UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

FELIPE AUGUSTO PASTORE DE LIMA

GERENCIAMENTO DE PROJETOS DE *SOFTWARE* COM *SCRUM*

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2011

FELIPE AUGUSTO PASTORE DE LIMA

GERENCIAMENTO DE PROJETOS DE *SOFTWARE* COM *SCRUM*

Trabalho de Diplomação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – CSTADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof Fernando Schutz.

MEDIANEIRA
2011

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

GERENCIAMENTO DE PROJETOS DE *SOFTWARE* COM *SCRUM*

Por

Felipe Augusto Pastore de Lima

Este Trabalho de Diplomação (TD) foi apresentado às 10:30 h do dia 15 de junho de 2011 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. O candidato foi argüido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. Fernando Schutz
UTFPR – *Campus* Medianeira
(Orientador)

Prof. Juliano Rodrigo Lamb
UTFPR – *Campus* Medianeira
(Convidado)

Prof[a]. Alessandra Hoffmann
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Juliano Rodrigo Lamb
UTFPR – *Campus* Medianeira
(Responsável pelas atividades de TCC)

A folha de aprovação assinada encontra-se na Coordenação do Curso.

"O negócio dos negócios são as pessoas, ontem, hoje e sempre" Herb Kelleher

RESUMO

LIMA, P. Felipe Augusto. Gerenciamento de projetos de *software* com *Scrum*. 2011. Trabalho de conclusão de curso (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Medianeira 2011.

O *Scrum* é um processo baseado na filosofia ágil, o qual busca a auto-organização da equipe, maior colaboração com o cliente, desenvolver o trabalho em equipe e aumentar a transparência sobre o que acontece durante o desenvolvimento. Estas características fazem com que a equipe alcance maior desempenho e produtividade. Este trabalho tem como objetivo demonstrar através de revisão literária e de um estudo experimental o que é e quais são os pensamentos da filosofia ágil, além de explicar o funcionamento do *Scrum*, um dos processos ágeis de maior aceitação no mercado. A implantação dos processos ágeis não é algo simples e exige uma adaptação à forma de trabalhar por todas as partes envolvidas, porém o esforço da equipe é recompensado quando os projetos são entregues dentro do prazo e atendendo as necessidades do cliente.

Palavras-chave: Filosofia ágil, Agilidade, Produtividade.

ABSTRACT

LIMA, P. Felipe Augusto. Gerenciamento de projetos de *software* com *Scrum*. 2011. Trabalho de conclusão de curso (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Medianeira 2011.

Scrum is an process based on the agile philosophy, which aims for self-organization teams, greater collaboration with costumers, teamwork development and increase transparency about what happens during development. These aspects make the team reach increased performance and productivity. This work aims on demonstrating, on a theoretical and experimental study way, what is and what are the thoughts of the agile philosophy, and explain the workings of Scrum, one of agile processes which have the widest market acceptance. The deployment of agile processes is not simple and requires an adaptation on the work way of all parties involved, but the team's effort is rewarded when projects are delivered on time and meeting customer needs.

Keywords: Agile Philosophy, Agility, Productivity.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Processos ágeis nasceram como uma alternativa aos pesados processos de desenvolvimento de *software* orientados a documentação. O *Scrum* é um processo ágil que, tendo como base a filosofia ágil e utilizando práticas iterativas e incrementais, busca entregar *software* de valor agregado ao cliente.

Os principais objetivos das atividades propostas no *Scrum* são os de trazer transparência, motivação, aperfeiçoamento contínuo, auto-organização da equipe e reduzir o risco de o projeto falhar. A busca por transparência se dá pois os *feedbacks* das ações da equipe devem ser dados o quanto antes, evitando assim que os problemas sejam encobertos na fase de início, onde é relativamente mais fácil de resolver, e descobertos em uma fase mais adiantada do projeto, onde as vezes estes tenham multiplicado seu tamanho. A busca por motivação existe porque indivíduos felizes e motivados são muito mais produtivos, principalmente em uma área que precisa de muita criatividade como é o caso da área de desenvolvimento de *software.* A idéia de aperfeiçoamento contínuo é a de que nada não está tão bom que não possa melhorar, o processo sempre pode ser melhorado e adaptado as necessidades da equipe e do projeto. O foco do *Scrum* é no produto e não nas funções dos integrantes, por isso da característica de auto-organização onde todos, independentes de sua função, trabalham além de suas especialidades para buscar entregar um produto de valor agregado para o cliente. O resultado de todas as práticas juntas adotadas pelo *Scrum* é o de redução no risco de o projeto falhar.

1.1 OBJETIVO GERAL

Fazer uma revisão bibliográfica sobre a filosofia ágil, *Scrum* e aplicá-lo em um projeto de desenvolvimento de *software,* como estudo experimental.

1.2 OBJETIVOS ESPECÍFICOS

- Desenvolver um referencial teórico sobre a filosofia ágil.
- Criar um referencial teórico sobre *Scrum* e suas atividades.

- Utilizar como estudo experimental o acompanhamento das etapas do gerenciamento de um projeto de *software* utilizando *Scrum*.

1.3 JUSTIFICATIVA

Os processos ágeis buscam minimizar o risco de o projeto falhar, trazendo boas práticas ao desenvolvimento, enfatizando o que realmente importa, deixando de lado coisas que não serão utilizadas ou que até prejudicariam o projeto.

Equipes ágeis estão vendo ganhos significativos de produtividade com diminuição correspondente no custo. Elas são capazes de levar produtos ao mercado muito mais rápido e com maior grau de satisfação do cliente. Elas estão experimentando uma maior visibilidade no processo de desenvolvimento, levando a uma maior previsibilidade. E para elas, projetos fora de controle estão se tornando uma coisa do passado (COHN, 2010).

Os processos ágeis preparam o projeto para aceitar a mudança, porque caso os requisitos mudem ao longo do projeto, se o projeto não aceitar mudanças, o produto entregue será algo que não agrega muito valor ao usuário.

Em um mercado cada vez mais globalizado e competitivo, é imprescindível que as empresas busquem se adaptar a esta realidade, buscando sempre o desenvolvimento, excelência técnica e maior atenção às necessidades do cliente. O processo *Scrum* e suas atividades prometem melhorar estes aspectos.

1.4 ESTRUTURA DO TRABALHO

O primeiro capítulo, que trata da revisão de literatura, busca introduzir a filosófica ágil, sua história, princípios e valores, explicar o conceito de projeto e demonstrar o processo *Scrum* e suas atividades.

O segundo capítulo é composto pelo estudo experimental, onde é desenvolvido um *software* utilizando o processo *Scrum*.

2 REVISÃO DE LITERATURA

Nas próximas subseções serão descritas a história da agilidade, os valores e princípios da filosofia ágil e o processo *Scrum* e suas atividades.

2.1 HISTÓRIA DA AGILIDADE

Em fevereiro de 2001, em uma estação de ski chamada *The Lodge at Snowbird* nas montanhas Wasatch de Utah, 17 pessoas se encontraram para passar um tempo de lazer, conversar e trocar idéias. O que emergiu foi o Manifesto para Desenvolvimento Ágil de *Software*. Representantes da *Extremme Programming*, *SCRUM*, DSDM, *Adaptive Software Development*, *Crystal*, *Feature-Driven Development*, *Pragmatic Programming*, e outros simpatizaram com a necessidade de uma alternativa aos pesados processos de desenvolvimento de *software* orientados a documentação (MANIFESTO ÁGIL, 2011).

2.2 FILOSOFIA ÁGIL

Algumas equipes falham ao adotar o *Scrum* ou qualquer outro método ágil, pois dão maior importância em aprender os processos envolvidos do que entender a filosofia ágil.

