UNIVERSIDADE FEDERAL DO PARANÁ
Especialização em Informática – Desenvolvimento Web

**RAFAEL FELIPE PAES**

# SISTEMA *SCRUM* E PROGRAMAÇÃO EXTREMA NO AMBIENTE DE TRABALHO

CURITIBA
2014

# Parecer de Aprovação
# Monografia de Especialização em Informática
# Ênfase em Desenvolvimento de Sistemas para a WEB
# Programa de Pós-Graduação em Informática/UFPR

Declaramos que o aluno **RAFAEL FELIPE PAES** entregou a versão final da sua Monografia de Especialização em Informática da Universidade Federal do Paraná, com Ênfase em Desenvolvimento de Sistemas para a WEB, intitulada ***SISTEMA SCRUM E PROGRAMAÇÃO EXTREMA NO AMBIENTE DE TRABALHO.***

Curitiba, 5 de outubro de 2014

Prof. Dr. Andrey Ricardo Pimentel
Professor Adjunto
Universidade Federal do Paraná
Setor de Ciências Exatas
Departamento de Informática
Caixa Postal 19081
CEP 81531-990 - Curitiba-PR

Prof. Me. Setembrino Soares Ferreira Junior
Professor Assistente
Universidade Federal do Paraná
Setor de Ciências Exatas
Departamento de Informática
Caixa Postal 19081
CEP 81531-990 - Curitiba-PR

**RAFAEL FELIPE PAES**

# SISTEMA *SCRUM* E PROGRAMAÇÃO EXTREMA NO AMBIENTE DE TRABALHO

Monografia apresentada como requisito parcial para obtenção do grau de Especialista em Informática, com ênfase em Desenvolvimento de Sistemas para a Web, ao Curso de Especialização em Informática, Departamento de Informática, Setor de Ciências Exatas da Universidade Federal do Paraná.
Orientador: Prof. Dr. Andrey Ricardo Pimentel

CURITIBA
2014

## RESUMO

Este trabalho tem como objetivo demostrar que com o suporte de um gerenciamento de desenvolvimento ágil e estratégico de *software* é possível facilitar a rotina de trabalho em determinadas empresas e andamentos de seus projetos. Para este projeto serão empregadas duas metodologias de desenvolvimento ágil: Programação Extrema (XP) e *Scrum*, que aqui serão apresentados seus valores, princípios e práticas. Demonstrando um conjunto de planos que tem como objetivo alcançar resultados sólidos com enfoque na visão do projeto em metas de curto, médio ou longo prazo da empresa. As ferramentas metodológicas a serem abordadas serão: pesquisa bibliográfica e virtual. A análise obtida neste projeto poderá ser reutilizada tanto como fonte de referências para profissionais que buscam aprimoramento, aperfeiçoamento e desenvolvimento dos seus serviços, quanto como fonte de pesquisa acadêmica.



**Palavras-chaves:** Programação Extrema (XP), Sistema Scrum. Desenvolvimento Ágil.

## ABSTRACT

The objective of this work is demonstrate with the support of agile development and strategic of software, is possible facilitate the routine of a work in estabilished companies the progress of their projects. To this project will be approached two methodologies in agile management, Extreme Programming (XP) and Scrum, which it will be showed their values, principals and praticals: Demonstrating a set of plans that has with an objective reach solid results with focus in the project view in deadlines of, short, medium, or long inside the company. The methodological tools that will be used are: bibliographic research and virtual. The analisys acquired can be reused as a source of reference for professionals looking for enhancement, improvement or development of their jobs, also as source of research in academic.



**Keywords:** Extreme Programming (XP), Development Scrum, Agile Management.

## SUMÁRIO

# 1. INTRODUÇÃO

Este trabalho tem como objetivo mostrar que com duas metodologias ágeis, *Scrum* e Programação Extrema, conhecida também como XP, trabalhando de forma conjunta, consegue-se beneficiar empresas de pequeno porte que não podem contar com uma equipe extensa em seu quadro de funcionários.

O sistema pode encontrar um grande risco se não obtiver uma metodologia adequada, pois pela sua magnitude tornar-se-ia complexo trabalhar em projetos sem um controle ou planejamento em etapas do que deve ser feito e desenvolvido, podendo até trabalhar em funcionalidades que não serão utilizadas no sistema. Pode também ocorrer, à falta de um gerenciamento próximo aos desenvolvedores de *software*, trazendo à perda de foco no objetivo a ser conquistado.

Como o objetivo deste trabalho é trazer informações que visam atender pequenas equipes, este tende a mostrar como é possível organizar grandes projetos para essas equipes ou até para um profissional em individual, que pode ou não ser supervisionado por um gerente de projetos trabalhando conjuntamente com o público alvo do sistema.

Assim, foi determinado o uso da metodologia ágil *Scrum* para organizar grandes projetos e Programação Extrema - XP para controle de pequenas ou médias equipes. Com o uso dessas duas ferramentas integradas são obtidos benefícios mútuos, que trazem a fusão dos prós destes dois sistemas funcionais. Além disso, podemos contar com mais funcionalidades, que serão abordadas no decorrer deste trabalho.

## 1.1 PROBLEMA

Devido à falta de qualidade diante dos projetos desenvolvidos nas empresas, à ausência de profissionais qualificados ou disponíveis para desempenhar tal atividade, foi determinada a necessidade de encontrar um meio que visasse facilitar a organização encontrando um planejamento estratégico para cumprir este objetivo.

Existia uma grande dificuldade nos projetos em encontrar maneiras de priorizar as atividades a serem desenvolvidas muitas vezes por indecisão do cliente, e quando estas finalmente eram priorizadas e enviadas para o desenvolvedor, acontecia de serem interrompidas pela metade por outras atividades que surgiam no dia-a-dia, ocasionando então na perda do raciocínio do profissional e também do tempo total do desenvolvimento no sistema/projeto, que neste caso o programador ainda poderia perder tudo que havia construído até o momento, pois a adaptação seria construída de forma diferente.

Os problemas descritos acima são fatores que dificultam e incomodam o andamento das atividades rotineiras e projetos mais complexos pois, por conta das diversas paralisações de usuários e gestores do sistema, era optado por desenvolvimentos de dimensões menores e que não trariam valores à empresa.

Isto tudo se deve à falta de tempo, planejamento estratégico, organização de prioridades, quadro de funcionários reduzido, qualidade das funções desempenhadas e várias atividades realizadas num mesmo momento.

Muitas equipes e empresas esquecem que atualmente existem sistemas funcionais que auxiliam e contribuem para o gerenciamento do sucesso de um projeto.

## 1.2 OBJETIVO GERAL

Levando em consideração todos os problemas expostos, o objetivo dessa monografia é a análise dos dois tipos de metodologias ágeis, *Scrum* e Programação Extrema e verificação de quais das funcionalidades melhor se

aplicam para determinada atividade no processo de análise e ou desenvolvimento do produto, assim facilitando o desenvolvimento e a organização de uma forma clara e objetiva para todas as partes envolvidas, tendo um projeto otimizado, com intuito de trazer experiência para toda a equipe, com um objetivo final que deve ser alcançado, trazer a satisfação do cliente para com o produto.

## 1.3 OBJETIVOS

Os pontos a serem trabalhados deverão ser o levantamento de requisitos ou objetivos que devem ser desenvolvidos perante o cliente, trazendo todas as funcionalidades que devem ser realizadas ou não, buscando assim a experiência da equipe ou do profissional através de projetos finalizados anteriormente.

