Perquirere, 11 (1): 113-129, jul. 2014

http://perquirere.unipam.edu.br

Estudo de caso: aplicação das metodologias ágeis de desenvolvimento: *Scrum* e XP no desenvolvimento do sistema Unidisciplina

Case study: application of agile development methodologies: Scrum and XP development of Unidisciplina System

Mislene Dalila da Silva
Especialista em Engenharia de Software (UNIPAM).
E-mail: mislene@unipam.edu.br

Fernando Corrêa de Mello Junior
Professor orientador (UNIPAM).
E-mail: fernandocmjr@unipam.edu.br

Resumo: Este artigo constitui na análise das metodologias ágeis de desenvolvimento de software XP (Programação Extrema) e Scrum aplicadas no Projeto Unidisciplina. Com a finalidade de identificar os benefícios que as metodologias oferecem no desenvolvimento de sistemas, são analisados seus pontos fortes no gerenciamento da equipe, atividades, requisitos, execução e entrega. O objetivo foi identificar os artefatos que as metodologias geraram no decorrer do desenvolvimento e onde as mesmas podem ser aplicadas facilitando e apoiando o processo de desenvolvimento de um software.
Palavras-chave: Metodologias Ágeis. Xp. Scrum.

Abstract: This article is the analysis of agile development methodologies XP (Extreme Programming) and Scrum software applied in the Unidisciplina Project. In order to identify the benefits that methodologies offer in the systems development, its strengths are analyzed in team management, in activities, in requirements, in execution and in delivery. The purpose of this paper was to identify artifacts that those methodologies generated during their development and where they can be applied to facilitate and support the process of software development.
Keywords: Agile Methodologies. Xp. Scrum.

1 INTRODUÇÃO

Metodologias de desenvolvimento de software são necessárias para esclarecer a importância do processo de desenvolvimento bem como explicar suas aplicabilidades e a participação das pessoas.

As ações e reações de equipes em projetos de softwares geram informações e conhecimentos sobre o produto e as mesmas utilizam recursos e ferramentas para auxiliar no desenvolvimento. Desde a história da evolução dos processos, houve

constantes aperfeiçoamentos em Metodologias de Desenvolvimento, com ferramentas automatizadas que possibilitam o desenvolvimento de um produto com rapidez e com qualidade.

Essas metodologias são chamadas de métodos ágeis de desenvolvimento e se originaram pelo manifesto Ágil, que tem como conceito a priorização dos indivíduos, equipes e interações, trabalhando com entregas contínuas e progressivas. As alterações nos requisitos consistem em formas de melhorar o software e são diagnosticadas como positivas. Os envolvidos no projeto, como clientes, desenvolvedores e gerentes de projeto, devem trabalhar sincronizados, em conjunto e motivados durante todo ciclo dele.

A simplicidade é outro ponto essencial para o desenvolvimento ágil. Ela remete à progressividade da equipe, de forma que todos abstraiam claramente as tarefas que estão sendo realizadas. Desenvolver códigos complexos e utilizar linguagens de difícil entendimento não é bem aceito para os padrões ágeis.

Os artefatos e ferramentas que essas metodologias geram apoiam todos os *Stakeholders* do projeto, pois são disponibilizados cartões de histórias que auxiliam no levantamento e na priorização dos requisitos a serem desenvolvidos. No decorrer do desenvolvimento, são gerados gráficos com estimativas de tempo e complexidade das funcionalidades que irão ser realizadas, quadro de tarefas para identificar o que cada membro da equipe está desenvolvendo e em qual nível está o projeto, além de realização de reuniões diárias para identificar as dificuldades da equipe e o andamento do projeto.

A finalidade deste artigo é analisar as vantagens da utilização das metodologias Programação Extrema e *Scrum* utilizadas pareadas no processo de desenvolvimento de sistemas que irão servir de repositórios de conhecimentos e referência para o início de um projeto de software.

2 REFERENCIAL TEÓRICO

2.1 ENGENHARIA DE SOFTWARE

A engenharia de software engloba todas as etapas de desenvolvimento do projeto para que seja garantida a solução com qualidade. É a base para o ciclo de vida da construção do software, desde as fases iniciais, especificações de requisitos do sistema, até sua implantação e manutenção (SOMMERVILLE, 2007).

De acordo com Rezende (2005), engenharia de software se caracteriza por um desenvolvimento de software sob medida, que atenda às necessidades dos usuários, adeque aos requisitos solicitados e respeite os prazos e orçamentos estimados do projeto. É uma ferramenta que proporciona soluções no desenvolvimento do software, indicando um roteiro de regras que pode utilizar diversas técnicas e artefatos para apoiar no desenvolvimento.

Como no processo de produção de um software estão envolvidas muitas etapas e atividades, as quais, às vezes, podem ser esquecidas no decorrer do desenvolvimento do projeto, a vantagem de utilizar o modelo de processo de desenvolvimento é que irá

fornecer estabilidade, controle, organização e qualidade, tanto para os envolvidos no projeto quanto para o próprio projeto (PRESSMAN, 2006).

2.2 METODOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE

Desenvolver um software é uma atividade que pode ser difícil e oferecer riscos. Entre os maiores riscos estão: gastos que superam o orçamento, tempo que excede o estipulado no cronograma, funcionalidades desenvolvidas que não atendem as necessidades do usuário, baixa qualidade dos sistemas e cancelamento do projeto por inviabilidade (DevMedia, 2013).

Metodologia de desenvolvimento de software pode ser conceituada como um conjunto de atividades e resultados associados que ajudam na produção de um software.

2.2.1 Desenvolvimento Ágil

Para Teles (2006), processos de desenvolvimento de software que são chamados de ágeis compartilham um conjunto de premissas fundamentais que diferem das adotadas pelos processos tradicionais. As atividades não são executadas de forma linear, não são aplicadas regras de início e fim e não são empregados passos exatos.

O aprendizado vem do *feedback* que o projeto fornece para seu criador, como se houvesse um diálogo entre o desenvolvedor e o projeto e vice-versa. À medida que é desenvolvido, o software mostra o que precisa ser aperfeiçoado. O trabalho se caracteriza por aprendizado permanente, o que leva a melhorias e torna o produto final adequado para o seu público.

