UNIVERSIDADE DE CAXIAS DO SUL

FELIPE MACHADO SCHMITT

UM PROCESSO DE *SOFTWARE* PARA EQUIPES PEQUENAS

CAXIAS DO SUL

2012

UNIVERSIDADE DE CAXIAS DO SUL

FELIPE MACHADO SCHMITT

UM PROCESSO DE *SOFTWARE* PARA EQUIPES PEQUENAS

Trabalho de Conclusão de Curso
do Título de Bacharel em Sistemas de Informação
pela Universidade de Caxias do Sul
Orientador Prof. Giovanni Ely Rocco

CAXIAS DO SUL
2012

AGRADECIMENTOS

Agradeço a Deus por proporcionar todas as condições necessárias para as minhas conquistas.

Aos meus pais, Leonir Antonio Schmitt e Neli Machado Schmitt pelo apoio em todos os momentos da minha vida, por todo o amor que me deram, por tudo que me ensinaram, tendo sido exemplos de caráter e honestidade e por terem me mostrado a importância do estudo e do conhecimento.

A minha namorada Larissa Giacomet por todo amor, apoio e compreensão que me fazem ter determinação para realizar os meus sonhos.

Ao meu orientador Giovanni Ely Rocco pela orientação no desenvolvimento deste trabalho e por ter me apresentado uma vasta gama de conhecimento.

A equipe que trabalhou comigo na aplicação do estudo de caso pelo apoio e colaboração.

RESUMO

No mercado de desenvolvimento de software existem muitas organizações que trabalham com equipes pequenas, muitas destas organizações buscam formas de organizar e gerenciar o processo de desenvolvimento de software, sem consumir recursos desnecessários da equipe e tendo uma visão da qualidade do processo. Este trabalho propõe um processo de *software* para equipes pequenas, cujos integrantes trabalham próximos e possuem uma comunicação direta. Para equipes pequenas que, além de implantar um processo de *software*, também pretendem obter certificação no modelo de melhoria de processos MPS.BR, este trabalho compara os resultados esperados pelo nível G do MPS.BR com os resultados obtidos com o processo proposto, demonstrando que o processo pode ser adaptado para este objetivo. Para elaborar o processo proposto, foram estudados os conceitos principais das metodologias ágeis, que são indicadas para equipes pequenas. O método ágil escolhido para estudo aprofundado foi o *Scrum*, do qual foram inspiradas muitas das técnicas descritas no processo proposto.

Palavras-Chaves: processo de software, método ágil, Scrum, MPS.BR.

ABSTRACT

In the software development world there are many organizations working with small teams, many of these organizations are looking for ways to organize and manage the process of software development, without consuming unnecessary team resources and having a vision of process quality. This paper proposes a software process for small teams, whose members work closely and have direct communication. For small teams that, in addition to implementing a software process, also intend to obtain certification in process improvement model MPS.BR, this paper compares the results expected by the G level of MPS.BR with the results obtained with the proposed process, demonstrating that the process can be adapted for this purpose. To develop the proposed process, we studied the key concepts of agile methodologies, which are suitable for small teams. The agile method chosen for detailed study was Scrum, which were inspired many of the techniques described in the proposed process.

Keywords: software process, agile method, SCRUM, MPS.BR.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

ETM	Equipe Técnica do Modelo
FCC	Fórum de Credenciamento e Controle
GRE	Gerência de Requisitos
GPR	Gerência de Projetos
MA	Modelagem Ágil
MA-MPS	Método de Avaliação
MN-MPS	Modelo de Negócio
MPS.BR	Melhoria de Processo do Software Brasileiro
MR-MPS	Modelo de Referência

SUMÁRIO

1 INTRODUÇÃO

O *software* nasce a partir de uma necessidade. Após a análise desta necessidade um longo caminho é percorrido até que ele tome forma e atenda os resultados esperados. Ao longo dos anos, estas necessidades que dão origem aos *softwares* vêm se tornando cada vez mais complexas e variadas, tornando o caminho entre o surgimento da necessidade e o *software* pronto igualmente complexo.

Para auxiliar no desenvolvimento de *software* foram definidos ao longo do tempo diversos processos de *software*. Pressman (2006) define processo de *software* como um roteiro que ajuda a criar a tempo um resultado de alta qualidade. Estes processos definem práticas a serem seguidas, que podem ser adaptadas conforme a necessidade, e que fornecem organização e controle para as atividades.

Existem diversos tipos de organizações, cada um com suas características e especificidades. Por tanto, para cada tipo de organização existem processos que melhor se adaptam. O tamanho das equipes com as quais as organizações trabalham e a forma como a comunicação ocorre entre os seus integrantes são fatores importantes na escolha do processo a ser utilizado.

Trabalhar com o processo ideal para a equipe é de extrema importância. Segundo Ambler (2004), nas estratégias adotadas podem-se encontrar dois extremos, no primeiro a modelagem não existe, o que freqüentemente resulta em retrabalho quando o *software* demonstra ter sido mal projetado. O outro extremo ocorre quando se produz documentação e modelagem em excesso, o que faz com que seus esforços de desenvolvimento caminhem de forma lenta. Pressman (2006) diz que a engenharia de software é realizada por pessoas criativas, com amplos conhecimentos, que devem adaptar um processo de software maduro apropriado para os produtos que elas constroem e para as demandas de seu mercado.

O que a organização vislumbra para o futuro também deve ser considerado na implantação de um processo de *software.* O processo deve atender as atuais necessidades da organização e atender ou ser capaz de se adaptar as futuras necessidades.

Uma das principais necessidades das organizações que desenvolvem *software* é a qualidade do *software*. Segundo Rocha (2001), grande parte da população mundial depende de aplicações de software para realizar suas atividades diárias. Se alguns sistemas de uso global deixarem de funcionar, uma grande parte da população mundial sofrerá as conseqüências. Melhorar a qualidade do *software* é o principal objetivo da engenharia de

software. Implantar um processo de *software* que esteja de acordo com as necessidades e as características da organização contribui para atingir este objetivo.

Algumas organizações podem ir além de implantar um processo de *software* em busca de qualidade. Estas organizações podem buscar uma certificação em algum modelo de qualidade, neste caso o processo de *software* utilizado precisa atender as normas do modelo de qualidade ou ser capaz de se adaptar de forma que atenda estas normas.

Da mesma forma que o processo de *software* deve estar de acordo com as características de empresa, o modelo de qualidade ao qual a empresa pretende se certificar, assim como o grau de certificação que a empresa pretende obter, também devem levar em consideração estas características.

1.1 OBJETIVOS

Este trabalho tem como objetivo geral apresentar um processo de *software* para equipes pequenas cujos integrantes trabalham próximos e possuem uma comunicação direta. Além de comparar os resultados esperados pelo nível G do MPS.BR com os resultados do processo, afim de demonstrar quanto o processo proposto atende ao modelo e quanto ele teria que ser adaptado para este fim. O processo proposto é baseado no *framework* SCRUM.

Para atingir o objetivo geral, este trabalho está organizado visando os seguintes objetivos específicos:

1 – Estudar processo de software, modelagem ágil, o *framework* SCRUM, qualidade de *software* e o modelo de qualidade MPS.BR;

2 – Propor um processo de *software* baseado na metodologia estudada;

3 – Verificar quanto o processo proposto atende aos resultados esperados pelo nível G do MPS.BR;

4 – Avaliar o processo proposto em um estudo de caso;

1.2 ORGANIZAÇÃO DO TRABALHO

O trabalho está organizado em 6 capítulos, este é o capítulo inicial e apresenta os temas estudados e os objetivos para a elaboração do trabalho.

O capítulo 2 aborda as definições gerais de processo de software, modelagem ágil e apresenta o *framework Scrum*.

O capítulo 3 discorre sobre qualidade de software e apresenta o modelo MPS.BR, com uma explicação mais detalhada do nível G do modelo.

O capítulo 4 apresenta o processo de software proposto neste trabalho.

O capítulo 5 descreve a aplicação do processo de software proposto em um estudo de caso.

O capítulo 6 apresenta a conclusão do trabalho.

2 SCRUM

O *Scrum* é um *framework* que define conceitos e práticas para a implantação de um processo de desenvolvimento de *software*. Para entender melhor ao que o *Scrum* se destina é importante entender os conceitos de processo de *software*. Pressman (2006) diz que vários autores já descreveram definições pessoais de engenharia de *software*, mas uma definição proposta por Fritz Bauer na conferência pioneira sobre o assunto ainda serve de base para a discussão: Engenharia de software é a criação e a utilização de sólidos princípios de engenharia a fim de obter softwares econômicos que sejam confiáveis e que trabalhem eficientemente em máquinas reais.

Pressman (2006) ainda apresenta uma definição mais abrangente ao dizer que engenharia de *software* é a aplicação de uma abordagem sistemática, disciplinada e quantificável, para o desenvolvimento, operação e manutenção do *software*, isto é, a aplicação de engenharia ao *software.*

O processo de software é definido por Sommerville (2011) como um conjunto de atividades que levam a produção de um produto de software. Essas atividades podem envolver o desenvolvimento do software a partir do zero em uma linguagem padrão de programação. No entanto, aplicações de negócios não são necessariamente desenvolvidas dessa forma. Atualmente, novos softwares de negócio são desenvolvidos por meio da extensão e modificação de sistemas existentes ou por meio da configuração e integração de prateleira ou componentes de sistema.

No entanto, o que é sistemático, disciplinado e quantificável para uma equipe de software pode ser cansativo para outra. Precisamos de disciplina, mas precisamos também de adaptabilidade e agilidade.

2.1 DESENVOLVIMENTO ÁGIL

A engenharia de software convenciona possui algumas limitações e os métodos ágeis nasceram com o intuito de superar estas dificuldades. Segundo Pressman (2006), o desenvolvimento ágil não é contrário à sólida prática de engenharia de software e pode ser aplicado como uma filosofia prevalecente a todo o trabalho de software.

Para Ambler (2004) as estratégias convencionais de modelagem muitas vezes não se mostram funcionais. A definição de modelagem ágil de Ambler (2004) é de um processo

baseado na prática que descreve como um modelador ágil deve ser. A engenharia ágil de software é uma alternativa para um mundo de software apressado e sempre mutável.

A aliança ágil, que é formada por Kent Beck e 16 outros notáveis desenvolvedores, produtores e consultores de software, assinaram em 2001 o “Manifesto para o desenvolvimento ágil de software”. Eles declararam:

Estamos descobrindo melhores modos de desenvolvimento de software fazendo-o e ajudando outros a fazê-lo. Por meio desse trabalho passamos a valorizar:

Indivíduos e interações em vez de processos e ferramentas.

Softwares funcionando em vez de documentação abrangente.

Colaboração do cliente em vez de negociação de contratos.

Resposta a modificações em vez de seguir planos.

Isto é, ainda que haja valor nos itens a direita, valorizamos mais os itens a esquerda.

