Aprimorando a Gerência e o Desenvolvimento de Software com Metodologias Ágeis

Mauricio Andreazza Sganderla[1], Guilherme Lacerda[1], Vinicius Gadis Ribeiro[1], Sidnei Renato Silveira[2]

[1]Faculdade de Informática – UniRitter (Centro Universitário Ritter dos Reis) – Curso de Bacharelado em Sistemas de Informação - Porto Alegre – RS – Brasil

[2]Departamento de Tecnologia da Informação – UFSM (Universidade Federal de Santa Maria)- Campus Frederico Westphalen – RS - Brasil

mauricio.webdev@gmail.com, guilhermeslacerda@gmail.com, vinicius.gadis@gmail.com, sidneirenato.silveira@gmail.com

Abstract.*This paper discusses the improvement of management and construction of software using agile methodologies like eXtreme Programming and Scrum. The best practices of both approaches where there was no well-defined software development process have been applied. The choice by the use of agile methods was defined because we needed to answer the dynamism of the current scenario, volatile requirements, environment more collaborative less bureaucratic having as main goal the software in operation and bring the ROI to the customer.*

Resumo. *Este artigo aborda a melhoria da gerência e construção de software utilizando as metodologias ágeis eXtreme Programming e Scrum. Foram aplicadas as melhores práticas de ambas as metodologiasem um ambiente em que não havia nenhum processo bem definido de desenvolvimento de software. A escolha pelo uso das metodologias ágeis foi definida, pois atende ao dinamismo do cenário atual, requisitos voláteis, ambiente mais colaborativo e menos burocrático, tendo como objetivo principal o software em funcionamento e que realmente traga retorno ao cliente.*

1. Introdução

Este trabalho propõe a aplicação das metodologias ágeis na gerência e construção de software considerando, como estudo de caso, um projeto de desenvolvimento de uma ferramenta de gerenciamento de processo seletivo de uma Instituição de Ensino Superior. Dentre as metodologias ágeis existentes, foram utilizados princípios, valores e práticas do *Scrum*, para gestão e acompanhamento, e da metodologia e*Xtreme Programming*, para práticas de implementação.

Com a globalização e o constante avanço tecnológico, a concorrência entre as empresas tornou-se muito acirrada. A necessidade de mudanças e o dinamismo no desenvolvimento de software cresceram juntos e as metodologias ágeis oferecem respostas rápidas a este novo cenário, pois trabalham com requisitos voláteis ou requisitos não esclarecidos totalmente, sempre com o foco no que realmente agrega valor ao cliente.

Neste contexto, o principal objetivo deste trabalho é aplicar gradativamente as metodologias ágeis dentro da gerência e construção de software, para que se tenha um ambiente mais colaborativo, menos burocrático, onde todos envolvidos no projeto participem efetivamente, trazendo satisfação tanto para a equipe como para o cliente, tornando-o um membro da equipe.

Como objetivo específico deste trabalho, pretende-se iniciar uma nova abordagem de desenvolvimento e gerência de projetos dentro da empresa em que foi realizado o estudo de caso, utilizando uma abordagem mais ágil, colaborativa e com o foco principal no desenvolvimento do software.

Este artigo aborda assuntos referentes ao processo de desenvolvimento de software e gerência de projetos. Na seção 2 é apresentado o referencial teórico, fundamentado em publicações de importantes autores na área. Na seção 3 encontra-se um comparativo com outros trabalhos realizados na área. Ao final encontra-se uma seção que contém a solução proposta para este trabalho com um mapa mental e descrição das ferramentas e técnicas aplicadas no estudo de caso realizado.

2. Gerenciamento de Projetos de Software

Esta seção apresenta uma introdução ao processo de desenvolvimento de software com conceitos básicos evoluindo até os principais conceitos e técnicas que envolvem o gerenciamento de projetos de software aplicando metodologias ágeis.

2.1. Processo de Software

Processo de *software* é um conjunto de atividades que leva à produção de um produto de software. Processos de *software* são complexos e, como todos os processos intelectuais e criativos, dependem do julgamento humano. Os processos evoluíram para explorar as capacidades das pessoas em uma organização e as características específicas do sistema que está sendo desenvolvido [Sommerville 2007].

Embora existam muitos processos de software diferentes, algumas atividades são fundamentais a todos os processos [Sommerville 2007]:

- Especificação de software: define a funcionalidade do software e as restrições sobre suas operações;
- Projeto e implementação: define a produção do software que atenda à especificação;

- Validação de software: visa garantir que o software faça o que o cliente deseja;
- Evolução de software: permite que o *software* evolua para atender as necessidades do cliente.

Muitas organizações não possuem processos definidos de desenvolvimento de software, pois não possuem recursos suficientes para adotar o uso de processos robustos. Os resultados desta falta de uso de processos envolvem a baixa qualidade do produto final, clientes insatisfeitos, entregas fora do prazo e alto custo de manutenção.

O modelo de processo de *software* é uma representação abstrata de um processo de *software*. Cada modelo é representado sob uma determinada perspectiva e fornece somente informações parciais.

2.2. Abordagens Tradicionais

Em 1970, *Royce* publicou o primeiro modelo de processo de desenvolvimento, chamado modelo em cascata. Devido ao encadeamento entre uma fase e outra é também conhecido como linear ou clássico, conforme ilustra a Figura 1 [Royce 1970].

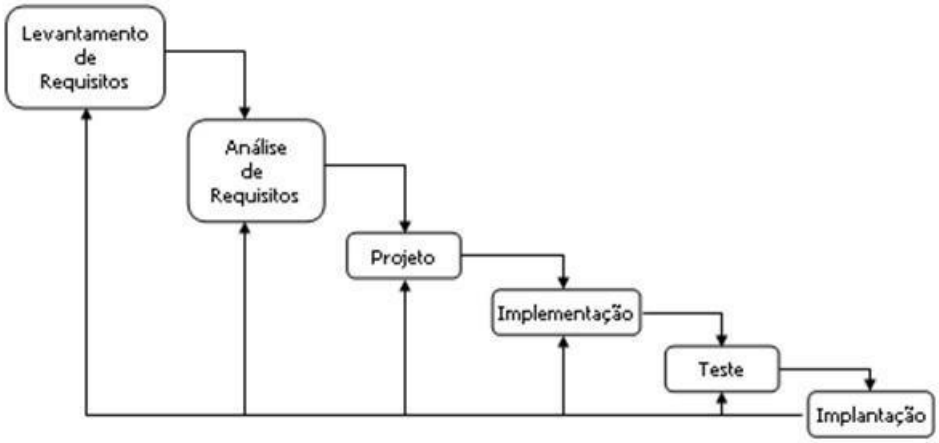


Figura 1. Modelo Cascata - fonte: [Royce1970].

De acordo com o modelo em cascata, cada fase não deve ser iniciada antes que a fase anterior tenha terminado e cada transição de fase consiste de um ou mais documentos aprovados. Devido aos custos de produção e aprovação de documentos, as iterações são onerosas e envolvem retrabalho significativo, pois o projeto só passa uma vez por este ciclo. Uma vez definidos os requisitos e iniciada a fase de implementação, os requisitos são congelados e não podem sofrer alterações. O congelamento prematuro de requisitos pode significar que o sistema não atinja o objetivo do cliente. É um modelo que dever ser utilizado em projetos onde os requisitos forem bem entendidos.

Muitas empresas utilizam este modelo, apesar das advertências de pesquisadores

na área que identificaram inúmeros problemas ao adotá-lo. Segundo Brooks (1987), a especificação total de um software antes do início de desenvolvimento é impossível.