O desenvolvimento ágil se baseia na premissa de que o cliente aprende ao longo do desenvolvimento, à medida que é capaz de manipular o sistema. Ele percebe os detalhes, compreende as dificuldades, percebe novas possibilidades e, consequentemente, solicita novas alterações para que o software se aproxime, ao máximo, daquilo que ele acredita que poderá resolver os seus problemas (TELES, 2006).

2.2.2 Programação Extrema

Programação Extrema ou XP é uma metodologia de desenvolvimento que se baseia fortemente em simplicidade, comunicação e *feedback*. Trabalha com um time reduzido de programadores e projetos pequenos, com um curto período de entrega (BECK, 2004).

Segundo Teles (2006), o *feedback* é a realimentação que o cliente fornece à equipe de desenvolvimento quando aprende algo novo sobre o sistema, quando aprende mais sobre os requisitos ou quando aprende mais sobre a forma de implementação. A partir deste aprendizado, surgem novas ideias sobre o que deve ser feito, além de como e quando deve ser feito.

É importante a simplicidade das ações de cada membro da equipe, para que, logo após a execução de uma ação, possa obter um *feedback* sobre essa ação e faça

pequenos ajustes, caso estes sejam necessários. Assim, será exigido o mínimo de esforço para que o pedido do cliente seja atendido e possa ser validado (TELES, 2006).

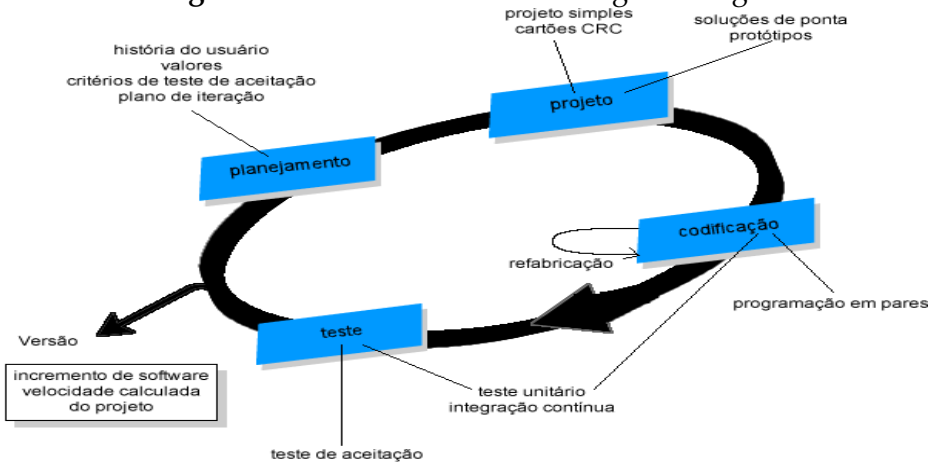
O XP é um processo de desenvolvimento de software baseado em diversas premissas que contrariam os processos tradicionais de desenvolvimento. Portanto, a aplicação do XP exige que a equipe de desenvolvimento seja capaz de:

- Desenvolver o software de forma incremental;
- Manter o sistema simples;
- Permitir que o cliente priorize as funcionalidades;
- Fazer os desenvolvedores trabalharem em par;
- Investir tempo em *refactoring*;
- Investir tempo em testes automatizados;
- Estimar histórias na presença do cliente;
- Expor o código a todos os membros da equipe;
- Integrar os sistemas diversas vezes ao dia;
- Adotar um ritmo sustentável;
- Abrir mão de documentação que serve como defesa;
- Propor contratos de escopo varável;
- Propor a adoção de um processo novo.

Segundo Pressman (2006), o modelo XP estabelece um conjunto de regras e contexto que ocorre em quatro atividades, conforme descritas na Figura 1.

Na fase de planejamento, os usuários descrevem um conjunto de história (metáforas) e exemplificam as características e funcionalidades do sistema. O cliente tem total liberdade de mudar e acrescentar elementos na história. Já na fase de projeto, o XP estabelece rigorosamente a simplicidade, nada mais apropriado que trabalhar com artefatos simples. Defende o uso de cartões CRC (Cartão Classe-Responsabilidade-Colaboração) que encoraja discussões entre os desenvolvedores. Ainda na fase do projeto, se diagnosticado um problema complexo neste, recomenda-se a criação imediata de um protótipo daquela fase. O XP encoraja mudanças e refatoração, caso haja necessidade. Na fase de codificação, após o levantamento da história, será feita uma série de testes unitários, o desenvolvedor focará no que será implementado na codificação e passará um breve *feedback* para o cliente.

Figura 1 – O Processo Extreme Programming



Fonte: Adaptado de Pressman (2006, p. 64)

Na fase de codificação, os programadores devem programar em pares, segundo Pressman (2006), garantindo qualidade em tempo real. À medida que eles finalizam o trabalho, o código é integrado aos outros, que é uma responsabilidade da equipe de integração.

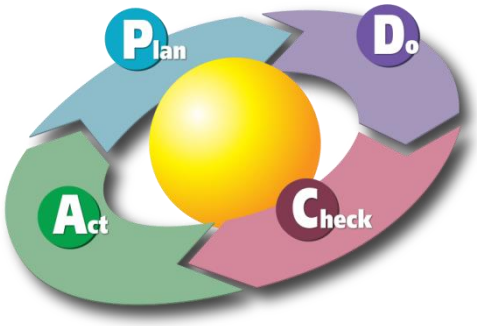
Por fim, a etapa de testes, que fornece segurança e progresso, garante equilíbrio e minimiza o problema, reduzindo o retrabalho dos programadores (PRESSMAN, 2006).

2.2.3 *Scrum*

O *framework Scrum* é uma metodologia ágil de desenvolvimento para gestão e planejamento de projetos de software. Foi elaborada por Jeff Sutherland e por sua equipe, no início da década de 90, e tinha como base o processo de produção de automóveis de indústrias japonesas (PRESSMAN, 2006).

O *Scrum* se baseia na ferramenta PDCA (Planejar-Executar-Verificar-Ajustar) ou ciclo de *Deming*, que aplica o desenvolvimento com foco na melhoria contínua. O ciclo é divido em quatro fases, como ilustrado na Figura 2:

Figura 2 – Ciclo PDCA



Fonte: DevMedia

- P – *Plan*: Planeje;
- D – *Do*: Faça;
- C – *Check*: Avalie;
- A – *Act*: Aja de forma corretiva.

