Universidade Federal do Rio de Janeiro

Escola Politécnica

MBA em Governança, Projetos e Serviços de TI
(MBGPS)

ESTUDO DE CASO DE MELHORIA NO PROCESSO DE DESENVOLVIMENTO DE SOTWARE COM ABORDAGENS ÁGEIS

Autor:

Jessyka Miele de Mendonça

Orientador:

Wagner Rodrigues Ribeiro, M. Sc.

Examinador:

Cláudio Luiz Latta de Souza, M. Sc.

Examinador:

Nilton José Rizzo, D. Sc.

Examinador:

Norberto Ribeiro Bellas, M. Sc.

Rio de Janeiro
Dezembro/2021

Declaração de Autoria e de Direitos

Eu, **Jessyka Miele de Mendonça** CPF 147.730.057-00, autora da monografia *Estudo de caso de melhoria no processo de desenvolvimento de software com abordagens ágeis*, subscrevo para os devidos fins, as seguintes informações:

1. A autora declara que o trabalho apresentado na defesa da monografia do curso de Pós-Graduação, Especialização MBA - Governança, Projetos e Serviços de TI da Escola Politécnica da UFRJ é de sua autoria, sendo original em forma e conteúdo.
2. Excetuam-se do item 1 eventuais transcrições de texto, figuras, tabelas, conceitos e ideias que identifiquem claramente a fonte original, explicitando as autorizações obtidas dos respectivos proprietários quando necessárias.
3. A autora permite que a UFRJ, por um prazo indeterminado, efetue em qualquer mídia de divulgação a publicação do trabalho acadêmico em sua totalidade ou em parte. Essa autorização não envolve ônus de qualquer natureza à UFRJ ou aos seus representantes.
4. A autora declara ainda ter a capacidade jurídica para a prática do presente ato, assim como ter conhecimento do teor da presente Declaração, estando ciente das sanções e punições legais no que tange a cópia parcial ou total da obra intelectual, o que se configura como violação do direito autoral previsto no Código Penal Brasileiro nos art. 184 e art. 299, bem como na Lei 9.610.
5. A autora é a única responsável pelo conteúdo apresentado nos trabalhos acadêmicos publicados não cabendo à UFRJ, aos seus representantes, ou ao(s) orientador(es), qualquer responsabilização/indenização nesse sentido.
6. Por ser verdade, firmo a presente declaração.

Rio de Janeiro, 04 de dezembro de 2021.

_____

Nome Completo

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Av. Athos da Silveira, 149 - Centro de Tecnologia, Bloco C, sala - 212, Cidade Universitária Rio de Janeiro – RJ - CEP 21949-900.

DEDICATÓRIA

Dedico esse trabalho a minha família que tanto admiro e que sempre me apoiou e aos meus professores que me passaram todo o ensinamento durante esse curso.

AGRADECIMENTO

Dedico este trabalho a minha família que contribuiu de forma significativa à minha formação, aos meus amigos que me ajudaram nessa jornada e a Deus. Este projeto é uma pequena forma de retribuir a confiança em mim depositada.

RESUMO

Este trabalho analisa o uso do *framework Scrum* dentro do estudo de caso de uma empresa privada que tem a cultura baseada em abordagem preditiva e costumava ter problemas em suas entregas finais. O trabalho também mostra a importância dos testes nesse processo. Os métodos ágeis surgiram décadas atrás, tendo como proposta uma nova abordagem de eliminar documentação desnecessária, focando o cliente e maximizando sua satisfação, da equipe e da liderança. No *framework Scrum* por exemplo, um dos métodos ágeis mais utilizados, o time trabalha com ciclos constantes de desenvolvimento e a cada iteração agrega algum valor ao produto desenvolvido ao mesmo tempo em que busca a melhoria contínua do processo. A proposta pelo *framework Scrum* agrega valores com soluções que se adequam a problemas complexos, com o benefício de transparência, adaptabilidade e motivação. Os processos iterativos são usados para melhorar a comunicação e aumentar a cooperação entre os membros do time tendo sempre como objetivo entregar um produto mais adequado às necessidades do cliente e de forma mais rápida que os métodos preditivos. A utilização do *framework* nesse estudo de caso mostrou grande melhoria em eficiência e eficácia para todo o projeto.

Palavras-Chave:
(*Scrum*, XP, Metodologias, Ágil, Teste).

ABSTRACT

This work analyzes the use of the Scrum framework within the case study of a private company that has a culture based on a predictive approach and used to have problems in its final deliveries. The work also shows the importance of testing in this process. Agile methods emerged decades ago, proposing a new approach to eliminating unnecessary documentation, focusing on the customer and maximizing satisfaction of the team and leadership. In the Scrum framework, for example, one of the most used agile methods, the team works with constant development cycles and each iteration adds some value to the developed product while seeking continuous process improvement. The proposal by the Scrum framework adds values with solutions that fit to complex problems, with the benefit of transparency, adaptability and motivation. Iterative processes are used to improve communication and increase cooperation between team members, always aiming to deliver a product better suited to the customer's needs and faster than predictive methods. The use of the framework in this case study showed great improvement in efficiency and effectiveness for the entire project.

Keywords:
(Scrum, XP, Agile, Methodologies, Test).

SIGLAS

C3	Crysler Comprehensive Compensation on System
PO	Product Owner
TDD	Desenvolvimento Dirigido por Testes
XP	Extreme Programming
GQS	Garantia de Qualidade de Software

LISTA DE FIGURAS

LISTA DE QUADROS

Sumário

Capítulo 1: Introdução

Para minimizar os problemas de falhas em projetos que adotaram métodos preditivos, diversos métodos ágeis de desenvolvimento estão sendo usados em projetos de software. Estes métodos possuem como foco a satisfação do cliente, preocupando-se com a entrega incremental de software desde o início do desenvolvimento até o fim. O trabalho em questão aborda um estudo de caso que se refere a um projeto de desenvolvimento de software da empresa privada Dllnet TI, que utiliza abordagem preditiva como processo. A utilização dessa abordagem para sistemas complexos causa diversos problemas como: sistemas que são entregues aos clientes com atraso, orçamentos acima do esperado e requisitos que não são devidamente atendidos causando a insatisfação do cliente. Baseado no problema do projeto, foi proposta a adoção do gerenciamento ágil.

O gerenciamento ágil de projetos propõe geração de soluções eficientes e eficazes que são implementadas em ambientes dinâmicos onde as mudanças são contínuas e benéficas e são absorvidas pelo projeto ao longo do seu desenvolvimento (Conforto, 2009). Um dos conceitos do gerenciamento ágil é proporcionar entregas rápidas e pequenas que sempre agreguem valor ao cliente (Vicente, 2010). Para certificar que o objetivo está sendo alcançado, uma constante comunicação entre o desenvolvedor e o cliente é realizada, garantindo que os requisitos solicitados por este sejam atendidos (Paetsch et al., 2003).

A dinâmica dos métodos ágeis impacta diretamente na condução de um projeto de software, por este motivo sua utilização indiscriminada sem a compreensão do contexto e o real problema que se deseja resolver tem provocado um forte impacto na forma de se conduzir projetos de software sensíveis a alterações (Vicente, 2010). O XP (*eXtreme Programming)* (Beck et al., 2001) e o Scrum (Schwaber, 2004) são frameworks ágeis muito adotados.

A qualidade é inerente ao conceito de agilidade, ou seja, não existe entrega ágil que não possua qualidade. Os testes de software consistem na observação da execução do próprio software para validação do comportamento esperado e identificar possíveis erros (Bertolino, 2007). Em várias etapas, os casos de testes devem ser planejados conforme seus objetivos, mostrando as desconformidades com os requisitos do usuário ou o modelo específico ou medindo *performace*, *stress*, usabilidade, confiabilidade entre outros (Paetsch et al., 2003). Na perspectiva do provedor de software a qualidade é uma condição indispensável para competir com sucesso (Paetsch et al., 2003).

A execução de teste de software no cenário dos métodos ágeis é de grande importância, pois, diferentemente dos métodos preditivos nos quais os testes são feitos apenas ao final do processo de desenvolvimento, os testes ágeis acontecem rotineiramente. É de interesse do time que os defeitos sejam encontrados o quanto antes. Também é de interesse do time receber feedbacks constantes do cliente (Paetsch et al., 2003). Contudo, ainda em muitas empresas que utilizam algum tipo de método ágil em seu processo de desenvolvimento se perdem na hora de aplicar os testes de software e por muitas vezes veem projetos fracassarem pela falta de prática na aplicabilidade do método ágil que geralmente não é adotado por completo.

O presente trabalho tem por objetivo realizar a análise da adoção do framework Scrum na empresa Dllnet TI em busca de oportunidade de melhorias que foram observadas no processo de transformação ágil tendo como piloto o desenvolvimento de um sistema farmacêutico. Essas melhorias envolvem também a participação do time de qualidade de software, visto que, a empresa não mostra a devida importância para a essa equipe, não envolvendo o time nos projetos mais importantes da empresa.

Este estudo se justifica na necessidade da melhoria do processo desta empresa, demonstrando como é possível alcançar seus objetivos aprimorando a gestão, organização e conquistando a satisfação de seus clientes.

Este trabalho está dividido em 5 seções. A primeira e a segunda seção apresentam os métodos de desenvolvimento ágeis e qualidade de software, a terceira apresenta a proposta para solucionar o problema do estudo de caso baseado nos métodos de pesquisa e a quarta apresenta os resultados obtidos. Por fim, a quinta sessão contém a conclusão final com os possíveis trabalhos futuros.

Capítulo 2: Métodos de Desenvolvimento Ágeis e Qualidade de *software*

Em desenvolvimento de *software* em abordagem preditiva os engenheiros de *software* só conseguiam ir para fase de implementação após as especificações de requisitos estarem devidamente feitas e bem definidas (Baresi et al., 2006). Especialistas em desenvolvimento de software, incomodados com o alto nível de insucesso da abordagem preditiva para um ambiente de grande incerteza como desenvolvimento de *software*, desenvolveram métodos iterativos, incrementais e adaptativos para um desenvolvimento mais eficaz focando em qualidade nas entregas, qualidade de vida dos times e satisfação do cliente (Vicente, 2010).

Desenvolvimento ágil tem por característica uma extensa colaboração e comunicação coletiva que incluem as partes interessadas compreendidas pelos desenvolvedores e os usuários que passam por ciclos de descoberta, ação e validação. Esse ambiente colaborativo gera um espaço para adaptação e melhoria contínua do processo (Nerur et al., 2005).

2.1 Manifesto Ágil

O Manifesto Ágil surgiu de uma reunião ocorrida em fevereiro de 2001 onde 17 profissionais de desenvolvimento de software que já adotavam práticas ágeis e já tinham obtido ganhos com a implementação de tais conceitos, os elencaram em um documento e este foi propagado por toda a comunidade (Beck, 2001). Tal documento é uma declaração de bons valores e princípios fundamentais para o desenvolvimento de *software*, que se postos em prática apresentam benefícios para clientes e desenvolvedores proporcionando um ambiente colaborativo onde a comunicação e a criatividade são sempre incentivadas e onde o sistema está sempre pronto, porém em constante evolução (Vicente, 2010).

O Manifesto Ágil possui valores descritos são explicitados melhor, conforme (Highsmith e Cockburn, 2001):

- **Indivíduos e interação** mais em consideração que os processos e ferramentas; em métodos de projetos preditivos os clientes são mais ativos na especificação do requisito com pouca participação em outra atividade. Em métodos ágeis, os desenvolvedores e clientes possuem participação mais frequente no projeto determinando de forma

síncrona as funcionalidades a serem implementadas e posteriormente testadas pelo cliente.