A engenharia de software ágil, segundo Pressman (2006), combina uma filosofia e um conjunto de diretrizes de desenvolvimento. A filosofia encoraja a satisfação do cliente e a entrega incremental do software logo de início, equipes de projeto pequenas, altamente motivadas, métodos informais, produtos de trabalho de engenharia de *software* mínimos e simplicidade global do desenvolvimento. Pressman (2006) ainda salienta que as diretrizes de desenvolvimento enfatizam a entrega em contraposição à análise, ao projeto e a comunicação ativa e contínua entre desenvolvedores e clientes.

Uma das principais características das modelagens ágeis é a maneira como ela atua com as mudanças ao longo do processo, o acolhimento de modificações é o principal guia para a agilidade. Os engenheiros de software devem reagir rapidamente se tiverem de acomodar as rápidas modificações.

A aliança ágil define 12 princípios para aqueles que querem alcançar agilidade:

1. Nossa maior prioridade é satisfazer ao cliente desde o início por meio de entrega contínua de software valioso.
2. Modificações de requisitos são bem vindas, mesmo que tardias no desenvolvimento. Os processo ágeis aproveitam as modificações como vantagens para a competitividade do cliente.
3. Entrega de software funcionando freqüentemente, a cada duas semanas até dois meses, de preferência no menor espaço de tempo.
4. O pessoal de negócio e os desenvolvedores devem trabalhar juntos diariamente durante todo o projeto.

5. Construção de projetos em torno de indivíduos motivados. Forneça-lhes o ambiente e apoio que precisam e confie que eles farão o trabalho.
6. O método mais eficiente e efetivo de levantar informação para dentro de uma equipe de desenvolvimento é a conversa face a face.
7. Software funcionando é a principal medida de progresso.
8. Processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante, indefinidamente.
9. Atenção contínua à excelência técnica e ao bom projeto facilitam a agilidade.
10. Simplicidade – a arte de maximizar a quantidade de trabalho não efetuado – é essencial.
11. As melhores arquiteturas, requisitos e projetos surgem de equipes auto-organizadas.
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais efetiva, então sintoniza e ajusta adequadamente seu comportamento.

Para Ambler (2004), um modelador ágil é qualquer um que siga a metodologia ágil, aplicando as práticas desta de acordo com os seus princípios e valores.

A agilidade pode ser aplicada a qualquer processo de *software*. Entretanto, para conseguir isso, Pressman (2006) afirma que é essencial que o processo seja projetado de modo que permita à equipe de projeto adaptar tarefas e aperfeiçoá-las, conduzir o planejamento para que se entenda a fluidez de uma abordagem de desenvolvimento ágil, eliminar tudo menos os produtos de trabalho mais essenciais e mantê-los simples.

Um processo ágil deve atender a três suposições que, segundo Pressman (2006), são:

1. É difícil prever antecipadamente quais requisitos de software vão persistir e quais serão modificados. É igualmente difícil prever como as prioridades do cliente serão modificadas à medida que o projeto prossegue.
2. Para muitos tipos de software, o projeto e a construção são intercalados, isto é, as duas atividades devem ser realizadas juntas de modo que os modelos de projeto sejam comprovados à medida que são criados. É difícil prever o quanto de projeto é necessário antes que a construção seja usada para comprovar o projeto.
3. Análise, projeto, construção e testes não são tão previsíveis como gostaríamos.

2.2 *FRAMEWORK SCRUM*

Modelo de processo de software é definido por Sommerville (2011) como uma representação simplificada de um processo de software. Cada modelo representa uma perspectiva particular de um processo e, portanto, fornece informações parciais sobre ele. Sommerville (2011) prossegue dizendo que estes modelos podem ser vistos como um *framework*, pois esses modelos não são descrições definitivas dos processos de software. São abstrações que podem ser usadas para explicar diferentes abordagens de desenvolvimento de *software.*

No desenvolvimento ágil existem vários modelos. Segundo uma pesquisa realizada pela *VersionOne* (2010) o modelo ágil mais utilizado no mundo é o *Scrum*. Segundo a pesquisa, realizada com participantes de 91 países, o Scrum é utilizado por 58% das empresas que utilizam metodologias ágeis.

O *Scrum* foi desenvolvido por Jeff Sutherland e por sua equipe no início da década de 1990. Scwaber e Beedle também são responsáveis pelo desenvolvimento do *Scrum* nos últimos anos. Pressman (2006) salienta que os princípios *Scrum* são consistentes com o manifesto ágil, são eles:

1. Pequenas equipes de trabalho são organizadas de modo a maximizar a comunicação, minimizar a supervisão e maximizar o compartilhamento de conhecimento tácito informal.
2. O processo precisa ser adaptável tanto a modificações técnicas quanto de negócio para garantir que o melhor produto possível seja produzido.
3. O processo produz freqüentes incrementos de software que pode ser inspecionados, ajustados, testados, documentados e expandidos.
4. O trabalho de desenvolvimento e o pessoal que o realiza é dividido em partições claras, de baixo acoplamento, ou em pacotes.
5. Testes e documentação constantes são realizados à medida que o produto é construído.
6. O processo *Scrum* fornece a habilidade de declarar o produto pronto sempre que necessário.

Dos artefatos gerados pelo *Scrum* existe o *product backlog* que é uma lista priorizada de requisitos. Itens podem ser adicionados ao *backlog* a qualquer momento. O gerente do produto avalia a pendência e atualiza as prioridades quando necessário.

É gerado também o *sprint backlog*, Schwaber e Sutherland (2010) definem *sprint backlog* como uma lista de tarefas para transformar o *product backlog*, por uma *Sprint*, em um incremento do produto potencialmente entregável.

São gerados dois gráficos, o gráfico de *Burndown* da *Release* que, segundo Schwaber e Sutherland (2010), registra a soma das estimativas dos esforços restantes do *Product Backlog* ao longo do tempo, e o *Burndown* do *Sprint*, que é um gráfico da quantidade restante de trabalho do *Sprint Backlog* em uma determinada *Sprint* ao longo do tempo da *Sprint*.

O *Scrum* incorpora as seguintes atividades: requisitos, análise, projeto, evolução e entrega. Os ciclos de atividades são chamados de *Sprint*. Pressman (2006) define os *Sprints* como unidades de trabalho que são necessárias para satisfazer a um requisito definido no *product backlog* que precisa ser cumprido em um intervalo de tempo predefinido. Não são introduzidas modificações durante a *Sprint*. Assim trabalha-se em um ambiente de curto prazo, mas estável.

Entre as atividades está a reunião de planejamento da *Sprint* que para Schwaber e Sutherland (2010) consiste de duas partes. A primeira parte é o momento no qual é decidido o que será feito na *Sprint*. A segunda parte é o momento no qual o Time entende como desenvolverá essa funcionalidade em um incremento do produto durante a *Sprint*.

Ao longo da *Sprint* ocorrem as reuniões diárias, que são reuniões curtas (normalmente de 15 minutos), feitas pela equipe *Scrum*. Pressman (2006) salienta três questões chave que devem ser formuladas e respondidas por todos os membros da equipe:

1. O que você fez desde a última reunião da equipe?
2. Que obstáculos você está encontrando?
3. O que você planeja realizar até a próxima reunião de equipe?

A reunião de revisão da *Sprint* acontece ao final da *Sprint*. “Durante a Revisão da *Sprint*, o Time de *Scrum* e as partes interessadas colaboram sobre o que acabou de ser feito” (Schwaber e Sutherland, 2010, p. 14).

Depois da revisão da *Sprint* o Time de *Scrum* tem a reunião de retrospectiva da *Sprint*. Segundo Schwaber e Sutherland (2010), nessa reunião, o *Scrum Master* encoraja o Time a revisar, dentro do modelo de trabalho e das práticas do processo do *Scrum*, seu processo de desenvolvimento, de forma a torná-lo mais eficaz e gratificante para a próxima *Sprint*.

Os papéis no *Scrum* são definidos por Schwaber e Sutherland (2010) da seguinte forma:

1. *Scrum Master*: É responsável por garantir que o processo seja compreendido e seguido.
2. *Product Owner*: É responsável por maximizar o valor do trabalho que o Time deScrum faz.
3. Time: Executa o trabalho propriamente dito. O Time consiste em desenvolvedores com todas as habilidades necessárias para transformar os requisitos do *Product Owner* em um pedaço potencialmente entregável do produto ao final da *Sprint*.

3 QUALIDADE DE SOFTWARE

Manter a qualidade no produto de *software* é uma das maiores preocupações das organizações. Pressman (2006) diz que qualidade de *software* é a satisfação de requisitos funcionais e de desempenho explicitamente declarados, normas de desenvolvimento explicitamente documentadas e características implícitas que são esperadas em todo o *software* desenvolvido profissionalmente. Em complemento a isso, Rocha (2001) define qualidade de software como um conjunto de características que devem ser alcançadas em um determinado grau para que o produto atenda às necessidades dos usuários. É por meio desse conjunto de características que a qualidade de um produto de *software* pode ser medida e avaliada.

Entretanto, segundo Rocha (2001), a qualidade do produto de *software* está fortemente relacionada à qualidade do processo de *software*. Para muitos engenheiros de *software*, a qualidade do processo de *software* é tão importante quanto à qualidade do produto.

Para atingir a qualidade desejada em seus processos, muitas empresas optam por implantar algum modelo de qualidade. Segundo Rocha (2001), as características que devem ser alcançadas pelo *software* podem ser detalhadas em vários níveis de subcaracterísticas, chegando-se a um amplo conjunto de atributos que descrevem a qualidade de um processo de *software*. Torna-se necessário escolher um modelo que organize os atributos e permita avaliar a qualidade *software*. Esses modelos permitem entender como as diferentes facetas contribuem para a qualidade do produto como um todo.

Um modelo de qualidade que pode ser utilizado é o MPS.BR.

3.1 MPS.BR

O objetivo do programa MPS.BR é a “Melhoria de Processo do Software Brasileiro” (SOFTEX, 2011, p. 3).

“O programa MPS.BR conta com duas estruturas de apoio para o desenvolvimento de suas atividades, o Fórum de Credenciamento e Controle (FCC) e a Equipe Técnica do Modelo (ETM). Por meio destas estruturas, o MPS.BR obtém a participação de representantes de universidades, instituições governamentais, centros de pesquisa e de organizações privadas, os quais contribuem com suas visões complementares que agregam qualidade ao empreendimento” (SOFTEX, 2011, p.3).

O modelo MPS baseia-se nos conceitos de maturidade e capacidade de processo para a avaliação e melhoria da qualidade e produtividade de produtos de software e serviços correlatos. Dentro desse contexto, o modelo MPS possui três componentes: Modelo de Referência (MR-MPS), Método de Avaliação (MA-MPS) e Modelo de Negócio (MN-MPS).

O modelo MPS está descrito por meio de documentos em formato de guias:

O Guia de Implementação fornece orientações para implementar nas organizações os níveis de maturidade descritos no Modelo de Referência MR-MPS, detalhando os processos contemplados nos respectivos níveis de maturidade e os resultados esperados com a implementação dos processos.

O MPS.BR possui 7 níveis de maturidade, do nível G, o primeiro, até o nível A, o último.

