COMBINAÇÃO DE MÉTODOS ÁGEIS NO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE: UM ESTUDO DE CASO

Deisy Braz dos Santos[1]
Paulo Roberto Córdova[2]

RESUMO

Este artigo apresenta o resultado de uma experiência de aplicação de diferentes métodos ágeis no processo de desenvolvimento de software em uma empresa de software da cidade de Caçador, SC. Nesse sentido, traz como principal objetivo, avaliar a eficácia da implantação de práticas ágeis combinadas. Sabe-se que utilização de métodos ágeis pode contribuir para o aumento da produtividade das equipes e da qualidade dos projetos e produtos. Nesse sentido, para realizar esta pesquisa, foi necessário um estudo sobre as metodologias tradicionais e ágeis mais utilizadas atualmente. A partir deste estudo, pôde-se avaliar cada um dos métodos, considerando seus prós e contras e elencar como cada um deles poderia contribuir para o aprimoramento do processo de desenvolvimento na empresa onde foi realizado o presente estudo de caso. Depois disso, boas práticas de desenvolvimento ágil puderam ser aplicadas e avaliadas. Por fim, uma análise dos resultados mediante extração e compilação de dados de uma ferramenta de gestão de incidentes e a aplicação de um questionário à equipe de desenvolvimento, permitiu mensurar a eficácia da implantação proposta e a percepção subjetiva da equipe, respectivamente. Neste sentido, foi possível concluir, entre outras coisas, que embora os períodos analisados tanto antes da implantação quanto depois, tenham sido relativamente curtos, as práticas ágeis contribuíram de forma significativa para a melhoria dos processos, tornando a equipe mais colaborativa e disposta a aderir a novas práticas.
Palavras-Chave: metodologias ágeis, software house, engenharia de software.

ABSTRACT

This article presents the results of an experience of applying different agile

[1] Especialista em Engenharia de Software – Universidade Alto Vale do Rio do Peixe UNIARP. E-mail: deisy_braz@yahoo.com.br.
[2] Mestrando em Desenvolvimento e Sociedade – Universidade Alto Vale do Rio do Peixe UNIARP. E-mail: cordova@uniarp.edu.br.

methods in the software development process at a software company in the city of Caçador, SC. In this sense, it has as main objective, to evaluate the effectiveness of the implementation of combined agile practices. It is known that the use of agile methods can contribute to increase the productivity of the teams and the quality of projects and products. In this sense, to carry out this research, it was necessary to study the traditional and agile methodologies most used today. From this study, it was possible to evaluate each of the methods, considering their pros and cons and listing how each one of them could contribute to the improvement of the development process in the company where the present case study was carried out. After that, good agile development practices could be applied and evaluated. Finally, an analysis of the results by extraction and compilation of data from an incident management tool and the application of a questionnaire to the development team allowed us to measure the effectiveness of the proposed implementation and the subjective perception of the team, respectively. In this sense, it was possible to conclude, among other things, that although the periods analyzed both before and after implementation were short, agile practices contributed significantly to improving processes, making the team more collaborative and willing to adhere to new practices.
Keywords: Agile, software house, software engineering.

INTRODUÇÃO

Apesar da constante evolução dos computadores e das tecnologias para desenvolvimento de *software*, construir um produto com o mínimo de erros possível, dentro do prazo e de acordo com os custos previamente definidos ainda é um desafio para profissionais da área.

Nesse sentido, as metodologias ágeis têm sido recomendadas como uma opção às abordagens tradicionais, também conhecidas como metodologias pesadas, para o desenvolvimento de *software*. Ademais, em função de as metodologias tradicionais serem orientadas a planejamento e documentação, recomenda-se que sejam aplicadas apenas em projetos em que os requisitos sejam estáveis e previsíveis. No que diz respeito à projetos de desenvolvimento de *software*, entretanto, onde os processos de desenvolvimento devem acompanhar o dinamismo das variações nos requisitos do produto, as metodologias pesadas podem constituir uma restrição que limita a criatividade e a produtividade das equipes de desenvolvedores. Essa característica pode levar a consequências negativas, impactando na qualidade do produto final e excedendo os prazos e os

custos estimados.

Nesse contexto, as metodologias ágeis surgiram com a intenção de priorizar mais as pessoas do que os processos, despendendo menos tempo com documentação e mais tempo com a resolução dos problemas do projeto.