Outro modelo muito utilizado é o evolutivo, que se baseia na ideia de um desenvolvimento a partir de prototipações iniciais, colhendo *feedback*do usuário e refinando este *feedback* por meio de várias versões, até que seja desenvolvido um sistema adequado. As atividades de especificação, desenvolvimento e validação são intercaladas. Ainda segundo Sommerville (2007), uma abordagem evolucionária para desenvolvimento de software é freqüentemente mais eficaz do que a abordagem em cascata na produção de sistemas que atendam às necessidades imediatas dos clientes.

Segundo Leite (2007), o modelo de transformação baseia-se na abordagem de métodos formais, isto é, utiliza-se de técnicas matemáticas para especificação, projeto e verificação. O desenvolvimento deve ser visto como uma seqüência de passos que gradualmente transforma uma especificação formal em um programa. O passo inicial é alterar os requisitos informais em uma especificação funcional formal, então esta descrição é transformada em outra descrição mais detalhada até chegar a programas que satisfaçam a especificação.

O modelo em espiral foi proposto por Boehm (1988). Conforme mostra a figura 2, ao invés de utilizar um processo de atividades sequenciadas uma após a outra, o processo é representado na forma de um espiral. Cada volta na espiral representa uma fase do processo de software e está dividida em quatro partes [Sommerville 2007]:

- Definição de objetivos: definição dos objetivos específicos. Os riscos de projeto são identificados e dependendo disso, estratégias alternativas podem ser planejadas;
- Avaliação e redução de riscos: para cada risco de projeto identificado, uma análise detalhada é realizada. Providências são tomadas para reduzir o risco;
- Desenvolvimento e validação: após a avaliação de risco, um modelo de desenvolvimento para o sistema é selecionado. Aqui conforme os riscos identificados pode-se utilizar um modelo de processo em cascata, prototipação ou baseado em transformações formais;
- Planejamento: o projeto é revisado e uma decisão é tomada para prosseguimento da próxima volta do espiral.

Ainda segundo Sommerville (2007), a principal diferença entre o modelo em espiral e outros modelos de processo de software é o reconhecimento explícito do risco, pois riscos podem causar problemas no projeto, tal como ultrapassar o cronograma e os custos.

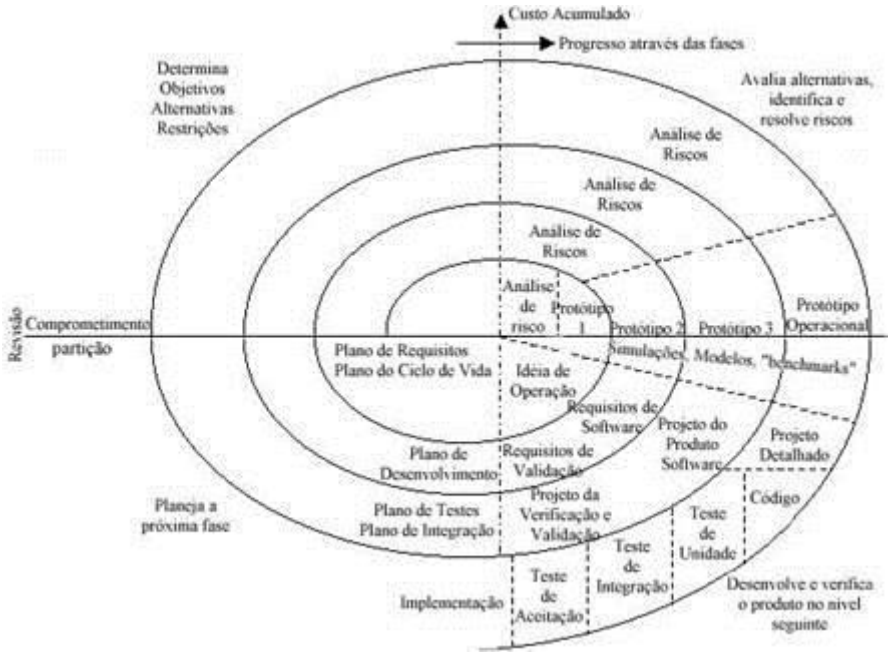


Figura 2. Modelo Espiral- fonte: [Donato 2010].

2.2. Metodologias Ágeis

Em fevereiro de 2001, um grupo inicial de 17 pessoas com várias áreas de formação diferente definiu um manifesto para encorajar melhores maneiras de desenvolver *software*, chamado de Manifesto Ágil. Além deste manifesto, este grupo de pessoas formulou um conjunto de princípios e valores para os processos de desenvolvimento ágil de *software* [Ambler 2004].

O manifesto ágil é definido por quatro valores [Beck et. al. 2001]:

Quadro 1. Trecho do manifesto ágil

"*Estamos descobrindo maneiras melhores de desenvolver software fazendo-o nós mesmos e ajudando outros a fazê-lo. Através deste trabalho, passamos a valorizar:* *Indivíduos e interação entre eles mais que processos e ferramentas* *Software em funcionamento mais que documentação abrangente* *Colaboração com o cliente mais que negociação de contratos* *Responder a mudanças mais que seguir um plano*"

Para ajudar as pessoas a compreenderem melhor o desenvolvimento de *software* ágil, foram definidos doze princípios aos quais as metodologias de desenvolvimento *software* ágeis estão adequadas. Estes princípios compreendem [Beck et. al. 2001]:

1. A maior prioridade é satisfazer o cliente mediante entregas de *software* de valor em tempo hábil e continuamente;

2. aceitar mudanças de requisitos, em qualquer fase do projeto;
3. entregar *software* na menor escala possível de tempo;
4. equipe de desenvolvimento e cliente são do mesmo time;
5. construir projetos com indivíduos motivados e comprometidos com o resultado;
6. comunicação efetiva;
7. *software* funcionando é a principal medida de progresso;
8. promover o desenvolvimento sustentável;
9. atenção contínua à excelência técnica;
10. simplicidade;
11. as melhores arquiteturas, requisitos e projetos provêm de equipes organizadas;
12. refletir sobre como se tornar mais eficaz, ajustando e adaptando o comportamento da equipe.

Com base nos valores e princípios citados anteriormente, eles serão seguidos na aplicação do estudo de caso apresentado neste artigo, introduzindo gradativamente os valores e princípios na equipe que participou do projeto, visando um fácil aprendizado, formando um alicerce de bom senso para basear o trabalho de desenvolvimento de software bem sucedido.

2.2.1. Scrum

O *Scrum*[1] é um processo baseado em práticas, artefatos e algumas regras simples e fáceis de aprender. Não é um processo prescritivo e não descreve o que deve ser feito em cada uma das situações. O *Scrum* é usado para trabalhos complexos, onde é impossível predizer tudo que irá acontecer. Ele oferece um *framework* e uma série de práticas que mantêm tudo bem claro, trazendo para os seus praticantes o conhecimento exato do andamento do projeto e a possibilidade de realizar ajustes de forma correta para manter o projeto no objetivo desejado [Prikladnicki et. al. 2014][Schwaber 2004].

O *Scrum* baseia suas práticas em processos iterativos e incrementais. Na Figura 3 são esquematizados dois círculos: o inferior representa a iteração de desenvolvimento, geralmente com duração de duas a quatro semanas, que acontecem uma após a outra. O resultado de cada uma das iterações é um incremento do produto. O círculo superior representa o trabalho diário que acontece dentro de cada iteração, onde o time se reúne para inspecionar cada uma das suas atividades e realizar adaptações se necessário. O que conduz esta iteração é a lista de requisitos e este ciclo se repete até o fim do projeto.

