CENTRO UNVERSITÁRIO DE ANÁPOLIS – UniEVANGÉLICA BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

WENDER GALDINO DE OLIVEIRA

ESTUDO DA ADOÇÃO DE MÉTODOS ÁGEIS EM DESENVOLVIMENTO DE SOFTWARE EM UMA EMPRESA DE PEQUENO PORTE

Anápolis – GO

2016

WENDER GALDINO DE OLIVEIRA

ESTUDO DA ADOÇÃO DE MÉTODOS ÁGEIS EM DESENVOLVIMENTO DE SOFTWARE EM UMA EMPRESA DE PEQUENO PORTE

Trabalho de Conclusão de Curso II apresentado como requisito parcial à obtenção do grau de Bacharel em Engenharia de Computação do programa de graduação dos Cursos Superiores de Computação do Centro Universitário de Anápolis – UniEVANGÉLICA.

Orientador: Mr. Marcelo de Castro Cardoso

Anápolis – GO

2016

WENDER GALDINO DE OLIVEIRA

ESTUDO DA ADOÇÃO DE MÉTODOS ÁGEIS EM DESENVOLVIMENTO DE SOFTWARE EM UMA EMPRESA DE PEQUENO PORTE

Trabalho de Conclusão de Curso II apresentado como requisito parcial à obtenção do grau de Bacharel em Engenharia de computação do programa de graduação dos Cursos Superiores de Computação do Centro Universitário de Anápolis – UniEVANGÉLICA. Sob orientação do Mr. Marcelo de Castro Cardoso.

Banca Examinadora

_____

Mr. Marcelo de Castro Cardoso

Orientador

_____

Prof. Mcgill Evaristo Dias

Convidado

_____

Professora Ms. Luciana Nishi

Convidada

Anápolis, 30 de novembro de 2016.

RESUMO

No contexto de engenharia de *software*, contamos com algumas metodologias de desenvolvimento, que nos auxiliam durante todo um processo. Algumas dessas metodologias são consideradas pesadas ou orientadas a documentação, onde o maior foco está no processo de desenvolvimento, enquanto outras metodologias, que foram definidas como metodologias ágeis, têm seu foco nas pessoas. Neste trabalho, serão apresentadas algumas metodologias ágeis em desenvolvimento de *software*, mais especificamente as metodologias *Scrum* e *Xp*. Também será apresentado uma análise de relatos de experiências onde foram feitas implantações de metodologias em desenvolvimento de *software*. Nesse mesmo contexto, será apresentado um estudo de caso, de uma empresa de gerenciamento de condomínios, sediada em Anápolis-GO e que não é do ramo de desenvolvimento de *software*, mas, que desenvolveu seu próprio *software* de gestão e automação de portaria em condomínio, aplicando uma adaptação das metodologias de desenvolvimento de *software* apresentadas neste trabalho.

PALAVRAS-CHAVE: *Software*; Engenharia; Processo; Desenvolvimento; Metodologias Ágeis.

ABSTRACT

In the context of software engineering, we have some of development methodologies, which help us throughout the process. Some of these methodologies are considered burdensome or directed the documentation, where the main focus is on the development process, while other methodologies, which were defined as agile methodologies, have their focus on the people. In this study, you will be presented with some agile methodologies in software development, more specifically the methodologies Scrum and Xp. It will also be presented an analysis of reports of experiences where they were made deployments in software development methodologies. In the same context, will be presented a case study of a company managing condominiums, located at Anápolis-GO and who is not in the business of software development, but which has developed its own software for management and automation of administrative rule in condominium, applying an adaptation of software development methodologies presented in this work.

KEYWORDS: Software; Engineering; Process; Development; Agile Methodologies.

LISTA DE SIGLAS E ABREVIATURAS

CASE - Computer-Aided Software Engineering

PMBOK - Project Management Body of Knowledge

PMI - Project Management Institute

SiSLQEE - Sistema de Administração e Monitoramento de Experimentos do Laboratório de Qualidade de Energia Elétrica

SVN - Subversion

TI - Tecnologia da Informação

XP - eXtreme Programming

WIP – Work in Progress

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

SUMÁRIO

INTRODUÇÃO

As metodologias ágeis tem sido uma alternativa as metodologias tradicionais no desenvolvimento de *software*. As metodologias tradicionais também conhecidas como metodologias pesadas e orientada a planejamento e documentação, são indicadas ou aplicadas apenas quando se tem um projeto com requisitos de sistemas estáveis e quando os requisitos futuros são previsíveis. Já no caso de projetos onde se tem o risco de ocorrer muitas mudanças, e que os requisitos são totalmente passiveis a alterações, são indicadas a adoção de métodos ágeis. Metodologias ágeis preveem, alterações nos códigos fontes que não geram um alto custo, contam também com equipes pequenas, entregas curtas e desenvolvimento rápido (SOARES, 2004).

As metodologias tradicionais para desenvolvimento de *software* são muitas vezes inviáveis para desenvolvedores e algumas empresas, por que os mesmos não possuem recursos para trabalharem com metodologias pesadas de desenvolvimento de *software*. Por esse motivo, muitas das pequenas empresas optam por não adotar nenhum processo, e isso acaba gerando muitos prejuízos e insatisfações com má qualidade do produto e não cumprimento da data de entrega, acarretando em um custo muito elevado no produto final (SOARES, 2004).

A engenharia de *software* nos proporciona uma série de conhecimentos dentro de um universo relacionado ao desenvolvimento, onde, podemos ter os modelos de processo de *software* juntamente com as metodologias tradicionais e as ágeis. De acordo com Dennis e Wixom (2012) metodologias em desenvolvimento de *software* são basicamente lista de etapas e resultados, podemos dizer que essas metodologias são um conjunto de regras, e padrões para controlar a construção de algum sistema, e são essas regras, que nos garantem que o sistema a ser construído possa estar dentro dos parâmetros de qualidade respeitando os prazos e custos estabelecidos.

A empresa Gestocon administradora de condomínios situada em Anápolis-GO hoje conta com mais de 30 clientes. Atualmente a empresa conta com um *software* para gerenciar as suas atividades internas e externas, um *software* de gestão e automação de condomínios, desenvolvida pela própria empresa. Na ocasião foram relatados que estava ocorrendo muitos problemas relacionados ao *software*, como: erros, falhas, instabilidades e gerando muita insatisfação por parte dos clientes.

O *software* em questão foi desenvolvido a partir da aplicação de metodologias ágeis em desenvolvimento de *software*. A questão a ser respondida é se foram satisfatórias ou não a

metodologia aplicada para esse projeto, levando em consideração tanto o projeto quanto o cenário da empresa, se adequando a realidade?

Para que se consiga chegar a uma resposta, este trabalho tem como objetivo geral estudar e analisar metodologias ágeis em desenvolvimento de *software*, analisar relatos de experiências de implantação de metodologias em desenvolvimento e, a partir dos resultados, avaliar se a metodologia em desenvolvimento aplicada pela empresa Gestocon foi adequada, levando em consideração o projeto do *software* juntamente com o cenário da empresa. E para alcançar o objetivo proposto faz-se dos objetivos específicos:

- Analisar as metodologias Scrum e *XP* e mostrar como foi aplicado e após, verificar se seus métodos, valores e praticas foram aplicados em sua totalidade ou parcial no projeto.

- Apresentar relatos de experiências de implantação de metodologias em desenvolvimento de *software*, que juntamente com o entendimento de métodos ágeis, servirá de base para uma analise conclusiva para o estudo de caso.

- Mostrar se a metodologia adotada pela empresa Gestocon e através de dados que foram gerados durante a realização do projeto foi ideal ou se poderia haver mudanças.

A justificativa mais relevante para este trabalho, é que empresas a cada dia, estão adaptando seus processos às metodologias ágeis, motivos pelo qual metodologias como: *Scrum* e *XP*, oferecem um processo de desenvolvimento de *software* mais flexível, capaz de adaptar as mais diversas mudanças e consequentemente, uma entrega de produto mais consistente para os clientes (SILVA, 2013).

Há também, um crescente aumento de empresas privadas e até mesmo órgãos públicos do Brasil, que estão adotando as metodologias ágeis, e isso se resume em decaída de adoções de métodos tradicionais e um aumento na adoção de métodos ágeis, conforme uma pesquisa publicada pelo ex-empreendedor e atualmente principal *consultant* da ThoughtWorks Brasil, Luca Bastos (2013).

De acordo com esse cenário, profissionais de TI - Tecnologia da Informação, mais especificamente os de engenharia de *software*, precisam se adequar a essa “nova” realidade, isso, se quiserem entrar ou permanecer nesse mercado competitivo, e para isso é de grande importância o conhecimento de metodologias ágeis para desenvolvimento de *software*.

Outro motivo é pelo fato de que, profissionais de TI ou pequenas empresas, que têm um cenário, igual, ou parecido com o que será apresentado neste trabalho, possam adquirir esse conhecimento e, consequentemente, dar início a adoção de processo de metodologias

ágeis em desenvolvimento de *software*.

Assim, o trabalho está estruturado em três capítulos, sendo o capítulo 1 o referencial teórico, onde serão apresentados conceitos de definições a respeito do objeto de estudo desta pesquisa. O capítulo 2 apresenta uma análise de trabalhos relacionados com o desenvolvimento de *softwares* utilizando metodologias ágeis e seus resultados. Em seguida, no capítulo 3 discorre-se acerca do estudo de caso e o desenvolvimento do mesmo, além da metodologia utilizada para o desenvolvimento do estudo. Por fim, são apresentadas as considerações a respeito da pesquisa e seus resultados.

1. REFERENCIAL TEÓRICO

É muito comum ouvir, nos mais diversos lugares, o termo desenvolvimento ágil, talvez o motivo se deve pelo grande aumento do mercado competitivo, e da grande busca por produtos de *software* que são entregues mais rapidamente e com maior qualidade. Empresas estão cada vez mais adotando novos métodos para desenvolver seus produtos, e podemos dizer que atualmente há um grande foco nas metodologias ágeis em desenvolvimento de *software*, mais especificamente em metodologias como *Scrum* e *XP*.

Nos dias atuais as empresas operam em um ambiente global, com mudanças rápidas, e com isso precisam se adaptar à nova realidade. *Softwares* fazem parte de quase todas as operações de negócio e com isso *softwares* são desenvolvidos mais rapidamente para poder aproveitar as novas oportunidades e responder a altura da concorrência. A entrega rápida de *software* é o requisito mais crítico para o desenvolvimento do *software* e essas empresas acabam deixando de lado a qualidade e o compromisso com requisitos do *software* por uma implantação mais rápida do *software* de que necessitam. (SOMMERVILLE, 2011).

Neste capítulo serão apresentados conceitos de *software*, engenharia de *software*, alguns modelos de processo de *software* como exemplo de metodologias tradicionais, e principalmente as metodologias ágeis em desenvolvimento de *software* que foram definidas para este trabalho, juntamente com alguns relatos de experiência que, combinados seus resultados, trará uma visão para o estudo de caso apresentado neste trabalho.

1.1 *Software*

Dentre vários outros conceitos pode-se definir *software* de uma forma bem simples: *Software* é: (1) instruções (programa de computador) que quando executadas, produzem a função e o desempenho desejados; (2) estrutura de dados que possibilitam que os programas manipulem adequadamente a informação; e (3) documentos que descrevem a operação e o uso dos programas (PRESSMAN, 1995, p. 12).

Outra definição de *software*: programas de computador juntamente com toda sua documentação. E o seu produto podendo ser desenvolvido para um cliente específico ou para o mercado em geral, independente de qual área de atuação. (SOMMERVILLE, 2011, p. 4).

Então podemos dizer que *software* não é somente um programa de computador, *software* envolve muitos outros artefatos, como é possível observar na visão de Hirama (2011,

p. 5): “[...] Na Engenharia de Software consideram-se “softwares” os artefatos resultantes das atividades de levantamento de requisitos, análise de requisitos, projeto, codificação, testes e outras dentro do seu ciclo de vida. [...]”.