> “A idéia básica da filosofia ágil é de resolver problemas sem criar outros. É identificar o que precisa ser mudado, mantido e adaptado. É estar sensível à mudanças sempre que necessário e saber também quando as mudanças não são necessárias. É uma maneira viva, dinâmica e inteligente de se fazer *software*, e não somente isso, mas também uma filosofia que orienta e sustenta as ações e reações, tanto de um indivíduo quanto de uma equipe ou até mesmo uma organização inteira.” (UNIVERSO ÁGIL, 2011)

São quatro valores e doze princípios que o Manifesto Ágil (2011) segue. Serão estes descritos nas próximas subseções.

2.2.1 Primeiro valor

“Indivíduos e interações mais do que processos e ferramentas”

Quadro 1 - Primeiro valor
Fonte: Manifesto Ágil (2011)

Quem gera o produto, afinal, são as pessoas quais possuem características, conhecimentos e culturas diferentes. São as pessoas que utilizam os processos e as ferramentas para obter resultado, não o contrário.

Ter uma boa interação entre os membros é a peça chave para que equipe caminhe para a direção correta.

2.2.1.1 Primeiro princípio

“O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através da conversa face a face”

Quadro 2 - Primeiro princípio
Fonte: Manifesto Ágil (2011)

A comunicação face a face diminui as chances de uma interpretação errada tanto quanto ao que o cliente está querendo e também ao que os integrantes da equipe estão discutindo. Em uma comunicação face a face pode-se observar além da comunicação verbal, a comunicação não verbal, qual confirma se o que a outra pessoa está dizendo é realmente o que ela quer. Por exemplo, no caso do cliente estar dizendo “Está bom”, mas mostrando em sua aparência que não era bem isso que ele queria, o time poderá tomar atitudes que, se este “Está bom” tivesse sido passado a equipe por meio de mensagem ou até mesmo de um contrato formal, esta não iria saber.

2.2.1.2 Segundo princípio

"As melhores arquiteturas, requisitos e design emergem de equipes auto-organizáveis"

Quadro 3 - Segundo princípio
Fonte: Manifesto Ágil (2011)

O pensamento das equipes auto-organizáveis é o de que as equipe tem que ser uma só unidade, que todos devem ter o mesmo foco e objetivo, e fazer coisas além de sua área de especialidade se for preciso para alcançar tal objetivo.

Equipes auto-organizáveis operam de uma maneira empreendedora. Deixam os cargos e posições dentro da empresa de lado para buscar resolver o problema, e a urgência para resolver este problema faz com que a equipe se auto-organize e alcance um estado de maior eficiência (HOWARDS e ROGERS, 2011).

Equipes auto-organizáveis dão espaço para as pessoas que estão motivadas fazer mais do que seus cargos tradicionais oferecem a elas.

Marcus Buckingham (2001) discute uma pesquisa feita pela empresa Gallup que descobriu que apenas uma pequena porcentagem das pessoas entrevistadas acreditavam que seus empregos lhe permitissem trabalhar em tarefas que os fizessem utilizar todo o seu potencial.

2.2.1.3 Terceiro princípio

"Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho"

Quadro 4 - Terceiro princípio
Fonte: Manifesto Ágil (2011)

Segundo John Dewey (apud CARNIAGE, 1995, p.59), "a mais profunda das solicitações na natureza humana é o desejo de ser importante". Então, a melhor maneira de fazer com que as pessoas do seu ambiente de trabalho fiquem motivadas é fazendo com que elas se sintam importantes. A responsabilidade e a

confiança nos integrantes da equipe para tomar decisões e resolver os problemas que os processos ágeis proporcionam fazem com que estes se sintam importantes e motivados a realizar o trabalho.

Não se pode obrigar as pessoas a fazer o que tem que ser feito, apenas lhes oferecer o ambiente necessário para que estas, naturalmente, desempenhem suas funções.

A equipe precisa de um ambiente onde haja confiança e atenção às suas necessidades para que estes busquem se desenvolver se tornando maior que as adversidades que encontrarem.

2.2.1.4 Quarto princípio

"Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente"

Quadro 5 - Quarto princípio
Fonte: Manifesto Ágil (2011)

Desenvolvimento sustentável significa ter uma constante produção de *software* que dure um longo período de tempo.

Equipes ágeis buscam evitar que ocorra uma situação em que no começo do projeto todos estão tranqüilos e relaxados, e quando a data limite para a entrega do projeto está próxima a equipe tenha que fazer horas extras e "correr atrás do prejuízo" para conseguir entregar o que foi proposto a tempo. O ideal é que a quantidade de tempo e esforço durante todo o decorrer do processo seja a mesma.

As principais estratégias para buscar ao máximo para que esta regra seja respeitada são, a entrega constante de *software* funcionando em curtos espaços de tempo e a busca por testar o que foi feito e corrigir os erros o mais cedo possível.

2.2.2 Segundo valor

“Software em funcionamento mais que documentação abrangente”

Quadro 6 - Segundo valor
Fonte: Manifesto Ágil (2011)

Clientes se interessam por resultados, querem ver resultado, e o resultado é *software* funcionando.

A documentação só é útil quando ajuda a realizar o trabalho. Ela não deve ser prioridade, e sim *software* em funcionamento.

2.2.2.1 Quinto princípio

“Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado”

Quadro 7 - Quinto princípio
Fonte: Manifesto Ágil (2011)

Software com valor agregado pode ser definido em agilidade como o *software* que atende as necessidades do negócio em uma determinada área.

Entrega contínua significa que, a cada porção de *software* que tenha um valor de mercado, este já deve ser entregue ao cliente.

Quando os clientes percebem que podem ter produtos de valor a cada *Sprint*, eles normalmente decidem que não há porque esperar para ter todas as funcionalidades de uma vez só (COHN, 2010).

Ter em mãos porções de *software* que tem um valor de mercado faz com que os clientes fiquem mais satisfeitos e seguros ao decorrer do processo, pois têm a certeza de que não serão surpreendidos no final do projeto com algo que talvez não agregue nenhum valor ao seu negócio.

2.2.2.2 Sexto princípio

"Simplicidade - a arte de maximizar a quantidade de trabalho não realizado - é essencial"

Quadro 8 - Sexto princípio
Fonte: Manifesto Ágil (2011)

Sistemas mais simples são mais fáceis de implementar, dar manutenção, mais fáceis de entender tanto pela equipe quanto pelo usuário.

Muitas das funcionalidades que implementam–se nos sistemas trazem pouco, se não, nenhum valor de negócio para o cliente, e são estas as funcionalidades que devem ser descobertas e evitadas pois gastam recursos desnecessários.

A simplicidade se aplica também ao processo de desenvolvimento. Ao invés de utilizar ferramentas complexas de gerenciamento, como softwares para controle de projeto, que talvez demande tempo até a equipe se adaptar, utilizar ferramentas simples como *post-its* em um mural, que deixam os objetivos mais visíveis e mais fáceis de gerenciar. Ao invés de utilizar ferramentas de desenho e modelagem complexas, utilizar um quadro branco que é de fácil acesso a todos.

2.2.2.3 Sétimo princípio

"Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência a menor escala de tempo"

Quadro 9 - Sétimo princípio
Fonte: Manifesto Ágil (2011)

Objetivos mais curtos fazem com que os integrantes da equipe acreditem com mais facilidade que eles podem ser alcançados. Como foi citada anteriormente, a entrega em um curto espaço de tempo ajuda também a manter um ritmo constante.

As entregas freqüentes mantém os clientes mais envolvidos durante o processo, fazendo-os assim poder dar um *feedback* adiantado, e com o *feedback*

adiantado, os erros, que seriam descobertos e talvez corrigidos mais à frente, podem ser corrigidos num tempo menor custando menos tempo e esforço.

2.2.3 Terceiro valor

"Software funcionando é a medida primária de progresso"

Quadro 10 - Terceiro valor
Fonte: Manifesto Ágil (2011)

Em agilidade nenhuma funcionalidade é considerada completa até que esta esteja devidamente testada e aprovada pelo usuário final.

