planos de produto: as datas críticas de release e responsabilidade para o produto, junto com os planos para a distribuição e prestação de serviço do produto.
3. Descrições de processo: os processos de desenvolvimento e serviço são padrões que devem ser usados para o gerenciamento e desenvolvimento de produto.
4. Metas e qualidade: as metas de qualidade e planos para o produto, incluindo uma identificação e uma justificativa para os atributos críticos de qualidade do produto.
5. Riscos e gerenciamento de riscos: os riscos mais importantes que podem afetar a qualidade do produto e ações que devem ser tomadas ao lidar com eles.

Ainda sobre esses paradigmas, Pressman (2011, p. 365) enfatiza que "Software bom o suficiente fornece funções e características de alta qualidade que os usuários desejam, mas, ao mesmo tempo, fornece outras funções e características mais obscuras".

Sommerville (2011) destaca que o teste de *software* serve para evidenciar que o programa faz o que ele realmente deve fazer e para evidenciar os defeitos que existem antes do uso. No processo de teste, existem dois objetivos distintos, que são demonstrar que o *software* atende seus requisitos e descobrir em que situação o *software* se comporta de forma incorreta.

O teste busca descobrir a maior quantidade de defeitos possível, é importante saber onde os defeitos podem estar. Saber como os defeitos são criados nos dá pistas sobre onde procurá-los durante o teste do sistema (PFLEEGER, 2004).

Pode-se concluir que determinar a qualidade do *software* pode ter diversos parâmetros a serem analisados de acordo com o entendimento do usuário.

Pressman (2011) afirma que a qualidade de *software* é difícil de definir, porém, é algo que é necessário ser feito e que envolve todas as pessoas (engenheiros de *software*, gerentes, todos os interessados, todos os envolvidos) na gestão de qualidade, que são responsáveis por ela. Se uma equipe de *software* enfatizar a qualidade em todas as atividades de engenharia de *software*, ela reduzirá a quantidade de reformulações que terá de fazer. Isso resulta em custos menores e, mais importante ainda, menor tempo para colocação do produto no mercado. Para obter um *software* de qualidade, devem ocorrer quatro atividades: processo e práticas comprovadas de engenharia de *software*, gerenciamento consistente de projetos, controle global de qualidade e a presença de uma infraestrutura para garantir a qualidade. Para garantir que o trabalho foi realizado corretamente, é importante acompanhar a qualidade por meio da verificação dos resultados de todas as atividades de controle de qualidade, medindo a qualidade, efetuando a verificação de erros antes da entrega e de defeitos que acabaram escapando e indo para a produção.

Processos

A atividade de teste de *software* possui uma série de limitações, dificuldades e características únicas, as quais necessitam ser tratadas durante a sua execução, a fim de garantir o seu sucesso. Com isso, planejar e controlar essas atividades se torna fundamental para ter um bom resultado. Definir prazos e gerenciar os riscos se torna imprescindível.

A atividade de planejamento consegue definir ações a serem realizadas na execução de uma tarefa, pois o escopo do projeto é composto pelas informações dos requisitos de *software* que definem o entendimento e execução das tarefas de testes. Com isso, planejar os testes deve

fazer parte do planejamento total. Com esses subsídios, é necessário que se obtenha um plano de teste que estabeleça os recursos, defina estratégias, critérios e técnicas de teste a ser aplicada à definição do conjunto de casos e procedimentos de teste.

Na linha de qualidade de *software* existem processos que padronizam as etapas, de acordo com a maturidade do projeto.

Pressman (2011) define processo como um conjunto de atividades de trabalho quando algum artefato de *software* precisa ser produzido. Cada uma dessas atividades faz parte de uma metodologia ou modelo que determinará seu relacionamento entre cada uma das partes.

Sommerville (2011, p. 24) enfatiza que:

> As quatro atividades básicas do processo – especificação, desenvolvimento, validação e evolução – são organizadas de formas diferentes conforme o processo de desenvolvimento. No modelo cascata são organizadas em sequência, enquanto no desenvolvimento incremental são intercaladas. A maneira como estas atividades serão feitas depende do tipo de software, das pessoas e das estruturas organizacionais envolvidas.