Pressman (1995) em suas definições de *software* classifica-os em sete (7) categorias:

- *Software* Básico: Programas escritos para dar apoio a outros programas.
- *Software* de Tempo Real: São programas que monitoram eventos do mundo real.
- *Software* Científico e de Engenharia: São algoritmos de processamento de números.
- *Software* Embutido: Programas que foram feitos para uma determinada tarefa e que são colocados nos mais diversos tipos de equipamentos.
- *Software* de Computador Pessoal: Programas que auxiliam as tarefas dos usuários.
- *Software* de Inteligência Artificial: Programas que fazem uso de algoritmos não numéricos para resolverem problemas complexos.
- *Software* Comercial: Programas desenvolvidos para as mais diversas atividades comerciais e podendo ser desenvolvidos para os mais diferentes tipos de sistemas operacionais (PRESSMAN, 1995).

> As aplicações dessa área reestruturam os dados de uma forma que facilita as operações comerciais e as tomadas de decisões administrativas. Além da aplicação de processamento de dados convencional, as aplicações de softwares comerciais também abrangem a computação interativa (por exemplo, processamento de transações em pontos-de-venda) (PRESSMAN, 1995, p. 20).

É nessa última categoria que se enquadra o sistema computacional que foi desenvolvido aplicando métodos ágeis a fim contribuir para maior eficácia na execução das atividades diárias da empresa Gestocon.

1.2 Engenharia de *software*

É uma disciplina que tem como foco atuar em todas as etapas de produção do *software*, desde seu início que são os levantamentos de requisitos ou especificação do sistema até a entrega para o usuário final, e durante a sua utilização (SOMMERVILLE, 2011).

Mas qual a importância de aplicar a engenharia de *software* no processo de *software*? “Um processo de *software* é uma sequência de atividades que leva à produção de um produto de *software*” (SOMMERVILLE, 2011, p. 5). Primeiramente um engenheiro de *software* trabalha de forma organizada, por ser mais eficiente ao desenvolver seu trabalho e

consequentemente leva a um produto de alta qualidade e, é exatamente isso que a engenharia de *software* nos propõem, mesmo por que, a maioria dos projetos envolvem cronogramas e orçamentos que, caso cumpridos, podem levar ao sucesso, caso contrário, pode até ser o fracasso, e ela é importante por dois motivos:

- A sociedade como um todo, está cada vez mais dependendo de sistemas de *software* avançados, e isso nos leva a produzir sistemas confiáveis econômicos e mais rapidamente.
- Os custos podem ser mais baratos em longo prazo quando usados técnicas de engenharia de *software*, em vez de desenvolvê-los como se fossem projetos pessoais. (SOMMERVILLE, 2011).

1.3 Camadas da engenharia de *software*

Na visão de Pressman (2011) a engenharia de *software* é retratada em camadas, da seguinte maneira:

Figura 1 - Camadas da engenharia de *software*
Fonte: Princípios da Engenharia de *Software*. Disponível em: <http://www.devmedia.com.br/principios-da-engenharia-de-software/29630>. Acesso: 16 ago. 2016.

De acordo com figura 1, são mostradas as camadas da engenharia de *software* onde, na camada, foco em qualidade, é a principal parte que sustenta todo esse contexto de engenharia de *software*. Processo permite integrar as camadas de métodos e de ferramentas para que se possa desenvolver um *software* dentro dos prazos e de maneira adequada. É no processo que permite fazer todo um planejamento e controle de *software*. Métodos são as atividades necessárias para realizar a construção de um *software*. Os métodos englobam um conjunto de atividades como: análise de requisitos, projeto, implementação, testes e manutenção. Ferramenta é o que dá a poio automatizado ou semi-automatizado para processo e métodos na engenharia de *software*. Essas ferramentas podem ser conhecidas como: CASE (Engenharia Apoiada por Computador do inglês *Computer-Aided Software Engineering*) (HIRAMA, 2011).

1.4 Ciclo de vida de *software*

No conceito de Rezende (2005) *softwares* normalmente possuem um ciclo de vida muito curto, podendo chegar no máximo 5 anos, quando não é feito nenhuma implementação. Não existe *software* pronto ou acabado, pois ao longo de toda a sua vida, podem acontecer várias mudanças, desde correções, melhorias, adequações ou implementações.

O ciclo de vida natural de um *software* abrange basicamente as seguintes fases:

- Concepção: nascimento do sistema ou *software*;
- Construção: suas análises e programação;
- Implantação: teste e disponibilização aos clientes e usuários;
- Implementação: são feitos ajustes pós-implantação;
- Maturidade e utilização plena: *software* sedimentado;
- Declínio: dificuldade de dar continuidade;
- Manutenção: realizar manutenção no *software*;
- Morte: descontinuidade do sistema ou *software*.

Ao realizar manutenções no *software*, essas atividades ficam em espiral ou *looping*, que retarda um pouco seu declínio total. Durante o ciclo de vida do *software*, as fases listadas acima são executadas de acordo com o modelo de processo de *software* escolhido, entre os clientes e a equipe de desenvolvimento, ou seja, todos os envolvidos durante todo o seu processo (KOSCIANSKI, 2007).

É nesse contexto que a engenharia de *software* oferece algumas atividades que auxiliam durante todo o ciclo de vida dos *softwares* e, essas atividades podem ser definidas como processos de *software*.

Um processo de *software* é um conjunto de atividades relacionadas que levam a produção de um produto de *software*. Essas atividades podem envolver o desenvolvimento de *software* a partir do zero em uma linguagem padrão de programação como: *Java* ou C. (SOMMERVILLE 2011, p. 18).

Dentre muitos processos de *software*, Sommerville (2011) define que todos devem conter quatro atividades fundamentais para a engenharia de *software*.

- Especificação de *software*: todas as suas funcionalidade e restrições devem ser definidas.
- Projeto e implementação de *software*: O *software* deve ser produzido para atender as suas especificações.

- Validação de *software*: o *software* deve ser validado para que possa atender as demandas dos clientes
- Evolução de *software*: o *software* deve estar em constante evolução para conseguir atender as necessidades dos clientes.

1.5 Desenvolvimento ágil

Um dos grandes desafios relacionados ao desenvolvimento de *software* é a possibilidade de entregar um *software* com qualidade, pois, conforme Pressman (2011), qualidade de *software* são as conformidades aos requisitos funcionais e aos de desempenhos que foram declarados, a padrões de desenvolvimento claramente documentados, e as características implícitas que os *softwares* desenvolvidos por profissionais esperam, ou seja, que atenda as reais necessidades dos clientes dentro dos prazos e dos custos previstos, e para isso a engenharia de *software* nos oferece métodos que auxiliam desenvolvedores de *software* durante o processo de desenvolvimento.

De acordo com Sommerville (2011) há uma insatisfação com as abordagens de metodologias pesadas na engenharia de *software* devido ao fato, de o tempo gasto na análise de como o sistema deve ser desenvolvido é muito maior do que no desenvolvimento do *software* e em testes, e isso levou um grande número de desenvolvedores de *software* a proporem, na década de 1990 novos métodos, que foram as metodologias ágeis, e essas metodologias permitiam que as equipes de desenvolvimento focassem no *software* em si e não somente na documentação.

1.5.1 Manifesto ágil

O termo “Método ágil” ficou popularmente conhecido em 2001 quando um grupo de desenvolvedores, autores e consultores da área de *software*, formaram uma aliança batizada de (“Agile Alliance” – “Aliança dos Ágeis”) e assinaram o “Manifesto para o desenvolvimento Ágil de *software*” (“*Manifesto for Agile Software Development*”) e sua filosofia por traz disso é: “Estamos descobrindo maneiras melhores de desenvolver *softwares* fazendo-os nós mesmos e ajudando outros a fazê-los. Através deste trabalho, passamos a valorizar:

- **Indivíduos e interação** entre eles mais que processos e ferramentas;
- ***Software* em funcionamento** mais que documentação abrangente;
- **Colaboração com o cliente** mais que negociação de contratos;
- **Responder a mudanças** mais que seguir um plano.

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda".

Estes valores são a base para doze princípios do desenvolvimento ágil:

1. Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de *software* de valor.
2. Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.
3. Entregar *software* funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos.
4. Pessoas relacionadas aos negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.
5. Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.
6. O Método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara.
7. *Software* funcional é a medida primária de progresso.
8. Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes.
9. Contínua atenção a excelência técnica e bom *design* aumenta a agilidade.
10. Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.
11. As melhores arquiteturas, requisitos e *designs* emergem de times auto organizáveis.
12. Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo (MANIFESTO ÁGIL, 2001).

Com a combinação desses 12 princípios juntos, formam os pilares no qual o manifesto ágil se apoia e para que se possa definir como um gerenciamento ágil, e fundamental que todos os doze princípios sejam aplicados e seguidos.

As metodologias ágeis mais conhecidas são *Scrum* e *XP* [*eXtreme Programming*:

Programação Extrema]. Estas metodologias vêm sendo utilizadas em complementação aos processos de sistemas já desenvolvidos, bem como, uma complementando a outra.

1.5.2 *Scrum*

Scrum é uma metodologia ágil para gestão e planejamento de projetos de *software* interativo e incremental que vem sendo utilizado desde a década de 1990 para desenvolvimento de produtos. *Scrum* é uma analogia à formação clássica do *Rugby*.

Historicamente, o termo *Scrum* surgiu em um artigo publicado por Hirotaka Takeuchi e Ikujiro Nonaka na *Harvard Business Review* de 1986. Nesse artigo, intitulado "The new product development game" ("O novo jogo do desenvolvimento de produtos"), Takeuchi e Nonaka descreveram uma abordagem holística, na qual equipes de projeto são compostas de pequenas equipes multifuncionais, trabalhando com sucesso rumo a um objetivo comum, que os autores compararam à formação *Scrum* do *rugby*. (PHAM e PHAM, 2011, p. 41)

Scrum é muito utilizado por desenvolvedores para seus projetos e esses projetos são divididos em ciclos chamados de *Sprints*, onde é feito o trabalho para alcançar os objetivos, que estão em uma lista de funcionalidades a serem implementadas que é conhecida como *Product Backlog*, (PHAM e PHAM, 2011). Logo abaixo podemos ver uma figura representativa de uma *Sprint* do *Scrum* que mostra as etapas que compõem a *Sprint* ou ciclo.

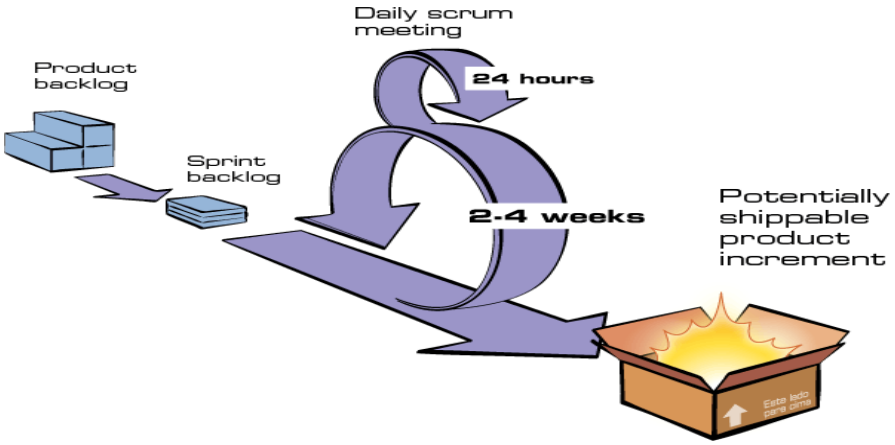

Figura 2: Ciclo de um *release* em *Scrum*
Fonte: SCRUM. Disponível em: <http://www.desenvolvimentoagil.com.br/scrum/>. Acesso: 30 ago. 2016.

1.5.2.1 Papéis do *Scrum*

Pham e Pham (2011) definem que, o time que compõem o *Scrum*, é dividido em três papeis principais: *Scrum Master*, *Product Owner* e Equipe.

- *Scrum Master*: Podemos dizer que ele é uma mistura de mediador e gerente da equipe, e tem suas responsabilidades voltadas para o incentivo da equipe, treinamento, organização e orientação dos demais papéis em relação ao processo do *Scrum*, onde seu principal papel é assegurar que toda a equipe esteja executando as práticas do *Scrum* com eficiência.
- *Product Owner*: Esse papel geralmente é desempenhado pelo cliente ou o gerente de produto de onde o *software* está sendo desenvolvido, entre suas atribuições tem a de, definir e priorizar o *Product backlog*.
- Equipe: Ela é formada por um grupo entre 5 e 9 pessoas e que trabalham de forma auto gerenciada e auto organizada, ou seja, ela é quem decide como o trabalho será executado e dividido entre os membros da equipe, ela é quem desenvolve e testa o produto.

