CENTRO UNIVERSITÁRIO UNIVATES
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE SISTEMAS DE INFORMAÇÃO

IMPLANTAÇÃO DE SCRUM EM UMA EMPRESA DE DESENVOLVIMENTO DE SOFTWARE

Rômulo Henrique Leidemer

Lajeado, Dezembro de 2013

Rômulo Henrique Leidemer

IMPLANTAÇÃO DE SCRUM EM UMA EMPRESA DE DESENVOLVIMENTO DE SOFTWARE

Monografia apresentada ao Centro de Ciências Exatas e Tecnológicas do Centro Universitário UNIVATES, como parte da exigência para a obtenção do título de bacharel em Sistemas de Informação.

Área de concentração: GERÊNCIA DE PROJETOS

Orientador: Prof. Ms. Pablo Dall'Oglio

Lajeado, Dezembro de 2013

Rômulo Henrique Leidemer

IMPLANTAÇÃO DE SCRUM EM UMA EMPRESA DE DESENVOLVIMENTO DE SOFTWARE:

UM ESTUDO DE CASO

Este trabalho foi julgado adequado para a obtenção do título de bacharel em Sistemas de Informação do CETEC e aprovado em sua forma final pelo Orientador e pela Banca Examinadora abaixo:

Prof. Ms. Pablo Dall'Oglio – orientador
Centro Universitário UNIVATES

Prof. Ms. Fabrício Pretto
Centro Universitário UNIVATES

Prof. Ms. Vilson Cristiano Gärtner
Centro Universitário UNIVATES

Lajeado, Dezembro de 2013

DEDICATÓRIA

Dedico este trabalho aos meus pais, em especial pela dedicação e apoio em todos os momentos difíceis.

AGRADECIMENTOS

Aos professores, pela oportunidade de realização de um trabalho focado em minha área de pesquisa, principalmente ao orientador Prof. Ms. Pablo Dall'Oglio, pelo privilégio de sua orientação, pela capacidade profissional e orientação necessária na realização deste trabalho.

Aos colegas do curso, pelo apoio na revisão deste trabalho.

RESUMO

O atual mercado exige que as empresas de desenvolvimento de software tenham rapidez e eficácia na entrega dos projetos. Dessa forma, é necessário que as empresas desenvolvam uma resposta rápida às mudanças que possam vir a ocorrer ao longo de um projeto, forçando-as a adotar meios inovadores para lidar com tais mudanças. As metodologias ágeis surgem como uma alternativa para a exigência do mercado, sendo possível produzir software de maior qualidade que atenda melhor às necessidades dos clientes, com maior rapidez e a um custo menor do que os meios tradicionais. A metodologia ágil Scrum apoia a gestão de projetos no cenário de mudanças constantes, possibilitando, assim, a entrega de produtos com maior rapidez e com maior qualidade. Este trabalho busca implantar a metodologia Scrum em uma pequena empresa de desenvolvimento de software além de documentar as atividades e as etapas de sua implantação, como também descrever as principais dificuldades na implantação do processo, relatar as principais melhorias obtidas por meio de descrição de cenários e entrevistas e comparar os procedimentos de gerência de escopo, tempo e qualidade realizados antes da implantação com os procedimentos realizados após a implantação do Scrum.

Palavras-chave: Metodologia Ágil, Scrum, Gerência de Projetos e Engenharia de Software.

ABSTRACT

The current market requires the companies that develop software to have speed and efficiency in the delivery of projects. It's necessary for companies to develop a quick response to changes that may occur throughout the project, forcing them to adopt innovative ways to deal with these changes. Agile methodologies emerge as an alternative to the requirement of the market, being possible to produce higher quality software that better meets the needs of customers, faster and at a lower cost than traditional ways. The Scrum agile methodology supports the management of projects in the scenario of constant change, which enables the delivery of products faster and with higher quality. This work seeks to implement Scrum in a small software development company and document the deployment activities of Scrum and its stages. Besides describing the main difficulties in the implementation process, as well as major improvements obtained through interviews and description of scenarios and compare the procedures for management of scope, time and quality conducted prior to implementation and post- implementation of Scrum.

Keywords: Agile, Scrum, Project Management and Software Engineering.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

CPM – Critical path method

FDD – Feature Driven Development

LTCAT – Laudo Técnico das Condições Ambientais do Trabalho

PCMAT – Programa de Condições e Meio Ambiente de Trabalho na Indústria da Construção

PCMSO – Programa de Controle Médico de Saúde Ocupacional

PB – Product Backlog

PO – Product Owner

PERT – Program evaluation and review technique

PPRA – Programa de Prevenção de Riscos Ambientais

RAD – Rapid Application Development

ROI – Return on Investment

SESMT – Serviço Especializado em Engenharia de Segurança e em Medicina do Trabalho

SWEBOK – Software Engineering Body of Knowledge

XP – Extreme Programming

SUMÁRIO

1 INTRODUÇÃO

O mercado exige que as empresas de desenvolvimento de software tenham rapidez e eficácia na entrega dos projetos. Dessa forma, é necessário que as empresas desenvolvam uma resposta rápida às mudanças que possam vir a ocorrer ao longo de um projeto, forçando-as a adotar meios inovadores para lidar com tais mudanças. Segundo Valle (2007), as empresas estão vulneráveis às forças de mudanças por questões de mercado, de cultura, de tecnologia, dentre outras, sendo necessária sua rápida adaptação para permanecerem competitivas.

