A IMPLANTAÇÃO DAS METODOLOGIAS ÁGEIS DE DESENVOLVIMENTO DE SOFTWARE *SCRUM* E *EXTREME PROGRAMMING* (XP): UMA ALTERNATIVA PARA PEQUENAS EMPRESAS DO SETOR DE TECNOLOGIA DA INFORMAÇÃO[1]

Rodrigo Dantas Nunes[2]

RESUMO

Este trabalho apresenta um caso de uso de implantação das metodologias ágeis de desenvolvimento, Scrum e Extreme Programming (XP), e ressalta tal abordagem como uma alternativa para pequenas empresas de Tecnologia da Informação (TI). Tal abordagem se faz necessária em função da falta de processos formais de desenvolvimento em empresas deste perfil, o que acarreta certos problemas à construção de softwares. O objetivo é viabilizar a solução para a falta de um modelo de desenvolvimento através da implantação das referidas metodologias. Este propósito será conseguido mediante o levantamento dos conceitos a volta do tema e a apresentação de como implantar estas metodologias. Ao final, este estudo evidencia que *Scrum* e *XP* podem contribuir para a organização e a formalização do processo de desenvolvimento de software utilizado em pequenas empresas de TI, além de favorecê-las com um ganho de produtividade.

Palavras-chave: Metodologias ágeis. Scrum. Extreme Programming. XP. Tecnologia da Informação.

1 INTRODUÇÃO

A cada dia organizações de todas as naturezas dependem mais e mais de Sistemas de Informação Gerencial (SIG), e consequentemente, de profissionais capazes de criar e manter soluções coerentes com seu planejamento estratégico.

Estes SIG's são desenvolvidos por empresas de todos os tamanhos, e com níveis de maturidade no processo de desenvolvimento no mínimo questionáveis, pois a forma como se guia o desenvolvimento de um software, impacta em vários aspectos, sendo prazo e orçamento, os mais críticos para o sucesso deste investimento.

Para grandes empresas do setor de informática, pode ser uma atividade trivial administrar projetos de desenvolvimento, usando sólidos processos de gestão de projeto de software. No entanto, para pequenas empresas do setor, vários fatores contribuem

[1] **Como citar este trabalho:** NUNES, R. D. A implantação das metodologias ágeis de desenvolvimento de software Scrum e Extreme Programming (XP): uma alternativa para pequenas empresas do setor de tecnologia da informação. **ForScience**: revista científica do IFMG, Formiga, v. 4, n. 2, e00117, jul./dez. 2016.

[2] Mestrando em Engenharia de Sistemas e Automação na Universidade Federal de Lavras (UFLA). (http://lattes.cnpq.br/5868835072659556). E-mail: rdantasnunes@gmail.com.

negativamente para a fluência do projeto, principalmente, o pequeno quadro de funcionários, o prazo para lançar os produtos no mercado e obter o retorno sobre o investimento, e também a falta de mão de obra qualificada a baixo custo.

Em função deste cenário, muitas empresas pequenas do setor, necessitam adotar um processo mais ágil, que permita a gerência destes projetos sem tanta burocracia, mas que ao mesmo tempo, possibilite o controle sobre o projeto.

Assim ocorre com várias empresas do setor de desenvolvimento de SIG's, onde há um quadro de funcionários reduzido e que não se utiliza de processos formais de desenvolvimento, visto que este consumiria muito esforço da equipe, impedindo que os projetos fossem entregues no prazo, e dentro do orçamento. Assim, as pequenas empresas de desenvolvimento, e que se encaixam neste perfil, são alvo deste trabalho.

O objetivo é viabilizar a implantação do *Scrum* juntamente com o *Extreme Programming* (XP). Como consequência, isso provocará uma mudança cultural nestas empresas, permitindo maior controle sobre os projetos desenvolvidos, e ajudando os dirigentes a obter melhor aproveitamento da mão de obra, aumentando a produtividade.

Tal abordagem se justifica porque a falta de um processo formal de desenvolvimento acarreta problemas à construção de um software. Sem um processo que lhe permita mensurar o trabalho a ser feito, diversos aspectos podem escapar ao controle da empresa. Embora estas sejam empresas pequenas, com equipes reduzidas e projetos de menor escala, com a falta da adoção de um processo não é possível, por exemplo, estimar de forma mais precisa o tempo de desenvolvimento de um software, ou o número de desenvolvedores necessários para determinado projeto.