- **Participação do cliente** mais considerado que negociação de contratos; o cliente deve estar próximo dos desenvolvedores, estando em um mesmo time e desta forma contribuir para uma comunicação mais estreita e clara. Ele deve inspecionar o que está sendo desenvolvido e a se adaptar a possíveis mudanças. Com uma participação mais ativa do cliente as mudanças podem ocorrer de forma mais rápida e bons resultados no projeto podem ser obtidos com menos recursos financeiros.
- **Software em operação** é mais importante que documentações detalhadas; satisfação do cliente ao receber entregas ágeis e contínuas testadas e funcionais sem a necessidade de uma documentação complexa e extensa.
- **Adequar a mudanças** mais importante seguir um planejamento; menos preocupação com planos de projetos adequando-se melhor as mudanças durante o projeto ágil. Através de pequenos ciclos de desenvolvimento e entre outras práticas do ágil o custo de mudança constante é reduzido.

Em métodos preditivos os clientes também podem escrever os requisitos sabendo do que o software precisa inicialmente, porém, muitas vezes, as necessidades mudam, e os desenvolvedores precisam de uma especificação detalhada para desenvolver levando em conta o que foi especificado. Essa dinâmica muitas vezes gera entendimento equivocado por parte dos desenvolvedores que implementam funcionalidades que não atendem totalmente ou em parte as necessidades do cliente e, portanto, precisarão ser alteradas ou geram atividades além do que foi solicitado (Vicente, 2010). Em métodos ágeis de desenvolvimento os desenvolvedores e clientes entendem juntos as necessidades e descrevem os requisitos necessários para se chegar a melhor solução. As distinções de pensamento entre os métodos preditivos e os métodos ágeis se apresentam em várias rotinas diferentes como na definição dos papeis e suas responsabilidades dentro do projeto. Essas diferenças são resumidas no quadro 2.1.1:

Quadro 2.1.1: Principais diferenças entre o desenvolvimento preditivo e o desenvolvimento ágil

	Desenvolvimento Preditivo	**Desenvolvimento Ágil**
Suposições Fundamentais	Sistemas são totalmente especificáveis, previsíveis e são feitos por meio de um planejamento detalhista e extensivo	Alta qualidade, software adaptativo, pode ser desenvolvido por equipes pequenas utilizando os princípios de melhoria contínua do projeto e testes baseados no feedback rápido e em mudanças
Processo envolvido no projeto	Deliberado e formal, com sequência linear de passos, formulação e implementação separadas, dirigida a regras.	Emergente, iterativo e exploratório, conhecimento e ação são inseparáveis, mais do que regras formais
Modelo de desenvolvimento	Modelo de ciclo de vida (Cascata, Espiral)	Modelo de entrega evolucionário
Controle	Centrado no processo	Centrado nas pessoas
Comunicação	Formal	Informal e importante
Papel do cliente	Importante	Crítico
Gerenciamento do conhecimento	Explícito	Tácito
Características Principais	Controle e direção Evitar conflitos Formaliza inovação Gestor é controlador Projeto precede a implementação	Colaboração e comunicação; integra diferentes visões de mundo Adota conflitos e argumentações Encoraja a exploração e criatividade; oportunismo Gerente é facilitado
Forma/Estrutura Organizacional Desejada	Burocrática, com alto grau de formalização (dirigido a grandes organizações).	Projeto e implementação são inseparáveis e se desenvolvem iterativamente Forma/Estrutura Organizacional Desejada Burocrática, com alto grau de formalização (dirigido a grandes organizações). Flexível e participativo, encorajando uma ação social cooperativa (dirigida a pequenas e médias organizações)

Fonte (Nerur e Balijepally, 2007; Nerur et al., 2005 *apud* Vicente, 2010)

O tamanho e o escopo da equipe podem alternar dependendo do tamanho do projeto, além disso, vários métodos podem ser aplicados independentemente da complexidade do projeto. Os métodos ágeis são os mais indicados para equipes pequenas, porém, outros métodos podem ser utilizados para equipes maiores (Vicente, 2010). Em síntese, métodos preditivos exigem processos detalhados e muito planejamento para que o desenvolvimento seja eficaz, ao contrário dos métodos ágeis que enfrentam a realidade de algo não previsível focando esforços em pessoas e na comunicação entre elas (Vicente, 2010).

2.2 *Scrum*

Scrum é um *framework* (estrutura) ágil e leve em termos de gerenciamento de projetos que agregam valor com soluções que se adequam a problemas complexos. Os processos se baseiam na iteratividade e integração das equipes tendo em vista que processos iterativos naturalmente estimulam uma melhoria na comunicação, aumento na cooperação entre as pessoas, e proteção da equipe contra impedimentos. Em resumo, o objetivo do *Scrum* é entregar um produto mais adequado as necessidades do cliente, e de forma mais rápida do que as obtidas com métodos preditivos (Cervone, 2011).

O desenvolvimento dos projetos é gerenciado através de *Sprints* que são períodos temporais que variam entre quinze e trinta dias. Durante esse tempo o time se reúne diariamente no rito conhecido como *daily* ou reunião de pé, onde cada integrante da equipe é estimulado a falar sobre as atividades nas quais está trabalhando. O *Scrum* deixa de livre escolha para os desenvolvedores técnicas especificas de desenvolvimento e práticas para o processo de implementação, ou seja, a atividade de gestão é constante e tem por objetivo identificar inconsistências, deficiências ou impedimentos dentro das práticas utilizadas (Abrahamsson, 2003).

O *Scrum* é indicado para projetos que apresentam um alto grau de mudanças e necessitam de flexibilidade quanto aos requisitos, tendo em vista que os processos dessa metodologia se destacam pela boa comunicação entre os integrantes do time e o cliente, assim como pelos constantes *feedbacks* do demandante (Vicente, 2010). Apesar disto, durante o estudo, descobriu-se de forma surpreendente que sua implementação foi verificada em apenas 5% das empresas de desenvolvimento. Em geral o *Scrum* vem sendo usado por empresas que procuram por certificações de qualidade. (Marchenko e Abrahamsson, 2008)

2.2.1 Práticas do *Scrum*

As práticas do *Scrum* destacam o uso de um conjunto de "modelos de processo de *software*" que são muito eficientes para projetos que tem prazos emergenciais e requisitos com constantes alterações. Para cada modelo de processo é definido um grupo de atividades de desenvolvimento que são conduzidas pelo PO (*Product Owner),* o *Scrum Master* e a equipe *Scrum* (Vicente, 2010). O PO é uma pessoa dentro da organização responsável por garantir que o produto a ser entregue atenda as expectativas do cliente e que ele tenha sido desenvolvido com qualidade, já o *Scrum Master* orienta a equipe *Scrum* como um todo atua como representante deles (Vicente, 2010). O ciclo de vida do *Scrum* será apresentado na Figura 2.2.1.1 e descritas logo a seguir:

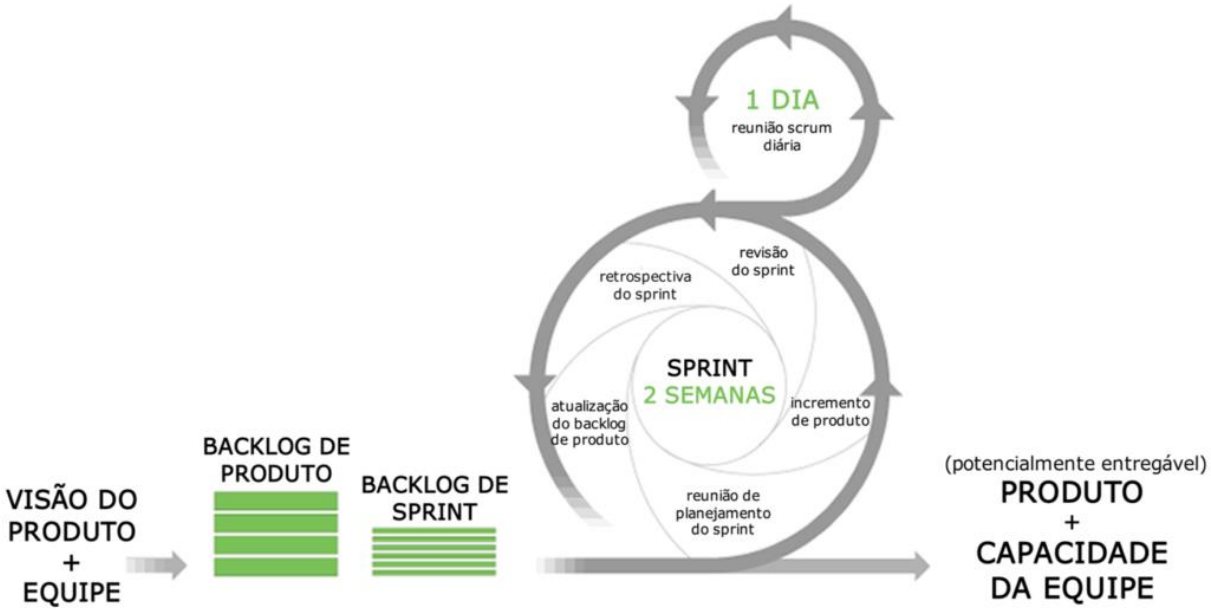

Figura 2.2.1.1 – Ciclo de Vida do Scrum.

Fonte: SEMERU (2011, online)

Visão do Produto e *Backlog*

O ciclo inicia-se com a visão do produto que precisa ser desenvolvido e com o objetivo a ser alcançado. Posteriormente é feito o *backlog* do produto que é uma lista de requisitos a serem feitos e que agregam valor de negócio para o cliente. Elas são chamadas de histórias (Vicente, 2010). Após isto essas histórias são quebradas em histórias menores e priorizadas pelo PO. Isso dá origem ao *backlog* da sprint. *Backlog* pode ser comparado a uma "pilha" de pedidos em espera paras serem feitos.

Planejamento do *Sprint*

Sprint é uma reunião de pessoas que estão envolvidas num projeto para promover um desenvolvimento mais focal no projeto. Em todo início de uma *Sprint* é realizada uma reunião de planejamento chamada *Sprint Planning*. Essa reunião é dividida em duas partes, a primeira baseia-se no que fazer, e a segunda em como fazer. A duração deve ser em média de 4 horas para cada.

Reunião de Planejamento I e II (Escolha do *Backlog*)

Na reunião do planejamento I o PO apresenta os itens de *backlog* que possuem uma maior prioridade para o time de desenvolvimento. A equipe então rascunha soluções técnicas e define quais itens serão possíveis de serem entregues até o final da *sprint*. Na reunião II o time irá se reunir para definir as tarefas que irão executar e estimá-las. (Vicente, 2010)

Sprint

O *Sprint* é a fase de desenvolvimento do produto, que pode levar de uma a quatro semanas. Uma vez decidia a duração da sprint ela deve ser mantida até o fim. Mudanças, remoções, adições e priorização de histórias só podem serem feitas pelo PO na reunião de planejamento. Em caso de mudanças a *Sprint* pode ser cancelada, caso contrário, a visibilidade da produtividade do time seria prejudicada. (Vicente, 2010).

Reunião *Scrum* Diária (*Daily*)

Daily é uma breve reunião que dura em torno de 15 minutos e se repete todos os dias durante a *Sprint.* Nessa reunião cada integrante do time faz um resumo das suas atividades do dia anterior, informa o que pretende fazer naquele dia e se existe algo impedindo que ele com suas atividades. O *Scrum Master* facilita a reunião e trabalha para que os impedimentos sejam

resolvidos. As *dailies (daily meetings)* são importantes o time inspecionar se o trabalho está fluindo em direção à meta da *sprint*.

Reunião de Revisão da *Sprint (Rewiew)*

A *rewiew* ocorre ao final de cada Sprint onde durante esta reunião, o time mostra o que foi alcançado durante o *Sprint*. Os participantes do *rewiew* incluem o *Product Owner*, o *Scrum Master*, o time, gerência, clientes e engenheiros de outros projetos. Durante a reunião o projeto é avaliado em relação ao objetivo da geral da sprint.

Reunião de Retrospectiva da *Sprint*

A reunião de retrospectiva por sua vez ocorre ao final de cada *Sprint*. Dela participam todo o time *Scrum* junto com o *Scrum Master* que é o responsável por facilitá-la. Nesse rito todos os integrantes da equipe são estimulados a dar suas percepções de como foi a *Sprint*, ou seja, o time analisa tudo o que ocorreu na última iteração de desenvolvimento buscando levantar o que foi positivo e, portanto, pode ser repetido mais vezes, e identificar os problemas que ocorreram a fim de levantar melhorias para que eles não mais ocorram. É nesse momento também que o time procura se antever a problemas futuros e definir o que fazer para que eles não venham a se materializar. Vale ressaltar que durante a retrospectiva o time é livre para falar sobre qualquer coisa, desde o processo em si, até situações que envolvam por exemplo o ambiente da empresa.