Com o intuito de organizar os processos e fornecer uma estrutura para a construção de software de alta qualidade é que a Engenharia de Software foi criada. A Engenharia de Software é uma aproximação sistemática para a produção de software que considera o custo para o desenvolvimento, cronograma e questões de confiabilidade, assim como as necessidades dos clientes e dos desenvolvedores (SOMMERVILLE, 2011).

Durante muito tempo, tentou-se adotar modelos tradicionais de Engenharia de Software, baseados na previsibilidade dos requisitos, ou seja, a ideia de que os requisitos de um software pudessem ser todos levantados antes da construção do mesmo (DALL'OGLIO, 2006). Essas metodologias tradicionais são geralmente referenciadas como modelo Cascata. Esse modelo utiliza uma abordagem sequencial, ou seja, para que o software possa ser construído, será necessário o levantamento de todos os requisitos, assim como de toda a documentação, para iniciar o processo de construção do software.

Segundo Pressman (2006), poucos projetos reais seguem o que foi documentado. Tratando-se de um modelo linear, qualquer modificação que venha a ocorrer após o início do desenvolvimento irá causar problemas no cronograma. Dessa forma, é sabido que normalmente o cliente não consegue estabelecer explicitamente todos os requisitos no início de um projeto, tornando o desenvolvimento de software para empresas de menor porte muito burocrático e demorado.

Segundo Valle (2007), quanto maior o projeto, maior a tendência de ocorrerem mudanças ao decorrer de seu desenvolvimento e, consequentemente, acarretar o aumento do risco de seu desenvolvimento. Essas mudanças não planejadas acabam ocasionando alterações bruscas no cronograma, assim como um aumento no custo do projeto.

Com a necessidade de acomodar uma série de mudanças ao longo do projeto, é possível utilizar modelos iterativos de desenvolvimento de software para amenizar os impactos negativos que essas mudanças podem trazer ao projeto. Modelos iterativos são caracterizados por pequenos ciclos de desenvolvimento que mesclam diversas atividades de análise, projeto, desenvolvimento e implantação em um pequeno período de tempo chamado de iteração.

Processos iterativos são a base e a origem para os princípios de desenvolvimento ágil de software. Segundo Sbrocco (2012), os conceitos do desenvolvimento ágil foram motivados por uma reação adversa contra os "métodos pesados" de desenvolvimento de software, os quais possuíam um formalismo muito grande na documentação e regulamentações. Com base nessa reflexão, os novos *frameworks* para processos de desenvolvimento de software, que começaram a surgir, foram nomeados inicialmente "métodos leves".

Em 2001, ocorreu uma importante reunião no estado norte-americano de Utah, formalizando a criação dos métodos ágeis para desenvolvimento de software. A reunião foi composta por dezessete profissionais de software com reconhecimento na comunidade de Engenharia de Software. Seu objetivo era discutir formas de melhorar o desempenho de seus projetos. Na reunião, chegaram a um consenso dos princípios básicos utilizados para que um projeto seja bem sucedido. Com base na experiência de todos os membros presentes, redigiram um documento chamado de "manifesto ágil", o qual descreve os princípios acordados na reunião (SCROCCO, 2012).

O manifesto ágil é um documento que encoraja o uso dos melhores métodos de desenvolver sistemas de software, valorizando os itens a seguir (AGILE MANIFESTO, 2013):

- Indivíduos e interações 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.

Muitas empresas de desenvolvimento de software vêm tentando se tornar mais ágeis, devido à necessidade de se manterem sempre à frente dos concorrentes, serem mais suscetíveis a mudanças e também para atender às necessidades do cliente de uma forma rápida e precisa. De acordo com Cohn (2011), equipes ágeis estão produzindo software de maior qualidade que atendem melhor às necessidades dos usuários, com maior rapidez e a um custo menor do que equipes tradicionais.

Fazendo uso de metodologias ágeis, as empresas de software estão conseguindo introduzir, no mercado, produtos com maior satisfação para o cliente. Conseguindo, dessa forma, visualizar melhor o processo de desenvolvimento, o que gera uma maior previsibilidade. Para elas, projetos fora de controle e sem previsão de conclusão se tornaram coisa do passado (COHN, 2011).

Segundo Highsmith e Cockburn (2001), a inovação obtida por essas novas práticas é o reconhecimento de que o ser humano é o principal responsável pela conquista do objetivo do projeto, levando em conta a eficácia e a capacidade de gerenciamento, produzindo uma combinação de valores e princípios responsáveis por definir uma visão do mundo ágil. Sbrocco (2012, p. 91) afirma também que “os métodos ágeis pregam que nenhum processo pode ser equivalente à habilidade dos integrantes da equipe de desenvolvimento. Assim, os processos existentes nos métodos ágeis têm o papel de dar suporte à equipe de desenvolvimento.”.

1.1 Motivação

Processos de desenvolvimento incrementais e iterativos como Scrum e *Extreme Programming* (XP) vêm ganhando espaço, pois visam que a equipe complete, a cada iteração, uma parte funcional do projeto. Cada iteração deve ser pensada para ser curta e com data para entrega definida. O foco é mantido em entregas de código funcionando em um curto espaço de tempo e as equipes não tem tempo para um processo de análise maçante. Isso ocorre, pois essas equipes estão focadas em ter partes funcionais do projeto prontas a cada iteração, aceitando que possam ocorrer erros ao longo do caminho e compreendendo que a melhor forma para identificar esses erros venha a ser construindo o produto (KNIBERG, 2007).