A fim de alcançar seu objetivo, este estudo relata o trabalho necessário para a implantação das metodologias ágeis de desenvolvimento *Scrum* e *XP* em empresas do setor de TI, no intuito de formalizar seus processos de desenvolvimento, e permitindo a autogestão dos produtos que são desenvolvidos pela equipe.

Ao final deste trabalho, é esperada a conscientização sobre o potencial destas metodologias, como funcionam na teoria, e como funcionaram na prática de acordo com o que foi relatado em outros trabalhos.

O trabalho apresentado neste artigo consistirá em teoria, com o levantamento dos conceitos a volta do tema, e prática, com a apresentação de um caso de uso com a implantação destas metodologias.

2 PROCESSOS DE DESENVOLVIMENTO DE *SOFTWARE*

2.1 Considerações gerais

Pressman (2006) cita que quando você elabora um produto de passos previsíveis – um roteiro que o ajuda a criar a tempo um resultado de alta qualidade é muito importante, e este roteiro que você segue é chamado de processo de *software*. E assim, muitos processos de desenvolvimento de *software* foram criados e adaptados ao longo do tempo. Não há dúvida quanto ao benefício obtido com tais processos, haja vista que até hoje são usados amplamente por esta indústria. Alguns são mais burocráticos, e ainda podem ser direcionados para determinados tipos de projetos. Mas o que realmente importa, é o benefício alcançado ao final do trabalho de desenvolvimento do projeto.

Pressman (2006) ainda cita a diferença entre métodos e processos, dizendo que métodos são "como fazer", e os processos, são "o que fazer" para construir *software*.

2.2 Processos comuns

Como modelos de processos mais conhecidos, Pressman (2006) cita o *Capability Maturity Model Integration* (CMMI), e o modelo de ciclo de vida clássico (Modelo Cascata), considerando vantagens e desvantagens de utilizá-los, onde afirma que os requisitos detalhados do *CMMI* devem ser seriamente considerados se uma organização construir sistemas grandes e complexos que envolvam dúzias ou centenas de pessoas durante muitos meses ou anos.

Considerando este e outros fatores, a grande maioria das empresas de pequeno porte opta por não usar um modelo formal de desenvolvimento, limitando-se a apenas documentar os códigos-fonte das aplicações, o que inicialmente é suficiente. No entanto, chega o dia que esta não é mais a realidade da empresa, e por isso, a necessidade de um modelo menos formal, que permita o controle das atividades, a geração de um documental de referencia, e que funcione bem com uma equipe reduzida de pessoas.

3 METODOLOGIAS ÁGEIS

3.1 Considerações gerais

Uma alternativa aos métodos formais (tais como o CMMI e o modelo Cascata), seria a utilização de metodologias ágeis, que Sommerville (2007) defende argumentando que os negócios atualmente operam em um ambiente global sujeito a rápidas mudanças. Eles têm de responder a novas oportunidades e mercados, mudanças de condições econômicas e ao surgimento de produtos e serviços concorrentes.

O que nos induz a associar, agilidade com produção rápida de sistemas. E ainda segundo o autor, muitos negócios estão dispostos a equilibrar a qualidade e compromisso, com requisitos do *software* e com a entrega rápida.

Apesar destes fatores, estas metodologias não são uma "licença" para improvisar modelos, ou desenvolver *software* em ritmo acelerado e sem controle de qualidade. Ao contrário disso, segundo Sommerville (2007), processos de desenvolvimento rápido são projetados para criar *software* útil rapidamente de forma iterativa nos quais a especificação, o projeto, o desenvolvimento e o teste são intercalados. Assim sendo, o produto não é desenvolvido e disponibilizado integralmente, mas em uma série de incrementos.

Assim como os modelos convencionais de desenvolvimento, os modelos ágeis possuem uma infinidade de processos dos mais diversos formatos, mas regidos pelos mesmos princípios, que estabelecem o cerne destes processos, a fim de que eles tenham sempre como ideal, menor número de processos burocráticos, e maior fluidez. Os Princípios do Software Ágil são apresentados no Quadro 1 abaixo.