3.1.1 Nível G

O nível G é o primeiro nível de maturidade do MR-MPS.

Como é definido pelo guia de implementação do MPS.BR (SOFTEX, 2011), ao final da implantação deste nível a organização deve ser capaz de gerenciar parcialmente seus projetos de desenvolvimento de software.

No nível G, o projeto pode usar os seus próprios padrões e procedimentos, não sendo necessário que se tenha padrões organizacionais. Se, porventura, a organização possuir processos já definidos e os projetos necessitarem adaptar os processos existentes, deve-se registrar essa adaptação durante o planejamento do projeto. Adaptações podem incluir alteração em processos, atividades, ferramentas, técnicas, procedimentos, padrões, medidas, dentre outras.

O MPS.BR define quais são os resultados esperados do processo mas não define como atingir estes resultados. A organização que pretende implantar o MPS.BR tem a liberdade para definir como pretende atingir os resultados esperados por ele.

O nível G define resultados esperados para dois processos, o processo de gerência de projetos e o processo de gerência de requisitos.

O MPS.BR (SOFTEX, 2011) define que o propósito do processo Gerência de Projetos é estabelecer e manter planos que definem as atividades, recursos e responsabilidades do projeto, bem como prover informações sobre o andamento do projeto que permitam a realização de correções quando houver desvios significativos no desempenho do projeto.

Os resultados esperados para o processo de Gerência de Projetos, segundo o guia de implementação (SOFTEX, 2011), são:

GPR1 - O escopo do trabalho para o projeto é definido: O escopo do projeto define todo o trabalho necessário, e somente ele, para entregar um produto que satisfaça as necessidades, características e funções especificadas para o projeto, de forma a concluí-lo com sucesso.

GPR2 - As tarefas e os produtos de trabalho do projeto são dimensionados utilizando métodos apropriados. O escopo do projeto, identificado na forma dos seus principais produtos de trabalho e das tarefas do projeto, deve agora ser decomposto em componentes menores, mais facilmente gerenciáveis e possíveis de serem dimensionados.

GPR3 - O modelo e as fases do ciclo de vida do projeto são definidos. O ciclo de vida de um projeto consiste de fases e atividades que são definidas de acordo com o escopo dos requisitos, as estimativas para os recursos e a natureza do projeto, visando oferecer maior controle gerencial.

GPR4 - (Até o nível F) O esforço e o custo para a execução das tarefas e dos produtos de trabalho são estimados com base em dados históricos ou referências técnicas. As estimativas de esforço e custo são, normalmente, baseadas nos resultados de análises utilizando modelos e/ou dados históricos aplicados ao tamanho, atividades e outros parâmetros de planejamento.

GPR5 - O orçamento e o cronograma do projeto, incluindo a definição de marcos e pontos de controle, são estabelecidos e mantidos. As dependências entre tarefas são estabelecidas e potenciais gargalos são identificados utilizando métodos apropriados.

GPR6 - Os riscos do projeto são identificados e o seu impacto, probabilidade de ocorrência e prioridade de tratamento são determinados e documentados. Projetos têm riscos e estes precisam ser identificados, analisados e priorizados. Para facilitar a identificação dos riscos, é interessante elaborar uma lista de riscos mais comuns a ser examinada pelo gerente do projeto e/ou equipe do projeto para identificar quais destes são potenciais riscos para o projeto em questão.

GPR7 - Os recursos humanos para o projeto são planejados considerando o perfil e o conhecimento necessários para executá-lo. O planejamento de recursos humanos determina funções, responsabilidades e relações hierárquicas do projeto.

GPR8 - (Até o nível F) Os recursos e o ambiente de trabalho necessários para executar o projeto são planejados. Este resultado faz referência à necessidade de se planejar, com base na EAP (ou estrutura equivalente), as tarefas e previstos os recursos e o ambiente necessários,

incluindo, por exemplo, equipamentos, ferramentas, serviços, componentes, viagens e requisitos de processo (processos especiais para o projeto).

GPR9 - Os dados relevantes do projeto são identificados e planejados quanto à forma de coleta, armazenamento e distribuição. Um mecanismo é estabelecido para acessá-los, incluindo, se pertinente, questões de privacidade e segurança. Os dados do projeto são as várias formas de documentação exigidas para sua execução.

GPR10 - Um plano geral para a execução do projeto é estabelecido com a integração de planos específicos. O objetivo deste resultado esperado é garantir que todos os planos que afetam o projeto estejam integrados e que a dependência entre estes planos tenha sido identificada e levada em consideração durante o planejamento, conciliando o trabalho a ser realizado aos recursos e condições existentes.

GPR11 - A viabilidade de atingir as metas do projeto é explicitamente avaliada considerando restrições e recursos disponíveis. Se necessário, ajustes são realizados. O estudo de viabilidade considera o escopo do projeto e examina aspectos técnicos (requisitos e recursos), financeiros (capacidade da organização) e humanos (disponibilidade de pessoas com a capacitação necessária).

GPR12 - O Plano do Projeto é revisado com todos os interessados e o compromisso com ele é obtido e mantido. Para obter compromissos dos interessados relevantes, é importante revisar o planejamento com eles e conciliar as diferenças existentes entre os recursos estimados e disponíveis.

GPR13 - O escopo, as tarefas, as estimativas, o orçamento e o cronograma do projeto são monitorados em relação ao planejado. A aderência aos diversos planos deve ser avaliada continuamente durante todo o ciclo de vida do projeto. Os resultados e os critérios de conclusão de cada tarefa são analisados, as entregas são avaliadas em relação às suas características (por meio de revisões e auditorias, por exemplo), a aderência ao cronograma e o dispêndio de esforços são examinados, bem como o uso dos recursos.

GPR14 - Os recursos materiais e humanos bem como os dados relevantes do projeto são monitorados em relação ao planejado. O objetivo deste resultado esperado é garantir que o projeto seja monitorado em relação aos itens planejados referentes a recursos materiais e recursos humanos.

GPR15 - Os riscos são monitorados em relação ao planejado. No decorrer do projeto novos riscos podem ser identificados para o projeto e os parâmetros dos riscos já identificados podem ser alterados. Além disso, pode ser necessário executar ações de mitigação para evitar

que os riscos aconteçam ou, no caso de riscos terem se concretizado, pode ser preciso executar ações de contingência para minimizar seus efeitos.

GPR16 - O envolvimento das partes interessadas no projeto é planejado, monitorado e mantido. Devem ser identificados os interessados relevantes no projeto, em que fases eles são importantes e como eles serão envolvidos (comunicações, revisões em marcos de projeto, comprometimentos, entre outros). Uma vez identificado e planejado o envolvimento, este deverá ser seguido, monitorado e mantido ao longo de todo o projeto.

GPR17 - Revisões são realizadas em marcos do projeto e conforme estabelecido no planejamento. Este resultado é importante porque as revisões em marcos são oportunidades para verificar, de forma ampla, o andamento de todo o projeto, independente do acompanhamento do dia-a-dia. Em projetos grandes essas revisões são fundamentais, questionando, inclusive, a viabilidade de continuidade do projeto.

GPR18 - Registros de problemas identificados e o resultado da análise de questões pertinentes, incluindo dependências críticas, são estabelecidos e tratados com as partes interessadas. Para completar o trabalho de monitoramento do projeto, os problemas precisam ser corrigidos e gerenciados até a sua resolução, com base em planos de ações, estabelecidos especificamente para resolver os problemas levantados e registrados. Caso não se consiga resolver os problemas neste nível, deve-se escalonar a resolução das ações a níveis superiores de gerência.

GPR19 - Ações para corrigir desvios em relação ao planejado e para prevenir a repetição dos problemas identificados são estabelecidas, implementadas e acompanhadas até a sua conclusão. Ações corretivas devem ser estabelecidas para resolver problemas que possam impedir o projeto de atingir seus objetivos se não forem resolvidos de forma adequada. As ações corretivas definidas devem ser gerenciadas até serem concluídas. O controle dos problemas levantados, as ações tomadas, os responsáveis pelas ações e os resultados devem ser registrados.

O MPS.BR (SOFTEX, 2011) define que o propósito do processo Gerência de Requisitos é gerenciar os requisitos do produto e dos componentes do produto do projeto e identificar inconsistências entre os requisitos, os planos do projeto e os produtos de trabalho do projeto.

O principal objetivo da Gerência de Requisitos é controlar a evolução dos requisitos. O processo Gerência de Requisitos (GRE) gerencia todos os requisitos recebidos ou gerados pelo projeto, incluindo requisitos funcionais e não-funcionais, bem como os requisitos impostos ao projeto pela organização.

Na implementação do MPS.BR espera-se que alguns resultados sejam atingidos, segundo o guia de implementação (SOFTEX, 2011) são eles:

GRE1 - O entendimento dos requisitos é obtido junto aos fornecedores de requisitos. O objetivo deste resultado é garantir que os requisitos estejam claramente definidos a partir do entendimento dos requisitos realizado junto aos fornecedores de requisitos.

GRE2 - Os requisitos são avaliados com base em critérios objetivos e um comprometimento da equipe técnica com estes requisitos é obtido. A avaliação e aprovação por parte do cliente após o entendimento dos requisitos por si só não é suficiente para que os requisitos sejam refinados e refletidos em modelos de análise e projeto para a codificação. A avaliação dos requisitos deve envolver, além do cliente, também, a equipe técnica da organização, podendo ser realizada de diversas formas.

GRE3 - A rastreabilidade bidirecional entre os requisitos e os produtos de trabalho é estabelecida e mantida. Este resultado indica a necessidade de se estabelecer um mecanismo que permita rastrear a dependência entre os requisitos e os produtos de trabalho.

GRE4 - Revisões em planos e produtos de trabalho do projeto são realizadas visando a identificar e corrigir inconsistências em relação aos requisitos. A consistência entre os requisitos e os produtos de trabalho do projeto deve ser avaliada e os problemas identificados devem ser corrigidos. Este resultado sugere, portanto, a realização de revisões ou de algum mecanismo equivalente para identificar inconsistências entre os requisitos e os demais elementos do projeto.

GRE5 - Mudanças nos requisitos são gerenciadas ao longo do projeto. Durante o projeto, os requisitos podem mudar por uma série de motivos. Desta forma, requisitos adicionais podem ser incorporados no projeto, requisitos podem ser retirados do projeto e/ou mudanças podem ser feitas nos requisitos já existentes. Ressalta-se que, devido às mudanças, os requisitos podem ter que ser revistos, conforme definido no GRE4.

4 PROCESSO DE *SOFTWARE* PROPOSTO

O processo de *software* proposto foi concebido para atender a realidade de empresas pequenas de desenvolvimento de software que buscam obter agilidade em seus processos.