A metodologia Scrum segue os princípios do manifesto ágil, baseando-se em seis características: flexibilidade dos resultados, flexibilidade dos prazos, times pequenos, revisões frequentes, colaboração e orientação a objetos.

Quanto ao Scrum, é importante ressaltar que o processo se concentra em entregas que sejam priorizadas pelo cliente. Assim, cliente e equipe se reúnem, definem as prioridades e a equipe define o que poderá ser entregue em um prazo pré-determinado, o qual pode variar de duas a quatro semanas (SCHWABER, 2004). Assim, a equipe recebe, periodicamente, o *feedback* do cliente referente a cada iteração. Com isso, as mudanças referentes ao escopo não irão surpreender a equipe, pois, com o uso do Scrum, ela está ciente de que possíveis mudanças, no escopo, poderão ocorrer ao longo do projeto.

Como é possível verificar, existem muitos benefícios na utilização de metodologias ágeis em projetos de desenvolvimento de software, principalmente para empresas que necessitem de rapidez para entregar soluções aos seus clientes e também para as que não possuem um processo de desenvolvimento definido ou organizado.

Não possuir um processo de desenvolvimento organizado faz com que as empresas trabalhem de forma desordenada e percam produtividade. Sem um processo bem definido, acabam ocorrendo desgastes internos, principalmente com as pessoas envolvidas diretamente com esses processos. Assim, podem ser citados os seguintes problemas:

- Novas funcionalidades incluídas no produto sem o conhecimento do suporte;
- Data prevista de entrega de *release* inferior a que a equipe de desenvolvimento poderia realmente produzir;

- Falta de *feedback* dos clientes referentes a novas funcionalidades.

A implantação do *Framework Scrum* busca resolver grande parte desses problemas, principalmente, valorizando as pessoas que são a parte mais importante do processo. Sendo assim, é possível, através das reuniões, manter todos informados sobre as novas funcionalidades do produto, receber o *feedback* do cliente sobre as melhorias, assim como mensurar os recursos e o tempo necessário para resolução de problemas ou criação de novas funcionalidades. Isso se deve ao fato do Scrum se focar na melhoria contínua, utilizando-se de ciclos de desenvolvimento, buscando sempre melhorar o processo.

Por mais que o processo seja benéfico, existem diferentes maneiras de implantar o processo Scrum em empresas de desenvolvimento de software. Cada empresa possui um contexto diferente e necessidades específicas. Assim, conhecer o processo de implantação, suas atividades, dificuldades e principais benefícios é muito importante. Dessa forma, o presente trabalho apresentará um estudo de caso sobre a implantação do *Framework* Scrum em uma pequena empresa de desenvolvimento de software, bem como apresentará as etapas da implantação, seus benefícios e principais dificuldades.

A empresa em questão trabalha no desenvolvimento de software para a área de Saúde e Segurança do trabalho e está há mais de seis anos no mercado. Possui nove colaboradores; destes, quatro na área de desenvolvimento de software. Seu principal problema é a necessidade de adaptação às frequentes mudanças de escopo, visto que o software atende à área de segurança e medicina do trabalho, a qual atende a uma legislação que frequentemente vem sendo atualizada, juntamente com novas funcionalidades sugeridas diariamente pelos clientes. Essas sugestões podem demorar cerca de seis meses para serem lançadas devido à falta de organização e à falta de rastreabilidade de mudanças no sistema.

Outro fator negativo é a falta de comunicação entre os setores de suporte e desenvolvimento, pois, muitas vezes, são desenvolvidas novas funcionalidades que são disponibilizadas aos clientes, mas as mesmas não são divulgadas a todos os membros da equipe, somente os membros que fazem parte diretamente do processo tomam conhecimento da melhoria implementada.

1.2 Objetivo

Com base na necessidade das pequenas e médias empresas em tornarem o seu processo de desenvolvimento mais ágil, a proposta do presente trabalho busca estudar e analisar o processo de implantação do Scrum em uma empresa de desenvolvimento de software, avaliando os impactos no gerenciamento de escopo, tempo e qualidade.

Os objetivos específicos são:

- Efetuar levantamento e descrição dos atuais procedimentos de gerência de escopo, tempo e qualidade;
- Implementar o *Framework* Scrum na empresa;
- Documentar as atividades de implantação do Scrum e suas etapas;
- Efetuar levantamento dos procedimentos de gerência de escopo, tempo e qualidade após implantação;
- Descrever as principais dificuldades na implantação do processo, bem como as principais melhorias obtidas por meio de descrição de cenários e entrevistas.

1.3 Organização do Trabalho

Este trabalho está organizado em cinco capítulos, conforme descrito a seguir:

- **Capítulo 1 – Introdução:** é feita uma breve descrição sobre este trabalho, contemplando objetivos, motivações, justificativas e a sua estrutura.
- **Capítulo 2 – Fundamentação Teórica:** nele é apresentado o referencial teórico deste trabalho.
- **Capítulo 3 – Metodologia:** é a definição da metodologia de pesquisa, público alvo, como se deu a coleta dos dados, a limitação da pesquisa e as etapas a serem desenvolvidas.
- **Capítulo 4 – Estudo de Caso:** o capítulo de estudo de caso apresenta a análise e discussão dos dados colhidos durante o processo de pesquisa deste trabalho. Estes resultados são apresentados de forma qualitativa.
- **Capítulo 5 – Conclusão:** no último capítulo do trabalho, são apresentadas as conclusões. Retomam-se os principais resultados da discussão do estudo de caso até o momento e são analisados os resultados obtidos.

