Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis - IMESA

ANDRÉ LUIS DE OLIVEIRA BERARDI

METODOLOGIAS ÁGEIS EM DESENVOLVIMENTO DE SOFTWARE

ASSIS
2011

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis - IMESA

ANDRÉ LUIS DE OLIVEIRA BERARDI

Pré-projeto de pesquisa apresentado ao curso de Tecnologia em Processamento de Dados do Instituto Municipal de Ensino Superior de Assis - IMESA e a Fundação Educacional do Município de Assis – FEMA, como requisito parcial a obtenção do Certificado de Conclusão.

Orientando: André Luis de Oliveira Berardi
Orientador: Dr. Alex Sandro Romeo de Souza Poletto

ASSIS
2011

FICHA CATALOGRÁFICA

BERARDI, André Luis de Oliveira.

Metodologias Ágeis em Desenvolvimento de Sistemas / André Luis de Oliveira. Fundação Educacional do Município de Assis – FEMA – Assis, 2011.

43p.

Orientadora: Prof. Dr. Alex Sandro Romeo de Souza Poletto

Trabalho de Conclusão de Curso – Instituto Municipal de Ensino Superior de Assis – IMESA.

1.Scrum. 2.Extreme Programming. 3.Metodologias Ágeis.

CDD:001.61

Biblioteca da FEMA.

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis - IMESA

METODOLOGIAS ÁGEIS EM DESENVOLVIMENTO DE SOFTWARE

ANDRÉ LUIS DE OLIVEIRA BERARDI

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito de Curso de Tecnologia em Processamento de Dados, analisado pela seguinte comissão examinadora:

Orientador: Dr. Alex Sandro Romeo de Souza Poletto

Analisador: Dr. Almir Rogério Camolesi

ASSIS
2011

AGRADECIMENTO

Primeiramente agradeço a Deus e á minha família, que fizeram parte desta jornada e que estiveram junto a mim e também agradeço aos Mestres pela paciência e dedicação ao longo desses anos, e aos ensinamentos que foram de grande valor e a VSM, juntamente com o Adriano Alves Dornelas, por ter aceito e contribuído pela conclusão deste trabalho.

RESUMO

Este trabalho apresenta um estudo das metodologias ágeis de desenvolvimento. Foi feito um breve estudo, detalhando as metodologias tradicionais e ágeis, suas diferenças, e comparações entre o *Scrum* e *XP (Extreme Programming).*

Palavras-chaves: *Scrum; Extreme Programming* ; Metodologias ágeis

ABSTRACT

This work presents a study of agile methodologies for development. It was done a short study, detailing the traditional methodologies and agile, their differences, and comparisons between the *Scrum* and *XP* (*Extreme Programming*).

Keywords: *Scrum; Extreme Programming* ; agile Methodologies

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

XP – Extreme Programming

PAF – Programa Aplicativo Fiscal

ECF – Emissor de Cupom Fiscal

SUMÁRIO

1. INTRODUÇÃO

Atualmente, e sem duvida nenhuma a área de desenvolvimento de *software* é uma das que mais cresce e tende a evoluir no mundo todo, a necessidade constante de um desenvolvimento ágil é fator que movimenta muitas empresas, dos mais variados ramos, sendo primordial para movimentação dessas muitas empresas. E é nesse contexto de agilidade e fluidez do mercado que acarretou no surgimento de várias metodologias ágeis para gestão de desenvolvimento de produto. Hoje o software tornou-se essencial para a manutenção do controle e da produtividade nos mais diversos setores organizacionais, novas metodologias e técnicas buscam garantir uma maior agilidade e ao mesmo tempo solidez no processo de desenvolvimento de software.

As metodologias ágeis no desenvolvimento de *software*, traz a integração de ferramentas, pessoas e processos e é capaz de reduzir custos, prazos e riscos (FILHO, PENTEADO, SILVA, BRAGA 2005). E com o aumento da competitividade dos mercados, a exigência de um produto desses tornou-se um diferencial determinante. O trabalho trouxe no seu contexto a história e o surgimento das metodologias ágeis, o conceito dessas metodologias. Procura-se conceituar um pouco do *SCRUM* e *Extreme Programming (XP)* e o comparativo delas. Uma pesquisa elaborada por meio de uma entrevista junto a uma Empresa teve como intuito de mostrar as vantagens e desvantagens na utilização das metodologias ágeis no desenvolvimento de software dentro de uma empresa.

1.1. OBJETIVOS

O objetivo do trabalho é mostrar o quanto é primordial um método ágil de desenvolvimento de *software* nas empresas, sendo elas tanto de grande, médio ou pequeno porte, o que lhe garantiriam maior agilidade, reduziria custos, prazos e principalmente riscos.

1.2. JUSTIFICATIVA E MOTIVAÇÃO

O desenvolvimento deste trabalho se justifica pelo fato de que empresas de médio e pequeno porte, que representam a maioria em nosso país, ainda não utilizam nenhuma metodologia para o desenvolvimento de seus *softwares,* provocando uma grande perda de tempo na criação de seus aplicativos, e até mesmo perdas financeiras, tornando o risco muito maior.

O interesse nesta metodologia surgiu em mostrar que as metodologias ágeis de desenvolvimento de *software* beneficiam hoje em muito as empresas, o que garantiria uma maior satisfação do cliente com desenvolvimento de *softwares* mais funcionais, sendo que hoje o mercado exige maior flexibilidade e rapidez nas mudanças constantes. Por isso a falta de conhecimento no meio acadêmico motivou a desenvolver uma pesquisa nesta área que é de grande relevância para as empresas de grande, médio e pequeno porte.

1.3. PERSPECTIVAS DE CONTRIBUIÇÃO

Esses métodos podem ser usados por qualquer empresa, trazendo grandes benefícios, uma vez que a maioria desses métodos tenta minimizar o risco pelo desenvolvimento do *software* em curtos períodos, funcionando em ambientes de negócios muito exigentes e que tem um grande número de incertezas, como novos concorrentes, novos produtos, novos modelos de negócio e por isso a adoção de novas tecnologias, novas ferramentas, onde contribuirá muito nas empresas, sendo elas de grande, médio ou pequeno porte.

1.4. ESTRUTURA DO TRABALHO

Este trabalho foi organizado em seis capítulos, sendo o primeiro esta introdução.

No segundo capítulo será apresentada a história das metodologias tradicionais.

No terceiro capítulo será apresentada a Metodologia Ágil de Desenvolvimento de *Software.*

No quarto capítulo será apresentado o comparativo entre as Metodologias tradicionais e Ágeis.

No quinto capítulo será apresentado um estudo de caso real, com a aplicabilidade do *SCRUM.*

No sexto capítulo serão apresentadas as considerações finais e as projeções futuras.

2. METODOLOGIA TRADICIONAL

Segundo Pressman (2006), na década de 70, a atividade de "desenvolvimento de *software*" era executada de forma desorganizada, desestruturada e sem planejamento, gerando um produto final de má qualidade. Não correspondia com as reais necessidades do cliente, além do custo que era muito alto, para se fazer alterações, pois o acesso aos computadores era limitado e não existiam modernas ferramentas de apoio ao desenvolvimento do *software*, como depuradores e analisadores de código. A partir deste cenário, surgiu a necessidade de tornar o desenvolvimento de *software* como um processo estruturado, planejado e padronizado. Sendo que muitas metodologias pesadas eram desenvolvidas com base no modelo cascata.