1.5.2.2 Funcionamento do *Scrum*

Os projetos que são desenvolvidos dentro da metodologia ágil *Scrum*, utilizam-se da *Sprint* para dar andamento ao projeto, e dentro desse ciclo, tem-se uma série de etapas que devem ser passadas para cumprir o que foi definido no *Product Backlog* e atingir os resultados que foram priorizados pelo *Product Owner*. Para uma melhor visão da *Sprint*, as suas etapas serão explicadas a seguir, de acordo com a figura 5: Ciclo de um *release* em *Scrum*.

1.5.2.3 *Sprint* e *Backlog* de Produto

Conforme Pham e Pham (2011) inicia-se a *Sprint*, e tudo começa com o *Product Owner*, ele é o responsável por obter as informações dos *Stakeholders* ou dos usuários que os representam, e com as informações obtidas é gerada a lista de requisitos para criar um *Backlog* de Produto. “O *Backlog* de Produto é uma lista de requisitos priorizada, que pode

incluir de tudo: de aspectos de negócio a tecnologias, questões técnicas e correções de *bugs*" (PHAM e PHAM, 2011, p.43). Ou seja, no *Product Backlog* são definidas todas as funcionalidades que serão entregues ao cliente, vale lembrar que essa lista pode a qualquer momento ser alterada.

1.5.2.4 *Sprint Backlog*

Logo após tem-se a *Sprint Backlog*

> Consiste em uma lista de itens selecionados do Product Backog que serão realizados no próximo Sprint, esta lista é definida durante uma reunião chamada Sprint Planning Meeting na qual a equipe decide o que vai ser realizado em determinado Sprint, aliás as reuniões são um dos pontos centrais do SCRUM. (BATISTA, 2012)

Ou seja, é uma lista de tarefas que foi selecionada do *Product Backlog* e priorizada pelo *Product Owner* que o time do *Scrum* se compromete a fazer durante a *Sprint*.

1.5.2.5 *Daily Scrum*

Libordi e Barbosa (2010) explicam que no processo de andamento da *Sprint* que pode durar em torno de 2 a 4 semanas, e a cada dia da *Sprint* é realizado uma reunião com a equipe do *Scrum*, chamada *Daily Scrum Meeting*, no qual ela é realizada geralmente na parte da manhã, antes de iniciar as atividades diárias do projeto, o principal objetivo da *Daily Scrum*, é disseminar sobre o que foi feito no dia anterior, identificando possíveis impedimentos e fazer uma priorização nas atividades que serão realizadas no dia que se inicia, e para isso, cada membro da equipe tem que responder as seguintes perguntas:

1. O que você fez ontem?
2. O que você fará amanhã?
3. Há algum impedimento no seu caminho?

E com as respostas para estas questões, os membros da equipe concentram no que cada pessoa fez no dia anterior e no que irá fazer, e com isso a equipe tem uma maior compreensão que trabalho foi feito e o que precisa fazer.

1.5.2.6 *Sprint Review*

Ao final de uma *Sprint* tem-se uma *Sprint Review* que é uma reunião, e tem como objetivo: que a equipe do *Scrum* apresente os trabalhos que não foram concluídos e os que foram concluídos, e a cada *Sprint* é gerado uma demo e entregue ao cliente, que por sua vez poderá participar do processo dando sugestões das mais diversas formas (BATISTA, 2012).

1.5.2.7 *Sprint Retrospective*

Chega se ao final da *Sprint*, com mais uma reunião, e com objetivo de identificar o que funcionou e também o que pode ser melhorado, e quais os próximos passos a serem tomados, tornado essa etapa muito importante, por que é através dela, que poderá ter uma visão do que fazer e não fazer nos próximos projetos, com intuito de sempre estar buscando melhorias (BATISTA, 2012).

1.5.3 *Extreme Programming*

Assim como o *Scrum*, o *Extreme Programming* (*XP*) também é uma metodologia ágil de desenvolvimento de *software*, que é utilizada para criar sistemas de melhor qualidade e que são produzidos em menos tempo de forma mais econômica.

O primeiro projeto *Extreme Programming* foi iniciado em 6 de março de 1996. O *eXtreme Programming* é um dos vários populares processos ágeis. Foi provado ser muito bem-sucedido em muitas empresas de todos os tamanhos e indústrias mundiais de diferentes portes.

Em *XP* a codificação é a principal tarefa, com embasamento em uma revisão permanente do código, testes, entregas frequentes, participação do usuário final, reconstrução contínua, refinamento contínuo da arquitetura, integração contínua, planejamento, projeto e recomeçar o mesmo a qualquer hora. Destaque-se que esse fator de desenvolvimento ágil que a *XP* oferece, desperta maior interesse na indústria e academia, uma vez que proporciona vários benefícios no processo de produção (SILVA, 2014).

Segundo Teles (2005) o *XP* tem foco em sistemas que são voltados para as seguintes

características:

- Projetos cujos requisitos são vagos e mudam com frequência;
- Desenvolvimento de sistemas orientado a objeto;
- Equipes pequenas, preferencialmente até 12 desenvolvedores;
- Desenvolvimento incremental (ou iterativo), onde o sistema começa a ser implementado logo no início do projeto e vai ganhando novas funcionalidades ao longo do tempo.

Criado por um idealizador chamado Kent Beck em parceria com Ward Cunningham, o *Extreme Programming* possui um conjunto de práticas de desenvolvimento que estão organizados em torno de quatro valores básicos, onde se apoia para alcançar seus objetivos (TELES, 2005).

O *Extreme Programming* visa melhorar um projeto de *software* de maneiras essenciais que nos conduz ao reconhecimento de que *XP* consiste em uma metodologia extremamente embasada em comportamentos e atitudes: comunicação, simplicidade, *feedback* e coragem. Os programadores *XP* comunicam constantemente com seus clientes e colegas programadores. Eles mantêm seu *design* simples e limpo. Obtêm *feedback*, testando seu *software* agilmente. Trabalham com afinco para entregar o sistema para os clientes o mais cedo possível e implementar mudanças como sugerido. Cada pequeno sucesso aprofunda o seu respeito pelas contribuições únicas de cada um e cada membro da equipe. Procedendo desta forma os programadores *XP* são capazes de responder com coragem a evolução das necessidades e tecnologia (XP, 2013).

1.5.3.1 Valores do *Extreme Programming*

- **Comunicação**

De acordo com Teles (2005) “Projetos de *software* normalmente envolvem a presença de ao menos duas pessoas, um usuário e um desenvolvedor, o que causa a necessidade de comunicação entre elas”. Muitos projetos de *software* envolvem muito mais do que duas pessoas, ou seja, vários desenvolvedores e vários e usuários, e muitas vezes no processo de comunicação entre ambas as partes, acaba gerando ruídos indesejáveis como: equívocos, desentendimentos ou a não compreensão correta do projeto, embora todos os meios de comunicação sejam importantes, o *XP* emprega uma comunicação mais rica, ou seja, um indivíduo frente a outro se comunicando, usando apenas os meios de comunicação

corporal como: expressões faciais, gestos, postura, palavras verbalizadas e tom de voz. O *XP* prioriza comunicações mais ricas por que entende, que os benefícios são bem maiores dentro de um projeto de *software* em comparação aos meios de comunicação pouco rico como: telefones e *e-mails* (TELES, 2005).

- **Simplicidade**

Uma pesquisa mostrada no artigo de Teles (2005) do grupo Standish mostra que 45% das funcionalidades encontradas em um sistema típico jamais são usadas, e 19% raramente são utilizadas, fazendo um total de 64% de funcionalidades inviáveis em um projeto, ou seja, funcionalidades que poderiam ser evitadas no desenvolvimento de um projeto de *software*. Vale lembrar que, um projeto de *software* frequentemente investe grande parte de seus recursos em tempo, pessoas e dinheiro e isso se torna prejudicial quando seus esforços se tornam desnecessários. E isso é contra o que o *XP* prega, ou seja, os desenvolvedores têm como prioridades para o desenvolvimento, somente o que foi estabelecido pelo cliente, e quando atendida essa especificação, o trabalho é simplificado e consequentemente a redução de esforços desnecessários (TELES, 2005).

- ***FeedBack***

Um dos grandes problemas em desenvolver *softwares* está nos requisitos que são levantados para o desenvolvimento do *software*, onde, os requisitos são fundamentais para o entendimento do que será desenvolvido, assim como, para o cliente que o solicitou, muitas vezes os próprios clientes têm dificuldades de entender até mesmo o que ele está pedindo devido à falta de clareza dos requisitos (TELES, 2005).

> Por estas razões, o Extreme Programming é organizado em ciclos curtos de feedback que possibilitem aos usuários solicitar funcionalidades e aprender sobre elas através de software funcionando em prazos curtos. Esse processo envolve a priorização de poucas funcionalidades a serem implementadas de cada vez e a simplificação das mesmas na medida do possível (TELES, 2005).

Com ciclos curtos de *feedback* o *XP* garante que poucas funcionalidades serão implementadas e entregues de cada vez, caso os resultados estejam corretos, a equipe dá continuidade no projeto, se não, são feitas as devidas correções antes de dar início às novas funcionalidades, com isso a entrega de uma funcionalidade para o cliente é mais rápida, gerando um retorno de resposta mais rápido por ambas as partes (TELES, 2005).

- **Coragem**

As pessoas envolvidas em um processo de desenvolvimento de *software* muitas vezes acabam insatisfeitas em consequência de alguns fatores, dentre vários, que podem ocorrer durante e depois do desenvolvimento do *software*, dentre eles estão os clientes, que por sua vez temem pelo produto não ser o que pediram ou se pediram a coisa certa, pagar demais por muito pouco e, o não conhecimento de que está sendo feito. Já os desenvolvedores temem ser solicitados a fazerem mais do que sabem, requisitos mal levantados e deixar de lado qualidade em função de prazo (TELES, 2005).

Com intuito de acabar ou amenizar esses problemas, o *XP* preza em seus valores a coragem, com objetivo de acreditar que riscos durante o desenvolvimento do *software* irão ocorrer dos mais diversos níveis, e que, através dos incrementos, seja possível verificar juntamente com o cliente, se as funcionalidades implementadas em cada iteração estão de acordo com o requisitado (TELES, 2005).

1.5.3.2 As boas práticas de XP

Conforme Pimentel (2015) *Extreme Programming* é dinâmica e flexível, contudo é necessária muita disciplina para sua utilização em um projeto. Nesse sentido, para demonstrar isso, Pimentel (2015) apresenta um conjunto sugerido de "boas práticas" em projetos utilizando *XP*.

- **O Cliente está sempre disponível**

O cliente está constantemente disponível para colaborar em dúvidas, alterações, e prioridades em um escopo, ou seja, conferindo maior dinamismo ao projeto. Não só para ajudar o time de desenvolvimento, mas para ser uma parte dele também. Todas as fases de um projeto *XP* requerem comunicação com o cliente, de preferência face a face, no local.

- **Metáfora**

Visando facilitar a comunicação da equipe, caso seja possível, estabelecem-se a utilização de metáforas em pontos cruciais do projeto, por exemplo, a definição de um nome que seja comum à equipe e simbolize algo de fácil assimilação como, por exemplo: "Vamos chamar nosso projeto de "cartão de ponto", para um sistema que gerencie as batidas de ponto de funcionários, gerando o provisionamento financeiro e mensal para módulo de folha de

pagamento" (PIMENTEL, 2015).

- **Jogo de planejamento**

Conhecida como *User Stories* consiste em uma simples descrição de uma característica relatada a partir da perspectiva da pessoa que deseja um novo recurso, normalmente um usuário ou cliente do sistema. Geralmente este requisito é observado em um parágrafo onde descreve a necessidade do usuário de forma sucinta, utilizando uma linguagem comum ao negócio (NOGUEIRA e ZAMARO, 2014).

- **Pequenas versões**

Segundo Nogueira e Zamaro (2014) geralmente a equipe de desenvolvimento libera pequenas versões iterativas ao cliente, ainda que o lançamento seja muito pequeno ele deve proporcionar valor para os negócios do cliente, como parte do planejamento de lançamento, o cliente trabalha em conjunto com a equipe de desenvolvedores a fim de definir as *'user stories'* e sua ordem de desenvolvimento.

