

MODELO DE PROCESSO PARA MICRO E PEQUENAS EMPRESAS DE SOFTWARE COM BASE EM METODOLOGIAS ÁGEIS

MIRILIAN CARLA ARAUJO CORILLO¹, ANDREA PADOVAN JUBILEU².

¹ Tecnóloga em Análise e Desenvolvimento de Sistemas pela FATEC – Presidente Prudente-SP. mirilianc@hotmail.com.

² Profa. Dra. do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas. FATEC – Presidente Prudente-SP. andrea.jubileu@fatec.sp.gov.br.

RESUMO

A maioria das micro e pequenas empresas (MPEs) não utilizam processos formalizados para o desenvolvimento de software, pois estas iniciativas demandam custos e, muitas vezes, burocratizam essas empresas. Todavia, muitas delas reconhecem a importância de se utilizar processos. Este trabalho tem por objetivo analisar as metodologias ágeis Extreme Programming (XP) e SCRUM, a fim de averiguar a possibilidade de utilização de ambas em um mesmo modelo de processo de desenvolvimento de software. Em linhas gerais, essas metodologias ágeis, minimizam a documentação a ser criada durante o desenvolvimento do software por meio da ênfase na comunicação entre os envolvidos no processo, inclusive o cliente. Entretanto, o XP tem maior ênfase na implementação do software enquanto o SCRUM enfatiza a gestão do projeto. Como resultado dessa pesquisa foi proposto o MPA-DS (Modelo de Processo Ágil para o Desenvolvimento de Software).

PALAVRAS-CHAVE: SCRUM. Extreme programming. Modelo de processo ágil de desenvolvimento de software.

1. INTRODUÇÃO

Existem vários modelos de processo de software que propõem sistemáticas para o desenvolvimento de software, tais como cascata, incremental, espiral, processo unificado, entre outros. Esses têm um grande enfoque na documentação, o que é considerado um fator crítico para muitos analistas/desenvolvedores. Com isso, surgiram metodologias ágeis que têm o enfoque na comunicação entre as pessoas e, por consequência, a minimização de documentação (SOMMERVILLE, 2003).

Segundo Sommerville (2003), geralmente, as metodologias ágeis contam com uma abordagem iterativa para especificação, desenvolvimento e entrega de software, e foram criadas principalmente para apoiar o desenvolvimento de aplicações de negócios nas quais os requisitos de sistema mudam rapidamente durante o processo de desenvolvimento. Alguns exemplos de metodologias ágeis são Extreme Programming (XP), SCRUM, FDD (*Feature-Driven Development*), DSDM (*Dynamic Systems Development Method*), ASD (*Adaptative Software Development*). Atualmente, as metodologias XP e SCRUM são bastante comentadas.

Conforme Habra et al.(2008 apud BERNI, 2010), faltam recursos nas micro e pequenas empresas³ (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. Pequenas empresas de software têm

³ Segundo o Estatuto da Micro e Pequena Empresa de 1999: É considerada *microempresa* aquela cuja receita bruta anual é igual ou inferior a R\$ 433.755,14 e *empresa de pequeno porte* aquela cuja receita bruta anual é superior a R\$ 433.755,14 e igual ou inferior a R\$ 2.133.222,00. O SEBRAE utiliza-se do número de funcionários nas empresas como referência para o enquadramento das empresas: *microempresa* emprega até 09 funcionários e *pequena empresa* emprega de 10 a 49 funcionários (SEBRAE, 2010).

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 onde os requisitos sofrem constante modificação, onde 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. Nesse contexto, as metodologias ágeis são indicadas. A organização define seu próprio modelo de gestão, fazendo com que sejam utilizadas somente atividades ou tarefas que permitam a fluência do processo produtivo (BERNI, 2010).

Sendo assim, considerando a sugestão de trabalho futuro do doutorado defendido por Jubileu (2008), o objetivo deste trabalho é fazer uma análise das possibilidades de uso em conjunto das metodologias ágeis Extreme Programming (XP) e SCRUM em um mesmo modelo de processo de desenvolvimento de software. Para isso, primeiramente foram identificadas quais as práticas dessas metodologias ágeis e proposto um modelo de processo que abrange atividades correspondentes às práticas do XP e SCRUM.

Este artigo está organizado em seções. Na seção 2 é apresentado o referencial teórico relacionado aos temas abordados: metodologias ágeis SCRUM e Extreme Programming. Na seção 3 é descrita a metodologia utilizada na realização deste trabalho de pesquisa. As seções 4 e 5 apresentam, respectivamente, os resultados e a conclusão do trabalho, e por fim são apresentadas as referências utilizadas.