Assim, com o intuito de abordar o tema de forma mais clara e objetiva, no decorrer deste artigo, dentre as metodologias tradicionais, serão apresentados o modelo cascata e modelo de prototipação. E em relação às metodologias ágeis, serão explanados o *Scrum* e a *Extreme Programming*, que vem se destacando na indústria de *software* com grande número de adeptos. Além disso, será também descrito o método de gestão de atividades conhecido como *Kanban*, que estabelece práticas dinâmicas e colaborativas, além de ser comumente associado às metodologias supracitadas.

É importante também ressaltar que este artigo foi desenvolvido a partir de um estudo exploratório bibliográfico sobre as metodologias tradicionais e ágeis para projetos de software, apontando suas características, bem como seus prós e contras. Em seguida, como estudo de caso, foi realizada a implantação combinada de dois diferentes métodos ágeis em uma *software house* do município de Caçador.

A avaliação dos resultados foi feita através da análise dos dados extraídos de uma ferramenta de gestão de incidentes e serviços, que serve como *backlog* do produto e de um questionário aplicado aos membros da equipe de desenvolvimento, onde pôde-se observar que houve uma significativa melhora nos processos de desenvolvimento e na satisfação e colaboração entre os membros.

UM RESUMO HISTÓRICO SOBRE A ENGENHARIA DE SOFTWARE

O ciclo de vida do desenvolvimento de *software* é composto por um conjunto complexo de processos que no decorrer dos anos foi sendo abordado por diferentes paradigmas. Inicialmente era considerada uma atividade artesanal, porém, na medida em que os *softwares* se tornaram mais presentes no cotidiano da sociedade, houve um exponencial crescimento da demanda e da complexidade das soluções a serem desenvolvidas.

Na década de 1970, observou-se que os projetos de sistemas

demonstraram uma queda brusca de qualidade, com códigos de difícil manutenção, além de ultrapassar os prazos e orçamentos estabelecidos devido à falta de maturidade e padronização nos processos. Foi neste cenário que emergiu o que ficou conhecido como a Crise do Software. Deste modo, na tentativa de conter esta crise e manter a qualidade dos produtos e processos de desenvolvimento, nasceu a Engenharia de Software.

A Engenharia de Software é, segundo a norma IEEE Std 610.12-190 (IEEE, 1990):

> A aplicação de uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, operação e manutenção de software, isto é, a aplicação de conceitos de engenharia para software. (IEEE, 1990, p.67).

Esta definição deixa clara a abrangência desta área do conhecimento, mostrando que não só todo o processo de desenvolvimento é por ela coberto, como também os processos relacionados ao uso e à manutenção dos produtos de *software*.

Nesse sentido, várias metodologias de desenvolvimento foram sendo apresentadas pela engenharia de software no decorrer dos anos. Dentre elas as metodologias estruturadas, também conhecidas como tradicionais, que são pouco flexíveis e implementam uma vasta gama de documentação ao longo da execução do projeto e as metodologias ágeis, cujo principal propósito é solucionar os problemas relacionados à prazos, custos e qualidade dos produtos, dividindo o projeto em pequenas etapas e promovendo a comunicação entre os membros da equipe, encorajando, sobretudo, a produção da documentação apenas quando necessária e útil.

Assim, segundo Hirama (HIRAMA, 2012), a importância da utilização de métodos de desenvolvimento de *software* consiste no fato de estabelecer, através de suas notações, a comunicação entre os membros da equipe de forma organizada e padronizada, facilitando a manutenção do *software* e também a incorporação de novos membros no time, visando a melhoria da qualidade do produto final.

O recente advento do surgimento das novas metodologias de desenvolvimento de *software* ocasionou a divisão das destas em dois principais grupos, sendo as metodologias tradicionais, baseadas no projeto e gerando

documentação para nortear o desenvolvimento, e as metodologias ágeis, baseadas no código, utilizando-se de menor número de documentos e processos mais simplificados.

PROCESSOS, MODELOS E MÉTODOS TRADICIONAIS DA ENGENHARIA DE SOFTWARE

As metodologias de desenvolvimento tradicionais, também conhecidas como metodologias pesadas, foram largamente utilizadas até o início dos anos 90. Durante este período, o *software* era totalmente planejado e detalhadamente documentado antes de ser implementado.

Assim, uma das características das metodologias tradicionais é o grande volume de documentação gerada durante o planejamento do produto. Esta característica, por sua vez, tem sido apontada como responsável por promover atrasos e aumento dos custos dos projetos.

Desse modo, além do grande volume de documentos que são gerados a partir dos modelos tradicionais, a inflexibilidade dos métodos dificulta possíveis alterações que são comuns durante a execução do projeto. Assim, alguns especialistas recomendam a aplicação dos modelos clássicos apenas para os projetos que possuem os requisitos bem compreendidos e definidos, e desencorajam o seu uso em grandes empreendimentos devido ao alto risco de não cumprimento dos prazos e dos custos.