- **Testes de aceitação**

Segundo Pimentel (2015) estes são definidos pelo usuário na fase inicial do projeto e são os critérios de aceitação do *software* de acordo com a estratégia de entrega e representa exatamente a métrica de aderência do *software* desenvolvido/implantado ao universo do cliente.

- **Primeiro teste**

São aplicados a partir de testes unitários do código produzido, além de serem preparados utilizando os critérios de aceitação definidos previamente pelo cliente. Visa garantir, também, a redução de erros de programação e, ainda, aumentar a fidelidade do código produzido ao padrão estabelecido para o projeto. Através da prática de testes unitários, são definidos antes da codificação os testes dos métodos críticos do *software* ou métodos simples que podem apresentar alguma exceção de processamento (PIMENTEL, 2015).

- **Integração Contínua**

Os diversos módulos do *software* são integrados várias vezes por dia e todos os testes unitários são executados. O código não passa até que se obtenha êxito em 100% dos testes

unitários, facilitando o trabalho de implementação da solução.

- **Simplicidade de Projeto**

O código está, a qualquer momento, na forma mais simples e mais clara, conforme os padrões definidos pela equipe de desenvolvimento, facilitando a compreensão e possível continuidade por qualquer um de seus membros.

- **Refatoração**

A cada nova funcionalidade adicionada, trabalha-se o *design* do código até ficar na sua forma mais simples, ainda que isso implique em "mexer" em um código que esteja em funcionamento. Destaque-se que a prática de refatoração nem sempre é aceita, uma vez que envolve questões como prazo e custo. Além disso, essa prática em si pode ser minimizada caso o projeto esteja usando 100% de orientação a objeto, onde se podem criar códigos mais genéricos e reutilizáveis possíveis, diminuindo o trabalho em caso de uma possível refatoração (PIMENTEL, 2015).

- **Programação em dupla**

Conforme Pimentel (2015) todo código de produção é desenvolvido por duas pessoas trabalhando com o mesmo teclado, o mesmo *mouse* e o mesmo monitor, somando forças para a implementação do código. À primeira vista pode parecer loucura, pois se imagina estar gastando dois recursos humanos ao mesmo tempo para fazer a mesma tarefa e sem possibilidade de avanço substancial no projeto. Mas na verdade, essa prática tem pontos positivos como:

- Compartilhamento de conhecimento sobre as regras de negócio do projeto por todos da equipe de desenvolvimento;
- Fortalece a prática de Propriedade Coletiva do Código;
- Nivelação de conhecimento técnico dos programadores;
- Elevação dos níveis de atenção ao código produzido, pois um “supervisiona” e orienta o trabalho do outro. Dessa forma, minimiza-se a possibilidade de erros no código, erros de lógica e produção de um código fora dos padrões estabelecidos pela equipe.

- **Rodízio de pessoas**

As duplas de programação são revezadas periodicamente, com o objetivo de

uniformizar os códigos produzidos, deixar todos os módulos do sistema com mesmo padrão de código/pensamento e compartilhar o código com todos da equipe (PIMENTEL, 2015).

- **Propriedade coletiva**

Uma vez aplicados a Programação em Dupla e o Rodízio de Pessoas, a equipe como um todo é responsável por cada arquivo de código. Não é necessário pedir autorização para alterar qualquer arquivo, mantendo claro, um padrão prático de comunicação da equipe.

- **Padronização do código**

Todo código é desenvolvido seguindo um padrão, qualquer que seja, entretanto, toda equipe deve seguir o mesmo padrão. Desse modo, todos da equipe terão a mesma visão do código.

- **40 horas semanais**

Pimentel (2015) relata que trabalhar por longos períodos pode ser bastante contraproducente. Sendo assim, sempre que possível, deve-se evitar a sobrecarga de trabalho de toda a equipe, criando condições favoráveis ao uso da carga normal de trabalho. É necessário deixar a equipe livre para relaxar para equilibrar o trabalho mental e físico.

1.6 *Kanban*

Kanban é uma ferramenta criado por Taiichi Ohno responsável pela criação do sistema Toyota de produção, a intenção era encontrar uma forma de melhoria que mantivesse um alto nível de produção (MARTINS, 2015).

De acordo com Boeg (2012) o método *kanban* é uma forma de implantar mudanças, e não é usado para prescreve papéis ou práticas, pelo contrário, ele oferece uma série de princípios para otimização de fluxo e a geração de valor dos sistemas de entrega de *software*.

Para as mais diversas abordagens sobre o *Kanban*, a maioria dos especialistas concorda que ele é um método de gestão de mudanças, e seus princípios básicos são:

- Visualizar o trabalho em andamento;
- Visualizar cada passo em sua cadeia de valor, do conceito geral até software que se possa lançar;

- Limitar o Trabalho em Progresso (WIP – *Work in Progress*), restringindo o total de trabalho permitido para cada estágio;
- Tornar explícitas as políticas sendo seguidas;
- Medir e gerenciar o fluxo, para poder tomar decisões bem embasadas, além de visualizar as consequências dessas decisões;
- Identificar oportunidades de melhorias, criando uma cultura *Kaizen*, na qual a melhoria contínua é responsabilidade de todos.

O *kanban* pode ser utilizado em diversas áreas; gestão de TI, novos negócios, design, finanças, marketing, operações e desenvolvimento de software, entre outras. O uso da ferramenta quando aplicado de forma adequada, pode propiciar agilidade em responder às mudanças que surgem ao longo da execução de tarefas e projetos, e organização, formando um time que trabalha de forma mais eficiente sem gerar muito ruído nas comunicações (MARTINS, 2015)

A utilização do *Kanban* pode ser implementada com o uso de cartões em um quadro ou mural, possibilitando que pessoas possam verificar rapidamente a situação do trabalho

Dos mais diversos exemplos do quadro *Kanban*, o que a empresa Gestocon adotou pode ser visualizado em uma representação na imagem abaixo

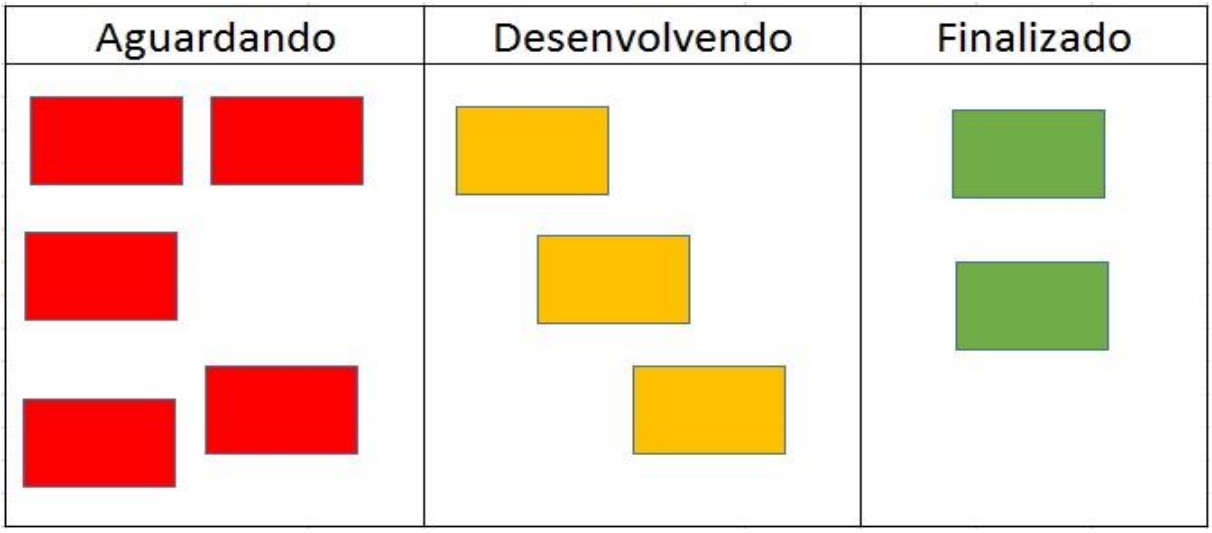

Tabela 1: representação do quadro *Kanban*
Fonte: O autor (2016)

Kanban quando aplicado corretamente proporciona uma visão melhor de projeto, auxiliando no gerenciamento das atividades, gerando inúmeros benefícios para a equipe,

projeto e empresa.

1.7 Teste de *software*

Antes de discorrer a respeito do assunto, é importante esclarecer que os testes de *software* são divididos em diversos tipos, de acordo com seu objetivo particular.

A fim de garantir a qualidade de um programa, as desenvolvedoras efetuam testes nele. Tais testes são de suma importância para que possíveis falhas sejam detectadas antes que o *software* seja colocado no mercado. Deste modo, realiza-se uma bateria de testes distintos, que envolvem desde análise da estrutura interna do *software* até a avaliação da *interface* (TESTE DE SOFTWARE, 2016).

Para que os testes ocorram mais rapidamente e com maior abrangência, existem ferramentas que automatizam alguns deles ou auxiliam na execução de outros. São as ferramentas de teste de *software*.

Conforme a revista eletrônica Teste de *software* (2016) testar *software* não é somente executá-lo com a intenção de encontrar erros, existem várias atividades como: planejamento e controle, escolha das condições de teste, modelagem dos casos de teste, checagem dos resultados, avaliação de conclusão dos testes, geração de relatórios como também a revisão dos documentos. Algumas pessoas confundem a atividade de depuração (*debbuging*) com a atividade de teste, mas elas diferem. Uma vez que a atividade de teste pode demonstrar falhas que são causadas por defeitos enquanto a depuração é uma atividade de desenvolvimento que repara o código e checa se os defeitos foram corrigidos corretamente para então ser feito um teste de confirmação por um testador com a intenção de certificar se o mesmo foi eliminado (BSTQB, 2007).

A seguir, apresentam-se alguns dos principais testes de *software* e suas finalidades.

Tipo de Teste	Descrição
Teste de Unidade	Teste em um nível de componente ou classe. É o teste cujo objetivo é um “pedaço do código”.
Teste de Integração	Garante que um ou mais componentes combinados (ou unidades) funcionam. Podemos dizer que um teste de integração é composto por diversos testes de unidade.
Teste Operacional	Garante que a aplicação pode rodar muito tempo sem falhar.
Teste Positivo-negativo	Garante que a aplicação vai funcionar no “caminho feliz” de sua execução e vai funcionar no seu fluxo de exceção.

Teste de regressão	Toda vez que algo for mudado, deve ser testada toda a aplicação novamente.
Teste de caixa-preta	Testar todas as entradas e saídas desejadas. Não se está preocupado com o código, cada saída indesejada é vista como um erro.
Teste caixa-branca	O objetivo é testar o código. Às vezes, existem partes do código que nunca foram testadas.
Teste Funcional	Testar as funcionalidades, requerimentos, regras de negócio presentes na documentação. Validar as funcionalidades descritas na documentação (pode acontecer de a documentação estar inválida)
Teste de Interface	Verifica se a navegabilidade e os objetivos da tela funcionam como especificados e se atendem da melhor forma ao usuário.
Teste de Performance	Verifica se o tempo de resposta é o desejado para o momento de utilização da aplicação.
Teste de carga	Verifica o funcionamento da aplicação com a utilização de uma quantidade grande de usuários simultâneos.
Teste de aceitação do usuário	Testa se a solução será bem vista pelo usuário. Ex: caso exista um botão pequeno demais para executar uma função, isso deve ser criticado em fase de testes.
Teste de Volume	Testar a quantidade de dados envolvidos (pode ser pouca, normal, grande, ou além de grande).
Testes de *stress*	Testar a aplicação sem situações inesperadas. Testar caminhos, às vezes, antes não previstos no desenvolvimento/documentação.
Testes de Configuração	Testar se a aplicação funciona corretamente em diferentes ambientes de *hardware* ou de *software*.
Testes de Instalação	Testar se a instalação da aplicação foi OK.
Testes de Segurança	Testar a segurança da aplicação das mais diversas formas. Utilizar os diversos papéis, perfis, permissões, para navegar no sistema.

Tabela 1: Tipos de testes de *software*
Fonte: Teste de *software* (2016). Disponível em: <http://testesdesoftware.com/tipos-de-teste-de-software/>. Acesso: 05 nov. 2016.