Revisar as dúvidas que os profissionais têm perante as necessidades que devem ser realizadas dentro da equipe ou diretamente com cliente específico do foco da atividade, determinando quais atividades devem ser trabalhadas e em qual nível de prioridade será trabalhado, passando um levantamento em horas da atividade a ser desenvolvida, sendo estimado este por práticas que vão ser apresentadas ao longo deste trabalho.

Facilitar ao longo do desenvolvimento os impedimentos que podem vir de forma externa ou interna à equipe.

Trazer gráficos e quadros de como está indo o incremento da atividade que está sendo desenvolvida no momento, assim comunicando toda a equipe e o cliente, sem a necessidade de informativos ou reuniões duradouras.

Realizar testes do produto antes de ser entregue ao cliente com possíveis *bugs*.

Reunir todos os prós e contras do projeto desenvolvido e entregue, realizando então um *feedback* para que possa ser executado um melhor sistema no próximo projeto a ser concluído na fila ou em outro projeto.

## 1.4 JUSTIFICATIVA

Com a utilização destas duas ferramentas pode ser facilitado o desenvolvimento das atividades, trazendo aprimoramento nas tarefas a serem desenvolvidas, colocando um produto sustentável e reduzindo custos operacionais para as empresas que não podem contar com uma grande equipe em seu local de trabalho ou que o ramo delas seja diferente da área de tecnologia.

Outro fato importante é que os clientes estarão com um produto claro e que pode ser disseminado através de outras áreas da própria empresa, pois as ferramentas tendem a analisar pontos que podem ser colocados à disposição do cliente que trarão muitos benefícios e facilidades para com as rotinas de trabalho de seus solicitantes.

Além disso, diante de todas essas questões apresentadas, será possível desenvolver um novo sistema com prazos definidos, robusto e de forma ágil, trazendo grande experiência e riqueza para toda a empresa.

## 1.5 ORGANIZAÇÃO DO DOCUMENTO

Este trabalho é composto por quatro capítulos:

O primeiro capítulo corresponde à introdução deste estudo, onde é apresentado o tema, a delimitação do problema, a justificativa do estudo, os objetivos gerais e específicos e a estruturação do trabalho.

No segundo capítulo é abordado o referencial teórico deste estudo, que aborda conceitos, considerações e informações.

No terceiro capítulo é feita uma análise do desenvolvimento dos sistemas de *software* propostos, as diferenças, prós e contras da utilização de cada um.

O quarto capítulo é composto pelas considerações finais deste estudo, bem como sugestões e recomendações.

# 2 CONCEITOS BÁSICOS

## 2.1 DESENVOLVIMENTO ÁGIL

Conforme definições o desenvolvimento ágil é um conjunto de métodos para o desenvolvimento de *software* ou para ajudar outras pessoas a desenvolvê-lo de uma forma que traga consigo conceitos da engenharia do *software* (BASSI; GOLDMAN, 2006).

Seu sistema consiste em um modelo incremental em que cada parte do ciclo completa todas as funcionalidades de requisitos (escopo), análise, programação, testes e liberação do *software*. Estes sempre completando uma parte funcional do sistema para que ele possa trazer algum valor para o produto final. (BISSI, 2007).

Dentre do desenvolvimento ágil existem diversos *frameworks* no mercado que podem atender as empresas, cada um contendo suas particularidades, alguns deles tratando mais do gerenciamento e outros tratando mais da engenharia. Dentre estes produtos temos: *DSDM*, *OpenUP*, *FDD*, *TDD*, *Programação Extrema* (XP) e *Scrum*, estes dois últimos que serão tratados aqui neste trabalho.

## 2.2 MANIFESTO ÁGIL

Segundo referências consultadas, para a criação do desenvolvimento ágil foi realizada uma reunião entre alguns responsáveis, dentre eles: Kent Beck, Dave Thomas, Martin Fowler e Jon Kern, para debater sobre as melhores práticas de gerenciar o desenvolvimento de *softwares*; após este encontro sairia o manifesto ágil.

Para melhor explicar sobre o que aconteceu foi utilizado como fonte o próprio manifesto, que nos diz:

"*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*"

Além destes, foram criados os doze princípios de um desenvolvimento ágil, conforme abaixo:

*"Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor.*

*Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos Ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.*

*Entregar software funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos.*

*Pessoas relacionadas a negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.*

*Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.*

*O Método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara.*

*Software funcional é a medida primária de progresso.*

*Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter, indefinidamente, passos constantes.*

*Contínua atenção à excelência técnica e bom design aumenta a agilidade.*

*Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.*

*As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis.*

*Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo."* (BECK, K. et.al., 2001)

## 2.3 *SCRUM*

Conforme referências, o nome *scrum* vem da formação do *rugby,* aonde é uma formação ordenada, que geralmente é utilizada após uma jogada irregular. Para realizar essa tática é necessário formar uma muralha defensiva, aonde a participação de todos os jogadores é importante e caso um venha a falhar toda a jogada é comprometida *(*BASSI; GOLDMAN, 2008).

Por isso o *scrum* determina que há um conjunto de práticas que auxiliam uma equipe a entregar partes do produto em um ambiente instável e desafiador.

Para organizar o *scrum* temos três papéis fundamentais, que devem ser muito bem trabalhados, em que todos devem confiar e ajudar mutuamente um ao outro. Eles são:

Dono do Produto: Representa a parte interessada no produto, o cliente, que detêm toda a informação necessária para o desenvolvimento do *software.* Ele tem que ter todo o conhecimento negocial do sistema, e também saber organizar a lista de prioridades que os desenvolvedores devem atuar, conhecida também como produto da *backlog*.

*Scrum Master*: Facilita para que todos os impedimentos sejam removidos da equipe que irá trabalhar, assim deixando a equipe focada em seu trabalho; também garante que todos os procedimentos do *scrum* sejam colocados em prática conforme determinado pela metodologia.

Equipe de Desenvolvimento: Geralmente composta de cinco a nove pessoas, são indivíduos com habilidades multifuncionais e que podem trabalhar em qualquer etapa do desenvolvimento, geralmente auto-organizada e auto conduzida, são elas que vão colocar o que foi descrito pelo Dono do Produto em programa funcional.

## 2.4 PROGRAMAÇÃO EXTREMA - XP

Para Teles (2004), a Programação Extrema é a junção de bons princípios e práticas de desenvolvimento de *software*. Sua criação se deu em 1996, ano em que Kent Beck começou a trabalhar no desenvolvimento de um grande sistema de folha de pagamento, chamado *Chrysler Comprehensive Compensation* (Sistema de Compensação Abrangente da *Chrysler*), ou C3. Apesar do longo processo de criação e evolução dos princípios e práticas usados da XP, foi durante o desenvolvimento do C3 que Kent Beck pela primeira vez os reuniu e aplicou de forma harmônica e coesa num grande projeto. Além disso, foi durante este projeto que tanto a metodologia quanto as práticas iniciais foram batizadas com os nomes atuais.