Dentre os diversos modelos tradicionais de desenvolvimento de *software*, pode-se citar, por exemplo, o modelo cascata e também o modelo de prototipação.

METODOLOGIAS ÁGEIS DE DESENVOLVIMENTO DE SOFTWARE

O grande desafio que permeia o desenvolvimento de *software* é a sua complexidade e imprevisibilidade. Projetos de *software* precisam ser capazes de responder rapidamente às constantes alterações de escopo e de cenários, e, para isto, precisam dispor de eficientes mecanismos de gestão, monitoramento e controle dos processos.

Nesse sentido, as metodologias ágeis podem ser uma boa alternativa, pois

dentre as suas características, pode-se citar a adaptabilidade, que através de constantes *feedbacks*, permite que de acordo com o andamento do projeto e diante das mudanças nos requisitos, medidas corretivas ou preventivas possam ser tomadas rapidamente.

No que concerne às atividades de codificação, ao adotar-se métodos ágeis de desenvolvimento, a necessidade de executar a etapa de integração dos módulos deixa de existir, visto que eles são continuamente integrados durante o próprio processo. Além disso, a execução de testes é constante, o que impacta diretamente na qualidade do produto final. Outra característica das metodologias ágeis são as entregas frequentes de partes funcionais do *software* para o cliente final, o que permite que este possa avaliar necessidades de mudança nos requisitos de forma dinâmica. Esta abordagem permite que o cliente faça parte do processo de desenvolvimento, aumentando consideravelmente as chances de sucesso do projeto.

A expressão "Metodologias Ágeis" ficou amplamente conhecida quando um grupo de especialistas em desenvolvimento de *software*, formado por dezessete membros, reuniu-se para discutir como poderiam melhorar o desempenho de seus projetos.

A partir disso, foi criada a Aliança Ágil, e estabelecido o que ficou conhecido como o Manifesto para Desenvolvimento Ágil de Software, que dispõe dos conceitos e princípios compartilhados por todos os métodos, os quais implicam em enaltecer os seguintes itens:

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

Entre as metodologias ágeis mais conhecidas e utilizadas estão o *Scrum*, e a *Extreme Programming*, que serão apresentadas a seguir.

SCRUM

De acordo com Schwaber e Sutherland (SCHWABER; SUTHERLAND, 2014), o *Scrum* é um *framework* para gerenciamento de projetos de *software*, e vem sendo utilizado desde o início dos anos 90, pregando uma visão estratégica onde a

equipe de desenvolvimento trabalha em unidades menores, buscando alcançar um objetivo comum. O Guia salienta ainda, que o *Scrum* não é um processo ou uma técnica para desenvolvimento de *software*, e sim um *framework*, no qual podem ser aplicados diversos processos ou técnicas.

Um dos pontos de destaque do *Scrum* é que durante o processo de desenvolvimento do produto de *software*, os clientes podem modificar seus requisitos sem a implicação de grandes impactos no projeto, dada a sua grande flexibilidade e capacidade de adaptação.

Assim, conforme afirmam Schwaber e Sutherland (SCHWABER; SUTHERLAND, 2014), o Scrum é embasado em teorias empíricas, utilizando uma abordagem incremental e interativa para aprimorar a estimativa de prazos, orçamentos e o controle sobre os riscos. Nessa abordagem, três pilares sustentam a implementação de controles de processo, sendo eles: transparência: todos os envolvidos devem ter a visão de tudo o que está acontecendo e possuir o mesmo entendimento sobre o que está sendo visto; inspeção: todos os aspectos do processo devem ser inspecionados com frequência suficiente para que seja possível identificar desvios indesejados; e adaptação: se após a inspeção for identificado que um ou mais aspectos estão incoerentes, o ajuste deve ser feito o mais rápido possível para evitar desvios maiores.