Antes de cada iteração, o time verifica os requisitos levantados do projeto e através das suas próprias habilidades e capacidade determina coletivamente como serão desenvolvidas as funcionalidades, modificando através dos encontros diários eventuais problemas, dificuldades, surpresas durante a iteração. O time define o que precisa ser

[1]*Framework* para gerência de projetos, baseados em uma abordagem ágil. Os termos utilizados na descrição (como *Scrum Master*, *ProductOwner*) não serão traduzidos, por fazerem parte da metodologia.

feito e escolhe a melhor maneira para ser feito. Este processo criativo é fundamental para a produtividade dentro do *Scrum* [Kniberg 2007].

Figura 3. O esqueleto do Scrum - fonte: [Schwaber 2004].

O *Scrum* disponibiliza alguns artefatos ao processo de desenvolvimento de software. Eles são utilizados durante o processo e são descritos da seguinte forma [Pham; Pham 2012][Prikladnicki et. al. 2014] [Schwaber 2004]:

> - *Product Backlog:* é uma lista com todos os requisitos ordenada pelo valor de negócio mantido pelo *Product Owner*. O *Product Backlog* nunca está completo e é dinâmico, pois novos requisitos podem surgir e devem fazer parte da lista;
>
> - *Sprint Backlog*: é uma lista de tarefas que o time define com base no *Product Backlog*. Somente o time pode realizar alterações na lista. O *Sprint Backlog* possibilita o acompanhamento em tempo real do trabalho planejado pelo time a ser realizado no *Sprint.*

Segundo Schwaber (2004), no *Scrum* existem somente três papéis definidos: o *Product Owner,* o time e o *Scrum Master.* Todas as responsabilidades de gerência no projeto são divididas entre esses três papéis. O *Product Owner* é responsável por representar os interesses de todos com participação no projeto. O *Product Owner* cria uma base inicial e permanente para o projeto através de uma lista de requisitos iniciais, planos de entrega e retorno sobre os objetivos de investimento. A lista de requisitos é chamada de *Product Backlog*. O *Product Owner* é responsável por usar o *Product Backlog* para assegurar que as funcionalidades que tragam maior valor para o cliente sejam produzidas primeiro. O time tem a responsabilidade de desenvolver funcionalidades, e é composto por pessoas auto-organizadas, auto-gerenciáveis e responsáveis por descobrir como transformar o *Product Backlog* em um incremento de funcionalidade dentro de uma iteração. Os membros do time são coletivamente responsáveis pelo sucesso de cada iteração e do projeto como um todo. O *Scrum Master* é responsável por ensinar, a todos os envolvidos no projeto, a implementação do *Scrum*,

para que ela caiba dentro da cultura da organização, além de assegurar que todos sigam as regras e práticas propostas.

As pessoas que preenchem estes papéis são as pessoas que estão comprometidas com o projeto. Outras pessoas poderão estar interessadas, mas não comprometidas. O *Scrum* diferencia bem estes dois grupos e assegura que aqueles que são responsáveis pelo projeto têm a autoridade para realizar alterações necessárias para o sucesso e aqueles que não têm responsabilidades não interfiram nas decisões.

Segundo Schwaber (2004), um projeto *Scrum* inicia com uma visão do sistema a ser desenvolvido. A visão no início pode ser vaga, às vezes em termos de negócio muito mais do que de sistema, mas com o andamento do projeto se torna mais clara. O *Product Owner* é responsável por manter a visão do projeto de maneira que maximize o retorno sobre o investimento. O *Product Owner* elabora um plano para isto, que inclui o *Product Backlog;* este é uma lista de requisitos funcionais e não funcionais priorizados, que garante que os itens mais importantes estarão no topo da lista e é dividido em entregas.

Cada *Sprint* é uma iteração de duas a quatro semanas. Cada *Sprint* é iniciado com uma *Sprint planning meeting*, uma reunião aonde o *Product Owner* e o time revisam juntos o que deve ser feito para o próximo *Sprint.* Selecionando os itens com maior prioridade no *Product Backlog*, o *Product Owner* diz ao time o objetivo do próximo *Sprint,* e o time diz ao *Product Owner* o quanto do objetivo poderá se tornar em funcionalidade durante o próximo *Sprint* [Pham; Pham 2012].

A *Sprint planning meeting* é dividida em dois momentos. O primeiro momento é responsável pelo *Product Owne r*apresentar os itens mais prioritários no *Product Backlog* ao time. O time questiona sobre o conteúdo, proposta, significado e intenção do *Product Backlog*. Quando o time já sabe o suficiente, ele seleciona o máximo de itens do *Product Backlog* que acreditam que seja possível tornar um novo incremento de funcionalidade a ser entregue ao final do *Sprint.* No segundo momento, o time planeja o *Sprint*, pois é ele o responsável por gerenciar o seu próprio trabalho. As tarefas que surgem durante o planejamento formam o *Sprint Backlog*, uma lista de tarefas elaborada pelo time a serem executadas para atingir o objetivo do *Sprint*.

Para manter o sincronismo das tarefas é realizada uma reunião diária com duração máxima de quinze minutos chamada *Daily Scrum*,onde cada membro do time responde três perguntas [Schwaber 2004]:

- O que foi realizado desde a última reunião?
- O que você pretende realizar entre hoje e a próxima reunião?
- Há algum obstáculo que o impede de prosseguir neste *Sprint*?

Ao final do *Sprint*, uma reunião chamada *Sprint review* é realizada. Esta reunião é composta pelo time, pelo *Product Owner* e pelas pessoas interessadas no projeto que desejam comparecer. Esta reunião informal tem como objetivo apresentar a funcionalidade a todas as pessoas envolvidas interessadas e ajudar coletivamente a

determinar o que o time deve realizar no próximo *Sprint.* Após esta reunião, o *Scrum Master* estimula o time a revisar, com base nos princípios e práticas do *Scrum*, o que pode ser feito para tornar os próximos *Sprints* mais agradáveis e efetivos. Esta reunião é chamada de *Sprint retrospective*. A figura 4 apresenta um esquema do processo *Scrum*.

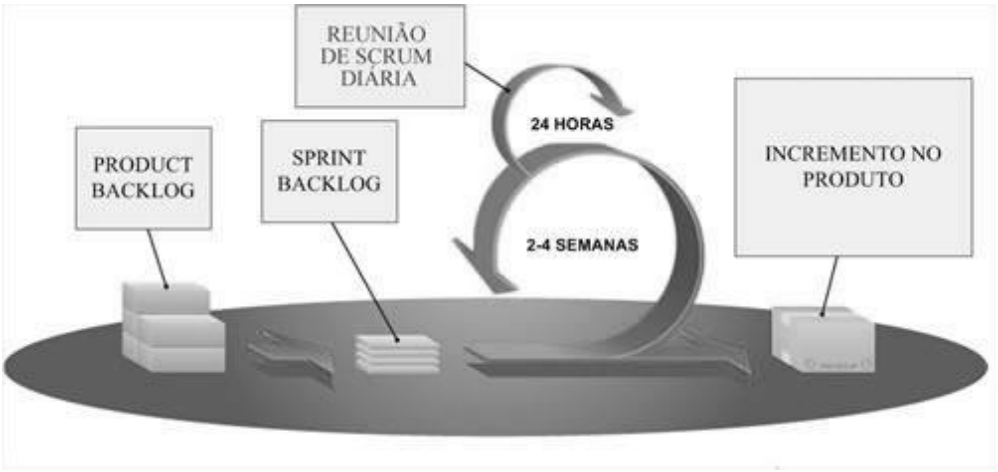


Figura 4. Processo Scrum - fonte: [Cohn 2005].