A diferença entre as reuniões é que a retrospectiva se preocupa com o como é desenvolvido e a revisão da *sprint* preocupa-se com o que foi desenvolvido (Vicente, 2010).

2.2.2 Papéis do *Scrum*

Segundo Vicente (2010) no *Scrum* existem três papeis: *Product Owner*, *Scrum Master* e Dev *Team* (na nova versão do *Scrum Guide*, chama-se *Developers*):

- ***Product Owner:*** tem o papel de garantir que o cliente esteja satisfeito com as entregas realizadas, bem como priorizar as funcionalidades que serão desenvolvidas. Além destas já citadas, suas responsabilidades são: definir o objetivo da *sprint*, gerenciar o *backlog*, reunir ideias e priorizá-las, aceitar e rejeitar resultados do trabalho de equipe ao fim de cada *sprint* juntamente com o cliente.
- ***Scrum Master:*** facilitar a adoção a do *Scrum*. Ele é encarregado de guiar a equipe, mediar a comunicação entre está e o PO, melhorar a produtividade e protegê-la de

interferências externas. O *Scrum Master* também trabalha para remover impedimentos, garantir o processo, facilitar as *cerimônias Scrum*, bem como propor ferramentas de facilitação de comunicação e gestão.

- ***Developers:*** É composto por todas as pessoas que participam do desenvolvimento da *Sprint*. Os integrantes da equipe são multifuncionais e auto gerenciáveis e são encarregados de desenvolver o produto e apresentar ao PO o que foi desenvolvido. Esses times podem ser compostos por vários perfis, como por exemplo: analistas de requisitos, *designers*, analistas da qualidade, desenvolvedores, etc. Comumente o tamanho de equipe se limita a no máximo 10 pessoas.

2.3 XP (*eXtreme Programming*)

Programação extrema (XP) é uma metodologia ágil que se adequa bem aos projetos de *software* que tem rotineiras mudanças de requisitos (Vicente, 2010). A ideia desse método se originou de conversas entre Kent Beck e Ward Cunningham a partir de suas experiencias com desenvolvimento de *software* na linguagem *Smalltalk* (Sato, 2007). O XP se mostra mais pertinente quando há necessidade de se apresentar valor imediato para o cliente, uma vez que o *software* vai sendo incrementado ao longo do processo de desenvolvimento (Beck, 2000). Seu objetivo principal é unir excelência, maior produtividade, baixo custo, geração de poucos defeitos e grande retorno de investimento. (Sato, 2007). As principais características do XP são:

- Método leve
- Método que destaca o desenvolvimento de software
- Funcional para time de qualquer tamanho
- Adaptável a qualquer tipo de mudança de requisitos

O XP tem iterações e incrementos de versões curtas além de *feedbacks* rápidos. O planejamento ocorre de forma incremental com participação do cliente e constante comunicação entre a equipe. O XP atua com uma prática que está diretamente ligada ao desenvolvimento do *software* a chamada refatoração, ou seja, melhoria do código, e também integração, programação pareada e testes de forma contínua. (Vicente, 2010)

Algumas características do XP como: refatoração, programação pareada, adaptação a mudança, desenvolvimento iterativo, integração continua e ênfase nos testes são elementos primordiais na cultura da comunidade da linguagem *Smalltalk* (Sato, 2007). Kent Beck publicou alguns artigos para tratar dessa metodologia. No primeiro deles tratou sobre testes automatizados *SUnit* e logo depois sobre o JUnit usando Java. Mais tarde foi criada ferramentas para permitir o uso de testes para diversas linguagens, ele ficou conhecido como XUnit (Sato, 2007).

Essas características e ideias foram aplicadas no projeto C3 (*Chrysler Comprehensive Compensation System*), onde pela primeira vez, todas as práticas de programações já conhecidas foram utilizadas de forma extrema, como revisão de código que é feita em pares, a escrita de testes unitários antes do próprio código (Sato, 2007). A partir dessa experiência se originou a XP.

Principais aspectos do método XP na definição de Beck e Andres (2004):

- Filosofia de desenvolvimento baseada na importância da comunicação, *feedback*, cumplicidade, respeito e coragem.
- Várias práticas comprovadamente funcionais para melhorar o desenvolvimento de *software*. As práticas são técnicas utilizadas no dia-a-dia dos membros de uma equipe XP.
- Um conjunto complementar de 14 princípios que são técnicas inteligentes que ajudam a colocar em prática os valores pregados.
- Um grupo que compartilha os mesmos valores e práticas.

2.3.1 Ciclo de vida do XP

O ciclo de vida do desenvolvimento do XP é composto de seis fases: exploração, planejamento, iterações para as versões, produção, manutenção e morte (Beck, 2000). O time de desenvolvimento praticamente faz tudo simultaneamente por meio de interações que são adicionadas semanalmente. O desenvolvimento é baseado em pequenas histórias que possuem um valor para o cliente, ou seja, toda semana uma história é entregue pelo time passando por todas as fases de desenvolvimento. A Figura 2.3.1.1 a seguir ilustra melhor o ciclo de vida XP. Em seguida cada uma das fases será descrita:

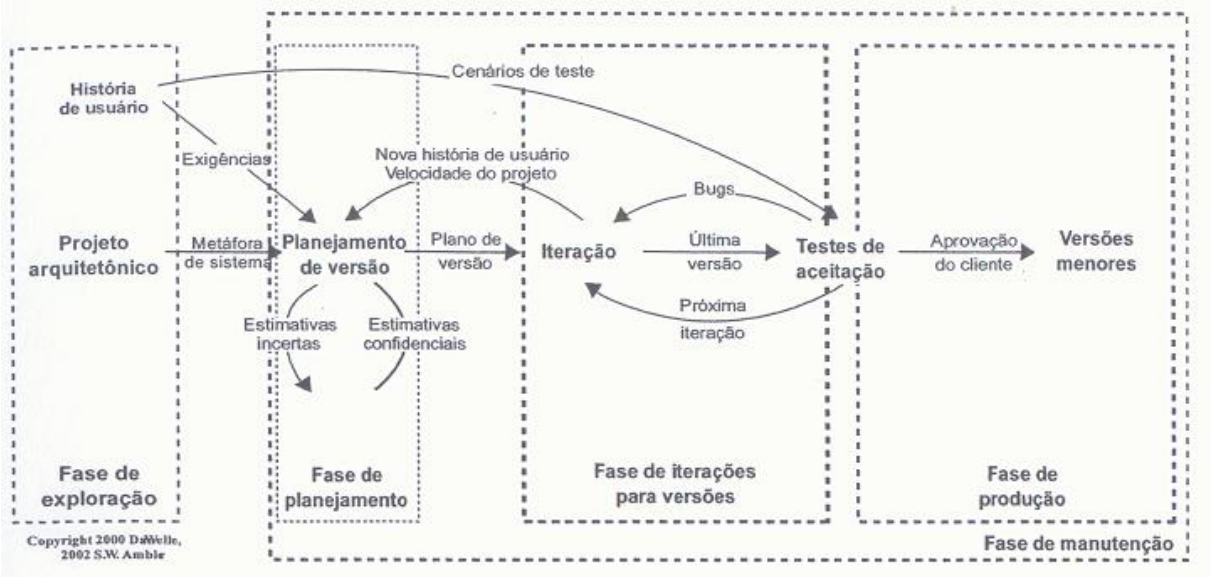

Figura 2.3.1.1 – Ciclo do *framework* do XP.

Fonte: SEMERU (2011, online) *apud* HENRAJANI, 2007

Fase de Exploração

Nesta fase as características e funcionalidades do sistema são brevemente descritas pelo cliente. Essas informações se transformam em histórias escritas em cartões para que os programadores já possam pensar em quais tecnologias e práticas que serão usadas para desenvolvê-las. (Vicente, 2010)

Fase de Planejamento

A fase de planejamento dura apenas poucos dias. Nela o cliente prioriza as histórias e os desenvolvedores estimam seus esforços para desenvolver cada uma delas. As histórias são priorizadas da forma mais importante para o negócio e é possível até mesmo haver dependência nelas. (Vicente, 2010 *apud* Pressman, 2006). O tempo é medido em semanas e as histórias não podem levar mais que três semanas para ficarem prontas. Os testes automatizados de aceitação já são criados para verificação de implementação de cada história de acordo com que foi pedido pelo cliente eles descrevem a necessidade do negócio e devem ser executados de forma automatizada no final da fase de interação. A data da primeira versão é concordada em conjunto com equipe e cliente. (Vicente, 2010)

Fase de interações para as versões

A fase de interação para versões é a fase na qual os programadores escolhem as histórias com limite de até 3 semanas que serão implementadas. Escolhendo as histórias é concebida a arquitetura inicial do sistema na qual o cliente escolhe qual delas será selecionada para cada interação (Vicente, 2010). As histórias podem ser implementadas na hora ou em semanas. No XP é utilizado a arquitetura de incremento para melhorar o projeto.

Nessa fase usamos o desenvolvimento dirigido a testes, pois, ele junta várias atividades como: teste, codificação, projeto e arquitetura. Com esse processo, os desenvolvedores trabalham em pares, o que aumenta a agilidade e capacidade da atividade implementada com menos falhas. Os desenvolvedores também gerenciam o ambiente de desenvolvimento fazendo o controle das suas próprias compilações (*build*) de forma automatizada. Isso garante que os outros desenvolvedores interajam entre si e o código esteja sempre saudável para ser integrado qualquer momento. Os testes de aceitação que são criados pelo cliente são executados no final de cada interação, e na última o sistema está pronto para próxima fase. (Vicente, 2010)

Fase de Produção

Para o sistema ser entregue é necessário que sejam feitos mais testes para verificar a qualidade da entrega e performance do código. Mesmo nessa fase é possível haver mudanças nas histórias, mas, elas precisam ser debatidas com o cliente para se averiguar a viabilidade de incluir tais alterações na versão a ser liberada. Em geral as ideias e sugestões são documentadas a fim de serem construídas em implementações futuras ou na fase de manutenção. (Vicente, 2010)

Fase de manutenção

Essa fase tem por objetivo permitir que melhorias e novas funcionalidades sejam desenvolvidas ao sistema após a primeira versão do sistema haver sido entregue. Nesta etapa novos integrantes podem ser agregados ao time de desenvolvimento. (Vicente, 2010)

Fase de Morte

A fase de morte começa quando não existe mais a geração de novas histórias. Nesse momento é necessário escrever documentações para que equipes posteriores que venham a atuar no sistema consigam efetuar sua manutenção. Caso não atinja o resultado esperado pelo cliente o projeto também pode morrer. (Vicente, 2010)

2.3.2 Valores do XP

Os valores são fundamentos maiores que são utilizados para uma determinada situação. O *feedback* é um dos valores mais fáceis de se enxergar no XP, pois nele a comunicação é constante. O XP é baseado nos seguintes valores: comunicação, simplicidade, *feedback*, coragem e respeito (Beck e Andres, 2004). Os valores dão razão as práticas, as práticas evidenciam esses valores e os princípios traduzem os valores em prática.