2.1. SURGIMENTO DAS METODOLOGIAS ÁGEIS

Segundo FAGUNDES (2005), a partir da década de 90 começaram a surgir novos métodos sugerindo uma abordagem de desenvolvimento ágil, em que os processos adotados tentam se adaptar às mudanças, apoiando a equipe de desenvolvimento em seu trabalho.

O termo "Metodologia Ágil" surgiu quando um grupo de 17 especialistas que eram referências mundiais em desenvolvimento de *software* e criou a Aliança Ágil e estabeleceram o Manifesto Ágil para o desenvolvimento de *software* tornando-se popular em fevereiro de 2001 (LUDVIG, REINERT 2009).

Os valores do Manifesto Ágil são:

• Indivíduos e interações valem mais que processos e ferramentas;

• Um *software* funcionando vale mais que documentação extensa;

• A colaboração do cliente vale mais que a negociação de contrato;

• Responder a mudanças vale mais que seguir um plano.

Para se ter a compreensão melhor do enfoque do desenvolvimento ágil os membros da Aliança Ágil, criaram uma coleção de doze princípios para adequação melhor desses métodos ágeis.

Os princípios são:

1. A prioridade é satisfazer ao cliente através de entregas de *software* de valor contínuas e freqüentes;

2. Entregar *softwares* em funcionamento com freqüência de algumas semanas ou meses sempre na menor escala de tempo;

3. Ter o *software* funcionando é a melhor medida de progresso;

4. Receber bem as mudanças de requisitos, mesmo em uma fase avançada, dando aos clientes vantagens competitivas;

5. As equipes de negócio e de desenvolvimento devem trabalhar juntas diariamente durante todo o projeto;

6. Manter uma equipe motivada fornecendo ambiente, apoio e confiança necessário para a realização do trabalho;

7. A maneira mais eficiente da informação circular dentro da equipe é através de uma conversa face a face;

8. As melhores arquiteturas, requisitos e projetos provêm de equipes organizadas;

9. Atenção contínua a excelência técnica e um bom projeto aumentam a agilidade;

10. Processos ágeis promovem o desenvolvimento sustentável. Todos envolvidos devem ser capazes de manter um ritmo de desenvolvimento constante;

11. Simplicidade é essencial;

12. Em intervalos regulares, a equipe deve refletir sobre como se tornarem mais eficazes e então se ajustar e adaptar seu comportamento.

3. METODOLOGIA ÁGIL DE DESENVOLVIMENTO DE *SOFTWARE*

Hoje tem que se destacar a importância de se ter uma visão de futuro bem estabelecida e aceita na empresa ampliando o leque de oportunidades de investimento em projetos inovadores. É necessário que se estabeleça um referencial que indique para onde a empresa deve apostar todos os seus recursos, por isso o desenvolvimento de *software* com metodologias ágeis traz uma nova perspectiva de um *software* desenvolvido com qualidade e satisfação para o cliente.

Um dos grandes diferenciais das metodologias ágeis é a questão dos prazos de entrega do *software,* visto que os *software* são desenvolvidos por partes, com maior relevância para os clientes, dando-se apenas prazo estimado, que varia entre uma ou três semanas, conforme a equipe atual e caso a caso. Na questão do desenvolvimento do *software* a equipe deve estar em sintonia e o gerente do projeto precisa pensar na equipe sempre antes de qualquer tomada de decisão, sendo que a comunicação entre cliente e desenvolvedor é primordial nas metodologias ágeis.

Pode-se dizer que hoje existem vários métodos de desenvolvimento ágil de *software* sendo que as principais que mais se destacam são *Extreme Programming* (*XP) e* o *Scrum,* descritas a seguir.

3.1. *SCRUM*

A metodologia *Scrum,* é uma metodologia ágil de simples abordagem, criada em 1995 como um *framework* para gerenciamento de projetos.

O *Scrum* trabalha com a complexidade, sendo que o processo deve ser controlado empiricamente para garantir a visibilidade, inspeção e adaptação, requisitos de um processo empírico (LIBARDI, BARBOSA, 2010).

Dentro de uma série de regras e práticas, o controle não significa controle para criar o que foi previsto e sim controlar o processo para orientar o trabalho em direção a um produto com o maior valor agregado possível.

Para se alcançarem os objetivos propostos do *Scrum*, emprega-se uma estrutura iterativa e incremental da seguinte forma: no início de cada iteração, a equipe

analisa o que deve ser realizado e então seleciona aquilo que acreditam para poder-se tornar um incremento de valor ao produto ao final da iteração. No desenvolvimento da interação e ao final apresenta-se o incremento da funcionalidade construído para que os *stakeholders* possam verificar e requisitar alterações no momento apropriado. Sendo assim, podemos dizer que o coração do *Scrum* é a iteração.

3.1.1. Conceitos e Artefatos dentro do Processo de Desenvolvimento com *Scrum*

3.1.1.1. *Product Backlog*

No *product backlog,* há uma lista priorizada dos requisitos, tanto funcional como não funcional. Em cada item desta lista se encontra um valor de negócio associado, onde pode-se medir o retorno do projeto e a priorização dos itens.

3.1.1.2. *Sprint*

Cada iteração do processo de desenvolvimento é denominada de *Sprint*. A duração de cada *Sprint* é recomendando que fique entre dois e quatro semanas.

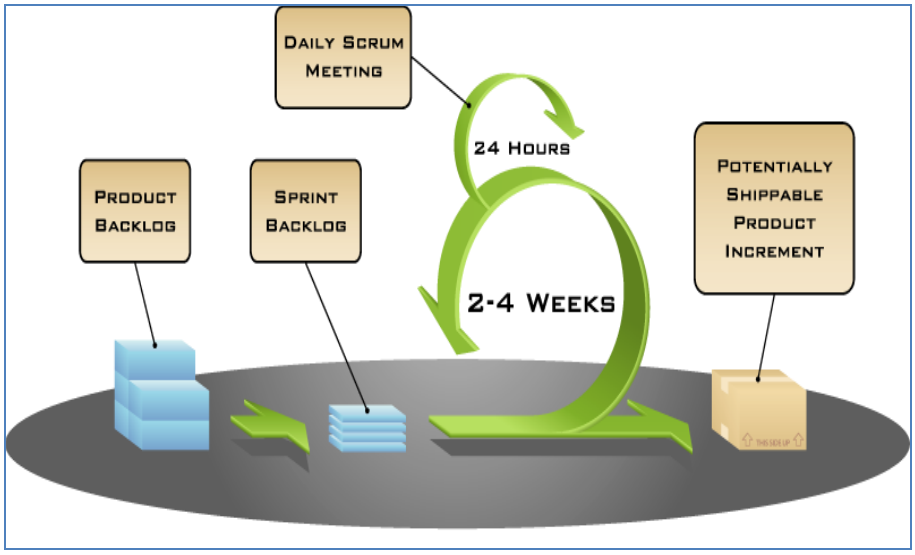

Figura 1. *Sprint* (Todo o processo de desenvolvimento seguido pelas regras e práticas do *Scrum*)

3.1.1.3. *Sprint Backlog*

Seria uma lista de tarefas, onde se encontra o trabalho da equipe em cada *Sprint* do processo. A lista nasce durante o planejamento do *Sprint*. As tarefas do *Sprint Backlog* são o que a equipe definiu como sendo necessário para a fluência da realização dos itens do *Product Backlog* nas funcionalidades do sistema. Cada tarefa é identificada pelo seu responsável e a sua quantidade estimada de trabalho restante.