A Programação Extrema, do inglês *Extreme Programming*, é um método tecnológico baseado em requisitos vagos, que se modificam frequentemente. É uma metodologia cujo objetivo é cumprir com expectativas e satisfação do cliente, destaca o desenvolvimento rápido do projeto de melhor qualidade, que são produzidos em menos tempo, de forma mais econômica e com menos estresse para os desenvolvedores e clientes que o habitual. Os quatro valores que norteiam a XP são: comunicação, simplicidade, *feedback* e coragem (BECK, 2004). É mais voltada a equipes pequenas e médias que desenvolvem *softwares.*

Abrange as idéias dos profissionais de desenvolvimento em atividades que determinam resultados rapidamente na forma de *software* alinhado e de acordo com as necessidades de seus usuários finais. Dessa forma, organizando e facilitando as atividades, combinando técnicas eficazes e abolindo atividades redundantes. Reduz o risco dos projetos desenvolvendo *software* de forma iterativa e reavaliando permanentemente as prioridades dos usuários.

A equipe de desenvolvimento, antes de começar a utilizar essa metodologia, deve conhecer e compreender os valores e princípios que justificam e direcionam o uso desta prática, se essas por sua vez são claras e concretas.

As práticas são ações concretas que objetivam facilitar um percurso, mas sem deixar claro o porquê de sua importância.

Os valores são fatores gerais e abstratos utilizados para justificar o que se vê, pensa ou faz, ou seja, são excessivamente abstratos para guiar as práticas em contextos específicos. Representam também a essência daquilo que aceitamos ou não. Ter os valores explícitos é importante, pois sem valores as práticas perdem sentido e tornam-se atividades sem utilidade, sem propósito, sem objetivo. Os valores direcionam o comportamento da equipe de desenvolvimento. São quatro valores essenciais que guiam o desenvolvimento de um software; Comunicação, Simplicidade, Coragem e Respeito.

Os princípios servem de elo entre os valores e as práticas. Os princípios são diretrizes direcionadas a comandos específicos. Além disso, os princípios podem ser utilizados para improvisar práticas complementares quando não for possível encontrar nenhuma prática que satisfaça ao que se deseja.

Juntando valores, práticas e princípios da XP tem-se como objetivo comunicar o que é necessário pensar e praticar no desenvolvimento de *software*.

As práticas da XP também buscam balancear as demandas dos negócios com as necessidades pessoais dos desenvolvedores.

Nesta metodologia as pessoas são fundamentais para o método de desenvolvimento; dessa forma, suas práticas são voltadas para potencializar o melhor que as pessoas têm para oferecer bem, como suprimir suas falhas. As atividades características do desenvolvimento são executadas simultaneamente e não em fases. Durante todo o projeto, a finalidade é manter um fluxo constante de pequenas entregas, gerando valor ininterruptamente, na forma de *software* funcional. Dessa forma, a XP trabalha com pequenas funcionalidades representadas sem muito detalhamento através de histórias. A cada atualização é colocado um pequeno conjunto de histórias, de forma que estas possam ser detalhadas, implementadas e colocadas à disposição dos usuários rapidamente, fazendo o sistema evoluir de forma natural e segura.

Apresenta como finalidade o princípio da comunicação, cujo objetivo é melhorar o relacionamento entre clientes e desenvolvedores, preferindo conversas pessoais a outros meios de comunicação. Também incentiva a comunicação entre

desenvolvedores e o gerente do projeto. Segundo Beck (2004), a metodologia XP baseia-se em 12 práticas, conforme abaixo:

- Planejamento: há interação entre programadores e o cliente, é onde o cliente descreve a respeito das funcionalidades que deseja do sistema. Estabelecem o que deve ser feito, os prazos e as prioridades de cada atividade.
- Entregas frequentes: a construção do *software* deve ser simples e funcional; conforme os requisitos surgem, há uma atualização e novas versões.
- Metáfora: são as descrições do *software* sem os termos técnicos, facilitando o entendimento do cliente e guiando o desenvolvimento de *software*.
- Projeto simples; o programa desenvolvido pelo método XP deve ser o mais simples possível e satisfazer os requisitos atuais, sem a preocupação de requisitos futuros.
- Testes: a XP focaliza a validação do projeto durante o processo de desenvolvimento, fazendo com que os programadores desenvolvam o *software* criando primeiro os testes. Estes devem funcionar como guias para o desenvolvimento.
- Reconstrução: há reestruturação do sistema para remover a duplicação, melhorar a comunicação, simplificar ou adicionar a flexibilidade.
- Programação em pares: todo o código é escrito com dois profissionais programadores em uma máquina.
- Propriedade Coletiva: todo o código poderá ser modificado em qualquer lugar no sistema, a qualquer hora.
- Integração Contínua: evita preocupações futuras, mantem o sistema integrado ininterruptamente.
- Semana de 40 horas: trabalho de 40 horas na semana em regra geral.
- Cliente junto aos desenvolvedores: incluir pelo menos um cliente na equipe, para que este possa sanar as dúvidas e questionamentos que venham a existir.

- Padronização do código: os desenvolvedores escrevem todo o código conforme as regras estabelecidas pela equipe inteira.

## 3. DESENVOLVENDO O *SCRUM* E XP

Conhecer as metodologias *Scrum* e XP é de extrema importância para profissionais que trabalham com desenvolvimento de *software*. Tal conhecimento contribui para a escolha da metodologia mais adequada a cada projeto, dessa forma definindo o êxito ou o fracasso. No entanto, estas metodologias podem ser utilizadas em conjunto.

A Programação Extrema determina uma metodologia de desenvolvimento ágil destinada a pequenas e médias equipes. O foco principal são as atividades de desenvolvimento.

As principais características da metodologia XP que diferem dos métodos tradicionais são:

- *Feedback* constante;
- Abordagem incremental;
- Comunicação encorajada entre pessoas;
- Presumir simplicidade;
- Trabalho de alta qualidade.

O *Scrum* é um *framework* focado em práticas de planejamento, organização e gerência de produtos complexos. Define uma abordagem para o gerenciamento das atividades considerando o processo de desenvolvimento iterativo e incremental.

Principais características:

- Flexibilidade
- Trabalho em equipe
- Comunicação

Princípios

- Visibilidade
- Inspeção
- Habilidade

## 3.1 COMO INICIAR O DESENVOLVIMENTO

Primeiramente precisamos ter uma ideia do que é necessário desenvolver, conhecer qual é o problema do cliente, o que ele necessita, ou seja, uma visão geral do produto. Nesta fase já pode ser criado o *backlog* do produto, que no *scrum* é um papel realizado pelo dono do produto, mas quando é trabalhado pela primeira vez, é importante que todo o *scrum team* esteja presente, ou seja, *scrum master* e time de desenvolvimento, que podem apresentar ao dono do produto o que realmente é necessário realizar para que o sistema diminua suas falhas e impedir os problemas ocorridos. Porém, existem atividades que são de responsabilidade do cliente, como determinar qual é a prioridade das atividades a serem desenvolvidas, adicionar ou remover itens do *backlog,* podendo trazer um nível de detalhe para o time de desenvolvimento.

Outro ponto importante nesta reunião é que seja definido que todos os envolvidos devem trabalhar de forma conjunta. O *feedback* imediato deve ser trabalhado de forma confiante e direta, e as reuniões pessoais devem ser tratadas em prioridade ao uso de e-mails, telefonemas e chats corporativos, pois esses podem trazer um prejuízo ao entendimento do negócio a ser tratado.

Para exemplificar, usamos o seguinte exemplo da visão do produto, tendo como uma ideia um projeto no qual:

- Uma rede de hospitais que está presente em seis estados no país, com dez unidades em cada.
- Alguns problemas existentes no *software* da empresa:
    - Não existe uma integração dos pacientes que são atendidos em um determinado hospital, sendo necessário realizar o cadastro caso tenha sido feito em outra filial.
    - Os médicos não têm uma ordem cronológica de seus pacientes chegando para serem atendidos, o que ocasiona pessoas chegando e não sendo atendidas, pois o sistema não envia um alerta corretamente.

