Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

ITALO PAOLO SATURNINO DE JESUS

METODOLOGIAS ÁGEIS DE GERENCIAMENTO DE PROJETOS PARA STARTUPS

Belo Horizonte
2016

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Especialização em Informática: Ênfase em Engenharia de Software

METODOLOGIAS ÁGEIS DE GERENCIAMENTO DE PROJETOS PARA STARTUPS

por

ITALO PAOLO SATURNINO DE JESUS

Monografia de Final de Curso

Profa. *Gisele Lobo Pappa*
Orientadora

Belo Horizonte
2016

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Jesus, Italo Paolo Saturnino de.

J58m Metodologias ágeis de gerenciamento de projetos para startups. / Italo Paolo Saturnino de Jesus. Belo Horizonte, 2016.
x, 53 f.: il.; 29 cm.

Monografia (especialização) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientadora: Gisele Lobo Pappa.

1. Computação 2. Engenharia de software. 3. Software - desenvolvimento 4. Empreendedorismo. I. Orientadora. II. Título.

CDU 519.6*32(043)

ITALO PAOLO SATURNINO DE JESUS

METODOLOGIAS ÁGEIS DE GERENCIAMENTO DE PROJETOS PARA STARTUPS

Monografia apresentada ao Curso de Especialização em Informática do Departamento de Ciências Exatas da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Especialista em Informática.

Área de concentração:

Orientadora: Profa. *Gisele Lobo Pappa*

Belo Horizonte

2016

UNIVERSIDADE FEDERAL DE MINAS GERAIS

INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
ESPECIALIZAÇÃO EM INFORMÁTICA: ÁREA DE CONCENTRAÇÃO ENGENHARIA DE SOFTWARE

METODOLOGIAS ÁGEIS DE GERENCIAMENTO DE PROJETOS PARA STARTUP

ÍTALO PAOLO SATURNINO DE JESUS

Monografia apresentada aos Senhores:

Profa. Gisele Lobo Pappa
Orientadora
DCC - ICEx - UFMG

Prof. Rodolfo Sérgio Ferreira de Resende
DCC - ICEx - UFMG

Belo Horizonte, 29 de fevereiro de 2016

AGRADECIMENTOS

Agradeço primeiramente a Deus, por ter me dado o dom da vida e a oportunidade de concluir mais esta etapa.

Agradeço aos meus pais, por todo incentivo, apoio e dedicação para que eu sempre pudesse alcançar meus objetivos. À minha namorada, pela paciência e apoio para chegar até aqui.

Por fim, agradeço aos professores, em especial à professora Gisele Lobo Pappa, por aceitar me orientar neste trabalho e também aos colegas de curso que estiveram ao meu lado durante esta jornada.

RESUMO

O planejamento e gerenciamento dos processos de desenvolvimento de *software* são fatores importantes e contribuem fortemente para o sucesso dos projetos. A utilização de metodologias para auxiliar no desenvolvimento de projetos de *software* é um tema bastante discutido. Existem várias metodologias diferentes disponíveis para a escolha, que deve basear-se nas características do projeto sendo iniciado. Dentre estas escolhas temos as metodologias ágeis, que tem o propósito de desburocratizar e agilizar o processo de desenvolvimento, garantindo uma maior satisfação dos clientes e o desenvolvimento de produtos de qualidade entregues dentro do prazo estipulado. Este trabalho apresenta um estudo sobre as principais metodologias ágeis (XP e Scrum), e propõe uma metodologia híbrida para gerenciamento de projetos baseado em seus princípios e fundamentos, adaptado para *startups*. *Startups* possuem certas características e limitações que podem impedir a utilização convencional destas metodologias, como o tamanho da equipe. Espera-se, com esse trabalho, que a metodologia proposta possa agregar valor nos processos de desenvolvimento de *software* de *startups*, permitindo que essas empresas possam ganhar vantagem competitiva no mercado.

Palavras-chave: Metodologias Ágeis, XP, Scrum, *startups*, desenvolvimento de *software*.

ABSTRACT

The planning and management of the software development process is an important factor that contributes greatly to the success of projects. The use of methodologies to assist software development projects is a widely discussed topic. There are several different methods available to choose from, including agile ones. Agile methodologies aim to reduce the bureaucracy and improve the streamline of the software development process, ensuring greater customer satisfaction and the delivery of quality products within the stipulated delivery time. This dissertation presents a study of the main agile methodologies (XP and Scrum) and proposes a hybrid methodology for project management in startup based on the principles and foundations of agile methodologies. The traditional methodologies need to be adapted because startups present certain limitations that may prevent the use of these conventional methodologies. We expect that the proposed hybrid methodology can add value to startup processes, making improvements in software development, and allowing competitive advantage for these companies in the marketplace.

Keywords*:* Agile Methodologies, XP, Scrum, startups, software development.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ASD *Adaptive Software Development* (Desenvolvimento Adaptável de Software)

CASE *Computer-aided software engineering* (Engenharia de Software assistida por computador)

FDD *Feature Driven Development* (Desenvolvimento Dirigido a Características)

MVP *Minimum Viable Product* (Produto Mínimo Viável)

PO *Product Owner* (Dono do Produto)

XP *eXtreme Programming* (Programação Extrema)

SUMÁRIO

1 INTRODUÇÃO

A utilização de processos durante o planejamento e a execução de projetos de desenvolvimento de *software* é fundamental para que sejam alcançados os resultados esperados, produzindo *software* de qualidade, respeitando o custo e o prazo estipulados. A utilização de metodologias de desenvolvimento consagradas no mercado contribui para que estes resultados sejam alcançados de forma satisfatória.

Nos últimos anos as metodologias ágeis têm ganhado importância no cenário de desenvolvimento de *software*. A principal diferença entre estas e as metodologias tradicionais está nos princípios utilizados para atingir os objetivos do projeto. O planejamento detalhado é feito apenas para a fase atual do projeto, e os planos para fases futuras são considerados rascunhos, que podem ser alterados de acordo com o aprendizado da equipe e a evolução do projeto (SATO, 2007).

Durante o desenvolvimento de *software,* é comum que os requisitos definidos inicialmente pelo usuário mudem. Normalmente as mudanças ocorrem devido a evolução do negócio ao qual o *software* está atrelado, e o custo para alteração de um requisito após a conclusão de um *software* é considerado muito alto. Um dos princípios das metodologias ágeis é aceitar as mudanças, mesmo que isto ocorra em estágios mais avançados do desenvolvimento, aceitando que tais mudanças podem trazer vantagem competitiva para o cliente (SATO, 2007).

Além da aceitação de mudanças, as metodologias ágeis propõem outros princípios que trazem mais dinamismo para o processo de desenvolvimento, incluindo a proximidade com o cliente - priorizando sua satisfação, e a motivação da equipe, para que desempenhem seus papéis da melhor forma possível.

A maioria dos processos de desenvolvimento de *software* foi concebida para empresas já bem estruturadas. Porém, com o crescimento do número de *startups* ao redor do mundo, processos mais apropriados para o perfil dessas empresas tiveram que ser pensados. A adoção de metodologias ágeis em *startups* pode contribuir para sua formação e crescimento por se tratarem de métodos adaptativos. Para Gitahy (2010), *startups* buscam um modelo de negócios em um ambiente de incertezas.

Isso pode significar que mudanças sejam necessárias para que se alcance um modelo de negócios mais sólido.

Este trabalho apresenta um estudo sobre metodologias ágeis, com enfoque nas duas principais metodologias: XP e Scrum. Focamos nessas duas metodologias por elas serem as mais conhecidas e utilizadas no mercado. Além delas, também apresentaremos a metodologia *Lean Startup,* que tem alguns conceitos valiosos para *startups.* Com base nas práticas das metodologias mencionadas, propomos uma metodologia híbrida adaptada para o ambiente de *startup*, e discutimos sua utilização com um estudo de caso em uma empresa específica, que será chamada de Startup X, e não terá seu nome revelado por questões de privacidade.

1.1 Objetivos

1.1.1 Geral

Fazer um estudo das metodologias ágeis e propor uma metodologia híbrida baseada em seus princípios que dê suporte a *startups* que trabalham com desenvolvimento de *software,* para que essas possam gerenciar seus projetos de maneira mais descomplicada obtendo melhores resultados.

1.1.2 Específicos

- Pesquisar sobre metodologias ágeis de gerenciamento de *software;*
- Entender o conceito de *startup* e suas diferenças para outras empresas;
- Identificar os princípios das metodologias ágeis que melhor se encaixam nas necessidades de *startup*s;
- Propor uma metodologia híbrida de gerenciamento de projetos baseado em metodologias ágeis para *startups*;
- “Implantar” a metodologia proposta em uma empresa que servirá de estudo de caso para avaliar os resultados obtidos.

1.1 Justificativa

O gerenciamento de projetos de desenvolvimento de *software* se mostra muito importante, sendo fator chave para o sucesso dos projetos. *Startups*, por terem características específicas, podem encontrar dificuldades para implantar determinadas metodologias, que não consideram as limitações deste tipo de empresa. O trabalho se justifica pela criação de uma metodologia híbrida que contempla as características das metodologias ágeis que se adequam à realidade do contexto de *startups*. Em especial, a *startup* que motivou o tema deste trabalho e que aparece no estudo de caso dessa dissertação não utilizava nenhuma metodologia para gerenciamento do desenvolvimento de *software* por não encontrar uma metodologia apropriada dentre as mais utilizadas pelas grandes corporações.

1.2 Metodologia

Este trabalho foi desenvolvido por meio de estudos bibliográficos sobre metodologias ágeis e gerenciamento de projetos, seguido da descrição dos principais processos das metodologias analisadas. O estudo bibliográfico serve de base para a geração da metodologia híbrida de gerenciamento de projetos proposto pelo trabalho.

Após a parte inicial de pesquisas e análise uma metodologia híbrida baseada nos processos das metodologias ágeis e que leve em consideração os recursos existentes na *startup* foi proposta, seguido de um estudo de caso com uma *startup* que adotou a metodologia proposta.

2 METODOLOGIAS ÁGEIS

O desenvolvimento de *software* não é uma tarefa simples. Ao se iniciar um projeto de desenvolvimento de *software*, deve-se considerar que ao final de todos os processos seja gerado um produto de qualidade, que atenda amplamente aos interesses e necessidades do cliente, e que seja entregue dentro do prazo e do orçamento previstos. Atingir estas exigências é o principal desafio do processo de desenvolvimento de *software*.

Segundo Sommerville (2011), na década de 1980 e início da década de 1990, acreditava-se que a melhor maneira de desenvolver um *software* era por meio de um planejamento cuidadoso do projeto, com qualidade da segurança formalizada, uso de ferramentas CASE (*Computer-aided software engineering*) e de um processo de desenvolvimento rigoroso e controlado.