3.1.2. Papéis Implementados Pelo *Scrum* e Responsabilidades:

3.1.2.1. *Product Owner*

É o “dono do produto”, é a pessoa que identifica o interesse de todos no projeto. Além de priorizar os requisitos do projeto é o responsável pelo seu ROI (*return of investment*). Trocando em miúdos: é o dono do produto que sabe o que vai gerar retorno ao projeto.

3.1.2.2. *Scrum Master*

É a pessoa responsável por fazer o *Scrum f*uncionar. Deve ensinar a metodologia a todos os envolvidos no processo, assim como assegurar que todos sigam suas regras e práticas. Trabalha juntamente com o *Product Owner* na organização dos requisitos.

3.1.2.3. Team Members

Desenvolvem as funcionalidades do produto, são responsáveis coletivamente pelo sucesso da iteração e conseqüentemente pelo projeto como um todo.

3.1.2.4. Fluxo de Desenvolvimento

No *Scrum,* um projeto se inicia com a visão do produto que será desenvolvido. Em seu desenvolvimento, são realizados vários *Sprints*. A cada *Sprint* são estabelecidos novos itens a serem desenvolvidos, e durante sua realização temos diariamente reuniões de aproximadamente 15 minutos com os membros da equipe. No final de cada *Sprint,* é feita uma retrospectiva para avaliar como esta o andamento do processo, posteriormente são levantados os próximos itens a serem desenvolvidos para a realização de um novo *Sprint.*

3.2. EXTREME PROGRAMMING

O *Extreme Programming*, é uma metodologia voltada para projetos cujos requisitos se alteram com freqüência e é utilizada no desenvolvimento orientada a objetos (KUHN, PAMPLONA, 2009). O *Extreme Programming*, é conhecido por *XP*, e é uma metodologia que enfatiza o desenvolvimento rápido do projeto visando e garantindo a satisfação do cliente, atendendo suas reais necessidades e favorecendo assim o cumprimento das metas e estimativas.

Teve seu início em 1996, quando Kent Beck, conhecido como um dos criadores dos padrões do desenvolvimento ágil colocou-os em prática no "Sistema de Compensação Abrangente da Chrysler" (*Chrysler Comprehensive Compensation System*).

O fator chave no desenvolvimento dos projetos é a forma de comunicação. Comunicação essa entre os desenvolvedores e o gerente do projeto. O princípio da comunicação é manter o melhor relacionamento possível entre clientes e desenvolvedores, com conversas pessoais a outros meios de comunicação.

A *XP* busca o máximo de valor a cada dia de trabalho da equipe para o seu cliente. Em um curto espaço de tempo o cliente terá um produto que possa ser utilizado, podendo aprender com o mesmo e reavaliar se o que foi desenvolvido é realmente o desejado.

3.2.1. Diretrizes e valores do *Extreme Programming*

As diretrizes da *XP* são seus valores e práticas, que definirão as atitudes das equipes e as principais prioridades da metodologia. Para uma empresa estar realmente utilizando o *Extreme Programming*, ela deve respeitar e utilizar todos os valores e práticas relacionados a seguir:

3.2.2. *Feedback*

O cliente aprende com o sistema que utiliza e com este aprendizado consegue reavaliar o produto recebido, com isso pode realimentar a equipe de desenvolvimento com as suas reais necessidades. Com o *feedbac*k, o cliente conduz o desenvolvimento do seu produto, estabelece prioridades e informa aquilo que é realmente importante.

O *feedback* dado pelo desenvolvedor ao cliente, onde o mesmo aponta riscos, estimativas e alternativas de desenvolvimento. Este retorno é o aprendizado do desenvolvedor sobre o que o cliente deseja.

3.2.3. Comunicação

Para que o *feedback* entre cliente e desenvolvedor possa ser efetuado com sucesso é necessário ter uma boa comunicação entre eles, sendo que essa comunicação ocorra da forma mais direta e eficaz possível, oferecendo agilidade aos assuntos tratados. Recomenda-se o contato direto (face a face) entre cliente e desenvolvedor, para evitar qualquer tipo de especulação ou mal entendido entre as partes e para que possíveis dúvidas possam ser resolvidas de imediato.

Além de sanar as dúvidas no desenvolvimento, o cliente deverá estar disponível para a equipe, ou mesmo presente no ambiente de trabalho da empresa. Isto fará com que o cliente entenda o sistema e enriquecerá os relacionamentos pessoais, criando um elo de parceria e confiança mútua.

3.2.4. Simplicidade

Para que o cliente possa aprender durante o projeto e consiga dar o *feedback* necessário à equipe, não basta apenas uma boa comunicação; é necessário que os desenvolvedores implementem da forma mais simples possível o que o cliente deseja.

3.2.5. Coragem

Por ser um processo de desenvolvimento novo e baseado em diversas premissas que contrariam o modelo tradicional, a *XP* exige que os desenvolvedores tenham coragem para: desenvolver *software* de forma incremental; manter o sistema simples; permitir que o cliente defina prioridades; fazer desenvolvedores trabalharem em pares; investir tempo em testes automatizados; estimar estórias na presença do cliente; expor código a todos os membros da equipe; integrar o sistema diversas vezes ao dia; adotar ritmo sustentável de desenvolvimento.

3.2.6. Cliente disponível ou presente

A *XP* trabalha com uma visão diferente do modelo tradicional em relação ao cliente. Ele sugere que o cliente esteja no dia-a-dia do projeto, acompanhando os passos dos desenvolvedores, onde a sua ausência representará sérios riscos ao projeto.

As funcionalidades do sistema são descritas brevemente em estórias em conjunto com os testes conceituais e serão estes os indicadores para uma boa implementação, sendo que o diálogo do cliente juntamente com os desenvolvedores se fará necessário para uma maior eficácia, o que acarretará a validação rapidamente, e a equipe recebendo o *feedback* necessário sobre a funcionalidade, criará ciclos rápidos e precisos.

3.2.7. Jogo de planejamento

A *XP* utiliza o planejamento para assegurar que a equipe de desenvolvimento esteja trabalhando naquilo que gere o máximo de valor para o cliente. Este planejamento é feito várias vezes durante o projeto, sendo o momento onde o cliente solicita as funcionalidades desejadas através de estórias, e onde a equipe estima o custo de cada estória e planeja as iterações.

3.2.8. *Stand up meeting*

O dia de trabalho de uma equipe *XP* sempre começa com um *stand up meeting*, que é uma reunião rápida pela manhã, com aproximadamente 20 minutos de duração e onde seus integrantes permaneçam preferencialmente em pé. Segundo estudos, uma reunião em pé é mais rápida, evita bate-papos paralelos e faz os integrantes irem diretamente ao assunto. Mais uma vez, ágil e simples.

A reunião tem por objetivo atualizar a equipe sobre o que foi implementado no dia anterior e trocar experiências das dificuldades enfrentadas. Neste momento também são decididas as estórias que serão implementadas no dia e em conjunto definir os responsáveis por cada uma delas.