- **Comunicação**: O XP dá maior importância ao valor da comunicação. Essa metodologia acredita que o maior problema de um projeto de *software* ocorre por questões de comunicação, por essa razão o XP preza pela comunicação defendendo inclusive que o time trabalhe junto, literalmente no mesmo local, inclusive para fornecer um ambiente que proporcione a programação em paralelo.
- **Simplicidade**: segundo valor do XP. Ele defende que a equipe busque mais simplicidade para resolver os problemas existentes. Segundo o Kent a simplicidade é o valor mais forte do XP, pois encontrar as soluções mais simples não é fácil (Beck e Andres, 2004).
- ***Feedback:*** O XP funciona a partir de ciclos pequenos nos quais os *feedbacks* são constantes. Os valores do XP se integram e o *feedback* possui um importante papel dentro da comunicação e simplicidade.
- **Coragem:** É o quarto valor do XP. Nele Kent Back associa a coragem a uma ação frente ao medo (Beck e Andres, 2004). Ter coragem não significa agir de qualquer forma sem ser pensar nas consequências. A coragem juntamente com os outros valores se torna muito influente. Pode ser observada a coragem em alguns pontos do XP como: influenciar o desenvolvedor a trabalhar em par, permitir a priorização do cliente entre outros.
- **Respeito:** O respeito serve como a base para todos os outros quatro valores do XP. Segundo Kent, se os membros da equipe não se tratam com respeito o XP não funcionará (Beck e Andres, 2004). O reconhecimento das pessoas na produção do desenvolvimento e seu respeito mútuo é muito importante para o sucesso do projeto.

2.3.3 Princípios do XP

Os princípios do XP atuam como tradutores dos valores mostrando como colocá-los em prática. O princípio da humanidade insinua que a conversa supre um melhor relacionamento humano, comunicação. A comunicação escrita, por outro lado, permite maior compreensão dos fatos com retornos e *feedbacks*.

Dentro de um projeto XP as oportunidades de crescimento são criadas com contribuição, relacionamento com o time desenvolvimento e participação. Toda a equipe deve procurar a melhoria contínua do projeto e refinar suas atividades ao longo do tempo. As atividades do XP devem trazer benefícios e aplicar uma solução de estrutura e escalas. No contexto de melhoria, toda a equipe deve se preocupar com um fluxo contínuo que traz valores ao negócio e ao projeto ao mesmo tempo em que separam as tarefas e aceitam as obrigações para diminuir as ameaças.

Os problemas são vistos como oportunidades para melhorar e aprender cada vez mais. Eles devem ser resolvidos juntando práticas e soluções diversas. Finalmente a qualidade deve ser sempre construída, no reflexo da melhoria de outras características como; eficiência, produtividade e motivação. A qualidade não é uma variável de controle. (Vicente, 2010)

2.3.4 Práticas do XP

Práticas são as técnicas utilizadas no dia a dia da equipe XP para que seu objetivo ideal seja alcançado. Uma prática complementa a outra aumentando o seu efeito. As práticas evidenciarão os valores e os valores dão sentido a prática. As equipes escolhem as mais diferentes práticas para atender as condições adversas que se apresentem (Vicente, 2010). As práticas foram divididas em primárias e corolárias na segunda versão do XP. As primarias são mais independentes e implementadas de maneira segura e individual já as corolárias são mais difíceis de implementar sem que haja experiências com as práticas primarias.

Práticas primárias

As práticas primárias são aplicadas para melhorar o desenvolvimento de *software*. Elas podem ser facilmente implementadas pois são consideradas seguras. São introduzidas de forma calma, para não haver mudança brusca na cultura da organização. A seguir as práticas primárias usadas no XP:

- **Histórias:** Pequenas descrições da necessidade do cliente**.**
- **Ciclos Semanais:** Deve se planejar o trabalho uma semana de cada vez havendo sempre uma reunião no início da semana para observar o progresso do projeto e quebrar em tarefas menores as histórias escolhidas pelo cliente para serem implementadas naquela semana.
- **Ciclo Trimestral:** Em um espaço temporal maior, as versões são planejadas em alto nível a cada trimestre. Durante o planejamento desse trimestre a equipe identifica os problemas e inicia as soluções escolhendo as histórias que serão implementadas.
- **Folga:** Após a criação de um vínculo de confiança entre a equipe e o cliente, a folga é acordada para evitar atrasos ou a necessidade de renegociação de escopo. O planejamento deve conter os espaços de folga.
- **Sentar Junto**: Os integrantes das equipes de desenvolvimento devem trabalhar juntos no mesmo espaço para que a comunicação seja mais favorável. Isto incentiva a troca de informações e os trabalhos em pares.
- **Time Completo:** A equipe precisa de pessoas adaptáveis e com todas as habilidades necessárias para o sucesso do projeto
- **Área de Trabalho Informativa**: No ambiente de trabalho é necessário que as informações estejam o mais acessível possível possibilitando a visualização como um todo do andamento do projeto. O ambiente deve ser agradável e comunicativo.
- **Trabalho Energizado:** Os desenvolvedores precisam estar bem e as horas de trabalho devem durar apenas pelo tempo em que o time se mantiver produtivo. Não adianta ter uma equipe desgastada pois a produtividade irá cair.
- **Programação em pares:** Sugere que o tipo de código seja escrito sempre por dois programadores estimulando o trabalho colaborativo. Isso permite que todos da equipe tenham a oportunidade de trabalhar juntos o que sempre melhora a comunicação entre todos. Essa é a prática mais conhecida no XP.
- ***Design* Incremental:** Para diminuir os custos, no início do projeto o design é implementado da forma mais simples e menos complexa possível. A medida que o projeto avança o *design* vai sendo incrementado.
- **Desenvolvimento Dirigido por Testes (TDD):** Nessa prática é necessário escrever o teste antes de se alterar ou se implementar um código novo. Isso permite que os

desenvolvedores já trabalhem na correção de erros e na implementação de melhorias antes que a funcionalidade trabalhada seja implantada no ambiente.

- ***Build* em 10 Minutos:** o sistema deve ser copilado e os testes automatizados devem ser executados por completo em no máximo 10 minutos. Isso é importante para se otimizar o tempo levado para se obter os feedbacks e descobrir possíveis problemas.
- **Integração Contínua:** garante que o código feito pela equipe será colocado em um repositório comum, facilitando a entrega do projeto mesmo com as variedades de mudanças do cliente.

Práticas corolárias

As práticas corolárias devem ser aplicadas incrementalmente, são mais complexas e difíceis de serem implementadas. Para aplicar essa prática é necessário ter bastante conhecimento nas práticas primárias, e também, só deve ser aplicada quando há baixo índice de defeitos no sistema. A seguir as práticas corolárias usadas no XP:

- **Envolvimento Real do Cliente**: pessoas que utilizam o sistema passam fazer parte do time, onde contribuem com o planejamento, define prioridades, histórias e testes.
- **Implantação Incremental:** substituição de poucas funcionalidades até que todo o sistema seja trocado.
- **Contrato de Escopo Negociável:** manter o escopo negociável, fixando qualidade, custo e tempo.
- **Pague pelo Uso:** é sugerido que o cliente pague toda vez que recebe uma nova versão do sistema.
- **Continuidade da Equipe:** a mesma equipe que mostrou eficiência permanece junta em outros projetos.
- **Diminuição da Equipe:** quanto mais capaz e produtivo o time for, é passível sua diminuição, para a partir desses formar novas equipes.
- **Análise da Causa Inicial:** a cada defeito encontrado, sua causa também precisa ser eliminada.
- **Código e Testes:** apenas os códigos e testes permanecem como documentação, outros tipos de documentos podem ser gerados posteriormente através deles.
- **Código Compartilhado:** o time inteiro pode alterar o código do projeto a qualquer momento, sem ter responsável por cada parte.

- **Repositório de Código Unificado:** todo o código deve ser mantido em um único repositório.
- **Implantação Diária:** todo o dia deve colocar uma nova funcionalidade em produção para que os usuários possam usar o quanto antes. A prática é realizada quando o número de defeito é baixo.

2.3.5 Papeis do XP

Os papeis no XP não são fixos, podem ser trocados e uma pessoa pode assumir mais de um papel. Na área gerencial, estes papéis devem estimular as pessoas da equipe buscando sempre uma melhoria contínua do projeto. Será descrito a seguir os principais papéis da equipe XP.

- **Usuários**: ajudam a escrever as histórias do XP e tomam decisões durante o desenvolvimento do projeto
- **Programadores**: estimam tarefas, dividem histórias em tarefas, escrevem o código-fonte, sugere soluções, ajudam os clientes.
- **Testadores:** ajudam a escrever os testes de aceitação, define os cenários em uma história, ajudam os clientes identificar erros, treinam os programadores a usar ferramentas de teste, ajuda a encontrar falhas no sistema.
- **Arquitetos de *software*:** realizam as refatorações no sistema, orientam a equipe, direciona a melhora da arquitetura, simplificam o sistema.

2.4 Teste de *Software*

É percebido que a área de teste está bastante em evidência nos métodos ágeis citados, com isso é importante entender melhor sobre teste de software e assim melhorar a qualidade da entrega do projeto. O Teste de *Software*, segundo Bertolino (2007) é um termo com diferente visão, de diferentes atividades, que vai desde o código feito pelo desenvolvedor (teste unitário) até a validação do cliente (teste de aceitação). Os casos de testes devem ser planejados de acordo com os objetivos, deixando em evidência as diferenças de requisitos, avaliando a força através do *stress*, invasão, medindo a performasse, usabilidade e confiabilidade.

A engenharia de *software* vem crescendo continuamente e consequentemente a área de teste de software vem evoluindo cada vez mais, preocupando-se com aplicações mais dinâmicas e reutilizáveis. Exemplos que podem ser explicitados, tem-se os testes embarcados e críticos,

serviço web e testes de componentes. Um ponto importante na atividade de teste é a sua automatização das técnicas, que tem como objetivo auxiliar a qualidade e produtividade dos testes.

2.4.1 Testes em projetos de vida cascata

A qualidade não é mais um ponto de vantagem, e sim essencial e indispensável para um projeto de sucesso. Segundo Tian (2005) a atividade de teste de software tem um papel principal nas atividades de garantia de qualidade de software (GQS), com o objetivo de mostrar os erros e defeitos no produto, aumentando a qualidade da entrega. O teste bem realizado é aquele que consegue objetivar os casos de teste pra observar a possível falha. Na figura 2.4.1.1 é exibida as atividades de teste e seus artefatos gerados.

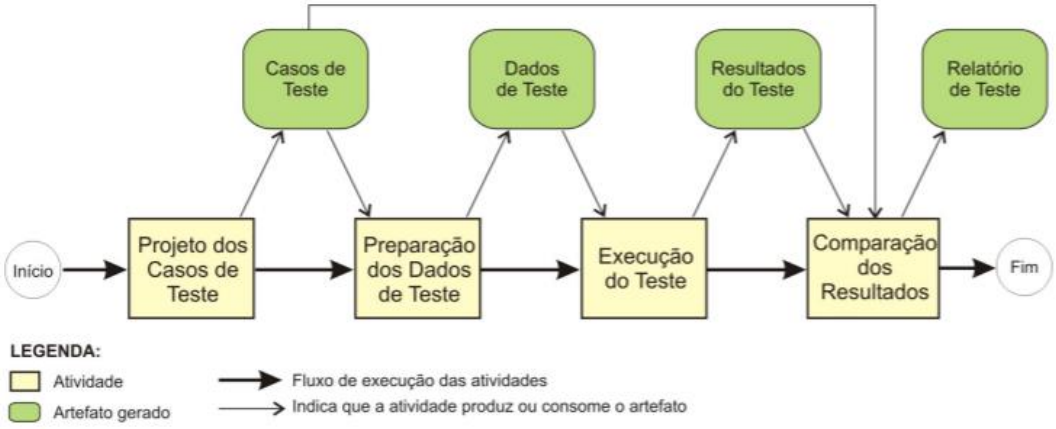

Figura 2.4.1.1 - Processo de teste de Software
Fonte: (Nakagawa, 2006 *apud* Vicente, 2010)

Vários objetos podem ser incluídos aos testes, como programas, estrutura dos dados, ou qualquer outro objeto que possa ser executável e utilizado na implementação da aplicação. Tudo isso representa casos de testes, que descreve um elemento de entrada e de saída. Os casos de testes são comparativos ao comportamento esperado e o comportamento da função exercida.