No que diz respeito à formação da equipe de desenvolvimento no *Scrum*, existem três papéis distintos a serem desempenhados, sendo eles: *Scrum Master*, *Product Owner* e o próprio time de desenvolvimento. Segue a seguir uma breve explicação sobre cada um deles:

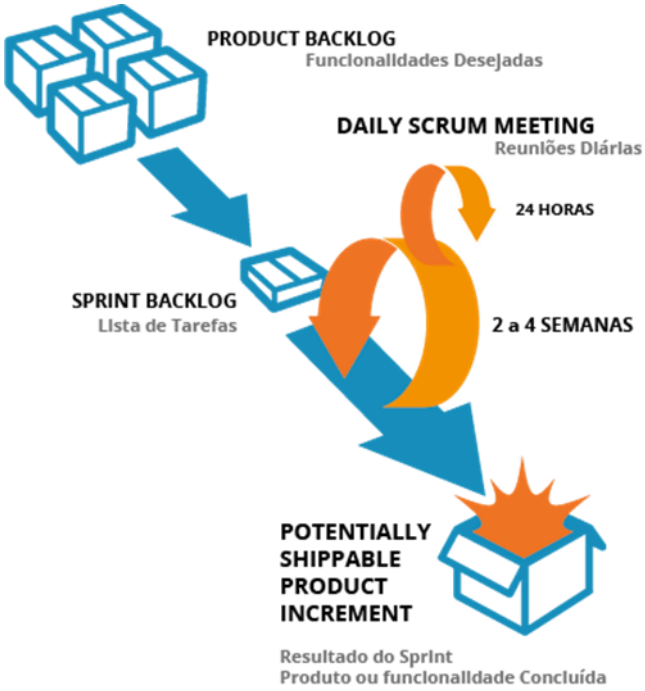
a) Scrum Master: responsável por assegurar que o processo seja assimilado e seguido, além de remover os impedimentos que possam comprometer o bom andamento do trabalho; protege o time de intervenções externas;

b) Product Owner: responsável por potencializar o trabalho que o time de *Scrum* executa, priorizando os requisitos e realizando a interface entre o time de desenvolvimento e o cliente; é responsável pelo retorno financeiro do produto; pode mudar os requisitos a cada *Sprint*; acata ou reprova o resultado de cada ciclo;

c) Time de desenvolvimento: o time é multifuncional, formado pelos desenvolvedores, testadores, administradores de bancos de dados e demais

profissionais capacitados a converter os requisitos do *Product Owner* em partes entregáveis ao final do *Sprint*.

Figura 4: Ciclo do *Scrum*



Fonte: (BRQ, 2016).

Como pode ser observado na Figura 4, o ciclo de desenvolvimento do *Scrum* é iniciado a partir de uma abstração de alto nível acerca do produto que irá ser desenvolvido. Nesse cenário, o *Product Owner* deverá ser o responsável por estabelecer e gerenciar o escopo do produto, organizando-o em forma de requisitos funcionais e não funcionais e priorizando-os em uma lista chamada *Product Backlog*. Esta priorização de atividades deve levar em conta a entrega sistemática de valor, ou seja, o que é mais importante para o cliente deverá ter prioridade no desenvolvimento.

Seguindo agora por meio de uma descrição lógica sobre como funciona o *scrum,* o próximo passo deverá ser organizar o *Product Backlog* em *realeses*. Esta divisão pode implicar em alterações nos requisitos e regras de negócio, e, consequentemente, no prazo de transformação desta lista de requisitos em produto entregável.

A execução do trabalho, nesta metodologia, é realizada através de ciclos

chamados de *Sprints*, que podem durar, preferivelmente, entre duas ou quatro semanas. No decorrer de cada *Sprint*, o time de desenvolvimento deve realizar um incremento de valor no produto, cuidando para que este seja devidamente testado, de modo que seja possível entregá-lo ao *Product Owner* já em condições de uso. Nesse contexto, é de suma importância que o *Product Owner* perceba o valor que está sendo entregue, ou seja, as funcionalidades implementadas durante a *Sprint*, devem representar algo importante e utilizável para o cliente.

Com relação aos eventos do processo, cada *Sprint* deve iniciar com uma reunião de planejamento, onde o *Product Owner* e o time de desenvolvimento definem o que deverá ser desenvolvido durante o próximo ciclo de trabalho. Nesse processo, cada item do *Product Backlog* em que a equipe irá trabalhar deverá ser decomposto em tarefas que irão compor o *Sprint Backlog*. Durante esta reunião de planejamento, o time compromete-se a implementar as tarefas enquanto o *Product Owner,* responsabiliza-se por não trazer novos requisitos ao produto durante o *Sprint.* Vale ressaltar que as variações de requisitos devem ser incentivadas, porém, apenas fora do *Sprint,* visto que a equipe deve permanecer concentrada no objetivo do ciclo de trabalho em andamento.