2 REFERENCIAL TEÓRICO

Neste capítulo serão abordados os temas relevantes ao estudo proposto e que servem de embasamento teórico para o estudo de caso realizado, apresentando os conceitos sobre Engenharia de Software, gerenciamento de projetos, processos de software, metodologias ágeis e Scrum, de forma a situar o leitor nos conceitos relativos aos mesmos.

2.1 Engenharia de Software

O software de computador, nos dias atuais, é a única e indispensável tecnologia para negócios, ciência e engenharia em todo o mundo. Embutido em sistemas de todas as espécies como transporte, medicina, telecomunicações, militar, industrial, entretenimento, dentro outros. Os produtos de software afetam praticamente todos os aspectos de nossas vidas, pois estão difundidos no nosso comércio, na nossa cultura, praticamente, em todas as atividades que realizamos diariamente (PRESSMAN, 2006).

Pressman (2006) define Software da seguinte forma:

> Software de computador é o produto que os profissionais de software constroem e, depois, mantêm ao longo do tempo. Abrange programas que executem em computadores de qualquer tamanho e arquitetura, conteúdo que é apresentado ao programa a ser executado e documentos tanto em forma impressa quanto virtual que combinam todas as formas de mídia eletrônica (PRESSMAN, 2006, p. 1).

Pressman (2006) também afirma que o software é um elemento do sistema lógico e não de um sistema físico, sendo assim, possui formas diferentes de ser produzido em relação à

maioria das coisas que os seres humanos produzem. Para explicar essas diferenças, o autor compara software e hardware e examina as três características principais entre eles (PRESSMAN, 2006, p. 4-6):

- O software é projetado e então desenvolvido, não possuindo um processo de manufatura e, por esse motivo, os custos para o desenvolvimento são concentrados no processo de engenharia;
- O software não "se desgasta", ou seja, não sofre influência dos males ambientais. As falhas do software são resultado de deficiência no processo de engenharia;
- Quanto à produção do software, atualmente o mesmo vem sendo, em grande parte, desenvolvido de forma manufaturada.

Segundo Sommerville (2011), a Engenharia de Software é a disciplina da engenharia responsável por cuidar de todos os aspectos da produção de software, desde os estágios iniciais até à manutenção do sistema após a implantação.

A IEEE define a Engenharia de Software como a aplicação de uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, a operação, a manutenção de software e o estudo dos mesmos (IEEE, 1993).

A Engenharia de Software é necessária, pois fornece controle e organização para uma atividade que, do contrário, tornar-se-ia caótica. Porém, uma abordagem moderna de Engenharia de Software precisa ser "ágil", exigir apenas as atividades, controles e documentação adequada à equipe do projeto e ao produto a ser produzido (PRESSMAN, 2006). Ou seja, evitar a produção de artefatos que não venham a ser utilizados pela equipe.

Dessa forma, compreende-se que a Engenharia de Software é uma tecnologia composta por camadas, com foco na qualidade, conforme a Figura 1. Apoia-se em um compromisso organizacional com a qualidade, sempre buscando abordagens mais efetivas, ou seja, a melhoria contínua (PRESSMAN, 2006).

Figura 1 - Engenharia de softwares em camada

Fonte: PRESSMAN (2006, p.17).

Ainda de acordo com a Figura 1, cada etapa é responsável por um conjunto de ações relacionadas que resultam em um produto final. Cada ação é indispensável para a construção do software, pois fornece as técnicas necessárias para a realização do mesmo, contemplando um conjunto de tarefas que incluem comunicação, análise de requisitos, modelagem de projeto, construção de programas, teste e manutenção.

A primeira camada foca na qualidade. A segunda camada - alicerce da engenharia de software - é a camada de processos, a qual forma a base para o controle gerencial de projetos de software. A terceira camada é responsável por guiar o processo de desenvolvimento. Ela é a camada de métodos, a qual fornece a definição de "como fazer" no processo de desenvolvimento de software. Por último, - responsável por apoiar os processos e métodos - a camada de ferramentas fornece o apoio automatizado ou semiautomatizado para os processos e métodos.

Devido ao fato da engenharia de software ser muito abrangente, sob o patrocínio da IEEE e com o objetivo de criar uma referência para os assuntos relacionados à engenharia de software foi criado o *Software Engineering Body of Knowledge* (SWEBOK) (SBROCCO, 2012). Segundo o SWEBOK (2013), o livro é um guia de uso e aplicação das melhores práticas de Engenharia de Software. Ele foi desenvolvido com conhecimentos de quatro décadas e foi revisado por inúmeros profissionais de engenharia de software de diversos países. Seu principal objetivo foi estabelecer um conjunto apropriado de critérios e normas para a prática profissional da Engenharia de Software.

