TÉCNICAS ÁGEIS:
CONSTRUINDO UMA MENTALIDADE ÁGIL

Gianncarlo dos Santos Soares[1]

RESUMO

O tempo é algo valiosíssimo e a cada dia está se tornando ainda mais importante para as pessoas e organizações, com esse intuito surgiram as metodologias ágeis para garantir a entrega do *software* de uma maneira rápida e eficiente, para isto não adianta apenas se auto intitular uma empresa ágil e sim, demonstrar isto aos seus clientes, contudo ainda é algo que pouco acontece nas empresas de *software*. Com base nisso que foi feito este artigo, ele tem como intuito exemplificar algumas técnicas para um melhor gerenciamento de *software*, ou seja, este artigo não tem como objetivo fazer revisão bibliográfica nos assuntos abordados, para aprofundar-se mais nos assuntos aqui aplicados existem *links* no final do arquivo.

Palavras-chave: ágil; desenvolvimento de *software;* gerenciamento; projetos; *tempo*

2. FUNDAMENTAÇÂO TEÓRICA

Desde o início da criação do *software* os projetos de TI têm sido sinônimo de falhas, o que acaba denegrindo sua reputação entre a maior parte de seus usuários (Al-Ahmad et al., 2009; Sauer & Cuthbertson, 2003) onde os seus maiores problemas são:

2.1 Escopo:

> “ Para Karlsson e Ryan (1996), um dos maiores riscos enfrentados por organizações que desenvolvem *software* comercial está associado ao não atendimento das necessidades e expectativas dos usuários. Para esses autores, esse risco pode ocasionar danos na reputação, perda de pedidos e redução dos lucros da empresa. Pesquisas que visam à identificação das causas para o problema citado apontam a fase de elicitação de requisitos, também conhecida como levantamento de requisitos, como básica para a melhoria do processo. “ (Universidade Estadual do Norte Fluminense Darcy Ribeiro, 2011)

1 Gianncarlo Soares, Graduado do curso de Sistemas de informação - UNIT

- Falha no levantamento de requisitos ocasionando retrabalho, onde muitas vezes analistas e desenvolvedores culpam os clientes com a famosa frase “cliente não sabe o que quer”, porém, o cliente sabe sim o que quer, ele pode não saber como passar para o analista tudo que ele considera necessário ou até mesmo por não saber o quanto este *software* pode facilitar o seu trabalho, fazendo com que ele sempre acabe inserindo alguma coisa no projeto e assim ocasionando retrabalho, mas tudo isso é trabalho do analista, saber interagir, realizar várias perguntas, manter-se sempre interligado com o cliente e por fim ter em mente que sempre haverá modificações no *software* ao longo do projeto, pois uma constante interação com cliente fará com que o analista enxergue pontos no qual no início do projeto não foram observados.

2.2 Artefatos:

- Artefatos são documentos ou idéias, gerados ou relacionados a um projeto, onde a falta deles ao decorrer do projeto pode ocasionar atrasos pelo fato que artefatos estão diretamente interligados a todas as etapas do projeto, principalmente ao escopo e o desenvolvimento, sem eles ou inconsistentes, pode acabar ocasionando retrabalho e o mais comum a falta de entendimento sobre o projeto, o seu andamento e sobre a própria equipe.

2.3 Código:

- O *software* é considerado como um produto criativo, pois vem da mente dos seus desenvolvedores em como fazer o mesmo, por exemplo, para se fazer um determinado *software* vários desenvolvedores podem ter idéias diferentes onde todas tem a mesma finalidade, contudo, considerar o *software* apenas como a “alma” do seu desenvolvedor pode acabar ocasionando vários problemas a curto e até longo prazo, pois apesar de cada desenvolvedor ter seu modo de pensar, em um determinado momento o mesmo sairá daquela empresa e um outro será obrigado a pegar os projetos feitos por

aquele que saiu e assim em quase 100% dos casos o novo responsável não entenderá o código do seu antigo companheiro.

2.4 Prazo:

> “ O planejamento de tempo é uma parte do planejamento do projeto que envolve entre outras atividades, as atividades a estimativa de esforço, a definição de atividades, sequenciamento de atividades, estimativa de recursos, estimativa de duração, e desenvolvimento do cronograma. “ (PMI 2008) (SEI 2010)

- Em muitos casos a data de entrega do *software* é feita ao acaso, ou melhor, no “eu acho”, ou pior ainda, o cliente dita quando deseja receber o produto e obviamente algo deste tipo irá terminar em atraso na entrega ou quando não ocorre atraso entrega-se um *software* totalmente diferente do pedido, cheio de erros e com um layout nenhum pouco legível.

2.5 Equipe:

- Muitas vezes o trabalho em equipe está só no nome, onde um faz sua parte da maneira que acha melhor e apenas repassa para o próximo e assim ocorrendo o famoso *crash* no projeto, ou ainda o integrante desconhece totalmente em que parte do projeto estão os outros integrantes e por fim o líder que passa o que é para ser feito e esquece totalmente de sua equipe ao decorrer do projeto.

2.6 Projeto Ágil só de nome:

- Por fim as famosas metodologias ágeis, que realmente tem como o intuito de dar uma alavancada no tempo de entrega de um projeto, contudo em muitas empresas existem o ágil apenas no nome, isto por diversos fatores como ler um livro por exemplo de *SCRUM* tentar implementar de qualquer jeito e deixar por isso mesmo ou até mesmo colocar um prazo de entrega curto e por algum milagre conseguir entregá-lo uma vez ou outra.