2.2.2. Extreme Programming (XP)

O*eXtreme Programing*[2], ou XP, é um processo ágil de desenvolvimento de *software* voltado para projetos com requisitos vagos e voláteis, desenvolvimento de sistemas orientado a objeto, equipes pequenas e desenvolvimento incremental, onde o sistema começa ser implementado logo no início e vai ganhando novas funcionalidades ao longo do projeto. XP e outros processos ágeis compartilham a mesma ideia de que o cliente aprende sobre suas necessidades, na medida em que usufrui do sistema que está sendo produzido. Com base no *feedback* do sistema, o cliente reavalia suas necessidades e possibilita que ele direcione o desenvolvimento para que a equipe produza sempre aquilo que tenha mais valor para o seu negócio [Prikladnicki et. al. 2014] [Teles 2004].

Ainda segundo Teles (2004), XP é baseado em um conjunto de valores e práticas que atuam de forma harmônica e coesa para assegurar que o cliente sempre receba um alto retorno do investimento em *software;* seus valores fundamentais são o *feedback*, comunicação, simplicidade e a coragem. Estes quatros valores juntos formam a base do XP para que cliente e equipe trabalhem juntos em harmonia de forma clara e objetiva. Seguir estes valores permite que todos os detalhes do projeto sejam tratados com a atenção e agilidade, pois a equipe também aprende que a simplicidade é implementar somente aquilo que é suficiente para atender cada necessidade do cliente.

O XP baseia-se nas seguintes práticas [Prikladnicki et. al. 2014] [Teles 2004]:

[2] Processo ágil para desenvolvimento de software. Os termos utilizados na descrição (como *Stand UpMeeting*, *Coach*, *Tracker*) não serão traduzidos, por fazerem parte da metodologia

- Cliente Presente: o cliente deve conduzir o desenvolvimento a partir do *feedback*que recebe do sistema; é essencial que ele participe ativamente do processo de desenvolvimento, pois viabiliza a simplicidade do processo em vários aspectos;

- Jogo do Planejamento: assegura que a equipe esteja sempre trabalhando naquilo que é o mais importante para o cliente. Por isto, XP é dividido em entregas e iterações para que o cliente tenha a oportunidade de revisar o planejamento;

- *Stand Up Meeting*: reunião em pé, com a equipe de desenvolvimento, que ocorre cada manhã para avaliar o trabalho que foi executado no dia anterior e priorizar o que será desenvolvido durante o dia;

- Programação em Par: desenvolvedores implementam funcionalidades em pares, ou seja, em cada computador, existem sempre dois desenvolvedores que trabalham juntos para produzir o mesmo código. Esta prática permite que o código seja revisado permanentemente, e torna a implementação mais simples e eficaz, pois os desenvolvedores se complementam e têm mais oportunidades de gerar soluções inovadoras;

- Desenvolvimento Guiado pelos Testes: desenvolvedores escrevem testes para cada funcionalidade antes de codificá-las, com isto eles aprofundam o entendimento das necessidades do cliente. Contribui para agilidade, pois o time passa a contar com uma massa de testes que pode ser usada a qualquer momento;

- Refatoração: desenvolvedores alteram código sem afetar a funcionalidade que ele implementa. Torna o software mais simples de ser manipulado e se baseia nos testes descritos, anteriormente, para garantir que as alterações não interrompam o funcionamento;

- Código Coletivo: todos desenvolvedores têm acesso a todas as partes do código e podem alterar aquilo que julgarem importante sem a necessidade de pedir autorização de outra pessoa. Não há desenvolvedores responsáveis por cada parte do sistema e sim todos os desenvolvedores são responsáveis;

- Código Padronizado: a equipe estabelece padrões para codificação, para que todos os desenvolvedores possam manipular qualquer parte do sistema e permitir que qualquer alteração futura seja efetuada mais rapidamente;

- Design Simples: a equipe opta sempre por um código que seja suficiente para atender às necessidades da funcionalidade que está implementando. Por isto, adota a simplicidade do design ao invés das generalizações dentro do código;

- Metáfora: a equipe estabelece junto ao cliente, por meio de metáforas, a transmissão de idéias complexas de forma simples;

- Ritmo Sustentável: XP recomenda que os desenvolvedores trabalhem apenas oito horas por dia e evitem fazer horas-extras, pois é essencial estar descansado a cada manhã para utilizar a mente na sua plenitude durante o dia;

- Integração Contínua: assegura que todo o sistema esteja sempre funcionando de forma harmoniosa, pois cada par de desenvolvedores integra seus códigos várias vezes ao dia. Cada vez que um par integra, ele executa todos os testes para assegurar que a integração tenha ocorrido de forma satisfatória;

- *Releases* Curtos: a equipe produz um conjunto reduzido de funcionalidades e coloca em produção rapidamente, de modo que o cliente já possa utilizar o software no dia a dia e se beneficiar dele. A cada nova entrega o sistema ganha novas funcionalidades, gerando mais valor para o cliente.

Teles (2004) afirma que, uma equipe que utilize XP normalmente é composta por pessoas que representam os seguintes papéis:

- Gerente de Projeto: é o responsável pelos assuntos administrativos do projeto, procurando liberar a equipe de questões que não estejam diretamente ligadas à atividade diária de desenvolvimento. Também mantém o relacionamento entre o desenvolvimento e o cliente;

- *Coach*: é o responsável técnico do projeto, tendo como objetivo assegurar o bom funcionamento do processo, buscando formas de melhorá-lo continuamente e que a equipe siga as boas práticas recomendadas pelo XP;

- Analista de Teste: é o responsável por ajudar o cliente a escrever os testes de aceitação e ajudar a tornar estes testes mais automatizados possíveis. Estabelece que os testes devam ser executados diversas vezes durante as iterações;

- Redator Técnico: é o responsável por ajudar a equipe de desenvolvimento a documentar o sistema e permite que os desenvolvedores se concentrem exclusivamente no desenvolvimento do software;

- Desenvolvedor: é a pessoa que analisa, projeta e codifica o sistema, pois dentro do XP não existem divisões entre os papeis citados acima. Durante o projeto, cada desenvolvedor exerce estes diferentes papéis em diversos momentos do projeto;

- *Tracker*: é o responsável por trazer para o time dados, gráficos, informações que mostrem o andamento do projeto e ajudem a equipe a tomar decisões de implementação, arquitetura e *design*.

3. Trabalhos Relacionados

Esta seção apresenta um estudo e traça um comparativo com outros trabalhos realizados na área, envolvendo a aplicação de metodologias ágeis no processo de desenvolvimento de software.

No estudo de caso realizado por Savoine (2009) foram aplicadas algumas práticas do *Scrum* no nível de gerenciamento e planejamento de projeto e eX*treme Programming* para práticas de implementação.

O projeto, contando com uma equipe de 15 colaboradores, tinha o objetivo de desenvolver um *software* de gestão de ensino, em uma fábrica de *software* que já desenvolvia outros produtos. Os resultados apresentam que houve sucesso no projeto e na transição da abordagem tradicional para ágil dentro da equipe, embora destaquem que metodologias ágeis não são aplicáveis em todas as situações e que freqüentemente requerem adaptações.

Os resultados também destacam que não foi possível realizar duas práticas do *eXtreme Programming* devido aos seguintes fatores: cliente sempre presente, pois o projeto iniciou não tendo patrocinador específico; semanas de 40 horas, pois a empresa possuía uma carga horária mensal de 44 horas.