Por mais que o código esteja limpo, organizado e fazendo muitas coisas, se o que ele faz não satisfaz as necessidades do usuário final, se ele não traz valor ao negócio do cliente, seu desenvolvimento então não é considerado um progresso.

2.2.3.1 Oitavo princípio

"Contínua atenção a excelência técnica e bom design aumentam a agilidade"

Quadro 11 - Oitavo princípio
Fonte: Manifesto Ágil (2011)

Processos ágeis não funcionam se não forem sustentados por uma série de práticas técnicas (RASMUSSON, 2010).

Bom design em agilidade significa um design eficiente, mas simples, que todos os integrantes da equipe entendam.

2.2.4 Quarto valor

"Colaboração com o cliente mais que negociação de contratos"

Quadro 12 - Quarto valor
Fonte: Manifesto Ágil (2011)

Deve-se lembrar que clientes e desenvolvedores estão do mesmo lado, ambos buscam que o *software* desenvolvido agregue valor ao cliente.

Muitas equipes de desenvolvedores buscam se proteger através de contratos para caso o produto desenvolvido não saia como o ideal para o negócio do cliente, sendo que a solução mais inteligente seria a de buscar garantir que o produto que está sendo desenvolvido atenda as necessidades do cliente.

Em uma situação em que haja uma briga por algo que está no contrato afeta negativamente as duas partes, pois o cliente não consegue ter o que precisa e a empresa pode acabar perdendo futuros trabalhos com o cliente.

2.2.4.1 Nono princípio

“Pessoas de negocio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto”

Quadro 13 - Nono princípio
Fonte: Manifesto Ágil (2011)

Os clientes, as pessoas que entendem do negócio para qual o sistema será desenvolvido, devem acompanhar os desenvolvedores para garantir que estão caminhando na direção certa.

As pessoas de negócio têm o domínio do que precisa ser feito e os desenvolvedores tem o domínio de como pode ser feito, é essencial que ambos cooperem entre si para desenvolver o melhor produto possível.

2.2.4.2 Décimo princípio

“Responder a mudanças mais que seguir um plano”

Quadro 14 - Décimo princípio
Fonte: Manifesto Ágil (2011)

Um plano só é útil quando ele leva ao lugar certo. Em projetos de *software* existem muitas incertezas, o cliente às vezes não sabe o que quer até ver o produto funcionando, por isso a equipe deve estar preparada para eventuais mudanças no plano ao decorrer do processo.

2.2.4.3 Décimo primeiro princípio

"Mudanças nos requisitos são bem vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente"

Quadro 15 - Décimo primeiro princípio
Fonte: Manifesto Ágil (2011)

Ter que tomar uma decisão no início do projeto, sem poder voltar atrás, sobre como o *software* deve se comportar é um risco que o cliente não quer e não precisa correr. Os processos ágeis tiram proveito disso oferecendo mais segurança, porque o cliente pode mudar os requisitos caso algo mude em seu negócio.

Projetos de *software* que seguem um processo ágil devem ter uma arquitetura preparada para aceitar mudanças de uma forma amigável.

2.2.4.4 Décimo segundo princípio

"Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo"

Quadro 16 - Décimo segundo princípio
Fonte: Manifesto Ágil (2011)

Processos ágeis não são receitas de bolo, não dizem exatamente como se deve prosseguir em todas as situações. A equipe deve, freqüentemente, parar para analisar o que está acontecendo e buscar melhorias, tanto no processo, quanto nas diversas áreas que envolvem o trabalho da equipe.

2.3 PROJETO

Projeto é algo temporário, ou seja, que tem início e fim e visa criar um produto, serviço ou resultado que seja único. O fim do projeto se dá quando os objetivos foram alcançados, quando não há mais necessidade do projeto existir ou quando os objetivos não serão ou não podem ser atingidos. Quando se diz temporário, não necessariamente o projeto precisa ser de curta duração. A definição de temporário não se refere ao produto resultado do projeto, pois este pode ter duração indeterminada (PROJECT MANAGEMENT INSTITUTE, 2008, p. 4).

2.4 SCRUM

O *Scrum* é um processo ágil de desenvolvimento de *software* que trabalha via iterações chamadas de *Sprints*. Este processo é ideal para projetos que mudam seus requisitos com grande freqüência, como softwares dinâmicos para a internet.

Scrum é baseado nas melhores práticas aceitas pelo mercado, utilizadas e provadas por décadas. Ele é definido então em uma teoria de processos empíricos (SCHWABER e SUTHERLAND, 2010).

A Figura 1 mostra graficamente os processos do Scrum.

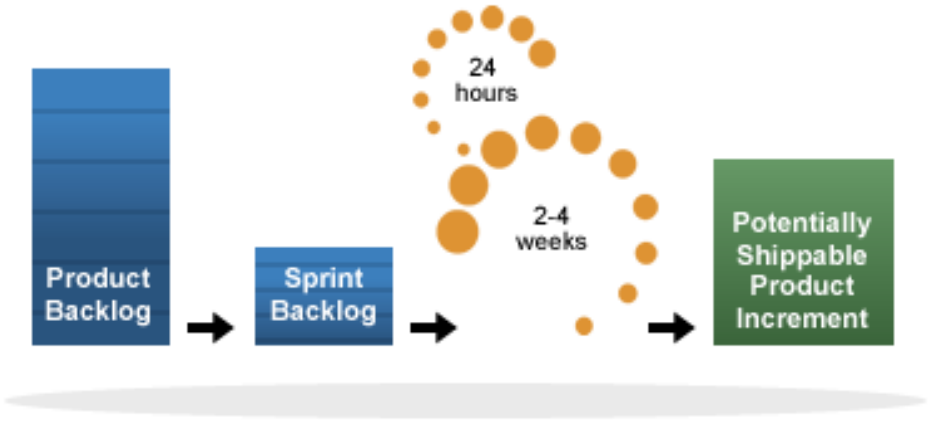

Figura 1 - Etapas do *Scrum*

Fonte: *Scrum* Alliance (2010)

2.4.1 Papéis

Para se explicar os papéis do *Scrum*, Schwaber e Sutherland (2010) utilizam a história do porco e da galinha.

> Uma galinha e um porco estão juntos quando a galinha diz: “Vamos abrir um restaurante!” O porco reflete e então diz: “Como seria o nome desse restaurante?” A galinha diz: “Presunto com Ovos!” O porco diz: “Não, obrigado, eu estaria comprometido, mas você estaria apenas envolvida!” (SCHWABER e SUTHERLAND, 2010, p. 6).

Então, segundo Schwaber e Sutherland (2010), para cada papel do *Scrum* existe um ou vários “porcos” em determinada área, que é ou são as pessoas comprometidas nesta área. O *Product Owner* é o “porco” do *Product Backlog*, o Time é o “porco” do trabalho da *Sprint*, e o *Scrum Master* é o “porco” do processo do *Scrum*. Qualquer outra pessoa é a “galinha”, pois está apenas envolvida.

2.4.1.1 Scrum Master

O trabalho do *Scrum Master* é o de gerenciar o processo e liderar as pessoas. Ele tem que buscar ter autoridade sobre as pessoas e não poder. As definições de autoridade e poder, segundo BLANCHARD e JOHNSON (2010), são as seguintes:

- Autoridade: A habilidade de levar as pessoas a fazerem de boa vontade o que você quer por causa de sua influência pessoal.
- Poder: É a faculdade de forçar ou coagir alguém a fazer sua vontade, por causa de sua posição ou força, mesmo que a pessoa preferisse não o fazer.

A grande diferença entre o gerente de projetos tradicional e o *Scrum Master* é justamente a de que o gerente de projetos tem mais poder, pois pode designar tarefas as pessoas, dizer o que elas devem fazer, enquanto no *Scrum*, o *Scrum Master* não tem o poder para fazer isso.