Softwares desenvolvidos desta forma exigem muito tempo e esforço em planejamento, projeto e documentação do sistema. Assim, essa abordagem se justifica em grandes projetos, com necessidade de gerenciamento e coordenação de várias equipes distintas, ou em sistemas críticos.

Ainda de acordo com Sommerville (2011), essa abordagem pesada de desenvolvimento dirigida a planos, quando aplicada a sistemas de pequeno e médio porte, nem sempre gera resultados satisfatórios. Essa insatisfação com estes tipos de abordagens levou um grande número de profissionais, na década de 1990, a proporem “metodologias ágeis” de desenvolvimento de *software*.

No início de 2001, profissionais especialistas em desenvolvimento de *software* se reuniram nos Estados Unidos para discutir alternativas às metodologias tradicionais de desenvolvimento. Estes especialistas não eram contra métodos, processos ou metodologias, e defendiam a modelagem e a documentação do *software*, mas não em excesso. Defendiam também o planejamento, mas reconheciam os limites do planejamento e da previsibilidade (DIAS, 2010).

De acordo com Dias (2010), a essência desse movimento foi a definição de um novo enfoque de desenvolvimento de *software* baseado na agilidade, flexibilidade, habilidade de comunicação e na capacidade de oferecer novos produtos e serviços

de valor em curtos períodos de tempo. Para Dias (2010), a agilidade não deve ser interpretada como falta de estrutura, mas sim como a habilidade de criar e responder a mudanças.

Como resultado desse encontro foi criada a Aliança Ágil e o estabelecimento do Manifesto Ágil, que é baseado em quatro valores:

> "Nós estamos descobrindo melhores maneiras para desenvolver software, praticando e auxiliando os outros a fazê-lo. Através deste trabalho nós valorizamos:
>
> - **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
>
> Ou seja, embora haja valor nos itens à direita, nós valorizamos mais os itens à esquerda." (BECK *et. al.*, 2001).

Para auxiliar a compreensão destes valores, foram definidos doze princípios para a prática de metodologias ágeis, conforme apresentado na Tabela 1.

Princípios
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 ter vantagens competitivas com o produto.
Entregar *software* funcionando com frequência, na escala de semanas até meses, dando preferência aos períodos mais curtos.
Pessoas relacionadas à 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, confiando que farão seu trabalho.
O método mais eficiente e eficaz de transmitir informações para fora ou 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*, aumentando 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, e então se ajustam e otimizam seu comportamento de acordo.

Tabela 1 - Princípios das metodologias ágeis de desenvolvimento de *software* (FONTE: Manifesto Ágil, 2001)

Segundo estes princípios, metodologias ágeis baseiam-se em uma abordagem incremental para a especificação, o desenvolvimento e a entrega do *software* (SOMMERVILLE, 2011). Eles se adequam melhor a mudanças de requisitos durante o processo de desenvolvimento do projeto, e tem como objetivo entregar o *software* em funcionamento aos clientes rapidamente e reduzir a burocracia do processo existente nas metodologias tradicionais, evitando gastar tempo com documentações que possivelmente não serão utilizadas pelo cliente (SOMMERVILLE, 2011).

Seguindo os princípios definidos pelo manifesto ágil, várias metodologias foram criadas para auxiliar o desenvolvimento de *software*, entre elas o XP (*eXtreme Programming* – Programação Extrema), o Scrum, FDD (*Feature Driven Development* – Desenvolvimento Dirigido a Características), ASD (*Adaptive Software Development* – Desenvolvimento Adaptável de *Software*), entre outras metodologias.

O presente trabalho irá focar nas duas metodologias mais utilizadas, que são o XP e o Scrum. A metodologia XP foi criada na década de 90 e vem fazendo sucesso em diversos países, ajudando equipes de desenvolvimento a criarem sistemas de qualidade, em menos tempo e com menos recursos que o habitual (TELES, 2005). O Scrum é um *framework* utilizado para gerenciar o desenvolvimento de produtos, dentro do qual é possível empregar vários processos e técnicas (SCHWABER; SUTHERLAND, 2013).

2.1 XP (*eXtreme Programming*)

Beck (1999) define *Extreme Programming* (Programação Extrema) como sendo uma metodologia leve, eficiente e de baixo risco, com forma flexível, previsível, científica e apropriada para pequenas e médias equipes de desenvolvimento de *software*, onde os requisitos são vagos ou mudam rapidamente.

XP é talvez a mais conhecida e utilizada das metodologias ágeis (SOMMERVILLE, 2011). Ele vem ganhando grande número de adeptos por oferecer condições para que os desenvolvedores respondam com eficiência a mudanças no projeto, mesmo que em estágios finais do ciclo de vida do processo (SOUZA, 2007).

Beck (1999) define o XP como uma disciplina de desenvolvimento de *software*, e cita que para utilizar a metodologia é preciso seguir o que o XP recomenda em termos de suas práticas. Não se pode escolher seguir ou não estas práticas. Caso elas não sejam seguidas, não se estará utilizando a metodologia XP.

O XP foi concebido para trabalhar com projetos que podem ser construídos por equipes pequenas, constituídas por dois a dez programadores, e onde um trabalho razoável de execução de testes pode ser feito em pelo menos um dia (BECK, 1999). Os integrantes da equipe devem estar cientes de todas as fases do desenvolvimento, devem estar interessados no projeto e ser pró-ativos, para garantir a alta produtividade da equipe. Além disso, o cliente precisa estar disponível sempre que solicitado para sanar as dúvidas e tomar decisões a respeito do projeto (SOUZA, 2007).

De acordo com Beck (1999), o XP segue um conjunto definido de valores, princípios e práticas, buscando alcançar eficiência e efetividade durante o processo de desenvolvimento, conforme descrito na próxima seção.

2.1.1 Os Valores do XP

Segundo Beck (1999), para se obter o sucesso em um projeto, deve ser adotado um estilo que contempla um conjunto consistente de valores que satisfazem tanto as

necessidades humanas quanto as comerciais. Os quatro principais valores adotados pelo XP serão apresentados a seguir:

- **Comunicação:** é o primeiro valor do XP. Problemas em projetos ocorrem invariavelmente porque alguém deixou de comunicar algo a outra pessoa. XP tem por objetivo manter a comunicação fluindo de forma rápida e eficaz entre os envolvidos no projeto, pois emprega muitas práticas que não podem ser feitas sem comunicação.
- **Simplicidade:** XP aposta que fazer o mais simples e futuramente gastar um pouco mais para fazer mudanças é mais vantajoso que fazer algo mais complexo, mas que pode nunca ser utilizado. Simplicidade e comunicação tem uma relação de apoio mútuo muito importante. Quanto mais comunicação houver, melhor será o entendimento do que precisa ser feito, o que implicará na simplicidade do projeto.
- ***Feedback:*** quanto mais rápido for o *feedback* do cliente, mais rápido será identificado se o projeto está seguindo na direção correta e, caso não esteja, o esforço para efetuar correções será menor. *Feedback* concreto trabalha junto com a comunicação e simplicidade. Quanto mais *feedback*, mais fácil se torna a comunicação.
- **Coragem*:*** combinada com os três valores anteriores, a coragem se torna extremamente valiosa. É preciso coragem para inovar, para propor mudanças, para pedir ajuda quando necessário, para comunicar problemas ao cliente e encarar mudanças no projeto.

Além dos quatro valores já mencionados, alguns autores consideram também um quinto valor, que seria o “Respeito”. Para Teles (2005), este é o mais básico dos valores e que dá sustentação aos demais. Respeitar os outros membros da equipe é essencial para o sucesso de um projeto de *software*, e somente com o respeito os outros valores poderão ser amplamente seguidos.

2.1.2 As práticas

As práticas representam o que as equipes XP fazem diariamente. Por si só, as práticas não fazem muito sentido, mas adicionando a elas um propósito, direcionado por um conjunto de valores, elas passam a fazer sentido. A adoção das práticas contribui para que o projeto caminhe em direção a um estado ideal de desenvolvimento eficaz, mas as práticas precisam ser usadas em conjunto, para que seja possível verificar melhorias no processo de desenvolvimento (TELES, 2005).

A seguir serão apresentadas as práticas do XP propostas por Kent Beck.

- **Cliente Presente:** a falta de participação do cliente é apontada como um dos principais fatores gerador de falhas no desenvolvimento de *software*. O XP se preocupa com essa questão e procura trazer o cliente para próximo do projeto. A presença do cliente ao longo do projeto viabiliza o contínuo *feedback* entre ele e a equipe de desenvolvimento, permitindo que pequenas mudanças sejam feitas ao longo do desenvolvimento de forma rápida. Isto também facilita no entendimento mais preciso da ideia que o cliente deseja transmitir para a equipe. E essa proximidade também melhora a relação de confiança entre as partes envolvidas (TELES, 2005).
- **Código coletivo:** equipes XP praticam o conceito de código coletivo, onde o desenvolvedor tem acesso a todas as partes do código (inclusive as que não são de sua autoria) e tem o direito de fazer alterações que julgar necessárias nesses códigos sem precisar da permissão de seu autor, assim como outros desenvolvedores podem alterar seu código sem que lhe seja solicitada permissão (TELES, 2005).

  Ainda segundo Teles (2005), alterações no código podem gerar erros. Portanto, a prática de código coletivo só pode ser adotada com segurança se a equipe utilizar testes automatizados para garantir o efeito das alterações.

  Esta prática ajuda a tornar a equipe mais robusta, pois os desenvolvedores se habituam a trabalhar em diferentes partes do sistema. Desta forma, o desenvolvimento não é fortemente afetado pela ausência de algum dos membros da equipe. Isto também permite que a equipe avance mais rapidamente, pois ninguém precisa esperar que outra pessoa tenha

disponibilidade para consertar alguma coisa, podendo ele mesmo resolver os problemas no código (TELES, 2005).

- **Padrão de código:** Teles (2005) diz que a adoção de um padrão ajuda a simplificar a comunicação, a programação em pares e a tornar o código coletivo. Ainda, como o código passa por vários níveis de revisão, ele facilita a detecção e correção de códigos fora do padrão estabelecido. Para Beck (1999), a adoção de um padrão deve ser realizada voluntariamente pelos membros do time e, deve facilitar a comunicação entre o time.
- **Projeto Simples:** "Cada projeto é realizado para atender às necessidades atuais, e nada mais (SOMMERVILLE, 2011)". Para Beck (1999), cada peça de *design* no sistema deve ser capaz de justificar sua existência. Para ilustrar o exemplo de projeto simples, o autor cita a construção de um gráfico, onde são inseridas as informações que se deseja exibir, e também elementos gráficos para facilitar a compreensão das informações. Estes elementos de *design* inseridos devem poder ser removidos a qualquer momento sem que o gráfico perca sua essência e se mantenha consistente. O projeto simples deve ser assim, qualquer elemento de *design* deve estar desatrelado das informações do sistema e pode ser removido sem que viole a consistência do projeto.
- ***Releases* curtos:** "O XP considera que um projeto de *software* é um investimento. O cliente investe uma certa quantidade de recursos na expectativa de obter um retorno dentro de certo prazo (TELES, 2005)."