2. REFERENCIAL TEÓRICO

2.1 SCRUM

Em 1986, os pesquisadores Takeuchi e Nonaka, publicaram um artigo no qual eles citavam que o cenário do desenvolvimento de software estava mudando, e que além de vários itens que influenciavam a qualidade do produto, era necessária a ênfase em flexibilidade e velocidade.

Em 1993, Jeff Sutherland baseado no artigo de Takeuchi e Nonaka implantou o SCRUM na Easel Corporation. Jeff trabalhou com Ken Schwaber e, juntos, formalizaram o SCRUM na Conferência OOPSLA⁴ em 1995. Eles melhoraram e implantaram o SCRUM em muitas empresas de software e participaram da escrita do Manifesto Ágil.

No SCRUM, os projetos são divididos em ciclos (tipicamente de 30 dias) chamados de *sprints*. Em cada *sprint* deve ser executado um conjunto de atividades. As funcionalidades a serem implementadas em um projeto são mantidas em uma lista que é conhecida como *product backlog*. No início de cada *sprint*, faz-se uma reunião de planejamento na qual o *product owner* prioriza os itens do *product backlog* e a equipe seleciona as atividades que será capaz de implementar durante o *sprint* que se inicia. As tarefas alocadas em um *sprint* são transferidas do *product backlog* para o *sprint backlog*.

A cada dia de um *sprint*, a equipe faz uma breve reunião, normalmente de manhã, com o objetivo de disseminar o conhecimento sobre o que foi feito no dia anterior, identificar impedimentos e priorizar o trabalho do dia que se inicia. Ao final de um *sprint*, a equipe apresenta as funcionalidades implementadas em uma reunião de revisão da *sprint*. Finalmente, faz-se uma retrospectiva da *sprint* e a equipe parte para o planejamento do próximo *sprint*, reiniciando o ciclo (IMPROVEIT, 2009).

2.2 EXTREME PROGRAMMING

O Extreme Programming (XP) é uma metodologia ágil para equipes pequenas e médias que

⁴ Object Oriented Programming Systems, Languages, and Applications

desenvolvem software com base em requisitos vagos e que são modificados rapidamente (KOSCIANSKI; SOARES, 2007).

O XP enfatiza o desenvolvimento rápido do projeto e visa garantir a satisfação do cliente, além de favorecer o cumprimento das estimativas. As práticas e valores do XP proporcionam um agradável ambiente de desenvolvimento de software para seus seguidores, que são conduzidos por quatro valores: comunicação, simplicidade, *feedback*, respeito e coragem (KOSCIANSKI; SOARES, 2007).

Para Wells (*online*, 1999), o aspecto mais surpreendente do XP refere-se à simplicidade das regras que tem por base os valores anteriormente citados. Extreme Programming é muito parecido com um quebra cabeça. Há muitos pedaços pequenos, individualmente, as peças não fazem sentido, mas quando combinados formam uma imagem completa que pode ser vista. As regras são baseadas em valores.

O XP trabalha com iterações curtas que podem variar de uma a três semanas. Dentro de uma iteração, a equipe implementa um conjunto de funcionalidades que serão validadas pelo cliente ao final da iteração. Isso resulta em um desafio para a equipe, pois é preciso implementar as funcionalidades com agilidade e qualidade em um curto prazo. Para que o XP funcione bem, é necessário, entre outros fatores, que a equipe utilize um conjunto de ferramentas que irá ajudá-la a desenvolver com segurança e qualidade. Teles (2006) sugere algumas ferramentas que comporiam o mínimo necessário para auxiliar o desenvolvimento com XP: IntelliJ e Eclipse, como ambiente de desenvolvimento; Ant, ferramenta para build; CVS, para repositório de código; JUnit, para testes de unidade.

3. METODOLOGIA DE PESQUISA

A pesquisa é uma atividade voltada para a solução de problemas, através do emprego de processos científicos. Os tipos de pesquisas podem ser classificados de várias formas, por critérios que variam segundo diferentes enfoques (ANDRADE, 2007; SEVERINO, 2007). Para a elaboração deste trabalho foi utilizado o método de pesquisa dedutivo, pois este se baseou em fontes de revisão bibliográfica de trabalhos já publicados.

Quanto à natureza, a pesquisa é classificada como pura ou básica, pois buscou gerar conhecimentos úteis para o avanço da ciência, sem aplicação prática imediata.