3.2.9. Programação em par

A *XP* exige que todo e qualquer código implementado no projeto seja efetuado em dupla, chamada de programação em par. É uma técnica onde dois desenvolvedores trabalham no mesmo problema, ao mesmo tempo e no mesmo computador. Um deles é responsável pela digitação (condutor) e outro acompanhando o trabalho do parceiro (navegador).

Um dos grandes benefícios da programação em par é a troca de experiências e idéias entre os desenvolvedores.

3.2.10. *Refactoring*

No *refactoring* ao deparar com um código mal escrito ou pouco legível a *XP* prega que todo desenvolvedor, ao encontrar um código duplicado, pouco legível, mal codificado, sem padronização, lento, com código legado ou uso incorreto de outras implementações, tem por obrigação alterar este código deixando-o mais legível e simples, porém, esta alteração não pode mudar o comportamento do código em questão.

3.2.11. Desenvolvimento guiado por testes

Esta atividade deve ser encarada pelos desenvolvedores com extrema naturalidade, pois todo código implementado deve coexistir com um teste automatizado, assim garantindo o pleno funcionamento da nova funcionalidade.

É com base nestes testes automatizados que qualquer desenvolvedor terá coragem para alterar uma parte do código que não tenha sido implementada por ele, já que executando os testes automatizados poderá verificar instantaneamente se a modificação efetuada alterou o comportamento do sistema.

3.2.12. Código coletivo

No modelo tradicional de desenvolvimento, é comum dividir o projeto em partes e colocar responsáveis por cada uma destas partes, tornando apenas uma pessoa conhecedora daquela parte, mas isso pode acarretar sérios problemas ao projeto, uma vez que se aquela parte necessitar de alterações, apenas uma pessoa estará capacitada para alterá-la, com estas inúmeras alterações, esta pessoa pode se tornar um gargalo no projeto.

No caso da *XP*, não existe uma pessoa responsável por uma parte do código, sendo que cada desenvolvedor terá acesso a qualquer parte do sistema e terá a liberdade para alterá-la a qualquer momento e sem qualquer tipo de aviso. Esta prática tem como conseqüência um código revisado por diversas pessoas e caso algo não

esteja claro, com certeza será alterado por alguma pessoa (*refactoring*) para que o mesmo torne-se legível.

3.2.13. Padrões de desenvolvimento

Um dos valores da *XP* é a comunicação, e o código também é uma forma da equipe se comunicar. Uma forma clara de se comunicar através do código, é manter um padrão no projeto para que qualquer um possa rapidamente entender o código lido, sendo que a *XP* recomenda a adoção de um padrão desde o início do projeto, preferencialmente padrões utilizados pela comunidade da linguagem de desenvolvimento.

3.2.14. *Releases* curtos

Release é um conjunto de funcionalidades bem definidas e que representam algum valor para o cliente. Um projeto *XP* pode ter um ou mais *releases* no seu ciclo de desenvolvimento.

A *XP* recomenda que pequenos investimentos sejam efetuados de forma gradual e que busque grandes recompensas o mais rapidamente possível. Com a prática de *releases* curtos, a *XP* pretende dar o máximo de valor econômico ao cliente em curto espaço de tempo.

3.2.15. A Equipe *XP*

Em uma equipe de *XP* existem papéis a serem desempenhados por um ou mais desenvolvedores. São eles:

3.2.16. Gerente de projeto

Pessoa responsável pelos assuntos administrativos do projeto, mantendo um forte relacionamento com o cliente para que o mesmo participe das atividades do desenvolvimento, sendo responsável por filtrar assuntos não relevantes aos desenvolvedores e aspectos que só devam ser implementados em iterações futuras.

3.2.17. *Coach* (técnico)

Pessoa responsável pelas questões técnicas do projeto. Recomenda-se que esta pessoa seja a que possua maior conhecimento do processo de desenvolvimento, dos valores e práticas da *XP*. É de responsabilidade do *coach* verificar o comportamento da equipe frente ao processo *XP*, sinalizando os eventuais erros cometidos pela equipe.

3.2.18. Analista de teste

Pessoa responsável em garantir a qualidade do sistema através dos testes escritos. Deve ajudar o cliente a escrever os casos de testes e no final de cada iteração verificar se o *software* atende todos os casos de testes. Recomenda-se que esta pessoa não seja um desenvolvedor, para evitar uma visão tendenciosa já que não conhece o código desenvolvido.

3.2.19. Redator técnico

Responsável em documentar o sistema, evitando um forte trabalho dos desenvolvedores neste tipo de atividade, permitindo uma maior dedicação ao trabalho de codificação, deve estar em plena sintonia com os desenvolvedores e clientes para que a documentação reflita o código escrito e as regras de negócio atendidas pelo sistema.

3.2.20. Desenvolvedor

Essa pessoa seria a responsável em analisar, projetar e codificar o sistema. Na *XP* não existe diferença entre analista, projetista e programador, uma vez que em vários momentos do projeto o desenvolvedor estará exercendo alguma destas atividades. Nas práticas da *XP*, com a programação em par, a tendência é a equipe se torne uniforme em suas habilidades.

4. COMPARATIVO ENTRE AS METODOLOGIAS TRADICIONAIS E AGÉIS

Segundo COCKBURN (2001), a maioria das metodologias ágeis nada possuem nada de novo. O que as diferencia das metodologias tradicionais são o enfoque e os valores.

A ideia das metodologias ágeis é o enfoque nas pessoas e não em processos ou algoritmos como as tradicionais, a preocupação de gastar menos tempo com documentação e mais com a implementação.

Uma característica das metodologias ágeis é que elas são adaptativas ao invés de serem preditivas. Com isso, elas se adaptam a novos fatores decorrentes do desenvolvimento do projeto, ao invés de procurar analisar previamente tudo o que pode acontecer no decorrer do desenvolvimento.

Para ser realmente considerada ágil a metodologia deve aceitar a mudança ao invés de tentar prever o futuro. O problema não é a mudança em si, mesmo porque ela ocorrerá de qualquer forma. O problema é como receber, avaliar e responder às mudanças. Como exemplo, as aplicações baseadas em *Web* são melhores modeladas usando metodologias ágeis, uma vez que o ambiente *Web* é muito dinâmico.

As metodologias pesadas devem ser aplicadas apenas em situações em que os requisitos do *software* são estáveis e requisitos futuros são previsíveis. Estas situações são difíceis de serem atingidas, uma vez que os requisitos para o desenvolvimento de um *software* são mutáveis.

Com *feedback* constante nas metodologias ágeis, é possível adaptar rapidamente a eventuais mudanças nos requisitos, sendo que estas alterações nos requisitos das metodologias tradicionais são muitas vezes críticas e não apresentam meios de se adaptar rapidamente às mudanças. Outro ponto positivo das metodologias ágeis são as entregas constantes de partes operacionais do software, sendo que o cliente não precisará esperar muito para ver o *software* funcionando ao contrário das metodologias tradicionais.

Também podemos dizer que a integração e o teste contínuos também possibilitarão a melhora na qualidade do *software*.

4.1. METODOLOGIAS TRADICIONAIS X ÁGEIS

Tradicionais

Requisitos do sistema são estáveis.

Requisitos futuros são previsíveis.

Foco nos processos de desenvolvimento.

Ágeis

Requisitos são passíveis de alterações.

Refazer partes do código não apresenta alto custo.

As equipes são pequenas.

Datas de entrega do *software* são curtas.