O ciclo PDCA sugere que, no início, se faça o planejamento com diretrizes e objetivos claramente definidos, por isso a necessidade de saber o que deve ser feito de acordo com a sua prioridade e tempo de realização. O segundo passo é fazer o que foi planejado, a implementação da execução do plano. Após isso, é necessário avaliar o que foi realizado, buscando erros e acertos de execução. Por fim, é realizada a identificação e correção das falhas.

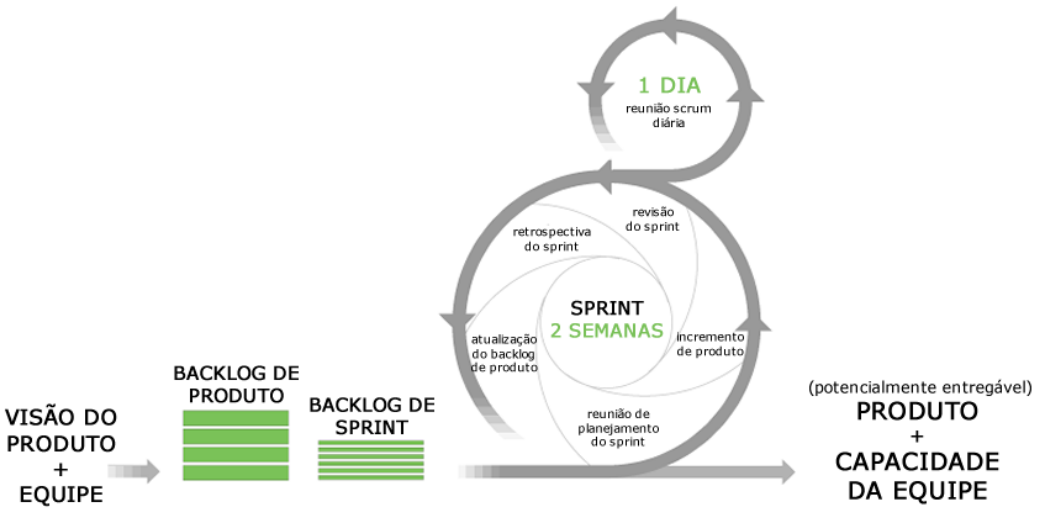
Os processos do *Scrum* agrupam as atividades de escopo: requisitos, análise, projeto, evolução e entrega. Em cada atividade de arcabouço, as tarefas de trabalho ocorrem dentro de um padrão de processo chamado de *Sprint*. O *Sprint* representa um *Time Box* no qual possui um conjunto de atividades que devem ser executadas. São atividades iterativas, unidades de trabalho necessárias para serem desenvolvidas e normalmente devem ser cumpridas em um intervalo de tempo predefinido (de 2 a 4 semanas).

O *Scrum* trabalha com grupos altamente gerenciáveis, divididos em três papéis, que são:

- *Product Owner*: é o receptor do projeto, defende os interesses do cliente, é responsável por priorizar os itens do *Product Backlog*, motivar a equipe e gerenciar novos requisitos.
- *Scrum Master*: é responsável por remover impedimentos, garantir que a equipe não sofra interferências e ajudar o *Product Owner* na priorização dos requisitos;
- Time *Scrum*: é responsável por estimar o tempo e os valores de desenvolvimento das tarefas do projeto, definir metas para produzir produtos de qualidade e valor para o cliente. O time deve conter de 5 a 9 pessoas (KNINBERG, 2007).

A Figura 3 representa o ciclo de vida do desenvolvimento de software com o *Scru.* Pode-se observar, na visão inicial do produto, que o *Product Owner* e a equipe definem as histórias que serão desenvolvidas. O *Product Owner* é responsável por priorizar as histórias do *Product Backlog* que serão repassadas para *Sprint*, originando o *Backlog* de *Sprint*, que são as listas de tarefas. Definido *o Backlog* de *Sprint*, a equipe se reúne em uma reunião chamada *Sprint Plane Meeting* (reunião de Planejamento da *Sprint*), em que se estima quanto tempo será necessário para completar a tarefa e sua dificuldade (KNINBERG, 2007).

Figura 3 – Ciclo de vida da Metodologia *Scrum*



Fonte: Blog Semeru

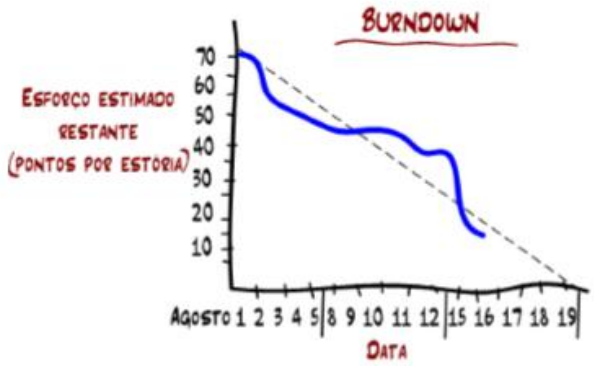
A seguir, o time trabalha no desenvolvimento do produto. Finalizada a etapa de incremento do produto, devidamente testado e integrado ao sistema, a equipe realiza a revisão do *Sprint*, expondo o que foi desenvolvido durante o *Sprint*, e apresenta as novas funcionalidades integradas. O *Product Owner* testa e examina se os itens atendem suas expectativas e determina se a meta do *Sprint* foi alcançada (COHN, 2011).

A próxima etapa, como pode ser observada no espiral, é a Retrospectiva da *Sprint*, na qual são avaliados os pontos positivos e negativos e o que pode ser melhorado. A finalidade dessa reunião é buscar a melhoria contínua do desempenho da equipe. Feito isso, o *Backlog* do Produto é atualizado e o ciclo reiniciado. Ao topo do círculo, tem um círculo menor que representa a reunião diária de 15 minutos, realizada pela equipe sempre no mesmo horário e local, momento em que cada membro pode expor o que foi feito e ainda pretende fazer e se está tendo alguma dificuldade.