  O XP trabalha com a prática de *releases* curtos, o que consiste em entregar ao cliente pequenas porções funcionais do sistema em prazos curtos. As primeiras iterações buscam entregar para o cliente versões do sistema com as principais funcionalidades, que agregam maior valor ao negócio. Com isso o cliente tem uma visão do que esperar do sistema mais rapidamente, podendo tomar decisões importantes do projeto ainda nas fases iniciais de programação, onde ainda não foram gastos muito tempo e recursos (TELES, 2005).
- **Jogo do planejamento:** no XP o desenvolvimento é feito em iterações e no início de cada iteração os desenvolvedores e cliente se reúnem para priorizar as funcionalidades. Nesta reunião o cliente define as prioridades e os

desenvolvedores fazem as estimativas (SOUZA, 2007). Para Teles (2005) as funcionalidades de sistemas podem ser categorizadas em “tem que ser feita”, “deveria ser feita” e “poderia ser feita”, e o planejamento é usado para garantir que a equipe esteja realmente trabalhando no que é mais importante para o projeto.

O XP busca dividir o tempo disponível utilizando os conceitos de *releases* e iterações. *Releases* tomam alguns poucos meses e se dividem em iterações. Um *release* representa um marco no tempo em que um conjunto de funcionalidades é finalizado e lançado para utilização pelos usuários. Iterações tem duração média de duas semanas e se dividem em tarefas que duram alguns dias. Ao final de uma iteração o cliente e os desenvolvedores avaliam as funcionalidades produzidas e as prioridades para a próxima iteração (TELES, 2005).

- **Metáfora:** para Beck (1999) o projeto no XP é guiado por uma metáfora abrangente, que ajuda os envolvidos no projeto a entenderem os elementos básicos e seus relacionamentos, facilitando a comunicação e o entendimento dos requisitos. “A metáfora no XP substitui muito daquilo que outras pessoas chamam de ‘arquitetura’ (BECK, 1999)”.
- **Programação em pares:** esta prática consiste em duas pessoas produzindo códigos do sistema olhando para uma máquina, com um teclado e um *mouse*. Cada um desempenha um papel, enquanto um codifica, o outro está analisando, inspecionando e pensando estrategicamente se o que está sendo produzido irá funcionar ou se existe outra maneira mais simples para realizar aquela tarefa (BECK, 1999). A programação em pares é dinâmica, e os pares são trocados com frequência.

  “A programação em par também ajuda a elevar a motivação dos desenvolvedores, tornando o trabalho mais divertido e facilitando a comunicação (TELES, 2005)”.
- **Desenvolvimento orientado a testes:** um dos problemas mais caros e recorrentes no desenvolvimento de *software* é a incidência de defeitos. Quando um defeito é encontrado, precisa-se depurar o sistema para identificar a causa e fazer sua correção. Porém, a depuração é demorada, pois os programadores se esquecem do que fizeram com o tempo, tornando o

processo mais moroso. Outro problema a se considerar é que, na correção de um *bug*, a chance de outro *bug* ser inserido no sistema pode chegar a 50% (TELES, 2005). Segundo Teles (2005), o XP lida com esses problemas adotando um mecanismo de testes automatizados, onde os testes são escritos antes da codificação. São dois tipos de testes: os de unidade, que tenta assegurar que o componente funcione corretamente, e os de aceitação, que busca testar a interação entre os componentes. Os testes de unidade são importantes para o programador ter um *feedback* mais rápido, e evitar que erros passem muito tempo despercebidos. Já os testes de aceitação, são utilizados para saber se o sistema funciona como um todo, garantindo que o sistema responda corretamente cada entrada que recebe.

Portanto, no XP os testes automatizados são muito importantes e ajudam a evitar desperdício de tempo solucionando problemas que poderiam ser identificados mais cedo e corrigidos com menor custo.

- **Refatoração:** a refatoração é o processo de realizar mudanças em um código já existente sem alterar seu comportamento externo. Essa mudança pode ser a inserção de um novo método ou a simplificação de métodos, mantendo ainda todos os testes executando (BECK, 1999). Para Beck (1999), isso significa que por vezes se irá trabalhar mais do que o necessário, mas esse trabalho extra facilitará a inserção da função no código.

  De acordo com Teles (2005), normalmente não é alocado tempo específico para refatoração, mas pequenas refatorações são realizadas constantemente no código. Existem situações que indicam a necessidade de refatoração, por exemplo, a existência de código duplicado, quando o código ou sua intenção não estão claras, ou quando são detectados problemas no código (os chamados *bad smells).*

- **Integração contínua:** o código é integrado e testado pouco tempo após ser desenvolvido. O par que o desenvolveu baixa a versão atual, confere as modificações, corrige problemas de colisão e executa os testes até que eles passem. Realizando a integração desta forma, fica óbvio saber quem deve fazer as correções quando o teste falha, pois o código estava funcionando anteriormente (BECK, 1999).

- **Semana de 40 horas:** o XP recomenda que os membros da equipe trabalhem apenas durante o tempo regulamentar e evitem fazer horas extras (TELES, 2005). Beck (1999) diz que hora extra é o sintoma de um problema no projeto, e recomenda que não se trabalhe duas semanas seguidas com horas extras. Segundo o autor, quando necessário trabalhar duas semanas com horas extras para alcançar os objetivos, então existe um problema que não pode ser solucionado com horas extras. Beck (1999) também recomenda que os membros da equipe tirem férias por pelo menos duas semanas consecutivas durante o ano, com pelo menos mais uma ou duas semanas disponíveis para pausas mais curtas.

2.2 Scrum

Schwaber e Sutherland (2013) definem o Scrum como um *framework* para desenvolver e manter produtos complexos, onde é possível tratar e resolver problemas complicados e adaptativos, e entregar produtos com o mais alto valor possível.

Scrum não é um processo ou técnica para construir produtos, mas sim um *framework* dentro do qual vários processos ou técnicas podem ser empregados. Ele vem sendo utilizado desde o início da década de 1990. O Scrum se baseia em times do Scrum associados a papéis, eventos, artefatos e regras, onde cada componente serve a um propósito específico. Estes componentes são essenciais para o uso e o sucesso do Scrum (SCHWABER; SUTHERLAND, 2013).

“Scrum emprega uma estrutura iterativa e incremental da seguinte maneira: no início de cada iteração, a equipe analisa o que deve ser feito e então seleciona aquilo que acreditam poder se tornar um incremento de valor ao produto ao final da iteração (LIBARDI; BARBOSA, 2010)”. A equipe trabalha focada nas funcionalidades escolhidas para esta iteração e ao final apresenta esse incremento de para que os *stakeholders* possam verificar e solicitar alterações (LIBARDI; BARBOSA, 2010).

Segundo Schwaber e Sutherland (2013), são três os pilares que apoiam a implementação de controle de processo:

- **Transparência:** os aspectos significativos do processo devem estar visíveis aos responsáveis pelos resultados. Estes aspectos devem ser definidos por um padrão comum, que facilite o entendimento dos observadores e que todos tenham o mesmo entendimento do que está sendo visto;
- **Inspeção:** os artefatos Scrum devem ser frequentemente inspecionados para identificar variações. A frequência deve ser definida em um fator que não atrapalhe a execução das tarefas.
- **Adaptação:** caso seja detectado que um aspecto do processo desviou e o produto resultante será algo inaceitável, o aspecto observado deve ser ajustado. Este ajuste deve ser realizado o mais breve possível para minimizar os desvios.

Como mencionado anteriormente, o Scrum define eventos, papéis, artefatos e regras para guiarem a utilização do *framework,* conforme apresentado a seguir.

2.2.1 O Time Scrum

“O Time Scrum é composto pelo *product owner*, o time de Desenvolvimento e o Scrum *master*. Times Scrum são auto-organizáveis e multifuncionais (SCHWABER; SUTHERLAND, 2013)”. Times multifuncionais possuem competências necessárias para completar o trabalho sem depender de outros que não fazem parte da equipe.

Os times Scrum fazem entregas de produto de forma iterativa e incremental, garantindo que uma versão potencialmente funcional do produto esteja sempre disponível (SCHWABER; SUTHERLAND, 2013). Os papéis de cada um dos membros do time são definidos a seguir.

2.2.1.1 *Product Owner* (Dono do Produto)

“O *product owner* é responsável por representar os interesses dos *stakeholders* no projeto. O product owner é a pessoa que define todos os itens de requisito do projeto numa lista chamada *Product Backlog* (LIBARDI; BARBOSA, 2010)”. Também é responsável por maximizar o valor do produto e do desempenho do Time de Desenvolvimento (SCHWABER; SUTHERLAND, 2013).

O dono do produto é a única pessoa responsável por gerenciar o *backlog* do produto. Dentre as tarefas de gerenciamento estão: ordenar os itens do *backlog* do produto para alcançar as metas e missões; expressar claramente os itens do *backlog* do produto e garantir que o time de desenvolvimento entenda todos os itens; garantir que o *backlog* do produto seja visível, transparente e claro para todos, além de mostrar o que o time Scrum vai trabalhar a seguir (SCHWABER; SUTHERLAND, 2013).

2.2.1.2 Time de Desenvolvimento

O time de desenvolvimento no Scrum é tipicamente composto por um grupo de 6 a 10 pessoas capacitadas para realizar o trabalho e entregar uma versão do produto ao final de cada período estipulado para o desenvolvimento. Todos trabalham juntos, não existe necessariamente uma divisão funcional baseada em papéis tradicionais, e são responsáveis por completar o conjunto de trabalho a cada iteração (LIBARDI; BARBOSA, 2010).

2.2.1.3 Scrum *Master* (Mestre do Scrum)

“O Scrum *Master* é responsável por garantir que o Scrum seja entendido e aplicado. O Scrum *Master* faz isso para garantir que o time scrum adere à teoria, práticas e regras do Scrum. O Scrum *Master* é um servo-líder para o time scrum (SCHWABER; SUTHERLAND, 2013)”.

O mestre do Scrum contribui com o dono do produto, ajudando-o a encontrar técnicas para o gerenciamento do *backlog* do produto, comunicar claramente a visão, objetivos e itens do *backlog* do produto ao time de desenvolvimento, compreender e praticar agilidade e a facilitar a ocorrência dos eventos Scrum (SCHWABER; SUTHERLAND, 2013).

Já para o time de desenvolvimento, o mestre do Scrum os serve treinando em autogerenciamento e interdisciplinaridade, removendo impedimentos para o progresso, auxiliando na criação de produtos de alto valor e também a facilitar os eventos Scrum (SCHWABER; SUTHERLAND, 2013).

A Figura 1 mostra as principais responsabilidades do mestre do Scrum dentro do projeto, e fica visível a importância de seu papel de liderança e gerenciamento para o sucesso do projeto.