O cenário ao qual este processo se aplica é de equipes com tamanho entre quatro e oito desenvolvedores, que segundo Schwaber (2011) é um tamanho ótimo para uma equipe de desenvolvimento, pois é suficientemente pequeno para se manter ágil, e grande o suficiente para completar uma parcela representativa do trabalho. Menos do que três membros em uma equipe de desenvolvimento diminui a interação e resulta em ganhos menores de produtividade. Havendo mais de nove integrantes é exigida muita coordenação. Equipes de desenvolvimento grandes geram muita complexidade para um processo empírico gerenciar. Os papéis de *Product Owner* e de *Scrum Master* não são incluídos nesta contagem, ao menos que eles também executem o trabalho do *Sprint Backlog*. Os membros desta equipe trabalham próximos e não possuem papéis definidos, podendo exercer o papel de desenvolvedor e *Scrum Master*.

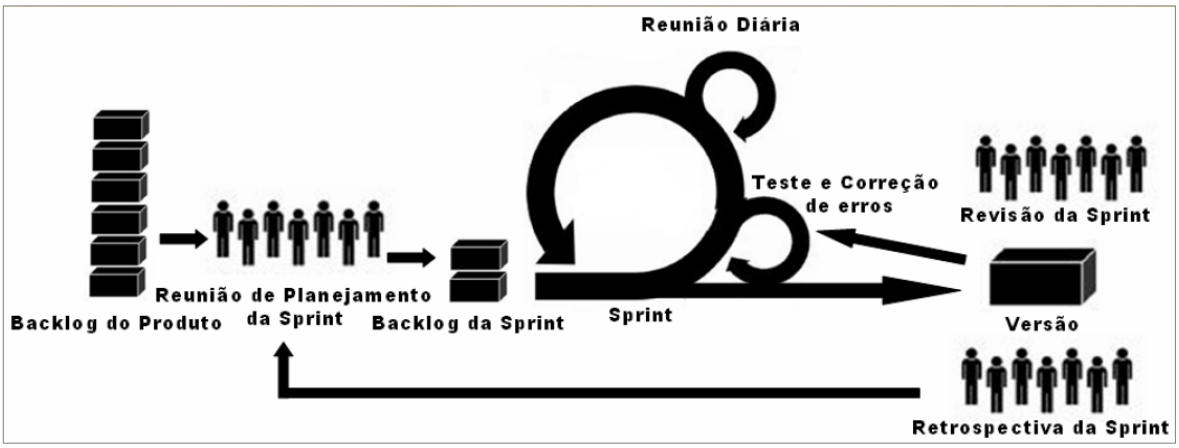
Com o objetivo de verificar o quanto o processo atende ao nível G do MPS.BR e quanto ele teria que sofrer adaptações, os resultados do processo são comparados aos resultados esperados pelo nível G do MPS.BR. A escolha pelo MPS.BR se deu pelo fato de ele ser aderente a organizações pequenas e atender aos padrões de qualidade internacionais.

Tendo em vista a necessidades que as organizações possuem, de reduzir custos, além da preocupação com a qualidade, esta proposta preocupa-se com a agilidade do processo. Com o objetivo de manter a agilidade, o processo proposto utiliza algumas técnicas que foram inspiradas pelo processo ágil Scrum.

A escolha pelo Scrum se deu pelo fato de ele trabalhar com equipes pequenas, ser flexível a mudanças, defender a geração apenas de documentação suficiente e ter uma boa aceitação mundial.

A Figura 1 apresenta o fluxo de atividades do processo proposto. Depois dos requisitos serem especificados através de estórias, o cliente os valida, em seguida é gerado o *product backlog* e a tabela de rastreamento. A equipe realiza uma reunião de planejamento da S*print*, na qual é gerado o *Sprint backlog*. Estes requisitos entram no ciclo de desenvolvimento, que tem duração de três semanas. Durante este ciclo são realizadas reuniões diárias. Ao final das três semanas uma nova versão do *software* é finalizada e por fim é feita a revisão e a retrospectiva da S*print*.

Figura 1 – Fluxo de Atividades do Processo



4.1 EQUIPE

A equipe deve ter entre quatro e oito desenvolvedores. Os desenvolvedores deverão ser capazes de exercer também o papel de Scrum Master.

A cada nova Sprint é desejável que os papeis dentro da equipe sejam redefinidos, tendo sempre um Scrum Master diferente. Desta forma todos os desenvolvedores da organização terão conhecimento para suprir uma possível perda ou ausência na equipe.

A equipe trabalha junta, com todos os desenvolvedores próximos, no mesmo ambiente. Esta equipe possui algumas características que Kniberg (2007) define da seguinte forma:

1. Audibilidade: Qualquer um da equipe pode conversar com o outro sem gritar ou sair da mesa.
2. Visibilidade: Todos da equipe podem ver todos os demais. Cada um da equipe consegue ver o quadro de tarefas. Não necessariamente perto o suficiente para lê-lo, mas pelo menos para vê-lo.
3. Isolamento: Se toda equipe de repente levantar e se envolver em uma espontânea e animada discussão sobre implementação, não haverá ninguém de fora da equipe perto o suficiente para ser perturbado. E vice-versa.

Dentro da equipe alguns papeis são definidos, cada um possui suas obrigações e responsabilidades.

1. *Product Owner*: O Dono do produto é a única pessoa responsável por gerenciar o Backlog do produto.

2. *Scrum Master*: O Scrum Master é um líder-facilitador para a equipe do Scrum e ajuda aqueles de fora da equipe do Scrum entender quais são as interações com a equipe do Scrum que são benéficas e quais não são.
3. Desenvolvedor: o ciclo de desenvolvimento de uma versão pode possuir um ou mais desenvolvedores, o *Scrum Master* também pode desempenhar o papel de desenvolvedor. Algumas de suas atividades são:
    - Transformar os requisitos em software funcionando.
    - Escolher quais estórias do *product backlog* entram no *sprint backlog*.

4.2 ATIVIDADES DO PROCESSO

O processo é dividido nas seguintes atividades:

1. Elicitação de requisitos.
2. Especificação.
3. Validação.
4. Elaboração do *Produto Backlog*.
5. Geração e gerenciamento do gráfico de *Burndown*.
6. Geração da tabela de rastreamento.
7. Reunião de planejamento da *Sprint*.
8. Reuniões diárias.
9. Teste e correção de erros.
10. Liberação da Versão
11. Revisão da *Sprint*.
12. Retrospectiva da *Sprint*.

Na elicitação de requisitos, os requisitos são coletados diretamente com o *Product Owner*, que representa todos os fornecedores de requisitos. Os fornecedores de requisitos podem ser clientes e usuários finais do sistema.

A técnica utilizada para a elicitação dos requisitos é a entrevista. Devem ser definidos um local e horário para a entrevista com os fornecedores de requisitos. O ponto de partida da entrevista é um conjunto de questões elaboradas previamente, a partir das respostas para estas questões podem surgir mais questões, que devem ser anotadas, assim como suas respostas.

Os fornecedores de requisitos também podem apresentar documentos ou funcionalidades de outros sistemas para descrever o que desejam do software. Deve ser documentado e anotado tudo que for pertinente.

Depois de elicitar os requisitos com os fornecedores de requisitos, eles devem ser especificados pelo *Product Owner*. Os requisitos são documentados através de estórias.

Cada estória é composta por uma série de atributos, alguns deles definidos conforme Kniberg (2007):

1. ID: Uma identificação única.
2. Nome: Deve ser curto e através do nome deve ser possível saber do que a estória trata.
3. Importância: a pontuação de importância dessa estória para o *product owner*. Em uma versão pode haver estórias de vários clientes. O *product owner* deve avaliar a importância da estória para o cliente e para o produto como um todo. Quanto maior o valor, mais importância. O valor da importância não é seqüencial em relação a estória com menos importância anterior, por exemplo, 1, 2, 3, ..., e sim é informado um valor com diferença de pelo menos 10 pontos do anterior, 10, 20, 30, ..., isso para o caso de surgir uma nova estória com uma importância que fique entre duas estória existentes.
4. Estimativa Inicial: As estimativas iniciais sobre quanto tempo é necessário para implantar aquela estória. A unidade é pontos por estória e corresponde a relação desenvolvedor/hora.
5. Como demonstrar: Uma descrição de alto nível de como a estória será demonstrada na apresentação da versão. Isso é uma especificação de teste.
6. Notas: quaisquer informações, esclarecimentos, referências e outras fontes de informação.
7. Solicitante: Qual cliente solicitou o item.
8. Dependências: O ID das estórias da qual esta tenha dependência, se este for o caso.
9. Versão: O ID da versão na qual a estória será implementada.
10. Componentes: O ID dos componentes de software que foram desenvolvidos ou alterados por causa da estória.
11. Situação: A estória pode possuir três situações: pendente, em andamento ou concluída.
12. Assinatura do cliente: o cliente deve validar a estória assinando ela.

Segundo Kniberg (2007), as estórias não devem ser muito pequenas (0,5 pontos estimados), para não ser vítima do micro-gerenciamento, nem muito grandes (40 pontos) para não terminarem parcialmente completas. É quase sempre possível quebrar uma estória em pedaços menores. O ideal é desenvolver estórias com 2 – 8 pontos.

O *product owner* define a importância das estórias e os desenvolvedores definem a estimativa inicial. As estórias podem ser feitas em qualquer ferramenta ou armazenadas em um sistema. Deve haver uma forma de controle do ID de cada estória. Ela deve ser impressa e validada pelo cliente.

Desde a concepção da estória até o início do seu desenvolvimento a sua situação é pendente. Quando é iniciado o desenvolvimento de uma estória, a sua situação é alterada para em andamento. A estória é considerada concluída quando o desenvolvimento foi concluído e já foram realizados os testes.

Se na fase de teste uma estória for reprovada ela volta para o desenvolvimento e a sua situação permanece em andamento até que ela seja aprovada no teste.

Depois de documentar os requisitos através de estórias, os requisitos devem ser validados pelo cliente. O cliente deve avaliar a estória e verificar se ela realmente atende o que ele necessita. No caso de a estória estar de acordo com o que o cliente deseja, ele deve assinar ela. Depois da assinatura do cliente a estória pode seguir no processo.

Caso a estória não descreva o que o cliente espera do software, as inconsistências devem ser levantadas e a estória deve ser refeita para que ela passe por uma nova validação.

A estória não entra para o *product backlog* sem que ela tenha sido validada pelo cliente.

Depois que todos os requisitos foram coletados, documentados através de estórias e aprovados pelos clientes o *product backlog* pode ser elaborado. Elaborar e gerenciar o *product backlog* são funções do *product owner*. É com base neste *backlog* que todo o desenvolvimento se segue, por isso ele tem uma importância muito grande dentro do processo.

Antes de começar o desenvolvimento das estórias a equipe deve ter as informações necessárias, para isso é feita a reunião de planejamento da *Sprint*.

Segundo Kniberg (2007), em uma reunião destas deve ser definido:

1. Um objetivo da *Sprint*.
2. Uma lista de membros da equipe.
3. O *Sprint backlog*.
4. Uma data definida para a apresentação da *Sprint*.
5. Data o local definidos para a reunião diária.

O *Scrum Master* deve começar a reunião definindo qual o objetivo para a *Sprint* e explicando para a equipe as estórias mais importantes. Em seguida os desenvolvedores estimam o tempo de cada estória, começando pela mais importante.