Do ponto de vista da forma da abordagem do problema, a pesquisa é qualitativa, pois a apresentação de seus resultados não se deu de forma estatística.

Quanto aos objetivos da pesquisa, é uma pesquisa exploratória, pois investiga e analisa as práticas das metodologias ágeis Extreme Programming e SCRUM. Para tanto, o procedimento técnico utilizado foi a pesquisa bibliográfica.

Para a elaboração deste trabalho foram realizadas as seguintes etapas: Etapa 1. Revisão bibliográfica; Etapa 2. Análise do SCRUM e Extreme Programming; Etapa 3. Proposição de um modelo de processo que utilize práticas do XP e SCRUM.

4. RESULTADOS

4.1 Práticas do XP e SCRUM

A partir de pesquisas bibliográficas fez-se a análise das metodologias ágeis XP e SCRUM, listando as práticas de ambas, descritas a seguir.

Práticas do XP:

- O cliente deve sempre estar presente nas reuniões e no dia a dia do projeto, porém o cliente cuida da parte de negócios e os desenvolvedores do sistema em si.
- Escrever *estórias de usuários* que devem ser registradas em cartões com texto curtos e são escritas pelo próprio cliente.

- Planejar as *releases*, definindo liberações em pouco tempo. Cada *release* será dividido em iterações.
- Cada iteração deve conter *estórias do usuário*, sempre lembrando que o cliente quem definirá a prioridade de cada estória, porém essas estórias podem ser afetadas por dependências técnicas.
- Para cada nova iteração é necessária uma reunião de planejamento da iteração.
- Durante o projeto são realizadas reuniões diárias, onde todos os participantes permanecem em pé e esta reunião deve ter um curto tempo de duração.
- Programação em par: dois programadores desenvolvem suas tarefas em um mesmo computador, enquanto um digita o outro analisa e identifica possíveis erros e mudanças, porém é importante que esses programadores também invertam os papéis.
- O código é propriedade de todo o grupo, assim todos os membros da equipe podem se sentir à vontade para fazer alterações no código.
- Refatorar o código sempre que necessário, ou seja, limpar o código, mantendo as funcionalidades e tornando-o mais enxuto e mais fácil de ser entendido. Todos os membros da equipe têm a liberdade de refatorar o código.
- É importante definir a velocidade das iterações, mantendo-as com uma mesma duração.
- Utilizar metáforas para o sistema, ou seja, formas simples de descrever o projeto e suas funções.
- Organizar a sala dos desenvolvedores, de modo que eles possam se comunicar e, ainda, colocar em prática a programação em par.
- Não adicionar funcionalidades além do necessário no seu código.
- Realizar testes de unidade considerando cada unidade do sistema (por exemplo: classe).
- Realizar testes de aceitação considerando cada *estória de usuário*.
- Criar e manter padrões de codificação para que a equipe de desenvolvimento consiga compreender todo o código do sistema.
- Integrar o código no repositório para que a equipe de desenvolvimento trabalhe sempre com a versão mais recente do sistema.
- Manter um ritmo sustentável para a equipe evitando excesso de trabalho e, assim, mantendo os desenvolvedores sempre dispostos e ativos.

Práticas do SCRUM:

- Os projetos são divididos em iterações (*sprint*).
- Fazer reuniões para o planejamento de *releases*, entrando em comum acordo com o *product owner* e verificando qual a verdadeira importância de cada item, e quais funcionalidades devem estar incluídas em cada *release*.
- Criar *product backlog* (lista de requisitos), de acordo com as necessidades do cliente. Cada item do *backlog* é chamado de estória. O *product backlog* deve ser priorizado de acordo com o *product owner*, que é o responsável pelo *product backlog*.
- Fazer reunião de planejamento para se iniciar cada *sprint* (fixada em 8 horas de planejamento), na qual se define o objetivo, os membros da equipe, o *sprint backlog*, e uma data para a apresentação da *sprint*.
- As estórias ou itens do *product backlog* devem ser quebradas em tarefas menores para que formem o *sprint backlog*.
- Durante o *sprint* realizar reuniões diárias: reuniões em pé que não ultrapassem 15 minutos. Nesta reunião deve-se atualizar o *sprint backlog*, de acordo com o que cada membro da equipe fez no dia anterior e o que fará no dia corrente.
- Medir o quanto falta para o término da *sprint* através de um gráfico de *burndown* da *sprint*.
- Medir o quanto falta para o término da *release* através de um gráfico de *burndown* da *release*.
- Realizar uma reunião de revisão da *sprint*: o time apresenta a funcionalidade que foi terminada para os *stakeholders*, para que estes tenham *feedback* do projeto.
- Realizar uma reunião de retrospectiva da *sprint* para mostrar os pontos fortes e fracos em se tratando de relações entre pessoas, regras SCRUM, ferramentas e processos de cada iteração.