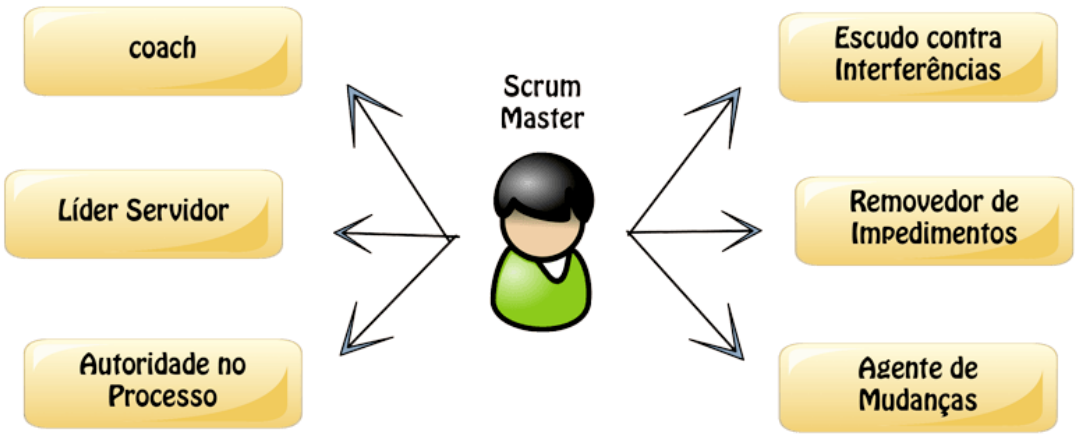

Figura 1 - Principais responsabilidades do Scrum Master (Fonte: http://www.mindmaster.com.br/scrum-master/)

2.2.2 Eventos Scrum

Os eventos são prescritos e usados no Scrum com o objetivo de criar uma rotina e minimizar a necessidade de eventos não definidos pelo *framework.* Os eventos Scrum tem duração máxima definida e cada evento é uma oportunidade de inspecionar e adaptar alguma coisa (SCHWABER; SUTHERLAND, 2013).

2.2.2.1 Sprint

O coração do Scrum é o *sprint*, que tem duração média de um mês, onde uma versão potencialmente utilizável do produto é criada para ser entregue ao cliente. Um novo *sprint* é iniciado imediatamente após o término do anterior (SCHWABER; SUTHERLAND, 2013).

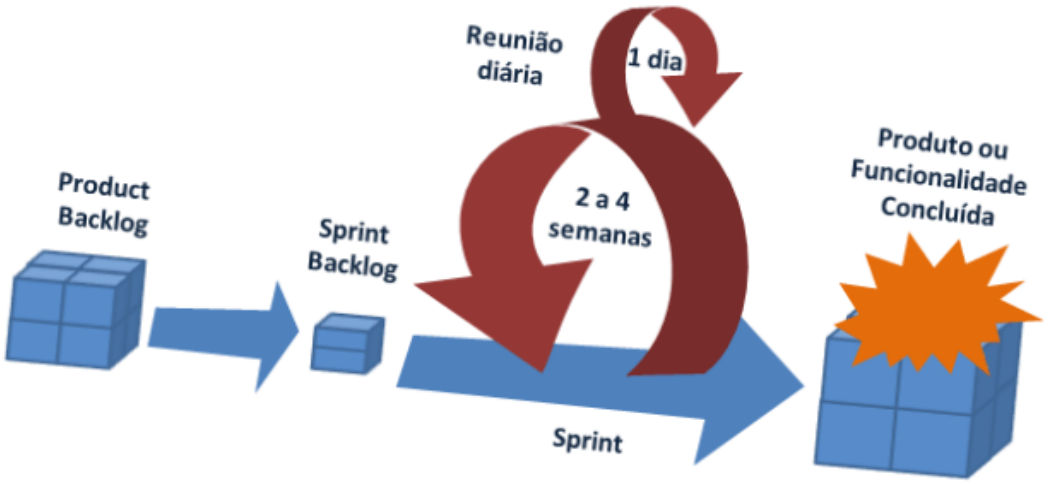

Figura 2 – Ciclo de vida do *Sprint* (Fonte: http://www.mindmaster.com.br/scrum/)

A respeito do tempo dos *sprints*, Sutherland (2014) diz que eles precisam ter duração consistente, não podendo fazer um *sprint* de uma semana, e depois um de três semanas, pois assim não é possível estabelecer um ritmo de trabalho e determinar o que pode ser feito em determinado período. Para Schwaber e Sutherland (2013), o tempo dos *sprints* deve ser limitado a um mês, pois caso o tempo seja muito longo a definição do que será construído pode mudar, o risco pode crescer e a complexidade aumentar.

> As Sprints são compostas por uma reunião de planejamento, reuniões diárias, o trabalho de desenvolvimento, uma revisão da Sprint e a retrospectiva da Sprint.
>
> Durante a Sprint:
>
> - Não são feitas mudanças que possam por em perigo o objetivo da Sprint;
> - As metas de qualidade não diminuem; e;
> - O escopo pode ser clarificado e renegociado entre o Product Owner e o Time de Desenvolvimento quanto mais for aprendido. (SCHWABER; SUTHERLAND, 2013).

Somente o dono do produto tem autoridade para cancelar um *sprint* antes do término de seu tempo. Um *sprint* pode ser cancelado caso o objetivo tenha se tornado obsoleto e não faz mais sentido para a organização (SCHWABER;

SUTHERLAND, 2013). O cancelamento consome recursos, já que todo o procedimento de planejamento para iniciar um novo *sprint* é refeito.

2.2.2.2 Reunião de Planejamento do *Sprint*

Para Kniberg (2007), a reunião de planejamento é provavelmente o evento mais importante do Scrum. O propósito desta reunião é dar à equipe informação suficiente para trabalhar por algumas semanas e para isto é importante a participação de toda a equipe e do dono do produto.

Ainda segundo Kniberg (2007), o dono do produto normalmente inicia a reunião resumindo o objetivo do *sprint* e as estórias mais importantes. Com isso, a equipe inicia o processo de estimar o tempo de cada estória, iniciando pela mais importante. Caso o tempo não seja o esperado pelo dono do produto isto pode fazer com que ele mude a importância da estória.

A reunião de planejamento do *sprint* deve durar no máximo oito horas para *sprint* de um mês de duração. Em *sprints* menores, este evento é usualmente menor (SCHWABER; SUTHERLAND, 2013).

2.2.2.3 Reunião Diária

“A Reunião Diária do Scrum é um evento de 15 minutos, para que o time de desenvolvimento possa sincronizar as atividades e criar um plano para as próximas 24 horas (SCHWABER; SUTHERLAND, 2013).”

Este evento tem o objetivo de inspecionar o trabalho desde o último evento deste tipo e deve ser realizado no mesmo horário e local todos os dias para reduzir complexidade. Durante a reunião diária três perguntas precisam ser respondidas pelo time de desenvolvimento:

- O que fiz ontem que ajudou a atender a meta do *sprint*?
- O que farei hoje para ajudar a atender a meta do *sprint*?
- Eu vejo algum obstáculo que impeça o atendimento da meta do *sprint*?

O mestre do Scrum deve assegurar que a reunião ocorra e que somente os integrantes do time de desenvolvimento estejam presentes. As reuniões diárias melhoram a comunicação e identifica e remove impedimentos para o desenvolvimento (SCHWABER; SUTHERLAND, 2013).

2.2.2.4 Revisão do Sprint

Este evento ocorre ao final do *sprint* e é planejada para durar no máximo quatro horas. Nesta reunião é apresentado o que foi desenvolvido durante o *sprint* e o produto gerado é avaliado em relação aos objetivos definidos para o *sprint* (LIBARDI; BARBOSA, 2010).

Esta é uma reunião aberta e qualquer pessoa relacionada ao projeto pode participar. O time de desenvolvimento só deve demonstrar o que está totalmente concluído e pode ser entregue sem qualquer esforço adicional. Pode até não ser o produto completo, mas deve ser um atributo concluído (SUTHERLAND, 2014).

2.2.2.5 Retrospectiva do Sprint

Ocorre após a equipe mostrar o que conseguiu fazer no *sprint* anterior, para que a equipe pense no que deu certo, o que poderia ter sido melhor e o que pode melhorar para o próximo *sprint*. Este evento ocorre para avaliar o processo, e não para encontrar culpados, portanto, para ser eficaz requer maturidade emocional (SUTHERLAND, 2014).

Sutherland (2014) diz que é essencial que as pessoas na equipe assumam a responsabilidade pelo processo e seus respectivos resultados, e também que tenham coragem de levantar as questões que realmente as incomodam. O restante da equipe precisa ter maturidade para ouvir o *feedback* e procurar uma solução.

2.2.3 Artefatos do Scrum

“Os artefatos definidos para o Scrum são especificamente projetados para maximizar a transparência das informações chave de modo que todos tenham o mesmo entendimento dos artefatos (SCHWABER; SUTHERLAND, 2013)”.

2.2.3.1 *Backlog* do Produto

O dono do produto é o responsável por determinar e gerir a sequência do *backlog* do produto e o apresentá-lo em forma de lista de prioridades. Para realizar esta tarefa ele conta com a ajuda das partes interessadas e a equipe Scrum (VIEIRA, 2014). “O *backlog* do produto é uma lista ordenada de tudo que deve ser necessário no produto, e é uma origem única dos requisitos para qualquer mudança a ser feita no produto (SCHWABER; SUTHERLAND, 2013)”. O *backlog* do produto lista todas as características, funções, requisitos, melhorias e correções que devem ser feitas no produto e em futuras versões.

O dono do produto também é o responsável por garantir que os itens sejam colocados na sequência correta de prioridades, priorizando as funcionalidades mais importantes e levando em consideração também fatores como valor, custo, conhecimento e risco (VIEIRA, 2014).

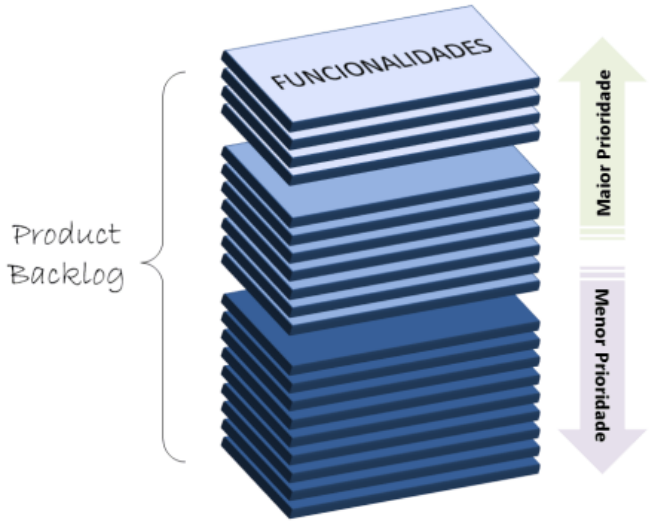

Figura 3 – *Backlog* do Produto (Fonte: http://www.mindmaster.com.br/scrum/)

O *backlog* do produto é um documento que está em constante evolução, podendo ter itens adicionados, removidos ou revistos pelo dono do produto devido a mudanças nas condições de negócios ou conforme a compreensão da equipe Scrum sobre o produto aumenta (VIEIRA, 2014).