Em síntese a estes princípios, o Beck et al. (2011), traz quatro premissas:

- **Indivíduos e interação entre eles** 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

Onde, ainda segundo o Beck et al. (2011), mesmo havendo valor nos itens à direita e sem destaque, valorizamos mais os itens destacados à esquerda das premissas.

Princípios do Software Ágil
Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor.
Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adéquam a mudanças, para que o cliente possa tirar vantagens competitivas.
Entregar software funcionando com freqüência, na escala de semanas até meses, com preferência aos períodos mais curtos.
Pessoas relacionadas à negócios e desenvolvedores devem trabalhar em conjunto e

diariamente, durante todo o curso do projeto.
Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.
O Método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara.
Software funcional é a medida primária de progresso.
Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes.
Contínua atenção à excelência técnica e bom design, aumenta a agilidade.
Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.
As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis.
Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.

Quadro 1: Os 12 (doze) princípios do software ágil.
Fonte: Beck et al. (2011)

Dentre os vários modelos de processos que seguem a filosofia ágil de desenvolvimento de *software*, foram escolhidos o *Scrum* e o *XP*. Levou-se estes em conta para implantação a forma como se completam, uma vez que as práticas da *XP* são aderentes às cerimônias do *Scrum*, como poderá ser observado a seguir nas próximas seções.

3.2 *Scrum*

Segundo a Scrum alliance (2011), o *Scrum* é um framework ágil destinado a projetos complexos e foi originalmente formalizado para projetos de desenvolvimento de *software*, mas funciona bem para qualquer escopo de trabalho inovador.

Pressman (2006), explica que o *Scrum* é um modelo ágil de processo, desenvolvido por Jeff Sutherland e por sua equipe no início da década de 1990, onde há um padrão de processo em que ocorrem as tarefas de trabalho, chamado *Sprint*. O trabalho conduzido dentro de um *Sprint* é adaptado ao problema em mãos, e freqüentemente modificado pela equipe *Scrum*.

De acordo com Pressman (2006), no *Scrum* existem três papéis conhecidos como *Scrum Master*, *Product Owner* (*PO*) e o *Scrum Team*, onde o *Scrum Master* tem a responsabilidade de manter a equipe (*Scrum Team*), focada no objetivo, evitando que problemas de natureza não técnica interrompam suas atividades dentro do *Sprint*. O *PO*, como sugerido pelo nome, é o dono do produto a ser desenvolvido, um usuário representante do cliente. Na falta deste representante, o papel do *PO* deve ser desempenhado por alguém da própria equipe de desenvolvimento com experiência no negócio. Assim, sua função será

auxiliar a equipe em dúvidas relativas ao produto em construção, e principalmente, com o *Product Backlog*, que é uma lista de desejos com as funcionalidades a serem implementadas.

Ainda na definição de Pressman (2006), o *Scrum Team* realiza todo trabalho operacional, e é composta por geralmente cinco a nove pessoas que decidem como o trabalho é organizado e como as atribuições serão divididas. No *Scrum Team*, não há papéis definidos, todos devem ser capazes de trocar as tarefas com outro membro, pois a idéia é que a equipe seja polivalente e uniforme tecnicamente.

O *Scrum* possui também quatro reuniões, ou cerimônias, como são chamadas na literatura. A primeira delas é a *Sprint Planning*, ou planejamento do *Sprint*, que é uma sessão de trabalho facilitada e liderada pelo *Scrum Master*, sendo dividida em duas partes. Na primeira parte o *Product Owner* apresenta e explica os itens de maior valor de negócio para o time. E na segunda parte, o time irá estimar o tamanho e/ou a complexidade das tarefas.

A segunda reunião, chamada de *Sprint Review*, é realizada depois de concluído o *Sprint*, onde o time apresentará para o *PO* o incremento de produto gerado nesta iteração. Apenas tarefas 100% concluídas devem ser apresentadas.

A terceira, conhecida como retrospectiva do *Sprint*, definida por Kniberg (2007) como sendo um dos momentos mais importantes, pois nessa reunião todo o time realiza uma auto-reflexão e procura identificar no *Sprint* que acabou o que funcionou bem, o que não funcionou bem, e o que pode ser melhorado.