Existem diversos modelos de processo de *software* que focam o desenvolvimento, por exemplo, cascata, incremental, espiral, processo unificado, entre outros. Nesses processos, existe um enfoque grande em documentação, no qual pode ser considerado um fator crítico para muitas empresas. Partindo desse pressuposto, surgiram metodologias ágeis que têm o enfoque na comunicação entre as pessoas e, por consequência, a minimização de documentação (SOMMERVILLE, 2011).

Teste de *software*

Pode-se encontrar várias definições sobre teste de *software*, entre elas, destaca-se que é o processo que visa a sua execução de forma controlada, com o objetivo de avaliar o seu comportamento baseado no que foi especificado. A execução dos testes é considerada um tipo de validação (RIOS; MOREIRA, 2013).

Na área de testes também existem diversos tipos de teste, os quais são aplicados em estágios diferentes. Conforme Rios e Moreira (2013, p. 16):

• Teste Caixa Preta (*Black Box*): visa verificar a funcionalidade e a aderência aos requisitos, em uma ótica externa ou do usuário, sem se basear em qualquer conhecimento do código e da lógica interna do componente testado.

• Teste Caixa (*White Box*): visa avaliar as cláusulas de código, a lógica interna do componente codificado, as configurações e outros elementos técnicos.

Pfleeger (2004) afirma que muitos tipos de testes são realizados antes da entrega do sistema para o cliente, alguns testes dependem do que está sendo testado, outros do que se pretende saber.

Em literaturas, encontram-se alguns tipos de testes conforme descrito a seguir:

• Teste de Unidade: o teste é realizado em cada componente do programa isoladamente, no qual se verifica se ele funciona de forma adequada aos tipos de entradas esperadas. Normalmente, esse tipo de teste é realizado em um ambiente controlado. Além disso, a equipe verifica as estruturas de dados internas, a lógica e as condições limite para os dados de entrada e saída (PFLEEGER, 2004).

• Teste de Integração: tem o objetivo de provocar falhas associadas às interfaces entre os módulos quando esses são integrados para construir a estrutura do *software* que foi estabelecida na fase de projeto (DIAS NETO, s.d.).

• Teste de Sistema: Dias Neto (s.d.) descreve que o teste de sistema avalia o *software* em busca de falhas utilizando este como se fosse um usuário final. Sendo assim, os testes são realizados com as mesmas condições e com os mesmos dados de entrada que um usuário utilizaria. Rios e Moreira (2013) acrescentam que é nesse estágio que são realizados os testes de carga, performance, usabilidade, compatibilidade, segurança e recuperação.

• Teste de Aceitação: é realizado em conjunto com os clientes e nele o sistema é verificado em comparação com a descrição dos requisitos do cliente (PFLEEGER, 2004).

Existem diversos outros tipos de testes que podem ser executados no processo de desenvolvimento adequando-se à realidade da empresa. Contudo, os testes automatizados têm ganhado bastante notoriedade, conforme afirma Sommerville (2011, p. 147):

> O uso dos testes automatizados tem aumentado consideravelmente nos últimos anos. Entretanto, os testes nunca poderão ser totalmente automatizados, já que testes automáticos só podem verificar se um programa faz aquilo a que é proposto. É praticamente impossível usar testes automatizados para testar os sistemas que dependem de como as coisas estão (por exemplo, uma interface gráfica de usuário), ou para testar se um programa não tem efeitos colaterais indesejados.

Partindo desse pressuposto, verifica-se que o ideal é fazer uma junção das técnicas existentes adequando-se ao processo de desenvolvimento de *software*.

Processo de teste de *software*

Processo

Para avaliar um processo de teste de *software* adequado, levam-se em consideração diversos fatores, incluindo o porte da empresa e a sua realidade. Baseado nesse fato, surgem novos olhares para as metodologias ágeis, que trazem uma forma alternativa no desenvolvimento de *software*. Essas metodologias têm por objetivo orientar o processo para se adequar a um processo mais dinâmico e eficiente.

Normalmente, o processo de testes deve ser baseado em metodologia aderente ao processo de desenvolvimento, em pessoal técnico qualificado, em ambiente e ferramentas adequadas. (RIOS E MOREIRA, 2013).