Outro importante evento que deverá ser realizado pelo time, é o que se chama no *Scrum*, de reunião diária. Nesse sentido, como o nome sugere, diariamente a equipe deverá se reunir por aproximadamente quinze minutos, com o objetivo de sincronizar e alinhar informações e atividades entre todos os membros da equipe e comunicar ao *Scrum Master* sobre possíveis impedimentos na execução das atividades do dia. Durante a reunião diária, o time deve responder basicamente a três perguntas, sendo elas: O que eu fiz ontem?; O que pretendo fazer hoje?; e existe algum impedimento que possa restringir a execução do meu trabalho hoje?;

KANBAN

Com o intuito de organizar e visualizar as tarefas de forma simples, o time pode dispor de um quadro de atividades, chamado de *Kanban.*

A utilização do *Kanban* possibilita que o andamento do trabalho seja visualizado por todos os membros da equipe, pois deve ser atualizado a cada vez que algum membro do time assume ou evolui a execução de uma tarefa. Assim, o

quadro de atividades em questão, poderá ser dividido em várias colunas, como por exemplo, a coluna *Sprint Backlog,* onde serão informadas todas as tarefas que deverão ser realizadas durante o *Sprint*; a coluna 'Fazendo', que relacionará as tarefas que estarão em andamento; a coluna 'Teste', onde estarão as tarefas já realizadas e que serão submetidas aos testes; e por fim, a coluna 'Concluído', que listará todas as tarefas devidamente implementadas e já testadas.

EXTREME PROGRAMMING

O método ágil *Extreme Programming*, mais conhecido pela sigla XP, foi criado em 1996, por Kent Beck, no Departamento de Computação da montadora de carros *Crysler*. Assim, de acordo com Gomes (GOMES, 2013), o XP é um dos métodos ágeis que melhor ampara os aspectos técnicos do desenvolvimento de *software* como codificação, design e testes.

O XP possui inúmeras características em comum com o *Scrum*, entre elas, a filosofia de programação enxuta, indicada para pequenas e médias equipes de desenvolvimento que dispõe de especificações vagas e instáveis para o produto.

Segundo Wells (WELLS, 2013), o *Extreme Programming* é capaz de aprimorar o desenvolvimento de projetos de *software* a partir de cinco valores, que podem ser considerados uma extensão natural dos nossos valores pessoais à valores coorporativos, sendo eles: comunicação, simplicidade, *feedback*, coragem e respeito.

a) Comunicação: presa por manter o melhor relacionamento possível entre clientes, desenvolvedores e o gerente de projeto;

b) Simplicidade: resume-se a criar apenas o que é necessário, evitando criar funcionalidades que possam ser importantes no futuro, dando ênfase aos requisitos atuais;

c) Feedback: o desenvolvedor terá acesso à informações sobre o cliente e também sobre o código, possibilitando que não conformidades sejam rapidamente reconhecidas e corrigidas. Por meio do *feedback* constante, o cliente terá regularmente uma parte do *software* totalmente funcional para validar. Quanto ao *feedback* do código, através dos testes, é possível identificar erros individuais e integrados. Desse modo, o *software* final tende a corresponder às expectativas do

cliente;

d) Coragem: como a XP é fundamentada em premissas que contrariam as metodologias tradicionais, é primordial que todos os membros da equipe tenham coragem para adotá-las e enfrentar seus obstáculos pessoais, como por exemplo, a dificuldade de comunicação;

e) Respeito: é o valor que dá suporte aos demais. Todos os membros da equipe são valorizados e se importam uns com os outros. É fundamental que a equipe saiba ouvir, compreender e respeitar o ponto de vista de cada um dos membros para que o projeto seja bem sucedido.

De acordo com Pressman, a XP adota uma abordagem orientada a objetos como paradigma de desenvolvimento e envolve quatro atividades metodológicas: planejamento, projeto, codificação e testes. Além disso, é também formada por cinco princípios: *feedback* rápido, presumir simplicidade, mudanças incrementais, abraçar mudanças e trabalho de alta qualidade.

Segundo Beck apud Soares (BECK, 1996 apud SOARES, 2004), para adotar os valores e princípios de desenvolvimento, o *Extreme Programming* sugere um conjunto de 12 práticas, onde é possível notar a coesão entre elas, pois os pontos fracos de cada uma são compensados pelos pontos fortes das demais.