- O histórico dos pacientes não é realizado de forma adequada.
- Esta rede deseja melhorar a qualidade de seus serviços, diminuindo assim o tempo de atendimento para com seus pacientes, podendo atender mais pessoas em um mesmo dia.

Com as informações coletadas é possível então ter uma ideia do *backlog,* conforme o que foi levantado até aqui.

QUADRO 1: BACKLOG DO PRODUTO

| Número | Item | Prioridade |
|---|---|---|
| 1 | Integrar cadastros. | 900 |
| 2 | Corrigir falha no alerta da fila ao médico. | 800 |
| 3 | Criar um histórico de problemas dos pacientes. | 750 |
| 4 | Automatizar a chamada de pacientes. | 600 |
| 5 | Permitir a emissão de relatórios gerenciais, como pacientes atendidos em um determinado período. | 450 |
| 6 | Emitir receitas de remédio e atestados. | 300 |
| 7 | Criar controle a exames médicos realizados em outro departamento. | X |

## 3.2 COMO SELECIONAR A PRIORIDADE

Com a prioridade de cada atividade definida, o dono do produto pode passar com mais detalhes as que estão no topo da lista para o time de desenvolvimento e com menos detalhes as da parte de baixo; com isso, deve ser sempre priorizado tendo como base:

- Valor do retorno do investimento para empresa.
- Dependência entre os módulos a serem construídos.
- Necessidades.
- Riscos.

Com isso os itens prioritários são desenvolvidos pela equipe e nessa parte o dono do produto adiciona novos itens para serem inseridos no *backlog* e que podem ser reordenados a qualquer momento. Contudo, neste ponto é importante especificar que uma vez iniciado o desenvolvimento é essencial que não haja mais alterações na priorização, pois muito tempo é desperdiçado parando o que foi desenvolvido para início de uma nova etapa e depois mais tempo se gasta para voltar ao desenvolvimento anterior.

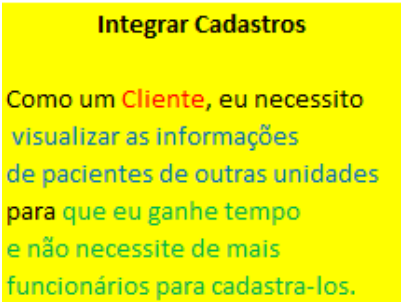
## 3.3 HISTÓRIAS DE USUÁRIO

Como alternativa ao *use case,* a história de usuário vem para descrever os requisitos de forma clara, objetiva, simples e eficaz. Essa técnica que surgiu no XP nos mostra o ponto a ser realizado sem muitos detalhes e que requer pouca manutenção, mas neste ponto é importante trabalhar de forma rápida e resumida, pois existem casos de uma história conter várias páginas, diminuindo assim sua eficiência.

As histórias ajudam a transformar um problema em pequenas partes, que irão ajudar no processo a ser realizado, possibilitando também um desenvolvimento interativo, permitindo que a equipe faça pequenas entregas enquanto interage com o cliente. Além de ajudar a facilitar o trabalho, pois os *uses cases* trabalham de forma mais complexa e são difíceis de trabalhar entre as equipes, pois suas técnicas são fáceis de se perder em detalhes de aplicações; geralmente é necessário ainda separar analista de desenvolvedores.

Para criar as histórias são usados cartões de tamanhos pequenos, geralmente *post-it,* que são fixados em quadros ou paredes, para toda a equipe poder visualizar com facilidade, e nela são descritos três fatores importantes, que são o autor, que é o proprietário da história ou interessado, podendo ser uma área / setor da empresa, a ação, que menciona o motivo pelo qual o autor deseja que seja realizado para alcançar sua meta, e o objetivo, que justifica o porquê da realização de tal ação e o que vai colaborar para o sistema. Com essas informações podemos criar a história conforme demonstrado abaixo:

FIGURA 1: HISTÓRIAS DE USUÁRIOS



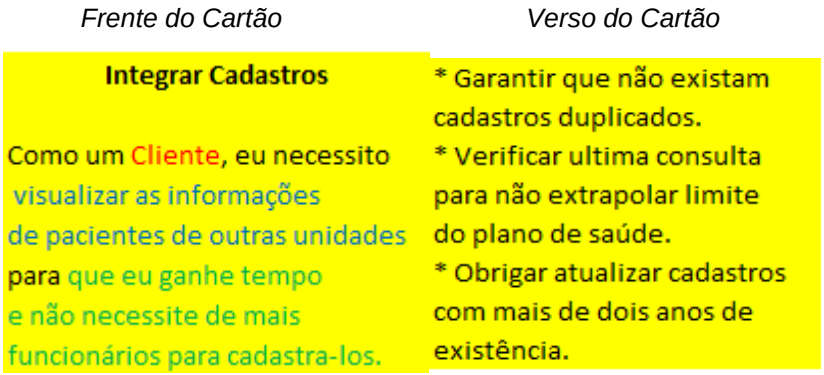

## 3.4 TESTE DE ACEITAÇÃO

Os testes de aceitação têm como finalidade confirmar que sua funcionalidade execute conforme esperado pelo cliente, o que garante que uma determinada condição seja verdadeira para que esteja concluída e está diretamente associada à história do usuário, ou seja, ela é um requisito do sistema.

Estes vêm a ser escritos no verso do cartão da história do usuário, trazendo em seu rodapé duas informações importantes, sendo definida a prioridade e a complexidade do produto pelo dono, que será estimado pelo *planning poker* e realizado pela equipe de desenvolvimento.

Para melhor visualizar segue o exemplo abaixo, para determinada história de usuário.

FIGURA 2: HISTÓRIAS DE USUÁRIOS – FRENTE E VERSO



## 3.5 ESTIMAR A COMPLEXIDADE - *PLANNING POKER*

O *planning poker* ajuda a estimar uma determinada tarefa e traz uma ideia da sua complexidade, ela não traz um valor em horas e apenas um comparativo. Sua ideia vem em cartas baseadas na sequência de Fibonacci, que é uma sucessão de números que aparecem em diversos fenômenos da natureza, que consiste em uma sequência infinita de números em que o próximo é resultado da soma dos dois números anteriores.

Dessa forma, pode ser utilizado um baralho para orientar na realização estimativa da seguinte forma: colocando as cartas em cima de uma mesa com as numerações como 0, 1, 2, 3, 5, 8, 13, 20, 40 e 100, e cada desenvolvedor terá o seu maço. Então o dono do produto será responsável em dizer qual história de usuário será estimada, desse modo o time de desenvolvimento trabalha em sua complexidade, mas ainda não o revelam, ficam aguardando o sinal para mostrar todas as cartas ao mesmo tempo. Quando existem divergências, os desenvolvedores que jogaram as cartas com maiores e menores valores lançados devem explicar os seus motivos, levantando questões que podem contribuir para os demais; caso todos estejam de acordo é levantado qual é a carta ideal e estimada a próxima história de usuário. Assim é possível comparar a história anterior com a próxima. Caso haja divergência nas questões é realizado o levantamento novamente.

Após conseguir realizar a estimativa de pontos é possível então uma estimativa de horas para a história de uso corrente, assim verificando as histórias a serem colocadas no próxima *sprint*, o que será discutido mais abaixo. Com a quantidade de pontos em mãos e ao longo do tempo podemos determinar quantos são necessários para a entrega.