Pfleeger (2004) afirma que as etapas do processo devem ser planejadas, que o processo de teste tem vida própria no ciclo de desenvolvimento e este pode ser realizado paralelamente com outras atividades de desenvolvimento.

Rios e Moreira (2013) caracterizam o processo de testes em execução das principais etapas e dos seus desdobramentos (subetapas), as quais são o planejamento, os procedimentos iniciais, a preparação, a especificação, a execução, a entrega.

Baseado nesses conceitos, percebe-se que o processo deve ser mais dinâmico e flexível. A monitoração deve ser constante para acompanhar a evolução dos resultados e sugerindo modificação quando necessário.

De acordo com Sommerville (2011), as abordagens de desenvolvimento nas metodologias ágeis levam em consideração o projeto e a implementação como sendo atividades centrais no processo de *software*. Eles incorporam outras atividades, como elicitação de requisitos e testes no projeto e na implementação.

Extreme programming

De acordo com Myers (2004), por volta dos anos 90, Kent Beck iniciou uma nova abordagem ao desenvolvimento de *software* num projeto chamado C3. Essa nova metodologia visava deixar o projeto mais leve, garantir que testes fossem feitos e refeitos, melhorar a comunicação entre os membros da equipe e entre os desenvolvedores e o cliente. Desse projeto, surgiu o *Extreme Programming* (Programação Extrema).

Com relação a isso, Sommerville (2011) afirma que, nesse método, a diferença está na forma como o sistema é testado. Não existe especificação do sistema que possa ser usada por uma equipe de teste externa. Para evitar problemas nos testes, a abordagem XP enfatiza a importância dos testes do programa, incluindo um foco de testes que reduz as chances de erros não identificados na versão atual do sistema. Existem características próprias de testes, que são:

• Desenvolvimento em *test-first*: é escrito primeiramente os testes, depois os códigos.

• Desenvolvimento de teste incremental a partir de cenários: os cenários ou estórias são os requisitos de sistema, e quem os prioriza para ser desenvolvido é o usuário.

• Envolvimento dos usuários no desenvolvimento de testes e validação: nesse modelo de desenvolvimento, o papel do cliente passa a ser fundamental, pois ele ajuda a desenvolver os testes de aceitação, na prática. Em vez de um único teste, é realizada uma bateria de testes de aceitação. Como conseguir o apoio do cliente normalmente é mais complicado, essa é uma grande dificuldade do processo de teste XP.

• Uso de *frameworks* de testes automatizados: os testes automatizados são escritos como componentes executáveis antes que a tarefa seja implementada, com isso, existe sempre um conjunto de testes que podem ser executados rapidamente.

Deve-se observar que todo processo de *software* tem suas falhas e que muitas organizações de *software* usaram, com êxito, a XP. O segredo é reconhecer onde um processo pode apresentar fraquezas e adaptá-lo às necessidades específicas de sua organização (PRESSMAN, 2011).

Scrum

Scrum é um método ágil de desenvolvimento de *software* criado por Jeff Sutherland e sua equipe no início de 1990. Recentemente, foram realizados desenvolvimentos adicionais nos métodos gráficos Scrum por Schwaber e Beedle (PRESSMAN, 2011).

O Scrum pertence à metodologia de desenvolvimento de *software* ágil. Ele considera uma abordagem mais humana ao solucionar os problemas existentes, ao invés de desperdiçar tempo criando documentações extensas e detalhadas que as pessoas acabam não lendo minuciosamente. No Scrum, as equipes trabalham com *sprints*. São realizadas reuniões curtas em que o time verifica quais as decisões que devem ser tomadas e os recursos do *product backlog* que entram nos *sprints*. Elas também decidem quem trabalha nos *sprints* e quanto tempo dura cada tarefa (DIMES, 2014).

Um líder de equipe chamado Scrum Master conduz a reunião e avalia as respostas de cada integrante. A reunião Scrum, realizada diariamente, ajuda a equipe a revelar problemas potenciais o mais cedo possível (PRESSMAN, 2011).