O que o *Scrum Master* pode e deve fazer é controlar o processo. Controlando o processo ele pode ter autoridade sem precisar dar ordens a ninguém, ele

consegue encorajar, através do processo, os integrantes da equipe a serem auto-organizáveis, a decidirem por eles mesmos. Por exemplo, ao se deparar com um problema no *software*, o *Scrum Master*, ao invés de tomar a decisão sozinho ou designar alguém para corrigir, ele pode definir uma reunião para que seja decidido entre os membros da equipe qual a melhor solução a se utilizar. Os integrantes, assim, se colocam na posição de voluntários na busca em resolver o problema.

O *Scrum Master* pode ser comparado a um *personal trainer*. O *personal trainer* ajuda a continuar seu regime de exercícios e realizá-los de forma correta. Um bom treinador motiva e, ao mesmo tempo, garante que você não está trapaceando. Contudo, a autoridade do treinador é limitada. O treinador não pode obrigar a realizar um exercício que você não queira. Ao invés, o treinador lembra dos seus objetivos e como você os escolheu (COHN, 2010).

2.4.1.2 Product Owner

O *Product Owner* é a pessoa que mostra o objetivo do projeto a equipe na reunião de planejamento do *Release*, e lembra disto constantemente durante o desenvolvimento do projeto. Ele quem escolhe o *Product Backlog* e o *Sprint Backlog* e define as prioridades das *User Stories*.

O *Product Owner* é responsável por garantir que a equipe trará bons retornos ao investimento que foi feito.

A responsabilidade de escolher quem responderá as perguntas da equipe também pertence ao *Product Owner*. Por exemplo, se Arthur é o gerente do departamento de RH, então quando a equipe tiver dúvidas como fazer a parte de cadastro de funcionários, o *Product Owner* pode delegar a equipe a conversar com Arthur. Mas, caso Arthur não esteja preparado ou estiver com dificuldades para responder as perguntas, é responsabilidade de o *Product Owner* delegar outra pessoa, ou, em ultimo caso, dar ele mesmo as respostas.

Quanto aos limites, o *Product Owner* é quem estipula, por exemplo, prazos de tempo para a entrega, quanto o time pode gastar ou quanto de memória o programa pode usar. O *Product Owner* tem todo o direito a estabelecer limites, ele só não pode

encobrir constantemente um problema nem estabelecer uma solução impossível de ser cumprida (COHN, 2010).

Quanto mais passar o tempo do projeto, mais o time precisará do *Product Owner* e menos ele precisará do *Scrum Master*, como mostrado na Figura 2. Isso acontece porque após algum tempo os membros do time já adquirirem uma natureza de auto-organização, além de os impedimentos já terem sido removidos, mas ao mesmo tempo eles precisam cada vez mais do *feedback* que o *Product Owner* traz (COHN, 2010).

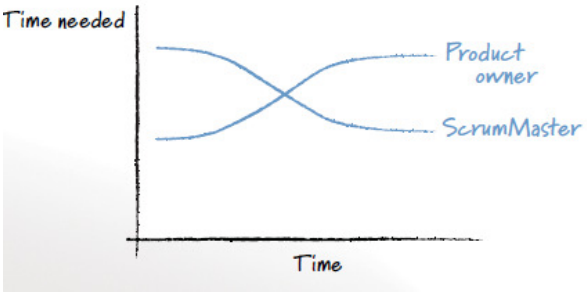

Figura 2 - Tempo necessário Product Owner x *Scrum* Master

Fonte: Cohn (2010, p. 128)

2.4.1.3 Time

O time é composto basicamente pelas pessoas que desenvolverão o produto. No time não há um líder técnico, todas as pessoas envolvidas buscam, além das suas especialidades, ajudar a equipe da maneira que for possível para transformar o *Product Backlog* em um produto de valor.

A maior mudança para os desenvolvedores ao adotar o *Scrum* é que estes não podem mais ficar isolados, com fones de ouvido, por exemplo, sem conversar com ninguém o dia inteiro. Deve haver uma boa e constante comunicação entre os membros do time durante todo o projeto, pois sem esta comunicação não seria possível adotar um cultura de auto-organização, onde todos os desenvolvedores, não importando se suas especialidades são a de desenvolver a análise, a arquitetura, a programação ou o teste, participam de todas as etapas, da análise a implantação.

O tamanho ideal para um time varia entre 5 e 9 pessoas. (SCHWABER e SUTHERLAND, 2010)

Segundo Cohn (2010) existem seis razões para manter as equipes pequenas:

- Menor sossego: Em um grupo pequeno, o sentimento de "não preciso fazer o trabalho, alguém irá fazê-lo" é menor do que em equipes maiores.
- Interação construtiva: O sentimento de confiança e responsabilidade mútua com equipes pequenas é maior.
- Menor tempo para coordenar os esforços: Times pequenos gastam menos tempo coordenando os esforços de seus membros. Até algo simples como organizar uma reunião com uma equipe grande se torna mais estressante.
- Ninguém fica de lado: Em equipes menores a chance de alguém ficar de fora das atividades de grupo e discussões é menor que em uma equipe grande.
- A satisfação entre os integrantes de uma equipe pequena é maior: Como em uma equipe menor as contribuições de uma só pessoa ficam mais visíveis e significantes, a satisfação dessa pessoa é maior.
- Indivíduos especializados em uma só área: Em equipes grandes os integrantes estão mais propensos a trabalhar somente em uma área, assim, reduzindo o nível de aprendizado.

2.4.2 Product Backlog

O *Product Backlog* é a lista de funcionalidades desejada para o produto. Quem define quais são e a prioridade da cada uma delas é o *Product Owner*.

A descrição das funcionalidades, escritas em forma de *User Stories*, em português “histórias do usuário”, é feita no início de forma simples, não buscando aprofundar-se, pois como os processos ágeis seguem uma idéia de mudanças constantes, seria considerado desperdício escrever funcionalidades complexas no inicio, por isso o amadurecimento das histórias vai ocorrendo longo do projeto de forma iterativa e incremental.

Boas *User Stories* devem ser independentes, negociáveis, testáveis, pequenas e estimáveis (RASMUSSON, 2010). Devem ser independentes, pois como

tudo muda muito depressa no projeto, se uma *User Storie* estiver atrelada a outra, a mudança fica mais complicada, devem ser negociáveis, pois existem diversas maneiras de desenvolver uma *User Storie*, às vezes precisamos de uma solução mais complexa, outras vezes não, a *User Storie* deve abrir espaço para negociações neste sentido, devem ser testáveis, pois devemos saber que ela está funcionando e devem ser pequenas e estimáveis, pois se deve garantir que a *User Storie* caiba dentro da *Sprint*, que geralmente varia entre duas a quatro semanas.

2.4.3 Sprint

A *Sprint* é um ciclo de trabalho que possui um curto espaço de tempo e que tem como objetivo entregar uma porção ou incremento de *software* que traga valor ao usuário.

O *Scrum* é iterativo e incremental e que, apesar de cada um ter um único significado, estes dois andam juntos. Em um processo incremental se desenvolve um aspecto do sistema e depois passa para o próximo, e em um processo iterativo, se constrói o sistema inteiro imperfeito primeiramente, depois vai se aprimorando ele. Combinando os dois as fraquezas de ser apenas incremental ou apenas iterativo desaparecem (COHN, 2010).

Dentro da *Sprint* o processo é iterativo e incremental, pois apesar de ser entregue uma porção de *software* que traz valor ao usuário, esta porção não precisa, necessariamente, ter todas as funcionalidades, esta porção de *software* pode ser incrementada depois.

2.4.4 Planning Poker

O *Planning Poker* é um método de estimativa das *User Stories*. Ele é um jogo onde inicialmente, cada integrante, individualmente, desenvolve estimativas para as *User Stories* utilizando um *deck* de cartas com números que variam entre um, dois, três, cinco e oito, e então a equipe compara os resultados definidos pelos membros. Se há divergência nas estimativas são feitas discussões até que se entre em um

consenso sobre o tamanho da *User Storie* discutida em questão (RASMUSSON, 2010).