A última reunião, que também é uma prática do *XP*, é a *Stand Up Meeting*, e deve ser realizada todos os dias, preferencialmente no mesmo lugar, e no mesmo horário, como recomenda Kniberg (2007).

Fechando a estrutura do *Scrum*, tem-se ainda os artefatos utilizados, que são o *Product Backlog*, uma lista com as funcionalidades que o sistema terá, o *Sprint Backlog*, uma lista apenas com as funcionalidades do *Product Backlog* que serão implementadas no *Sprint* atual e o *Burndown Chart*, um gráfico utilizado para acompanhar o progresso do desenvolvimento.

Com esta composição, segundo a Scrum alliance (2011), o *Scrum* oferece benefícios para qualquer tipo de equipe, incluindo a melhoria no trabalho em conjunto, melhor comunicação e resultados mais rápidos.

3.3 *Extreme Programming - XP*

Sommerville (2007) afirma que o *XP*, é talvez o mais conhecido e mais amplamente usado dos métodos ágeis. Segundo ele, o nome foi cunhado por Kent Beck, seu criador, porque a abordagem foi desenvolvida pelo avanço de reconhecida boa prática, tal como o desenvolvimento iterativo e o envolvimento do cliente em níveis "extremos".

Segundo Teles (2006), o *XP*, é um processo de desenvolvimento de software voltado para projetos cujos requisitos são vagos e mudam com freqüência, desenvolvimento de sistemas orientados a objeto, equipes pequenas, preferencialmente até 12 desenvolvedores, e desenvolvimento incremental (ou iterativo), onde o sistema começa a ser implementado logo no início do projeto e vai ganhando novas funcionalidades ao longo do tempo.

Teles (2006), ainda destaca que o *XP* assim como os processos ágeis na sua totalidade, compartilham a premissa de que o cliente aprende sobre suas necessidades, na medida em que é capaz de manipular o sistema que está sendo produzido.

Para isso, o *XP* conta com quatro valores fundamentais sendo eles o *Feedback* (Quando o cliente aprende com o sistema que utiliza e re-avalia as suas necessidades, ele gera feedback para a equipe de desenvolvimento), a Comunicação (ocorrida entre o cliente e a equipe permitindo que todos os detalhes do projeto sejam tratados com a atenção e a agilidade que merecem), a Simplicidade (Se aplica ao aprender a implementar apenas aquilo que é suficiente para atender a cada necessidade do cliente), e a Coragem (A equipe precisa ser corajosa e acreditar que, utilizando as práticas e valores do *XP*, será capaz de fazer o software evoluir com segurança e agilidade) como cita Teles (2006).

Estes valores são importantes para manter o foco das equipes que usam o *XP* em um processo diferente do convencional, com maior dinamismo e capacidade de resposta.

Haja vista a viabilidade da adoção do *XP*, a Ágil Coop (2012) traz uma lista com diversos casos de sucesso, onde destaca-se a Assembléia Legislativa de São Paulo, a Paggo, e a Locaweb pelo porte e pela importância que a Tecnologia da Informação tem para suas atividades finais.

Teles (2006) traz em sua obra 13 (treze) importantes práticas que compõem o *XP*. Estas são apresentadas no Quadro 2 a seguir.

Prática do *XP*	**Descrição**
Cliente Presente	O *XP* trabalha com a premissa de que o cliente deve conduzir o desenvolvimento a partir do feedback que recebe do sistema.
Jogo do Planejamento	No início de cada iteração ocorre o jogo do planejamento. Trata-se de uma reunião onde o cliente avalia as funcionalidades que serão implementadas.