a) Planejamento: resume-se em definir o que será feito e o que será adiado no projeto, combinando prioridades do negócio e estimativas técnicas;

b) Entregas frequentes: o desenvolvimento do *software* deve ser simples, e de acordo com o surgimento de novos requisitos, este deve ser atualizado;

c) Metáfora: são as descrições do *software* sem os termos técnicos com o intuito de facilitar a compreensão do cliente;

d) Design Simples: o *software* deve ser o mais simples possível e satisfazer os requisitos atuais;

e) Cliente presente: com a intenção de evitar atrasos e até mesmo o desenvolvimento incorreto do *software*, é importante que o cliente esteja disponível para sanar possíveis dúvidas nos requisitos;

f) Testes: com XP, os testes são escritos antes do código propriamente dito;

g) Semana de 40 Horas: a XP segue a premissa de que não se deve fazer

horas extras com frequência. Porém, caso seja necessário fazê-las pela segunda semana consecutiva, significa que há um sério problema a ser resolvido, não, porém com o aumento de horas trabalhadas e sim com melhor planejamento;

h) Propriedade Coletiva: o código do projeto pertence a todos os membros da equipe, de modo que qualquer um pode incrementar um código, desde que execute os testes;

i) Programação em pares: todo o código é desenvolvido em duplas, onde é aplicada a técnica na qual dois programadores trabalham em um mesmo problema, ao mesmo tempo e em um mesmo computador;

j) Padronização do código: é importante haver a padronização do código para que o mesmo possa ser compartilhado e compreendido por todos os programadores;

k) Refatoração: deve ser realizada exclusivamente quando de fato necessária, quando a dupla perceber que é possível simplificar um módulo sem que haja prejuízo nas funcionalidades;

l) Integração contínua: é a prática de construir o *software* várias vezes ao dia, possibilitando uma constante sincronia entre os desenvolvedores. Aconselha-se a aplicação desta prática para evitar surpresas e garantir que o sistema esteja sempre funcionando a cada nova integração. Deste modo é possível que os defeitos sejam mais rapidamente identificados e corrigidos.

Segundo seu criador, Kent Beck (BECK, 1996), dentre os problemas da XP está a resistência à programação em pares e também às equipes geograficamente separadas. Isto porque o propósito do princípio de comunicação é prover um melhor relacionamento entre clientes e desenvolvedores, preferencialmente, conversas mais pessoais a outros meios.

COMBINAÇÃO DE METODOLOGIAS ÁGEIS PARA MELHORIA DO PROCESSO DE DESENVOLVIMENTO

Antes desta pesquisa, o processo de desenvolvimento de *software* na empresa onde foi realizado o presente estudo de caso, não contava com qualquer *framework* ou metodologia de desenvolvimento definida. Desse modo, o produto

era incrementado a medida em que eram realizadas solicitações de clientes ou quando se faziam necessários ajustes ou correções.

Nesse cenário, a partir de uma ferramenta que registra as solicitações de correções e implementações no software, também chamadas de *tickets*, foi possível extrair alguns dados relevantes, como pode ser analisado na Tabela 1.

Tabela 1: *Tickets* do período de Outubro/2015 a Janeiro/2016

Mês	Outubro 2015	Novembro 2015	Dezembro 2015	Janeiro 2016
Tickets abertos	17	18	10	31
Tickets resolvidos	14	16	4	12

Fonte: (o autor, 2016).

Com base nestes dados, pode-se observar que no cenário anterior a esta pesquisa, em média 62,5% dos tickets eram resolvidos no mesmo mês em que eram abertos, e que em meses com maior número de tickets, como é o caso de janeiro de 2016, somente 38,7% dos incidentes foram resolvidos dentro do mesmo mês. Além disso, foi possível ainda, identificar que o tempo médio para resolução das solicitações era de aproximadamente 9 dias, e que dentro do período observado, um total de 4 *tickets* foram reincidentes.

Este cenário demonstra uma baixa eficiência da equipe, principalmente levando em conta que a quantidade de tickets abertos é também relativamente baixa. Assim, pode-se inferir da análise destes dados, que a equipe não estava pronta para um aumento significativo na demanda de projetos, ou seja, não havia escalabilidade.

A análise acima, foi realizada antes da implementação dos métodos ágeis combinados, que se deu pela adoção de práticas do *Scrum* e do *Kanban,* potencializadas por processos advindos da *Extreme Programming.* Esta combinação de boas práticas, por sua vez, teve como principal intuito, adicionar a flexibilidade necessária e eliminar a complexidade dispensável no processo de desenvolvimento de *software*. Isto foi possível graças ao fato de que ambas as metodologias possuem características comuns que podem ser mescladas e que não são mutuamente exclusivas.

Sendo assim, para viabilizar a revisão de processos proposta, a ferramenta de software que registra as solicitações de serviço para a equipe de desenvolvimento, aqui chamadas de tickets, será usada como meio para geração

do *Product Backlog*. Deste modo, cada pacote de requisições incluídas na liberação de cada versão do *software* formará o S*print Backlog* da equipe de desenvolvimento.