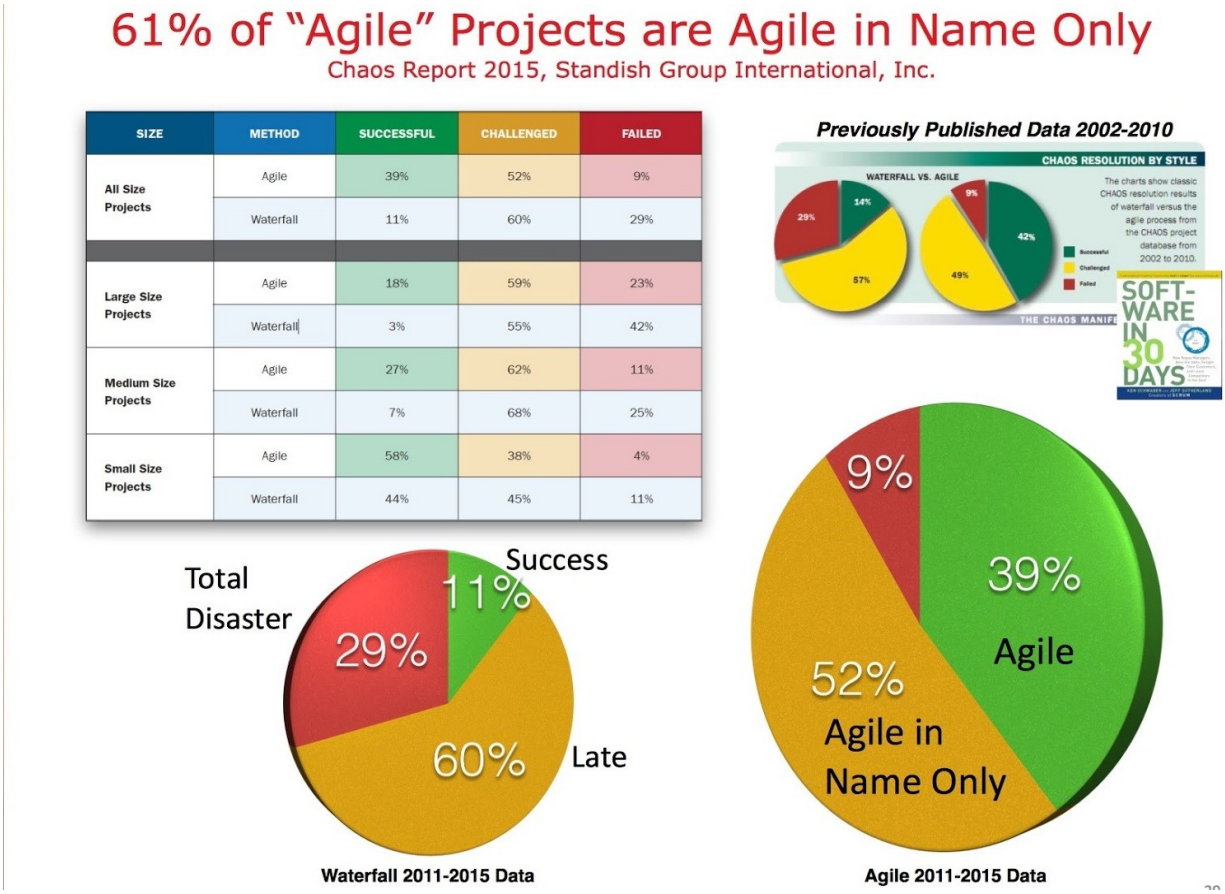

Figura 1. 61% dos projetos ágeis são ágeis apenas no nome. (Fonte*: Scrum Inc*.)

As técnicas ágeis não são um processo padronizado onde metodicamente você segue uma série de etapas sequenciais e consegue o que desejava e isto é o que muitas empresas que se auto denominam "ágeis" não entendem como por exemplo o *SCRUM,* ele é um conjunto de princípios e práticas que fornecem a base para que a sua organização adicione suas práticas particulares de engenharia e gestão e que sejam relevantes para a realidade da sua empresa e o resultado disto será uma versão do *SCRUM* exclusivamente sua.

Neste artigo não será discutido sobre qual é a melhor técnica ágil e sim como as várias técnicas se complementam como, *SCRUM*, *Planning Poker*, *Kanban* e *XP*, além de falar um pouco sobre *Gemba*, lembrando que neste artigo não tem como objetivo uma revisão bibliográfica e sim técnicas que podem auxiliar a gestão de *software*, para aprofundar-se mais no assunto ao final do artigo serão colocados *links* sobre estas técnicas.

3. DESENVOLVIMENTO

3.1 Gemba – Método para gestão de comunicação

O *Gemba* significa literalmente “local real” ou, como pode ser encontrado em algumas literaturas, “lugar verdadeiro”. Para resolvermos um problema é necessário entendê-lo totalmente e ir até ao local fará com que o funcionário tenha sua própria visão dos fatos que compõem o problema.

3.1.1 Mas o que isto tem a ver com a TI?

Quando recebemos um e-mail ou um relatório ou uma ligação sobre determinado problema, essa é a visão de quem o escreveu. Nesse processo, fatos poderão ser ocultos ou perdidos e diversas vezes podem ser cruciais para o melhor entendimento sobre o que está acontecendo e ir ao *Gemba* não significa uma questão desconfiança e sim uma questão de objetividade, aplicando isso na TI não só demonstra que a organização está disposta a resolver o problema como isto também faz com que obtenha se um conhecimento maior sobre isto e que não ocorra em projetos futuros.

3.1.2 Porque ir ao *Gemba* é tão importante?

Pois, se um problema precisa ser resolvido e é de sua responsabilidade, então nada melhor do que você levantar os fatos e dados e ter sua própria visão do problema e essa atitude faz toda diferença nas empresas, pois com ela os funcionários, gerentes e diretores resolverão os problemas com rapidez, poupando esforço, tempo, etc.

3.2 Kanban – Método para gestão de mudanças

O *Kanban* é um método para gestão de mudanças onde tem como o objetivo visualizar o andamento do projeto, medir cada passo da sua cadeia de valor do conceito geral até o *software* e restringir o total de trabalho permitido para cada estágio. Ele é um dos métodos de desenvolvimento de *software* menos prescritivos, se tornando assim adaptável a quase qualquer tipo de cultura.

- Visualizar o fluxo de trabalho (*workflow*) - Quando criamos um modelo visual do fluxo de trabalho da equipe, fica possível identificar o que realmente está sendo feito. Todos podem enxergar o contexto do outro, levando instantaneamente o aumento da comunicação e colaboração. A visibilidade vai permitir que você perceba o impacto das mudanças. Desta forma o *Kanban* proporciona uma visão ampla do que está sendo feito, em qual etapa, o que está pronto, quanto está pronto e o quanto a equipe consegue entregar, lhe concedendo previsibilidade.

- Limitar a quantidade de trabalho em andamento (WIP) – WIP significa *Work in process,* ao limitar o WIP o ritmo da equipe se torna equilibrado, assim ela não se compromete com muito trabalho de uma só vez e reduz o tempo gasto em um item.

3.2.1 – Quando se deve considerar utilizar o *Kanban*?

> “ Você tem se esforçado para implementar o Agile em sua organização, mas sem muito sucesso?
>
> Você tem usado o Agile por algum tempo, mas as melhorias de desempenho começaram a estagnar?
>
> Você está utilizando tempo precioso em práticas ágeis que já não parecem mais se encaixar no contexto em que está trabalhando?
>
> Tem usado Agile como um checklist, mas sem compreender completamente os princípios básicos?

Sente a necessidade de flexibilidade que vá além da oferecida por iterações fixas e planejadas?

Suas prioridades mudam diariamente?