Stand Up Meeting	A equipe se reúne a cada manhã para avaliar o trabalho que foi executado no dia anterior e priorizar aquilo que será implementado no dia que se inicia.
Programação em Par	Os desenvolvedores implementam as funcionalidades em pares, ou seja, diante de cada computador, existem sempre dois desenvolvedores que trabalham juntos para produzir o mesmo código.
Desenvolvimento Guiado p/ Testes	Testes são escritos para cada funcionalidade antes de codificá-las. Fazendo isso, eles aprofundam o entendimento das necessidades do cliente.
Refactoring	O *refactoring* é o ato de alterar um código sem afetar a funcionalidade que ele implementa. O objetivo é tornar o *software* mais simples de ser mantido.
Código Coletivo	Os desenvolvedores têm acesso a todas as partes do código e podem alterar aquilo que julgarem importante sem pedir autorização de outra pessoa.
Código Padronizado	Para facilitar a manutenção no código por parte de toda a equipe, padrões de codificação são definidos tornando o sistema mais homogêneo e permitir que qualquer membro da equipe tenha condições de dar manutenção no sistema.
Design Simples	Para que o cliente possa obter *feedback* logo, a equipe precisa ser ágil no desenvolvimento, o que a leva a optar pela simplicidade do *design*.
Metáfora	Para facilitar a criação de um *design* simples, a equipe de desenvolvimento utiliza metáforas, já que elas têm o poder de transmitir ideias complexas de forma simples.
Ritmo Sustentável	Para garantir que a equipe tenha sempre o máximo de rendimento e produza software com melhor qualidade possível, o *XP* recomenda que os desenvolvedores trabalhem apenas oito horas por dia e evitem fazer horas-extras, visto que é essencial estar descansado a cada manhã, de modo a utilizar a mente na sua plenitude.
Integração Contínua	Prática utilizada com o objetivo de checar/testar toda a aplicação, sempre que uma nova funcionalidade é implementada, seja de forma manual, ou automática, forma esta, que se utiliza de ferramentas especializadas para tal.
Releases Curtos	O *XP* tem como objetivo gerar um fluxo contínuo de valor para o cliente. Sendo assim, ele trabalha com *releases* curtos, ou seja, a equipe produz um conjunto reduzido de funcionalidades e coloca em produção rapidamente.

Quadro 2: Práticas do *XP*
Fonte: TELES (2006)

Uma particularidade que pode ser observada entre *XP* e *Scrum* é que ambos contemplam que a cada iteração uma parte do sistema deve ser entregue funcionando. Onde cada iteração deve ser curta e com prazo definido, além disso, com a participação constante do usuário, para que forneça seu *feedback* à equipe de desenvolvimento.

3.4 *Scrum* e *XP* juntas

Segundo Kniberg (2007), o *Scrum* é focado nas práticas de gerenciamento e organização, enquanto o *XP* dá mais atenção às tarefas de programação mesmo. Aí está o porquê de elas trabalharem bem juntas, pois, elas abrangem áreas diferentes e uma complementa a outra.

Embora seja comum o uso conjunto de *Scrum* e *XP*, o próprio autor alerta para o fato de que nem todas as práticas são utilizadas juntas por todas as equipes. Tudo varia de acordo com o perfil da empresa e da equipe que está adotando-as.

Para isto, devem ser também ponderados fatores que implicam no dia-a-dia da equipe, na limitação do numero de funcionários, e inclusive, a realidade dos projetos ora desenvolvidos.

4 PROCESSO DE IMPLANTAÇÃO

4.1 Considerações gerais

É comum que processos envolvendo mudanças gerem desconfiança, e principalmente resistência. Assim, é muito importante que qualquer que seja o profissional a guiar esta mudança, esteja muito bem embasado teoricamente e que busque sempre se apoiar em casos de sucesso a fim de vencer os mais pessimistas.

O modelo de processo sugerido neste artigo para pequenas empresas de TI foi baseado no caso de uso de Kniberg (2007) considerando o sucesso das equipes de trabalho citadas pelo autor em sua obra, e pela similaridade do perfil destas com o perfil das empresas alvo deste trabalho.

4.2 Equipes de trabalho

Kniberg (2007) considera que muitas coisas se complicam quando se tem várias equipes trabalhando no mesmo produto. Esse problema é universal e não tem nada a ver com o *Scrum*. Assim sendo, o recomendável é que se trabalhe com equipes de até onze pessoas, pois um número maior poderia incidir na perda do controle e organização.

Considerando o ponto de vista do autor, uma solução para pequenas empresas com um número de desenvolvedores maior que isso, seria a divisão da equipe, assim como seu treinamento, no intuito de instruir, e aumentar o foco naquilo que realmente interessa.