Durante o desenvolvimento *Scrum*, faz-se uso de alguns artefatos para apoiar o desenvolvimento do projeto. Seguem alguns:

- **Gráfico de Burndown ou Gráfico de Consumo**: oferece uma estatística do progresso e velocidade da equipe. Pode-se observar, na Figura 4, que, no dia 1º de agosto, a equipe estimou que havia aproximadamente 70 pontos por história de trabalho a serem feitos; em 16 de agosto, a equipe estimou que havia aproximadamente 15 pontos por histórias de trabalho a serem desenvolvidos. A linha de tendência tracejada mostra que eles estão aproximadamente dentro do prazo, ou seja, conseguiram cumprir tudo até o final da *Sprint*. O *Scrum Master* é responsável por garantir que a equipe aja quando surgirem sinais de alarme (KNINBERG, 2007).

Figura 4 – Exemplo de Gráfico *Burndown*



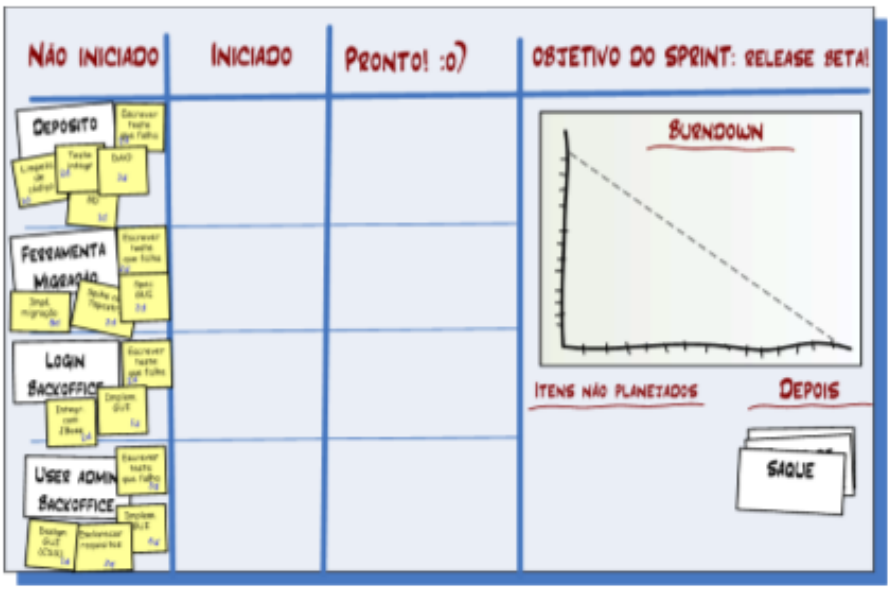
Fonte: Livro XP e *Scrum* Direto das Trincheiras

- **Planning Poker:** realizar a estimativa de cada *Sprint Backlog* é uma tarefa da equipe por meio das histórias. Essas, às vezes, podem ser difíceis de serem definidas, pois envolvem diversas pessoas com distintas características, tais como design de interface de usuário, codificação, testes etc. Para promover uma

estimativa, o membro da equipe precisa de algum tipo de conhecimento. Geralmente, conseguir chegar a um entendimento com todo o time por meio de conversação pode gerar conflitos e discrepâncias. Uma técnica produtiva para evitar esse tipo de situação pode ser resolvida através do *Planning Poker*. Ele funciona da seguinte forma: cada membro da equipe recebe um baralho com 13 cartas; quando uma história é estimada, cada membro escolhe sua carta, identificando sua estimativa de tempo e dificuldade, a carta é colocada virada para baixo sobre a mesa, quando todos tiverem feito sua escolha, as mesmas são reveladas simultaneamente, de forma que cada membro é forçado a pensar em sua estimativa, não sendo influenciado pelos outros elementos (KNINBERG, 2007).

- **Quadro de Tarefas:** seu objetivo é organizar, de forma cronológica e visível, o desenvolvimento das funcionalidades, representado na Figura 5. Na primeira coluna, em "Não Iniciado" ou "A Fazer", são colocadas as tarefas que ainda estão na fila para serem realizadas; na segunda coluna, em "Iniciado" ou "Em Execução", são identificadas as tarefas que estão sendo realizadas por algum membro da equipe; na terceira coluna, em "Pronto", são colocadas as tarefas finalizadas por algum membro da equipe, nas quais ninguém mais pode mexer. Ao lado do quadro de tarefas, como pode ser observado na Figura 5, possui o objetivo da *Sprint*, com o Gráfico de *Burndown*, onde é tracejada uma linha a cada dia depois da reunião diária. E ainda possuem os itens não planejados e os que virão depois (KNINBERG, 2007).

Figura 5 – Quadro de Tarefas



Fonte: Livro XP e *Scrum* Direto das Trincheiras

3 METODOLOGIA DE TRABALHO

O estudo de caso foi desenvolvido baseado nas Metodologias Ágeis de Desenvolvimento, Programação Extrema e *Framework Scrum*. Foi desenvolvido seguindo os seguintes tópicos:

- Escolha dos processos a serem analisados;
- Análise teórica de cada processo;
- Definição do sistema a ser desenvolvido;
- Utilização do Processo XP e *Scrum* para desenvolvimento do Sistema e análise para o projeto.
- Planejamento para aplicação de cada processo no sistema definido e desenvolvimento;
- Execução do desenvolvimento do sistema, com base nas características dos processos;
- Avaliação do resultado do desenvolvimento;

Para dar sequência ao desenvolvimento do Sistema e análise na utilização dos Processos XP e *Scrum*, foram descritas as fases de trabalhos, atividades, ferramentas e artefatos utilizados.

Na fase de planejamento, foram levantados os requisitos do sistema através do artefato Cartão de Histórias que o XP oferece, sendo desenvolvido um cartão de história para cada requisito. Uma vez definidas as histórias, foi preenchido o *Product Backlog*, utilizando a ferramenta de gerenciamento *Scrum Half*.

Definido o *Product Backlog* e as prioridades, foi identificado para cada funcionalidade seu valor agregado e sua estimativa. A estimativa/tempo foi calculada através de experiências de desenvolvimento sem utilizar o artefato que o *Scrum* oferece ao *Planning Poker*. Após a definição do *Product Backlog,* foram definidas as tarefas que compuseram cada *Sprint*. O projeto teve, no total, seis *Sprints*.

No decorrer do desenvolvimento de cada *Sprint,* foram utilizados os artefatos Gráfico de *Burndown,* para estimar o tempo e a complexidade do projeto, e Quadro de Tarefas, para identificar o andamento das funcionalidades.