Organizações com ambientes dinâmicos.

Foco nas pessoas.

Dedicar menos tempo com documentação.

Dedicar mais tempo com resolução de problemas de forma iterativa.

Hoje no mercado nota-se que as metodologias ágeis ainda não são utilizadas em larga escala, por causa da burocracia.

4.2. COMPARATIVO ENTRE AS METODOLOGIAS *SCRUM E EXTREME PROGRAMMING*

Atualmente tem-se uma variedade de metodologias, sendo *Scrum* e *Extreme Programming* (*XP*) as mais utilizadas. O *Scrum* e *XP* são muito práticos e nada burocráticos. Muitos acham que *Scrum* e *XP* conflitam no mundo *Agile,* pelo

contrário eles se complementam. *Scrum* atua no lado gerencial do projeto fazendo com que o produto especificado seja entregue ao final da iteração, já o *XP* atua nas práticas de engenharia de software a serem aplicadas no projeto como*: test-driven development, refactoring, pair programming, simple design*, etc. Além disso, *Scrum* pode ser aplicado além dos projetos de *software*, pois ele não especifica prática como *pair programming* ou *test driven development*, mas especifica como gerenciar os requisitos ou novos recursos solicitados pelo cliente.

As diferenças existem e são pequenas, mas são importantes.

- Equipes *Scrum* tipicamente trabalham em iterações (*sprints*) que vão de 2 semanas até 1 mês. As equipes *XP* também trabalham em iterações que vão de 1 a 2 semanas (LANUSSE 2010).

- Equipes *Scrum* não permitem alterações em seus sprints, o *XP* é mais flexível em relação a mudanças no *sprint*, desde que o time não tenha começado a trabalhar em um dos itens anteriormente definidos.

- Equipes *XP* trabalham em ordem de prioridade, recursos a serem desenvolvidos são priorizados pelo cliente e o time é obrigado a seguir esta ordem, já o *Scrum* o *product owner* define a prioridade do *product backlog*, mas o time irá definir a sequência de implementação dos itens do *backlog*.

- *Scrum* não estabelece quaisquer práticas de engenharia de software, *XP* sim.

5. APLICABILIDADE DO *SCRUM*

Neste capítulo será descrito a utilização do *Scrum* no desenvolvimento de um projeto real feito pela empresa de software VSM.

A VSM vem desenvolvendo *softwares* especializados no gerenciamento de farmácias e drogarias desde 1992. Tudo começou em Assis-SP, no interior de São Paulo, onde a primeira versão do STF (Sistema de Frente de Loja) e do OURO (Sistema de Retaguarda) foi desenvolvida e implantada em algumas drogarias da cidade. Com a evolução do sistema e a satisfação dos clientes, a VSM logo conseguiu centenas de clientes no interior de São Paulo e no norte do Paraná.

Em 2005 iniciou-se o desenvolvimento na nova geração do STF/OURO, para a plataforma Windows, que passou a ser chamado de FrontFarma e OuroFarma. Utilizando e adicionando recursos tecnológicos para facilitar ainda mais o dia a dia das farmácias, o FrontFarma/OuroFarma tornou-se rapidamente um dos principais *softwares* de gestão de drogarias do mercado, principalmente por ser totalmente *on-line* e multi-loja.

Em 2007 a VSM deu mais um passo importante, retomou um projeto antigo, o OuroFórmulas, e o desenvolveu totalmente integrado ao FrontFarma/OuroFarma, oferecendo assim uma solução completa e integrada para drogarias e farmácias de manipulação.

Atualmente a VSM está presente nos estados de São Paulo, Paraná, Mato Grosso do Sul, Mato Grosso, Rio Grande do Sul, Piauí, Minas Gerais e Goiás.

Um dos objetivos da empresa é continuar melhorando seus produtos, adicionando novos recursos, atendendo a legislação vigente e realizando as homologações necessárias, para que possa aumentar ainda mais a satisfação de seus clientes.

5.1. O PROJETO

O projeto em questão se trata de uma homologação PAF-EFC, instituído pelo convenio ICMS 15/08. O PAF-ECF é Programa Aplicativo Fiscal que faz interface com o Emissor de Cupom Fiscal, que serve para padronizar a forma como os sistemas de automação enviam informações ao FISCO. Por isso, toda empresa que

hoje tenha obrigação de emitir cupom fiscal deve ter o seu software homologado junto a Secretaria da Fazenda, para que possa produzir um software dentro da legislação nacional. A homologação PAF-ECF é obrigatória em todos os estados do território nacional, sendo assim por a VSM emitir cupons fiscais, teria de adequar seu software conforme os requisitos solicitados pelo convenio ICMS 15/08.

5.2. OS PERSONAGENS

Na VSM ficou bem definido quem deveria assumir cada papel do time. O gerente da área de desenvolvimento, no caso Adriano Alves Dornelas, se tornou o *Product Owner* e também o *Scrum Master,* por ser a pessoa com maior conhecimento dobre a metodologia *Scrum,* e também por ser o responsável por todo o desenvolvimento do projeto.

O Time que irá trabalhar com um foco de 70% no desenvolvimento do *software*, foi formado por quatro integrantes, sendo todos do departamento de desenvolvimento, onde todos os quatro são desenvolvedores.

5.3. O PLANEJAMENTO

O *Product Owner* que como dito anteriormente também tinha o papel de *Scrum Master,* primeiramente fez um levantamento de tudo o que seria preciso para o processo de desenvolvimento, gerando o *Product Backlog,* ou seja, uma lista contendo todas as tarefas que ele deseja que sejam desenvolvidas.

Após gerar o *Product Backlog, o Product Owner* definiu a estimativa de cada tarefa.

A Figura 2 mostra uma das estórias do *Product Backlog:*

CONFERIR PRODUTOS E LOTES DA VENDA NO CAIXA	Importância
	480
	Estimativa
Como Testar:	**1,5**
Ativar essa configuração; Realizar pré-venda; puxar no caixa e verificar se irá abrir tela para conferir produtos pelo códigode barras e lotes; em caso de venda direta no caixa deverá abrir tela da mesma maneira; permitir liberar produtos com senha, porém deve registrar no LOG qual venda e Produtos;	
Notas: Verificar quando produto não tem código de barras ou quando está marcado no cadastro que permite venda sem código de barras.	ID:34

Figura 2. Exemplo de como foi definida a estória do *Product Backlog*

Nota-se que para uma melhor organização a VSM definiu as estórias do *Product Backlog* em formatos de ficha, onde estão disponíveis informações importantes como: título, descrição de como fazer os testes necessários, a estimativa, uma ID para um determinado controle e a importância, onde cada tarefa recebe um determinado valor por sua prioridade de desenvolvimento, seguindo as seguintes regras:

- Prioridade imediata - importância de 400 a 499;
- Prioridade urgente - importância de 300 a 399;
- Prioridade alta - importância de 200 a 299;
- Prioridade média - importância de 100 a 199;
- Prioridade baixa - importância de 0 a 99.

Depois de ter sido concluído pelo *Scrum Master* todo o levantamento de requisitos necessários para o *Product Backlog,* se inicia a *Planning Meeting,* que seria a reunião de planejamento, onde a finalidade era criar um *Sprint* bem detalhado para que toda a equipe possa trabalhar consciente no desenvolvimento do projeto.