Você está utilizando processos criados para o desenvolvimento ágil num contexto em que não se adaptam facilmente, como por exemplo em manutenção e operações?

Precisa de uma transição gradual da execução em cascata para o Agile, para evitar altos níveis de resistência organizacional? “ (*Kanban* em 10 passos pág 4 e 5, 2013)

Caso uma das respostas seja sim, utilizar o *Kanban* pode ser uma boa alternativa.

3.2.2 Como é o *Kanban*

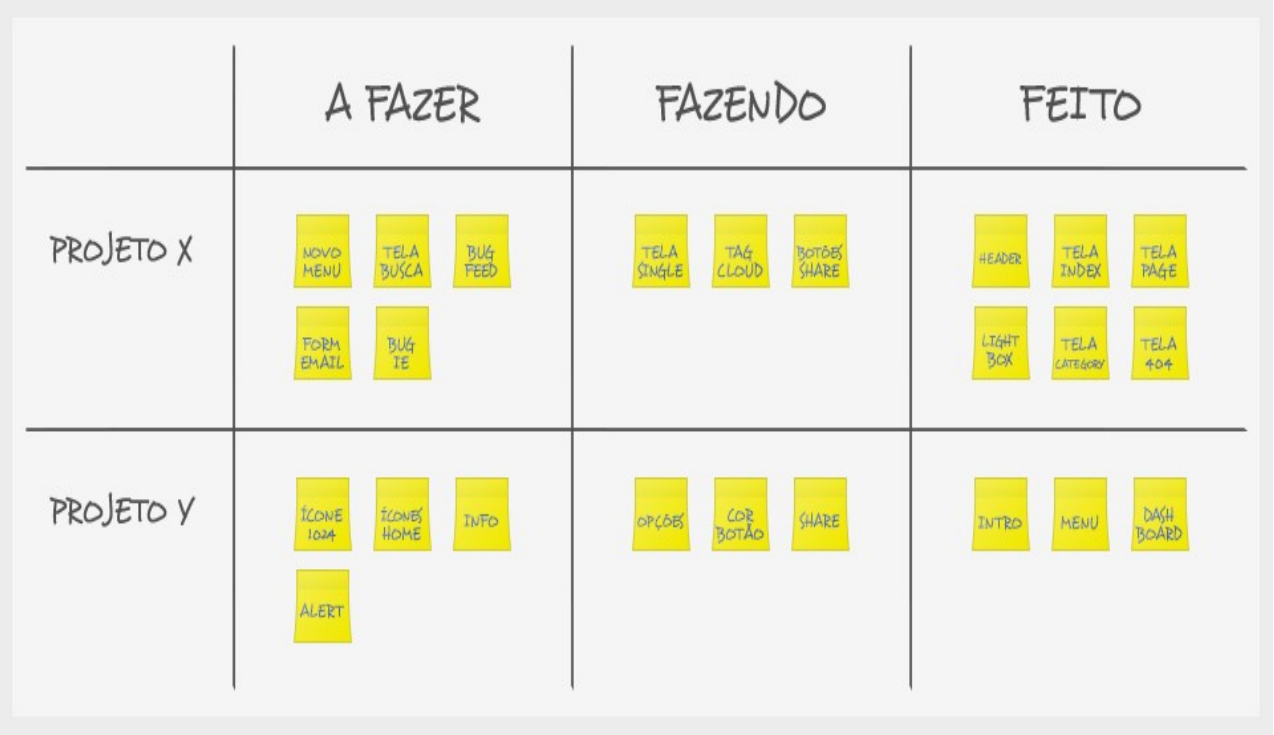

Figura 2. Cultura ágil *Kanban* Simples (Fonte: http://larissaherbst.com)

Esta é uma imagem de um *Kanban* simples, a partir do que é necessário para a organização existem técnicas para deixa-lo ainda mais completo e complexo.

3.3– *Planning Poker* – Método para estimativa de esforço e tempo

Planning Poker é um método de estimativa baseado em julgamento do especialista amplamente utilizado em métodos ágeis, especialmente *SCRUM* e *Extreme Programming* (Mahnič e Hovelja 2011).

Segundo [Raith et al. 2013], o fluxo de execução do *Planning Poker*, envolve todos os membros do time de desenvolvimento, como: estimadores, um moderador (um *Scrum Master*, por exemplo) e, opcionalmente, um representante do cliente.

Comparado também com a média das estimativas, o processo necessário para realização do *Planning Poker* oferece estimativas mais acuradas e menos otimistas. Além do fato de estimativas realizadas em grupo resultarem em estimativas mais acuradas que estimativas individuais [Raith et al. 2013].

3.3.1 Como utilizar o *Planning Poker*?

O uso do *Planning Poker* reforça fortemente o conceito de colaboração e comprometimento. Em um projeto tradicional alguém estima para você usando os entendimentos dele e passa a atividade para você e um prazo. Você mesmo sabendo que não vai conseguir fazer pega a atividade e inicia o ciclo sem saber se vai ou não entregar naquele prazo. Já o *Planning Poker* faz com que todos votem na complexidade e não somente uma pessoa. Com esse espírito de união e colaboração chega-se a uma complexidade que seja o consenso do grupo.

A escala de complexidade é baseada na sequência de Fibonacci (0, 1, 2, 3, 5, 8, 13) e cada participante do time que estiver comprometido recebe um conjunto de cartas sendo cada uma com o número de complexidade. O grupo se reúne e geralmente na reunião, *Sprint Planning,* esclarece as estórias com o PO (*Product Owner*) para depois estimar uma a uma. Após estimar confere-se as cartas de cada um e se houver divergências cada membro pode argumentar explicitando o que o levou a dar aquele valor, se considerou algo complexo ou simples, com o tempo ouvirá opiniões como “eu já implementei algo parecido” e essa troca de informações fará com que os membros acabem aprendendo e descobrindo que aquela rotina complexa não é tão complexa, assim chegando a um consenso sobre o quanto deverá levar para se fazer cada estória.