De acordo com Sbrocco (2012), no SWEBOK a engenharia de software é divida em dez áreas:

- Gerência de engenharia de software: Consiste nas boas práticas para a gerência de desenvolvimento de software;

- Gerência de configuração de software: Identifica e controla sistematicamente as configurações do sistema de modo a manter a integridade e a rastreabilidade durante o ciclo de vida do sistema;
- Processos de engenharia de software: define, implementa, avalia, mensura, gerencia as mudanças e melhorias do processo de ciclo de vida de software;
- Ferramentas e métodos: Pesquisar ferramentas e métodos com objetivo de aumentar a produtividade dos desenvolvedores e reduzir a ocorrência de falhas no desenvolvimento;
- Qualidade de software: Estabelecer métodos para a construção de produtos com qualidade dentro do prazo estipulado;
- Requisitos de software: Boas práticas para o levantamento, análise, especificação e gestão dos requisitos de software;
- Design de software: Tradução dos requisitos para uma descrição com objetivo de explicar a resolução do problema;
- Construção de software: Implementação do software por meio de codificação, verificação, testes de unidade e testes de integração;
- Teste de software: É um elemento utilizado para garantir a qualidade do software. Devem ser realizados testes para verificar se os requisitos foram implementados de forma correta;
- Manutenção de software: Manutenções para garantir o funcionamento correto do software que podem ocorrer antes ou após a entrega do software.

2.2 Gerência de Projetos

Segundo Valle (2007), análises históricas apontam para o fato de o conceito de gerenciamento de projetos ser empregado na humanidade, há muito tempo. Segundo historiadores, eram empregados, no antigo Egito, técnicas de engenharia e gerenciamento bem sofisticados para a construção de esgotos, irrigações, embarcações e canais. Além da construção das próprias pirâmides serem um grande esforço de gerenciamento de projetos devido ao emprego de enormes matérias e recursos humanos, sendo utilizados 100 mil trabalhadores em 30 anos para sua construção. O autor também cita a muralha da China, o Coliseu e o Parthenon como exemplos de outras obras que utilizaram os conceitos de gerência

de projetos, pelo fato do tamanho das construções, da grande quantidade de pessoas e dos recursos envolvidos para a construção dessas obras.

Com o lançamento do satélite Sputnik, pela União Soviética, no auge da guerra fria, foi utilizado, pela primeira vez, o conceito isolado de gerência de projetos. Quando, após serem surpreendidos pela União Soviética, os Estados Unidos decidiram investir no desenvolvimento de novas técnicas e ferramentas com o objetivo de acelerar a implementação de projetos militares. Esse esforço deu origem ao *program evaluation and review technique* – (PERT), utilizado para a construção do míssil nuclear Polaris (VALLE, 2007).

Pouco tempo depois, foi desenvolvida, pela DuPont, uma técnica similar chamada de *critical path method* (CPM). Nesse mesmo período, Drucker popularizou o termo gerenciamento por objetivo dentre as grandes organizações. Com essa técnica, o corpo diretivo e os funcionários concordam em objetivos comuns e estabelecem prazos, métricas e o modo para atingi-los. O conceito de gerenciamento por objetivos influenciou significativamente na formulação da teoria de gerenciamento de projetos (VALLE, 2007).

2.2.1 Definição de projeto

Kerzner (2006, p.15) define projeto como "um empreendimento com objetivo bem definido, que consome recursos e opera sob pressões de prazos, custos e qualidade. Além disso, projetos são, em geral, consideradas atividades exclusivas em uma empresa." Já, segundo o PMBoK (PMI, 2008, p.11), projeto é definido como "um esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo." Sbrocco (2012), por sua vez, define o projeto como sendo um empreendimento com características próprias, que possui início e fim, conduzido por pessoas, com o objetivo de atingir metas estabelecidas dentro de parâmetros de prazo, custo e qualidade.

Quanto ao objetivo do projeto, o mesmo deve ser redigido com o máximo de cuidado e deve conter a ação, que vem a ser um verbo no infinitivo, iniciando a declaração do objetivo. Deve conter, também, o objeto, item sobre o qual a ação se exerce ou resulta. E, por fim, deve conter os requisitos, restrições ou condições complementares de desempenho, de tempo, de local, de qualidade, dentre outras (SBROCCO, 2012).

Segundo PMBok (PMI, 2008), como resultado, um projeto pode criar um produto final ou um produto componente de outro item, ou a capacidade de criar um serviço ou um resultado.

2.2.2 Conceituação e características da gerência de projetos

O gerenciamento de projetos busca aprender o máximo possível com as falhas das outras pessoas para tentar influenciar o projeto, positivamente, no futuro. É possível notar, com base no conhecimento de projetos acumulados do passado até hoje, que, por mais diferentes que eles sejam, possuem características em comum, como aprendizado por meio de erros, temporariedade, singularidade e progressividade dos projetos (VALLE, 2007).

A gerência de projetos busca, através das falhas do passado, evitar que esses mesmos erros ocorram na execução de um novo projeto. Todo o projeto é finito, tendo início, meio e fim e somente será finalizado ao atingir seus objetivos, ou quando não forem mais necessários, ou caso não seja possível atingir esses objetivos. Os objetivos de um projeto geralmente tendem a gerar produtos, serviços singulares de forma progressiva, utilizando-se de etapas incrementais.

De acordo com o PMBoK, o gerenciamento de projetos é a aplicação de conhecimento, habilidades, ferramentas e técnicas às atividades do projeto, a fim de atender aos seus requisitos (PMI, 2008). Já, Valle (2007) descreve a gerência de projetos como:

> A aplicação de conhecimento, habilidades, ferramentas e técnicas às atividades do projeto, a fim de atender às suas demandas, sendo realizado por meio de integração dos seguintes processos: iniciação, planejamento, execução, monitoramento e controle, e encerramento (VALLE,2007, p.35).