2.4.2 Fases do teste do software

A atividade de teste de software, em projetos preditivos, costuma-se ocorrer em dois tipos de estratégias, ao final de toda a construção do software ou de forma incremental. Segundo Myers et al. (2011) são descritas as seguintes fases do teste de software:

- **Teste de unidade:** o foco são as menores partes do sistema, que pode variar de classes a funções. Objetivo é mostrar os defeitos de lógica e evidenciar que cada unidade funciona de forma separada corretamente.
- **Teste de Integração:** ocorre logo após ao teste de unidade, e é nessa fase que as outras partes dos *softwares* são integradas e executadas, verificando se sua funcionalidade está ocorrendo de forma correta.
- **Teste de Sistema:** ocorre quando o sistema está com todas as partes integradas, completo. Seu objetivo é a verificação das funcionalidades que foram implementadas e especificadas. Todos aspectos são abordados até mesmo os de segurança e performace.
- **Teste de Regressão:** ocorre durante a manutenção do sistema, cada alteração pode impactar no que já foi homologado, ocorrendo riscos de novos defeitos, por isso é necessário realizar novos testes que garantem que as novas funcionalidades foram implementadas sem impactos.

2.4.3 Técnicas de testes

As técnicas de testes são escolhidas através do custo, prazo e recurso que será utilizado no projeto. As técnicas e seus critérios são fornecedores de fundamentos para condução e avaliação da qualidade do teste de *software*. As técnicas são; funcional, estrutural e baseada em defeitos (Vicente, 2010).

- **Técnica Funcional:** utiliza as especificações de requisitos para montar os casos de testes. Há dois problemas nessa técnica, um que as especificações podem estar incompletas e segundo é a limitação dos critérios que não garante que todas as críticas do produto sejam realizadas. O lado bom é que essa técnica pode ser utilizada em todas as fases de testes.
- **Técnica Estrutural:** chamada de caixa branca, é complementar a técnica funcional e se baseia em estrutura programada para descrever seus casos de teste.

- **Técnica Baseada em Defeitos:** defeitos mais frequentes cometidos pelos desenvolvedores são utilizados como informações.

2.5 Testes em métodos ágeis

Os métodos preditivos têm as suas atividades de teste ao final do processo de desenvolvimento, já os testes de métodos ágeis ocorrem durante as fases, procurando antecipar o quanto antes os defeitos em ciclos curtos de desenvolvimento, com *feedback* constante. A mudança de foco da equipe ágil é mais constante, com isso, os testes de regressão são fundamentais para que seja garantido que nada foi impactado. Nos projetos ágeis a uma grande preocupação em testar sobe a visão do cliente (Vicente, 2010). Para criar esses testes – teste de aceitação – há uma participação de todos da equipe de desenvolvimento.

Na criação de cada história, é necessário no mínimo um caso de teste de aceitação, que são fundamentados em exemplos dados pelo cliente. Esses testes ajudam a melhor compreensão das histórias. Nos projetos ágeis, toda equipe costuma ser responsável pela qualidade, e os testadores fazem parte da equipe. Desenvolvedor e testador, por muitas vezes, trabalhando juntos, melhorando a comunicação sobre a qualidade da entrega.

Preferivelmente toda a equipe deve saber da importância dos testes, desde o mais simples até os mais complexos. O teste ajuda toda a equipe a entender como deve funcionar a aplicação, servindo também, como *feedback* de tarefas concluídas. A qualidade de software possui várias dimensões, cada uma requer diferente abordagem de testes (Vicente, 2010). Neste contexto, as atividades de teste em métodos ágeis foram divididas em 4 categorias: TDD, Teste de negócio, Teste de negócio sobre o produto (usabilidade, exploratórios e etc...) e Teste de perfomarce (Crispin e Gregory, 2009).

Conforme o método ágil é escolhido a estratégia de teste de *software* e práticas são aplicadas. Conforme Crispin e Gregory (2009), a estratégia aplicada que possui um grande sucesso na indústria, contém as atividades; Visão geral do produto e planejamento, Ciclo interativo, Teste de sistema, Nova versão do produto. Na figura 2.5.1 é explicitado as fases das atividades.

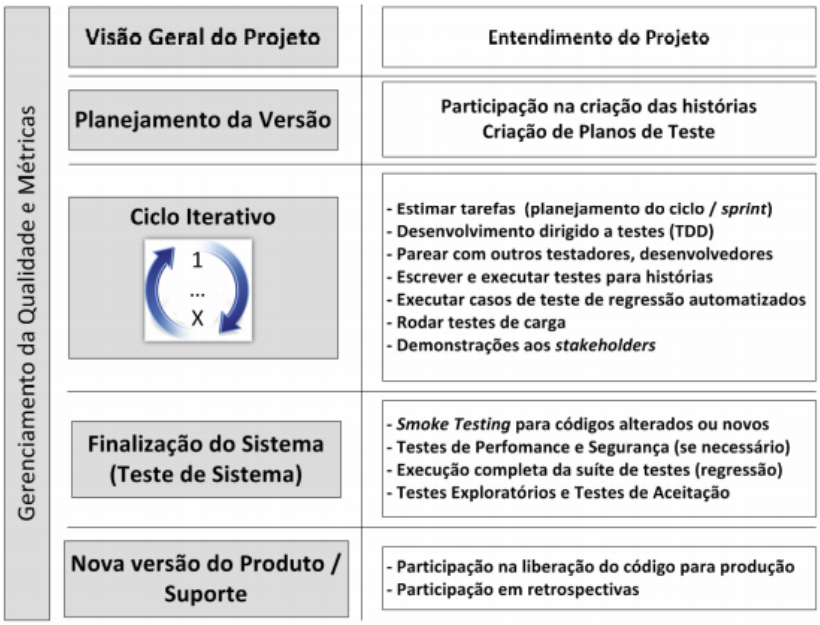

Figura 2.5.1 Estratégia de testes em projetos ágeis
Fonte: (Crispin e Gregory, 2009 *aput* Vicente, 2010)

Conforme a figura acima, a visão geral do projeto e o planejamento da versão, é a fase onde a equipe de teste é responsável pela ajuda no entendimento do projeto e histórias, além de criar os planos de teste. No ciclo interativo, é realizada a estimativa das tarefas, onde desenvolvimento é dirigido a testes TDD, e os testes são escritos e executados, contendo os testes regressivos automatizados, testes de cargas e demonstrações ao cliente. A finalização do sistema, é a fase onde o sistema é de fato testado, com testes exploratórios, de regressão e performance. Na nova versão do produto, a equipe de teste participa da liberação para produção e participa da retrospectiva.

O teste atribui-se grande importância no processo de desenvolvimento de um software utilizando os métodos ágeis. O teste fornece *feedback*, ajuda na comunicação e também na manutenção do projeto para que nada seja afetado. Há os testes extremos, que são os mais utilizados em testes tradicionais, ou seja, ao final da construção do sistema. Tem-se também o incremental, que é feito diariamente, toda vez que algo é criado no sistema, utilizado em métodos ágeis por meio de TDD (*Test Driven Development*), ou seja, escrever o teste unitário antes de desenvolver o código fonte. Os métodos ágeis têm as fases de testes diferenciadas, enfatizando o teste de unidade e de aceitação. Os ciclos de desenvolvimentos são mais curtos e por isso os testes são mais frequentes (Vicente, 2010).

Capítulo 3: Propostas

3.1 *Assessment*

O trabalho aborda um estudo de caso que se refere a absorção de *framework* de *Scrum* na área de planejamento de projetos de desenvolvimento de software visando melhorar a qualidade das entregas do produto da empresa privada Dllnet TI. A empresa Dllnet TI é uma consultoria de TI de pequeno porte que está situada na cidade do Rio de Janeiro. A sua principal insatisfação é a falta de qualidade e organização nas entregas.

Visando a melhoria que a empresa deseja alcançar, foi realizada uma reunião com todos os funcionários para entender melhor a estrutura da empresa, seus processos, ferramentas utilizadas, organograma, papéis e responsabilidade e o método de desenvolvimento. Nessa reunião foi exposto todo o problema da empresa; de processo, de comunicação interna, os atrasos nos projetos grandes e os prejuízos financeiros. Os desenvolvedores e analistas deixaram claro seus pontos de vista, principalmente nos problemas de comunicação.

Ao final dessa reunião foi realizado todo levantamento dos pontos mais falhos e entre eles a falta de planejamento, comunicação interna e com os clientes eram a mais evidente. Com realização dessa reunião, ficou claro também, que a empresa tem mostrado um resultado muito abaixo da sua expectativa, com todas as entregas fora do prazo, sem nenhuma qualidade e não atendendo as necessidades e satisfação do cliente. Visto todo o levantamento realizado nessa reunião, ficou inegável que o método de trabalho deveria mudar para uma abordagem eficiente e eficaz que organizasse o processo como um todo.

Processos da DLLnet

Ao realizar a reunião foi observado os seguintes pontos foram levantados:

- Organograma atual

A área de projetos de *software* da Dllnet é composta por 1 diretor, 2 pessoas responsáveis pela área comercial, 1 gerente de projetos, 2 analistas de requisitos, 4 testadores, 2 designers, 14 desenvolvedores. O organograma da equipe de projetos é exibido na Figura 3.1.1:

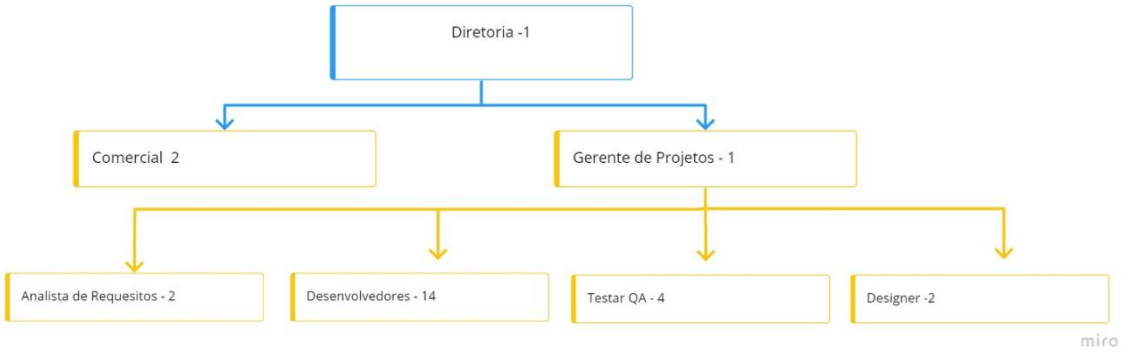

Figura 3.1.1 – Organograma da Empresa Dllnet TI

Fonte: Autor

- Método de desenvolvimento

A abordagem de desenvolvimento utilizada na empresa é preditiva, controle centralizado no processo, com comunicação formal e gestão controladora. A empresa não organiza as entregas e os documentos de requisito são mal descritos. Os desenvolvedores não conseguem ter uma comunicação clara com seus gestores, muitas vezes gerando retrabalho. As priorizações não eram geridas,

- Papéis e responsabilidades

Os papéis não eram bem definidos dentro do processo. Os analistas de qualidade muitas vezes faziam o papel de analista de requisito, não havia uma definição do que cada um deveria fazer e atuar, causando falta de comunicação e até mesmo retrabalho em todo o time.

- **Comunicação**

A comunicação interna era formal, através de e-mails, não sendo objetiva e clara. A comunicação com o cliente era muito pontual, não havendo muito contato durante o projeto.

- **Treinamento**

Não havia treinamento de processo interno e nem incentivo a melhoraria de capacitação de cada funcionario.

- **Formato de Teste**

Era utilizado um formato preditivo de testes, ao final de toda a construção do software. Esse formato de teste além de gerar muito retrabalho, baixava também a qualidade nas entregas por muitas vezes não ter tempo habil para realização completa dos testes até a data de entrega

- **Ferramentas**

Na gestão não havia uma ferramenta que controlasse o decorrer do processo de entrega do projeto, apenas um cronograma com as datas de finalização no excel, que muitas vezes nao era exposto para todo o time. Na área de desenvolvimento a empresa utilizava as ferramentas necessarias para o desenvolvimento de software que cada cliente necessitava. Na área de qualidade de software, os testadores reportavam os bugs encontrados em planilhas do excel, sem controle centralizado de cada entrega.