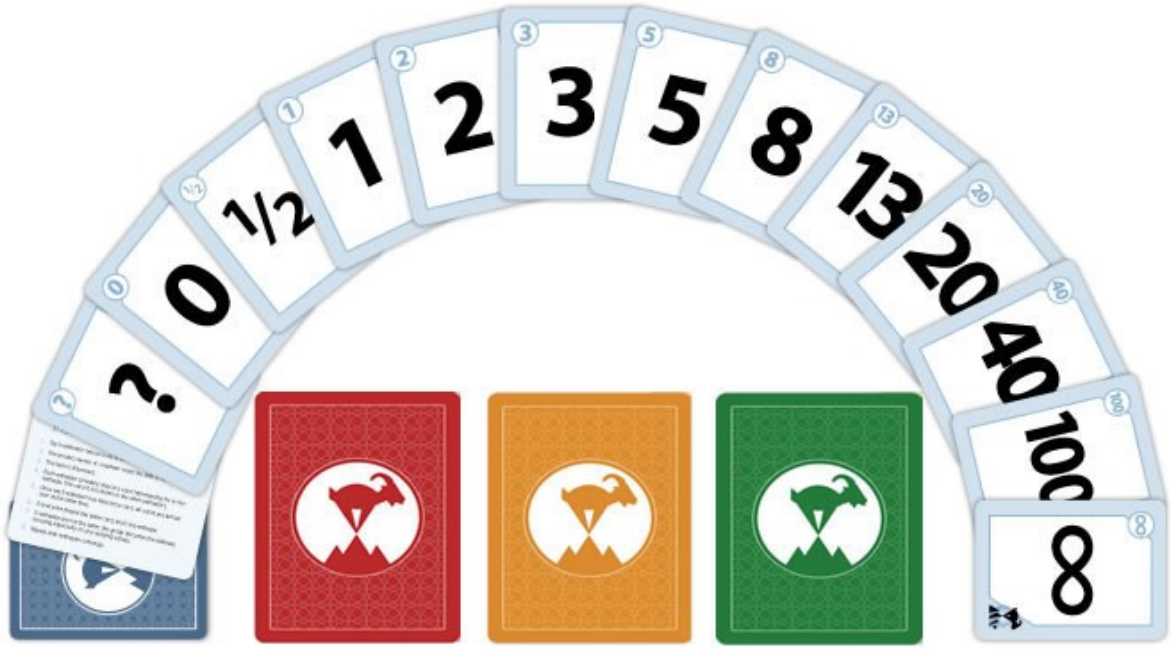

Figura 3. Cartas do *planning poker (*Fonte*: Mountain Goat.*)

3.4 SCRUM

O *SCRUM* apresenta recursos que visam manter estável o controle do desenvolvimento mesmo em ambientes instáveis, onde predominam constantes mudanças no escopo de requisitos, ele é fundamentado na teoria de controle de processo onde tem por objetivo aperfeiçoar a previsibilidade e controlar os riscos de um projeto a transparência, *inspeção* e adaptação são os três pilares que sustentam a metodologia *SCRUM* (Revista computação aplicada, 2013). De acordo com a revista computação aplicada (2013, pág. 43):

> A transparência é a garantia que os processos que envolvam o resultado sejam claros para ambas as partes envolvidas no projeto.
>
> A inspeção é realizada durante todo o projeto e tem por objetivo detectar qualquer variação e ajustar o processo evitando assim problemas futuros.
>
> A necessidade de adaptação surge da inspeção e tem por finalidade adaptar o processo para qualquer variação detectada na inspeção.

A partir disso o *SCRUM* é formado por três partes:

Figura 4. Equipe *Scrum* (Fonte: http://blog.myscrumhalf.com/)

- *Product Owner* é o proprietário do produto que representa a empresa que será aplicada o projeto.

- *Scrum Master* é o papel assumido pelo gerente do projeto.

- A equipe de desenvolvimento que é composto por um grupo de até 7 pessoas e que são responsáveis pela análise, programação e testes do projeto.

3.4.1 Sprint Planning Meeting

Sprint Planning Meeting é uma reunião onde estão as três partes do *SCRUM*, durante esta reunião o *Product Owner* descreve as funcionalidades de maior prioridade para a equipe (note que não são todas as funcionalidades e sim as de maior prioridade), a equipe deve fazer perguntas para poder quebrar as funcionalidades em várias técnicas após a reunião. As funcionalidades descritas pelo *Product Owner* irão dar origem ao *Product Backlog* e ao pegar uma funcionalidade e a quebrar em várias técnicas se dá a origem do Sprint Backlog.

3.4.2 Product Backlog

É uma lista de tarefas onde é organizada de acordo com a prioridade de cada item, sendo que os itens que tem maior importância devem estar no topo da lista, esta lista deve ser constantemente atualizada sempre priorizando os itens com maior importância.

3.4.3 Sprint Backlog

O *Sprint Backlog* é uma lista de tarefas que o *Scrum Team* se compromete a fazer em um *Sprint*. Os itens do *Sprint Backlog* são extraídos de cada item do *Product Backlog*, pela equipe, com base nas prioridades definidas pelo *Product Owner* e a percepção da equipe sobre o tempo que será necessário para completar as várias funcionalidades. Cabe a equipe determinar a quantidade de itens do *Product Backlog* que serão trazidos para o *Sprint Backlog*, já que é ela quem irá se comprometer a implementá-los.

3.4.4 Sprint

Um *Sprint* nada mais é que um Ciclo que pode durar 2 ou 4 semanas no qual o *Sprint Backlog* deverá ser executado neste tempo.

3.4.5 Daily SCRUM

Daily SCRUM é uma reunião diária durante o *Sprint* deve-se durar no máximo 15 minutos. Ela tem como objetivo disseminar conhecimento sobre o que foi feito no dia anterior, identificar impedimentos e priorizar o trabalho a ser realizado no dia que se inicia. Durante o *Daily SCRUM* cada membro da equipe provê respostas para as seguintes perguntas:

- O que você fez ontem?
- O que você fará hoje?
- Há algum impedimento no seu caminho?

Assim a equipe tem compreensão do que foi feito e o que ainda necessita ser feito. Lembrando que esta reunião não deve ser usada para resolução de problemas, questões levantadas devem ser levadas para fora da reunião para serem tratadas apenas com quem está envolvido diretamente com o problema e quem pode ajudar a solucioná-lo

3.4.6 Sprint Review Meeting

É uma reunião que tem como objetivo mostrar o que foi alcançado durante o *Sprint* e avaliar em relação ao que foi planejado para o *Sprint* no *Sprint Planning Meeting* para ver se a equipe alcançou os seus objetivos.

3.4.7 Sprint Retrospective

É uma reunião ao final de um *Sprint* e serve para identificar o que funcionou bem, o que pode ser melhorado e que ações serão tomadas para melhorar. (Note que no *Sprint Review Meeting* a reunião serve para ver se a equipe alcançou os seus objetivos e no *Sprint Retrospective* identificar ações que podem ser melhoradas).