2.4.5 Sprint Backlog

O *Sprint Backlog* é composto por *User Stories* definidas anteriormente no *Product Backlog* e que, geralmente, são *User storie*s que o Product Owner definiu serem mais urgentes e que se encaixaram no espaço de tempo definido pela *Sprint*.

Como as *User Stories* definidas no *Product Backlog* não vão muito a fundo tecnicamente sobre a funcionalidade, elas podem ser decompostas pela equipe em conjunto ao Product Owner. Por exemplo, uma *User Storie* definida como “Para que eu possa armazenar meus dados no sistema, eu cliente, desejo que o sistema tenha um cadastro de cliente” pode ser decomposta nas seguintes tarefas: “criar modelo entidade relacionamento para a entidade cliente”, “configurar a persistência para a entidade cliente”, “criar o *up-front* para o cadastro do cliente”.

2.4.6 Gráfico *Burndown*

O Gráfico *Burndown* segue a idéia da filosofia ágil de ter um *feedback* constante e o mais cedo possível do que está acontecendo.

No eixo y se controla a quantidade de trabalho restante, que pode ser definida em horas, dias ou pontos. No eixo x se controla o período. A Figura 3 apresenta um exemplo de gráfico *Burndown*, onde a linha vermelha apresenta as *User Stories* completas e a linha azul o ideal de *User Stories* completas até o determinado ponto.

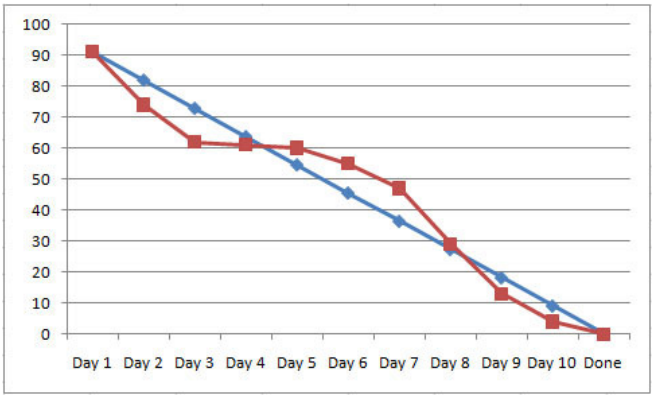

Figura 3 - Exemplo de Gráfico *Burndown*

Fonte: Girl Writes Code (2008)

Apesar de ser um gráfico simples, o *Burndown* nos mostra a quantidade de trabalho que foi realizado, a quantidade de trabalho que resta, a velocidade da equipe e o dia previsto da entrega.

Existem dois tipos de gráficos *Burndown* que são utilizados no *Scrum*, o primeiro é o *Burndown* do *release*, que mede progresso do *Product Backlog* e o *Burndown* de *Sprint,* que mede os itens do *Sprint Backlog* restantes.

2.4.7 Reunião de planejamento do release

A reunião de planejamento do *release é* também conhecida como a reunião do *Product Owner* com a equipe. É nesta reunião onde se definem o *Product Backlog*, os objetivos do projeto como um todo e garante que todos estão com as mesmas idéias de como será o projeto.

Segundo Schwaber e Sutherland (2010, p. 9), "o propósito do planejamento do *release* é o de estabelecer um plano e metas que o time de *Scrum* e o resto da organização possam entender e comunicar".

É na reunião de *release* que se define a viabilidade do projeto.

Segundo Rasmusson (2010), existem dez difíceis questões que se deve definir logo no início do projeto:

- Perguntar por que o time se encontra onde está: Esclarecer o porquê de o time estar onde está e quais são os verdadeiros objetivos do cliente faz com que o time tome decisões mais coerentes ao decorrer do projeto. Se o time conseguir entender o que realmente o cliente precisa, ao tomar uma decisão sobre como desenvolver uma funcionalidade, por exemplo, o time terá mais segurança de que a funcionalidade irá trazer valor para o cliente.
- Criar uma breve descrição do projeto: Ao criar uma breve descrição do projeto, o time é forçado a responder perguntas sobre o que é o produto e para quem é o produto, trazendo assim mais clareza para a equipe sobre seus objetivos. Tendo uma breve descrição do projeto sempre a vista faz com que os integrantes se lembrem qual o caminho correto a seguir.
- Desenvolver um slogan para o produto: Desenvolver um slogan do produto e se perguntar por que alguém o compraria, faz com que a equipe se foque no que realmente é atraente para o cliente. Sendo isso algo para se levar em conta na hora de desenvolver o produto.
- Criar uma lista do que não fazer: Tendo uma lista do que está fora do escopo do projeto faz com que, ao decorrer do projeto, a equipe não gaste tempo se preocupando com algo que não será desenvolvido.
- Conhecer os demais envolvidos com o projeto: Desenvolver uma relação com os demais envolvidos no projeto, e não só com sua equipe, é uma boa prática, pois, quando o time precisar deles, eles não serão completos estranhos, fazendo com que estejam mais dispostos a cooperar.
- Mostrar a solução técnica escolhida: Esclarecendo a solução técnica para o que será desenvolvido, por exemplo, qual banco de dados será usado, faz com que confirme se todos estão realmente de acordo com a solução proposta.
- Definir quais são os riscos para o projeto: Definir os riscos do projeto traz uma discussão sobre quais serão os desafios que a equipe encontrará ao longo do projeto, se prevenindo de serem pegos de surpresa. Existem riscos que valem e outros que não valem a pena correr no projeto, e o quanto mais cedo se descobrir quais deles existem no projeto, melhor.
- Estimar o tamanho do projeto: Logo no início do projeto não há como ser muito preciso quanto ao tempo que levará para desenvolver todas as

atividades propostas, mas os clientes precisam ter ao menos uma idéia do que eles podem esperar para o *software* ser entregue.

- Ser claro no que será cedido: São quatro aspectos que controlam o seu projeto e que estão em constante conflito. Estes aspectos são escopo, orçamento, tempo e qualidade. Alterar qualquer um deles resulta em impacto nos outros. Como qualidade é fator número um, o orçamento normalmente é limitado e aumentando o tempo deve se aumentar o orçamento, o que sobra é o escopo, sendo este assim o mais fácil de negociar.
- Mostrar o que irá custar: Discutir qual a responsabilidade de cada indivíduo, inclusive a do cliente, é essencial para que a equipe saiba quem é que toma as decisões sobre tal área. Deve-se também estimar quanto irá custar para desenvolver o produto em termos financeiros, para deixar isso claro ao cliente.

2.4.8 Reunião de planejamento da Sprint

Na etapa da reunião de planejamento da *Sprint,* que ocorre sempre antes do desenvolvimento da *Sprint*, o *Product Backlog* já foi desenvolvido e resta então ao *Product Owner* escolher quais as *User Stories* que possuem maior urgência, para que estas tenham maior prioridade a serem desenvolvidas, levando em conta que estas devem encaixar no tempo da *Sprint*.

Durante esta etapa a equipe levanta também questões ao *Product Owner* sobre as *User Stories* onde estes buscam mais detalhes. Por exemplo, no *Product Backlog*, em uma *User Storie* definida como “cadastro dos dados do cliente”, podem ser esclarecidas questões como quais são os dados que o cliente precisa para se cadastrar e quais as validações desejadas.

As *User Stories* escolhidas pelo *Product Owner* são decompostas em atividades que, nesta mesma reunião de planejamento da *Sprint*, os integrantes do time decidem como serão desenvolvidas.

Todas as atividades definidas no planejamento do *Sprint* fazem parte do *Sprint Backlog*.