2.2.3.2 *Backlog* do *Sprint*

Schwaber e Sutherland (2013) definem o *backlog* do *sprint* como um conjunto de itens do *backlog* do produto selecionados especificamente para o *sprint*, juntamente com o plano de entrega do incremento do produto. O *backlog* do *sprint* é a previsão do time de desenvolvimento sobre as funcionalidades que estarão pontas no próximo incremento e o trabalho necessário para entregá-la.

Ainda segundo Schwaber e Sutherland (2013), o *backlog* do *sprint* é um plano com detalhes suficientes para que sejam entendidos durante a reunião diária. Este artefato evolui conforme o trabalho é realizado e é atualizado e reestimado pelo time de desenvolvimento. Somente o time de desenvolvimento pode alterar o *backlog* do *sprint* durante o *sprint* corrente, podendo remover elementos do plano que se tornaram desnecessários ou adicionando um novo trabalho que se fez necessário para o desenvolvimento do *sprint*.

2.2.3.3 Definição de “pronto”

A definição de “pronto” pode ser diferente para cada equipe Scrum, porém todos da equipe devem entender o significado de “pronto”. Esta definição é utilizada para assegurar que o trabalho está completo no incremento do produto (SCHWABER; SUTHERLAND, 2013).

Esta mesma definição orienta o time de desenvolvimento a reconhecer quantos e quais os itens do *backlog* do produto podem ser relacionados para a reunião de planejamento do *sprint*. Como ao final de cada *sprint* um incremento do produto é entregue, a definição de “pronto” utilizada pela equipe Scrum deve ser condizente com as necessidades e intenções relacionadas ao estado final do incremento a ser entregue (SCHWABER; SUTHERLAND, 2013).

3 METODOLOGIAS ÁGEIS EM *STARTUPS*

Como visto no capítulo anterior, as metodologias ágeis foram criadas para ser uma alternativa às metodologias tradicionais de desenvolvimento de *software*, que consumiam muitos recursos, se preocupavam massivamente com documentação e demoravam muito para entregar o produto (o que só ocorria ao final de um longo ciclo de desenvolvimento), o que resultava em *softwares* que frequentemente não atendiam a demanda do cliente ou do mercado para o qual foram criados.

As metodologias ágeis foram propostas por profissionais experientes da área de gerenciamento e desenvolvimento de *software* e suas práticas adotadas por diferentes empresas, de tamanhos variados.

Porém, quando se trata de *startup,* a implantação de metodologias ágeis em sua completude não é algo simples, devido a limitações encontradas neste tipo de empresa. Os exemplos mais comuns são limitação de recursos, tamanho da equipe, incerteza a respeito do modelo de negócios, entre outros fatores, e que por isso podem não trazer o resultado esperado. Para Blank (2013), nos últimos tempos, uma nova metodologia vem se mostrando uma importante força para o sucesso de projetos de *startups*, o chamado *lean startup.* As próximas seções abordarão melhor os termos mencionados anteriormente.

3.1 Startup

Uma *startup* consiste em um grupo de pessoas buscando um modelo de negócios repetível e escalável, e que não necessariamente tem certeza a respeito desse modelo de negócios (GITAHY, 2010).

Esta definição envolve alguns conceitos importantes. O modelo de negócio é a forma com que a *startup* gera valor e transforma seu trabalho em dinheiro. Ser repetível é a capacidade da *startup* de entregar o mesmo produto novamente, sem que sejam feitas muitas customizações ou adaptações para diferentes clientes em escala ilimitada. Isso pode ser feito com produção do produto e tendo-o sempre disponível ou vendendo o mesmo produto várias vezes (um *software* por exemplo). Ser

escalável significa crescer cada vez mais sem que gere impactos no modelo de negócio. A intenção é que a receita cresça, e que os custos cresçam mais lentamente. E por sua vez, o cenário de incerteza significa que não há como garantir que a ideia ou o projeto realmente darão certos ou que sejam ao menos sustentáveis (GITAHY, 2010).

Algumas características são comuns em *startups*, por exemplo, juventude e imaturidade, recursos limitados, múltiplas influências e tecnologias e mercados dinâmicos (SUTTON 2000).

Para Sutton (2000), a característica mais básica de uma *startup* é que os fundadores são novos ou relativamente jovens e inexperientes em comparação com grandes corporações. Desta forma, a imaturidade normalmente não está apenas nos processos, mas também em sua organização. Outra característica comum são os recursos limitados, onde geralmente os primeiros recursos investidos são em atividades voltadas para o exterior da empresa, como a obtenção de um produto, a promoção deste e buscando alianças estratégicas para a *startup*.

Ainda segundo Sutton (2000), nos estágios iniciais de uma *startup*, ela pode ser sensível a influências de várias fontes, como investidores, clientes, parceiros e concorrentes. Isso pode levar a empresa a ficar ajustando constantemente o que faz e como faz. Mudanças tecnológicas e de mercado também interferem no andamento destas empresas. Novas linguagens de programação, novas arquiteturas e novas tecnologias influenciam a criação do produto, pois muitas vezes estas novas empresas são criadas para desenvolver produtos inovadores e estes podem requerer tecnologias ou ferramentas tecnológicas de ponta.

3.2 Lean Startup

O *Lean Startup* é uma metodologia criada pelo americano Eric Ries, onde “o conceito de *lean* – que pode ser traduzido como "enxuto" – é bastante conhecido na gestão e indústria tradicional, e envolve a identificação e eliminação sistemática de desperdícios (GITAHY, 2015)”.

Segundo Gitahy (2015), a estratégia desta metodologia é atuar em cada item que envolve desperdício, seja de tempo, custo ou recursos, para alcançar uma qualidade maior e chegar mais rápido ao *time-to-market.*

Gitahy (2015) também fala sobre dois conceitos importantes utilizados na metodologia *lean*. Um deles é o MVP (*Minimum Viable Product* ou Produto Mínimo Viável), que é um marco importante no ciclo de vida de um empreendimento. O segundo é o conceito de “pivotar”.

Para melhor explicar os conceitos do MVP, quebrando a sigla em letras, teríamos:

- ***Minimum*****:** o menor tamanho possível, que possa ser entregue no menor tempo possível.
- ***Viable*****:** uma proposição de valor importante o suficiente para que seu principal cliente adote esse produto, se possível gerando receita.
- ***Product*****:** funcionalidades encaixadas em uma entrega que se assemelhe a um produto coeso e útil. (GITAHY, 2015).

O conceito de MVP não é fácil de ser entendido. Integrar o entendimento entre “mínimo”, “produto” e “viável” exige muito esforço para encontrar o ponto de equilíbrio entre algo que o cliente dê valor, consumindo poucos recursos e que se assemelhe a um produto. Por isso, a tarefa de encontrar um MVP não é simples, ainda mais se tratando de um negócio inovador. Normalmente isso só ocorrerá após várias iterações, tentativas e erros, protótipos testados com usuários e muito esforço (GITAHY, 2015).

O outro conceito é o chamado “pivotar”, em tradução livre para o português. Este conceito, para uma *startup*, significa “(...) girar em outra direção e testar novas hipóteses, mas mantendo sua base para não perder a posição já conquistada (GITAHY, 2015).”. Esta tática de negócios é importante e pode definir se o projeto irá crescer ou morrer.

Para Blank (2013) o *lean startup* é uma metodologia baseada em três grandes princípios, que são: a experimentação em vez do planejamento minucioso; a opinião do cliente em vez da intuição; e o projeto iterativo em vez da concepção de um produto acabado já de início.

O primeiro destes princípios prega que ao invés de investir meses com planejamento e pesquisa, o empreendedor aceita que possui apenas uma série de hipóteses não

comprovadas. E, diferentemente de empresas tradicionais, que fazem um plano de negócios detalhado, o empreendedor faz uma espécie de mapa, chamado "*canvas* do modelo de negócios", que na prática, é um diagrama que mostra como a empresa irá criar valor para si e para seus clientes (BLANK, 2013).

O segundo princípio é testar as hipóteses com a abordagem chamada "desenvolvimento com clientes". Ele consiste na empresa ir ao mercado pedir a opinião de potenciais usuários, compradores e parceiros sobre os elementos do modelo de negócios, incluindo características do produto, preços, canais de distribuição e estratégias econômicas de aquisição de clientes. A empresa cria com rapidez um produto mínimo viável e busca imediatamente a opinião dos clientes. A partir das informações coletadas, faz a revisão de suas hipóteses e inicia o ciclo novamente, testando as versões reformuladas e fazendo ajustes necessários a cada iteração ou realizando mudanças radicais em ideias que não estão funcionando (BLANK, 2013).

Por fim, o terceiro grande princípio está relacionado ao "desenvolvimento ágil", que está ligado à prática de estar próximo ao cliente e desenvolvimento iterativo e incremental, diferentemente das metodologias tradicionais para desenvolvimento de produto. O processo de criar um produto mínimo viável utilizando o desenvolvimento ágil faz com que não haja perda de tempo ou de recursos (BLANK, 2013).

A Figura 4 exemplifica o funcionamento dos princípios adotados pelo *lean startup*, onde o desenvolvimento de um protótipo do produto é feito rapidamente e o *feedback* do cliente é importante para o início de um novo ciclo.