Mas antes de se iniciar o *Sprint,* é mostrado o grau de comprometimento de cada integrante do time, sendo assim foi definido da seguinte forma:

- Programador 1: comprometimento 100%;
- Programador 2: comprometimento 100%;
- Programador 3: comprometimento 100%;
- Programador 4: comprometimento 100%;
- Testador: comprometimento 50%; (Por questões da empresa, os testes eram realizados fora do *Sprint*).

É importante observar, que esta é uma informação muito importante para que se possa ter uma idéia de quanto tempo seria o prazo para o *Sprint.*

A próxima etapa é decidir quantas estórias do *Product Backlog* farão parte do *Sprint Backlog* e também o tempo de duração do *Sprint,* onde a estimativa foi calculada da seguinte forma: Soma-se todas as estimativas das tarefas do *Product Baklog,* dividido pelo foco de desenvolvimento, dividido pela quantidade de programadores, resultando em dezoito dias de *Sprint* .

Depois de todas estas etapas concluídas, o *Scrum Master* juntamente com o time, definiu o *Sprint Backlog* da seguinte forma:

Instalar MySQL 5.1 **Estimativa:0 ID: 71**	Cadastrar formulários e campo para o manual do Form **Estimativa:1,5 ID: 70**	Alterar integração com a AmplaCard (SisCred/Comunix) **Estimativa:4 ID: 40**	Conferir produtos e lotes da venda no caixa **Estimativa:1,5 ID: 34**
Exportar dados de venda CasBrasil (igual MultiDrogas **Estimativa:0,5 ID: 51**	Melhora e automatizar atualização de preços **Estimativa:3 ID: 50**	Gravar dados de ECF offline e transmitir para servidor central **Estimativa:3,5 ID: 53**	Registrar todos os documentos emitidos pelo ECF **Estimativa:9 ID: 4**
Gerar arquivo de movimento por ECF **Estimativa:3,5 ID 5**	Controlar Pré-Vendas **Estimativa:5 ID: 6**	Controlar DAVs (Documento Auxiliar de Vendas) **Estimativa:5 ID: 7**	Imprimir dados da entrega no cupom fiscal **Estimativa:1 ID: 73**
Menu Fiscal **Estimativa:6 ID: 8**	Validar se ReduçãoZ foi Emitida automaticamente **Estimativa:1 ID: 74**	Imprimir MD5 no Cupom Fiscal **Estimativa:0,5 ID: 10**	Consultar tabela de produtos **Estimativa:0,5 ID: 9**
Não permitir configurar qual ECF, somente a porta (Requisito 22) **Estimativa:1,25 ID: 11**	Armazenar GT e validar n° de série e GT ao entrar no sistema **Estimativa:1 ID: 12**	Permitir somente consultas quando não for possível utilizar a ECF **Estimativa:1 ID: 13**	Consultar movimentação do Mês (Requisito 26) **Estimativa:2 ID: 14**

Tabela 1. *Sprint Blacklog*

É importante observar que o *Sprint* trouxe de forma bem detalhada e abreviada cada item do *Product Backlog,* sua estimativa e ID.

Mais uma etapa definida, também já foi marcada a data do inicio do *Sprint,* na sala de programação. O *Scrum máster,* juntamente com o time decidiu que o *Daily Scrum Meeting* (reunião diária) seria de 10 a 15 minutos iniciando-se as 08h10 na sala de desenvolvimento. Após todas estas etapas concluídas, todos já estavam preparados para iniciarem o *Sprint*.

5.4. *SPRINT* EM DESENVOLVIMENTO

O *Scrum Master* passou para cada integrante do time, o que deveria ser feito no *Sprint.*

5.4.1. REUNIÃO DIÁRIA

Como combinado, o primeiro dia do *Daily Scrum Meeting* começou às 8h10 na sala de programação. Foi comentado por cada membro do time, quais as atividades que cada um iria fazer neste primeiro dia, após esta conversa, todos seguiram para seu trabalho.

Apos concluído o primeiro dia de *Sprint,* no segundo dia todos do time citaram as tarefas feitas no dia anterior, e também o que iria ser feito neste novo dia de trabalho. O time somou todos os pontos das tarefas realizadas no dia anterior e acrescentou no gráfico de *BurnDown.*

É importante ressaltar, que o gráfico de *BurnDown* é uma ferramenta do *Scrum.* O gráfico de *BurnDown* é uma forma visual e rápida de enxergar o status atual do projeto. Ele possui uma estrutura simples, onde:

- Eixo x: representa os dias do *Sprint*;
- Eixo y: representa o restante do trabalho.

Abaixo as figuras 3 e 4 mostram a atualização e evolução do gráfico de *BurnDown*:

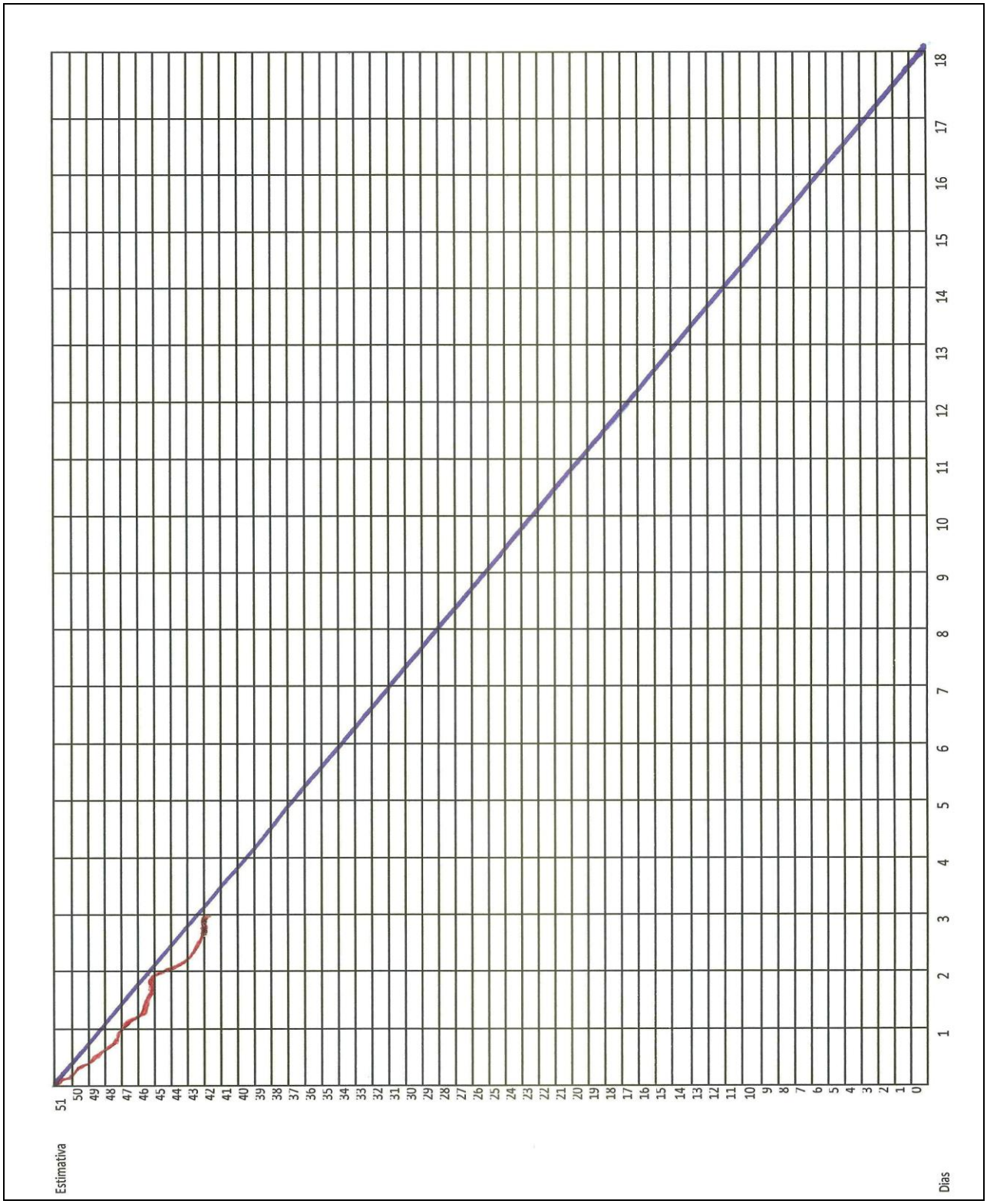

Figura 3. *Gráfico de BurnDown* (3° dia de *Sprint,* 9,0 pontos de estimativa concluídos).

Nota-se que o eixo ‘y’ traz a soma das estimativas (51 pontos) e o eixo ‘x’ traz os dias do *Sprint.*

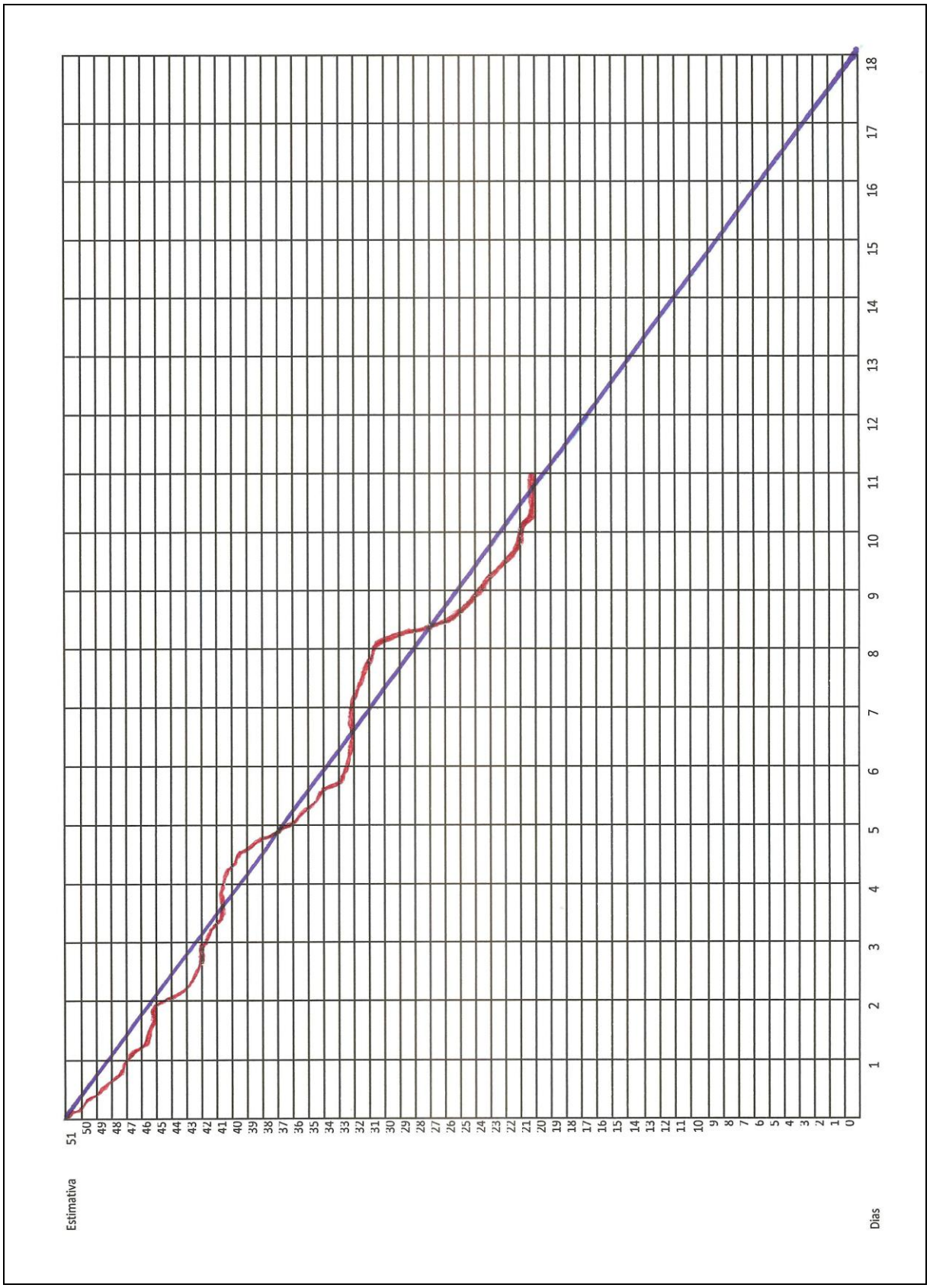

Figura 4. Gráfico de BurnDown (11° dia de Sprint, 30 pontos de estimativa concluídos).

5.4.2. DIFICULDADES NO CAMINHO

Ao longo do desenvolvimento do *software* HOMOLOGAÇÃO PAF-ECF surgiram algumas dúvidas, a respeito de regras e legislação. Por não ser muito claro, devido ao lado prático, o time juntamente com o *Scrum Master* entraram em contato com os órgãos que faziam a homologação para esclarecer dúvidas.

Por ser um dos primeiros *Sprints* realizados pela VSM, o *Scrum Máster* estava contente com os resultados do time, mas mesmo assim, percebeu que deveria ser dedicado mais tempo ao planejamento, para que se pudesse ter uma estimativa mais precisa do tempo das tarefas e do *Sprint.*

Mas mesmo com pequenos obstáculos, o time não se abalou e estava dentro do tempo previsto, e cada vez mais perto do fim.

5.5. REUNIÃO DE APRESENTAÇÃO

Sprint finalizado. O *Scrum Master* estava satisfeito, já que o time foi muito competente e conseguiu desenvolver tudo o que havia sido planejado, cumprindo a regras e prazos.

Depois da conclusão do *Sprint,* é feito a *Sprint Review* (Revisão), onde todo o time se reúne juntamente com o *Scrum Master,* e também todos aqueles que estejam interessados a assistir a apresentação do *Sprint.*

Na reunião, foi apresentado a todos aqueles presentes, o gráfico de *BurnDown* finalizado.