1.8 Análise comparativa entre metodologias ágeis e tradicionais

Segundo Soares (2004) as metodologias consideradas tradicionais também conhecidas como “pesadas” ou orientadas a documentação. Têm como característica marcante dividir o processo de desenvolvimento em etapas e/ou fases bem definidas. Essas metodologias surgiram em um contexto de desenvolvimento de *software* muito diferente do atual, baseado apenas em um *mainframe* e terminais de funcionalidade limitada.

Muitas metodologias pesadas são desenvolvidas em cima do Modelo em Cascata. É

um modelo em que as fases definidas são seguidas de maneira linear (LUDVIG; REINERT, 2007).

Já as metodologias ágeis, de acordo com Pressman (2011), constituem parte da Engenharia de *Software*, a área de conhecimento voltada para a especificação, desenvolvimento e manutenção de sistemas de *software*. A Engenharia de *Software* abarca três componentes básicos: métodos, que proporcionam os detalhes de como construir um *software*, tratando do planejamento, estimativas, análises de requisitos e arquitetura, entre outros; ferramentas, que sustentam cada um dos métodos; e procedimentos, que definem a sequência em que os métodos são aplicados, e fazem o elo entre os métodos e as ferramentas (PRESSMAN, 2011).

Nesse contexto, tem-se que a abordagem tradicional costuma ser a escolha mais indicada para cenários mais formais, nos quais a documentação detalhando as diferentes fases de um projeto representa uma exigência do negócio, mais precisamente em decorrência de obrigações junto a processos de auditoria. O gerenciamento em tais casos é realizado tendo por base as orientações propostas pelo PMBOK, um famoso guia de boas práticas na condução de projetos elaborado pelo PMI (*Project Management Institute*) (GROFFE, 2015).

Groffe (2015) argumenta que a adoção de um modelo ágil se observa a tendência a ser o melhor caminho para equipes menores, onde a flexibilidade diante de constantes mudanças é de suma importância na execução das atividades rotineiras. Isto não significa que uma metodologia como *Scrum* não possa ser empregada em grandes projetos. Observa-se, ainda, uma tendência em se combinar elementos das duas abordagens, tradicional e ágil, selecionando os pontos que melhor se adaptam à situação.

Os diferentes pontos sob análise podem ser observados na tabela abaixo:

ABORDAGEM TRADICIONAL	**ABORDAGEM ÁGIL**
Planejamento rígido	Maior liberdade no planejamento das ações
Resistência a mudanças	Flexibilidade e uma postura positiva diante da necessidade de mudanças (mesmo em fases finais do projeto)
Decisões tomadas em uma abordagem top-*down*	Liberdade para que o time tome decisões em conjunto
Forte centralização em torno da figura do ‘gerente de projetos’	Reponsabilidade compartilhada entre os membros da equipe, espírito de colaboração
Uma liderança que monopoliza toda a comunicação, já que está preocupada com o comando e o controle minucioso das ações	Comunicação fluída e livre entre os membros do time

Líderes indicando ‘O que fazer’, ‘Como’, ao invés de dizer o ‘Porquê’	Equipes auto organizáveis; a divisão do trabalho é resultado do entendimento do projeto e de um consenso entre o time
Problemas geralmente escalados até a gerência	Atuação conjunta do time para a resolução de problemas
O planejamento costuma prever um trabalho extenso, com a entrega do produto somente nos estágios finais do cronograma (o que muitas vezes leva a conflitos junto ao cliente)	Entregas de partes do projeto de forma contínua e incremental (iterações), a fim de obter um rápido feedback do cliente acerca do andamento do projeto
Uma longa fase de análise; em muitos casos, parte da equipe é deixada de lado nestes estágios iniciais (já que considera que tais membros ingressarão apenas na fase de execução)	Reuniões diárias entre o time; o intuito disso está em discutir o que será feito naquele momento, revendo o planejamento a médio e curto prazo, além de prováveis impedimentos
Um forte enfoque na geração de documentos e no controle através destes artefatos	Embora existam documentos e se estimule a criação dos mesmos, há um pragmatismo maior (sem conferir uma importância exagerada a estes artefatos)
Um maior foco em processos do que no produto esperado	Menos formalidade e maior ênfase em se chegar ao produto esperado
Maior envolvimento do cliente em estágios iniciais, com certo relaxamento de postura uma vez que o projeto tenha se iniciado	Participação ativa do cliente, inclusive enquanto o projeto está sendo implementado
Foco na ‘antecipação’ (algo difícil em um ambiente sempre sujeito a mudanças repentinas)	Ênfase na ‘adaptação’ (requer ‘jogo de cintura’)

Tabela 2: Comparativo entre metodologias tradicionais e ágeis
Fonte: Groffe (2015).

De modo geral, é possível inferir que, para o desenvolvimento de *softwares*, as metodologias baseadas em metodologias ágeis têm potencial de apresentar resultados melhores que os resultados apresentados pelas metodologias tradicionais. Para as pessoas que desenvolvem o sistema, as metodologias orientadas a pessoas apresentam melhores condições de trabalho e melhores resultados que as metodologias orientadas a processos.

No próximo capitulo serão abordados projetos que utilizaram metodologias ágeis em seu desenvolvimento.

2. TRABALHOS RELACIONADOS

No processo de desenvolvimento desse trabalho foram investigadas algumas pesquisas sobre a experiência de implantação de metodologias ágeis em empresas de pequeno, médio e/ou grande porte. É importante enfatizar que esta investigação se faz necessária, pois proverá subsídios pontuais para solução do problema a ser abordado neste trabalho.

Nos últimos tempos um dos assuntos mais falados no desenvolvimento de *software* é a respeito das metodologias ágeis, em todas as suas variações: *Lean*, *XP*, *Scrum*. Um dos principais pontos de discussão é em relação a como introduzir métodos ágeis nas empresas, o que muitos comentam como sendo algo extremamente difícil. Sempre existe aquela história de que "essa metodologia é legal, mas para o nosso modelo de negócio não funciona". Enquanto muitas vezes isso pode ser verdade, às vezes não (SOUZA, 2010).

- **Laboratório de Qualidade de Energia Elétrica da FEIS/UNESP**

Silva (2014) apresenta a pesquisa realizada no Laboratório de Qualidade de Energia Elétrica da FEIS/UNESP, onde foi desenvolvido o SiSLQEE - Sistema de Administração e Monitoramento de Experimentos do Laboratório de Qualidade de Energia Elétrica. Para o desenvolvimento desse sistema foi realizado o estudo da Engenharia de *Software* e a escolha do uso das Metodologias Ágeis. Optou-se pela escolha da junção das ferramentas ágeis *Scrum* e *XP*, onde se observou a viabilidade por ser apenas um analista a desenvolver, podendo focar apenas nas práticas que poderiam agilizar o processo de desenvolvimento do SiSLQEE, as práticas escolhidas foram bem destacadas e trabalhadas no decorrer do processo de desenvolvimento do SiSLQEE.

No processo não foram encontradas dificuldades em relação à equipe, onde todos compreenderam sua função, fazendo com que houvesse boa dinâmica de trabalho e o desenrolar do processo se deu dentro do previsto.

Entretanto, conforme Silva (2014, p. 81) foram encontradas dificuldades no processo de desenvolvimento técnico, tais como a:

- Definição de ferramentas de desenvolvimento;
- Aprendizagem da linguagem Java;
- Equipamentos com limitações para implantação;
- Bloqueio do *Windows Server*, necessidade de uso de outras ferramentas disponíveis na comunidade;

- Alteração do código fonte do *ThinVNC*;
- Execução dos testes: Colocar o sistema em funcionamento aos alunos.

Contudo, Silva (2014) relata que apesar das dificuldades técnicas foi possível aplicar a solução desenvolvida e deixar funcionado nos servidores de aplicação e experimento.

Nesse sentido, conclui-se que a solução proposta tem potencial de ser melhorada e ampliada, tornando-a automática e dinâmica, sem a necessidade da intervenção do administrador, além de integrada aos sistemas de ensino a distância (SILVA, 2014).

- **Projeto de desenvolvimento de *software* em uma organização pública situada no estado de Minas Gerais conduzindo na UFLA**
  - **ProManager:** consiste em um sistema de gestão estratégica que permite elaboração e acompanhamento de projetos e planos diretores e de negócios, públicos ou privados de forma rápida e segura (COELHO, 2001, p. 31).

Coelho (2001) explica que o ProManager permite o cadastro de diferentes tipos de usuários com escala de permissões, além de cadastro de áreas, temas, metas, ações, indicadores de desempenho e planos de gestão. Permite, ainda, alterar e excluir cadastros. A funcionalidade “Painel de Bordo” permite ao usuário acompanhar o desempenho da organização.

O processo de desenvolvimento do ProManager teve como base as práticas do método ágil *Scrum* com seus papéis, formalidade e artefatos. Entretanto, as práticas *Scrum* utilizadas pela equipe são as cerimônias *Sprint Planning* e *Daily Meeting*, a fim de gerar os artefatos *Product Backlog*, *Sprint Backlog* e documento de requisitos.

A equipe do ProManager agrega algumas técnicas de desenvolvimento advindas do *XP* como *pair programming*, *stand up meetings*, integração contínua de forma bem simples armazenando builds em um servidor de desenvolvimento/produção, SVN (*Subversion*), triagem de erros, refatoração, propriedade coletiva do código, uso de ferramentas para monitoramento de atividades, testes unitários e *design* incremental. Aplicou-se, também, a prática de uso de metáforas, oriundas do método ágil *XP*.

Uma das maiores dificuldades que a equipe encontrou, ao adotar o método ágil *Scrum*, foi se adaptar às práticas deste método e ajustar o projeto e a equipe, de forma a obter um produto que atenda as expectativas dos usuários e dos envolvidos no projeto.

Considerando-se, tal dificuldade pode ser exemplificada pelo fato de não haver possibilidade de uma fase do processo não executada corretamente, conforme definição *Scrum*, é a reunião diária (*Daily Meeting*), isso ocorreu devido a incompatibilidade de horários de trabalho da equipe, visto que é composta por estudantes de graduação da Universidade Federal de Lavras com compromissos acadêmicos considerados como prioritários.

Foram apontadas como dificuldades enfrentadas no processo:

- Falhas na comunicação
- Planejamento das *Sprints*
- Falhas no processo de desenvolvimento
- Erros de especificação e atrasos nas entregas, impactando na qualidade do processo e, consequentemente, na qualidade do produto.

Diante desse resultado, e com vistas a encontrar o ponto de equilíbrio entre as práticas de garantia da qualidade e a abordagem *Scrum*, a equipe incorporou práticas de qualidade de forma a adaptar a metodologia *Scrum* às necessidades do projeto.

Nesse sentido, para superar as possíveis limitações a equipe busca incorporar práticas de desenvolvimento, ágeis ou não, para garantir melhoria no desempenho e mesmo no produto de *software*. Sendo: programação em pares, integração contínua, pequenas entregas, triagem de erros, refatoração, propriedade coletiva do código, *build* de 10 minutos, testes unitários e *design* incremental (COELHO, 2001).

- **IFBA *campus* Vitória da Conquista**
- **Projeto: Desenvolvido em *python* (v 1.5.), usando o *framework Django* (v 2.7.). Pacotes "*Pillow*" (Imagem), "Pisa" (PDF) e *Bootstrap* V. 3.3.2**

Andrade *et al*. (2016) relata a experiência da utilização da metodologia *Scrum*, na coordenação dos trabalhos no curso de Bacharelado em Sistemas de Informação do IFBA *campus* Vitória da Conquista no semestre de 2015.1.

Inicialmente, foram realizadas tarefas básicas, como leitura de conteúdo teórico e configuração do ambiente de trabalho, a apresentação dos conceitos da própria metodologia. Em seguida, desenvolveram-se tarefas mais elaboradas, com foco na adaptação dos alunos, para, posteriormente, serem apresentados ao projeto final.