- Utilizar um quadro de tarefas que deve ser atualizado nas reuniões diárias. Esse quadro deve apontar as atividades que já foram feitas, as que estão parcialmente terminadas e as atividades a serem realizadas.
- Definir equipes pequenas, de no máximo 10 pessoas, contendo o *scrum master* e o *product owner*.

Ao examinar as práticas do XP e do SCRUM, fica visível que por se tratarem de metodologias ágeis ambas tem práticas comuns, como dividir o projeto em iterações, medir velocidade de desenvolvimento, realizar reuniões ao início das iterações, entre outras. Porém, pode-se chegar à conclusão de que o XP tem algumas práticas que não são cobertas pelo SCRUM; essas práticas são pertinentes à programação do código-fonte. Porém, segundo Kniberg (2010), mesmo que o XP contenha a maioria das práticas do SCRUM, há algumas específicas do XP (como afirmado anteriormente) voltadas à atividade de programação, o que permite o uso do XP junto ao SCRUM, uma vez que o SCRUM descreve práticas que dão um enfoque maior no planejamento e organização.

4.2 MPA-DS (Modelo de Processo Ágil de Desenvolvimento de Software)

Utilizando as práticas apresentadas, foi possível criar um modelo de processo de desenvolvimento de software intitulado Modelo de Processo Ágil de Desenvolvimento de Software (MPA-DS), com base nas práticas das metodologias ágeis XP e SCRUM. Para a elaboração deste modelo foi usado o *framework* Rational Unified Process (RUP) que, segundo Sommerville(2003), é um exemplo de modelo de processo moderno que foi derivado do trabalho sobre UML e do Processo Unificado de Desenvolvimento de Software (Rumbaugh, et al., 1999b apud SOMMERVILLE(2003)). É um bom exemplo de modelo híbrido de processo, por trazer elementos de todos os modelos genéricos de processo, apóia a iteração e ilustra boas práticas de especificação e projeto. As fases do RUP são: (1) Concepção: o objetivo dessa fase é estabelecer um *business case* para o sistema; (2) Elaboração: tem como objetivos o entendimento do domínio do problema, o estabelecimento de um *framework* de arquitetura para o sistema, e o desenvolvimento do plano de projeto e identificação dos principais riscos do projeto; (3) Construção: esta fase está essencialmente relacionada ao projeto, programação e teste do sistema; (4) Transição: trata-se da fase final do RUP e está relacionada à transferência do sistema da comunidade de desenvolvimento para a comunidade dos usuários.

O modelo de processo MPA-DS apresenta suas fases (com base nas fases do RUP) e pode ser visualizado na Figura 1.