Outro papel fundamental na metodologia é o *Product Owner*, o dono do produto. Fornece o requisito do negócio para a equipe assim como sua ordem de aplicação, ou seja, o *Product Owner* é a interface entre a empresa e os clientes. É ele o ponto de contato para esclarecimento das dúvidas da equipe sobre as regras do produto. Trabalhando em conjunto com a equipe, ele

ajuda a definir a ordem de execução das atividades conforme a necessidade do cliente, definindo também o cronograma para a liberação e fazendo as validações necessárias (RODRIGUES, s.d.).

A equipe, no *framework* Scrum, é multidisciplinar e é composta por pessoas que fazem o trabalho de desenvolvimento e teste do produto. A equipe é responsável pelo desenvolvimento do produto e também tem a autonomia para tomar decisões de como executar o seu trabalho. Os membros da equipe decidem como dividir o trabalho em tarefas e, ao longo da *sprint*, decidem a ordem de execução das tarefas. Nove pessoas na equipe é a quantidade ideal para uma boa comunicação e não afetar a produtividade (RODRIGUES, s.d.).

Práticas de desenvolvimento

TDD - *Test-driven development* (desenvolvimento guiado por testes)

Desenvolvimento guiado por teste é aquele em que se escreve primeiramente os testes para posteriormente escrever o código. O TDD é parte do processo de desenvolvimento ágil, utilizado em metodologias como o XP (Programação Extrema) e sendo uma das técnicas que auxiliam na melhoria de qualidade do processo de desenvolvimento. O TDD torna mais eficiente o processo (ROCHA, s.d.).

Pires (s.d.) ressalta que o processo de desenvolvimento do TDD aborda os parâmetros Red, Green e Refactor:

1. Escrever um teste, mesmo sem ter escrito o código real a ser testado.
2. Executar os testes e acompanhar a falha (Red).
3. Escrever a funcionalidade do sistema que irá ser testada.
4. Testar novamente, agora para passar (Green).
5. Refatorar a funcionalidade e escrever por completo (Refactor).
6. Próxima estória ou caso de uso e iniciar novo teste.

O TDD é um conjunto de técnicas que culminam em um teste de ponta a ponta (Rocha, s.d.).

DDD - *Domain-driven design* (desenvolvimento guiado ao domínio)

Desenvolvimento guiado ao domínio são padrões e princípios que ajudam em seus esforços para construir aplicações que refletem uma compreensão e a satisfação das exigências do seu negócio. Trata da modelagem do domínio real por primeiramente entendê-la completamente e então colocar todas as terminologias, regras, e lógica em uma representação abstrata dentro do seu código, tipicamente em forma de um modelo de domínio. DDD não é um *framework,* mas ele tem um conjunto de blocos ou conceitos que podem ser incorporados em sua solução (SCHISSATO E PEREIRA, s.d.).

O foco é no domínio do *software*, no propósito que o *software* deve atender, é a automatização de um processo de negócio. O DDD traz abordagens de como fazer isto, como atender um domínio complexo de informações. Qualquer abordagem de DDD é muito bem aceita numa metodologia ágil (PIRES, s.d.).

Pires (s.d.) afirma ainda que o DDD é importante em um sistema complexo e não é aconselhável para um sistema simples. Na maioria dos sistemas corporativos, são encontradas diversas regras de negócio e cada uma com sua particularidade e complexidade. Iniciar um projeto usando a abordagem de DDD previne que o sistema cresça cada vez mais de uma forma

não orientada ao domínio.

BDD - *Behavior-driven development* (desenvolvimento guiado por comportamento)

BDD é uma evolução do TDD. De forma explícita, BDD relaciona *Test-Driven Development com Domain-Driven Design*, tornando a relação entre essas duas abordagens consideravelmente mais evidente. BDD colabora para que o desenvolvimento foque na entrega de valor, através da formação de um vocabulário comum, reduzindo a distância entre negócio e tecnologia (ELEMAR JR., s.d.).

BDD se destina a satisfazer as necessidades de ambos os usuários (técnicos e de negócio) BDD pode ser realizado utilizando *frameworks* de testes de unidade, ou com *frameworks* específicos de BDD que têm surgido em diversas linguagens (SCHISSATO; PEREIRA, s.d.).