Barros (2009) apresenta um trabalho em que foi aplicado somente o *Scrum* em uma equipe com quatro colaboradores, com o foco na gestão e acompanhamento do projeto de desenvolvimento de software.

O software em questão tinha como objetivo informatizar o processo de cadastro e acompanhamento familiar realizado por agentes comunitários de saúde. A aplicação do *Scrum* permitiu que vários requisitos, não definidos na primeira entrevista para levantamento de requisitos, fossem incluídos no projeto ao longo do mesmo, sem problemas.

Neste estudo de caso não foi relatada nenhuma restrição referente à adoção do *Scrum* para gestão e acompanhamento de projeto e nenhuma outra metodologia ágil foi utilizada.

Segundo Chapiewski (2008), a empresa Globo.com necessitava de um processo de desenvolvimento de *software*, pois poucos projetos eram entregues na data acordada e muito dos projetos falhavam ou não satisfaziam as necessidades dos clientes.

A adoção das metodologias ágeis dentro da empresa surgiu de baixo para cima, através de um movimento tímido entre alguns desenvolvedores ao aplicar praticas ágeis do *eXtreme Programming* nos projetos. Aos poucos novas práticas foram aplicadas e uma organização no nível de gerenciamento e planejamento começou a ser aplicada com base no *Scrum*. Com a difusão das práticas e já colhendo resultados, a empresa decidiu investir em um curso de *Scrum* para alguns membros da equipe.

A partir desse momento o *Scrum* foi adotado dentro da empresa, até mesmo em um projeto que já estava em andamento, mas utilizando uma abordagem mais tradicional. As práticas foram introduzidas aos poucos no projeto e, em pouco tempo, a equipe conseguiu abstrair a abordagem ágil.

Como resultado, destaca-se que um projeto foi entregue no prazo, com alto nível de qualidade, colocado em produção em poucas horas, com baixíssimo nível de *bugs* e o cliente satisfeito com o produto.

3.4 Conclusões e Comparativo

Em todos os estudos de casos relatados observa-se a utilização de pelo menos uma das metodologias ágeis que é proposta neste trabalho (*Scrum* e/ou *eXtreme Programming*). Com base no resultado dos trabalhos citados é possível concluir que todos chegaram a um resultado positivo e muito se assemelham a proposta deste trabalho.

No Quadro 2 é possível identificar um comparativo da proposta deste trabalho em relação aos outros estudados.

Quadro 2. Comparativo entre os Trabalhos Estudados e o Trabalho Proposto

	Scrum	***eXtreme Programming***	**Integrantes da Equipe**	**Tipo de Projeto**
Software Gestão de Ensino	X	X	15	Web
Software Gestão da Saúde	X		4	Web
Globo.com	X	X	●[3]	Web
Trabalho Proposto	X	X	8	Web

4. Solução Implementada

Esta seção descreve o motivo da escolha das metodologias ágeis para aplicação no estudo de caso proposto, além das experiências na introdução dos métodos ágeis. Inicialmente o problema é apresentado, bem como os motivos que levaram à escolha da adoção de uma abordagem ágil. Na seção seguinte são abordadas as experiências na adoção dos métodos ágeis dentro do projeto, levantando pontos chaves que auxiliaram no andamento do projeto. Finalizando a seção, é traçado um comparativo envolvendo as dificuldades e os benefícios encontrados com base no desenvolvimento deste trabalho.

4.1 O Problema

Este trabalho surgiu a partir da proposta de desenvolver um projeto de alto risco para uma Instituição de Ensino, mais especificamente um gerenciador de processo seletivo. Alguns fatores iniciais, tais como (i) cliente distante geograficamente, (ii) a existência de um sistema legado com vários anos de existência, (iii) o curto prazo para

[3] Não foi informada a quantidade de integrantes da equipe.

desenvolvimento e (iv) inexperiência da equipe de desenvolvimento, já demonstravam uma prévia das dificuldades que seriam encontradas no projeto.

Além dos fatores relacionados, o setor de desenvolvimento não possuía um processo bem definido se estendendo desde gestão e acompanhamento do projeto como fatores referentes à Engenharia de Software. A maioria dos projetos já desenvolvidos seguiu uma linha mais tradicional, onde ocorriam reuniões com os representantes dos setores solicitantes definindo o escopo do projeto e cronograma para entrega. Após estas definições eram gerados artefatos como modelos ER (Entidade-Relacionamento) de banco de dados, diagramas de classes e casos de uso. A partir destes artefatos o programador designado codificava o sistema e, caso tivesse alguma dúvida referente ao projeto, o programador se reportava ao responsável pelo projeto dentro do setor.

Durante todo este processo, os solicitantes do sistema não tinham nenhum contato com o software a ser desenvolvido até a entrega do produto. Muitos projetos acabaram sendo entregues atrasados e os projetos que foram entregues no prazo tiveram que voltar para o desenvolvimento, pois ou tiveram problemas no levantamento de requisitos e quando o cliente visualizou o sistema não era o que ele realmente pretendia ou problemas devido ao tempo de desenvolvimento e entrega já que, devido ao tempo da entrega do sistema, os requisitos levantados já não atendiam o que foi levantado ou até mesmo as pessoas que solicitaram não estavam mais presentes na Instituição.

A primeira mudança detectada foi a maneira de gerenciar o projeto, adotando o *Scrum* para gestão e acompanhamento. Em paralelo foram adotadas práticas do *eXtreme Programming* para engenharia e a consolidação de um time para o projeto. Nas próximas seções serão relatadas as experiências e a adoção das metodologias ágeis no referido projeto.

4.2 A Implementação

Nesta seção são abordadas as experiências na introdução das metodologias ágeis durante o desenvolvimento do projeto. A primeira seção relata como foi a adoção das metodologias ágeis no contexto do planejamento e gestão do projeto utilizando o *Scrum*. Na seção seguinte é abordada a utilização do *eXtreme Programming* na utilização de práticas de engenharia que, juntamente ao *Scrum*, fornecem sustentação ao desenvolvimento do projeto.

4.2.1 Planejamento e Gestão do Projeto

O projeto apresentava as seguintes características: (i) prazo curto para desenvolvimento; (ii) requisitos voláteis, com modificações freqüentes; (iii) cliente geograficamente distante, dificultando a comunicação e *feedback* do trabalho realizado. Com base nestas premissas, optou-se por adotar o *Scrum*.

Optou-se por utilizar pequenas iterações, de uma semana, pois a dificuldade em definir os requisitos no inicio do projeto era grande. Para o início do projeto já se

possuía uma quantidade de requisitos que tinham sido levantados em uma reunião inicial com o cliente. Como era tudo muito novo para a equipe em termos de conhecimento tanto do projeto como do produto, não se tinha uma estimativa mais precisa para passar ao cliente. Neste sentido, a maneira mais eficaz encontrada foi deixar com o time o compromisso de o quanto eles conseguiriam produzir em uma semana nos *Sprints* iniciais, até conseguir a velocidade do time por pontos[4]. Após alguns *Sprints* já foi possível ter um estimativa mais precisa do que o time conseguia produzir por semana e, a cada nova entrega semanal, obtinha-se mais conhecimento do produto através do *feedback* do cliente. Por meio da prática de se trabalhar em pequenas iterações foi possível notar a facilidade de acolher as mudanças e o baixo custo que isso gera dentro do projeto, pois quanto mais cedo é validado com o cliente menor o custo da mudança, como mostra a Figura 5.