## 3.6 *TIME-BOX*

*Timebox* é um período acordado e fixo para realização de uma determinada atividade com um objetivo a ser alcançado pela equipe. Para *o Scrum* e o XP os

eventos são todos definidos com uma duração constante e suficiente para executá-los, o *Sprint* discutido anteriormente é um evento *time-boxed*.

## 3.7 *SPRINT*

*Sprint,* que tem seu nome originado do inglês e traduzido como corrida de velocidade, trata de uma iteração no ciclo de desenvolvimento e tem como base o *backlog* do produto que foi priorizado pelo dono do produto, seu *time-box* dura de duas semanas até quatro semanas, o qual transforma o item da lista em funcionalidades para o sistema, que incrementará o *software* no final das quatro semanas.

Para a contagem dos dias não podem ser subtraídos os feriados e finais de semana, ou seja, são dias corridos. Além disso, não pode ser executado em um tempo maior que já estimado, ou seja, não mais que quatro semanas. Geralmente a decisão é tomada dessa forma para que tenha um *feedback* constante do dono do produto e *stakeholders* para que não haja desperdício de tempo em funcionalidades que tenham o escopo a ser alterado ou que o sistema venha de uma forma diferente do que o solicitado. Com isso um *software* funcional e finalizado pode conter diversos *sprints* ao longo de sua duração, sendo realizado um em seguida do outro, sem pausas.

Um *sprint* é composto de alguns artefatos e eventos, que serão descritos abaixo; todos elas deverão ser compostos dentro do tempo já discutido anteriormente.

### 3.7.1 Planejamento do *Sprint*

Sendo o primeiro evento oficial do *scrum,* nele vai ser discutido o que será desenvolvido e como será desenvolvido, nele é importante todos os participantes trabalharem de forma colaborativa, seu *time-box* é oito horas para um *sprint* de quatro semanas; caso seja duas semanas, seu *time-box* cai para quatro horas.

Ele será dividido em duas partes para discutir: o que será feito e como será feito.

Na primeira reunião, sendo a metade do *time-box,* todos da equipe *scrum* participam e nela serão definidos a priorização do *sprint* com a análise dos itens do *backlog* do produto e a definição da meta do *Sprint.* É necessário que o dono do produto venha organizado com a lista de prioridades através de histórias de usuário, requisitos ou casos de uso. Assim serão tratados os itens mais importantes e o time de desenvolvimento tentará entender o que deve ser feito; eles também podem ajudar o dono do produto a organizar os itens do *backlog* quebrando-o em partes menores, ou até mesmo organizando-o em um mesmo item; nesta parte é comum que o dono do produto explique sobre o *sprint* atual e mais alguns próximos *sprints.*

Aqui será definido, a partir do *planning poker,* a complexidade em pontos de cada item do *backlog* do produto; deste modo, o time calcula, através da soma dos pontos, quantas atividades eles conseguem entregar dentro de um *sprint*; mesmo tendo uma negociação do dono do produto, somente o time pode responder por sua capacidade de desenvolvimento. A partir desses pontos é definida a meta do *sprint*, o que torna a equipe focada para tê-la como objetivo e deixa claro o porquê é realizado o trabalho. Caso seja percebido algum problema ao longo do caminho, deverá ser avisado diretamente o dono do produto para trabalhar em alternativas para solucioná-lo.

Na segunda parte da reunião a presença do dono do produto é opcional, discute-se sobre como será realizado o programa. Nela serão tratados o tema do planejamento do *Sprint*, com o do *backlog* do *sprint* a partir do *backlog* do produto e histórias do usuário, e a estimativa em horas do seu *backlog*.

Ela se inicia logo que finaliza a primeira reunião e o grande desafio é como implementar as funcionalidades dos itens trazendo como resultado o *backlog* do *sprint*, nela podem ser discutidos assuntos de como serão feitas a análise, construção, testes e demais atividades; também será realizada a divisão dos itens do *backlog* para tarefas menores.

Neste ponto vai ser estimada a quantidade de horas para o *sprint* através da estimativa da complexidade em pontos da reunião anterior. Para os primeiros dias do *sprint* é importante que a atividade não dure mais que um dia, podendo ao

longo do tempo organizar melhoras nas atividades que venham a ser mais complexas; neste ponto é importante que o dono do produto seja avisado caso a equipe atrase a entrega do programa para que o problema seja solucionado a tempo e, da mesma forma, se o time verificar que vai poder entregar antes do previsto.

### 3.7.2 *Backlog do Sprint*

É uma lista de atividades a serem desenvolvidas em um *sprint* pela equipe de desenvolvimento, em que somente eles devem administrá-lo e gerenciá-lo, e todos devem ter acesso e visualizá-lo sempre que preciso.

No *backlog* deverão estar presentes todos os itens ordenados pela prioridade e com sua estimativa de tempo conforme imagem abaixo. Aqui também podem ser usados o quadro de tarefas junto com o gráfico *burndown*:

QUADRO 2: BACKLOG DA SPRINT

**Item do *backlog* do Produto: Integrar cadastro**

| **Prioridade** | **Item** | **Estimativa** |
|---|---|---|
| 1 | Criar rotina de atualização para cadastros desatualizados. | 8h |
| 2 | Criar função para trazer todos os cadastros duplicados e excluí-los. | 8h |
| 3 | Criar tabela única de cadastros, vincular ao programa. | 12h |
| 4 | Testar os itens 1,2,3 | 6h |
| 5 | Alterar o sistema para complementar apenas uma janela para todas as filiais. | 20h |
| 6 | Criar uma tabela com as informações das filiais agrupadas. | 10h |
| 7 | Testar itens 6 e 7. | 10h |

### 3.7.3 Reunião Diária

É um evento que deve acontecer todos os dias no período do *sprint.* Ela é *time-boxed* em quinze minutos e devem estar presentes o *scrum master*, o qual guia os outros membros para que não extrapolem o tempo, junto com a equipe de desenvolvimento, que deverá responder três perguntas básicas:

- O que eu fiz desde a última reunião?
- O que eu vou fazer até a próxima reunião?
- Tem algum problema ou impedimento em meu trabalho?

Esta última pergunta, caso seja identificado algum problema, é trabalho do *scrum master* identificá-lo e corrigi-lo o mais breve possível; caso seja algo que algum outro integrante tenha conhecimento, poderá falar na hora da reunião que poderá ajudá-lo quando a reunião acabar.

O objetivo dessa reunião é trazer uma ideia a todos os integrantes do time de desenvolvimento do que cada um está fazendo e o que está acontecendo fora do seu trabalho atual, ou seja, apenas uma sincronização de informações; nisso podemos realizar a inspeção e adaptação dos processos de forma rápida e segura, melhorando a comunicação de todos.

O importante para essa reunião é que seja em um mesmo local aonde todos podem ficar em pé, pois a reunião é breve, em um mesmo horário, de preferência no primeiro horário de trabalho da manhã para não interromper o trabalho ao longo do dia.