BDD associa os benefícios de uma documentação formal, escrita e mantida pelo negócio, com testes de unidade que demonstram que essa documentação é efetivamente válida. Na prática, isso garante que a documentação deixa de ser um registro estático, que se converte em algo gradualmente ultrapassado, em um artefato vivo que reflete constantemente o estado atual de um projeto (ELEMAR JR., s.d.).

ATDD - *Acceptance test-driven development* (desenvolvimento guiado por testes de aceitação)

As equipes Scrum aprenderam a diminuir o fluxo de trabalho que passa por uma sprint executando o desenvolvimento baseado em teste de aceitação. No ATDD, o trabalho ocorre em resposta a testes de aceitação. O ATDD pode ser considerado como análogo ao TDD (GRIEBLER, s.d.).

FDD - *Feature driven development* (desenvolvimento guiado por funcionalidades)

FDD serve para gerenciar e desenvolver projetos de *software* através de um conjunto de atividades simplificadas, de maneira a estimular o compartilhamento do conhecimento acerca do *software* e da criação de bons códigos, permitindo então que o principal objetivo da FDD, resultados frequentes, tangíveis e funcionais, seja alcançado num projeto de desenvolvimento de *software* (GRIEBLER, s.d.).

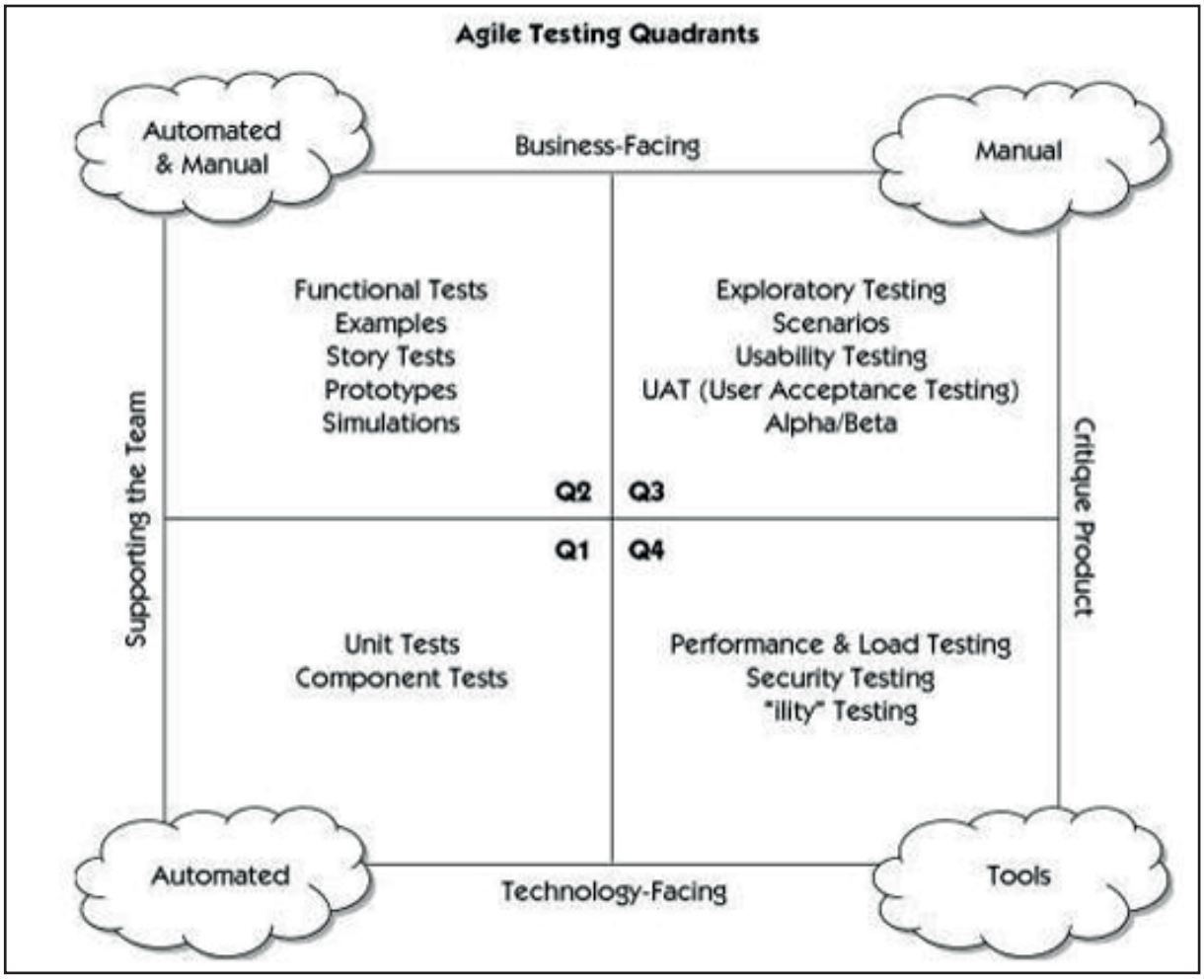
Modelo de processo de teste de *software* para pequenas empresas de desenvolvimento de sistemas