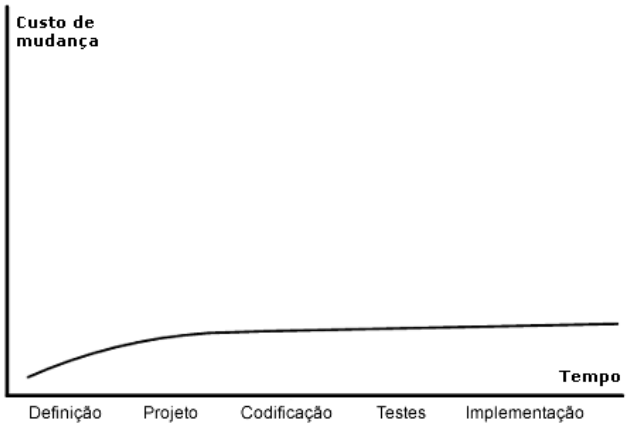


Figura 5. Custo da mudança – fonte [Beck 2008].

O maior risco que tinha sido identificado no projeto e também uma restrição imposta pelo cliente, era que este novo sistema atendesse a todas as funcionalidades que o outro sistema já atendia. Desde o início do projeto ficou bem claro que não seria possível contemplar todo o sistema antigo no novo, devido ao curto prazo estipulado para entrega. Então foi necessário definir prioridades nos requisitos, sempre com foco no que traria alto valor ao cliente e que tivesse alto risco, pois assim eliminamos os maiores riscos do projeto logo no início, como sugere Cohn (2005) - Figura 6. Esta prática, aliada com as pequenas iterações e com a participação efetiva do cliente com os *feedbacks*, foram pontos chaves para a resolução dos riscos do projeto.

Depois de seis meses de desenvolvimento, o sistema foi efetivamente colocado em produção e disponibilizado para os candidatos que desejavam participar do processo seletivo realizarem suas inscrições com sucesso, superando as expectativas, pois além de superar os riscos do projeto também se conseguiu automatizar vários processos e realizar a integração deste novo produto com um produto financeiro já existente na

[4] Utilizamos a métrica de Pontos por Estórias para estimativas.

Instituição. Como o projeto foi dividido em alguns marcos[5]e o primeiro marco foi o módulo de inscrições, o projeto não se deu por encerrado, mas tudo o que o cliente precisava e que já poderia lhe trazer um retorno foi entregue. A partir de agora serão desenvolvidos novos requisitos, sempre priorizados pelo cliente, entregando-os ao final de cada *Sprint* e, após a sua validação, serão novos incrementos ao produto já existente em produção.

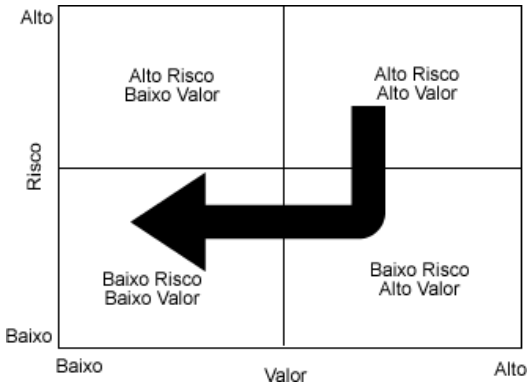


Figura 6. Combinando risco e valor na priorização de requisitos - fonte: [Cohn, 2005].

A métrica *Project Velocity* está relacionada a quanto de software o time consegue entregar por iteração, em relação às atividades planejadas [Cunningham 2007]. A base para medição são as *story points* ou dias ideais de desenvolvimento. Esta métrica é extremamente importante para o time, pois é por meio dela que se consegue fazer um nivelamento de produção e encontrar o *takt time*[6]. Aí está a importância de existir uma iteração *time-boxed*[7]. Por meio do gráfico apresentado n figura 7 foi possível realizar um acompanhamento da evolução do time em termos de estimativas ágeis. Nos *Sprints* iniciais nota-se uma grande variação dos pontos previstos, demonstrando que o time ainda não possuía um padrão para estimativas e estava aprendendo. A partir do *Sprint* 22 o time conseguiu encontrar o seu *takt time* e os acertos nas estimativas se tornaram naturais.

[5] Representa a conclusão de uma fase, passiva de aprovação do cliente.
[6] Termo utilizado no *Lean* para representar o compasso de trabalho do time
[7] Períodos de tempo distintos e fixos, sem variações.

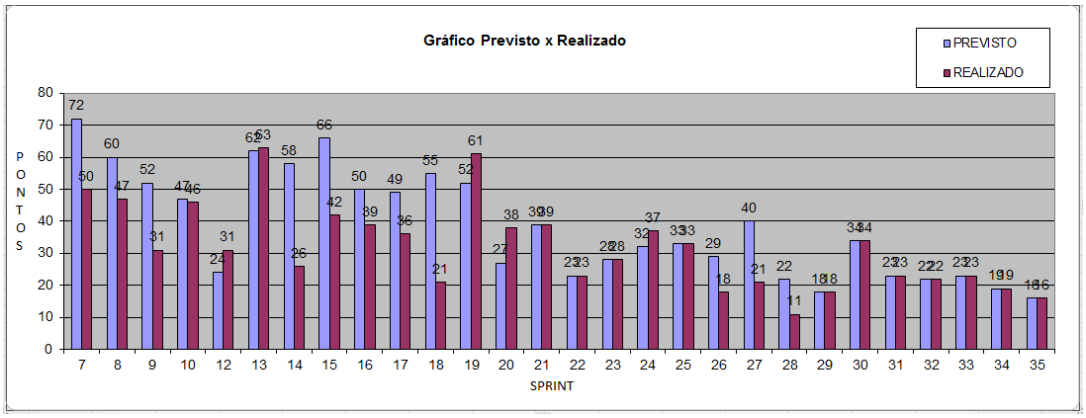


Figura 7. Gráfico Previsto x Realizado (fonte: Dos autores).

4.2.2 Práticas de Engenharia

O *Scrum* nos permitiu adotar um padrão de gestão. Porém, faltava aplicar práticas efetivas para a melhoria da construção do produto. Neste sentido as práticas do XP começaram a ser implementadas. As práticas foram inseridas ao longo do projeto utilizando uma prática chamada Passos de Bebê [Teles 2006], que permitiu que o time assimilasse de forma natural o XP, aumentando os benefícios da sua aplicação.

A primeira prática adotada foi definir alguns padrões em termos de codificação, documentação de código e nomenclaturas para banco de dados. Esta adoção ajudou a criar e manter um bom fluxo das atividades, pois como o padrão foi definido pelo time, não se perdeu tempo no momento em que outros programadores alteravam um código já existente criado por outro membro do time. Outra prática adotada foi o uso do *Subversion*, ferramenta de controle de versão permitindo a edição de trabalho em equipe.

Ao longo do projeto pode-se notar um ganho expressivo em termos de qualidade de código e padronização dentro do time. A cada novo *Sprint* era possível notar essa melhora, por meio das reuniões de revisão do processo, *Sprint Retrospective*, o time sempre buscava discutir e traçar melhorias a serem implementadas na próxima iteração. Para que fosse possível obter este ganho de qualidade ao longo do projeto, sem afetar a produtividade da equipe, foi aplicada outra prática do XP chamada Refatoração, que tem como objetivo alterar o código sem afetar a funcionalidade, tornando o software mais simples de ser manipulado. No início foi difícil para o time absorver esta prática, pois em alguns momentos no desenvolvimento de um requisito eram detectadas possíveis melhorias no código mas, ao invés de realizar pequenas modificações no código ao longo das iterações, acabava-se realizando uma grande refatoração, afetando, consequentemente, a produtividade do *Sprint*.