2.4.9 Revisão da Sprint

Ao final de cada *Sprint* é realizada uma reunião com o cliente para mostrar todo o trabalho que foi realizado durante a *Sprint*, incluindo, é claro, o produto de valor agregado que foi desenvolvido.

Lembrando que uma das atividades mais incentivadas pela filosofia ágil é a análise do que está acontecendo e melhoria contínua. Na revisão da *Sprint*, também, os membros da equipe param para refletir o que foi feito e como este processo pode melhorar.

2.4.10 Daily Scrum

O *Daily Scrum* é um encontro que acontece diariamente, tem uma duração de 15 minutos e preferencialmente deve ser feito no mesmo local e no mesmo horário durante as *Sprints* (SCHWABER e SUTHERLAND, 2010). A reunião é normalmente feita em pé para lembrar a equipe para não exceder os 15 minutos.

Esta reunião ocorre com o objetivo de que os membros continuem sincronizados, dia após dia, durante o desenvolvimento da *Sprint*.

O discutido na reunião é basicamente o que cada integrante desenvolveu no dia (anterior ou atual, dependendo do horário que acontece a reunião), se teve algum obstáculo e o que ele como integrante ou toda a equipe pode fazer para melhorar o desenvolvimento.

2.4.11 Programação em Par

A programação em par refere-se à prática de dois desenvolvedores juntos escreverem o código.

Esta prática favorece o companheirismo e a troca de conhecimento entre os membros da equipe, além de evitar, muitas vezes, que erros passem despercebidos.

2.4.12 Práticas técnicas

Por todo o tempo gasto em planejamento e gestão de expectativas, processos ágeis não funcionam a menos que sejam apoiados por um sólido conjunto de práticas de engenharia de *software* (RASMUSSON, 2010).

2.4.12.1 Refatoração

Refatoração é o processo de mudança de um sistema de *software* de tal forma que não altera seu comportamento externo, mas melhora a estrutura interna de seu código. É um modo disciplinado de limpar o código e melhorar sua estrutura (FOWLER, 2002).

Refatorar é fazer o máximo para minimizar a dívida técnica do código.

Dívida técnica é o acúmulo contínuo de atalhos, *hacks*, duplicação, e outros pecados que cometemos regularmente contra a nossa base de código em nome da velocidade e do prazo (RASMUSSON, 2010).

O que acontece é que estes atalhos, apesar de no início parecer que trarão vantagens, acabam sendo muito custosos em um longo prazo, pois o preço de dar manutenção neste código é muito alto. Na Figura 4, pode-se comparar o custo de manutenção de um código refatorado e outro não.

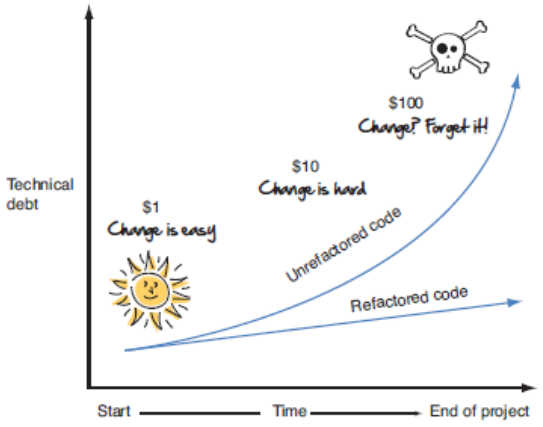

Figura 4 - Custo de mudança

Fonte: Rasmusson (2010, p. 215)

2.4.12.2 Desenvolvimento orientado a testes

Desenvolvimento orientado a testes é a prática na qual os testes são desenvolvidos antes de se desenvolver o código da aplicação, ela se baseia na idéia de que em agilidade se busca resolver o quanto antes os problemas.

Sua aplicação é composta por vários pequenos ciclos compostos por três etapas.

A primeira etapa é de fazer o design e escrever o teste para a funcionalidade. O teste irá falhar, levando em conta que o código da aplicação não foi escrito.

A segunda etapa é a etapa em que se escreve o código do *software* referente ao teste, para que este passe.

A terceira etapa é a de refatoração, nela se volta ao código escrito e verifica se há alguma maneira de diminuir os débitos técnicos deste.

Além de dar uma maior segurança na hora de continuar o desenvolvimento, escrever os testes antes garante que o código que for escrito seja testável, e códigos testáveis tendem a ser mais claros de se entender.

2.4.12.3 Integração contínua

A integração contínua refere-se a integração de código novo ou alterado em uma aplicação o mais rapidamente possível e, em seguida, testar a aplicação para se certificar de que nada foi quebrado (COHN, 2010).

Tendo uma integração contínua, a equipe evita de correr o risco de alguém estar utilizando uma versão obsoleta do código, além de descobrir possíveis erros da integração em uma etapa inicial. A integração contínua evita a dor de cabeça de descobrir que, após várias semanas ou meses, ao juntar o trabalho de todo mundo, algumas “peças” não estão se encaixando.

2.5 FERRAMENTAS DE AUXILIO PARA EQUIPES DISTRIBUIDAS

Para auxiliar a comunicação e o trabalho em conjunto das equipes que não podem estar no mesmo espaço de trabalho ao mesmo tempo existem ferramentas, algumas das quais serão citadas nas próximas subseções.

2.5.1 Google Docs

O Google Docs é um conjunto de ferramentas grátis composto por uma ferramenta de processamento de texto, um editor de planilhas, um editor de apresentações, um editor de formulários e um editor de desenhos. Este conjunto de ferramentas pode ser acessado de qualquer computador conectado a internet. Todos os documentos que são produtos da utilização destas ferramentas são armazenados nos servidores da empresa Google. Este conjunto de ferramentas permite que os usuários criem e editem documentos online enquanto colaboram em tempo real uns com os outros. Na Figura 5 pode ser visto a *interface* inicial do Google Docs.

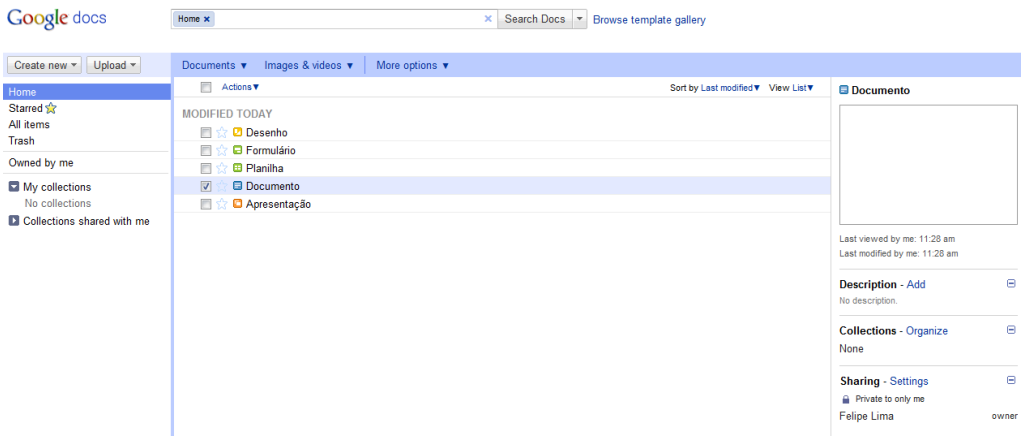

Figura 5 - Google Docs

2.5.2 TeamViewer

O *TeamViewer* é um software para controle remoto, compartilhamento de desktop, conversa por áudio e transferência de arquivos entre computadores. Com ele é possível que duas ou mais pessoas compartilhem do mesmo computador, através da internet, e consigam se comunicar, fazendo com que a experiência da programação em par – atividade do *Scrum* - se aproxime da ideal.

3 ESTUDO EXPERIMENTAL

A aplicação desenvolvida para este trabalho tem como objetivo informatizar o processo no qual em um estabelecimento, normalmente em consultórios médicos e odontológicos, a secretária registra as consultas dos clientes em diversas agendas, uma para cada profissional.