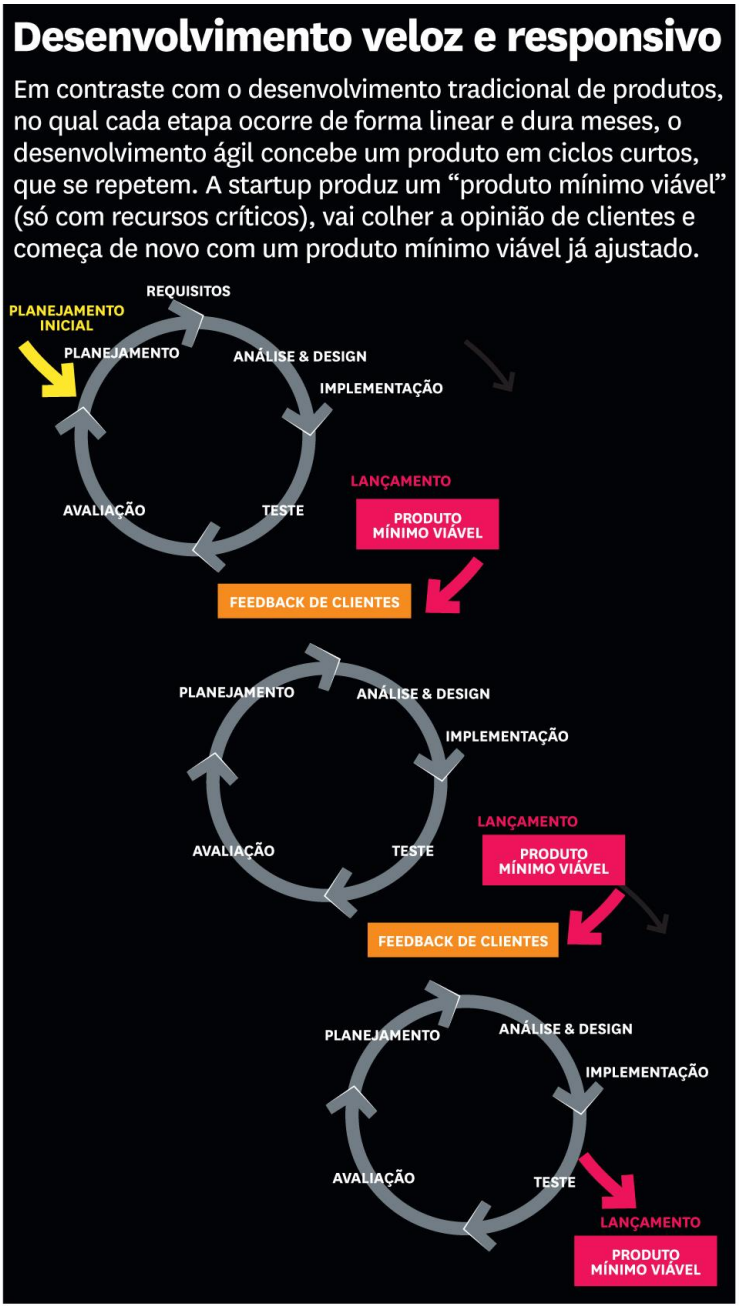

Figura 4 – Desenvolvimento veloz e responsivo (Fonte: Por que o movimento lean startup muda tudo – Harvard Business Review Brasil. Disponível em: <http://goo.gl/LozOQm>)

Outro ponto abordado por Blank (2013) é a forma de agir proposta pelo *lean*, pois antes as *startups* agiam na "surdina" para evitar que potenciais concorrentes fossem alertados sobre a oportunidade de mercado, e por isso somente durante testes finais seu produto era exposto para os clientes. A metodologia *lean startup* prega que o *feedback* do cliente é mais importante que o sigilo, e que *feedback* constante produz melhores resultados que exposição cadenciada (BLANK, 2013).

A metodologia *lean startup* pode ser considerada recente, para Blank (2013) o fenômeno da *startup* enxuta ainda não exerceu seu pleno impacto. À medida que suas práticas se alastram, vem ganhando vários adeptos e pode beneficiar desde *startups* à empresas de grande porte.

Gitahy (2015) diz que não existe uma prática única que garanta resultados se tratando de metodologias de gestão, com o *lean startup* não seria diferente. É uma ferramenta que deve ser usada combinada com outras para que se consiga obter resultados satisfatórios. A metodologia híbrida proposta será baseada nas práticas das três metodologias apresentadas anteriormente (XP, Scrum e *lean startup*), buscando extrair os pontos de cada uma que possam agregar valor e facilitar a gestão de projetos em *startups.*

4 STARTUP X: UM ESTUDO DE CASO DO USO DE METODOLOGIAS ÁGEIS

4.1 Startup X

A *startup* que motivou este estudo, que será chamada de Startup X, possui as características comuns às *startups* descritas anteriormente. São dois os sócios e fundadores da empresa, que são relativamente jovens, porém possuem experiência em suas áreas de atuação, pois já trabalharam em empresas e desempenharam atividades relacionadas a área de formação de ambos, que é a engenharia elétrica.

Por outro lado, a experiência em outras áreas ainda não atingiu um nível de maturidade alto, e eles ainda estão aprendendo a lidar com isso através das necessidades que surgem diariamente com a evolução da empresa.

O fato de ter recursos limitados também é outra característica presente na Startup X, que não conta com uma grande equipe de trabalho, tendo apenas quatro funcionários compondo o quadro. Além da limitação para força de trabalho, a falta de recursos também se estende para produção em larga escala de produtos e *hardware* que compõem o projeto, e que são importantes para o crescimento da empresa.

Influências externas e de possíveis investidores também já estiveram presentes na Startup X. Além das influências tecnológicas e de mercado, a crise econômica em que o Brasil vive atualmente, fizeram a empresa “pivotar” e alterar o foco de seu produto principal partindo para uma solução diferente, mas ainda na mesma área de atuação. A empresa ainda busca resolver o mesmo problema definido inicialmente, mas utilizando uma estratégia que pode ter baixo custo final e que permita que ela alcance uma grande fatia do mercado rapidamente.

Atualmente o foco da empresa está voltado para o desenvolvimento de um *software* e um *hardware* de baixo custo para seus usuários finais, o que implica também em baixo custo de produção que torne viável uma produção em grande escala, sem que cause impacto no modelo de negócio.

A Startup X possui experiência no desenvolvimento de *hardware*, mas o desenvolvimento de *software* é algo relativamente novo na empresa, e por isso ainda não conta com processos maduros e experiência suficiente para

gerenciamento e desenvolvimento de *software*. Devido a isso, o presente trabalho está sendo realizado para propor uma metodologia híbrida baseada em metodologias ágeis para o gerenciamento de *software*, levando em consideração as características da Startup X, como por exemplo limitação de recursos, tamanho da equipe e necessidade de criação de solução disruptiva e inovadora para o mercado.

4.2 Metodologia híbrida proposta para *Startups*

Para Sutton (2000), o desenvolvimento de *software* ainda ocorre por meio de processos, mesmo que sejam improvisados, e, por se tratar de uma atividade de engenharia de *software*, depende de um grau de rigor e ordem. A criação de processos contribui para a experiência e a repetição de processos leva ao amadurecimento da *startup*. Toda essa repetição e amadurecimento refletem no sucesso da *startup.*

Buscando melhorar os processos da Startup X, para atingir melhores resultados e facilitar o gerenciamento e desenvolvimento de *software*, foi proposta uma metodologia híbrida, baseada nas metodologias apresentadas anteriormente, e simples o suficiente para que a *startup* implante a metodologia e valide se ela gera ou não benefícios para o gerenciamento e desenvolvimento de *software*.

O motivo de não se utilizar uma ou outra metodologia e sim propor uma híbrida que contempla práticas de ambas se deve a limitações e dificuldades para executar determinadas práticas em *startups,* como programação em pares, por exemplo. Na Tabela 2 são apresentados os conceitos e práticas das metodologias e suas respectivas classificações, se estão presentes, ausentes ou opcionais na metodologia proposta, e a seguir serão apresentadas as justificativas de tal classificação.

Conceitos propostos pela metodologia	Origem	Adesão
Cliente presente	XP	–
Código coletivo	XP	✓
Padrão de código	XP	✓
Projeto simples	XP	✓
Releases curtos	XP	–
Jogo do planejamento	XP	–
Metáfora	XP	–
Programação em pares	XP	–
Desenvolvimento orientado a testes	XP	O
Refatoração	XP	✓
Integração contínua	XP	✓
Semana de 40 horas	XP	✓
Sprint	Scrum	✓
Reunião de planejamento do *Sprint*	Scrum	✓
Reunião diária	Scrum	✓
Revisão do *Sprint*	Scrum	O
Retrospectiva do *Sprint*	Scrum	O
Backlog do produto	Scrum	✓
Backlog do *sprint*	Scrum	✓
MVP	Lean Startup	✓

Legenda	
O	Opcional
✓	Presente
–	Ausente

Tabela 2 – Características da metodologia híbrida proposta

Com base nas características e do tipo de negócio principal da Startup X, as atividades propostas pela metodologia híbrida não contemplam a presença constante do cliente, pois, por vezes, este tipo de empresa ainda não possui um cliente específico, e está buscando a definição de um modelo de negócio que possa ser sustentável, repetível e escalável. A ausência desta prática se justifica pela

escolha do MVP e os princípios que giram em torno desta prática proposta pela metodologia *Lean Startup*.

A propriedade coletiva do código é um item que pode ser frequentemente visto em *startups* devido ao número reduzido de membros da equipe, e principalmente em projetos centralizados. Desta forma todos tem liberdade de fazer alterações em qualquer parte do código, mas com a responsabilidade de garantir o seu funcionamento após as modificações. Como as equipes são menores e focadas no mesmo projeto, a interação entre os membros ocorre com mais facilidade e naturalidade.

Da mesma forma que ocorre no XP, a propriedade coletiva do código e o padrão de código tem uma forte relação e, por isso, ambas foram inseridas na metodologia híbrida. Como todos têm acesso a qualquer parte do código, é importante que exista um padrão de codificação para que facilite o entendimento de todos os membros da equipe. Além disso, o código sendo padronizado contribui para a diminuição da curva de aprendizado de novos membros da equipe. Esta é uma prática que deve ser realizada voluntariamente, como apresentado na seção 2.1.2 do presente trabalho, e seguida cuidadosamente por todos para que o padrão definido seja realmente utilizado tanto na criação quanto na manutenção dos códigos.

O projeto simples é outra prática selecionada para estar presente na metodologia híbrida. Esta é uma prática chave para qualquer projeto de desenvolvimento de *software*, pois colabora para o bom funcionamento e aplicação de outras práticas. Tudo no projeto deve ser feito na forma mais simples possível, mas isto não quer dizer que deva ser feito de qualquer forma e sem um padrão de qualidade.

Para que a simplicidade do projeto permaneça presente, é preciso pensar na melhor forma de se solucionar um problema ou de adicionar uma nova funcionalidade sem comprometer a estrutura do código. Isso garante que posteriormente seja fácil adicionar uma nova funcionalidade ao projeto, além de fazer com que o *software* seja fácil de ser adaptado e tenha fácil manutenção. Com isso, qualquer alteração ou correção que seja necessária no projeto não será uma tarefa difícil e indesejável pelos desenvolvedores.

A prática de *releases* curtos presente no XP não foi escolhida para a metodologia proposta, mas o conceito geral desta prática, que é a entrega de pequenos

incrementos em curtos períodos de tempo, também está presente no *sprint,* proposto pelo Scrum, e que foi selecionado para a metodologia. Por se tratar de práticas com características similares, a escolha de apenas uma delas insere o conceito geral de ambas à metodologia proposta.

Os *sprints* têm um tempo definido de trabalho e também as tarefas que deverão ser realizadas durante este período de tempo. A equipe de desenvolvimento deve se organizar para cumprir os prazos definidos na reunião de planejamento do *sprint*, e ao final do tempo entregar uma versão utilizável do produto.

Ter versões funcionais do sistema em curtos períodos para verificação e validação das funcionalidades desenvolvidas é muito importante, principalmente para *startups,* que buscam propor soluções inovadoras e inexistentes no mercado, para que com isso possam identificar rapidamente se as funcionalidades propostas por seu sistema realmente interessam aos seus possíveis usuários.