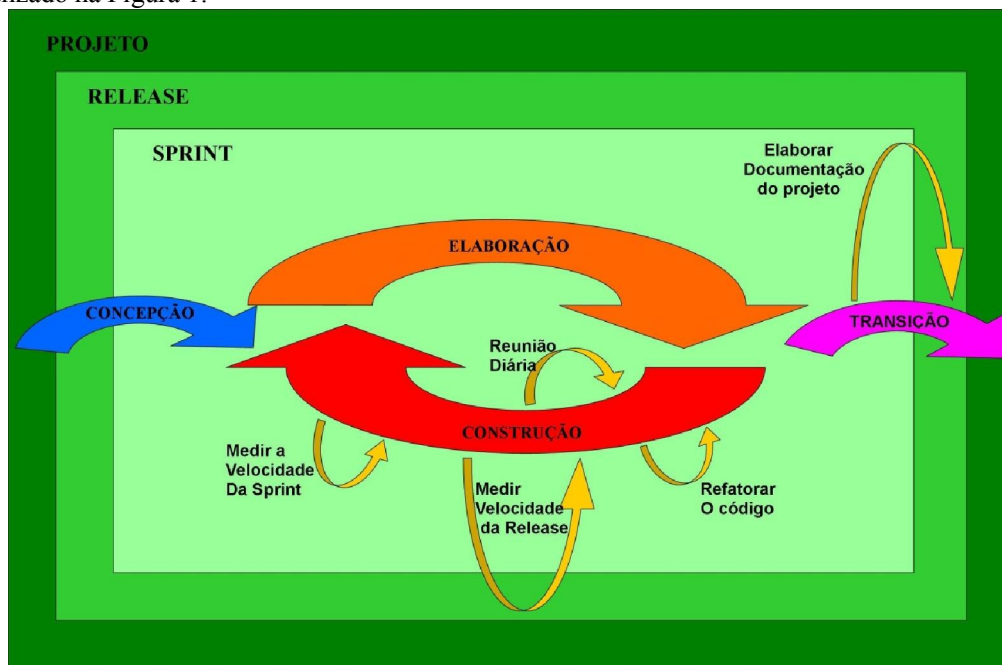


Figura 1 - Modelo de Processo Ágil de Desenvolvimento de Software (MPA-DS)

As fases do MPA-DS são: Concepção (CP), Elaboração (E), Construção (CT) e Transição (T). O projeto se inicia na fase de concepção, onde é dividido em partes menores, as *releases*, que serão decompostas em partes menores ainda, as *sprints*, que são as iterações do processo. As fases de elaboração e construção formam uma espécie de *looping*, já que pode-se passar por essas fases várias vezes durante a realização de um projeto. O projeto se encerra na fase de transição, na qual as *releases* saem do ambiente de produção e são colocadas em funcionamento em seu ambiente real.

Cada fase do modelo de processo MPA-DS é composta por atividades. Recomenda-se que essas atividades sejam seguidas cronologicamente, porém algumas devem ser realizadas diariamente, ou sempre que for possível e necessário; essas atividades foram destacadas no modelo MPA-DS (Figura 1): a reunião diária a ser realizada todos os dias ao longo do projeto; as atividades medir velocidade da *sprint* e medir velocidade da *release* que devem ser realizadas ao longo do projeto para se obter o controle do quanto de trabalho está sendo realizado; a atividade refatorar o código que deve ser realizada sempre que a equipe de desenvolvimento tiver tempo entre as atividades, afim de manter o código limpo. Todos os documentos que surgirem ao longo do projeto devem ser anexados a documentação do projeto.

Fase de Concepção

CP01: Escrever o Product Backlog

Descrição: O objetivo desta atividade é desenvolver a listagem das estórias que o sistema deve ter, definindo um número de identificação, um nome e a prioridade de cada estória.

Entradas: Requisitos do sistema.

Saídas: Product Backlog (ordenado por prioridade).

Recursos: Planilha Eletrônica.

Responsável: Product Owner.

Tarefas Específicas: Listar estórias necessárias para o sistema; Definir um nível de prioridade para cada estória.

Observações: (1) É importante lembrar que o Scrum Master é responsável por fazer com que a equipe coloque em prática as regras do SCRUM, portanto ele não aparece como responsável nas atividades, mas está presente na maioria delas; (2) Assim com o *scrum master* no SCRUM, há um *coach* no XP.

CP02: Dividir o projeto em releases

Descrição: O objetivo desta atividade é agrupar estórias do Product Backlog dividindo o projeto em releases.

Entradas: Product Backlog.

Saídas: Plano de liberação de Releases.

Recursos: Editor de texto.

Responsável: Time de desenvolvimento; Product Owner.

Tarefas Específicas: Estabelecer um tamanho fixo e um objetivo para cada release; Agrupar as estórias de maior prioridade para que formem um release; Definir a ordem de liberação dos releases de acordo com a necessidade do Product Owner e dependências técnicas.

Observação: Ao decorrer do projeto, o Product Owner pode alterar a ordem das *releases* e substituir estórias existentes, tendo em vista que o MPA-DS trabalha com base em metodologias ágeis que possuem o escopo do projeto aberto.

Fase de Elaboração

E01: Planejar a release

Descrição: O objetivo desta atividade é estabelecer o plano de atuação da *release* estimando a meta, riscos, características, possíveis resultados e data de entrega.

Entradas: Plano de liberação de *releases*; Arquivo de código (Última versão disponível).

Saídas: Plano de atuação para *release*.

Recursos: Editor de Texto.

Responsável: Time de desenvolvimento; Product Owner.

Tarefas Específicas: Separar as entregas e sub-entregas da *release*; Definir as atividades (macro) para obtenção das sub-entregas e entregas; Estimar riscos dessas atividades; Definir data de entrega do *release*.

E02: Dividir a release em Sprints.

Descrição: O objetivo desta atividade é dividir uma *release* em unidades de tempo menores, as *sprints*.

Entradas: Plano de atuação para *release*.