3.1 MATERIAIS E MÉTODOS

Para o desenvolvimento do projeto foi realizada uma pesquisa bibliográfica, além de consultas em sites que falam sobre *Scrum* e sua aplicação, a fim de dar subsídios suficientes a gestão do projeto.

Devido aos membros da equipe estarem distribuídos geograficamente, foram utilizadas as ferramentas Google Docs, para colaboração na edição de documentos, a fim de documentar as atividades e os objetivos e a utilização do *TeamViewer* para a comunicação via voz e compartilhamento de um mesmo computador através da internet.

3.2 EQUIPE

A equipe de desenvolvimento foi composta por três integrantes, o primeiro integrante assumiu o papel de *Scrum Master* e desenvolvedor, o segundo assumiu o papel de *Product Owner* e desenvolvedor e o terceiro assumiu o papel de desenvolvedor.

3.3 REUNIÃO DE PLANEJAMENTO DO RELEASE

Durante a reunião de planejamento do *release* foram definidos os objetivos principais do projeto, os riscos provenientes, as *User Stories* principais, os esboços das telas. Foi definido também o tempo máximo para o término.

O *Product Owner* definiu e esclareceu os objetivos principais do sistema, que consiste em gerenciar os profissionais que fazem parte do estabelecimento e agendar horários para consulta dos clientes.

Foram discutidos os riscos do projeto, e o mais evidente era o de a equipe não estar co-alocada, ou seja, durante o desenvolvimento os desenvolvedores não estariam juntos na mesma sala. Apesar do risco, foi decidido continuar com o projeto, pois a equipe possuía as ferramentas Google Docs e *TeamViewer* para auxiliar na comunicação entre os membros, mesmo a distância.

Por se tratar de um projeto pequeno, foi definido um prazo de três meses para o término do mesmo.

A escala de tamanho para cada *User Storie* foi definida em pontos, quais eram estimados utilizando a prática de *Planning Poker*.

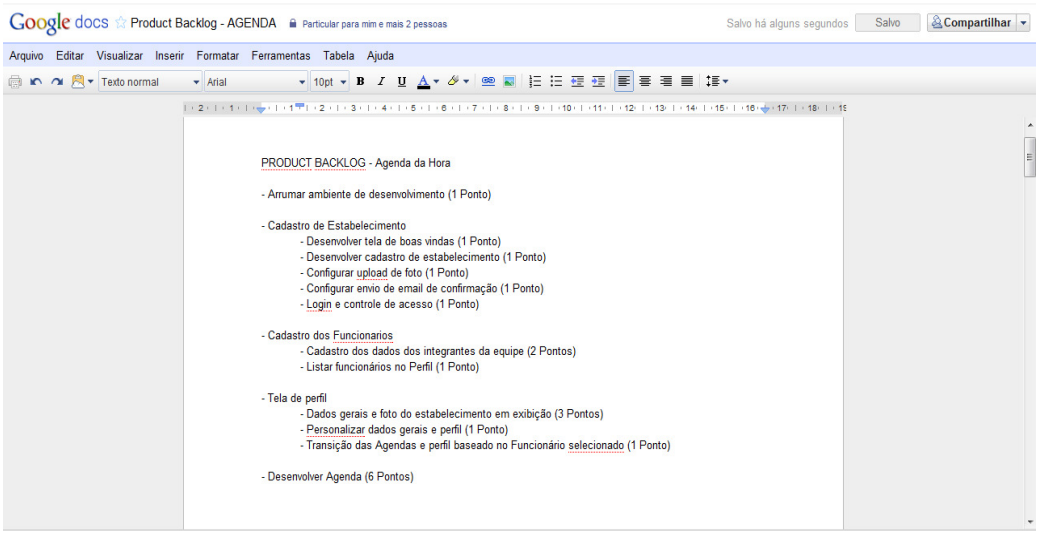

Figura 6 - *Product Backlog*

As *User Stories* foram definidas no *Product Backlog,* que pode ser visto na Figura 6, serão descritas nas seções seguintes:

- Cadastro de Estabelecimento: Para que o administrador do estabelecimento possa cadastrar empregados e gerenciar suas respectivas agendas, desenvolver um cadastro de estabelecimento. O tamanho da *User Storie* foi definido em cinco pontos.

Figura 7 - Esboço da *User Storie* "Cadastro de Estabelecimento"

- Cadastro de Empregados: Para que cada empregado tenha seus dados armazenados e uma agenda referente a ele, desenvolver um cadastro de empregados. O tamanho da *User Storie* foi definido em três pontos.

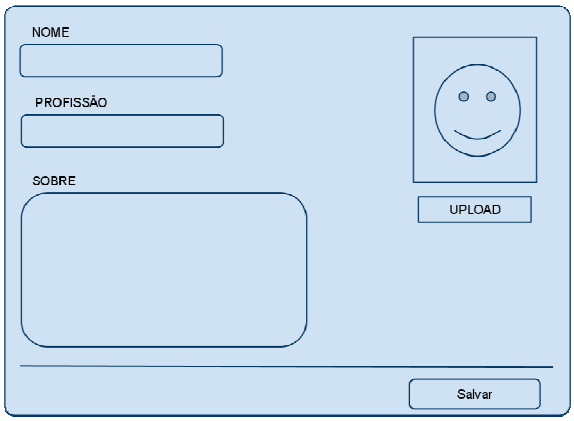

Figura 8 - Esboço da *User Storie* "Cadastro de Empregados"

- Desenvolvimento da Agenda: Para que se o administrador possa marcar e alterar horários para seus clientes, desenvolver a agenda. O tamanho da *User Storie* foi definido em seis pontos.

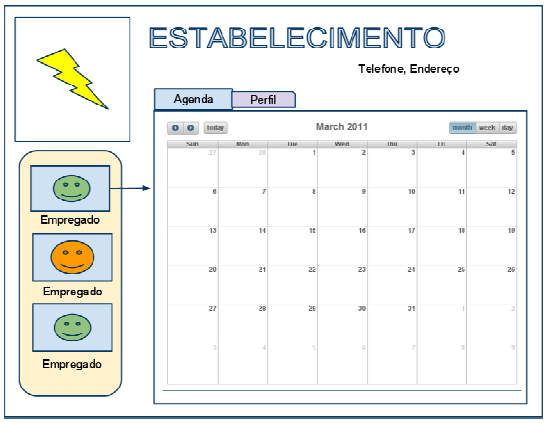

Figura 9 - Esboço da *User Storie* "Desenvolvimento da Agenda"

- Desenvolvimento do Perfil: Para que o administrador possa navegar entre os empregados, alterar os dados e acessar as agendas, desenvolver uma tela de perfil. O tamanho da *User Storie* foi definido em cinco pontos.

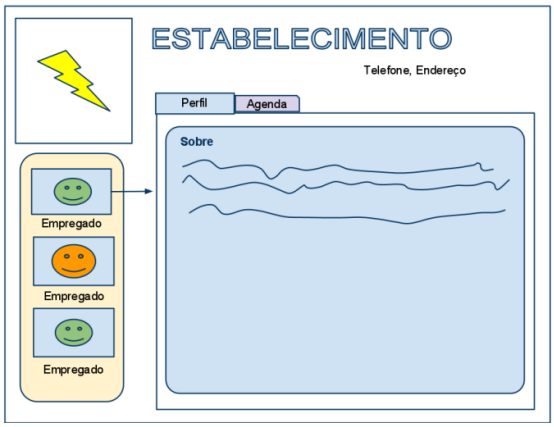

Figura 10 - Esboço da *User Storie* "Desenvolvimento do Perfil"

Foi definido que as *Sprints* teriam duração de um mês e calculado, através de uma estimativa dos membros da equipe, que a média de velocidade aproximada para que a equipe pudesse alcançar o objetivo era a de seis pontos por *Sprint*.