A prioridade de cada solicitação deverá ser definida pelo *Scrum* Master, papel a ser representado pelo gerente de desenvolvimento e pelo *Product Owner*, representado pelo proprietário da empresa. E em relação às reuniões diárias, deverá ser estabelecido um consenso quanto ao horário, de acordo com a demanda de serviços.

A prática do *Kanban* será adotada pela equipe, com o intuito de que o processo de desenvolvimento se torne mais transparente, estimulando uma cobrança natural em busca da resolução dos problemas e execução das tarefas.

Para realizar a etapa de planejamento da *Exteme Programming*, algumas particularidades serão adaptadas à realidade da empresa. Um dos princípios da XP é o de que a equipe e o cliente trabalhem juntos e em constante comunicação. Neste caso, será incluído mais um membro na equipe: o técnico de suporte, que deverá atuar como porta-voz do cliente.

A metodologia XP prega o princípio da simplicidade, entretanto, com relação a este, foi identificado um contraponto em relação ao que é praticado atualmente na empresa. Ocorre que a equipe de desenvolvimento é motivada a implementar as funcionalidades de modo que problemas futuros sejam previstos e eliminados de forma preventiva, indo na contramão do referido princípio, visto que este valoriza o projeto simples ao invés de representações complexas.

Entretanto, ainda na etapa de projeto emerge uma característica apontada como benéfica entre as práticas da XP e que é almejado dentro do processo atual de desenvolvimento: a refatoração.

Assim, segundo Fowler (FOWLER, 2000), refatoração é o processo de modificação do *software* de modo que o comportamento externo do código não seja alterado, porém a estrutura interna seja aperfeiçoada.

Quanto aos processos de testes de *software*, talvez seja necessário rever está prática na empresa futuramente. Entretanto, neste primeiro momento os testes deverão continuar a ser realizados pelo analista na medida em que as tarefas do *backlog* são concluídas e integradas. Sendo assim, no final de cada *Sprint*, deverá ser efetuado uma bateria completa de testes para assegurar que

todas as funcionalidades correspondam aos requisitos previamente estabelecidos.

AVALIAÇÃO DOS RESULTADOS

Com relação à mudança de paradigma de trabalho e considerando a própria adoção de uma metodologia de desenvolvimento, houve uma questão relacionada à mudança cultural na equipe que precisou ser considerada. A resistência à mudança. Sabe-se que este é um dos fatores que muitas vezes dificulta a implantação das metodologias ágeis, principalmente em equipes maiores. Entretanto, como a equipe de desenvolvimento em questão já possuía saberes básicos em relação à Engenharia de Software e às Metodologias Ágeis, pode-se dizer que a adoção de algumas das práticas foi relativamente descomplicada.

Assim, ao final do período analisado por esta pesquisa, pôde-se observar uma considerável melhora nos processos de desenvolvimento de *software* da empresa. Também ficou evidente, que a equipe se tornou mais colaborativa e disposta a aderir a outras práticas a fim de acelerar as implementações e agregar mais qualidade aos produtos de *software*.

Deste modo, após a implantação das práticas ágeis citadas no capítulo anterior, foi realizada uma nova análise dos dados usando as mesmas fontes que a análise efetuada anteriormente. Nesse novo cenário, constatou-se que apesar de a quantidade de tickets abertos ter sido superior às do período analisado antes da implementação dos novos processos, o número de tickets resolvidos foi muito maior também. Outro ponto positivo, é que o número de solicitações reincidentes também foi reduzido. Os resultados podem ser vistos na Tabela 2.

Tabela 2: Tickets do período de Outubro/2015 a Janeiro/2016

Mês	Março 2016	Abril 2016	Maio 2016	Junho 2016
Tickets abertos	31	23	19	36
Tickets resolvidos	27	20	15	36

Fonte: (o autor, 2016).

Ao analisar os dados apresentados na Tabela 2, evidencia-se agora que 88,3% do *tickets* passaram a ser resolvidos dentro do mesmo mês, ainda que a quantidade de incidentes tenha aumentado significativamente com relação à

primeira análise. Também é possível observar, que no mês com maior número de *tickets* abertos a equipe teve seu melhor desempenho, ao contrário do primeiro cenário. Outro dado importante que foi possível observar, é que o tempo médio para resolução das solicitações que era de aproximadamente 9 dias, passou a ser de 5 dias, e que dentro do intervalo analisado, a reincidência de *tickets* reduziu de 4 para apenas 2.