3.4.8 Considerações sobre o *SCRUM*

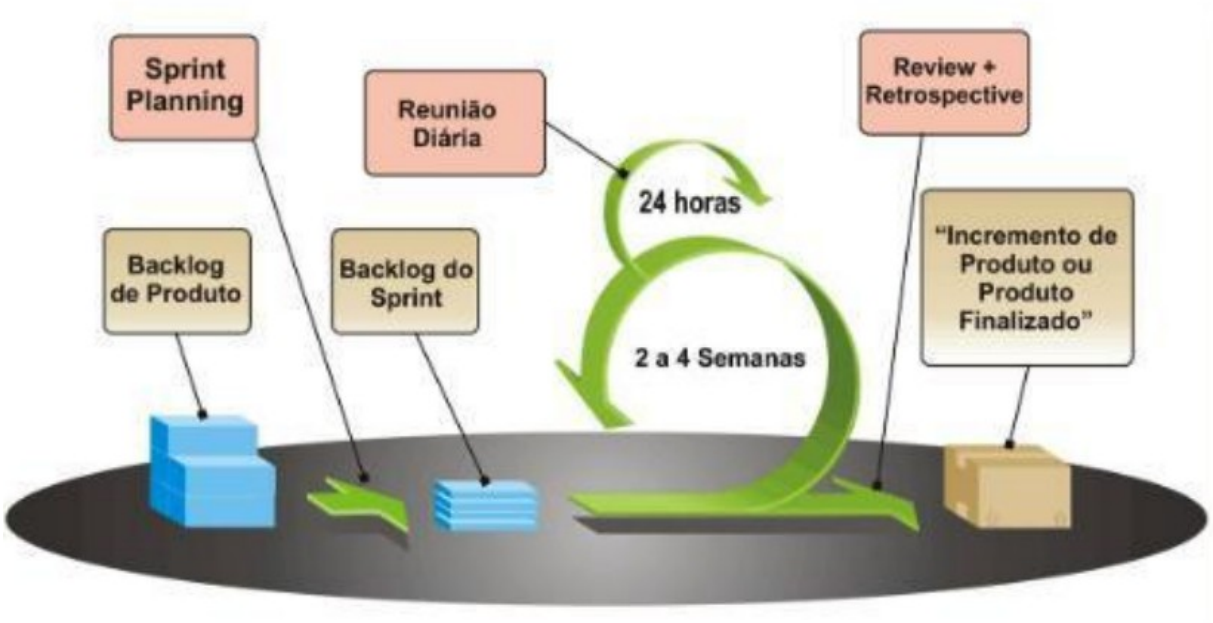

Figura 5. Etapas do *SCRUM (Fonte: Mountain Goat.)*

As 7 etapas descritas anteriormente sobre o *SCRUM* estão em ordenadas para cada *Sprint* do *SCRUM* onde ao final de cada *Sprint* estas etapas devem ser feitas novamente até o final do projeto.

3.5 Extreme Programming (XP)

A *Extreme Programming* é uma metodologia leve, eficiente, flexível, previsível, científica, possui um baixo risco (BECK, 2000) e é direcionada para equipes pequenas e médias. Um dos objetivos da XP é diminuir o custo com as mudanças no *software*, sendo que o código realizado em cada etapa serve como indicador de progresso do projeto. A XP não considera documentação, processos, ferramentas, planejamento como um valor primário, pois o mesmo foca no bom funcionamento do *software* e a iteração com o cliente. A principal preocupação da XP é com a entrega rápida do *software* e a satisfação do cliente, tendo como premissas cumprir seus custos e prazos. Para isso, a XP possui alguns papéis fundamentais (BECK, 2000):

> “ Programador;
>
> Instrutor: responsável pelo processo como um todo, realizando treinamentos e instruindo os membros da equipe. Esse papel é exercido por uma pessoa com ampla experiência e conhecimento na metodologia;
>
> Fiscal: responsável por fornecer o feedback dos processos, realizando estimativas e avaliando as metas para alcançar os objetivos conforme os recursos e o tempo especificado;
>
> Cliente: responsável por descrever as prioridades no projeto e suas *user stories.* O Cliente também faz parte da equipe, com o objetivo de sanar dúvidas em qualquer fase do projeto;
>
> Testador: ajuda o cliente a realizar testes funcionais regularmente;
>
> Consultor: membro externo que auxilia a equipe com assuntos técnicos;
>
> Gerente de projetos: pessoa responsável pelo projeto, estando sempre em comunicação com a equipe. Esse papel tem por função resolver os problemas e tomar as decisões do projeto; ” (BECK, 2000)

O XP se baseia em cinco valores (BECK, 2000) (BECK, 2004) (WELLS, 2009):

- Comunicação - para que um projeto tenha sucesso e qualidade é necessária muita comunicação “cara-a-cara” entre o provedor de serviço e o cliente;
- *Feedback* - as decisões devem ser ágeis e decisivas, sendo que todos os integrantes do projeto devem estar cientes do que está acontecendo;
- Coragem - otimizações de códigos devem ser feitas sem criar novos bugs no sistema;
- Simplicidade - redução do código-fonte com funções e procedimentos desnecessários. Trabalhar com simplicidade acelera o processo de produção dos requisitos do projeto, pois o que os clientes precisam, geralmente, é mais simples do que o analista descreve e;
- Respeito – entregar o sistema para o cliente no prazo determinado e conforme o solicitado.

Assim nessa metodologia as iterações devem ser curtas, para fornecer constantes atualizações sobre o sistema pois, ela tem como objetivo iterações rápidas para acelerar o processo de *feedback* e tomar como referência os

comentários da versão atual para a próxima etapa. Por esse motivo, a XP requer que os profissionais tenham um perfil mais maduro e experiente, além de serem comunicativos e com facilidade de relacionamento, pois os analistas e programadores fazem contatos frequentemente com os clientes. (*Extreme Programming and RUP*, 2013)

3.5.1 Práticas da *Extreme Programming*

A XP possui uma doze de práticas derivadas de seus valores. Essas práticas definem seu uso (CARDOSO, 2006).