3.2 – Proposta de melhoria

Com o levantamento de algumas falhas de comunicação, de processos da empresa mal administrados e as insatisfações da equipe e da empresa como um todo, foi proposto para as partes interessadas a mudança do tipo de abordagem de desenvolvimento da empresa, mudando de preditiva para o ágil, utilizando o *framework Scrum*. O *Scrum* foi escolhido pois os seus processos e práticas permitem a melhoria da comunicação e aumenta a cooperação, mostrando ser eficiente para projetos em ambientes de grandes incertezas.

Com o novo tipo de abordagem ágil, a empresa precisará passar por alguns processos internos de transformações e aprendizagem, onde as partes interessadas estavam dispostas a realizar. Dentre as transformações que será necessária para a nova abordagem ágil acontecer, a

mudança de organograma foi um ponto no qual as próprias partes interessadas solicitaram, já que uma nova estrutura e processo estavam começando.

Com a nova abordagem definida as melhorias a seguir precisam ser implementadas:

- **Treinamento e capacitação**

Para aplicação do *Scrum* ser efetivamente realizada com sucesso, a equipe precisará de treinamentos e estudos para entender tanto da metodologia quanto do *framework Scrum*. Com isso, será realizado um curso com todas as partes envolvidas para o melhor entendimento da abordagem, processos, papeis e responsabilidades dentro do *Scrum*. Além da implementação da melhoria em incentivo de estudos aos profissionais para retirada de novas certificações voltadas as abordagem ageis.

- **Papéis e reponsabilidades**

Com a realização dos novos treinamento e o melhor entendimento da abordagem agil a ser ultilizada, alguns papeis e responsabilidades serão modificados dentro da equipe. Essa alteração só ficará clara através dos treinamentos que serão realizados pelo time. Os papeis de cada um dentro do processo ficará claro e definido com base no perfil de cada funcionário. Com isso, o organograma que já era de interesse ser alterado, passará ter uma definição mais objetiva no processo.

- **Organograma**

Com a nova proposta da utilização do *framework Scrum* e a visão dos perfis de cada funcionário, a equipe teve um novo organograma, sendo separada em 2 times, representado na figura 3.2.1:

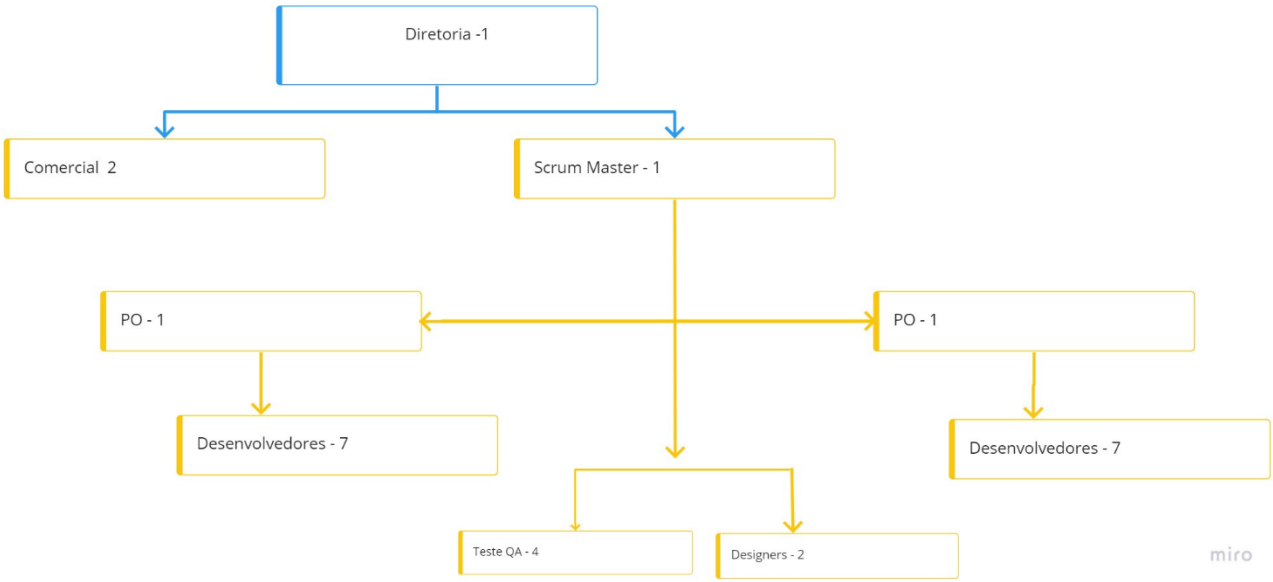

Figura 3.2.1 – Organograma da equipe utilizando *Scrum*

Fonte: Autor

Na imagem acima pode ser percebido que através da capacitação, alguns perfis começaram a ser identificados e alterados para serem melhores explorados na nova abordagem ágil. O gerente de projeto migrou para o papel de *Scrum Master*, os dois analistas de requisitos foram divididos em 2 PO (*Product Owners*), cada um com 7 desenvolvedores, ou seja, a metade. A equipe de qualidade e *designer* estão abaixo do *Scrum master* atendendo o PO de cada time. Com a atualização do novo organograma e a melhor organização dos funcionários, ficou mais fácil dividir a equipe para a nova abordagem ágil, sendo possível aplicar as cerimonias *Scrum* ao cotidiano da equipe.

- **Comunicação**

Com a aplicação do *Scrum* em pequenas equipes há um maior foco a máxima habilidade da equipe em responder rapidamente os problemas, tornando mais prático resolver qualquer tipo de barreira que apareça, uma vez que, todos da equipe tem a transparência e a visão do que acontece no decorrer do projeto deixando a comunicação mais clara e objetiva.

- Método de desenvolvimento

Com o time definido e cada membro com o seu papel, os novos projetos terão uma maior visão das *sprints* necessárias para entregas das histórias previstas. As novas entregas serão feitas com a abordagem ágil. Ao realizar um novo projeto, as separações das tarefas serão realizadas com as suas respectivas regras, as priorizações começarão a ser discutida na equipe, criando um *backlog* das funcionalidades a serem entregues, colocando em prática o que foi aprendido no treinamento de toda a equipe.

No decorrer da nova sprint acontecerá algumas cerimônias responsáveis por garantir a transparência através da comunicação e proporcionar momentos de inspeção e adaptação. As cerimônias aplicadas são: *Sprint Planning*, a *Daily Meeting*, a *Sprint Review* e a *Sprint Retrospective*, essas cerimonias organizarão a Sprint e conduzirão o processo ajudando na melhoria contínua.

Foi proposto inicialmente a *Sprint* ter 2 semanas de duração para que haja uma melhor adaptação da equipe na nova abordagem. As *dailys* são realizadas diariamente com o propósito de inspecionar o trabalho em direção ao atingimento da meta da *sprint*. Com a nova abordagem envolvendo novas cerimonias, uma nova ferramenta foi proposta para que a organização ficasse mais clara e mais transparente para todos da equipe.

- Ferramenta de organização

A ferramenta proposta para organização e realização dos registros das histórias, tarefas e *bugs* encontrados, é o JIRA. Com o JIRA é possível montar e planejar toda a estrutura das equipes envolvidas nos projetos, gerando controle visual, métricas e relatórios. A ferramenta apresenta a visualização de *Kanban* (um cartão de sinalização que indica e acompanha o processo) exibindo o processo de testes com maior evidência, sendo feito em paralelo com o desenvolvimento, obtendo apenas a visão de “concluído” depois de ter passado pela área de qualidade. Na figura 3.2.2 é exibido um exemplo da organização das sprints de um projeto gerenciado pela ferramenta:

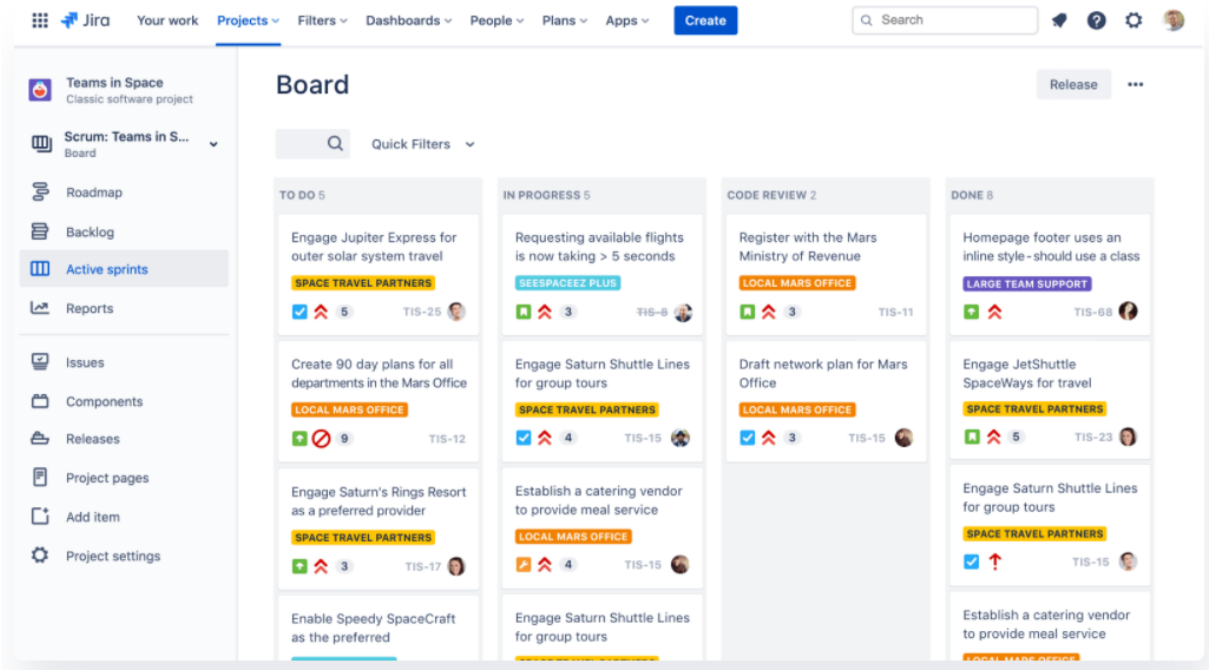

Figura 3.2.2 – *Board* do Jira

Fonte: ATLASSIAN (2021, online)

Para a análise de resultados das atividades, sendo de grande ajuda para o PO e *Scrum Master*, a ferramenta JIRA dispões de vários dashboards de relatórios completos, exibindo as informações por projeto ou *sprints*, conforme a necessidade. Na figura 3.2.3 podemos observar alguns modelos de *dashboard*:

Figura 3.2.3 – Relatórios do Jira

Fonte: TECHTUDO (2021, online)

A utilização da ferramenta JIRA além de organizar, deixa a área de qualidade mais evidenciada, mostrando a sua importância dentro de um projeto de software.

- **Testes de software**

Para essa proposta os testes dos métodos ágeis procuram antecipar a identificação de defeitos em ciclos curtos de desenvolvimento, com *feedback* constante. A mudança de foco da equipe ágil é mais constante, com isso, os testes de regressão são fundamentais para que seja garantido que nada foi impactado. A visibilidade da importância da qualidade fica em evidência quando ela faz parte do processo.

Capítulo 4: Resultados Obtidos

Com a melhoria proposta, a Dllnet implantou o *framework Scrum* em um dos seus principais projetos. Baseado no novo organograma, no treinamento realizado, na definição dos novos papéis, na nova ferramenta implementada, a equipe teve sua primeira experiência com o ágil. Na primeira semana da iniciação do projeto, a equipe fez todas as cerimonias iniciais, dividiram as histórias em tarefas de modo que alcançasse o objetivo da *Sprint*, organizou o que ficaria no *backlog* e utilizou a pontuação dos cards com *planning poker* (uma técnica jogada e baseada em consenso para estimar) para conseguir fazer medição da equipe e então saber a agilidade do time.