O gerenciamento de projetos, segundo Valle (2007), é realizado por uma pessoa responsável e denominado de gerente de projetos. O gerente de projetos deve gerenciar a aplicação de conhecimento, habilidades, ferramentas e técnicas em prol do objetivo do projeto. São suas principais atribuições:

- A identificação das necessidades do projeto;
- O estabelecimento de objetivos claros e palpáveis;
- O atendimento às expectativas de todas as partes interessadas;
- O devido balanceamento entre qualidade, escopo, tempo e custo, é realizado obedecendo-se à chamada teoria da tripla restrição.

De acordo com Valle (2007) o balanceamento entre os quatro fatores conflitantes, denominada tripla restrição, de acordo com a visão adotada pelo gerente de projetos, irá definir o balanceamento. Ou seja, se o gerente definir investir no tempo, custo e escopo, à consequência será a qualidade do projeto. Já se ele definir tempo, custo e qualidade, a consequência será o escopo.

Já, segundo o PMBoK (PMI, 2008), o gerenciamento de projetos inclui o balanceamento das restrições conflitantes do projeto como: escopo, qualidade, cronograma, orçamento, recursos e risco. As restrições não se limitam às restrições citadas, e a relação entre elas faz com que, se alguma delas mudar, pelo menos um dos outros itens será alterado.

Valle (2007) conclui que projetos bem-sucedidos são aqueles que atingem seus objetivos, entregam o produto dentro do escopo, prazo, orçamento e com qualidade.

Com o objetivo de fornecer diretrizes para o gerenciamento de projetos foi desenvolvido o Guia *PMBoK*, o qual consiste em uma norma reconhecida para o gerenciamento de projetos. Ou seja, é um documento formal responsável por descrever as normas, métodos, processos e práticas estabelecidas (PMI, 2008).

2.2.3 Ciclo de vida e processos de gerenciamento

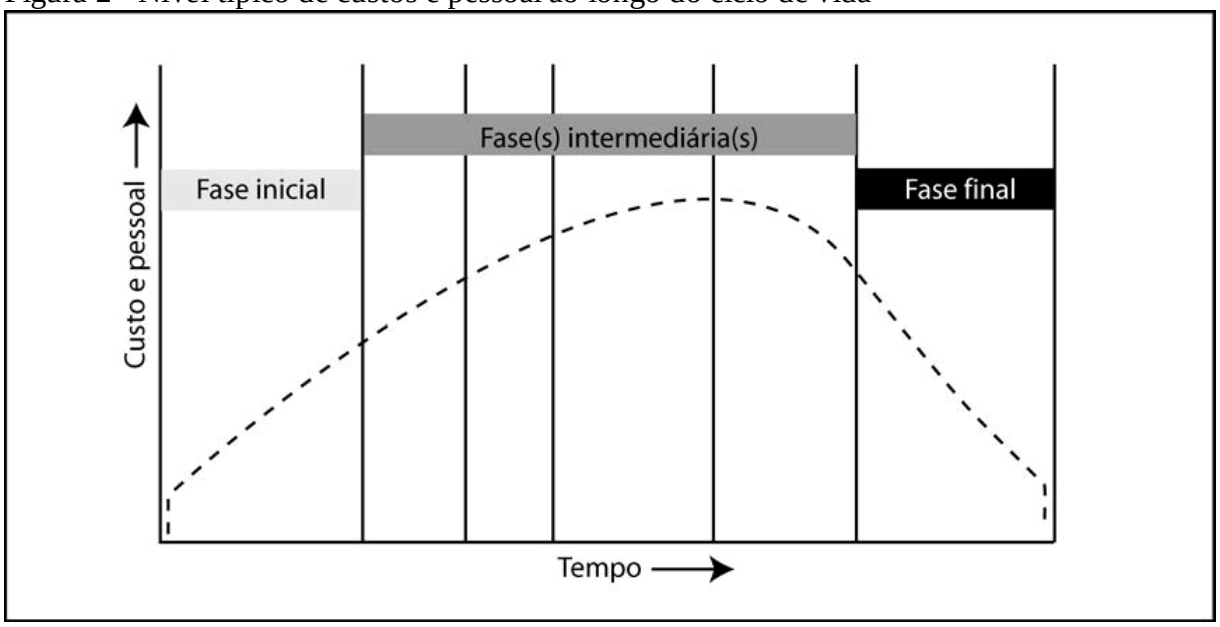
Como um projeto consiste em esforço temporário finito, o qual possui início e fim, a melhor forma para gerenciá-lo é através do uso do ciclo de vida proposto pelo PMBoK. Segundo PMBoK (PMI, 2008), o ciclo de vida consiste em um conjunto de fases sequenciais que o projeto terá de executar ao longo de sua vida. Ele pode ser adaptado de acordo com as necessidades do projeto e oferece uma estrutura básica para o gerenciamento.

Independente do tamanho e nível de complexidade, o PMBoK oferece um ciclo de vida genérico que pode ser adaptado a qualquer situação. A Figura 2 apresenta o ciclo completo de um projeto. Na figura, podemos identificar as fases de início do projeto, a fase intermediária (composta pelas fases de organização e preparação e de execução do trabalho do projeto) e a fase final (fase de encerramento) (PMI, 2008).