4.2.1 Treinamento da equipe

Não basta que apenas os gestores das empresas de TI tenham conhecimento e interesse em implantar as metodologias em questão, é necessário que toda a equipe esteja motivada e bem informada a respeito do que será feito e de como cumprir seu papel neste processo, assim como cita Silva (2010, p. 66): "Saber fazer é uma questão de conhecimentos, habilidades ou atitudes. Logo, uma questão de treinamento".

Para cumprir esta etapa, vários meios podem ser utilizados, desde o investimento em um curso com uma empresa especializada, ou até mesmo, a formação de grupos de estudos dentro da empresa. O que realmente fará a diferença é o entendimento do processo que será adotado por parte de todos os envolvidos.

4.3 Modelo utilizado para implantação

Kniberg (2007) sugere a adoção de seis práticas *XP* (Integração contínua, Programação em par, Desenvolvimento guiado pelos testes (TDD), Código coletivo, Código padronizado, Ritmo sustentável), e outras quatro práticas também do *XP* que serão sobrepostas pelo *Scrum* (Cliente presente, *Releases* curtos, Jogo do planejamento, *Stand Up Meeting*).

Em se tratando de equipes com pequeno número de integrantes não serão utilizadas as práticas programação em par e TDD haja vista que a primeira dividiria a equipe, e a segunda exigiria um treinamento adicional, e em um momento de adaptação, mais novidades só aumentariam a curva de aprendizado e a resistência dos pessimistas.

Assim, abordamos em seguida, como alguns autores recomendam, ou exercem estas práticas no seu dia-a-dia.

4.3.1 Cliente presente

Teles (2006) aborda a questão da presença do cliente como uma contribuição para o sucesso do projeto, e também como sendo responsável por viabilizar a simplicidade do processo. Pressman (2006) diz que é ele, o cliente, quem definirá os detalhes operacionais do software de modo que o objetivo do negócio possa ser alcançado, reafirmando assim, a importância de sua presença no projeto.

No entanto, pequenas empresas de TI, podem não ter à sua disponibilidade essa participação, contando apenas com a experiência de membros da equipe com maiores informações sobre as regras de negócio que envolvam o produto que está sendo desenvolvido.

4.3.2 *Releases* curtos

Kniberg (2007) conta que é importante que as equipes experimentem tamanhos variados de *Sprint* para que descubram qual a medida certa de modo a permitir certa agilidade corporativa, mas que também possibilite aos desenvolvedores adquirirem fluidez, e se necessário, se recuperarem de possíveis problemas ao decorrer do *Sprint*.

4.3.3 Jogo do planejamento

Também conhecido como *Planning Poker*, é realizado por toda a equipe, exceto o *PO*, cujo objetivo é estimar cada atividade do *Sprint*.

Kniberg (2007) enfatiza a importância desta cerimônia no *Scrum* por viabilizar a explanação do que se refere à atividade, e fazer com que questões sobre a mesma surjam cedo, evitando assim, retrabalho em função de erros de interpretação. E para que os envolvidos possam expressar sua estimativa, é utilizado um baralho, contendo cartas com os números aceitos de estimativa.

4.3.4 *Stand Up Meeting*

A *Stand Up Meeting*, ou reunião de pé, é uma prática tanto do *Scrum*, como do *XP*. Kniberg (2007) baseado em sua experiência profissional recomenda que esta seja realizada todos os dias no mesmo horário, no mesmo lugar, e que assim como Teles (2006), concorda que deve ser breve, onde cada elemento da equipe limita-se a responder o que fez no dia anterior, o que irá fazer na presente data, e se está com algum impedimento. Após isso, a equipe contabiliza as atividades prontas, e marca o *Burndown Chart* de acordo com este dado.

4.3.5 Código padronizado

A padronização de código é algo extremamente importante para toda empresa de desenvolvimento, independente da metodologia de desenvolvimento de software que utilize. Kniberg (2007, p. 83) sugere: “Comece com um padrão simples e vá crescendo conforme a necessidade. Escreva apenas regras que não sejam óbvias para todo mundo e correlacione com algum material já existente, sempre que possível”.