Foram desenvolvidos os módulos:

- Gerenciamento de Decisões e Padrões: Possibilita o gerenciamento (excluir, pesquisar, editar e cadastrar) das decisões pertencentes a um projeto maior.
- Avaliação da Arquitetura: Auxilia o arquiteto a documentar uma avaliação da arquitetura.
- Gerenciamento da Arquitetura: Permite gerenciar as informações da arquitetura. Através deste módulo é possível descrever: tecnologias utilizadas, cenários de qualidade e requisitos funcionais e não funcionais. Os módulos de Avaliação e Gerenciamento da Arquitetura auxiliam o arquiteto no processo de criação do documento de arquitetura de domínio e cenários de qualidade com foco na avaliação da arquitetura (ANDRADE *et al*., 2016, p. 5).

Com vistas a possibilitar o desenvolvimento paralelo entre as equipes, foram criados ambientes de desenvolvimento para cada equipe utilizando a ferramenta *GitHub* e, no fim do projeto, realizou-se a sincronização destes ambientes em um projeto único.

Após a análise e documentação dos requisitos, foram iniciados os ciclos de *Sprints*. Para isso, antes de se iniciar uma Sprint, os times selecionavam os itens que deveriam ser implementados, recorrendo assim, ao *Sprint Backlog*. A cada nova implementação concretizada ou não, atualizava-se o *status* de andamento do requisito listado e, caso o mesmo não fosse implementado a tempo do fim da Sprint, o mesmo deveria ser alocado para a próxima *Sprint Backlog* (ANDRADE *et al*., 2016).

Foi definido o prazo máximo de 15 (quinze) dias para cada *Sprint.* Houve apenas um momento em que o prazo estipulado não foi atendido, em função de um problema de greve da instituição (ANDRADE *et al*., 2016).

Para o *Daily Scrum Meeting*, na forma virtual, utilizou-se recursos de videoconferência e *chat*, recorrendo às ferramentas *Skype, Hangouts e Facebook.* Foi utilizado, também, o *Trello*, gerenciador de atividades que permitiu que cada time pudesse controlar de forma eficaz as tarefas, estabelecendo uma organização, ao mesmo tempo em que era possível classificar as atividades a serem realizadas em “Tarefas não iniciadas”, “Em progresso” e “Concluídas”. Substituindo assim o *dashboard* (ANDRADE *et al*., 2016).

Após o término de cada *Sprint*, as atividades concluídas eram reorganizadas e agrupadas de acordo a numeração da *Sprint*, mantendo assim, um histórico.

Os entregáveis foram durante todo o projeto versionados no *GitHub*, sendo escolhido por ser gratuito e possuir funcionalidades que permitem a documentação, gerenciamento de código, repositório de fonte aberto e possibilitar a criação de grupos de

colaboradores. No fim, depois de todas as funcionalidades implementadas, foi realizada uma união das equipes, a fim de unificar e criar uma versão estável da ferramenta desenvolvida, na qual se encontra disponível no sítio https://github.com/cretchas/projeto_es_2015_1 (ANDRADE *et al.*, 2016).

As dificuldades encontradas no desenvolvimento do projeto consistem, principalmente, na adaptação das equipes ao novo método e à compreensão das tarefas que deveriam ser realizadas. Basicamente as dificuldades foram mais em relação à compreensão da equipe com relação à linguagem, a utilização correta de ferramentas de versionamento (*Git*) bem como suas operações de *commits*, *rollback*, *merges*, etc., a junção dos módulos desenvolvidos numa só aplicação, a utilização de uma nova linguagem de programação e de um novo *framework* de desenvolvimento. Contudo, tais dificuldades foram contornadas de forma hábil, por iniciativa da própria equipe, a partir do estudo do conteúdo técnico apresentado em livros, fóruns e pelo compartilhamento de ideias e informações entre os times.

O relato consistiu na experiência da utilização da metodologia ágil *Scrum* em um projeto de desenvolvimento de *software* distribuído, para auxiliar no processo de ensino e aprendizagem da disciplina de Engenharia de *Software*. Sendo a metodologia utilizada na coordenação dos trabalhos de desenvolvimento de um projeto durante um semestre.

Andrade *et al*., (2016) concluem que a experiência prática dos alunos possibilitou aos mesmos identificar desafios e soluções no desenvolvimento do projeto real. Ao final da disciplina, a equipe disponibilizou um módulo estável, com novas funcionalidades e com erros considerados irrelevantes para o usuário final.

Observa-se que, as metodologias ágeis têm sido uma resposta ao insucesso de projetos desenvolvidos com base nas metodologias tradicionais. Contudo, percebe-se que não se trata de um padrão ímpar de desenvolvimento, pelo contrário, os métodos ágeis podem ser adaptados ou até mesmo utilizados em conjunto, visando o melhor desempenho do projeto. Fato que se comprova nos relatos apresentados, onde cada projeto exigiu as adaptações que atendessem às suas necessidades e, assim, cada desenvolvedor lançou mão dos métodos associando-os às particularidades de cada método de modo a atender as necessidades do *software* em desenvolvimento.

Percebe-se que ainda há bastante a se fazer para resolver possíveis problemas no processo de desenvolvimento, contudo, tem-se que isto está sendo trabalhado, e cada experiência observada, mostra a superação e o aperfeiçoamento nesse campo.

3. ESTUDO DE CASO

A proposta apresentada neste trabalho é o resultado de um projeto de desenvolvimento de *software*, em um período de cinco meses, no qual foram utilizados valores, práticas e conceitos de metodologias ágeis.

O estudo de caso que será apresentado terá somente um objetivo, mostrar como as metodologias ágeis que foram propostas nesse trabalho, foram aplicadas em sua totalidade ou em partes e, após, serão mostrados os resultados obtidos com essa experiência.

Neste capítulo serão destacados os principais métodos adotados para a realização do estudo de caso da adoção de métodos ágeis para o desenvolvimento de *software*, em uma empresa do ramo de administração de condomínio, no qual não se tem como foco o desenvolvimento de software, mas, por uma questão de necessidade e redução de custos optou por iniciar um projeto, adotando alguns conceitos que as metodologias ágeis oferecem mais especificamente *Scrum* e *XP*.

3.1 Metodologia

Pode-se classificar o presente trabalho como de natureza aplicada, devido ao interesse prático na aplicação de seus resultados na resolução de problemas reais. Apresenta uma abordagem qualitativa com propósito exploratório, utilizando como procedimento técnico o estudo de caso em uma empresa de pequeno porte que atua na área de desenvolvimento de *softwares* (FACHIN, 2001).

Este trabalho baseia-se nas teorias dos métodos ágeis em desenvolvimento de *software*, considerando os processos, os métodos e melhorias da qualidade dos produtos. Jung (2004) diz que na pesquisa qualitativa admite-se a interferência dos valores do pesquisador e considera-se a existência de múltiplas realidades (GIL, 2002).

Em relação aos procedimentos técnicos, a pesquisa é rotulada como estudo de caso. Desta forma, primeiramente foi apresentada uma revisão bibliográfica sobre a adoção de métodos ágeis de desenvolvimento de *softwares*, sendo verificadas as diferentes abordagens existentes, com características clássicas e ágeis, analisando-se seus princípios, conceitos, aplicações, ferramentas e técnicas (WAZLAWICK, 2009).

A revisão bibliográfica foi baseada em livros conceituados de autores renomados como Pressman, Sommerville e Kent Beck, além de materiais disponíveis em bibliotecas

virtuais, como artigos e produções acadêmicas de outros autores.

Posteriormente, passou-se ao estudo descritivo de natureza qualitativa na modalidade de relato de experiência. Neste ponto, são expostos os relatos de experiência na utilização dos métodos ágeis de desenvolvimento de *software*, em diferentes contextos.

Por último, neste capítulo é apresentado um estudo de caso na empresa Gestocon, uma empresa de pequeno porte do ramo de administração de condomínios. O estudo foi baseado na experiência de adoção de métodos ágeis para o desenvolvimento de um software, um projeto realizado na empresa durante o primeiro semestre de 2016, no qual foi realizado uma análise da utilização total ou parcial do método ágil no desenvolvimento do *software*, juntamente com dados coletados do seus respectivos resultados.

O estudo de caso apresenta informações acerca da adoção dos métodos ágeis de desenvolvimento, por fim, a comparação dessas informações com outros relatos de experiência.

3.2 A Empresa

Gestocon é uma empresa sediada na cidade de Anápolis-GO desde 2009, e hoje é referência no Estado Goiás no quesito de administração de condomínios, que atualmente conta com mais de 30 clientes/condomínios. Não existe um setor de desenvolvimento de *software* na empresa, mas, era utilizado um *software* de automação de portaria de condomínio, porém, o mesmo não era tão completo quanto o que foi desenvolvido e que serviu de base para esse trabalho, com esse mesmo *software* implantado para os seus clientes, corresponde 3,3% do total de clientes. Com intuito de aumentar esse percentual e melhorar suas atividades internas e externas, a ideia de refazer o *software* surgiu. Aumentaram os profissionais da área de TI, mais especificamente da área de desenvolvimento de *software*.

3.3 A Equipe

Para o desenvolvimento do projeto, a empresa contou com uma equipe com dois desenvolvedores experientes no ramo de desenvolvimento de software, dentre os quais um deles é o atual dono da empresa, e que de acordo com os papeis do *Scrum*, assumiu a função de *Product Owner*.

3.3.1 O Projeto

Por se tratar de uma empresa do ramo de gerenciamento de condomínios, a proposta de desenvolver o *software* para atender as necessidades no que refere à automação de condomínios e que auxilie nas atividades de gestão da empresa, foi avaliada pelo sócio-diretor da empresa que, também, é um dos membros da equipe envolvida no projeto.

O *software* desenvolvido no projeto foi muito bem planejado, devido à grande experiência dos atuais donos com serviços oferecidos a seus clientes. O pré-requisito para dar início ao projeto foi à contratação de mais um profissional qualificado em desenvolvimento de *software* e, também, que já tenham alguma experiência, ou conhecimento com as metodologias ágeis adotadas no projeto totalizando duas pessoas na equipe.

3.3.2 Adoção dos métodos ágeis

Após uma reunião com a equipe para analisar a complexidade do *software* e decidir quais metodologias ágeis seriam adotadas, concluiu-se que seriam aplicadas algumas técnicas do *XP* e que não era possível a utilização total do *framework Scrum* por possuir uma equipe reduzida, e para a organização das tarefas seria utilizado o quadro *Kanban* que proporciona para a equipe uma melhor visualização das atividades e do andamento do projeto.

Depois de ter sido analisado o nível de complexidade do *software*, e, ter decidido quais metodologias ágeis a serem usadas ou partes dela, deu-se início ao desenvolvimento do *software* utilizando as metodologias *Scrum* e *XP*.

Em função da quantidade de membros da equipe, algumas modificações foram feitas para se adaptarem a realidade da empresa e do projeto. O tempo da *sprint* foi reduzido para um a duas semanas, e foi descartado as *Daily Scrum Meeting*, uma vez que a equipe entendeu que não seria necessário para o projeto e que a qualquer dia algum membro poderia faltar e consequentemente a reunião não poderia ser realizada. Em relação aos papeis do *Scrum* somente foram adotados Equipe e *Product Owner*, tendo em vista que a empresa não possui uma equipe no projeto grande suficiente para atribuir os papeis de acordo a metodologia *Scrum*.

Como a equipe foi considerada pequena, o foco principal foi produzir código, visando clareza e padrões definidos pela equipe, sendo esta uma das principais tarefas e práticas do *Extreme Programming* (*XP*) e, a exemplo da *Scrum*, a *XP* não foi aplicada na sua

totalidade, mas, foram adotados alguns de seus valores e práticas, que combinados foram de suma importância para o projeto.

Durante uma *sprint*, caso tenha finalizado alguma funcionalidade, a mesma é entregue ao cliente, para que seja colocado em prática o que foi desenvolvido, entre outras, práticas do *XP* foram utilizadas, Padronização de código e 40 horas semanais.

Ao final de uma *Sprint*, ou para cada funcionalidade finalizada, o desenvolvedor preenchia um formulário com as informações necessárias para as reuniões, essa tática foi adotada, pois foi entendido que seria mais produtivo nas reuniões finais das *sprints*, haja vista que as informações estariam anotadas e não seriam esquecidas durante as reuniões futuras.

A imagem abaixo representa a adaptação da metodologia *Scrum* e as boas práticas da *XP*, realizada pela equipe, com vistas a controlar os problemas que vinham surgindo no ambiente do sistema em produção.