O PMBoK (PMI, 2008), define três características principais para o ciclo de vida genérico do projeto. Sendo o primeiro, visível na Figura 2, onde o nível de custo e o nível

pessoal são mínimos na fase inicial do projeto, atingem o valor máximo enquanto o projeto é executado e voltam a cair na fase final do projeto.

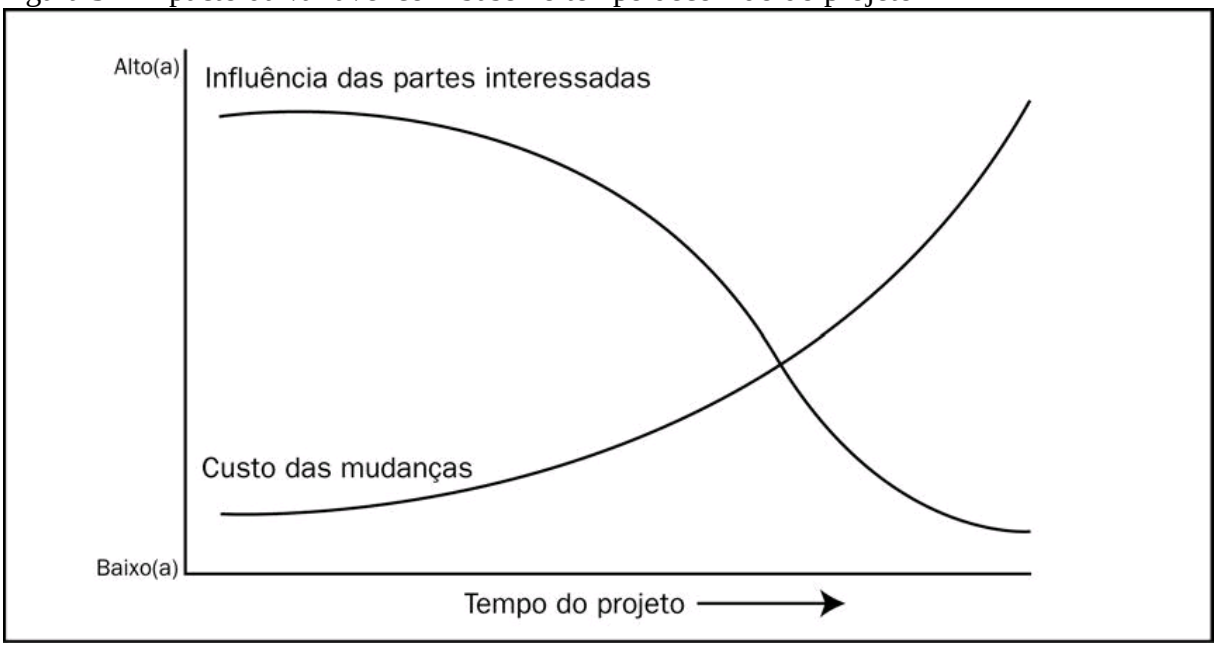
Figura 2 - Nível típico de custos e pessoal ao longo do ciclo de vida



Fonte: PMI (2008, p.22).

A segunda, visível na Figura 3, demonstra que, no início do projeto, a influência das partes interessadas, riscos e incertezas é alta e, conforme chega próximo ao término, começa a decair. Já, na terceira, também visível na Figura 3, é possível identificar que a capacidade de influenciar as características do produto no início do projeto é maior e torna-se menor ao decorrer do projeto. Isso se deve ao fato do custo com as mudanças e correções de erros, no decorrer do projeto, tornar-se mais caro próximo ao término projeto.

Figura 3 - Impacto da variável com base no tempo decorrido do projeto



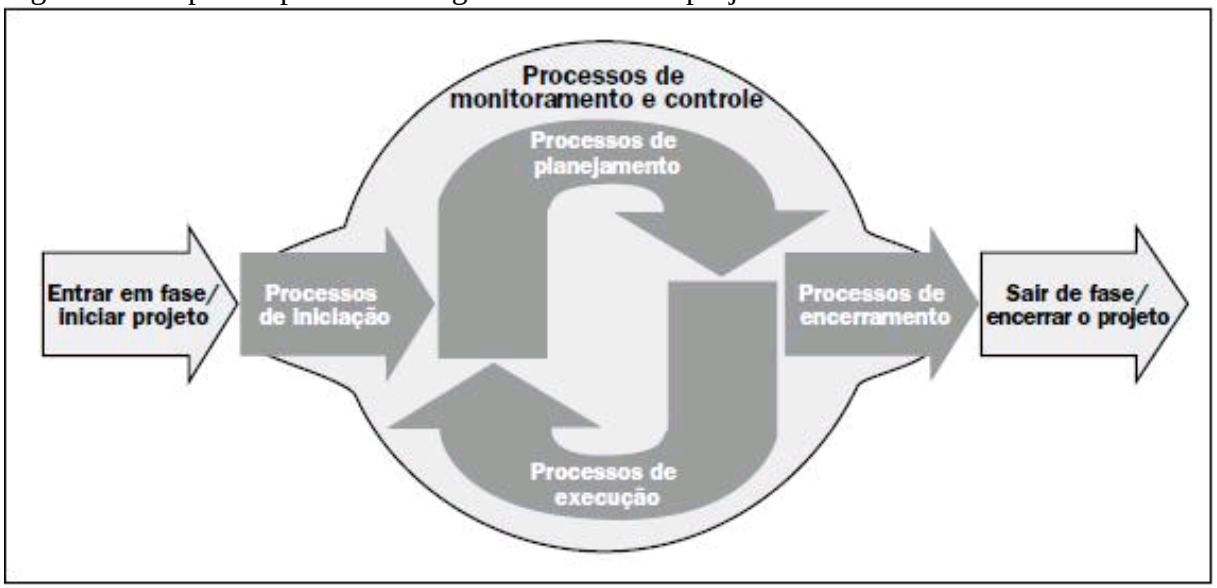
Fonte: PMI (2008, p.22).