Uma prática que causou muitas discussões e desconfianças no início foi aPrograma ção em Pares. Desde o primeiro dia que esta prática começou a ser inserida no time ocorreram discussões a respeito de como dois programadores, trabalhando

juntos no mesmo código, iriam render mais que cada um em suas máquinas desenvolvendo sua parte. Para não impor uma prática onde o time não se sentia confortável, apresentou-se aos poucos os benefícios da Programação em Pares, tais como a propagação do conhecimento dentro do time, facilidade em propagar a padronização do código fonte definido pelo time e a diminuição de *bugs*. Após alguns meses realizando pequenas sessões e experimentos os benefícios apareceram. O time atualmente aplica a prática de forma natural realizando rotações entre os pares, fazendo parte de sua cultura (Figura 8).

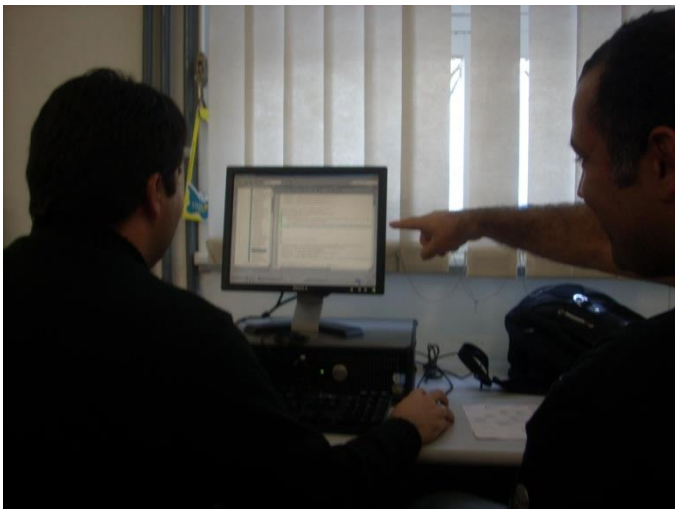


Figura 8. Programação em Par – fonte: Dos autores.

A utilização destas práticas ao longo do projeto só foi possível devido à formação de um time e a criação de um ambiente propício (Figura 9). Além de um ambiente colaborativo, era necessária a utilização de ferramentas que auxiliassem a propagação dos conhecimentos, tal como um *Wiki*, incentivando a contribuição e gerando novos conhecimentos dentro do time. A cada nova iteração era visível a busca do time pela melhoria das práticas através das reuniões de revisão do processo e até mesmo em reuniões informais entre membros do time fora do horário de expediente.

Figura 9. Ambiente de desenvolvimento, célula em forma de U. Isso facilita o trabalho em par, pois não se tem obstáculos e facilita também reuniões rápidas do time – fonte: Dos autores.

4.3 Benefícios e Dificuldades

Neste projeto, o uso das práticas, tanto de gestão de projetos quanto de construção de software, trouxe vários benefícios, apesar de apresentar algumas dificuldades encontradas durante suas implementações. No quadro 3 são relacionados estes benefícios e dificuldades.

Quadro 3. Benefícios e Dificuldades Encontradas

Benefícios	Dificuldades
Confiança e autoestima adquirida pelo time ao longo do projeto.	Mudança cultural na formar de gerenciar e desenvolver software.
Crescimento e nivelamento técnico dentro do time.	Execução de dois papéis, *Product Owner* e *Scrum Master*, em alguns momentos.
Criação de um ambiente colaborativo e motivado.	Conflitos de implementação, devido à utilização de uma metodologia mais tradicional pela matriz.
Antecipação na entrega do software.	Introduzir no time a ideia de acolher mudanças.
Conquista da confiança do cliente.	Disciplina e comprometimento.
Formação de um time multidisciplinar.	Introdução dos métodos ágeis, pois foi uma sugestão "*bottomup*".
Transparência no gerenciamento do projeto.	
Busca natural por novidades sobre métodos ágeis	

pelo time.	
Criação de um padrão de trabalho, passando pela parte de gestão (planejamento, estimativas, priorização) até a parte técnica (padrões de código, métricas; refatoração)	

5. Considerações Finais

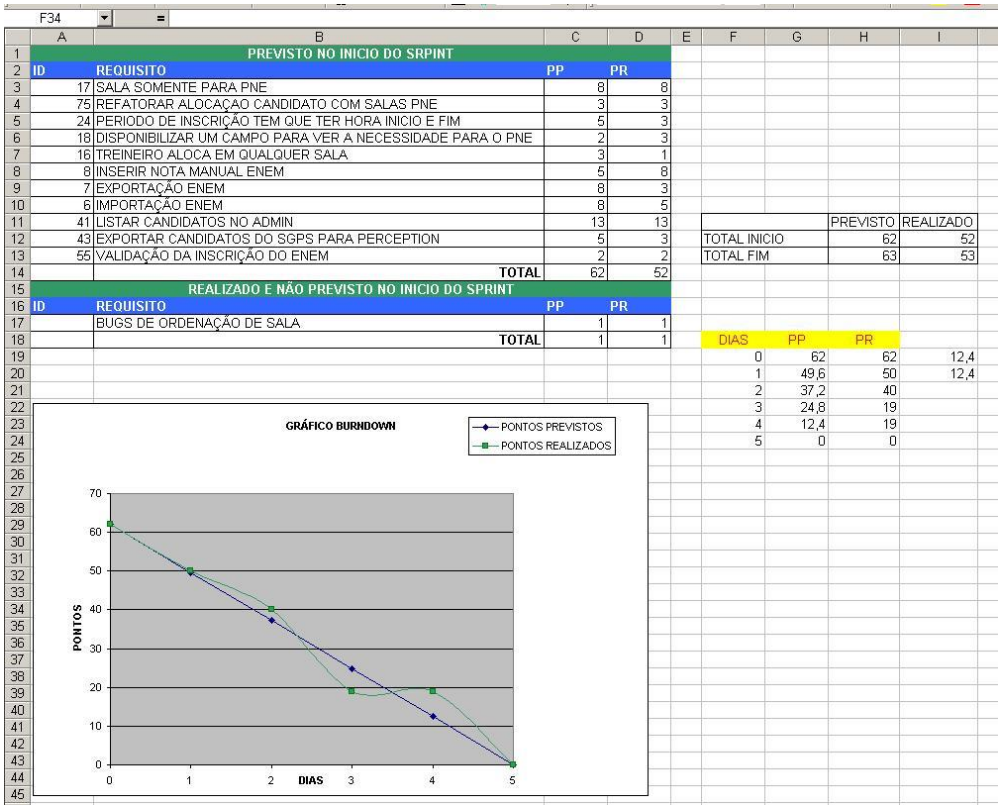
Este trabalho marcou uma importante mudança dentro da instituição onde foi realizado o estudo de caso, envolvendo a forma de gerenciar projetos de software, desde a forma como o projeto era iniciado, planejado e entregue até como as pessoas envolvidas dentro do projeto trabalhavam.