Saídas: Plano de atuação para release (dividido em sprints); Sprints Backlog.

Recursos: Editor de Texto.

Responsável: Time de desenvolvimento; Product Owner.

Tarefas Específicas: Analisar as histórias da release; Dividir as histórias em sprint.

E03: Planejar a Sprint

Descrição: O objetivo desta atividade é reunir o time para definir como os itens do sprint backlog serão feitos na Sprint.

Entradas: Sprint Backlog; Plano de atuação para release (dividido em sprints); Incremento do sistema (mais recente do produto se houver).

Saídas: Sprint Backlog e definição de horário e local da reunião diária.

Recursos: Editor de Texto.

Responsável: Time de desenvolvimento.

Tarefas Específicas: Definir um tempo de duração para sprint; Identificar tarefas menores, que detalham como transformar o Product Backlog em software funcional; Agrupar essas tarefas menores formando o Sprint Backlog.

E04: Criar o quadro de tarefas

Descrição: O objetivo desta atividade é criar um quadro que indica qual o status de desenvolvimento das tarefas da Sprint.

Entradas: Sprint Backlog.

Saídas: Quadro de tarefas.

Recursos: Quadro, post-its.

Responsável: Time de desenvolvimento.

Tarefas Específicas: Escrever as tarefas nos *post-it*; definir status das tarefas. Sugestão: Não Iniciado, Iniciado, Pronto; Criar divisões dos status no quadro de tarefas; Colocar cada *post-it* com as tarefas escritas em seus respectivos *status*.

Fase de Construção

CT01: Programar (Em par)

Descrição: O objetivo desta atividade é codificar o sistema com pares de programadores.

Entradas: Quadro de tarefas.

Saídas: Arquivo de código.

Recursos: IDE de desenvolvimento.

Responsável: Time de desenvolvimento.

Tarefas Específicas: Implementar cada tarefa escolhida no quadro de tarefas, mantendo os padrões de codificação.

CT02: Escrever testes de unidade

Descrição: O objetivo desta atividade é escrever testes de unidade enquanto o sistema é codificado.

Entradas: Quadro de tarefas e arquivo de código. *Saídas:* Arquivo de testes e critérios de testes.

Recursos: Ferramentas de teste

Responsável: Time de desenvolvimento.

Tarefas Específicas: Escrever testes de unidade para cada parte do código implementado.

Observação: As atividades CT01 e CT02 podem ser executadas em paralelo, já que o XP defende a programação orientada a testes.

CT03: Realizar teste de unidade

Descrição: O objetivo desta atividade é realizar os testes de unidade nos métodos codificados, identificando erros.

Entradas: Arquivo de testes.

Saídas: Relatório de teste; Arquivo de código (testado).

Recursos: Ferramentas de teste.

Responsável: Time de desenvolvimento.

Tarefas Específicas: Executar teste no código selecionado; Avaliar resultados dos testes; Corrigir possíveis erros.

CT04: Integrar o código no repositório

Descrição: O objetivo desta atividade é integrar o código no repositório, após codificação e realização de testes de unidade.

Entradas: Arquivo de código (testados).

Saídas: Arquivo de código armazenado no repositório.

Recursos: Repositório do sistema.

Responsável: Time de desenvolvimento.

Tarefas Específicas: Integrar o código no repositório sempre que houver alguma parte do código pronta e testada; Realizar testes de integração; Manter no repositório sempre a última versão do

sistema.

CT05: Revisar a Sprint.

Descrição: O objetivo desta atividade é apresentar a nova funcionalidade que acabou de ser terminada e seus resultados aos *stakeholders*.

Entradas: Última versão do produto (disponível no repositório); Plano de Atuação para release (atualizado).

Saídas: Última versão do produto (disponível no repositório); Relatório de revisão da sprint.

Recursos: Projetor multimídia; Editor de texto.

Responsável: Time de desenvolvimento; Product Owner; Stakeholders.

Tarefas Específicas: Reunir o time e os stakeholders; O time de desenvolvimento apresenta a nova funcionalidade aos stakeholders; O time de desenvolvimento deve responder aos stakeholders perguntas pertinentes a projeções de datas, quais as partes do projeto que estão prontas e quais as próximas etapas do projeto.

CT06: Fazer retrospectiva da Sprint

Descrição: O objetivo desta atividade é fazer uma retrospectiva da *sprint* demonstrando os pontos fortes e fracos em relação a pessoas, regras SCRUM, ferramentas e processos de cada iteração.