Com entregas frequentes aliadas ao projeto simples, alterações de requisitos não causam impactos muito grandes ou prejudiciais ao projeto como um todo. Além disso, ajudam a identificar as funcionalidades importantes do sistema e a definir a prioridade de desenvolvimento das funcionalidades presentes no *backlog* do produto para as próximas iterações do projeto.

O jogo do planejamento, apesar de ser uma prática fundamental no XP e para o desenvolvimento de *software* em geral, não foi selecionado para a metodologia. Porém, outra prática com efeito similar foi incluída, justificando a ausência do jogo do planejamento. A reunião de planejamento do *sprint,* que foi selecionada ao invés do jogo do planejamento, conforme mencionado na seção 2.2.2.2, é um dos eventos mais importantes do Scrum, e é onde serão definidos os itens em que a equipe irá trabalhar nos próximos dias ou semanas. Nesta reunião, todos os envolvidos no projeto participam, a fim de definir quais itens do *backlog* do produto estarão presentes no *sprint.*

Na reunião de planejamento do *sprint*, os membros da equipe recebem as informações que precisam sobre os requisitos que deverão desenvolver no *sprint* atual. Para que a equipe possa estimar o tempo e esforço que será empregado no *sprint* esta metodologia propõe o uso do *Planning Poker,* que é uma técnica utilizada para obter a estimativa por meio de um jogo de cartas, onde todos os envolvidos no

processo de desenvolvimento participam, indicando sua visão a respeito das estórias, levando em consideração o esforço e o tempo que serão necessários para desenvolver tal item. Após a avaliação individual, cabe a cada membro explicar o motivo de sua escolha, para que cheguem a um denominador em comum (RITTER, 2014).

No *Planning Poker* cada participante tem em mãos um conjunto de 12 cartas numeradas de forma similar à sequência de *Fibonacci*, como pode ser visto na Figura 5, sendo que uma das cartas contém o símbolo de interrogação, que deve ser utilizada quando o participante não souber estimar o item em questão. Cada tarefa a ser realizada no *sprint* deve ser estimada pelos participantes com a utilização das cartas, e um valor em comum deve ser definido em consenso pela equipe. Valores acima de 20 são considerados altos, e indicam que a estória é grande e pode ser melhor detalhada dividindo em outras estórias menores, reduzindo sua complexidade (RITTER, 2014).

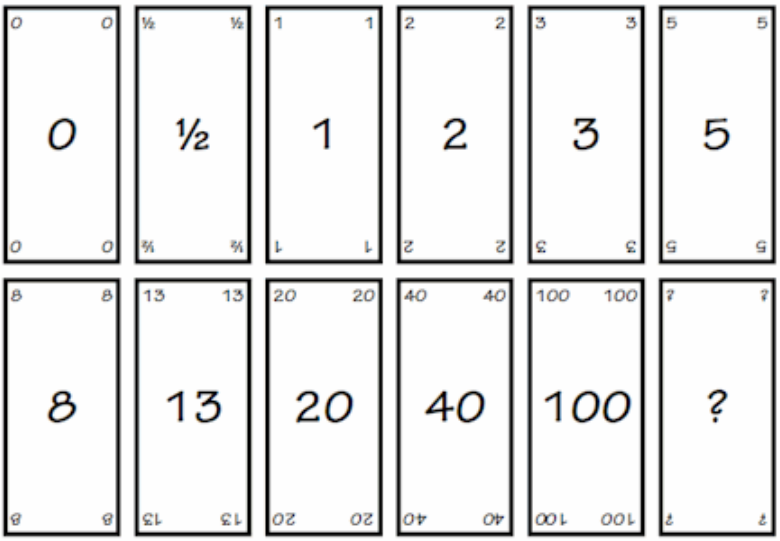

Figura 5 - Cartas de *Planning Poker* (Fonte: http://goo.gl/eUOsOJ)

A programação em pares foi uma prática do XP que também não foi selecionada para esta metodologia. Devido ao reduzido número de membros na equipe, comum em *startups*, fica inviável a adoção desta prática. A Startup X, por exemplo, conta hoje com dois sócios fundadores e quatro funcionários, sendo que destes, apenas dois estão envolvidos diretamente no desenvolvimento do sistema da startup, e os outros dois atuam em projetos secundários que podem não estar relacionados com

desenvolvimento. Portanto, não haveria rotatividade entre pares e também não conseguiriam serem aproveitados os benefícios que esta prática pode gerar.

Para esta metodologia, o desenvolvimento orientado a testes é uma prática opcional de ser seguida e implantada. A forma como esta prática funciona no XP pode não funcionar de forma satisfatória em *startups,* especialmente em ambientes em que o sistema é desenvolvido utilizando os conceitos do *lean startup* e as funcionalidades são testadas e experimentadas com os usuários antes de serem completamente desenvolvidas. Desta forma, os testes podem ser aplicados em outros momentos que não sejam no início do desenvolvimento do requisito. A automatização dos testes é algo desejável e recomendada, pois, considerando a limitação de tamanho da equipe, os testes automatizados poupariam um tempo valioso da equipe de desenvolvimento, além de garantir uma melhor qualidade do sistema produzido.

A refatoração é uma prática do desenvolvimento de *software* que deve estar presente em qualquer projeto. Seus benefícios podem não ser visíveis aos olhos dos usuários, mas para a equipe de desenvolvimento o resultado pode ser muito vantajoso e facilitar os processos de manutenção e evolução do *software*. A refatoração também contribui para manter o projeto simples, uma vez que seu objetivo é eliminar problemas no código e simplificar códigos já existentes.

A integração contínua deve fazer parte do cotidiano da equipe de desenvolvimento, principalmente por desenvolverem tarefas relativamente pequenas. Assim, ao término de cada tarefa (desenvolvimento e teste) o código deve ser integrado para evitar que possíveis erros sejam adicionados ao produto e se propaguem, sendo difícil encontrar a causa em um futuro.

Semana de 40 horas é uma prática do XP que será adotada pela metodologia, pois a utilização de horas extras não soluciona os problemas que o projeto tem para cumprir os prazos estipulados, conforme apresentado na seção 2.1.2. Para Teles (2004), a fadiga do funcionário pode produzir resultados indesejáveis, dentre eles a redução da qualidade e o uso ineficaz do tempo durante as horas regulares de trabalho. A adoção desta prática e seus conceitos contribuem para que a fadiga seja evitada, uma vez que não são recomendadas jornadas de trabalhos mais extensas e também que os membros da equipe tenham férias regularmente, desta forma, o

princípio da metodologia *lean startup* também é respeitado, pois assim evita desperdício durante o processo de desenvolvimento.

A reunião diária, que é um evento oriundo do Scrum, foi selecionada para esta metodologia. Como mencionado na seção 2.2.2.3 deste trabalho, as reuniões diárias tem o objetivo de inspecionar o trabalho realizado desde a última reunião deste tipo para sincronizar as atividades e montar o plano de ação para as próximas 24 horas. Outro ponto importante desta reunião é a possibilidade de acompanhamento pelos líderes do trabalho realizado pela equipe. E, em poucos minutos por dia é possível saber em que a equipe está trabalhando, identificar gargalos no desenvolvimento e pontos em que uma atenção especial é requerida. A reunião diária contribui para o maior controle dos requisitos e tarefas que estão sendo desenvolvidas e para certificar que a estimativa realizada anteriormente será cumprida.

A revisão do *sprint* foi definida como opcional, pois ela pode ser realizada gradativamente junto com as reuniões diárias, estendendo um pouco sua duração e apresentando os resultados gerados, uma vez que o acompanhamento do processo de desenvolvimento já é realizado constantemente. Por isso, caso seja de interesse da empresa, pode-se optar por realizar ou não a revisão do *sprint.*

O mesmo ocorre com a retrospectiva do *sprint,* que pode ser opcional e ocorrer de acordo com o entendimento da equipe sobre a necessidade de uma reunião para pontuar sobre os aspectos positivos, negativos e de melhoria que ocorreram no *sprint.*

Do Scrum também foram selecionados o *backlog* do produto e do *sprint.* O *backlog* do produto é um artefato importante para que a equipe tenha conhecimento das funcionalidades e tarefas que serão necessárias desenvolver e incorporar ao produto, tornando-o utilizável pelos usuários. Como em *startups* o produto final pode não ser conhecido no início do projeto, os itens do *backlog* do produto podem sofrer alterações à medida que o produto vai evoluindo. Tarefas e funcionalidades podem ser removidas ou adicionadas de acordo com a opinião dos usuários que testaram a aplicação com as funcionalidades já existentes.

Assim como o *backlog* do produto, o *backlog* do *sprint* serve para a equipe se nortear e saber quais tarefas irá realizar nos próximos dias ou semanas. Para gerenciar os itens do *backlog* do *sprint* e seus respectivos *status*, pode-se utilizar um

quadro *Kanban* simples. “O Kanban lhe ajuda a assimilar e controlar o progresso de suas tarefas de forma visual (BERNARDO, 2014).”

O quadro *Kanban* simples (Figura 6) é normalmente um quadro branco com *post-its* colados, que representam as tarefas que a equipe tem pra fazer. O quadro é dividido em três colunas, que representam o status de cada tarefa. A medida que a tarefa evolui seu *post-it* vai sendo trocado de coluna. Desta forma é fácil visualizar como o trabalho da equipe está fluindo (BERNARDO, 2014).

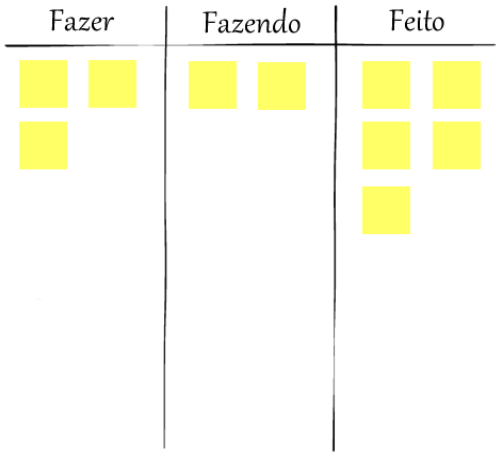

Figura 6 - Quadro *Kanban* simples (Fonte: *Kanban*: Do início ao fim! – Cultura Ágil. Disponível em: <http://goo.gl/fh0tsl>)

A técnica do MVP também foi selecionada para a metodologia, uma vez que se trata de uma técnica interessante para que *startups* possam validar se a solução que pretendem lançar para o mercado é realmente atrativa para seus usuários alvo. Uma vez que a *startup* consegue compreender o significado de MVP e gerar uma versão de seu produto nos moldes propostos por essa técnica, ela apresenta esta versão aos usuários e colhe as informações que necessita, direcionando o foco para o que deve ser desenvolvido para seu público, não gastando tempo e recurso trabalhando em funcionalidades que não serão aproveitadas ou que não interessam aos usuários.