***Sprint* 1**

Na *sprint* 1, em relação ao tempo a *Planning* consumiu 4 horas, a *review* 2 horas e a retrospectiva 1 hora e 30 minutos. Realizadas as cerimonias iniciais, a primeira *sprint* do projeto foi um pouco difícil para a equipe, houve vários impedimentos que não foram previstos e não foi planejado incialmente colocar a pontuação do time de QA nas tarefas, o que fez a equipe ter menos agilidade. Como apresentado na figura 4.1 a quantidade de tarefas planejadas dentro da sprint e a quantidade entregue.

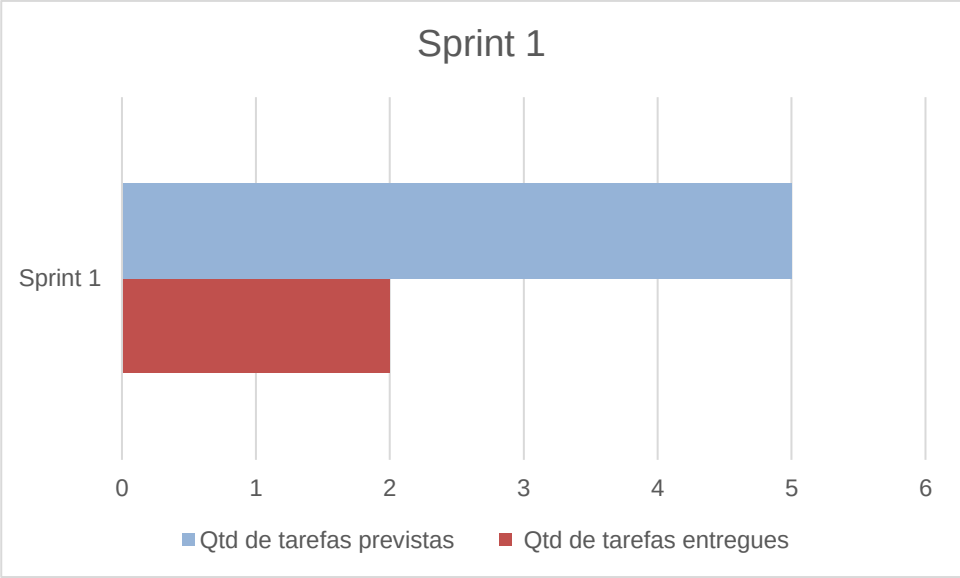

Figura 4.1 – Gráfico de quantidade de tarefas previstas e entregues - *Sprint* 1

Fonte: Autor

O resultado esperado pelo time inicialmente não foi bom, porém a retrospectiva da sprint foi fundamental para identificar os pontos de falha e melhorar. A equipe entrou na sprint 2 com maior engajamento para melhorar o que foi ruim na primeira sprint.

***Sprint* 2**

Na *sprint* 2, algumas lições foram aprendidas e a *planning* foi realizada com mais maturidade e experiencia. O tempo das reuniões foram os mesmos da sprint 1, porém a quantidade de novas tarefas diminuiu pois faltava entregar as tarefas passadas. Com isso, a segunda *sprint* do projeto colocou 1 nova entrega de tarefa e manteve as 3 não entregues anteriormente na *sprint* 2. A pontuação da nova tarefa era bem menor do que as tarefas que vieram da sprint 1, fazendo com que coubesse de uma forma confortável na *sprint*. Na figura 4.2 é possível ver a quantidade de tarefas previstas com as tarefas entregues.

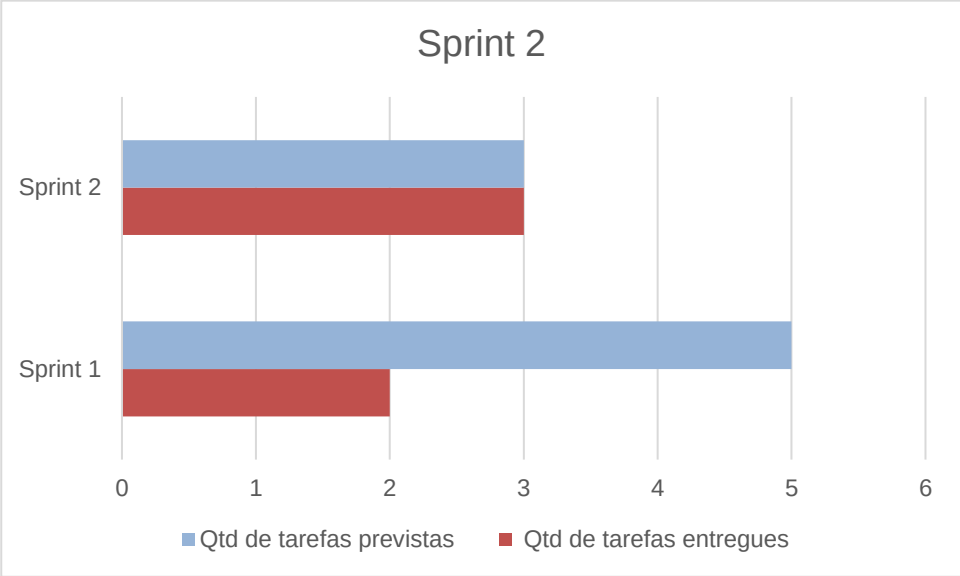

Figura 4.2 – Gráfico de quantidade de tarefas previstas e entregues - *Sprint* 2

Fonte: Autor

Nas reuniões de *review* e retrospectiva foi observado que o resultado da *sprint* foi alcançado, porém, apenas 1 tarefa nova foi de fato entregue na *sprint* e foi visto também que poderiam ter colocado mais tarefas para entrega que caberia no tempo da sprint. Com a próxima *sprint* zerada de débitos anteriores a *sprint* 3 tende a ser mais produtiva e eficaz.

***Sprint* 3**

A *sprint* 3 foi a *sprint* mais produtiva, as suas cerimonias mantiveram-se o mesmo tempo que das *sprints* passadas. A equipe conseguiu avaliar melhor seu tempo de desempenho e então o time puxou para a *sprint*, 6 tarefas devidamente estimadas, ou seja, com todos os times envolvidos determinando seu esforço, inclusive a área de qualidade. As reuniões diárias estavam sendo de grande valia nessa *sprint*, a comunicação foi fundamental para a produtividade da equipe. Na figura 4.3 podemos observar a melhor entrega da equipe.

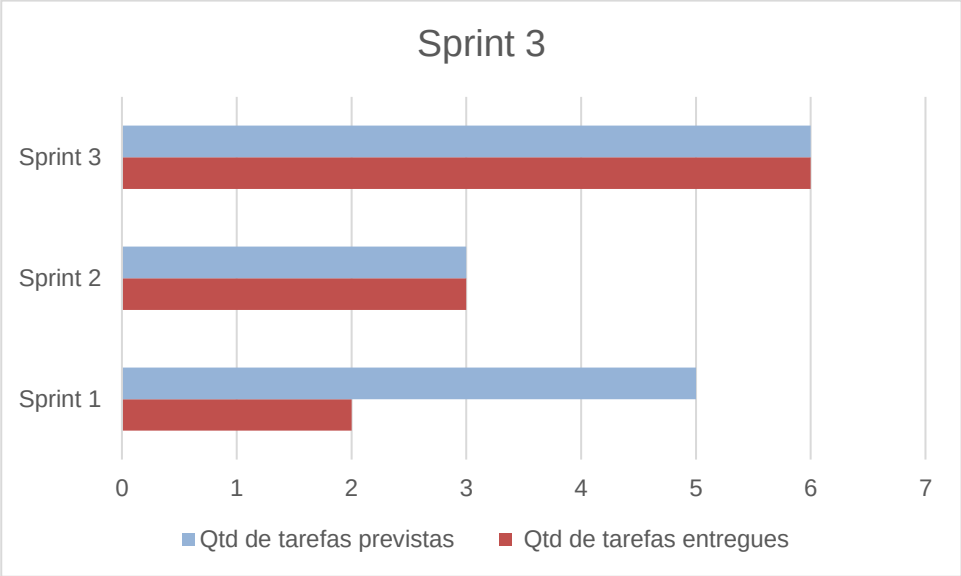

Figura 4.3 – Gráfico de quantidade de tarefas previstas e entregues - *Sprint* 3

Fonte: Autor

Na reunião de *review* e retrospectiva foi observado que a estimativa da equipe começou a melhorar e a equipe mostrou uma grande satisfação na entrega. Com o bom resultado da *sprint* anterior poucos cards ficaram no *backlog*, fazendo com que a sprint 4 fosse mais tranquila para a finalização da entrega.

***Sprint* 4**

Na *sprint* 4 o time estava mais confiante, as cerimônias foram mais rápidas por haver menos entregas restantes, menos itens no *backlog*. A equipe puxou os itens faltantes para a *sprint*, as reuniões diárias aconteciam sem quase nenhum impedimento. Algumas melhorias propostas não conseguiram ser efetivamente realizadas por falta de algumas definições do cliente, o que fez 2 tarefas não ser devidamente entregue. Na figura 4.4 podemos observar melhor a quantidade de tarefa que faltava no *backlog*, com as tarefas entregues.

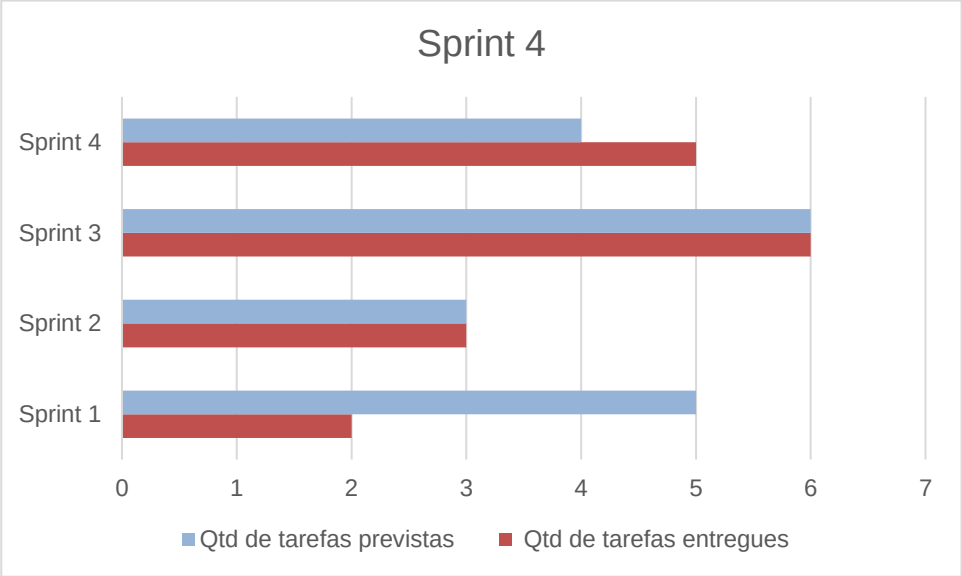

Figura 4.4 – Gráfico de quantidade de tarefas previstas e entregues - *Sprint* 4

Fonte: Autor

- **Melhoria da entrega**

Com a finalização do *backlog* o produto foi entregue e devidamente finalizado. Os resultados obtidos foram extremamente favoráveis tanto para o cliente, quanto para a empresa. O *backlog* do projeto no total teve 18 tarefas previstas e 16 foram entregues, as 2 faltantes não foram priorizas por falta de definição e não impactaram na entrega. Com base nesses dados a figura 4.5 exibe a quantidade em porcentagem de tarefas entregue na final do projeto e o que ficou no *backlog*, como futuras melhorias.