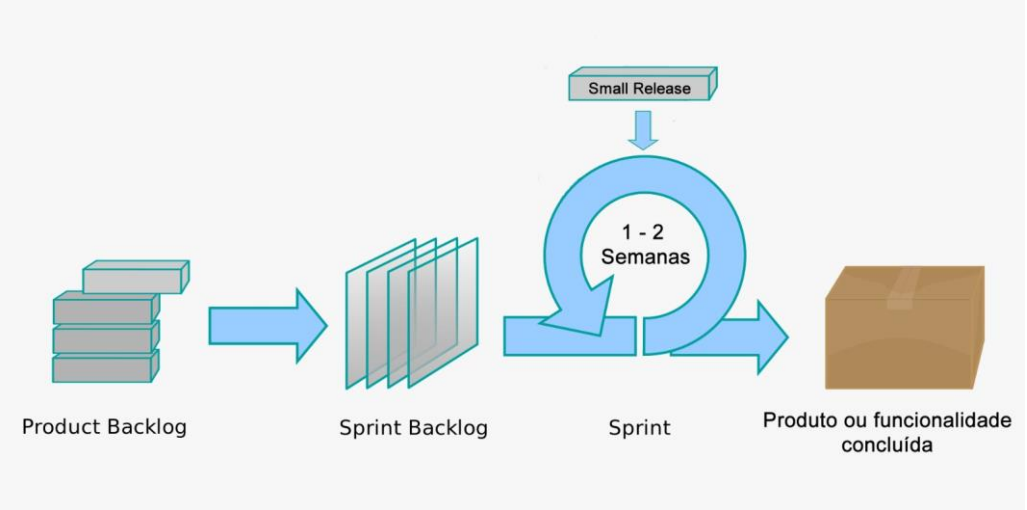

Figura 3: Adaptação da metodologia *Scrum* e as boas práticas da *XP*
Fonte: Adaptado de Scrum (https://en.wikipedia.org/wiki/Scrum_(software_development), 2016)

Foram utilizados elementos da metodologia *Scrum* bem como da *XP*. Considerando-se a dificuldade em relação à equipe, não se pode aplicar na sua totalidade o método *Scrum* e quanto ao papeis, optou-se por estabelecer a equipe e *Product owner*. Quanto aos valores e práticas *XP*, adotou-se a Padronização do código, a fim de que todos da equipe tivessem a mesma visão do código. Adotaram-se, também, a prática de 40 horas semanais, motivo no qual a empresa adota esse regime de jornada de trabalho, outros motivos estão relacionados ao desempenho da equipe, evitando estresse e que pudesse refletir no desenvolvimento do projeto. A prática de entrega de pequenas versões foi adotada no projeto, motivo no qual, as tarefas a serem desenvolvidas nas *sprints* eram finalizadas rapidamente e não se via motivo de segurar incrementos finalizados, e que refletiu a satisfação dos clientes.

Algumas práticas do *XP* não foram utilizadas, entendeu-se que, para o projeto e cenário da empresa não era viável a sua adoção.

Nesse sentido, o que ocorreu, foi que a equipe do projeto, composta por dois integrantes, em comparação ao ideal, buscou personalizar o *Scrum*, acrescentando novos recursos até que se alcançou um processo que funcionasse para a empresa, no caso a Gestocon.

3.3.3 Planejamento das *sprints*

O projeto teve seu início em 08/02/2016 e foi finalizado em 30/06/2016, logo no primeiro dia de projeto foi verificada a complexidade do *software* e a definição das metodologias a serem adotadas. Não foram feitas nenhuma estimativa para saber quanto tempo gastaria para finalizar cada tarefa da *sprint backlog*, e não foi adotada nenhuma medida para dimensionar os itens do *Product Backlog*. As *sprint* eram planejadas após a finalização da anterior exceto para as duas primeiras, em cada início de uma *sprint* uma *Sprint Planning Meeting* era realizada para definir e priorizar as tarefas, e analisar juntamente com equipe o tempo que seria gasto em cada uma, de acordo com a complexidade da funcionalidade e o nível de conhecimento da equipe de desenvolvimento.

Foram planejadas duas *sprints* para dar início ao projeto que se deu entre 11/02/2016 a 24/03/2016, percebeu-se que, o tempo estimado das *sprints* eram muito longas para a realização das tarefas, ou seja, os itens priorizados do *product backlog* terminaram antes da *sprint* ser finalizada, logo abaixo é possível ver uma representação do cronograma a partir da estimativa feita pela equipe das duas primeiras *sprints*.

Sprint	Funcionalidades		Dt. Inicial	Dt. Final	Semanas trabalhadas	Dt. Inicial alterada	Dt. Final alterada
1	Definir ferramentas para o projeto Estruturar banco de dados	Definir ferramentas para o desenvolvimento do software; Definir ferramenta para o desenvolvimento do banco de dados; Criar as tabelas do banco de dados.	11/02/2016	03/03/2016	3/2		25/02/2016
2	Manter usuário Manter morador	Definir layout; Cadastrar, Listar, Alterar, Excluir; Definir regras de acesso ao sistema para os usuários; Correções.	04/03/2016	24/03/2016	3/2	26/02/2016	11/03/2016

Tabela 3: representação de cronograma das *sprints*
Fonte: O autor (2016)

Pode-se observar através da imagem que, o planejamento não coincidiu com as estimativas feitas pela equipe, e por uma melhor organização e visão de projeto, a equipe realizou uma nova análise de como seriam realizadas as *sprints*, observou-se que muitas vezes a equipe de desenvolvimento não estava compromissada com as tarefas, ou que elas seriam consideradas fáceis de serem implementadas, e isso levou a dois pontos de vistas diferentes:

1 - Aumentar a quantidade de tarefas a serem realizadas nas *sprints*;

2 - Diminuir as tarefas da *Sprint Backloge* e reduzir o tempo de cada *sprint*;

Decidiu-se, então, que as os itens do *Sprint Backlog* seriam reduzidos para que fossem possíveis de realizar em uma *sprint* de duração de no máximo duas semanas.

O gráfico abaixo mostra o resultado final do planejamento de todas as *sprints* que foram executadas no projeto.

Gráfico 1: *Sprints*
Fonte: O autor (2016)

Em consequência de uma estimativa não muito precisa e que acarretou um mal planejamento das *sprints*, algumas tarefas não puderam ser realizadas dentro do planejado, devido ao pouco conhecimento, por parte da equipe de desenvolvimento, referente a algumas funcionalidades que dependiam de tecnologias mais avançadas como: equipamentos de reconhecimentos biométricos ou coletores de digitais, e isso fez com que algumas tarefas dependentes de outras como: liberação de acesso ao condomínio por meio de identificação biométrico entre outras, não puderam ser finalizadas dentro do planejado.

Segue abaixo representado no gráfico o percentual das tarefas que obtiveram sucesso dentro da *sprint*, e outras que por alguma falha no planejamento da *sprint* não puderam ser

100% concluídas.

Gráfico 2: Tarefas realizadas
Fonte: O autor (2016)

3.3.4 Resultados obtidos

Após o término do projeto, que durou aproximadamente cinco meses. Através de dados reais, mas, não documentados, pode-se perceber que, com a adoção das metodologias ágeis para o desenvolvimento do *software*, o ganho em produtividade foi muito grande, em comparação ao desenvolvimento realizado anteriormente na empresa, sem a adoção de nenhuma metodologia.

Como o projeto não tinha prazo estabelecido, a organização das tarefas foi bem planejada pela equipe, o que resultou numa visão mais abrangente do que seria feito, o que está sendo feito e os possíveis desenvolvimentos futuros, caso queira manter o *software*, resultando em ganhos significativos de novos clientes.

Alguns resultados negativos, consequentes da não realização dos testes de *software*, foram percebidos durante esse projeto, o que pode resultar em uma experiência muito ruim para qualquer negócio, como a perda de clientes. Devido aos poucos recursos da empresa e o efetivo reduzido da equipe, não se pode aplicar uma forma de como seriam realizados os testes após cada finalização de uma nova funcionalidade, outro ponto negativo foi a falta de documentação, mesmo os métodos ágeis não tendo seu principal foco em gerar documentos, não quer dizer que não se deve documentar nada, e foi isso, um dos pontos negativos desse projeto, por que atualmente não se tem a mesma equipe que desenvolveu o *software*, e a dificuldade de entender os requisitos, regras de negócio dentre outras funcionalidades,

dificulta a evolução do sistema.

No que se refere à evolução dos resultados, após a adoção dos métodos ágeis, tem-se que, dentre os 32 clientes que a empresa possui, houve boa aceitação do novo *software.* Devido a adoção dos métodos ágeis, houve uma boa organização e maior produtividade no desenvolvimento do projeto, as entregas de funcionalidades em sua maior parte eram feitas dentro do planejado e com melhor qualidade, gerando ganhos significativos de novos clientes O gráfico a seguir apresenta o percentual de aceitação e não aceitação do *software* durante todo o processo de desenvolvimento do mesmo.

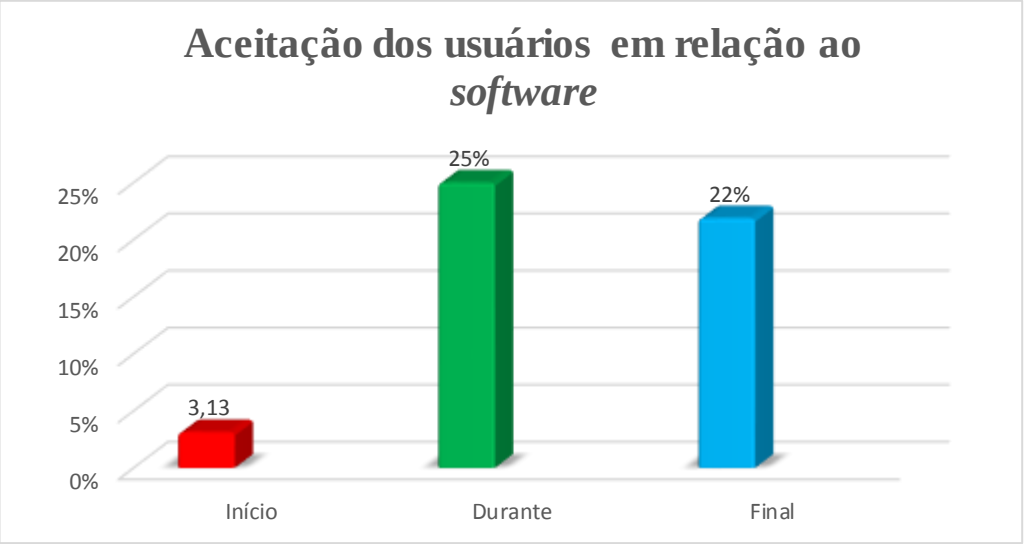

Gráfico 3: Aceitação dos usuários em relação ao *software*
Fonte: O autor (2016)

Observa-se que houve significativa aceitação por parte dos clientes da empresa. De acordo com as informações obtidas durante o projeto observou-se que os usuários encontraram dificuldade no que se refere ao funcionamento do sistema, que em algumas situações apresentou falha, em virtude da deficiência no processo de desenvolvimento inviabilizando os testes de *software*. Os dados mostrados no gráfico 3, estão ligados somente aos clientes que por algum meio ficou ciente da existência do software que a Gestocon oferecia entre seus serviços prestados e quiseram implantar o sistema em suas unidades/condomínio, lembrando que, os clientes que a empresa possui são condomínios verticais ou horizontais ou seja, são prédios e casas que os mesmos possuem uma média de cento e quinze moradores tornando-se clientes indiretamente.

Todos os dados informados na **tabela 3, gráficos 1** e **2**, foram obtidos durante todo o processo de desenvolvimento do software, que pode ser visto na tabela que segue em anexo.

CONSIDERAÇÕES FINAIS

Os métodos ágeis para desenvolvimento de *software* constituem-se numa alternativa para o desenvolvimento tradicional. Para adotar métodos ágeis em uma organização, são necessários diversos passos de planejamento, bem como uma execução cuidadosa.

Neste trabalho foram apresentados os resultados obtidos no estudo de caso de adoção de métodos ágeis em desenvolvimento de *software* em uma empresa de pequeno porte situada no município de Anápolis-GO.