Apesar das desconfianças no início do projeto, devido à proposta de aplicação de uma metodologia desconhecida pela maioria das pessoas envolvidas no projeto, os resultados foram positivos. A principal dúvida desde o surgimento do projeto, foi se era possível desenvolver o software dentro do prazo estipulado. Antes de se utilizar uma abordagem tradicional foi feita uma tentativa de levantar todos os requisitos que o projeto necessitaria, o que durou cerca de dois meses. Estes foram os principais fatores pelo atraso no início do desenvolvimento. Com a introdução do *Scrum*, trabalhando os requisitos sempre priorizados e em pequenas porções, o projeto se tornou cada vez mais claro a cada nova iteração junto com a contribuição dos *feedbacks* do cliente. Aos poucos o cliente foi criando confiança no projeto e na equipe, pois estava em contato diário, passando informações e removendo dúvidas de requisitos que eventualmente a equipe tivesse.

Por meio dos artefatos e reuniões que o *framework* do *Scrum* oferece, foi possível colher métricas para que sempre houvesse a melhoria contínua no processo de trabalho. A utilização do gráfico de *Burndown*, visualizado na Figura 10,foi extremamente importante, principalmente nos *Sprints* iniciais para detectar a velocidade do time, mas também para detectar motivos para a não entrega do que tinha sido planejado na reunião de planejamento. De posse destas métricas e mais os *feedbacks* das pessoas envolvidas, eram discutidas, nas reuniões de retrospectivas, formas de aprimorar o processo. Atualmente o time já possui um padrão para as estimativas e já consegue estimar mais precisamente, conseguindo entregar o que foi planejado no inicio do *Sprint*.

O enfoque deste trabalho foi melhorar a gestão dos projetos relacionados à T.I na instituição, mas não seria possível chegar neste resultado sem utilizar boas práticas de engenharia de software ao qual o *eXtreme Programming* se propõe. Devido à baixa maturidade técnica da equipe e pouco tempo disponível, não foi possível aplicar todas as técnicas disponíveis tais como: Desenvolvimento Guiado por Testes, Testes de Unidade e Integração Contínua. Mas foi possível inserir algumas práticas que auxiliaram diretamente no resultado tais como Código Coletivo e Programação em Par.

Acredita-se que o resultado final deste trabalho foi satisfatório. A primeira entrega foi colocada em produção em seis meses, sendo que anteriormente, utilizando-se uma abordagem tradicional, a estimativa era de cerca de um ano e seis meses. O foco deste trabalho foi abordar primeiramente melhorias no gerenciamento do projeto e, em paralelo, introduzir melhores práticas para desenvolvimento de software para a formação de um time, onde ambas se complementavam.



	ID	REQUISITO	PP	PR
		PREVISTO NO INICIO DO SRPINT		
	17	SALA SOMENTE PARA PNE	8	8
	75	REFATORAR ALOCAÇAO CANDIDATO COM SALAS PNE	3	3
	24	PERIODO DE INSCRIÇÃO TEM QUE TER HORA INICIO E FIM	5	3
	18	DISPONIBILIZAR UM CAMPO PARA VER A NECESSIDADE PARA O PNE	2	3
	16	TREINEIRO ALOCA EM QUALQUER SALA	3	1
	8	INSERIR NOTA MANUAL ENEM	5	8
	7	EXPORTAÇÃO ENEM	8	3
	6	IMPORTAÇÃO ENEM	8	5
	41	LISTAR CANDIDATOS NO ADMIN	13	13
	43	EXPORTAR CANDIDATOS DO SGPS PARA PERCEPTION	5	3
	55	VALIDAÇÃO DA INSCRIÇÃO DO ENEM	2	2
		TOTAL	62	52
		REALIZADO E NÃO PREVISTO NO INICIO DO SPRINT		
	ID	REQUISITO	PP	PR
		BUGS DE ORDENAÇÃO DE SALA	1	1
		TOTAL	1	1

	PREVISTO	REALIZADO
TOTAL INICIO	62	52
TOTAL FIM	63	53

DIAS	PP	PR	
0	62	62	12,4
1	49,6	50	12,4
2	37,2	40	
3	24,8	19	
4	12,4	19	
5	0	0	

Figura 10. Sprint Backlog com Burndown – fonte: Dos autores.

Há muitos processos a serem melhorados e introduzidos, principalmente nas áras de testes e engenharia. Com base nas lições aprendidas, que sempre são revisadas a cada iteração, pretende-se ajustar e introduzir melhorias no processo tanto na continuação deste projeto assim como nos novos projetos que surgirem, sempre na busca da melhoria contínua.

Referências

Ambler, Scott W (2004). "Modelagem Ágil: Práticas eficazes para Programação Extrema e o Processo Unificado". Porto Alegre: Bookman.

Barros, Pablo de et al (2009). "SCRUM: Uma metodologia ágil para projetos WEB com pequenas equipes". Disponível em:

<http://www.eventosufrpe.com.br/jepex2009/cd/resumos/R1106-3.pdf>. Acesso em julho de 2014.

Beck et. al. (2001). "Agile Manifesto". Disponível em: <www.agilemanifesto.org>. Acesso em maio de 2014.

Beck, K. (2008). "Programação Extrema (XP) Explicada: Acolha as Mudanças". Porto Alegre: Bookman.

Boehm, B. A. (1988). "A Spiral Model of Software Development and Enhancement".Disponível em: < http://portal.ou.nl/documents/114964/2986739/T24331_02.pdf>. Acesso em julho de 2014.

Brooks, F. (1987). "No Silver Bullet: Essence and Accidents of Software Enginnering". IEEE Computer, volume 20.

Chapiewski, G. (2008). "Como estamos indo com a adoção de Scrum na Globo.com". Disponível em: <http://gc.blog.br/2008/05/27/como-estamos-indo-com-a-adocao-de-scrum-naglobocom/>. Acesso em Julho de 2014.

Cohn, M. (2005). "Agile Estimating and Planning". Nova Jersey: Prentice Hall.

Cunnigham, W. (2007) "Project Velocity".Disponível em: <http://c2.com/cgi/wiki?ProjectVelocity>. Acesso em Abril de 2014.

Donato, R. (2010). "Fábrica de Software – Definir processo de desenvolvimento". Disponível em: <http://voat.com.br/rdal/?p=76>. Acesso em Maio de 2014.

Kniberg, H. (2007). "Scrum and XP from the Trenches".Canada: C4Media.

Leite, J. (2007). "Engenharia de Software". Disponível em: <http://engenhariadesoftware.blogspot.com/2007/03/o-modelo-transformao.html>. Acesso em Maio de 2014.

Pham, A.; Pham, P. (2012). "Scrum em Ação". São Paulo: Novatec.

Prikladnicki, R.; Willi, R.; Milani, R., Orgs. (2014) "Métodos Ágeis para Desenvolvimento de Software". Porto Alegre: Bookman.

Royce, W. (1970). "Managing the Development of Large Software Systems", IEEE WESCON.

Savoine, M. et. al. (2009). "Análise de Gerenciamento de Projeto de Software Utilizando Metodologia Ágil XP e Scrum: Um Estudo de Caso Prático". Disponível em: <http://pt.scribd.com/doc/51919057/Analise-de-Gerenciamento-de-Projeto-de-Software-Utilizando-Metodologia-Agil-XP-e-Scrum-Um-Estudo-de-Caso-Pratico#scribd>. Acesso em Julho de 2014.

Schwaber, K. (2004). "Agile Project Management with Scrum". Washington: Microsoft Press.

Sommerville, Ian (2007). “Engenharia de Software”. 8. ed. São Paulo: Pearson Addison-Wesley.

Teles, V. (2004). “Extreme Programming: Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade”. São Paulo: Novatec.

Teles, V. (2006). “Passos de Bebê”. Disponível em: <http://improveit.com.br/xp/principios/passos_bebe>. Acesso em Junho de 2014.