Isto evidencia um aumento significativo na eficiência da equipe, além de demonstrar o desenvolvimento do seu potencial de escalabilidade.

Outro dado importante é com relação às percepções subjetivas do time de desenvolvimento. Nesse sentido, através de um questionário que foi aplicado à equipe, notou-se que o time cresceu profissionalmente, com melhoras na comunicação interna e simplificação do código fonte. O impacto destas mudanças pôde ser percebido com o aumento da produtividade e da qualidade do produto final. Além disso, os membros da equipe tornaram-se mais cooperativos entre si e demonstraram-se amplamente interessados em dar continuidade à implantação de outras práticas e metodologias ágeis que possam contribuir para a melhoria dos processos internos.

CONCLUSÃO

Este trabalho desempenhou o importante papel de realizar um estudo sobre metodologias de desenvolvimento de software, tanto tradicionais quanto as ágeis, com o propósito de combinar suas melhores práticas e aplicar em uma empresa real. O objetivo principal foi implementar melhorias no processo de desenvolvimento de software, preocupando-se com a qualidade do produto entregue ao cliente final, sem perder de vista requisitos importantes com relação à projetos, como prazo, custo e escopo.

Assim, a partir dos saberes adquiridos por meio da elaboração desta pesquisa, bem como dos resultados apresentados, entende-se que a implantação combinada de práticas ágeis pode contribuir para a melhoria dos processos de desenvolvimento de software.

REFERÊNCIAS

ALVES, Renan Batista; JUNIOR, Sidenei Mendes Pontes. **Sistema de Gerenciamento de Estoque Loja de Artesanato**. 2011. Disponível em: <http://docplayer.com.br/449004-Instituto-federal-do-parana-renan-batista-alves-sidenei-mendes-pontes-junior-sistema-de-gerenciamento-de-estoque-loja-de-artesanato.html>. Acesso em: 21 de Março de 2016.

BRQ. **Metodologias Ágeis de Desenvolvimento de Software**. Disponível em: <http://www.brq.com/metodologias-ageis/>. Acesso em: 18 de Abril de 2016.

FOWLER. Martin. **The New Methodology**. 2005. Disponível: <http://www.martinfowler.com/articles/newMethodology.html#N8B>. Acesso em: 14 de Julho de 2016.

GOMES, André Faria. **Agile: Desenvolvimento de Software com Entregas Frequentes e Foco no Valor de Negócio.** São Paulo: Casa do Código, 2013.

HIRAMA, Kechi. **Engenharia de Software: Qualidade e Produtividade com Tecnologia**. Rio de Janeiro: Elsevier, 2012.

INTERNATIONAL SCRUM INSTITUTE. **Sprint Burndown Report/Charts.** Disponível em <http://www.scrum-institute.org/Sprint_Burndown_Reports.php>. Acesso em: 18 de Abril de 2016.

PRESSMAN, Roger S. **Engenharia de Software: Uma abordagem profissional**. 7ª ed. São Paulo. AMGH, 2011.

_____. **Engenharia de Software**. 6ª ed. São Paulo. Mc. Graw-Hill, 2006.

REZENDE, Denis Alcides. **Engenharia de Software e Sistemas de Informação.** 3ª ed. Rio de Janeiro: Brasport, 2005.

ROSA, Valcir Júnio Dalla; Sperotto, Fabio Aiub. **Engenharia de Software e o Software Livre**. 2009. Disponível em: <http://pt.slideshare.net/fabioaiub/engenharia-de-software-e-o-software-livre>. Acesso em: 21 de Março de 2016.

SOARES. Michel dos Santos. **Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software**. 2004. Disponível em: <http://jeltex.googlecode.com/svn/trunk/Jeltex/Material%20de%20Leitura/UML/Compara%C3%A7%C3%A3o%20entre%20metodologias.PDF>. Acesso em 03 de Agosto de 2015.

SCHWABER, Ken; SUTHERLAND, Jeff. **Guia do Scrum**. 2013. Disponível em: <http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf>. Acesso em: 28 de Março de 2016.

WEELS, Don. **Extreme Programming: A gentle introduction**. Disponível em <http://www.extremeprogramming.org/>. Acesso em: 30 de Junho de 2016.

WAZLAWICK, Raul Sidnei. **Engenharia de Software: Conceitos e Práticas.** Rio de Janeiro: Elsevier, 2013.

IEEE - Institute of Electrical and Electronics Engineers. **ISO/IEEE Std. 12207**: Systems and software engineering - Software life cycle processes. Montréal, 2008.