Ao final de cada *Sprint,* foi gerada uma nova funcionalidade para o sistema que foi integrada a outras versões e, assim, gerou uma nova versão para o sistema. Assim foi realizado até que o ciclo de *Sprints* fosse finalizado.

As ferramentas que foram utilizadas para o estudo de caso das metodologias foram:

- **Java** – Linguagem de Programação utilizada.
- **Prime** Faces e JSF – Framework de apoio para desenvolvimento.
- **Eclipse** – Ambiente de desenvolvimento integrado utilizado na codificação do sistema.
- **Hibernate** – Framework Objeto Relacional para acesso ao banco de dados.
- **Case Studio** – Software utilizado na modelagem do banco de dados.

- **Microsoft Word** – Editor de textos usado para criação dos documentos do projeto.
- **Scrum Half** – Software utilizado para aplicar a metodologia *Scrum*.

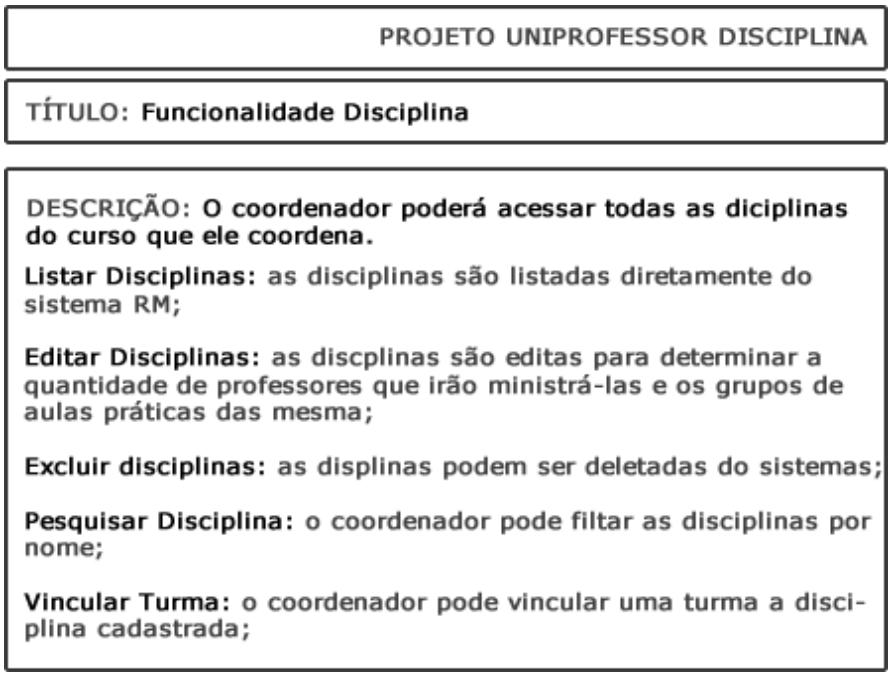
4 ANÁLISE E RESULTADOS

A análise e os resultados deste artigo foram definidos a partir das informações e artefatos que cada Metodologia de desenvolvimento gerou para viabilizar o processo de desenvolvimento de um software.

4.1 PLANEJAMENTO

A gestão de Planejamento, utilizando as metodologias XP e *Scrum,* ofereceu um suporte positivo para a inicialização do Projeto. Aplicando o XP para coleta de requisitos, foi possível elaborar Histórias de Usuário, identificando, detalhadamente, e abstraindo, de uma forma mais simples, a real necessidade da funcionalidade solicitada pelo cliente. Na Figura 6, pode ser identificado um cartão de História de Usuário da Funcionalidade Disciplina, onde são identificados o Título da História, sua descrição e funcionalidades comentadas.

Figura 6 – Cartão de História: Funcionalidade Disciplina



Fonte: Dados do Projeto

A partir dessas histórias, foi possível obter um detalhamento minucioso das tarefas a serem desenvolvidas, evitando os riscos de entendimento equivocado dos requisitos levantados pelo *Product Owner*. Ambas as metodologias empregam um

diálogo constante com o cliente, o que facilitou bastante na descrição e validação dos Cartões de Histórias escritos.

O projeto foi cadastrado na ferramenta *Scrum Half* com todas suas informações. Na Figura 7, seguem os itens inseridos. Em descrição, são colocados os dados do projeto, tais como Nome, Data Início, Data Início Game, que indica quando o *Product Backlog* foi criado, Administrador do Plano, que indica quem criou o projeto, uma breve descrição do projeto, a definição das histórias preparadas para serem incluídas no *Product Backlog*, os participantes envolvidos no projeto e seus papéis e configurações de contato do usuário.