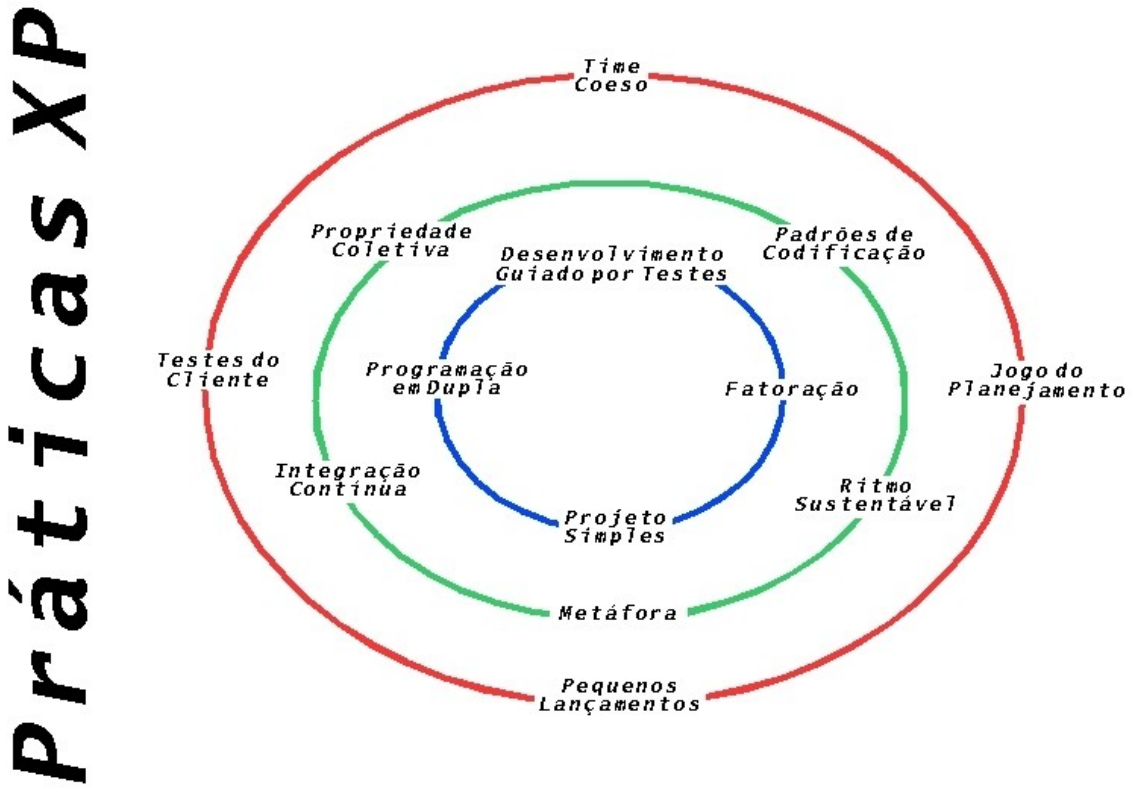

Figura 6. Práticas XP (Fonte: DevMedia.)

3.5.2 Cliente Presente

Um dos paradigmas do desenvolvimento de *software* tradicional é que o cliente não precisa estar presente durante o processo desenvolvimento, já na XP o cliente se torna de vital importância para o sucesso do projeto. O *feedback* que o cliente fornece é parte essencial de uma iteração. (CARDOSO, 2006). Para que a troca de *feedback* possa ocorrer e o cliente possa obter o máximo de valor do projeto, é essencial que ele participe ativamente do processo de desenvolvimento. (TELES, 2004).

3.5.3 Jogo do Planejamento

No início de cada iteração o cliente é convidado a escrever em um cartão de forma rápida e clara sobre as funcionalidades que deseja no sistema, esses cartões são chamados de *user stories.* Ciente do tempo e custo o cliente organiza os cartões de acordo com suas prioridades, esta prática leva a equipe a trabalhar no que é mais importante para o cliente inicialmente.

3.5.4 Programação em par

Ela tem como o objetivo fazer com que dois desenvolvedores peguem um único *user story,* o que tem menor experiência fica responsável de codificar e aquele com maior experiência inspeciona o código e procura por erros e defeitos, além de buscar estratégias mais simples para o código. Um dos benefícios dessa prática é a revisão constante do código, e também a disseminação do conhecimento entre os pares, o que ajuda no nivelamento técnico de toda a equipe (TELES, 2005).

3.5.5 Releases curtos

Segundo Beck (2004) essa prática visa, por meio de releases curtos, entregar versões atualizadas do *software* ao cliente ao longo do processo de desenvolvimento.

3.5.6 Desenvolvimento guiado pelos Testes

Os desenvolvedores escrevem testes para cada funcionalidade antes de codifica-las. Desta forma as interfaces externas dos métodos e classes são planejadas antes de codificação. Esta prática gera uma massa de testes que pode ser usada a qualquer momento para validar todo o sistema (TELES, 2004).

3.5.7 Refatoring

É o processo de reorganizar o código fonte de um *software* para melhorar sua qualidade interna, facilitar a leitura e diminuir o tempo gasto com manutenção, sem, contudo, prejudicar o desempenho e alterar seu comportamento externo. Essa técnica é fundamental para tornar o código mais legível e detectar erros em algoritmos mal escritos (GONÇALVES JR., 2009).

3.5.8 Código coletivo

Os desenvolvedores têm acesso a todas as partes do código e podem alterar aquilo que julgarem importante. Isto cria um mecanismo de revisão e verificação do código assim se um código parecer confuso ele passa pelo processo de *refatoring*.

3.5.9 Código Padronizado

Para poder manipular qualquer parte do sistema de uma forma rápida a equipe estabelece padrões de codificação assim torna o sistema mais homogêneo e facilita manutenção futura.

3.5.10 Integração contínua

Esta prática serve para unir o código de um par de programadores com o código coletivo, assim o par deve testar e juntar seu código à versão mais recente do código coletivo. Isso deve ser feito várias vezes ao dia, para sincronizar as atividades individuais (BECK, 2004).

3.5.11 Ritmo Sustentável

Essa prática consiste em trabalhar respeitando os limites físicos e demonstrando respeito pela individualidade. Para isso, a XP recomenda que a carga horária de trabalho não ultrapasse 8 horas diárias e 40 horas semanais (GONÇALVES JR., 2009).

3.5.12 Metáforas

Esta prática serve para utilizar comparações para que a equipe transmita ideias de modo que todos entendam. O uso de metáforas possibilita transmitir a ideia de modo a esclarecer aos ouvintes. Esta prática facilita a comunicação entre desenvolvedor e o cliente, estabelecendo um vocabulário comum a ambos (BECK, 2004).

3.5.13 *Stand Up Meeting*

Trata-se de uma reunião rápida toda manhã feita para avaliar o que foi executado no dia anterior e priorizar aquilo que será implementado no dia que se inicia, esta reunião recebe o nome de *stand up meeting*, que em inglês significa reunião em pé (TELES, 2004).

3.5.14 Princípios fundamentais para a *Extreme Programming*

Os princípios fundamentais da XP são: *feedback* rápido, simplicidade presumida, mudanças incrementais, aceitação das mudanças e alta qualidade.

- *Feedback* rápido – Obtê-lo, interpretá-lo e aplicar o que é aprendido deve acontecer rápido.