4.3 Estudo de caso

Após a definição das práticas da metodologia híbrida, o próximo passo foi a implantação na Startup X, para que fosse possível verificar se o conjunto de práticas definido traria resultados benéficos para os processos da *startup.*

Devido a limitações de tempo e pela necessidade da Startup X em concluir determinados acordos comerciais e projetos, a metodologia proposta não pôde ser completamente implantada. Portanto a análise e validação não foram completas, ficando em aberto uma conclusão final sobre a metodologia proposta para a *startup*.

Das práticas propostas, algumas estão sendo seguidas pela empresa, como por exemplo o *sprint,* as reuniões diárias, integração contínua, MVP e *backlog do sprint.*

Como a empresa não possui métricas estabelecidas nem processos anteriores a estes, não é possível apresentar um quadro comparativo entre os resultados anteriores e os novos resultados, mas algumas melhoras foram notadas pela equipe e pelos responsáveis pela *startup*.

Com a implantação do *sprint* e das reuniões diárias, o objetivo de melhorar o controle e o gerenciamento do que a equipe tem feito foi alcançado, permitindo que os sócios da empresa pudessem dedicar um tempo maior em outras tarefas, tendo a garantia que ao final do *sprint* as atividades propostas inicialmente no *backlog* do *sprint* estarão prontas para serem testadas em uma versão utilizável do sistema.

O MVP também é outra técnica que a Startup X já aderiu, mas a versão MVP não necessariamente é gerada ao final de um *sprint*. Pode ser preciso mais de um *sprint* para que se atinja uma versão que possa ser apresentada a usuários, traga valor e os motive a utilizar a aplicação.

Após atingir o ponto ideal para uma versão MVP, esta foi gerada e apresentada a um usuário chave, para que este verificasse se a aplicação contribuiria com melhora em atividades que exerce em seu cotidiano. Com a aprovação deste usuário chave foi realizado um teste com um grupo de usuários beta, onde a opinião destes foi coletada para um refinamento dos requisitos. A partir disso foi possível definir as funcionalidades mais atrativas e os pontos em que houve rejeição por parte dos usuários. Este processo levou a modificações no *backlog* do produto e do *sprint*.

Para armazenar os itens do *backlog* do produto e também do *sprint* a equipe começou utilizando a ferramenta *Trello,* onde é possível simular um quadro *Kanban*, adicionando colunas e cartões com as atividades a serem desenvolvidas, como pode ser visto na Figura 7. Nos cartões ainda é possível definir subtarefas e adicionar comentários para a tarefa, o que facilita para a descrição do que deve ser realizado na tarefa e também para descrever o que foi feito para que a tarefa chegasse ao estado de “pronto”. Outra funcionalidade da ferramenta é a possibilidade de utilizar etiquetas com diferentes cores, que podem ser usadas para direcionar tarefas a determinado membro da equipe. No *Trello* também é possível adicionar a cada tarefa sua pontuação recebida no *planning poker,* e assim ter uma estimativa, por coluna, dos pontos recebidos pelas tarefas e qual o status da pontuação (quantos pontos por coluna).

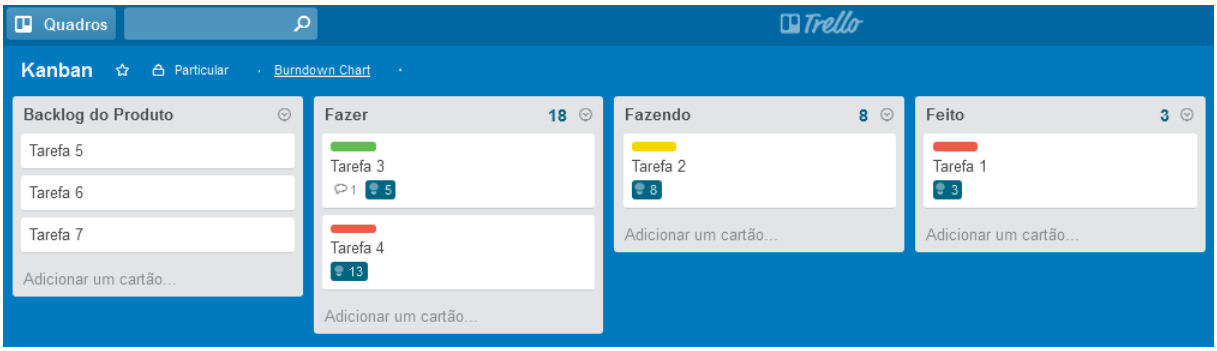

Figura 7 - *Kanban* via *Trello* (Fonte: trello.com)

Com a utilização desta ferramenta, a equipe de desenvolvimento tem uma espécie de documentação e descrição dos requisitos, o que facilita saber o que é preciso ser feito para concluir uma tarefa, evitando que ocorram esquecimentos e perda de tempo buscando informações sobre as tarefas.

Para os sócios da Startup X, esta ferramenta contribui para o acompanhamento da evolução do projeto de forma fácil e rápida. Com os indicadores dos pontos atribuídos a cada tarefa e o somatório de cada coluna é possível definir planos de ação para evitar que ocorram atrasos nos *sprints*.

Com a adoção de apenas estas práticas foi possível notar uma melhora nos processos de desenvolvimento. Com a repetição dos processos e adoção das outras práticas propostas será possível que a empresa gere indicadores importantes para

mensurar a velocidade de desenvolvimento de sua equipe e predizer o tempo que será gasto para desenvolver determinados requisitos do sistema.

5 CONSIDERAÇÕES FINAIS

A partir do estudo realizado ficou evidente a importância da adoção de metodologias para gerenciamento de projetos, em especial as metodologias ágeis, que podem ser adaptadas a diferentes cenários e garantir a entrega de *software* de qualidade dentro do que foi planejado. O fato de responderem melhor às necessidades de mudança que possam surgir durante o desenvolvimento do projeto também faz delas ferramentas indispensáveis para empresas como *startups.*

Com o conhecimento adquirido sobre *startups* e a vivência em um empreendimento deste tipo foi possível perceber a necessidade da utilização de uma metodologia para dar suporte ao desenvolvimento de *software,* e também identificar as práticas propostas pelas metodologias ágeis que melhor se encaixam ao contexto das *startups*.

A implantação, mesmo que parcial, da metodologia híbrida proposta comprovou que há melhora nos resultados quando se seguem determinados processos e práticas, e que isso pode contribuir muito para o crescimento da empresa, gerando maior organização, aproveitando melhor os recursos disponíveis e economizando tempo. Estas melhoras podem ajudar a empresa a conquistar vantagens comerciais cruciais, aproveitando melhor as oportunidades de negócio que surgirem.

Como perspectivas futuras, fica a implantação da metodologia por completo na Startup X, e também em outras *startups* de tecnologia, com o objetivo de validar, refinar e melhorar as práticas propostas por esta metodologia, para que se torne mais robusta e contribua para o crescimento de *startups*. Essas práticas também poderão ser adaptadas de acordo com os resultados obtidos, para que se possa agregar valor e melhorar a qualidade dos processos.

REFERÊNCIAS

BECK, Kent. **Extreme Programming Explained**. Set. 1999.

BECK, Kent *et al*. **Manifesto for agile software development**. 2001.

BERNARDO, Kleber. **Kanban: Do início ao fim!.** Cultura Ágil, Dez. 2014. Disponível em: <http://www.culturaagil.com.br/kanban-do-inicio-ao-fim/>. Acesso em Janeiro, 2016.

BLANK, Steve. **Por que o movimento lean startup muda tudo.** Harvard Business Review Brasil, Jul. 2013. Disponível em: <http://goo.gl/LozOQm>. Acesso em Janeiro, 2016.

DIAS, Marisa Villas Boas. **Métodos Ágeis de Desenvolvimento de Software.** Revista Engenharia de Software edição 20, 2010. Disponível em: <http://goo.gl/dnMfXn>. Acesso em: Dezembro, 2015.

GITAHY, Yuri. **Entenda o que é Lean Startup.** Revista Exame, Dez. 2015. Disponível em: <http://goo.gl/hzpUxo>. Acesso em Janeiro, 2016.

GITAHY, Yuri. **O que é uma startup?.** Revista Exame, Out. 2010. Disponível em: <http://goo.gl/zL4Mgj>. Acesso em Janeiro, 2016.

KNIBERG, Henrik. **Scrum e XP direto das Trincheiras.** InfoQ, 2007.

LIBARDI, Paula L.O.; BARBOSA, Vladimir. **Métodos Ágeis.** 2010. 35 f. Monografia (Especialização) - Curso de Pós Graduação, Faculdade de Tecnologia, Universidade Estadual de Campinas – Unicamp, Limeira, 2010. Disponível em: <http://goo.gl/UMolJm>. Acesso em Janeiro, 2016.

RITTER, Roger. **Planning Poker e Ideal Day: Técnicas de Abordagem de Estimativa Ágil.** DevMedia**,** Ago. 2014. Disponível em: <http://goo.gl/eUOsOJ>. Acesso em Janeiro, 2016.

SATO, Danilo Toshiaki. **Uso eficaz de métricas em métodos ágeis de desenvolvimento de software.** Ago. 2007. 155 f. Dissertação (Mestrado) – Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2007. Disponível em: <http://goo.gl/CorZlS>. Acesso em Janeiro, 2016.

SCHWABER, Ken; SUTHERLAND, Jeff. **Guia do Scrum.** Um guia definitivo para o Scrum: As regras do jogo. Jul. 2013. Disponível em: <http://goo.gl/KbiaTS>. Acesso em Dezembro, 2015.

SOMMERVILLE, Ian. **Engenharia de Software.** 9ª Edição. Editora Pearson, 2011.

SOUZA, Luciano Malaquias de. **Método Ágil XP (Extreme Programming).** Academos, Revista Eletrônica da FIA, Vol.III. Jul - Dez. 2007. Disponível em: <http://goo.gl/vAkYfp>. Acesso em Outubro, 2015.

SUTHERLAND, Jeff. **Scrum: A arte de fazer o dobro do trabalho na metade do tempo.** Traduzido por Natalie Gerhardt. São Paulo: Leya, 2014.

SUTTON, Stanley M.. 2000. **The Role of Process in a Software Start-up.** *IEEE Software*. 17, 4 (Jul. 2000), 33-39.

TELES, Vinícius Manhães. **Um Estudo de Caso da Adoção das Práticas e Valores do Extreme Programming.** 2005. 181 f. Dissertação (Mestrado) – Instituto de Matemática e Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro – UFRJ, Rio de Janeiro, 2005. Disponível em: <http://goo.gl/frltc1>. Acesso em: Novembro, 2015.

VIEIRA, Denisson. **Scrum: A Metodologia Ágil Explicada de forma Definitiva.** MindMaster Educação Profissional, Jun. 2014. Disponível em: <http://www.mindmaster.com.br/scrum/>. Acesso em Janeiro, 2016.