Figura 7 – Informações do Projeto

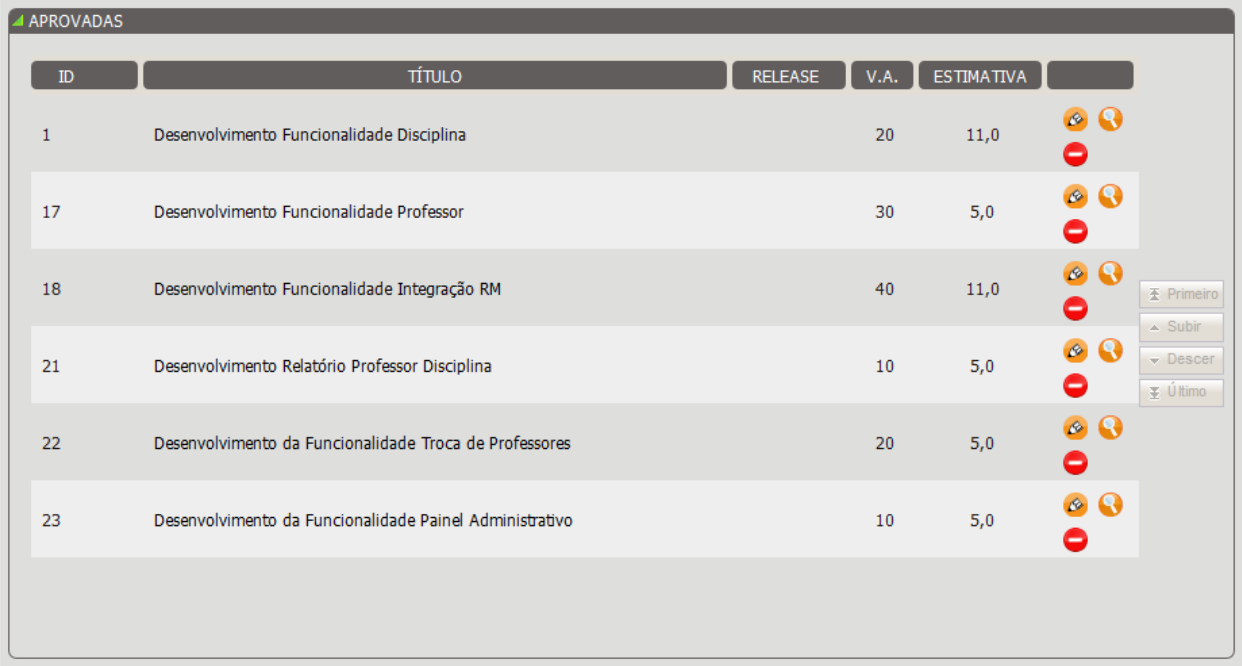
Fonte: Dados do Projeto

4.2 GERENCIAMENTO DAS TAREFAS

Com as histórias definidas, foi determinada a lista do *Product Backlog*, onde foram inseridas as funcionalidades do sistema. A ferramenta utilizada para gerenciamento desta lista foi o *Scrum Half*. O uso da mesma para a organização das tarefas foi bastante satisfatório, pois, como pode ser observado na Figura 8, foram cadastradas todas as funcionalidades do sistema e, a cada funcionalidade cadastrada, já era estimado um Valor Agregado. A importância da tarefa foi definida com sua estimativa de progresso/pontos que são contados no gráfico de *Burndown* e no desenvolvimento da *Sprint*. Como a ferramenta é integrada com todos os envolvidos

do projeto a partir do gerenciador *Scrum Half*, todos tinham uma prévia do tempo e da complexidade das tarefas a serem desenvolvidas.

Figura 8 – *Product Backlog* do Projeto



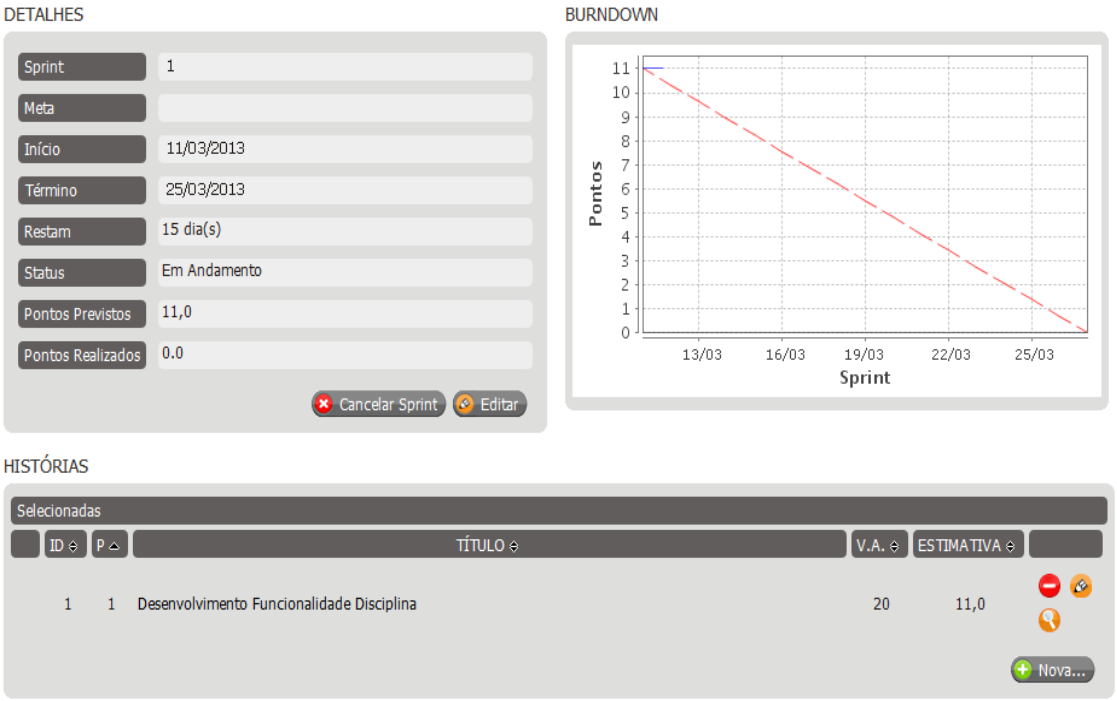
Fonte: Dados do Projeto

Portanto, o *Product Backlog* foi o ponto inicial para que todos pudessem obter o conhecimento do projeto e de sua extensão.

4.3 EXECUÇÃO DAS TAREFAS

Definidos os itens do *Product Backlog*, foram identificadas as tarefas que compuseram cada *Sprint Backlog*. A Figura 9 ilustra a primeira *Sprint* cadastrada na Ferramenta *Scrum Half*. Com essa possibilidade de cadastrar a *Sprint*, foi possível definir a data de início e a data de término e, assim, acompanhar o andamento de cada *Sprint*. A linha pontilhada tracejada na Figura 9 é o ponto de equilíbrio, ou seja, a estimativa perfeita que deve ser cumprida até o final da *Sprint*. A linha azul indica os pontos que foram cumpridos no decorrer da *Sprint*, não podendo ultrapassar o ponto final da linha vermelha, que indica que houve um atraso na entrega da *Sprint*. Como todos do time acompanham a *Sprint* e têm um diálogo constante, se for diagnosticado, em uma reunião diária, que o programador está com dificuldades e está próximo da data de entrega, é importante que o *Scrum Master* intervenha e coloque mais pessoas experientes para ajudar nessa *Sprint*, para que a entrega não atrase.