A estimativa de tempo de cada estória é feita utilizando a técnica do *Planning Poker*. Conforme Kniberg (2007), esta técnica consiste em distribuir um baralho de 13 cartas para cada membro da equipe. Quando uma estória deve ser estimada, cada membro da equipe escolhe uma carta que representa a sua estimativa de tempo e coloca-a virada para baixo sobre a mesa. Quanto todos tiverem feito a sua estimativa, as cartas são reveladas. Dessa forma, cada membro da equipe é forçado a pensar por si próprio ao invés de basear-se na estimativa de outra pessoa.

Se houver uma grande divergência entre duas estimativas, a equipe discute as diferenças e tenta chegar a uma visão comum do trabalho envolvido na estória.

Algumas dúvidas em relação a escopo de uma estória podem surgir durante a reunião. Estas dúvidas devem ser tiradas com o cliente solicitante da estória através de telefone, e-mail ou pessoalmente. Independente de a dúvida poder ser sanada ou não, a reunião segue normalmente.

Ao final desta reunião deve estar definido o *Sprint backlog*. Por último, todos os membros da equipe devem assinar uma ata, onde eles confirmam que participaram da reunião, confirmam que entenderam os requisitos apresentados e incluídos no *Sprint backlog* e também confirmam que estão comprometidos com o desenvolvimento destes requisitos.

Tendo as estórias com os tempos estimados é possível ter a soma dos esforços necessários para a conclusão da *Sprint* e do produto. Desta forma é possível acompanhar ao longo do desenvolvimento a quantia de esforços restantes através dos gráficos de *Burndown* da *Sprint* e do produto.

Para garantir a rastreabilidade entre requisitos e produtos de trabalho, são geradas as tabelas de rastreamento. Com estas tabelas é possível verificar a dependência entre requisitos, quais produtos de trabalho um requisito gerou e de quais requisitos um produto de trabalho

teve origem. Para documentar a dependência entre requisitos esta tabela deve possuir o identificador das duas estórias, desta forma é possível rastrear quais os requisitos dependentes e de quais requisitos se é dependente. Na tabela de rastreamento de produtos de trabalho deve existir a coluna com o identificador da estória e a coluna com o identificador do produto de trabalho, desta forma é possível rastrear as estórias que deram origem ao produto e a quais produtos a estória deu origem.

Cada novo incremento de software entregue é uma versão. As versões são o software original acrescido da implementação das estórias que a compuseram. As versões devem ser documentadas e devem possuir uma identificação única, assim como as estórias.

O ciclo de desenvolvimento de uma *Sprint* tem duração de 3 semanas, que, segundo Kniberg (2007), é um tempo curto o bastante para dar agilidade e longo o bastante para a equipe conseguir fluidez e se recuperar de eventuais problemas durante o desenvolvimento.

Toda *Sprint* precisa ser planejada, ter uma equipe alocada para o seu desenvolvimento e ter prazos e prioridades estabelecidas. O planejamento da *Sprint* tem início a partir do *product backlog*, que vai dar origem, depois da reunião de planejamento da *Sprint*, ao *Sprint backlog*.

O *Sprint backlog* é uma lista das estórias que vão compor a versão do produto. Os desenvolvedores são responsáveis por escolher quais estórias serão incluídas na *Sprint*.

As estórias são extraídas da lista do *product backlog* e, com base nas escolhas dos desenvolvedores, é formado o *Sprint backlog*. As estórias na lista de requisitos do produto devem estar ordenadas por importância e cada estória deve possuir a sua estimativa de tempo. As estórias incluídas no *Sprint backlog* são as de maior importância. A quantidade de estórias incluídas depende da estimativa de tempo das estórias, a soma dos tempos não pode ser maior e nem muito menor do que o tempo de duração que foi definido para a *Sprint*.

Um fator importante, que deve ser levado em consideração na hora de escolher quais estórias vão fazer parte do *Sprint backlog* é a dependência entre as estórias. Não pode ser inserida uma estória que seja dependente de outra e deixar a estória da qual ela depende de fora do *Sprint backlog*.

Cada estória possui a informação da versão. As estórias selecionadas para fazerem parte do *Sprint backlog* devem ter esta informação preenchida com o identificador da versão da qual farão parte.

Ao longo do processo são realizadas reuniões diárias, estas reuniões são realizadas de manhã, tem um tempo de duração de 15 minutos e os participantes ficam em pé. Nela cada membro da equipe diz o que fez no dia anterior e o que vai fazer no dia. Neste momento os

requisitos são revisados pela equipe. Pode ser encontrada alguma inconsistência na estória. Se houver qualquer alteração no escopo da estória, esta precisa ser validada novamente pelo cliente, ele continua no *Sprint backlog* para depois seguir o seu desenvolvimento.

No final da *Sprint* é gerada uma nova versão do software com todas as estórias do *Sprint backlog* que estiverem com a situação concluída, ou seja, já tiveram a sua implementação finalizada e já foram aprovadas pela fase de testes.

No fim do ciclo de desenvolvimento ocorre a retrospectiva da *Sprint*. Depois que a versão é liberada, é reservado um horário para esta reunião, nela o *Scrum Master* mostra o *Sprint backlog* e resume a versão. Cada pessoa tem o direito de dizer o que foi bom, o que poderia ter sido melhor e o que teria que ser diferente para a próxima *Sprint*.

Quando uma versão é lançada possivelmente ela contenha alguns erros. Um tipo de erro que pode ser encontrado em uma versão é erro de codificação. O programador pode ter codificado um cálculo utilizando uma fórmula errada, pode haver erro de sintaxe da linguagem de programação utilizada, etc

Outra situação que pode ocasionar problemas depois da versão finalizada é o mau entendimento do requisito. Em alguma etapa do processo, algum dos envolvidos pode interpretar o requisito de forma errada, seja na hora de escrever as estórias, ou o desenvolvedor, na hora de implementar. Esta estória pode ter sido validada pelo cliente, mesmo contendo erros. Normalmente este tipo de erro torna-se visível na demonstração para o cliente ou apenas depois que começa a ser utilizado.

Se algum erro for encontrado durante a fase de teste, ou seja, antes da liberação da versão, o requisito volta para o desenvolvimento para que seja corrigido o erro. Se este erro teve origem em alguma má definição da estória, esta deve ser corrigida e validada novamente com o cliente.

No caso de o erro ser encontrado depois da liberação da versão, a correção deste erro passa a ser uma nova estória, que vai conter o identificador da estória que ela estará corrigindo.

Provavelmente a equipe estará trabalhando em uma nova versão, mas a correção de um erro normalmente não pode ser inserida no *Sprint backlog* na qual a equipe trabalha no momento. O cliente, na maioria das vezes, não pode esperar até que uma próxima versão seja liberada para ter o erro corrigido.

Em uma situação destas, esta nova estória fará parte de uma versão de correção, que deverá ser liberada antes da versão na qual a equipe está trabalhando.

Quando uma nova versão é planejada a equipe deve considerar o fato de ter que corrigir algum erro da versão anterior e deixar algum tempo disponível para isso.

4.3 COMPARATIVO COM MPS.BR

A Tabela 1 mostra um comparativo com os resultados esperados pelo nível G do MPS.BR e quais o processo proposto atende e de que forma atende.

Tabela 1 – Comparativo entre MPS.BR e o Processo Proposto

Resultado Esperado (MPS.BR)	Resultado do Processo Proposto
Gerência de Projetos	
GPR1	As estórias e os *backlogs* evidenciam que o escopo foi definido.
GPR2	As tarefas são dimensionadas através da técnica de estimativas *planning poker*.
GPR3	O ciclo de vida do projeto é definido através da *Sprint.*
GPR4	Não atende.
GPR5	Não atende.
GPR6	Não atende.
GPR7	Não atende.
GPR8	Não atende.
GPR9	Os dados relevantes são identificados e armazenados em forma de estórias e estão disponíveis para o time *Scrum*.
GPR10	As estórias são documentadas e organizadas nos *backlogs* e a *Sprint* é planejada com data de início e fim.
GPR11	Não atende.
GPR12	As estórias são assinadas pelo cliente, se este está de acordo com elas, diariamente são realizadas reuniões, onde é possível revisar o projeto e a equipe de desenvolvimento assina uma ata de reuniões firmando o compromisso com o que foi estabelecido.
GPR13	O projeto é monitorado através das reuniões diárias, da situação das estórias que fazem parte da *Sprint* e do gráfico de *burndown*.
GPR14	Não atende.

GPR15	Não atende.
GPR16	O cliente precisa validar a estória e qualquer alteração que ocorra nela para que ela faça parte da *Sprint* e a equipe de desenvolvimento confirma o entendimento e o comprometimento com o projeto diariamente nas reuniões.
GPR17	O projeto é revisado diariamente nas reuniões.
GPR18	Quando um problema é identificado à estória é revista com o cliente, que deve validá-la novamente, e com a equipe.
GPR19	Não atende.
Gerência de Requisitos	
GRE1	No processo proposto os requisitos são documentados através de estórias, estas estórias passam pela avaliação do cliente, se forem aprovadas o cliente deve assinar a estória e só assim ela segue no processo.
GRE2	Na reunião de planejamento da *Sprint* as estórias que farão parte da *Sprint* são selecionadas pela equipe e esta assina uma ata afirmando o entendimento e comprometimento com os requisitos.
GRE3	Tabela de rastreamento.
GRE4	Diariamente são realizadas reuniões, onde as estórias e o plano de trabalho são revistos e discutidos pela equipe, no caso de erros e inconsistências os requisitos são alterados e passam pelo processo de validação novamente.
GRE5	Qualquer mudança que ocorra nos requisitos ao longo do projeto são gerenciadas pela equipe, o planejamento das atividades é a curto prazo e a elicitação e documentação dos requisitos ocorre ao longo do projeto de forma incremental, o que diminui as chances de mudanças e, se estas ocorrerem, serão facilmente absorvidas pela equipe, os requisitos e as prioridades podem ser facilmente alterados.

5 APLICAÇÃO DO PROCESSO PROPOSTO EM UM ESTUDO DE CASO

O processo proposto foi aplicado em um estudo de caso em uma empresa de desenvolvimento de sistema ERP. A empresa não possuía um processo de software definido, porém eram utilizados alguns recursos para gerenciamento e controle do desenvolvimento. A equipe de desenvolvimento da empresa é composta por quatro desenvolvedores. Além dos desenvolvedores o time da empresa é composto também pelo dono do negócio.