4.3.6 Código coletivo

Uma vantagem da utilização desta prática, segundo Kniberg (2007), é a robustez das equipes que a utilizam, fazendo assim, com que esta tenha uma dependência menor a determinados membros da equipe, diminuindo as chances do *Sprint* morrer por falta de alguém capaz de auxiliar os demais.

4.3.7 Ritmo sustentável

Teles (2006), trata desta prática do *XP* enfatizando a importância sobre o descanso da equipe, recomendando que esta trabalhe apenas 8 horas por dia, para que a mesma mantenha um ritmo de produtividade. Assim, ele não aconselha que sejam feitas horas-extras em excesso.

Kniberg (2007) confirma de forma empírica esta premissa, citando uma experiência que teve, onde pôde comprovar o quanto o excesso de horas de trabalho diário compromete o desempenho da equipe.

4.3.8 Integração Contínua

Kniberg (2007) comenta sobre a importância de automatizar este processo através de ferramentas que pouparão tempo, e consequentemente, dinheiro. Segundo o autor, estas ferramentas serão acionadas toda vez que alguém atualizar alguma funcionalidade. Neste momento, o servidor de construção contínua, irá “acordar”, construir tudo desde o início em um servidor compartilhado e realizar todos os testes, em caso de erro, todos são avisados.

5 CONCLUSÃO

Voltando à proposta inicial deste artigo, sobre como pequenas empresas de software podem se beneficiar das metodologias ágeis, pode-se observar como é fácil a adoção das mesmas com um ganho de produtividade satisfatório, e uma curva de aprendizado relativamente pequena.

Conclui-se que estas metodologias têm muito a agregar para aquelas empresas que decidirem adotá-las, e a um baixo custo. Além de simplistas, estas metodologias são flexíveis, permitindo às empresas adaptá-las à sua realidade.

Este artigo demanda um maior aprofundamento principalmente no que diz respeito à prática da implantação. Assim sendo, na sequência deste trabalho, poderia ser realizado um trabalho prático fazendo experimentações, ou mesmo pesquisa de campo com casos de uso da implantação das metodologias ágeis para um levantamento dos dados apurados, e comprovação da sua eficácia.

THE IMPLANTATION OF AGILE METHODOLOGIES OF SCRUM SOFTWARE DEVELOPMENT AND EXTREME PROGRAMMING (XP): An alternative for small companies in the Information Technology sector

ABSTRACT

This paper presents a case for using the implantation of the development agile methodologies, Scrum and Extreme Programming (XP), and highlights this approach as an alternative for small Information Technology (IT) companies. This approach is needed because of the lack of formal development in companies with this profile, which causes some problems to the construction of software. The objective is to enable the solution to the lack of a development model through the implementation of these methodologies. This purpose will be achieved by raising the concepts around the theme, and presenting how to implement these methodologies. The study showed that Scrum and XP can contribute to the organization and formalization of the process of software development used in small IT companies and favor them with a productivity gain.

Keywords: Agile methodologies. Scrum. Extreme Programming. XP. Information Technology.

REFERÊNCIA

ÁGIL COOP. **Casos de sucesso**. [S.l:s.n], 2012. Disponível em: <http://ccsl.ime.usp.br/agilcoop/casos_de_sucesso>. Acesso em: 13 jan. 2017.

BECK, K. et al. **Os doze princípios do software ágil**. [S.l:s.n], 2011. Disponível em: <http://www.manifestoagil.com.br/principios.html>. Acesso em: 13 jan. 2017.

KNIBERG, H. **Scrum e XP direto das trincheiras**: como nós fazemos Scrum. São Paulo: InfoQ, 2007.

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

SCRUM ALLIANCE. **Learn about Scrum**. [S.l:s.n], 2011. Disponível em: <https://www.scrumalliance.org/why-scrum>. Acesso em: 13 jan. 2017.

SILVA, G. **Gestão de pessoas & clima organizacional**: Guia de Estudo. GEPOS - UNIS/MG, 2010. 85p.

SOMMERVILLE, I. **Engenharia de software**. 8. ed. São Paulo: Pearson Addison Wesley, 2007.

TELES, V. M. **Extreme programming**. São Paulo: Novatec Editora, 2006.

Recebido em: 14/12/2016
Aprovado em: 29/12/2016
Publicado em: 26/02/2017