Figura 9 – Gerenciador da Sprint – *Scrum Half*



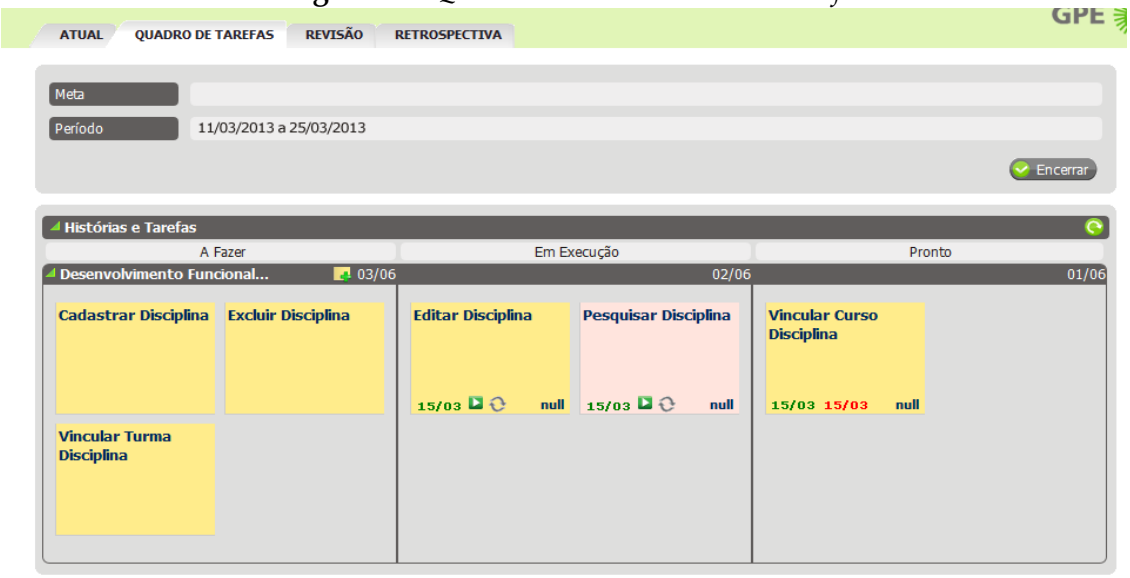
Fonte: Dados do Projeto

No decorrer do desenvolvimento da *Sprint*, foi utilizado outro artefato do *Scrum*, que é o Quadro de Tarefas.

Na Figura 10, é apresentado o Quadro de Tarefas da *Sprint Backlog* Disciplina, que foi quebrada em pequenas tarefas, no total de seis. No início, todas estavam em espera para desenvolvimento, na coluna A FAZER. A primeira tarefa que entrou em execução foi a "Vincular Curso Disciplina", pois estava ligada a todas as outras tarefas que deveriam ser realizadas e, consequentemente, a primeira a ser finalizada, ficando, assim, com o status PRONTO.

No decorrer do desenvolvimento das tarefas desta *Sprint*, foi diagnosticado um bug na tarefa "Pesquisar Disciplina", destacado na ferramenta *Scrum Half*. Essa ficou por alguns dias em execução, pois exigiu um pouco mais de pesquisa para solucionar o problema, o qual foi solucionado em alguns dias e passado para o status PRONTO.

Figura 10– Quadro de Tarefas – *Scrum Half*

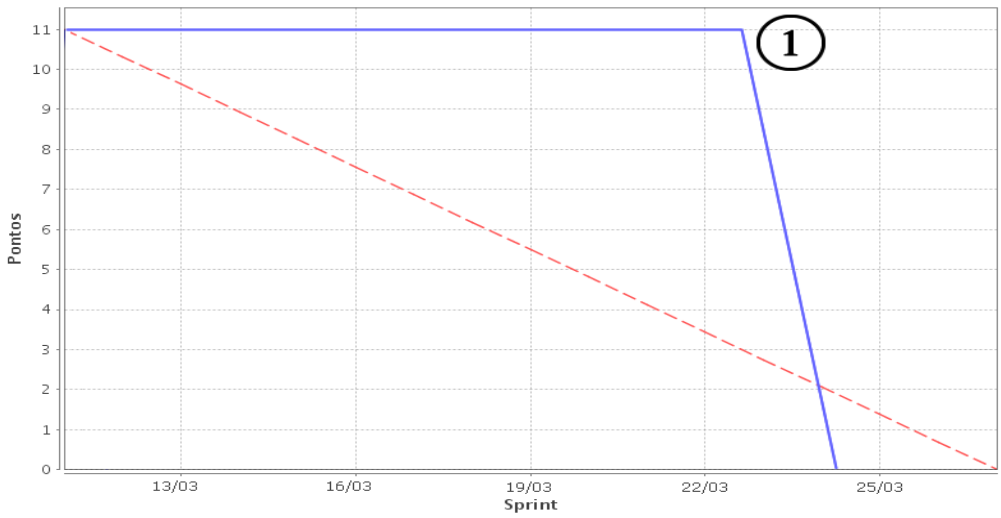


Fonte: Dados do Projeto

O Quadro de Tarefas, juntamente com as reuniões diárias, foi um dos principais componentes que o *Scrum* ofereceu no desenvolvimento. Foi feita uma excelente organização e gerenciamento das tarefas e da equipe, proporcionando um resultado positivo no acompanhamento de todo o processo da *Sprint*.

Após a finalização da *Sprint* Disciplina, a mesma foi para etapa de Revisão. Nesta etapa, a equipe apresentou o que foi construído e foi gerado o gráfico de *Burndown* da primeira *Sprint*, apresentado na Figura 11, sendo a tarefa foi finalizada antes da data prevista e a meta alcançada, como pode ser observado através da linha identificada pelo número 1, que todos os pontos foram cumpridos antes da data final.

Figura 11 – Gráfico de *Burndown Sprint* Disciplina



Fonte: Dados do Projeto

4.4 FINALIZAÇÃO DA TAREFA

Nesta etapa, foi realizada a Retrospectiva da *Sprint* ou *Sprint Retrospective.* Foram avaliados os pontos positivos e negativos ocorridos no decorrer da implementação da *Sprint*. Essa retrospectiva final foi interessante, pois, juntamente com o *Scrum Master,* foi possível realizar uma reflexão sobre o desenvolvimento e iniciar o próximo ciclo com um pouco mais de experiência.

4.5 QUALIDADE DO PROJETO DESENVOLVIDO

A qualidade final do projeto foi bastante satisfatória. Devido à facilidade de gerenciamento que se deu no decorrer do desenvolvimento, os objetos, a exemplo do cartão de história, do quadro de tarefas e do gráfico de *Burndown,* que as metodologias geram facilitam bastante a ordem cronológica que cada tarefa deve ser cumprida. Isso ajuda o programador, pois ele não se perde e nem se dispersa do que deve ser cumprido. Assim, como as metodologias ágeis empregam a simplicidade, o projeto está em um padrão de fácil manutenção.