- Simplicidade presumida – É tratar cada problema como se pudesse ser resolvido de maneira simples. Em 98% dos problemas são solucionados de maneira simples, e encará-los desta maneira, a priori, economizará o tempo, necessário para a resolução dos 2% de problemas complexo presentes no projeto.

- Aceitação das mudanças - consiste que: a melhor estratégia é aquela que preserva o maior número de opções, enquanto resolve o problema mais urgente.

- Alta Qualidade – Todo desenvolvedor gosta de entregar trabalhos de alta qualidade do contrário não apreciam seu trabalho.

4. CASO PRÁTICO DE EMPRESAS DE SERGIPE

Ao decorrer da criação deste artigo foram avaliadas o dia a dia da equipe de *software* de duas empresas que utilizam *SCRUM* e *Kanban* com o intuito de utilizá-las como apoio no artigo para dissuadir sobre a junção com algumas das técnicas apresentadas anteriormente além de disseminar sobre os problemas comuns ocorrentes em empresas que utilizam o *SCRUM*.

4.1 Empresa 1

Por uma semana foi avaliada a equipe de software da Empresa 1, onde a mesma possui um software gerenciador de tarefas próprio onde a cada nova tarefa, a mesma é jogada em uma lista e um desenvolvedor a pega ou a depender da complexidade da tarefa, ela é designada a um desenvolvedor mais experiente para desempenhá-la. Após isso a tarefa é colocada na aba de tarefas a se fazer daquele desenvolvedor e o líder passa a acompanhar o progresso dela. Além disso a empresa decidiu implementar o *SCRUM* então todas as sextas-feiras ocorre uma reunião com a equipe e cada um diz sobre a sua semana, problemas enfrentados e o que falta fazer, essas reuniões duram em torno de uma hora e meia.

4.1.1 O Problema

Na duração desta avaliação foram encontrados vários problemas onde poderia agravar a equipe inteira por exemplo: Apesar da equipe possuir um software gerenciador de tarefas, apenas o líder consegue ver o que cada um está fazendo, assim os integrantes da equipe desconhecem o que ocorre com a equipe durante a semana e isto é um ponto falho, pois caso um integrante precise se ausentar por alguns dias a sua tarefa estará completamente estagnada até o seu retorno e a equipe muitas vezes só saberá do problema quando ele se tornar algo mais sério.

Outro ponto a ser avaliado são as reuniões, elas duram aproximadamente 1h30 são muito longas, por causa da quantidade de tarefas desempenhadas pela equipe durante a semana.

4.1.2 Solução

No momento a empresa ainda está engatinhando com o *SCRUM,* contudo uma boa forma de precaver dos problemas descritos anteriormente seria tornar as reuniões diárias e não semanais e manter uma duração de 15 minutos, assim a quantidade de tarefas a serem descritas na reunião iria reduzir drasticamente além disso a equipe estaria mais interligada com os afazeres de seus colegas de trabalho.

4.2 Empresa 2

Por causa da grande quantidade de tarefas esta empresa foi avaliada ao decorrer de três dias, esta empresa tem como objetivo atender chamados dos clientes sobre o seu produto de software e entregar resoluções sobre os problemas dos clientes ou alterações do seu produto, por causa da grande quantidade de clientes, no dia cada integrante da equipe conversa em média com 5 clientes além disso, trabalha como desenvolvedor para solucionar o problema dos seus atendimentos, para isso eles utilizam do *Kanban* para ter uma melhor visão de suas listas de tarefas.

4.2.1 O Problema

Por causa da quantidade de clientes atendidos por dia, cada integrante não consegue solucionar todos os problemas enviados a cada dia, isso acaba sobrecarregando a pilha de tarefas e atrasando todo o projeto de solução, além de definirem como prioridade os clientes melhores financeiramente, ao invés de fazer um levantamento das possíveis causas de cada problema e tentar solucionar os mais simples primeiro para assim poder se focar inteiramente a um problema mais demorada, nota, não está sendo explicitado a gravidade do problema e sim o tempo para a sua resolução.

4.2.2 Solução

A melhor solução inicialmente seria contratar mais profissionais, pois mesmo com uma metodologia para agilizar o processo ainda assim não se pode fazer muito nos dias que ocorre um aumento de ligações de seus clientes. Em relação a um método para diminuir a carga de cada profissional inicialmente o mais viável seria a técnica do *planning poker* junto ao *Kanban,* assim em uma rápida reunião de equipe, pode se observar quais tarefas serão mais simples de resolver qual profissional seria o mais capacitado e também responder a qualquer dificuldade que algum profissional tinha em relação a alguma tarefa aumentando assim o entrosamento da equipe e a habilidade de solução da mesma.

5. CONCLUSÃO

Como pôde ser visto neste artigo, foram abordadas técnicas como o *Kanban*, *Planning Poker*, *SCRUM* e *Extreme Programming* no qual cada uma delas apesar de serem métodos ágeis possuem métodos diferentes voltados a resoluções diferentes, por exemplo:

- Kanban - Tem como seu objetivo visualizar o andamento do projeto, medir cada passo do projeto e restringir o total de trabalho permitido para cada estágio, a partir de um quadro dividido em cada etapa do projeto e cartões com as funcionalidades a serem cumpridas.

- *Planning Poker* – Tem como seu objetivo medir o esforço e o tempo em relação a cada funcionalidade a ser feita no projeto, isso tudo com uma reunião da equipe e cartas com valores baseados na sequência de Fibonacci.

- *SCRUM* - Tem como objetivo o controle de processo para assim aperfeiçoar a previsibilidade e controlar os riscos de um projeto, tudo isso a partir de das funcionalidades descritas no *Product Backlog*, *Sprints* e reuniões entre o *Product Owner*, *Scrum Master* e *Team Scrum*.

- *Extreme Programming* – Tem como o objetivo a entrega rápida e a satisfação do cliente a partir de código simples e de fácil mudança, testes e verificação em tempo de codificação, além de trabalhar em pares e entregas de várias versões para obtenção do *feedback* do cliente.

Como pode observar cada técnica é diferente onde muitas vezes equipes optam pelo SCRUM ou XP, contudo como foi descrito neste artigo ele tem como o objetivo alinhar essas técnicas e partir de tudo que foi redigido levanta-se uma grande questão.

Porque decidir por utilizar o *SCRUM*, XP, *Kanban* ou *Planning Poker*, ao invés de juntar todas as qualidades dessas metodologias em uma só?