Entradas: Instância do MPA-DS (Modelo de Processo Ágil para Desenvolvimento de Software) para o projeto em questão.

Saídas: Lista de melhorias.

Recursos: Editor de texto.

Responsável: Time de desenvolvimento.

Tarefas Específicas: Reunir a equipe; Identificar melhorias para o processo em se tratando de métodos de comunicação, ferramentas, composição do time, preparativos para reuniões.

Observação: A atividade CT06 é necessária, pois permite que o time consiga modelar e se adaptar ao processo.

CT07: Realizar Reuniões Diárias

Descrição: O objetivo desta atividade é reunir a equipe diariamente em frente ao quadro de tarefas para discussão do que cada membro do time fez no dia anterior e o que fará no dia que se inicia.

Entradas: Quadro de tarefas.

Saídas: Quadro de tarefas atualizado.

Recursos: Quadro de tarefas, sala disponível todos os dias para esta reunião.

Responsável: Time de desenvolvimento.

Tarefas Específicas: Cada desenvolvedor apresenta o que realizou no dia anterior e quais as dificuldades encontradas na realização desta tarefa; Cada desenvolvedor escolhe o que fará no dia que se inicia; Realocar as tarefas nos seus *status* adequados no quadro de tarefas.

Observação: Tanto o SCRUM quanto o XP, sugerem que na reunião diária, todo o time permaneça em pé para que todos prestem atenção e a reunião não demore muito para terminar. Sugere-se, ainda, que esta reunião não ultrapasse 15 minutos.

CT08: Medir a velocidade da Sprint

Descrição: O objetivo desta atividade é registrar graficamente o quanto de trabalho resta no Sprint Backlog.

Entradas: Sprint Backlog e arquivo de código.

Saídas: Gráfico de Burndown da Sprint.

Recursos: Planilha Eletrônica.

Responsável: Scrum Master.

Tarefas Específicas: Analisar as atividades realizadas; Analisar as atividades não realizadas; Traçar graficamente quantos dias há disponíveis para o fim da sprint e quanto de trabalho falta ser realizado do sprint backlog.

CT09: Refatorar o código

Descrição: O objetivo desta atividade é refatorar e manter o código limpo e simples.

Entradas: Arquivo de código.

Saídas: Arquivo de código refatorado.

Recursos: Repositório do sistema.

Responsável: Time de desenvolvimento.

Tarefas Específicas: Refatorar o código sempre que algo desnecessário for encontrado.

CT10: Medir a velocidade da Release

Descrição: O objetivo desta atividade é quantificar o quanto falta para ser feito do product backlog.

Entradas: product backlog.

Saídas: Gráfico de burndown da release.

Recursos: Planilha Eletrônica.

Responsável: Scrum Master.

Tarefas Específicas: Analisar as atividades realizadas; Analisar as atividades não realizadas; Traçar graficamente quantos dias há disponíveis para o fim da release e quanto de trabalho falta do product backlog.

Observação: As atividades CT07, CT08, CT09, CT10 não necessitam ser realizadas nessa ordem, pois servem de apoio para as outras. Com exceção da atividade CT07, que deve ser realizada diariamente, as outras atividades devem ser realizadas sempre que a equipe tiver tempo livre entre as atividades.

Fase de Transição

T01: Escrever Testes de Aceitação

Descrição: O objetivo desta atividade é escrever teste de aceitação ao fim de cada sprint ou release.

Entradas: Arquivo de código (repositório).

Saídas: Testes de aceitação.

Recursos: Ferramentas de teste e critérios de testes. *Responsável:* Time de desenvolvimento.

Tarefas Específicas: Escrever testes de aceitação, a partir dos sprints backlogs.

T02: Executar Testes de Aceitação

Descrição: O objetivo desta atividade é executar os testes de aceitação da release.

Entradas: Arquivo de código (repositório); teste de aceitação.

Saídas: Arquivo de código (testado); Relatório de teste.

Recursos: Ferramentas de teste.

Responsável: Time de desenvolvimento.

Tarefas Específicas: Executar o teste de aceitação; Identificar possíveis erros; Corrigir erros.

T03: Elaborar documentação do projeto.

Descrição: O objetivo desta atividade é documentar o projeto.

Entradas: Product backlog; Plano de liberação de releases; Plano de atuação de cada release; sprint backlog; Gráfico de burndown da sprint; Gráfico de burndown da release; Arquivo de código; Arquivos de teste.

Saídas: Documentação projeto.

Recursos: Editor de texto.