Na metodologia tradicional, as equipes são divididas por área. Cada área representa uma função distinta e não há uma integração entre elas. No Scrum, todas as equipes (Configuração/ Instalação, Negócios, Teste, Desenvolvimento, entre outros) fazem parte de um mesmo Time (FERREIRA, RAMOS; LAGARES, s.d.).

Papo (s.d.) afirma que, nesse modelo, o analista de testes passa por uma grande transformação. Ele deixa de ser reativo para ter um papel fundamental na interação com os desenvolvedores, analistas de negócio e clientes.

Pode-se observar que, ao implantar o modelo de desenvolvimento ágil, o teste está bem mais dinâmico, conforme Figura 1:

Figura 1. *Agile Testing Quadrants*



Fonte: Crispin e Gregory (2009)

Conforme Crispin e Gregory (2009):

• Q1 – o quadrante 1 mostra o foco na arquitetura. Neste estágio, são realizados os testes unitários e de componentes, que são os próprios desenvolvedores que realizam. O papel do analista de testes é apoiar os desenvolvedores sempre que necessário, auxiliando também a elaborar os testes automatizados.

• Q2 – no quadrante 2, os testes focam no negócio. Neste momento, os testes executados são os funcionais, isto é, são testes e cenários de exemplo realizados pelos testadores em conjunto com os clientes, usuários e analistas de negócio. Uma parte desses testes serão automatizados antes ou em paralelo com o desenvolvimento do cenário. O diferencial nesses testes não é encontrar o maior número de erros e sim auxiliar os clientes e desenvolvedores.

• Q3 – no quadrante 3, são os testes que evidenciam o produto. Esses são os testes de aceitação feitos na homologação do produto, que podem ser chamados também de exploratórios. Nesse momento, os testes têm o objetivo de encontrar defeitos. Analistas de testes possuem técnicas para encontrar defeitos que poucos desenvolvedores conhecem.

• Q4 - testes que focam na arquitetura e criticam o produto. Nessa etapa, são realizados os testes de performance, de carga e de segurança. Esses são executados pelos analistas de

testes e costumam ser feitos quando parte do *software* já está terminado e antes da sua entrega.

O analista de teste no Scrum assume vários papéis, como líder, arquiteto e automatizador. Em cada fase do projeto, ele "coloca o chapéu" necessário. Um dos papéis que o analista de teste pode exercer, é o de *Product Owner* (PO). O PO é o responsável pela visão do produto, ou seja, a representação da sua necessidade é o que deve ser satisfeito ao fim do projeto (FERREIRA, RAMOS, LAGARES, s.d.).

Ferreira, Ramos e Lagares (s.d.) afirmam que existem vários benefícios com a participação do time de teste no Scrum:

- Integração do time.
- Apoio de quem está desenvolvendo código durante a execução dos testes.
- Apoio de quem está testando código durante a codificação.
- Participação mais direta e ativa do profissional.
- Profissionais que estão desenvolvendo código interessados em aprender sobre teste.
- Profissionais que estão testando código interessados em aprender sobre programação.
- Agilidade, interação com testes.
- Acompanhamento de defeitos pelo profissional que está testando o *software*.
- Analistas de teste deixam de ser reativos para serem proativos.