Considerando-se tanto o projeto quanto o cenário da empresa buscou-se compreender se a metodologia aplicada para o projeto de *software* se adequou a realidade da mesma. Nesse sentido, o objetivo da pesquisa consistiu em estudar e analisar metodologias ágeis de desenvolvimento de *software*, analisar relatos de experiências de implantação de metodologias de desenvolvimento e, a partir dos resultados, avaliar se a metodologia de desenvolvimento aplicada pela empresa Gestocon foi adequada, levando em consideração o projeto do *software* juntamente com o cenário da empresa.

Acredita-se que os objetivos da Gestocon foram alcançados, apesar das dificuldades no desenvolver do projeto, onde se pode destacar como principal dificuldade o número reduzido de integrantes na equipe de desenvolvimento, o que influenciou de modo significativo na adoção total do método *Scrum* para a execução do projeto.

A escolha do *Kanban* foi definida a fim de obter uma organização das tarefas da equipe, uma vez que a mesma não possuía nenhum método para organizá-las. Já a aplicação do *Scrum* foi definida pelo fato de a equipe possuir de conhecimento referente a essa metodologia, contudo, não sendo possível fazer a sua aplicação total, já que a empresa não possui pessoal suficiente para a formação de uma equipe *Scrum* (um time *Scrum* deve ter, no mínimo, 5 integrantes).

Nos três relatos de experiência apresentados nesta pesquisa observou-se que foram utilizadas as metodologias *Scrum* e *XP* em função das ideias que ambas trazem. Acredita-se que uma completa a outra, ou seja, *Scrum* traz várias boas práticas, assim como a *XP* também agrega coisas interessantes. Nesse aspecto, compreende-se que, adotar as boas ideias do *Scrum* e *XP*, possivelmente o resultado será uma ótima metodologia de trabalho.

Nesse sentido, observa-se que o projeto teve um diferencial, que foi a implementação da ferramenta *Kanban*, sendo que nos demais trabalhos essa técnica não foi abordada. Além do *Kanban*, foram aplicados os métodos ágeis *Scrum* e *XP*, já que o *framework* é muito utilizado (fato verificado em todos os trabalhos citados) e que também tem grande ênfase no

meio acadêmico.

Deste modo, no desenvolvimento do *software* da empresa Gestocon foi consenso de que a utilização dos métodos supramencionados seria de grande valia para o *software* específico para esta empresa. Apesar dos problemas relacionados à falta de pessoal qualificado para compor a equipe, pode-se dizer que foram alcançados resultados positivos, uma vez que o *software* foi implantado e correspondeu à expectativa da maior parte dos clientes que aderiram o software.

Claro que, necessário é destacar a dificuldade de realizar os testes necessários, uma vez que a Engenharia de Software nos proporciona uma vasta gama de conhecimentos relacionados a testes de sistemas, no qual muitos deles podem ser vistos claramente nesse trabalho e que poderiam ser aplicados no projeto, e a não aplicação constituiu um ponto bastante negativo, considerando-se que a execução de testes favorece a identificação, análise e mitigação de fatores de riscos associados aos requisitos do produto de *software*. Assim, compreende-se a necessidade de adequação da equipe e um melhor planejamento, a fim de conceber um produto que passe por todos os requisitos necessários ao desenvolvimento de um *software*.

Diante do exposto é possível inferir que a aplicação dos métodos *Scrum* e *XP* atende de forma ampla as necessidades do desenvolvimento de processos de *software*, sem maiores custos ou burocracias como ocorre nas aplicações dos processos tradicionais. Os métodos *Scrum* e *XP* caracterizam-se por seu foco em resultados e na comunicação entre equipe e cliente, com preservação do bom relacionamento entre ambos.

Sendo assim, sugere-se a replicação desta pesquisa-ação em outras empresas da mesma natureza, que observem problemas na gestão de desenvolvimento de serviços/produtos, como mais uma proposta de trabalhos futuros. Com isso, o desdobramento dessa pesquisa em novos trabalhos irá contribuir para a consolidação e validação dos resultados aqui obtidos.

REFERÊNCIAS

ANDRADE, B. A. L.; BRITO, M. S.; SAMPAIO, A. S.; COSTA, I. R.; SANTOS, D. L.; LIMA NETO, C. **Aplicando e Adaptando a Metodologia Ágil Scrum no Processo de Ensino e Aprendizagem de Engenharia de Software Baseado no Desenvolvimento com Equipes Distribuídas.** Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBA) – BA, 2016.

BASTOS, Luca. **Cresce a adesão aos métodos ágeis.** 2013. Disponível em: <http://cio.com.br/opiniao/2013/10/28/cresce-a-adesao-aos-metodos-ageis/>. Acesso em: 07 set. 2016.

BATISTA, Pablo Pinheiro. **Entendo o SCRUM.** DEVMEDIA, 2012. Disponível em: <http://www.devmedia.com.br/space/pablo-pinheiro-batista>. Acesso: 04 ago. 2016.

BOEG, Jesper. **Kanban em 10 passos.** InfoQ Brasil, 2012.

BSTQB – **Brazilian Software Testing Qualifications Board**, 2007. Disponível em: <http://www.bstqb.org.br/>. Acesso: 30 out. 2016.

COELHO, C. S. **Relato de experiência na implantação de um método ágil em uma equipe de desenvolvimento de *software*.** [Monografia]. Universidade Federal de Lavras: UFLA, Minas Gerais, 2001. 57p.

FACHIN, O. **Fundamentos de metodologia.** 3. ed. São Paulo: Saraiva, 2001.

GIL, A. C. **Como elaborar projetos de pesquisa.** 4. ed. São Paulo: Atlas, 2002.

GROFFE, R. **Projetos de software:** uma comparação entre metodologias ágeis e tradicionais. NetCoders, 2015. Disponível em: <http://netcoders.com.br/blog/comparacao-metodologias-ageis-e-tradicionais/>. Acesso: 05 out. 2016.

HIRAMA, K. **Engenharia de software** (Qualidade e produtividade com tecnologia). 1ª. Ed. Rio de Janeiro: Elsevier, 2011.

JUNG, C. F. **Metodologia aplicada a projetos de pesquisa:** Sistemas de Informação & Ciência da Computação. Proposta de TCC e Projeto de Pesquisa; Taquara, 2009.

KOSCIANSKI, A. **Qualidade de software**. 2ª. Ed. São Paulo: Novatec, 2007.

LIBORDI, P.L.O.; BARBOSA, V. **Métodos Ágeis.** Universidade Estadual de Campinas – UNICAMP, 2010.

LUDVIG, D.; REINERT, J. D. **Estudo do uso de metodologias ágeis no desenvolvimento de uma aplicação de Governo Eletrônico.** 2007. 157fl. Trabalho de Conclusão de Curso (Bacharelado em Sistema de Informação) - Departamento de Informática e Estatística, Universidade Federal de Santa Catarina (UFSC), Florianópolis, 2007.

MANIFESTO ÁGIL. **Manifesto ágil**, 2001. Disponível em: <http://www.manifestoagil.com.br/ >. Acesso em: 25 ago. 2016.

MARTINS, Rafael. **Kanban:** 4 passos para implementar em uma equipe. DEVMEDIA**.** 2015. Disponível em: <http://www.devmedia.com.br/kanban-4-passos-para-implementar-em-uma-equipe/30218>. Acesso: 15 nov. 2016.

NOGUEIRA, M. F.; ZAMARO, M. Metodologias de desenvolvimento ágeis de software scrum e extreme programing. **Perspectivas em Ciências Tecnológicas**, v. 3, n. 3, Maio 2014, p. 9-29.

PHAM, A; PHAM, P. **Scrum em Ação:** gerenciamento e desenvolvimento ágil de projetos de software. Tradução: Edgard B. Damiani. – São Paulo: Novatec Editora: Cengage Learning, 2011.

PIMENTEL, M. **Extreme Programming** – Conceitos e Práticas. 2015. Disponível em: <http://www.devmedia.com.br/extreme-programming-conceitos-e-praticas/1498>. Acesso: 13 out. 2016.

PRESSMAN, R.S. **Engenharia de software**. 3ª. Ed. São Paulo: Pearson, 1995.

PRESSMAN, R.S. **Engenharia de software**. 7ª. Ed. Porto Alegre: AMGH, 2011.

REZENDE, D. A. **Engenharia de software e sistemas de informação**. 3ª. Ed. Rio de Janeiro: Brasport, 2005.

SCRUM. **Desenvolvimento ágil**, 2013. Disponível em: <http://www.desenvolvimentoagil.com.br/scrum/>. Acesso: 30 ago. 2016.

SILVA, A. K. V. **Acesso remoto seguro de experimentação em tempo real.** [Dissertação de Mestrado]. Faculdade de Engenharia de Ilha Solteira – UNESP – Ilha Solteira, 2014.

SILVA, Josué. **Adoção de metodologias ágeis.** DEVMEDIA, 2013. Disponível em: <http://www.devmedia.com.br/space/josue-silva>. Acesso em: 07 set. 2016.

SOARES, M. S. **Comparação entre metodologias ágeis e tradicionais para o desenvolvimento de software.** 2004. Disponível em: <http://www.dcc.ufla.br/infocomp/artigos/v3.2/art02.pdf>. Acesso: 06 out. 2016.

SOMMERVILLE, I. **Engenharia de software**. 9ª. Ed. São Paulo: Pearson, 2011.

SOUZA, L. Como é a implantação de métodos ágeis na sua empresa? **InfoQ Brasil.** Disponível em: <https://www.infoq.com/br/news/2010/02/metodologia_agil_empresa>. Acesso: 22 set. 2016.

TELES, V. M. **Extreme Programming:** aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade. 1. ed. São Paulo: Novatec, 2005.

TESTE DE SOFTWARE. **Tipos de teste de software**, (2016). Disponível em: <http://testesdesoftware.com/tipos-de-teste-de-software/>. Acesso: 02 nov. 2016.

XP - EXTREME PROGRAMMING. **Desenvolvimento ágil**, 2013. Disponível em: <http://www.desenvolvimentoagil.com.br/scrum/>. Acesso em: 13 ago. 2016.

XP. **Extreme Programming:** Uma introdução suave. 2013. Disponível em:

<http://www.extremeprogramming.org/>. Acesso: 13 out. 2016.

WAZLAWICK, R. S. **Metodologia de pesquisa para ciência da computação**. 6ª. Ed. Rio de Janeiro: Elsevier, 2009.

WIXON, H. B; DENNIS, A. **Análise e projeto de sistemas.** 2. ed. Rio de Janeiro: LTC, 2012.

ANEXO

ANEXO A - Planejamento das sprints e tarefas

Sprint	Funcionalidade/tarefa		Dt. Inicial	Dt. Final	Semanas trabalhadas	Dt. Inicial alterada	Dt. Final alterada
1	Definir ferramentas para o projeto Estruturar banco de dados	- Definir ferramentas para o desenvolvimento do software; - Definir ferramenta para o desenvolvimento do banco de dados; - Criar as tabelas do banco de dados	11/02/2016	03/03/2016	3/2		25/02/2016
2	Manter usuário Manter morador	- Definir layout; - Cadastrar, Listar, Alterar, Excluir; - Definir regras de acesso ao sistema para os usuários; - Correções	04/03/2016	24/03/2016	3/2	26/02/2016	11/03/2016
3	Manter veículo Manter unidade	- Definir layout; - Cadastrar, Listar, Alterar, Excluir; - Criar regra para os cadastros das unidades de acordo com o tipo de condomínio: vertical ou horizontal; - Correções	14/03/2016	29/03/2016	2/2		
4	Manter encomendo Manter acesso	- Definir layout; - Cadastrar, Listar, Alterar, Excluir; - Criar regras para os tipos de acesso ao condomínio: morador, visitantes ou prestadores de serviços; - Correções	30/03/2016	13/04/2016	2/2		
5	Controle de acesso Visualização de acessos	- Definir layout; - Cadastrar, Listar, Alterar, Excluir; - Correções	14/04/2016	29/042016	2/2		
6	Integrar novas tecnologias ao sistema	- Analisar funcionamento dos equipamentos biométricos; - Integração dos hardwares via software; - Leitor de digital; - Leitor de tag - Realizar a integração ao sistema; - Correções.	02/05/2016	16/05/2016	2/4		06/06/2016
7	Manter biometria	- Definir layout; - Cadastrar, Listar, Alterar, Excluir; - Correções.	07/06/2016	21/06/2016	2/2		