Responsável: Pessoa designada para documentação (Membro do time de desenvolvimento).

Tarefas Específicas: Agrupar a documentação que vai surgindo durante as sprints e releases; Elaborar manual do usuário.

Observação: Mesmo ao trabalhar com metodologias ágeis é necessária uma documentação do projeto, porém estas metodologias utilizam uma documentação não tão “pesada” como as metodologias tradicionais. A atividade T03 não precisa seguir a ordem cronológica do projeto, devendo ser realizada sempre que surgir documentos do projeto.

T04: Implantar o Release na empresa

Descrição: O objetivo desta atividade é implantar o release finalizado pelos desenvolvedores para que os usuários possam ir se adaptando ao sistema e manter o *feedback* constante.

Entradas: Arquivo de código (Última versão do sistema); Documentação do projeto.

Saídas: Release implantada na empresa.

Recursos: Máquinas para instalação do sistema; Material para ministrar treinamento.

Responsável: Equipe de implantação.

Tarefas Específicas: Instalar a release ou última versão do sistema na empresa; Entregar documentação; Ministrar treinamento.

5. CONCLUSÃO

A engenharia de software surgiu para dar um tratamento sistemático e controlado ao desenvolvimento de software. Porém, micro e pequenas empresas (MPes) ainda encontram

dificuldades na utilização de modelos de processo existentes. Muitas vezes, essas dificuldades referem-se ora à forte ênfase na documentação do sistema sugerida pelos modelos tradicionais e ora à falta de detalhamento das atividades a serem realizadas. Na tentativa de contornar essas dificuldades, este trabalho propôs a utilização das metodologias ágeis XP e SCRUM em conjunto, tendo como resultado o Modelo de Processo Ágil de Desenvolvimento de Software – MPA-DS.

Por utilizar as metodologias ágeis, o modelo MPA-DS propõe a minimização da documentação por meio da ênfase na comunicação entre os envolvidos no projeto. Além disso, esse modelo fornece um maior detalhamento das atividades do processo por meio da definição de *tarefas* a serem realizadas, de *entradas* necessárias para a realização das atividades, de *saídas* resultantes da realização das atividades e de *recursos* que podem ser utilizados para a realização das atividades.

REFERÊNCIAS

ANDRADE, Maria Margarida de. **Introdução à metodologia do trabalho científico**: elaboração de trabalhos na graduação. 8 ed. São Paulo: Atlas, 2007.

BERNI, Jean Carlo Albieri. **Gestão para o processo de desenvolvimento de software científico, utilizando uma abordagem ágil e adaptativa na micro empresa**. Disponível em: <http://cascavel.cpd.ufsm.br/tede/tde_arquivos/12/TDE-2010-05-07T112147Z-2605/Publico/BERNI,%20JEAN%20CARLO%20ALBIERO.pdf>. Acesso em: 30 set. 2010.

IMPROVE IT. **SCRUM** (2009). Disponível em: <<http://improveit.com.br/scrum>>. Acesso em: 8 jun. 2010.

JUBILEU, Andrea Padovan. **Modelo de Gestão de Venda e Desenvolvimento de Software On-Demand para MPEs** (2008). Disponível em <<http://www.teses.usp.br/teses/disponiveis/18/18140/tde-09022009-131035/pt-br.php>>. Acesso em: 25 mai. 2010.

KNIBERG, Henrik. **Scrum e XP direto das trincheiras**: Como nós fazemos Scrum (2007). Disponível em <<http://infoq.com/br/minibooks/scrum-xp-fromthe-trenches>>. Acesso em: 30 abr. 2010.

KOSCIASNKI, André ; SOARES, Michel dos Santos. **Qualidade de software**: Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. São Paulo: Novatec Editora, 2007.

SEBRAE. **Critérios e conceitos para classificação de empresas**. Disponível em: <http://www.sebrae.com.br/customizado/estudos-e-pesquisas/integra_bia?ident_unico=97>. Acesso em: 16 out. 2010.

SEVERINO, Antonio Joaquim. **Metodologia do trabalho científico**. 23 ed. São Paulo: Cortez, 2007.

SOMERVILLE, Ian. **Engenharia de Software**. 6 ed. São Paulo: Addison Wesley, 2003.

TELES, Vinícius Manhães. **Extreme Programming**: Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade. São Paulo: Novatec, 2006.

WELLS, Don. **The Rules of Extreme Programming** (1999). Disponível em: <<http://www.extremeprogramming.org/rules.html>>. Acesso em: 30 mai. 2010.