Levando em consideração os quadrantes que Crispin e Gregory (2009) definem para o processo de teste de Scrum, Ferreira, Ramos e Lagares (s.d.) reafirmam que este pode ser adequado a empresas de pequeno porte da seguinte maneira:

• *Planning Meeting*: inicia-se com a estimativa, o analista de teste participa desse processo. Nesse momento, é definida a meta da *sprint* e os itens prioritários do *Product Backlog.* O time seleciona o que vai ser feito. O analista de teste participa dessa fase garantindo que os itens selecionados estejam de acordo com a meta do projeto e teste, analisando indicadores de desempenho, revisando a estimativa, gerenciando os riscos encontrados, tirando dúvidas com o *Product Owner* e definindo quais serão os tipos de teste (sistema, aceitação, regressão) necessários.

• Na segunda parte da *Planning Meeting*, o time colhe mais detalhes dos itens do *Select Product Backlog* e os divide em tarefas gerando o *Sprint Backlog*. Nesse momento, cada membro do time escolhe as tarefas que deseja executar durante a *sprint*. O analista de teste participa desta fase definindo o nível de regressão do teste automático (o desenvolvedor pode utilizar a prática do TDD - Desenvolvimento Dirigido a Testes), atualizando a matriz de teste por funcionalidade e atualizando o mapa mental.

• O quadro de acompanhamento é preparado após definição da *sprint*, o qual pode ser alterado de acordo com a necessidade. O *Scrum Master* é o responsável por facilitar o trabalho do time, garantindo uma boa aplicação do *Scrum*.

• Durante a execução da *Sprint*, o analista de teste monta e configura o ambiente de teste, executa testes (pode utilizar as técnicas – ATDD e AFDD), automatiza casos e tarefas de teste, auxilia os desenvolvedores na elaboração dos testes unitários automáticos, evidencia os resultados, acompanha os defeitos encontrados. O analista de teste é a pessoa que aprova, nada é considerado pronto em uma sprint até que ele diga que está. Ele é o responsável por ficar focado na meta da *sprint*.

• Na *Daily Scrum* (reunião diária de 15 minutos), o time visualiza como está o andamento da meta e realiza o planejamento do dia seguinte. O *Scrum Master* é o facilitador e o analista de testes contribui com relatórios, evidências, gráficos, lista de defeitos, lista de impedimentos

e o quadro Kanban atualizado. O analista de testes participa das reuniões de revisão e reunião de retrospectiva.

• Na reunião de revisão, o *Product Owner* avalia se a meta foi alcançada e a reunião de retrospectiva é facilitada pelo Scrum Master, na qual são apresentadas as lições aprendidas.

No Scrum, as equipes são pequenas, sendo uma vantagem para aplicação em empresas de pequeno porte, pois permite que se tenha um processo de teste definido.

Considerações finais

Levando em consideração que empresas de pequeno porte normalmente não têm um processo definido e muitas não possuem uma equipe de testes, surgiu a necessidade de verificar, baseados nas metodologias existentes, um processo que se adequasse a esse cenário.

Com embasamento nessa proposta, verificou-se, através das abordagens encontradas na literatura, que é possível aplicar um processo de teste em empresas de pequeno porte, pois os benefícios são aparentes. Pode-se constatar também que uma das metodologias que mais se adéquam a essa realidade é o processo baseado no Scrum com foco nos Quadrantes Ágeis de Teste (*Agile Testing Quadrants*) ou seja, um processo ágil em que o analista de teste participa ativamente. Se a empresa já possuir um processo, mesmo que empírico, é possível e simples adequar esta proposta à realidade da empresa pois, ela somente agrega benefícios e padroniza os passos a serem seguidos, entregando um produto muito mais estável e com qualidade superior.

Referências

BARTIÉ, Alexandre. **Garantia da Qualidade de Software**. Rio de Janeiro: Editora Campus, 2002.