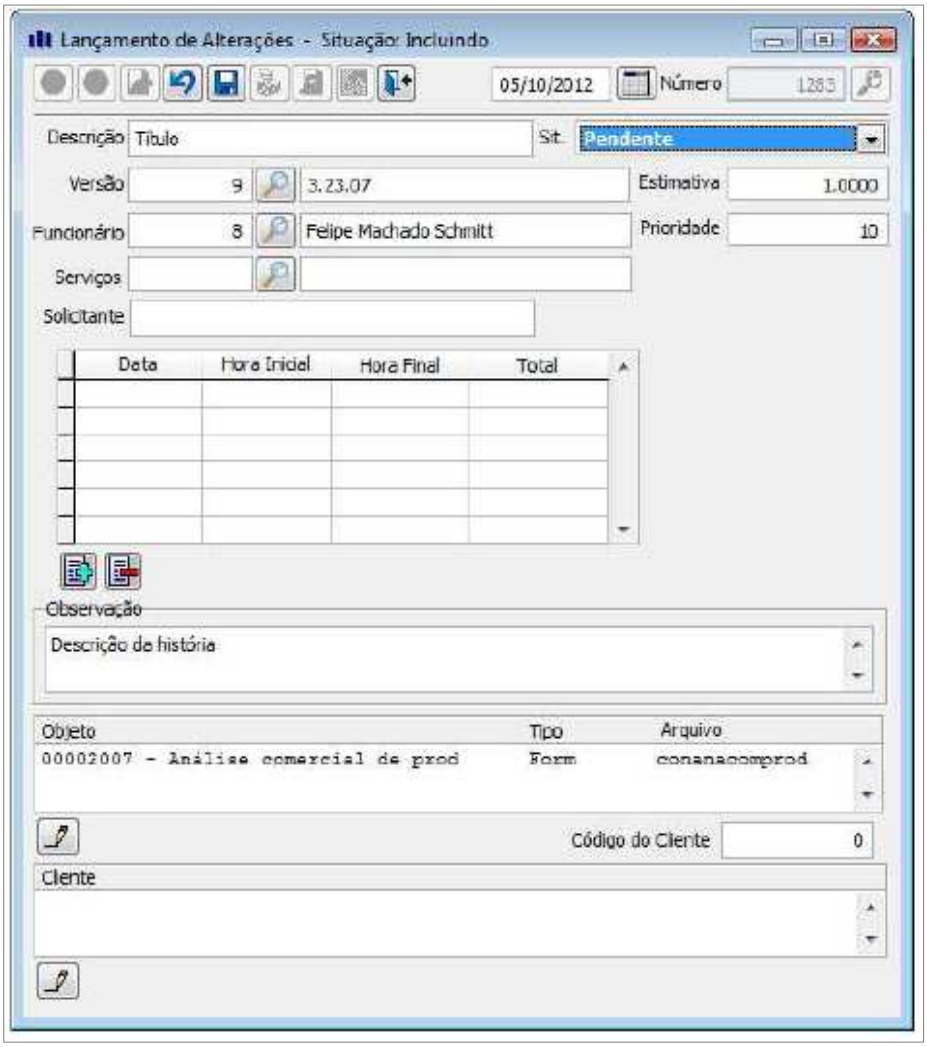
A etapa inicial da implantação foi apresentar o processo proposto para a equipe. Todos foram reunidos e uma visão geral do processo foi apresentada, além de uma breve descrição de cada papel, atividade e artefato gerado.

Os papeis foram definidos. O dono do negócio é o *product owner*, o *Scrum Master* ficou definido que seria um dos desenvolvedores, que continuou exercendo o papel de desenvolvedor.

O dono do negócio, sendo o dono do produto, fez o levantamento de todos os requisitos do sistema que ainda não haviam sido implantados. Estes requisitos foram especificados na forma de estórias e são originados de solicitações dos clientes da empresa, sugestões de clientes, usuários ou membros da equipe, obrigações legais, etc. Todos os requisitos foram priorizados pelo *product owner* utilizando o sistema de pontuação proposto, onde as estórias com valores maiores teriam mais prioridade.

A empresa já possuía uma ferramenta de desenvolvimento próprio para documentar os requisitos, boa parte das estórias já estavam descritas nesta ferramenta. Foram necessárias apenas algumas alterações na ferramenta para torná-la apta para documentar as estórias. A figura 2 mostra o programa onde as estórias são documentadas. Este programa armazena a estória com um ID e nele é possível informar o título, a situação, a estimativa, a prioridade, a versão de qual fará parte, o desenvolvedor, o serviço (alteração, manutenção, desenvolvimento, etc), a pessoa que solicitou, o cliente, os artefatos gerados (*forms*, *reports*, etc), as datas e horários de desenvolvimento e a descrição da estória. Todos os integrantes da equipe possuem acesso a esta ferramenta e podem visualizar e alterar qualquer estória.

Figura 2 – Programa de Documentação de Estórias.



O trabalho de levantamento, documentação e priorização das estórias, realizado pelo *product owner*, tornou possível a geração do *product backlog*. O sistema utilizado pela empresa já possuía uma ferramenta para a geração de relatório de requisitos. Foram feitas algumas modificações para esta ferramenta atender o Scrum e gerar os relatórios de *backlog*. A Figura 3 apresenta um *product backlog* gerado por este sistema.

Figura 3 – *Product Backlog*.

Product Backlog

Código	Descrição	Prioridade	Estimativa
Cliente 01			
1362	Conversor dos cursos técnicos	230	1,0000
1360	Rematricula com vendedor padão	220	1,0000
1351	Número de disciplinas por curso em Análise de Curso	184	2,0000
1318	E-mails na Relação de Inadimplências	100	0,5000
1266	Criação de tela e formulário para relatório de rematrícula.	20	5,0000
Cliente 02			
1425	Rateio de frete, seguro, redespacho e despesa na Ordem	316	10,0000
1414	NF-e de devolução de exportação	315	3,0000
1411	Pedido verifica se produto foi atendido por item ou	290	1,0000
1397	Mostrar notas de serviço na tela de Pedidos	246	2,0000
1385	Relatório para controle de produção de chão-de-fábrica	245	2,0000
1374	Leiaute dos relatórios da carteira de pedidos	240	1,0000
1359	Notas Fiscais de Serviço, não mostrar O.S. atendidas	190	2,0000
1345	Otimização da configuração de horários	181	1,0000
1329	Retenção de ST na nota fiscal de entrada	160	2,0000
1311	Especificações para as lentes	91	10,0000
1275	Horário de Trabalho no cadastro de Funcionário	48	1,0000
1258	Validação da ISO 9000 nas Ordens de Compra	10	2,0000
Cliente 03			
1438	Geração de Arquivos em pastas	340	2,0000
1433	Alterações no cadastro de imobilizado	330	10,0000
1342	Programa de estrutura permite exclusão não sequencial	175	0,5000
1333	Análise de Margem de Contribuição sobre Preço Padrão	170	1,0000
1321	Data de emissão para limite inferior de comissão	150	5,0000
1316	Cálculo de comissão da Análise da Margem de	92	1,0000
1282	Criação da tela de Importação de comissão específica	90	5,0000
1276	Ficha cadastal padrão	60	2,0000
1267	Data de emissão para limite inferior de comissão	45	5,0000

Neste *product backlog* as estórias estão separadas por clientes, acima de cada grupo de estórias aparece o nome do cliente dos quais as estórias tiveram origem. O *product backlog* é dividido em quatro colunas. A coluna código mostra o identificador da estória, a coluna descrição mostra o título da estória e por fim aparecem as colunas com os pontos de prioridades e o tempo estimado para cada estória.

A *Sprint*, planejada para ter uma duração de três semanas, teve início no dia 15/10/2012 e teve o seu fim planejado para o dia 01/11/2012.

Então, no dia 15/10/2012 foi dado início a primeira reunião de planejamento da *Sprint*. A reunião, que teve início na parte da manhã, foi começada com a apresentação de todas as estórias do *product backlog*, com ênfase e detalhamento maiores nas estórias mais importantes. As explicações acerca de cada estória não geraram dúvidas na equipe, não havendo questionamento durante a apresentação.

Uma das estórias apresentadas, a estória de código 1276, por exemplo, possui a seguinte descrição:

“A geração da ficha cadastral de funcionários e clientes deve ser padronizadas e possuir os seguintes botões, imprimir, visualizar, exportar, e-mail e sair.”

O próximo passo da reunião foi estimar cada estória. Utilizando a técnica do Planning Poker, cada participante da reunião recebeu um baralho desenvolvido manualmente pela própria equipe com números de 1 a 10 e ainda valores menores como 0,25 e 0,50. Os números do baralho equivalem a horas, sendo 0,25 equivalente a 15 minutos e 0,50 equivalente a 30 minutos. Foi combinado pela equipe que poderia ser utilizado na estimativa mais de uma carta, neste caso os valores seriam somados. Por exemplo, para uma atividade que um integrante da equipe acredite que são necessárias 2:30 horas de desenvolvimento, ele deve utilizar as cartas de valor 2 e valor 0,50 para fazer a estimativa.

O *Scrum Master* anuncia qual estória será estimada e cada membro da equipe escolhe as cartas com os valores desejados, quando todo mundo tiver chegado a uma conclusão sobre o tempo necessário para o desenvolvimento da estória e já tiver selecionado as cartas, estas são apresentadas a equipe. O *Scrum Master* verifica os valores apresentados, se os tempos estimados por todos forem iguais, está definida a estimativa daquela estória e este valor é atualizado no cadastro da estória no sistema. No caso de divergência entre os valores, a equipe discute o porque de cada um ter estimado a estória de forma diferente e chega-se a um consenso.

Na estimativa das estórias houve bastante divergência de valores. A maioria das estórias foram estimadas com valores diferentes pelos integrantes, principalmente as estórias maiores. Nestes casos, depois de alguma conversa, chegou-se a uma estimativa única para a estória.

As estórias que compõem o *Product Backlog* têm origem de diversas fontes, porém a mais comum é a solicitação de clientes. O produto no qual a equipe trabalha atende diversos clientes. Cada cliente possui um determinado número de horas mensais para o desenvolvimento de suas necessidades, que devem ser consideradas na hora de fazer o *Sprint Backlog*.

Para facilitar a elaboração do *Sprint Backlog* o sistema gera um relatório com as estórias pendentes separadas por cliente e ordenadas por prioridade, da mais importante até a menos importante para cada cliente.

A equipe de desenvolvedores escolhe quais estórias entram no *Sprint Backlog* tendo como base a duração da *Sprint*, que neste caso foi definido que seria uma *Sprint* de três semanas, o número de horas a que cada cliente tem direito e a prioridade de cada estória.

Com esta divisão por cliente, pode acontecer de uma estória com prioridade menor entrar na *Sprint* no lugar de uma estória com prioridade maior. No caso do cliente já ter excedido o número de horas de desenvolvimento, mas for necessário e inclusão de estórias adicionais na Sprint, a possibilidade de exceder o número de horas é negociada com o cliente.

As estórias que foram selecionadas para fazer parte da Sprint tiveram o seu cadastro atualizado no sistema, sendo possível visualizar o *Sprint Backlog* como na Figura 4.

Figura 4 – *Sprint Backlog*.