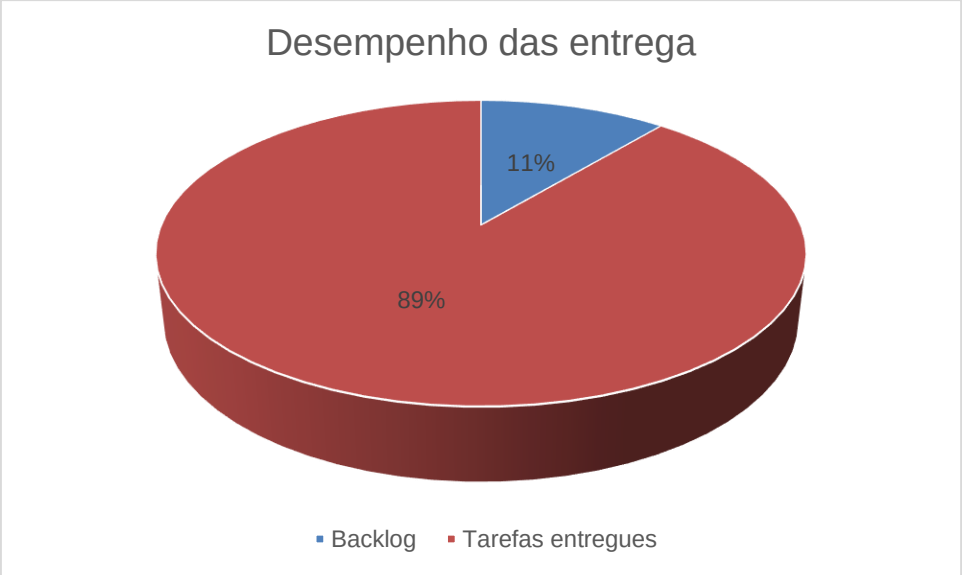

Figura 4.5 – Gráfico em porcentagem do desemprenho da entrega

Fonte: Autor

O projeto no qual foi entregue utilizando *framework* ágil *Scrum*, mostrou uma grande efetividade, sendo um dos melhores projetos já entregues na empresa. O *feedback* do cliente foi muito bom trazendo uma confiança enorme na equipe, e na empresa como um todo.

- **Melhoria da equipe**

Com a aplicação do *framework Scrum*, o alinhamento entre os times se tornou mais fácil de avaliar o que deve ser feito e o que já foi executado, o que evitou retrabalho e consequentemente melhorou o resultado final. A comunicação entre as equipes melhorou, e cada membro envolvido entendeu o seu papel e sua responsabilidade, o que facilitou o andamento do projeto dentro da equipe. O trabalho foi mais harmonioso, respeitando principalmente o limite e a capacidade de cada funcionário atuante no projeto.

A melhoria que ficou evidente internamente e externamente, foi da equipe de qualidade. Com a aplicação do *Scrum* foi realizada a abordagem de teste diferente na área de qualidade. Com a capacitação que os funcionários obtiveram antes da implementação da metodologia, a equipe de qualidade teve seu processo modificado para a nova adaptação do ágil, ocorrendo durante as fases procurando sempre antecipar os problemas, com isso os feedbacks eram constantes.

- **Evolução da empresa**

Após análise dos fatores principais para o projeto (tempo, qualidade dos produtos entregues, satisfação do cliente, satisfação da equipe, satisfação da liderança) é notável a evolução da empresa ao trabalhar no projeto utilizando o *framework Scrum*. Tal aumento destes fatores se dá pela cultura do *Scrum*, no qual foi possível obter uma maior visualização e organização das atividades. Além da organização das atividades, foi percebido também, que o controle das tarefas da equipe se tornou mais fácil de visualizar, observando o bom desempenho do time e trabalhando conforme os objetivos de cada sprint.

Com o desempenho do projeto a empresa se tornou mais confiante nas suas entregas. A comunicação que o *Scrum* trouxe foi de extrema importância para o sucesso. A satisfação da empresa e do cliente com essa entrega tornou o clima da empresa mais leve e sadio, mostrando a importância de todos da equipe, priorizando a transparência e comunicação.

Capítulo 5: Conclusão e Trabalhos Futuros

5.1 – Conclusão

Com surgimento de novas tecnologias, a cobrança mais exigente dos clientes, a velocidade das mudanças e concorrência acirrada, as empresas de desenvolvimento de software buscam ganhar ou manter vantagem competitiva no mercado, serem reconhecidas por serem inovadoras, entregar produtos e serviços de qualidade e manter a satisfação dos clientes. Com a aderência às abordagens ágeis para este contexto de desenvolvimento de *software*, as empresas atingem cada vez mais seus objetivos.

Neste trabalho, foi apresentado um pouco da história da engenharia de software, dos métodos ágeis de desenvolvimento de software e da aplicação, assim como, as abordagens e práticas relacionas aos métodos ágeis, levando em conta alguns dos princípios da Programação Extrema (XP) e o *Scrum*. Neste trabalho foi apresentado também a diferença dos tipos de testes realizados em metodologias preditivas e no método ágil, enfatizando sua importância dentro do processo. Com base nessa metodologia e *framework* (*Scrum*) o estudo de caso da empresa Dllnet TI conseguiu adquirir uma melhoria importante dentro da sua cultura como empresa.

Através de um estudo de caso, este trabalho avaliou a importância de identificar a abordagem de acordo com o contexto do projeto. Foi proposta uma alteração de metodologia de um projeto crítico e de grande risco e impacto financeiro dentro da empresa, no qual o toda a organização da empresa foi alterada e uma abordagem foi adotada para utilização do *Framework Scrum*. Essa alteração foi muito benéfica para a equipe como um todo.

A adoção do *Scrum* foi de grande auxílio para a visibilidade de todas as tarefas que estavam sendo executadas, suas evoluções ou identificação de gargalos. O projeto como um todo, se mostrou muito mais organizado. As tarefas foram mais definidas, explicitadas e observadas. A área de qualidade foi muito mais valorizada, mostrando sua grande importância dentro da organização.

Com uma nova ótica que a abordagem proporcionou, a empresa abriu novos horizontes para mudanças, valorizando mais as pessoas. A mudança de visão sobre o processo de um projeto causou um grande impacto positivo para todo o time. O trabalho foi mais harmonioso, respeitando principalmente o limite e a capacidade de cada funcionário dentro da equipe.

Com todos os benefícios e melhorias citadas, conclui-se que o objetivo desse estudo de caso trouxe resultados que comprovam melhorias do projeto da empresa Dllnet TI. A utilização

do *framework Scrum* cumpriu seus objetivos propostos, conseguindo realizar as entregas dos projetos com qualidade, em curto tempo, atingido o esperado pelo cliente e mostrando satisfação interna e externa. Conclui-se também, que a organização e comunicação são fatores de extrema importância para o sucesso de um projeto. O resultado final trouxe bastante motivação tanto para empresa, quanto para a equipe.

5.2 – Trabalhos Futuros

O estudo de caso apresentado neste trabalho avaliou apenas a utilização do *Scrum* como framework em um único e específico projeto de desenvolvimento de *software* dentro de uma empresa de consultoria. Seria interessante conduzir mais estudos para possível utilização da XP (programação extrema), pois é uma metodologia ágil que se adequa bem aos projetos de *software* com rotineiras alterações. Neste trabalho a metodologia ágil XP, foi bem explicitada com suas práticas e conceitos, deixando evidenciado sua possível eficácia. Existem fatores pessoais que podem influenciar a adoção de um Método Ágil. Novos estudos podem ser conduzidos para investigar como as mudanças culturais trazidas pelos métodos ágeis podem influenciar tais fatores pessoais. Além disso pode ser feito um estudo mais aprofundado do impacto da qualidade dentro da metodologia ágil, já que as diferenças de tipo de testes são bem citadas neste estudo

Referências Bibliográficas

[1] ABRAHAMSSON, Pekka et al. New directions on agile methods: a comparative analysis. In: 25th International Conference on Software Engineering, 2003. **Proceedings**. Ieee, 2003. p. 244-254.

[2] BARESI, Luciano; DI NITTO, Elisabetta; GHEZZI, Carlo. Toward open-world software: Issues and challenges. **Computer**, v. 39, n. 10, p. 36-43, 2006.

[3] BECK, Kent; ANDRES, Cynhia. **Extreme programming explained: embrace change**. 2end ed. Boston, MA, EUA: Addison-Wesley, 2004.

[4] BECK, Kent; HENDRICKSON, Mike; FOWLER, Martin. **Planning extreme programming**. Addison-Wesley Professional, 2001.

[5] BECK, Kent. **Extreme programming explained: embrace change**. addison-wesley professional, 2000.

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

[7] BERTOLINO, Antonia. **Software testing research: Achievements, challenges, dreams. In**: Future of Software Engineering (FOSE'07). IEEE, 2007. p. 85-103.

[8] BOARD. **Atlassian**, 2021. Disponível em: <https://www.atlassian.com/br/software/jira> Acesso em: 29 jun. 2021.

[9] CERVONE, H. Frank. **Understanding agile project management methods using Scrum. OCLC Systems & Services:** International digital library perspectives, 2011.

[10] CONFORTO, Edivandro Carlos. **Gerenciamento ágil de projetos:** proposta e avaliação de método para gestão de escopo e tempo. 2009. Tese de Doutorado. Universidade de São Paulo.

[11] CRISPIN, Lisa; GREGORY, Janet. **Agile testing:** A practical guide for testers and agile teams. Pearson Education, 2009.

[12] EDER, Samuel et al. **Diferenciando as abordagens tradicional e ágil de gerenciamento de projetos**. Production, v. 25, n. 3, p. 482-497, 2015.

[13] HIGHSMITH, James A.; HIGHSMITH, Jim. **Agile software development ecosystems**. Addison-Wesley Professional, 2002.

[14] HIGHSMITH, Jim; COCKBURN, Alistair. **Agile software development**: The business of innovation. Computer, v. 34, n. 9, p. 120-127, 2001.

[15] LINDSTROM, Lowell; JEFFRIES, Ron**. Extreme programming and agile software development methodologies**. In: IS management handbook. Auerbach Publications, 2003. p. 531-550.

[16] MARCHENKO, Artem; ABRAHAMSSON, Pekka. **Scrum in a multiproject environment:** An ethnographically-inspired case study on the adoption challenges. In: Agile 2008 Conference. IEEE, 2008. p. 15-26

[17] MYERS, Glenford J.; SANDLER, Corey; BADGETT, Tom. **The art of software testing**. John Wiley & Sons, 2011.

[18] NAKAGAWA, Elisa Yumi. **Uma contribuição ao projeto arquitetural de ambientes de engenharia de software.** 2006. Tese de Doutorado. Universidade de São Paulo.

[19] NERUR, Sridhar; BALIJEPALLY, VenuGopal. **Theoretical reflections on agile development methodologies**. Communications of the ACM, v. 50, n. 3, p. 79-83, 2007.

[20] NERUR, Sridhar; MAHAPATRA, RadhaKanta; MANGALARAJ, George. **Challenges of migrating to agile methodologies**. Communications of the ACM, v. 48, n. 5, p. 72-78, 2005.

[21] O ciclo de vida do framework extreme programmig (xp). **Semeru**, 2011. Disponível em: <https://www.semeru.com.br/blog/o-ciclo-de-vida-do-framework-extreme-programming-xp/> Acesso em: 09 jun. 2021.

[22] O ciclo de vida do scrum. **Semeru**, 2011. Disponível em: <https://www.semeru.com.br/blog/o-ciclo-de-vida-do-framework-scrum> Acesso em: 09 jun. 2021.

[23] Método Scrum. **Otmza**, 2020. Disponível em <https://www.otmza.com.br/metodo-scrum/> Acesso em: 20 jul. 2021

[24] PAETSCH, Frauke; EBERLEIN, Armin; MAURER, Frank. **Requirements engineering and agile software development**. In: WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. IEEE, 2003. p. 308-313.

[25] Pressman, R. S. **Engenharia de software**. 6 ed. São Paulo: McGraw-Hill, 2006.

[26] Relatorios. **Techtudo**, 2021. Disponível em: <https://www.techtudo.com.br/tudo-sobre/jira.html> Acesso em: 01 jul. 2021.

[27] SATO, Danilo Toshiaki. **Uso eficaz de métricas em métodos ágeis de desenvolvimento de software**. Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, v. 139, 2007.

[28] SCHWABER, Ken. **Agile project management with Scrum**. Microsoft press, 2004.

[29] TIAN, Jeff. **Software quality engineering: testing, quality assurance, and quantifiable improvement**. John Wiley & Sons, 2005.

[30] VICENTE, André Abe. **Definição e gerenciamento de métricas de teste no contexto de métodos ágeis**. 2010. Tese de Doutorado. Universidade de São Paulo.