### 3.7.4 Programação em Par (ou em pares?)

A “Programação em Par” ou a “*Pair programming*” é uma das práticas mais conhecidas e usadas pelos que adotam a metodologia *Extreme Programming*. Consiste na prática de dois desenvolvedores trabalharem na mesma máquina para a mesma tarefa. É sem dúvida uma estratégia de desenvolvimento que causa mais receio no universo de desenvolvimento de *software*. Ela passa a sensação

que uma tarefa que poderia consumir horas de um profissional acaba consumindo de dois. Porém essa prática pode motivar benefícios:

- Aprendizado: simplifica o aprendizado de determinada tecnologia ou regra de aplicação. Troca de conhecimento.
- Concentração: os pares são menos interrompidos, focam nas atividades o tempo todo. Não se distraem com e-mails pessoais no meio do desenvolvimento do projeto. Melhor aproveitamento do tempo e mais produtividade.
- Agilidade e Qualidade: Resultados e soluções são pensadas e organizadas de maneira mais rápida e com mais qualidade quando as atividades são realizadas por outros profissionais.

### 3.7.5 Testes do Sistema XP

Com base nos testes os programadores têm condições de avaliar e efetuar mudanças no desenvolvimento dos projetos. Dá segurança e confiança ao grupo.

São fundamentos mais importantes de XP. Podem ser feitos das seguintes maneiras:

- Testes de Unidade ou *Unit tests* - São escritos pelos desenvolvedores enquanto codificam o sistema, antes da implementação (TDD); é um meio de comunicação, tem que ser rápido de executar e automatizado, devem ser criados para todas as classes do sistema.
- TDD - Desenvolvimento guiado por testes ou *Test Driven Development* - É mostrado ao desenvolvedor do projeto que, antes de criar a funcionalidade, deve-se criar um teste da funcionalidade. Primeiramente, é preciso criar um teste falho, a funcionalidade ainda não existe, desta forma, o teste deve retornar um erro. Posteriormente, será desenvolvido o código correto para fazer com que o teste esteja certo; uma vez que o desenvolvedor sabe quais funcionalidades deve programar, fica mais prático o seu desenvolvimento. Por último deve ser feita a refatoração, ou seja, melhorar a codificação.

  No desenvolvimento guiado por testes, acontece um ciclo de repetições:

- Desenvolvimento do projeto
- Teste
- Implementação
- Teste

- Refatoração ou *Refactoring* - é o processo de reestruturar o sistema, sem alterar suas funcionalidades originais.

  O código é alterado de forma clara e objetiva, gera maior flexibilidade, de maneira a acolher mudanças sem grandes impactos, onde todos que olham o conseguem entender, não causa nenhuma modificação na funcionalidade do sistema. Pode ser aplicado no desenvolvimento e manutenção do *software*, possibilita a inclusão de uma nova funcionalidade, evitando a reestruturação do código.
- Testes de Aceitação ou Testes de Funcionalidade - são escritos pelo cliente para testar a funcionalidade do sistema; no momento da escrita da história, este sabe quando a funcionalidade é terminada. Para cada história deve existir um teste de aceitação. Uma história só é finalizada quando os testes de aceitação passarem.

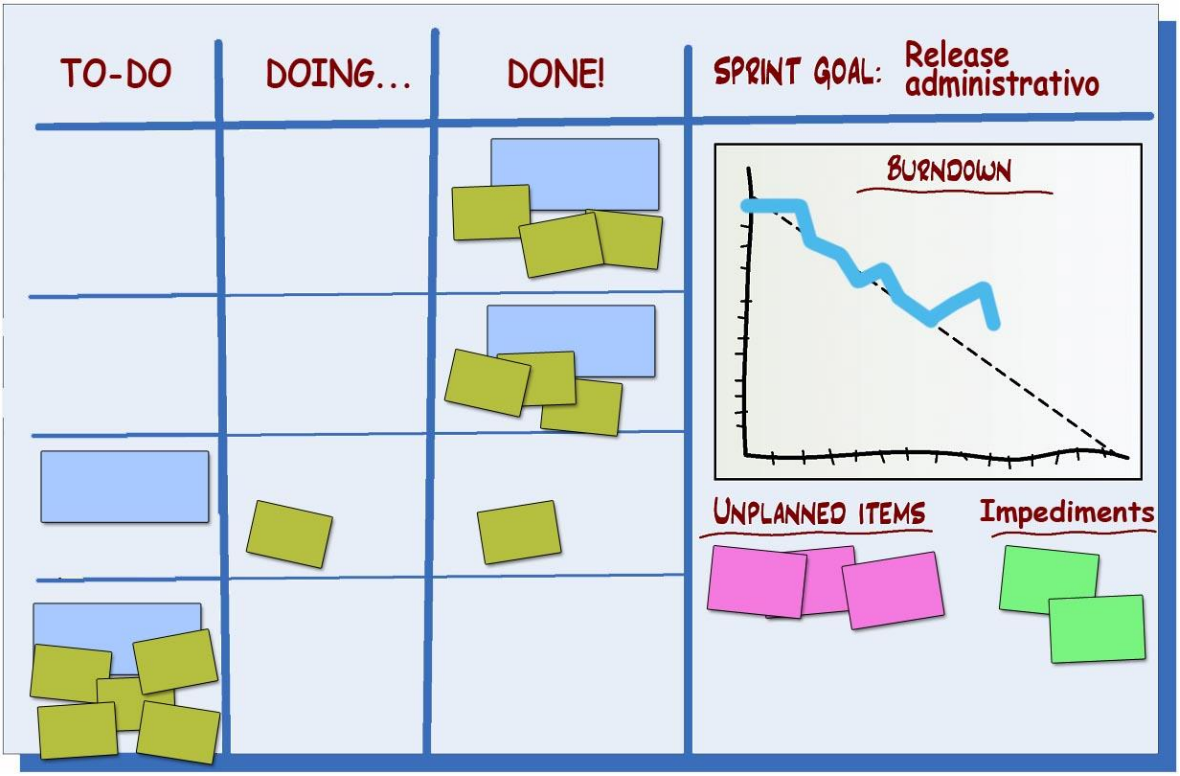
### 3.7.6 Quadro de Tarefas ou *Agile Radiator*

O quadro é utilizado para acompanhamento, comunicação, controle das tarefas diárias entre membros da equipe; também conhecido como *kanban,* seu significado sugere *Kan*(Visual) e *Ban*(Cartão ou Quadro). Aqui é possível observar e acompanhar a evolução do *Sprint* por todos da equipe *scrum* e até mesmo *stakeholders*; também utilizado para gerenciar as atividades de maneira visual. Um método prático e rápido, o *Kanban* auxilia as pessoas a desenhar,

desenvolver e a evoluir seus sistemas de trabalho. Seu propósito é provocar discussões a respeito do sistema de trabalho.

O quadro é separado em três ou mais colunas: a primeira é destinada ao planejamento, a central ao andamento e a última aos itens finalizados. Conforme exemplo e explicação dos pontos abaixo:

Figura 3. Ilustração de um quadro *Kanban*.




Fonte: http://webinove.wordpress.com/2011/07/07/kanban-o-inicio/

- *To-Do* - Atividades que ainda serão realizadas no *Backlog* do *Sprint*.
- *Gráfico Burndown* - Mostra as informações de como está indo o desenvolvimento através de um gráfico debatido no capítulo abaixo.
- *Doing* - Atividades que estão sendo desenvolvidas no momento.
- *Done* - Atividades que já foram finalizadas pela equipe de desenvolvimento.

O quadro de tarefas é fixado na parede da sala onde a equipe trabalha, com tamanho suficiente para que todos acompanhem, de maneira visual, o andamento das atividades. Cada equipe monta seu quadro conforme suas necessidades.

### 3.7.7 Gráfico *Burndown*

O gráfico de *Burndown* tem como finalidade mostrar o quanto de trabalho restante ainda tem-se que finalizar, e conseguimos também verificar através do tempo planejado fazer um comparativo com o que está sendo desenvolvido no momento e se a equipe está tendo um bom rendimento ou mau rendimento ao decorrer do *sprint*.