Sprint Backlog: 3.23.09

Código Descrição	Prioridade	Estimativa
Cliente 01		
1362 Conversor dos cursos técnicos	230	1,0000
1360 Rematricula com vendedor padão	220	1,0000
1351 Número de disciplinas por curso em Análise de Curso	184	2,0000
1318 E-mails na Relação de Inadimplências	100	0,5000
Cliente 02		
1425 Rateio de frete, seguro, redespacho e despesa na Ordem	316	10,0000
1414 NF-e de devolução de exportação	315	3,0000
1411 Pedido verifica se produto foi atendido por item ou	290	1,0000
1397 Mostrar notas de serviço na tela de Pedidos	246	2,0000
1385 Relatório para controle de produção de chão-de-fábrica	245	2,0000
1374 Leiaute dos relatórios da carteira de pedidos	240	1,0000
Cliente 03		
1438 Geração de Arquivos em pastas	340	2,0000
1433 Alterações no cadastro de imobilizado	330	10,0000
1342 Programa de estrutura permite exclusão não sequencial	175	0,5000
1333 Análise de Margem de Contribuição sobre Preço Padrão	170	1,0000
1321 Data de emissão para limite inferior de comissão	150	5,0000
1316 Cálculo de comissão da Análise da Margem de	92	1,0000

O *Sprint Backlog* possui o mesmo formato do *Product Backlog*, nele consta a versão que vai ser gerada ao final daquela *Sprint.* Os dois exemplos de *backlogs* apresentados demonstram apenas as atividades de três clientes, portanto não são *backlogs* completos. O nome dos clientes foram preservados.

Existe no sistema um cadastro de versões. Neste cadastro constam as informações de data de início do desenvolvimento da versão e a data em que a versão é liberada. Temos também uma versão chamada "*Product Backlog*", sempre que uma nova estória é cadastrada no sistema é informada esta versão. Quando é definido que uma determinada estória fará parte

da próxima versão, ou seja, fará parte da próxima *Sprint,* o cadastro da estória recebe a código da próxima versão a ser gerada.

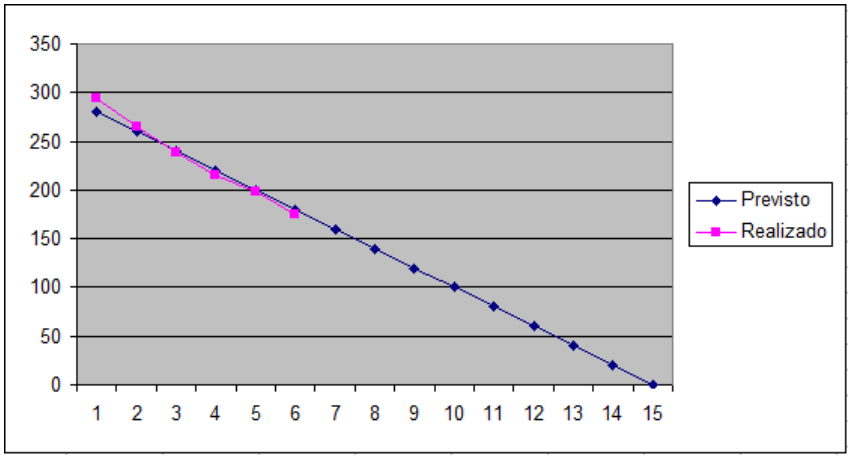
Estando pronto o primeiro Sprint Backlog a reunião de planejamento da Sprint foi encerrada. A reunião teve início de manhã e foi até metade do expediente da tarde.

Após a reunião algumas tarefas foram distribuídas e se deu início ao processo de desenvolvimento.

Um documento no *Excel*, conforme a Figura 5, foi criado para a elaboração dos gráficos de *Burndown*

.

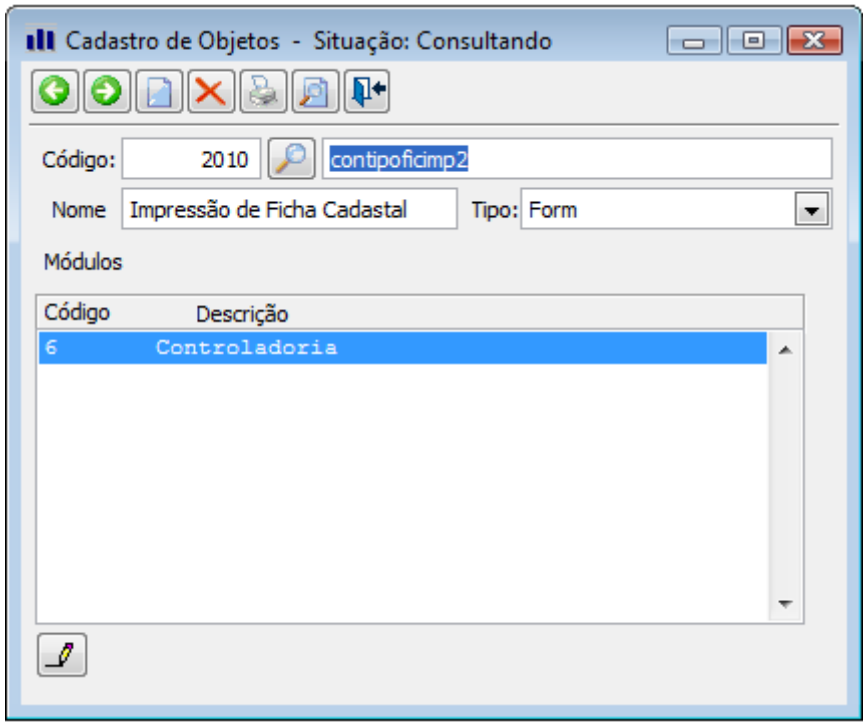
Figura 5 – Gráfico de *Burndown* da *Sprint.*



A tabela de rastreamento de produtos de trabalho é gerada pelo software que armazena as estórias, portando é possível localizar as estórias que deram origem a criação ou alteração em algum produto, assim como é possível localizar quais produtos foram criados ou alterados por alguma determinada estória.

Para esta tabela ser gerada é preciso informar na estória os identificadores dos produtos de trabalho na lista de objetos. Existe um cadastro destes objetos. A Figura 6 mostra um cadastro de produtos de trabalho.

Figura 6 – Cadastro de produtos de trabalho.

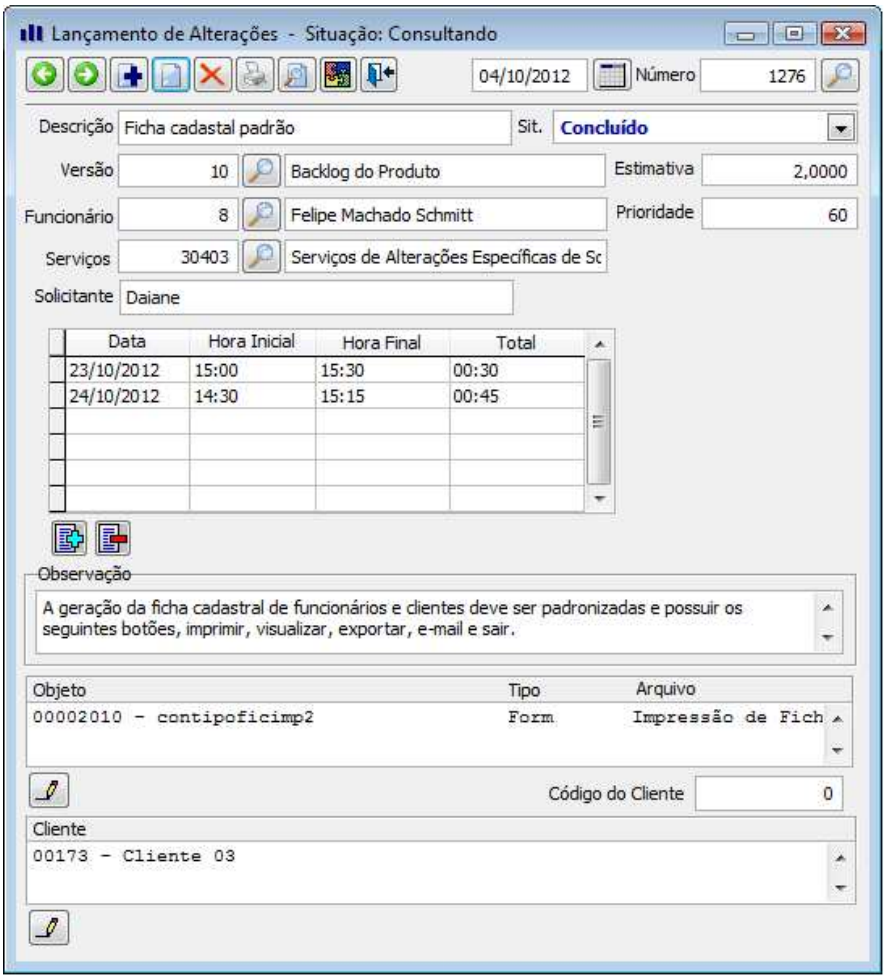


No dia seguinte à reunião de planejamento, foi feita a primeira reunião diária. Esta reunião foi bastante rápida, cada desenvolvedor falou o que já tinha iniciado no dia anterior e o que pretendia fazer durante o dia. Desta forma se seguiu durante toda a Sprint, diariamente a equipe se reunia de manhã, durante alguns minutos, para falar o que havia sido feito no dia anterior e o que pretendiam fazer naquele dia.

Quando uma estória é cadastrada no sistema ela fica com a situação “Pendente”. Ao iniciar o desenvolvimento de uma estória o desenvolvedor altera a sua situação para “Em Andamento” e, ao longo do seu desenvolvimento, ele atualiza as datas e horários que foram utilizados para desenvolver aquela estória. Quando finalmente o desenvolvimento da estória estiver concluído o desenvolvedor atualiza a situação dela para “Concluída”.

A Figura 7 mostra uma atividade concluída e com todas as informações preenchidas.

Figura 7 – Atividade concluída.



Ao longo do ciclo de desenvolvimento de uma estória o documento gerado no Excel para o controle dos gráficos de *Burndown* foi sendo atualizado.

A equipe de desenvolvimento tem a possibilidade de entrar em contato diretamente com o cliente no caso do surgimento de alguma dúvida em relação a qualquer estória. Ao longo da primeira *Sprint* surgiram algumas dúvidas, mas puderam ser sanadas com o *Product Owner*, o contato com o cliente não foi necessário em nenhuma situação.

Os desenvolvedores também possuem autonomia para modificar qualquer estória, se necessário, porém nesta *Sprint* também não houve a necessidade de qualquer tipo de alteração.

Coube aos desenvolvedores testar e corrigir o que foi feito, portanto os testes seguiram em paralelo com o desenvolvimento. As estórias só tiveram a sua situação atualizada para "Concluídas" depois de testadas e funcionando perfeitamente.