3.4 PRIMEIRA SPRINT

Como objetivos da primeira *Sprint*, foram escolhidos os de desenvolver o Cadastro de Estabelecimento juntamente com a configuração do ambiente de desenvolvimento.

A escolha da *User Storie* “Cadastro de Estabelecimento” como primeira se deu pois ela não possuía nenhuma dependência de outras *User Stories*.

3.5 SEGUNDA SPRINT

Para a segunda *Sprint* os itens do *Product Backlog* escolhidos foram o cadastro de empregados, o desenvolvimento do perfil, além dos itens que foram retirados do escopo na primeira *Sprint*: o envio de email de confirmação e o *upload* de foto.

A segunda *Sprint* foi concluída antes do tempo, fazendo com que a terceira *Sprint* começasse antes.

3.6 TERCEIRA SPRINT

A atividade proposta para a terceira e última *Sprint* foi a de desenvolver a agenda em si, onde o administrador poderia marcar horário para seus clientes.

3.7 REVISÃO DAS SPRINTS

Na revisão da primeira *Sprint* foi constatado que as expectativas quanto a conclusão das tarefas propostas na reunião de planejamento desta *Sprint* não foram alcançadas, e a causa da diminuição do desempenho da equipe foi a de um acidente de trânsito envolvendo um dos integrantes da equipe, que deixou a equipe desfalcada por algumas semanas. Sendo que o risco de a equipe ter um de seus integrantes ausentado novamente era baixo, a estimativa de velocidade não foi alterada. As tarefas que não puderam ser concluídas na primeira *Sprint* voltaram ao *Product Backlog*.

Na revisão da segunda *Sprint* foi constatado que a equipe teve um aumento significativo na velocidade, pois as tarefas propostas foram desenvolvidas antes do prazo.

A terceira revisão de *Sprint* foi marcada pela avaliação da versão final do sistema, qual alcançou as expectativas definidas no inicio do projeto.

3.8 GRÁFICO *BURNDOWN*

A equipe utilizou o gráfico *burndown* para acompanhar o ritmo de seu desenvolvimento. Segue na Figura 11 o resultado final do gráfico, onde a linha azul apresenta as *User Stories* completas e a linha vermelha o ideal de *User Stories* completas até o determinado ponto.

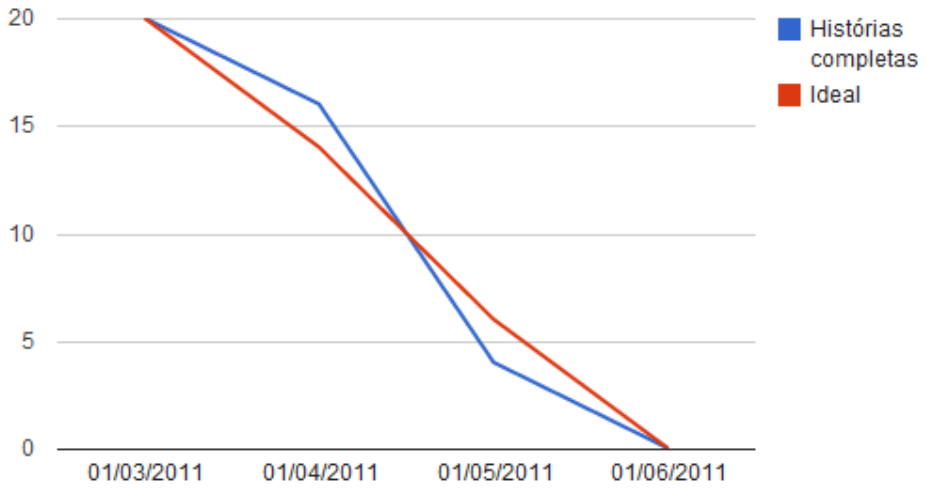

Figura 11 - Gráfico burndown do desempenho da equipe

3.9 DAILY SPRINTS

As *Daily Sprints* eram realizadas todos os dias do desenvolvimento do projeto, sempre no início da tarde (o horário de desenvolvimento ia entre as duas horas da tarde até as cinco horas da tarde, de segunda à quinta). Eram discutidas questões sobre o que cada um fez no dia anterior e cada integrante escolhia uma tarefa para realizar ou continuar realizando durante o dia.

4 CONSIDERAÇÕES FINAIS

Os processos ágeis, através dos seus valores como a busca pela excelência técnica, do desenvolvimento do trabalho em equipe, da comunicação e colaboração entre o cliente e o desenvolvedor prometem e cumprem o papel de trazer agilidade no desenvolvimento, *software*s com maior valor para o cliente e ambos, clientes e desenvolvedores, mais satisfeitos. A adoção de processos ágeis, seja onde for, não é como um projeto que tem início e fim, ele deve sempre estar se adaptando, afinal sempre existirão novos projetos com características diferentes para serem realizados e integrantes com culturas diferentes participando da equipe. A adaptação tem que buscar aperfeiçoar o processo para que cada integrante consiga alcançar todo o seu potencial, ou pelo menos chegar perto disto. Colocando como exemplo o estudo experimental, a idéia padrão do *Scrum* de manter sempre as *Sprints* com o mesmo tamanho teve de ser adaptada, pois a equipe achou que seria mais eficiente definir o término da *Sprint* baseado em conjuntos de funcionalidades prontas ao invés de definir um tempo padrão para o término. Para a característica do projeto do estudo experimental a adaptação teve resultados positivos, mas isso não significa que esta prática funcionaria em todos os projetos ágeis.

Deve-se tomar muito cuidado ao realizar adaptações no processo para que os valores da filosofia ágil não sejam feridos. Deixar de lado as práticas técnicas de engenharia de *software*, por exemplo, pode trazer problemas que atrasem a equipe por um tempo considerável, ao mesmo tempo em que deixar de lado as técnicas de liderança, que buscam trazer motivação aos integrantes e melhorar o ambiente de trabalho, acaba diminuindo o comprometimento dos integrantes em buscar qualidade.

4.1 TRABALHOS FUTUROS

O projeto de estudo experimental foi desenvolvido em um curto espaço de tempo e com uma equipe de tamanho menor que a ideal. Uma sugestão para trabalhos futuros seria a de implantar o *Scrum* em projetos de longa duração e envolvendo uma quantidade de indivíduos entre cinco e nove integrantes, o que se aproxima mais do proposto pelo processo.

REFERÊNCIAS BIBLIOGRÁFICAS

BLANCHARD, Kenneth; JOHNSON, Spencer. O Gerente Minuto. 2010.

BUCKINGHAM, Marcus; CLIFTON Donald. Now, Discover your Strenghts. 2001.

CARNIAGE, Dale. Como fazer amigos e influenciar pessoas. 1995.

COHN, Mike. Succeeding With Agile: Software Development Using Scrum. 2010.

FOWLER, Martin. Refactoring: Improving the Design of Existing Code. 2002.

GIRL WRITES CODE. Disponível em: <http://www.invisible-city.com/sharon/2008/09/sprint-heartbeat-visual-task-tracking.html>. Acesso em: 17/06/2011.

HOWARD, Ken; ROGERS, Barry; Individuals and Interactions: An Agile Guide. 2011.

MANIFESTO ÁGIL. Disponível em: <http://agilemanifesto.org/>. Acesso em: 10/03/2011.

PROJECT MANAGEMENT INSTITUTE. A Guide to the Project Management Body of Knowledge – PMBOK® Guide Fourth Edition. 2008.

RASMUSSON, Jonathan. The Agile Samurai: How agile masters deliver great software. 2010.

SCHWABER, Ken; SUTHERLAND, Jeff. SCRUM Guide. 2010.

SCRUM ALLIANCE. Disponível em: <http://www.scrumalliance.org/pages/what_is_scrum>. Acesso em: 20/05/2011.

UNIVERSO ÁGIL. Disponível em: <http://universoagil.com.br>. Acesso em 10/03/2011.