Para realizar seu levantamento é necessário multiplicar a quantidade de pessoas no time de desenvolvimento pela quantidade de horas que o *sprint* tem em sua totalidade; após isso, cada dia trabalho deve ser descontado das horas planejadas e deve ser subtraída também a quantidade de tempo que a equipe conseguiu desenvolver em um mesmo dia conforme tabela abaixo, considerando um *sprin*t de trinta dias e três desenvolvedores trabalhando oito horas por dia cada:

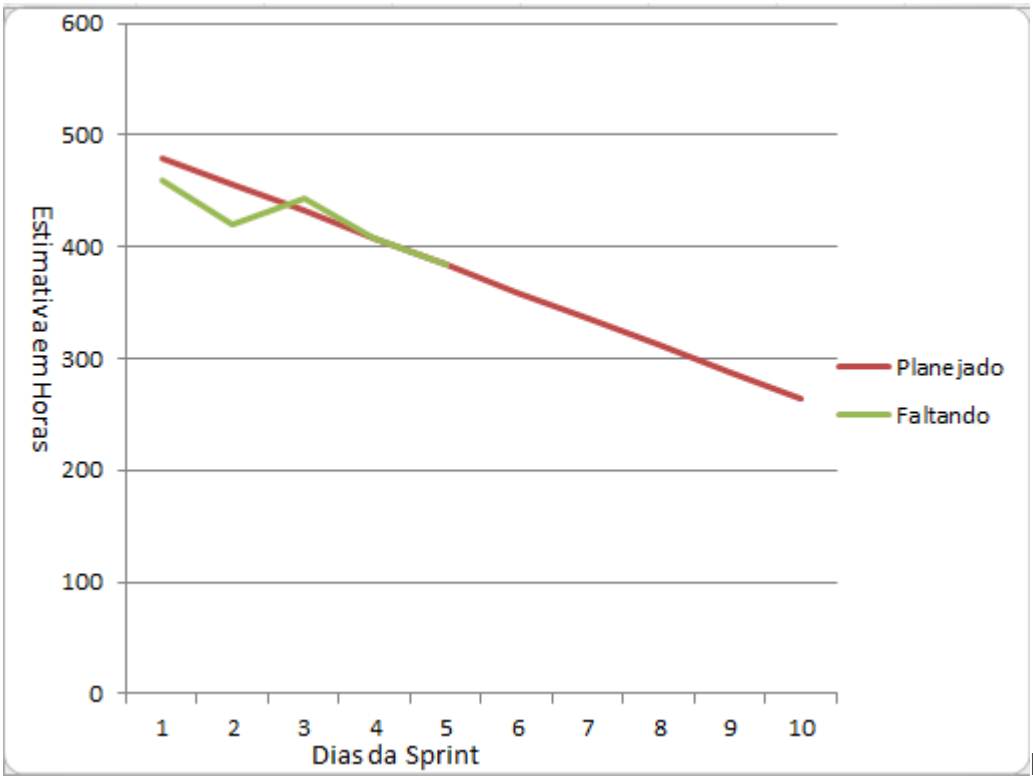
QUADRO 3: LEVANTAMENTO PARA O GRÁFICO BURNDOWN

| **Dia** | **Planejado** | **Faltando** |
|---|---|---|
| 0 | 480 | 460 |
| 1 | 456 | 420 |
| 2 | 432 | 443 |
| 3 | 408 | 408 |
| 4 | 384 | 384 |
| 5 | 360 | |
| 6 | 336 | |

| | | |
|---|---|---|
| 7 | 312 | |
| 8 | 288 | |

Com as informações levantadas podemos realizar a configuração do gráfico conforme a imagem abaixo, que pode ser feito utilizando inclusive ferramentas simples como o Microsoft Excel.

FIGURA 4: GRÁFICO BURNDOWN




Mudar escrita do eixo para “Dias do Sprint”

Com o gráfico em mãos fica fácil de analisar como o trabalho da equipe está sendo desenvolvido ao longo da *sprint*. Caso a linha verde fique muito acima da vermelha provavelmente o *sprint* vai ser atrasado, então é importante neste momento avisar o dono do produto sobre o que está acontecendo, com este podendo retirar ou não itens do *sprint*.

Caso no gráfico a linha verde esteja muito abaixo da vermelha, então quer dizer que a equipe está trabalhando mais rápido do que o estimado, podendo

então ser incluídos novos itens para o desenvolvimento. Em um último caso, em que as linhas estejam sempre próximas uma da outra, nos mostra que o *sprint* está sendo desenvolvido conforme programado.

Para ajudar a montar o gráfico é essencial que, no final do dia, cada programador ou par de programadores que estiver trabalhando via programação em pares faça uma estimativa de quanto trabalho já desenvolveu ao longo do dia e, caso verifique que uma atividade esteja em atraso, todos possam conversar abertamente sobre o que aconteceu na reunião diária, assim um podendo ajudar o outro, de forma transparente e integrada.

### 3.7.8 Revisão do *Sprint*

É uma reunião *time-boxed* em quatro horas no final do *sprint*, aonde seus objetivos principais são mostrar ao dono do produto, *stakeholders* ou quem mais for convidado para verificar o *software* executando de maneira funcional; aqui é essencial que seja apresentado apenas o material que foi realmente finalizado, evitando assim que partes inacabadas apareçam.

Neste ponto é essencial que o dono do produto forneça um *feedback* do que está sendo apresentado e mostrar aceitação do que foi desenvolvido. Também é importante que a meta do *sprint* seja alcançada em sua totalidade; caso não tenha sido, os itens que não foram finalizados devem voltar ao *backlog* do produto e repriorizados pelo seu dono.

O *software* funcional pode ser demonstrado em um servidor interno da empresa, podendo ser criado pelo próprio time de desenvolvimento. A maior parte dessa reunião é mostrar sua execução e responder perguntas que podem vir a aparecer em seu caminho, também podendo aqui aparecer novas funcionalidades a serem desenvolvidas.

### 3.7.9 Retrospectiva do *Sprint*

É uma reunião *time-boxed* em três horas e acontece logo após a revisão do *sprint.* Nela vão ser tratadas as lições aprendidas ao longo da duração do *sprint* e tem como objetivo responder a três perguntas:

- O que foi feito de bom durante o *sprint*?
- O que gostaríamos de mudar?
- Como podemos implementar estas mudanças?

Nesta parte são apenas solicitadas as presenças do time de desenvolvimento e do *scrum master,* este último tendo um papel essencial para encorajar o restante do time a falar tudo que sentiu no desenrolar do processo.

Sua meta final é tentar melhorar tudo aquilo que venha trazendo barreira para a equipe de desenvolvimento, facilitando assim seu trabalho no próximo *sprint* a ser trabalhado.

## 3.8 QUANDO USAR O *SCRUM*

É de conhecimento que, para se trabalhar com o *scrum,* o ideal é que a equipe tenha de três até nove pessoas. Assim como o *scrum,* o XP também trabalha com equipes de pequeno a médio porte.

Em se tratando de projeto de grande porte, o *scrum* pode ser utilizado através de escalonamento, aonde uma equipe é subdividida em várias e todas trabalham em um mesmo *backlog* do produto, tendo aqui que existir algumas interações ao longo do *sprint* através da *scrum of scrums,* uma reunião que traz atualização dos acontecimentos de todas as equipes, aonde um representante de cada time de desenvolvimento participa, e este, quando volta, avisa e atualiza os demais colegas.