4.6 RESULTADOS DOS ARTEFATOS GERADOS

Com os artefatos que as duas metodologias geraram, foi possível obter um gerenciamento mais organizado do projeto. Na fase inicial de levantamento de requisitos com a metodologia XP, foi realizada, por meio dos cartões de história, a priorização dos requisitos mais importantes que o cliente desejava. Após esta etapa, o projeto passou a ser gerenciado através da ferramenta *Scrum Half*. O primeiro passo foi definir as histórias que compuseram o *Product Backlog* do projeto Unidisciplina e, assim, desenvolvida a primeira *Sprint* do projeto.

No processo da execução da *Sprint,* utilizando os artefatos do *Scrum,* foi possível elaborar um plano de desenvolvimento com a lista de tarefas chamada de *Backlog* da *Sprint*. Na execução dessas tarefas, foi acompanhado, passo a passo, o desenvolvimento de forma cronológica e organizada, através de reuniões de planejamento, reuniões diárias para acompanhar o que o time estava fazendo, gráfico de *Burndown,* que permitiu medir as dificuldades do desenvolvimento, e o quadro de tarefas, que possibilitou trabalhar de forma organizada e sincronizada com as tarefas do *Backlog*. Com todos esses artefatos, a equipe trabalhou de forma auto gerenciável, sem se perder no projeto.

Realizados esses processos, o *Backlog* do produto foi atualizado e o ciclo de vida do *Scrum* foi reiniciado para desenvolvimento da próxima *Sprint* da fila.

5 CONCLUSÃO

Tendo como base a análise e resultados descritos anteriormente, o gerenciamento de um projeto, utilizando métodos ágeis, apresenta resultados positivos em grande escala. Todo o processo de desenvolvimento é acompanhado minuciosamente e as tarefas são designadas exatamente pra quem tem maior domínio

sobre a atividade a ser desenvolvida. Vale destacar, principalmente, o uso da ferramenta de gerenciamento *Scrum Half*, que apoiou o time em todos os processos. Desde a criação das histórias dos usuários, a *Sprint Restrospective,* que abre o ciclo para novas *Sprints*, possibilitou uma visão ampla para quem estava envolvido no projeto.

Alguns conjuntos de características que colaboraram para a entrega de um produto com qualidade e dentro do prazo estimado foram:

- A participação do cliente desde o início do projeto até o final, acompanhando o time em todos os processos;
- A organização das histórias do usuário e a priorização das funcionalidades;
- O versionamento das implantações de cada *Sprint*, gerando um produto e este é implantado para o cliente validar e utilizar;
- A simplicidade de desenvolvimento, em que a equipe toda trabalha em um mesmo nível e linguagem;
- A refatoração do código fonte, em que o mesmo é melhorado no decorrer de cada ciclo e reaproveitado;
- A comunicação entre a equipe através das reuniões diárias, quando é identificada a tarefa que cada membro realizou;
- A interatividade e companheirismo entre os membros, uma vez que, caso algum se diagnostique com dificuldade, ele tem total ajuda e apoio do seu time;
- A responsabilidade da equipe em cumprir o prazo determinado em cada *Sprint*, para que não atrase o projeto;
- A organização do *Product Backlog* com uma visão de todo o escopo do produto a ser cumprido e entregue para o cliente;
- Estruturação das tarefas dentro da *Sprint*, em ordem cronológica de execução, estimando o tempo de desenvolvimento;
- Quadro de tarefas que determina o que cada um irá desenvolver e em qual etapa o indivíduo está;
- O gráfico de *Burndown* que representa o desempenho de cada *Sprint* no decorrer do projeto;
- A revisão da *Sprint* com os desenvolvedores e o *Scrum Master* (Gerente de Projeto) que revisa e apresenta as tarefas construídas na *Sprint*;
- O *Spritg restrospective,* em que é avaliada e validada a *Sprint* desenvolvida para iniciar-se uma nova.

Trabalhar com todos estes itens gerados por essas metodologias foi uma experiência bastante produtiva, uma vez que não é necessário utilizar padrões de desenvolvimento ou linguagens complexas. O mais importante é definir um nome para funcionalidade e identificar o que aquele requisito irá fazer e se é isso mesmo que o cliente deseja.

Portanto, todas essas etapas garantem que o processo seja seguro e atenda às necessidades do cliente. O desenvolvimento desta pesquisa, utilizando o XP e o *Scrum*, foi importante para compreender e abstrair os conceitos e práticas envolvidas no desenvolvimento de sistemas, possibilitando entender as características que compreende cada um e como colocá-las em execução. Demonstrou ainda a importância

de uma equipe unida e comunicativa, o valor de uma equipe responsável para obter um produto nos altos padrões de qualidade.

Como resultado, este trabalho é uma referência para os demais alunos da área T.I. e profissionais, podendo aplicar os conceitos aqui observados e analisados no desenvolvimento dos seus projetos.

REFERÊNCIAS

BECK, Kent**.** *Programação Extrema (XP) explicada.* Porto Alegre: Bookman, 2004. 168 p.

COHN, Mike. *Desenvolvimento de Software com Scrum:* aplicando métodos ágeis com sucesso. Porto Alegre: Bookman, 2011.

DevMedia. *Introdução ao desenvolvimento ágil***.** Disponível em: <http://www.devmedia.com.br/introducao-ao-desenvolvimento-agil/5916>. Acesso em: 20 jul. 2011.

KNINBERG, Henrik. *Scrum e Xp direto das Trincheiras:* Como nós fazemos Scrum. Estados Unidos: C4media, 2007. 148 p.

PRESSMAN, Roger S. *Engenharia de software.* 6. ed. Rio de Janeiro: McGraw-Hill, 2006.

REZENDE, Denis Alcides. *Engenharia de software e sistemas de informação.* 3. ed. Rio de Janeiro: Brasport, 2005.

SEMERU. *O ciclo de vida do framework scrum*. Disponível em: http://www.semeru.com.br/blog/o-ciclo-de-vida-do-framework-scrum/. Acesso em dezembro de 2013.

SOMMERVILLE, Ian. *Engenharia de software.* 8. ed. São Paulo: Pearson Education - Br, 2007.

TELES, Vinícius Manhães. *Extreme Programming:* aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade. 3. ed. São Paulo: Novatec, 2006.