Observe:

- O *Kanban* utiliza-se de um quadro dividindo as etapas onde cada cartão com suas funcionalidades se encontram no projeto, o *SCRUM* utilizar uma Lista de funcionalidades e a XP utiliza cartões denominados *user stories* para que o cliente descreva as funcionalidades do projeto e definir suas prioridades, **porque então não utilizar o *Kanban* para este controle de mudanças sendo que este *framework* objetiva isso?**

- O *Planning Poker* utiliza-se de uma reunião entre a equipe para poder medir o esforço e o tempo para cada funcionalidade e não apenas isso ele também faz com que a equipe interaja entre si e aqueles mais experiente possam ensinar que algo que pareça complexo para uma determinada pessoa do grupo ao final não é algo tão complexo assim e ressaltando a página anterior **não é isso o que a XP aborda sobre simplicidade presumida e ainda ajuda a definir com maior segurança sobre o tempo de cada *Sprint* no *SCRUM*, além de definir a complexidade de cada item?**

- O *SCRUM* utiliza-se de reuniões diárias chamadas de *Daily Scrum* a cada Sprint com o intuito de levantar questões como, **o que fiz ontem? O que farei hoje? Possui algum tipo de impedimento?** Exatamente como as reuniões diárias da XP conhecidas como *Stand Up Meeting*, levando em conta que pelo XP utilizar programação em

pares, ele utiliza-se de testes, verificação de código e melhores técnicas para determinadas abordagens, no qual o *SCRUM* necessitaria de uma nova *Sprint* para fazer a mesma coisa, **por que então não utilizar-se dos itens 5.1.3 ao 5.1.9 da XP em cada *Sprint* do *Scrum* já que isto proporcionaria um resultado mais rápido, *Sprints* mais curtas combinado com o controle dos processos do *SCRUM*?**

- A XP utiliza-se de um ritmo consideravelmente rápido para cada entrega de seus *releases*, necessitando assim de uma equipe mais madura e experiente, o que muitas vezes é difícil de montar uma equipe totalmente assim, **porque então não reduzir um pouco o ritmo e utilizar o *SCRUM* em conjunto para que mesmo com uma equipe menos experiente, consiga entregar resultados de alto desempenho e com um controle alto sobre cada um de seus processos?**

Ficou claro? Os métodos ágeis tem como o objetivo entregas rápidas e de alta qualidade, mas elas não precisam ser seguidas à risca, pode-se utilizar as partes que mais se adequam a sua empresa, ou até mesmo juntar as várias metodologias para que cada uma compense o que falta na outra, isto tudo não é rápido ou até mesmo simples de se fazer, leva-se tempo e disciplina, sobre isso se trata o *Kaizan* que tem como seu conceito o aprimoramento contínuo, é uma ótima ferramenta para equipes em geral e se adequa a todas essas que foram descritas nesse artigo. Por fim o *Gemba*, o *Gemba* tem como seu objetivo ir ao local real do problema, pois, conhecer o problema por um *email* ou telefonema nos mostra apenas 10% do que o realmente ele é e obter essa mentalidade numa equipe de *software* não só pode diminuir os problemas dos projetos, as constantes mudanças, como também irá proporcionar ao seu cliente uma maior satisfação no produto final.

6. ABSTRACT

Time is something priceless and every day is becoming even more important for people and organizations to this end have arisen agile methodologies to ensure delivery of the software quickly and efficiently, so that is no use only themselves entitle an enterprise agile but demonstrate this to their customers, yet it is still something that just happens in software companies. It is on this basis that was made this article, it has the intention to illustrate some techniques for better software management, that is, this article is not meant to make literature review in the topics discussed, to deepen more on matters applied here are links at the end of the article.

***Keywords**: Agile; management; Projects; software development; time;*

7. REFERÊNCIAS BIBLIOGRÁFICAS

http://www.scruminc.com/wp-content/uploads/2015/06/Scrum-at-Scale-Keynote-Jun-2015.pdf. Acesso em Outubro, 2015.

Universidade Estadual do Norte Fluminense Darcy Ribeiro. Dica de Leitura. Disponível em: <http://revista.ibict.br/ciinf/index.php/ciinf/article/viewArticle/1858>. Acesso em Outubro, 2015.

Kanban em 10 passos. Dica de Leitura. Disponível em: <http://www.ramosdainformatica.com.br/files/Apostilas/Kanban10Passos.pdf>. Acesso em Outubro, 2015.

http://www.culturaagil.com.br/kanban-do-inicio-ao-fim/ - imagem figura 1

Mahnic, V. and Hovelja, T. (2011) ***"On using planning poker for estimating user stories"***. *Journal of Systems and Software*, 85(9):2086 - 2095. *Selected papers from* 2011 *Joint Working IEEE/IFIP Conference on Software Architecture* (WICSA 2011).

Raith, F., Richter, I., Lindermeier, R., and Klinker, G. (2013) ***"Identification of inaccurate effort estimates in agile software development"***. *In Software Engineering Conference* (APSEC, 2013 20th Asia-Pacific, *volume* 2, pág. 67–72.

Revista Computação Aplicada. Dica de Leitura. Disponível em: <http://revistas.ung.br/index.php/computacaoaplicada/article/viewFile/1408/1194>. Acesso em Outubro, 2015.

BECK, K. **Extreme *Programming Explained*.** Boston, Massachusetts: Addison-Wesley Longman, 2000.

BECK, K. **Extreme *Programming Explained*: *Embrace Change*. 2. ed. *Reading*,** Massachusetts: Addison-Wesley, 2004.

WELLS, D. ***Extreme Programming*. *A Gentle Introduction*, 2009.:** Acesso em: Outubro, 2015.

***Extreme Programming and RUP*.** Dica de Leitura. Disponível em: <http://essentiaeditora.iff.edu.br/index.php/vertices/article/view/1809-2667.20130035/2986>. Acesso em Novembro, 2015.

CARDOSO, Carlos H. R. **Aplicando práticas de *Extreme Programming* (XP) em equipes SWCMM nível 2.** Centro universitário SENAC e Centro de pesquisas Renato Archer, São Paulo,2004. Disponível em:

http://www.simpros.com.br/simpros2004/Apresentacoes_PDF/Artigos/
Art_05_Simpro. Acesso em: Outubro, 2015.

TELES, Vinícius M. ***Extreme Programming*.** São Paulo: Novatec Editora, 2004.

TELES, Vinícius M. **Um estudo de caso da adoção das práticas e valores do *Extreme Programming*. 181 p. Dissertação (Mestrado em Informática)** – Universidade Federal do Rio de Janeiro, UFRJ, 2005.

GONÇALVES JR, José Pinto. **O uso da Metologia XP no Desenvolvimento de *Software* e os impactos na Gestão de Riscos. 46p. Monografia (Sistemas de**

Informação) - Centro Universitário da Fundação Octávio Bastos, São João da Boa Vista, 2009.