### 3.8.1 Como *Scrum* e XP se encaixam

Mesmo tendo um *framework* mais focado na gerência e o outro mais no planejamento, existem diversas práticas que combinam entre os dois: assim como temos o *sprint* tratado no *scrum, o* denominamos de desenvolvimento interativo no XP, o que é reunião diária no *scrum* é reunião em pé no XP, e o planejamento do *sprint* equivale ao jogo do planejamento no XP, e assim encontramos diversos caminhos que se encontram.

Como encontramos partes que são praticamente idênticas citadas anteriormente, por se tratarem de metodologias ágeis, também existem outras que se encaixam bem entre as duas. Quando por exemplo trabalhamos com certas práticas do desenvolvimento que são essenciais para o desenrolar do projeto de *software,* verificamos que isso já não faz parte do *scrum* como programação em par, refatoração.

Também encontramos partes que pode ser tratadas de forma diferente, sendo optado pela empresa, como o desenvolvimento do *sprint* no *scrum:* uma vez começado, a equipe não permite mais alteração em seu desenvolvimento, já no XP é um pouco mais flexível, desde que a equipe já não o tenha começado.

Então concluímos que só não é importante como é uma combinação perfeita que vai trazer benefícios à equipe ao utilizar as duas metodologias simultaneamente.

### 3.8.2 O que as equipes acham do XP e do *Scrum*

As equipes, quanto mais entendem dos processos das metodologias ágeis, mais descobrem que precisam aprender muito mais ainda. Também concordam que é necessário estarem em um ambiente agradável e que tenham um reconhecimento do seu trabalho, pois pessoas infelizes e insatisfeitas na equipe

podem trazer um prejuízo para os demais colegas. Segundo Deborah Silveira, “O *scrum* e o XP utilizados de forma conjunta ajudaram a derrubar barreiras entre a TI e o cliente final, trazendo sempre informações atualizadas, pois o usuário já sabia que o sistema estava sendo desenvolvido e que iria ser entregue no prazo estabelecido e o *feedback* é sempre constante.”

## 4.CONCLUSÃO

Aqui verificamos que as duas metodologias trabalhando em conjunto podem trazer diversos benefícios para empresas de pequeno porte, com poucos profissionais, pois o *scrum,* com a parte gerencial, nos mostra como proceder nos pontos citados neste trabalho e com o XP, com suas técnicas avançadas de desenvolvimento, pregando sempre que a informação deve ser expandida para os profissionais, principalmente em empresas com alta rotatividade de seus colaboradores.

O estudo realizado poderá ser utilizado como fonte de referência para os profissionais das áreas tecnológicas, administrativas, entre outras, como também no meio acadêmico, e pode ser o ponto de partida para novos estudos e pesquisas para formulação de projetos e propostas de melhorias e inovações.

## REFERÊNCIAS


Agilcoop. ***Scrum***: Uma breve apresentação. http://www.slideshare.net/moacybarros/3-scrum-presentation. 2008. (Acessado em novembro de 2013).

CASTRO, V A. **Desenvolvimento ágil com Programação Extrema eficácia e disciplina extrema no desenvolvimento orientado a objetos de *software*.** http://devagil.files.wordpress.com/2008/01/desenvolvimentoagil_com_xp_beta10_vinicius_ac0.pdf. 2006. (Acessado em setembro de 2013).

BECK, K. **Programação Extrema Explicada.** Bookman Companhia Ed, 2004.

BECK, K.; BEEDLE, M.; VAN, BENNEKUM,.; COCKBURN, A.; CUNNINGHAM, W.; FOWLER, M.; GRENNING, J.; HIGHSMITH, J.; HUNT, A.; JEFFRIES, R.; KERN, J.; MARICK, B.; MARTIN, R. C.; MELLOR, S.; SCHWABER, K.; SUTHERLAND, J.; E THOMAS, D. **Manifesto for Agile Software Development**. http://manifestoagil.com.br. 2001.(Acessado em outubro de 2013).

CASTRO, F. S. **Gráfico *Burndown***: Sugestão de Uso. http://www.agileway.com.br/2009/08/18/grafico-burndown-sugestao-de-uso. 2009. (Acessado em novembro de 2013).

COHN, M. ***Agile Estimating and Planning***. New Jersey: Prentice Hall, 2005.

COHN, M. ***Succeeding with Agile****: Software Development Using Scrum*. Massachusetts: Addison-Wesley Professional, 2009.

COHN, M. ***User Stories Applied For Agile Software Development***. Massachusetts: Addison-Wesley Professional, 2004.

FARIAS, L. J. **Regras da Reunião Diária de *Scrum***. http://www.bluesoft.com.br/regras-da-reuniao-diaria-de-scrum/. 2007. (Acessado em novembro de 2013).

FREIRE, F. **Por que utilizar *User Stories* (Ou histórias de usuário) ao invés de Casos de Uso?.** https://www.ibm.com/developerworks/community/blogs/rationalbrasil/entry/user-stories. 2010. (Acessado em outubro de 2013).

LAYTON, M.C. ***The Agile Management Sprint Review***. http://www.dummies.com/how-to/content/the-agile-management-sprint-review.html. (Acessado em novembro de 2013).

LOPES, C. **Refatoração**. http://www.slideshare.net/camilolopes/seminario-refatoracao. 2009. (Acessado em novembro de 2013).

KNIBERG, H. ***Scrum* e XP direto das Trincheiras**. C4Media, 2007.

MEDEIROS, M.P. **Planejando seu projeto com *Extreme Programming***: Parte I. http:/www.devmedia.com.br/planejando-seu-projeto-com-extreme-programming-parte-i/4273. 2007. (Acessado em novembro de 2013).

MEDEIROS, H. **Introdução ao *Extreme Programming* (XP)**. http://www.devmedia.com.br/introducao-ao-extreme-programming-xp/29249. 2013. (Acessado em outubro de 2013).

PINTO, R; RIBEIRO, J; CASASANTA, L. **Um comparativo entre XP e *Scrum***: Entenda o funcionamento e as principais diferenças entre XP e *Scrum*. http://www.devmedia.com.br/um-comparativo-entre-xp-e-scrum-revista-engenharia-de-software-magazine-51/25752. 2012. (Acessado em Novembro de 2013).

REBELO, P. ***Extreme Programming (XP)***: Desenvolvendo *Software* com Qualidade e Produtividade. http://www.slideshare.net/phrebelo/introduo-ao-xp. 2010. (Acessado em novembro de 2013).

ROCHA, F. G. **Introdução ao desenvolvimento guiado por teste (TDD) com *JUnit***. http://www.devmedia.com.br/introducao-ao-desenvolvimento-guiado-por-teste-tdd-com-junit/26559. 2012. (Acessado em novembro de 2013).

TELES, V. M. ***Extreme Programming***: Aprenda como encantar seus usuários desenvolvendo *software* com agilidade e alta qualidade. Novatec Editora, 2004.

WEBI9. ***Kanban, o início da revolução.*** http://webinove.wordpress.com/2011/07/07/kanban-o-inicio/. 2011. (Acessado em outubro de 2014).

BISSI, W. ***SCRUM – Metodologia de Desenvolvimento Ágil.*** http://www.proa.pa.gov.br/bitstream/prodepa/34/1/scrum.pdf. 2007. (Acessado em outubro de 2014).