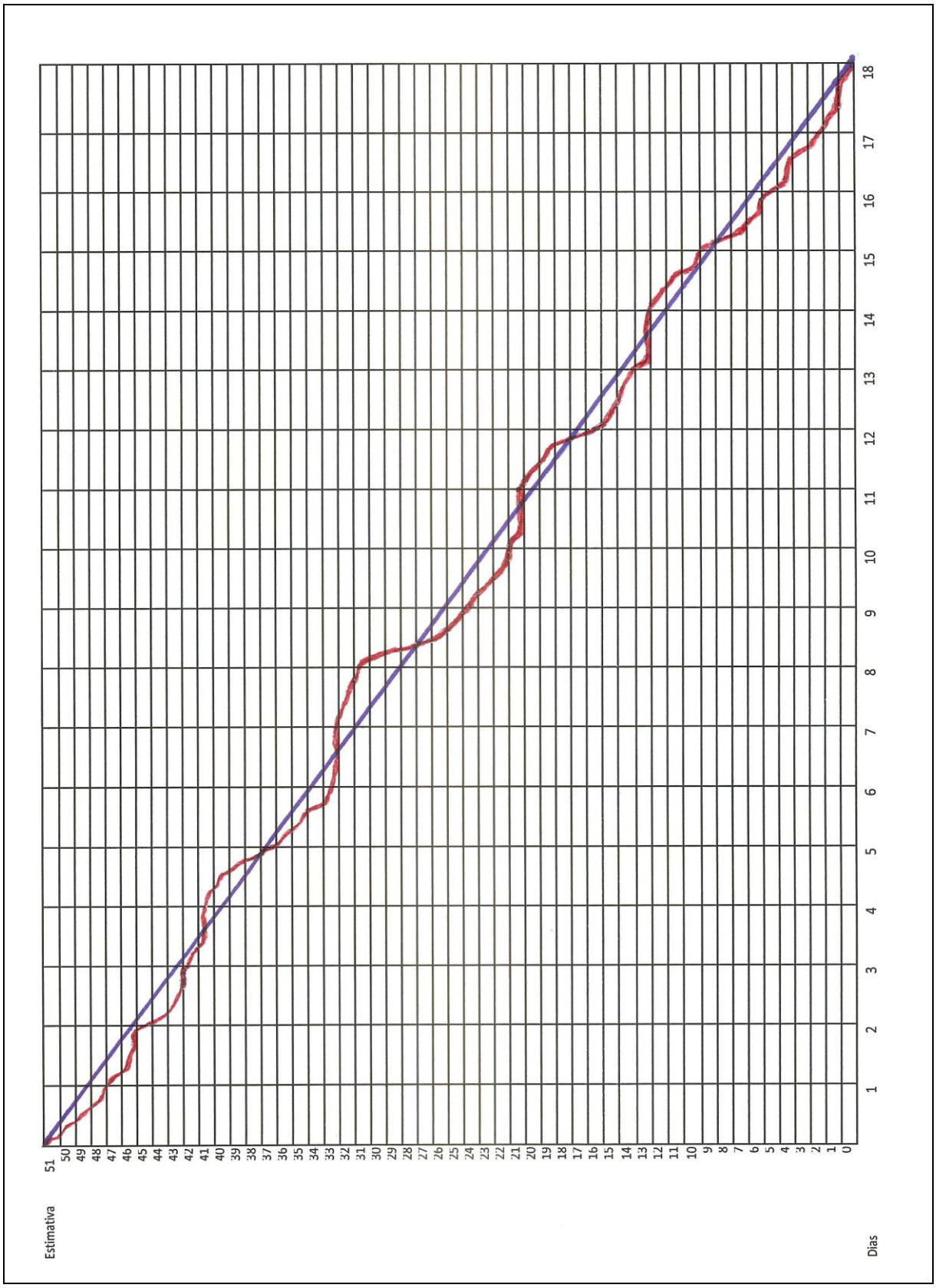

Figura 5. Gráfico de BurnDown finalizado (18° dia de Sprint, 51 pontos de estimativa concluídos).

5.6. RETROSPECTIVA

Para finalizar, é feita uma ultima reunião chamada de Retrospectiva. Nesta reunião cada membro do time cita suas dificuldades, e o que esperariam aperfeiçoar para melhorar em um próximo *Sprint,* não deixando de lado a qualidade e o espírito de equipe criado neste *Sprint* que se finalizou.

6. CONCLUSÃO

Com a competitividade na área de desenvolvimento de software aumentando cada vez mais, faz com que as empresas busquem um diferencial no desenvolvimento para seus clientes.

Neste trabalho foram apresentados conceitos e noções sobre as Metodologias Ágeis de desenvolvimento de software *Scrum* e *Extreme Programming*.

Com base na empresa VSM, o *Scrum* foi à metodologia utilizada no estudo de caso, obtendo-se os seguintes resultados.

O *Scrum* organizou e gerenciou todo o desenvolvimento de software; funciona perfeitamente, desde cada pessoa do time assuma seu papel; conseguiu-se desenvolver o *software* nos prazos programados pela empresa.

Além disso, o *Scrum* contribuiu para outros diversos fatores:

- Melhorar a integração da equipe, já que todos têm o mesmo objetivo, eles se comunicam mais, trocam mais ideias;
- Aumentar a motivação da equipe no sentido de atingir uma meta que é a conclusão do *Sprint;*
- Melhorar a qualidade do que é desenvolvido devido a várias pessoas estarem pensando juntas no mesmo assunto;
- Organizar a lista de prioridades;
- Melhorar o processo continuamente através das reuniões retrospectivas.

Porém, quando a empresa não utilizava o *Scrum* ela se deparava com pequenos problemas como:

- Somente o programador que desenvolvia determinado módulo conhecia totalmente como aquilo funcionava, causando uma dependência dele para futuras alterações;
- Os programadores terminavam as suas tarefas em períodos diferentes dificultando o fechamento das versões para serem liberadas para os clientes;
- Encontravam-se dificuldades em definir prazos;
- O responsável pela equipe tinha que pensar individualmente no que cada um iria fazer.

Fica então a dica, as metodologias ágeis vieram para somar, agora cabe a cada desenvolvedor ou gerente colocar em prática aquilo que pode ser um diferencial para sua empresa.

REFERÊNCIAS

COCKBURN, A. **"Agile Software Development: The Business of Innovation"**, IEEE Computer, Sept., pp. 120-122, 2001

FAGUNDES, Priscila B**. Framework para comparação e análise de métodos ágeis.** Dissertação de Mestrado, Universidade Federal de Santa Catarina, Florianópolis, 2005

FILHO, Edes Garcia da Costa. PENTEADO, Rosângela. SILVA, Júnia Coutinho Anacleto. BRAGA, Rosana Teresinha Vaccare. **Padrões e Métodos Ágeis**: **agilidade no processo de desenvolvimento**. Disponível em: http://edges.com.br/files/sugarloaf_2005.pdf

KUHN, Giovane Roslindo. PAMPLONA, Vitor Fernando. **Apresentando XP**. Encante seus clientes com Extreme Programming.
Disponível em: http://www.javafree.org/content/view.jf?idContent=5

LANUSSE, Andreano . **Metodologias ágeis, Scrum ou XP**. Disponivel em: http://www.andreanolanusse.com/pt/metodologias-ageis-scrum-ou-extreme-programming-xp/

LIBARDI, Paula L. O. BARBOSA, Vladimir. **Métodos Ágeis**. Disponível em: http://www.ft.unicamp.br/liag/Gerenciamento/monografias/monogafia_metodos_ageis.pdf

LUDVIG, Diogo. DAVSON REINERT, Jonatas. **Estudo do uso de Metodologias Ágeis no Desenvolvimento de uma Aplicação de Governo Eletrônico**
Disponível em: http://projetos.inf.ufsc.br/arquivos_projetos/projeto_589/Artigo_Diogo_Jonatas.pdf

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