BERNI, Jean Carlo Albieri. **Gestão para o processo de desenvolvimento de software científico, utilizando uma abordagem ágil e adaptativa na microempresa**. 2010. Disponível em: <http://livros01.livrosgratis.com.br/cp127300.pdf>. Acesso em: 19 fev. 2015.

CRISPIN, Lisa; GREGORY, Janet. **Agile Testing**: A Practical Guide for Testers and Agile Teams. Boston: Pearson Education Inc., 2009.

DIAS NETO, Arilo Cláudio. **Engenharia de Software Magazine** – Introdução a Teste de Software. Disponível em: <http://www.comp.ita.br/~mluisa/TesteSw.pdf>. Acesso em: 20 fev. 2015.

DIMES, Troy. **Scrum Essencial**. Canadá: Babelcube Inc, 2014.

ELEMAR JR. **BDD na prática** – Parte 1 – Conceitos básicos e algum código. Disponível em: <http://elemarjr.net/2012/04/11/bdd-na-prtica-parte-1-conceitos-bsicos-e-algum-cdigo/>. Acesso em: 3 mar. 2015.

ENGHOLM JR, Hélio. **Engenharia de software na prática**. São Paulo: Editora Novatec, 2010.

FAPEG. **Manual do Processo de Teste de Software**. UFG. Disponível em: <http://www.inf.

ufg.br/~auri/freetest/manualFreeTest.pdf>. Acesso em: 20 set. 2014.

FERREIRA, R. E. P.; RAMOS, S. E.; LAGARES, V. C. **Scrum no Teste de Software**. Disponível em: <http://www.omegabrasil.com.br/userfiles/produtos_categorias_14_catalogo.pdf> Acesso em: 3 mar. 2015.

GRIEBLER, Fabricio. **DDD, FDD, TDD, ATDD, BDD**. Que tal começar não confundindo as siglas? Disponível em: <http://blog.fasagri.com.br/?p=113>. Acesso em: 4 mar. 2015.

MYERS, Glenford J. **The Art of Software Testing**. 2. ed. Nova Jérsei: John Wiley & Sons, 2004.

PAPO, Jose Paulo. **O papel do analista de Testes dentro dos processos ágeis**. Disponível em: <http://josepaulopapo.blogspot.com.br/2009/09/testes-agil-papel-analista.html>. Acesso em: 22 fev. 2015.

PFLEEGER, Shari Lawrence. **Engenharia de software, teoria e prática**. 2. ed. São Paulo: Pearson, 2004.

PIRES, Eduardo. **DDD, TDD, BDD, afinal o que são essas siglas?** Disponível em: <http://eduardopires.net.br/2012/06/ddd-tdd-bdd/>. Acesso em: 3 mar. 2015.

PRESSMAN, Roger S. **Engenharia de Software, uma abordagem profissiona**l. 7. ed. Porto Alegre: AMGH, 2011.

RIOS, Emerson; MOREIRA, Trayahú. **Teste de software**. 3. ed. Rio de Janeiro: Alta Books, 2013.

ROCHA, Fabio Gomes. **Introdução ao desenvolvimento guiado por teste (TDD) com JUnit**. Disponível em: <http://www.devmedia.com.br/introducao-ao-desenvolvimento-guiado-por-teste-tdd-com-junit/26559>. Acesso em: 3 mar. 2015.

RODRIGUES, Luiza. **Quais são os papéis do Scrum?** Disponível em: <http://blog.myscrumhalf.com/2011/07/quais-sao-os-papeis-do-scrum-faq-scrum/>. Acesso em: 22 fev. 2015.

SARTORI, Lucia Emi Shiraisi. **Melhoria do processo de teste para pequenas empresas**. Marília: Dissertação de Mestrado. Departamento de Ciência da Computação, Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha. 2005.

SCHISSATO, Jéssica; PEREIRA, Rodolfo. **TDD, DDD e BDD – Práticas de desenvolvimento**. Disponível em: <http://www.princiweb.com.br/blog/programacao/tdd/tdd-ddd-e-bdd-praticas-de-desenvolvimento.html>. Acesso em: 3 mar. 2015.

SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Pearson, 2011.

Artigo recebido em 15/06/16. Aceito em 18/08/16.