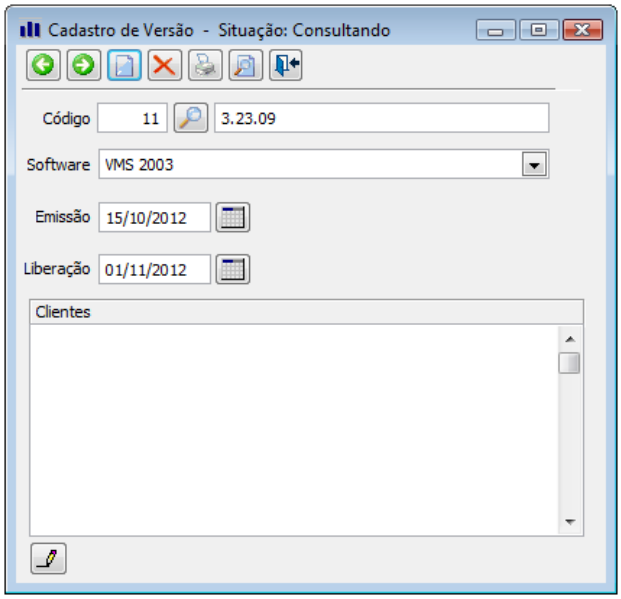
Ao final das três semanas, na data estipulada para o fim da *Sprint*, foi feita a reunião de revisão da *Sprint*. Nesta reunião a equipe de desenvolvimento apresentou às estórias que

estavam concluídas até aquela data, no caso desta *Sprint*, todas as estórias que faziam parte foram concluídas. Verificou-se então o que entraria na versão e, após a reunião, pode-se gerar uma nova versão do sistema.

A versão foi gerada pelo *Scrum Master* e consiste em gerar um software instalador da nova versão do sistema com todas as novas funcionalidades desenvolvidas.

A figura 8 mostra um cadastro de versões, onde é informado a data na qual foi iniciado o seu desenvolvimento e a data em que ela foi gerada.

Figura 8 – Cadastro de Versões.



Ao final do processo de geração da versão o Time *Scrum* se reuniu para a Retrospectiva da *Sprint*, onde todos puderam opinar sobre o processo, o que trouxe de melhorias, o que poderia ser feito para aperfeiçoar, etc.

5.1 CONSIDERAÇÕES FINAIS DO CAPÍTULO

A empresa possuía alguns mecanismos para documentação de requisitos e versões, porém não possuía um processo de *software* definido.

A busca pela implantação de um processo de *software* se deu pela necessidade de possuir um planejamento das atividades executadas. Para atender as cobranças e necessidades do mercado, é interessante para a empresa poder apresentar estimativas e prazos bem definidos, ter a capacidade de gerenciar o andamento do trabalho da equipe e ter uma visão geral do produto, tendo consciência de quais são as pendências e a prioridade de cada uma.

Ao iniciar a implantação do processo, boa parte dos requisitos já estavam documentados. A coleta de requisitos normalmente é feita pessoalmente, através de entrevistas, mas pode ser feita por e-mail, telefone, etc

A equipe preferiu não utilizar a validação dos requisitos pelo cliente. Como grande parte das estórias tem origem através de e-mails e ligações telefônicas, não é possível ter o cliente disponível para assinar o requisito. O cliente recebe a descrição da estória por e-mail e confirma se está de acordo por e-mail ou telefone.

Seguindo no processo, o gerenciamento do gráfico de *Burndown* foi mais trabalhoso do que o esperado. O gráfico é uma excelente ferramenta de gerenciamento, mas trabalhou-se com ele em uma ferramenta separada da que utilizamos para documentar as estórias e tempos trabalhados, portanto havia a necessidade que duplicar a informação. Devido ao grau de utilidade do gráfico, pretende-se estudar uma forma de gerar ele no sistema utilizado, com as informações já lançadas, assim também se evitam erros na hora de informar os dados no gráfico.

A tabela de rastreamento de requisitos e produtos de trabalho foi utilizada, porém a tabela de rastreamento entre requisitos não foi utilizada. O software utilizado não possui esta opção e optamos por não gerar este artefato nesta *Sprint*.

A atividade do processo que mais tem necessidade de ser aperfeiçoada é o Planning Poker. Foi bastante interessante utilizar esta técnica, porém ela precisa evoluir para que as estimativas se tornem mais exatas.

Uma das atividades que mais trouxe benefício para a equipe foi a reunião diária. Neste momento os integrantes da equipe demonstram o que fizeram e o que pretendem fazer, isso tornou a equipe mais engajada ao processo e, embora não tenha sido utilizada nenhuma métrica de velocidade da equipe, aparentemente esta se tornou mais produtiva. Nesta atividade a equipe também preferiu não utilizar as atas de reuniões.

As correções de erros de versões anteriores não foram documentadas. O dono do negócio optou por não documentar as versões de correção, pelo menos nesta primeira *Sprint*, mas provavelmente este processo será adotado no futuro.

Na retrospectiva da *Sprint*, a principal conclusão da equipe foi de que poderiam ter sido definidas mais estórias para compor o *Sprint Backlog*, isso ocorreu pelo fato de a equipe não estar apta para estimar as atividades, por isso foi bastante generosa na definição das estimativas.

Uma adaptação bastante interessante do processo foi a divisão dos *backlogs* por cliente, trabalhando com as solicitações de cada um dentro do que eles pré estabelecido.

6 CONCLUSÃO

Documentação de software é um desafio para os desenvolvedores. É um assunto que divide muitas opiniões e não possui uma resposta definitiva.

O principal ponto de partida para definir um processo a ser utilizado é o contexto no qual ele será inserido. A partir deste princípio surgem muitas variáveis para serem consideradas como tamanho da empresa, tamanho da equipe, capacitação dos membros da equipe, grau de complexidade do software desenvolvido, etc.

Mas independente do contexto no qual se está trabalhando, a qualidade nunca deve ter seu valor reduzido para a equipe, por isso, independente do grau de agilidade esperado pelo processo, algumas atividades se tornam necessárias, para garantir que os objetivos de todos os envolvidos sejam atingidos.

Este trabalho apresentou um processo de software para equipes pequenas, baseado no *Scrum*, buscando uma forma de gerenciar e documentar o trabalho realizado de maneira ágil. Depois da apresentação do processo este foi submetido a uma comparação com os resultados esperados pelo nível G do MPS.BR, como uma espécie de verificação do quanto o *Scrum* pode atender o modelo de qualidade e quanto seriam necessárias adaptações. Para isso foram realizados estudos acerca de processo de software, metodologia ágil, *Scrum*, qualidade de software e MPS.BR.

Foi realizado um estudo de caso aplicando-se o processo proposto em uma empresa de desenvolvimento de *software* ERP com o objetivo de avaliar o processo proposto.

O processo apresentado é um guia para equipes pequenas que não possuem um processo de software ou possuem e pretendem adotar técnicas do *Scrum*. Além das atividades do *Scrum* este processo define algumas atividades extras para reforçar o comprometimento da equipe e do cliente, que são cruciais para o bom funcionamento do *Scrum*.

Este processo melhorou o entendimento dos desenvolvedores em relação às necessidades dos clientes e ajuda toda a equipe a ter uma visão mais ampla do sistema. Com a reunião de planejamento da *Sprint*, toda a equipe toma conhecimento das novas funcionalidades que o sistema vai adquirir, isso não fica restrito exclusivamente ao desenvolvedor que implementa a funcionalidade. As reuniões diárias contribuem para reforçar esta situação.

Uma perda ou ausência na equipe passa a ser menos traumática, pois todos estarão aptos a trabalhar em qualquer parte do sistema.

O trabalho da equipe fica muito mais visível para o cliente com o planejamento das atividades com base nas estimativas e prioridades. O cliente tem plena consciência do que está sendo desenvolvido para ele, qual a quantidade de trabalho é requerida pra isso e em quais prazos ele terá as suas necessidades atendidas. Estes pontos melhoram muito o relacionamento com o cliente e a forma como os desenvolvedores trabalham.

O dono do produto pode ter informações sobre o que já foi e o que está sendo feito sempre que for necessário, além de poder acompanhar a evolução do que foi planejado. Através dos *backlogs* e dos gráficos de *burndown* é possível acompanhar o trabalho da equipe e a evolução do produto de forma fácil e rápida.

O número de retrabalho diminui na medida em que aumenta o entendimento da equipe em relação ao que precisa ser feito e na medida em que aumenta a visibilidade do cliente em relação ao que a equipe entendeu sobre a sua necessidade. Muitas inconsistências são verificadas ao escrever a estória. O cliente, ao ver a sua necessidade descrita, passa a ter uma visão muito mais clara do que ele realmente precisa. Sendo assim ele vai solicitar algo de forma mais completa.

Os objetivos principais da empresa com a implantação do processo de *software* foram atingidos. Esta se tornou capaz de ter uma visão geral do produto e poder apresentar isso para os seus clientes. Ter consciência das pendências a serem desenvolvidas e da importância de cada uma, torna cada versão do sistema mais relevante para o cliente.

O estudo de caso realizado demonstrou que o processo proposto pode ser aplicado e traz bons resultados. Todos os envolvidos obtêm vantagens, o dono do produto por poder gerenciar melhor, a equipe de desenvolvimento por trabalhar em um ambiente organizado e transparente e o usuário final por poder utilizar um produto de qualidade maior.

Algumas atividades do processo tendem a ter um desempenho melhor em longo prazo, como é o caso das estimativas através do *planning poker*. Esta técnica requer um nível de conhecimento bom por parte da equipe e certa experiência na atividade de estimar tempos. Estas estimativas iniciais tornam-se ainda mais imprecisas quando se trabalha com atividades mais complexas e quando o planejamento é feito para um período maior, como é o caso da *Sprint* de três semanas.

Analisando o processo, em comparação ao MPS.BR, pode-se verificar que é possível realizar algumas adaptações e melhorias para atender os itens não contemplados. Fazer estas adaptações traria mais qualidade ao processo e tornaria a empresa apta a buscar uma certificação.

7 REFERÊNCIAS

AGILE MANIFESTO. **Manifesto for Agile Software Development**. 2001. Disponível em <http://agilemanifesto.org>. Acesso em: 16 de junho, 2012.

AMBLER, Scott W.,. **Modelagem ágil**: práticas eficazes para a programação extrema e o processo unificado. Porto Alegre: Bookman, 2004. 351 p.

ASSOCIAÇÃO PARA PROMOÇÃO DA EXCELÊNCIA DO SOFTWARE BRASILEIRO SOFTEX. **MPS.BR - Melhoria de Processo do Software Brasileiro**: Guia de Implementação – Parte 1: Fundamentação para Implementação do Nível G do MR-MPS, 2011. Disponível em <http://www.softex.br>. Acesso em: 15 de março, 2012.

KNIBERG, Henrik. **Scrum e XP direto das Trincheiras**: Como nós fazemos Scrum. São Paulo: C4Media, 2007.

PRESSMAN, Roger S.. **Engenharia de software**. 6.ed. São Paulo: McGraw-Hill, 2006. 720 p.

ROCHA, Ana Regina Cavalcanti da; MALDONADO, José Carlos; WEBER, Kival Chaves. **Qualidade de software**: teoria e prática. São Paulo: Prentice Hall, 2001. xvi, 303 p.

SCHWABER, Ken; SUTHERLAND, Jeff. **O Guia do Scrum**: O Guia definitivo para o Scrum. Scrum.org, 2011. 18 p.

SCHWABER, Ken; SUTHERLAND, Jeff. **Scrum**. Scrum.org, 2010. 23 p.

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

VERSIONONE INC. **State of Agile Development Survey**. Atlanta: VersionOne, Inc., 2010.