Os processos de gerenciamento de projetos, segundo PMBoK (PMI, 2008), são agrupados em cinco categorias. Essas cinco categorias podem ser visualizadas na Figura 4 e são descritas a seguir:

1. Iniciação: reconhecem que um projeto ou fase deve ser iniciado e se comprometem com a sua execução;
2. Planejamento: planejar e manter uma estratégia de trabalho visando a atingir os objetivos de negócio que determinam a existência do projeto;
3. Execução: coordenar pessoas e recursos com o objetivo de cumprir o planejado;
4. Controle: assegurar, através da monitoração e avaliação do progresso do projeto, que os objetivos estão sendo atingidos e, caso necessário, tomar ações corretivas;
5. Finalização: finalizar e aceitar o projeto ou fase e realizar o seu encerramento.

Conforme a Figura 4, pode-se realizar um mapeamento desses grupos de processo propostos pelo PMBoK com o ciclo PDCA (*Plan*, *Do*, *Check*, *Act*), o que corresponde a um ciclo de melhoria contínua.

Figura 4 - Grupos de processos de gerenciamento de projetos



Fonte: PMI (2008, p.40).

2.2.4 Áreas de conhecimento e processos propostos pelo PMBoK

Segundo Sbrocco (2012), o PMBok é organizado em nove áreas de conhecimento e cada uma é descrita por um processo. A não execução de um desses processos afetará negativamente um projeto, pois o projeto é um esforço integrado por todas as áreas. As áreas que compõem o modelo serão descritas a seguir.

- Gerência de qualidade do projeto: descreve os processos necessários para assegurar que os objetivos do projeto sejam atendidos;
- Gerência de recursos humanos: descreve os processos necessários para garantir que as pessoas e recursos envolvidos no projeto sejam utilizados da melhor forma possível;
- Gerência de escopo do projeto: descreve os processos necessários para garantir que o projeto contemple todo o trabalho necessário para que seja concluído;
- Gerência das aquisições do projeto: descreve os processos necessários para a compra de mercadorias e serviços externos ao projeto;
- Gerência da integração do projeto: descreve os processos e atividades necessárias para coordenar adequadamente diversos elementos do projeto;
- Gerência de comunicações do projeto: descreve os processos necessários para garantir a geração, a captura, a distribuição, o armazenamento e a apresentação final das informações do projeto de forma adequada e pontual;

- Gerência do custo do projeto: descreve os processos necessários para garantir o término do projeto dentro do orçamento aprovado;
- Gerência de riscos do projeto: descreve os processos necessários para garantir a identificação, análise e resposta aos riscos com o objetivo de prevenir ou antecipar o riscos do projeto;
- Gerência do tempo do projeto: descreve os processos necessários para garantir que o projeto seja concluído no tempo correto.

Neste trabalho, será dada ênfase para as áreas de Gerência de escopo, Gerência de tempo e Gerência de Qualidade. As quais serão aprofundadas nos próximos parágrafos.

2.2.4.1 Gestão de escopo

O gerenciamento de escopo é o processo necessário para assegurar que o projeto irá contemplar o requerido e para assegurar o seu sucesso (SBROCCO, 2012, p. 91).

O PMBoK (PMI, 2008, p.113), define a gestão de escopo em cinco processos, são eles:

- Coletar os requisitos: o processo de definição e documentação das necessidades das partes interessadas para alcançar os objetivos do projeto;
- Definir o escopo: o processo de desenvolvimento de uma descrição detalhada do projeto e do produto;
- Criar a EAP: o processo de subdivisão das entregas e do trabalho do projeto em componentes menores e mais facilmente gerenciáveis;
- Verificar o escopo: o processo de formalização da aceitação das entregas terminadas do projeto;
- Controlar o escopo: o processo de monitoramento do progresso do escopo do projeto e escopo do produto e gerenciamento das mudanças feitas na linha de base do escopo.

2.2.4.2 Gestão de tempo

O gerenciamento de tempo, também como citado anteriormente, mapeia os processos necessários para que o projeto seja finalizado dentro do prazo (SBROCCO, 2012).

O PMBoK (PMI, 2008, p.113), define a gestão de tempo em seis processos, são eles:

- Definir as atividades: o processo de identificação das ações específicas a serem realizadas para produzir as entregas do projeto;
- Sequenciar as atividades: o processo de identificação e documentação dos relacionamentos entre as atividades do projeto;
- Estimar os recursos da atividade: o processo de estimativa dos tipos e quantidades de material, pessoas, equipamentos ou suprimentos que serão necessários para realizar cada atividade;
- Estimar as durações da atividade: o processo de estimativa do número de períodos de trabalho que serão necessários para terminar atividades específicas com os recursos estimados;
- Desenvolver o cronograma: o processo de análise das sequências das atividades, suas durações, recursos necessários e restrições do cronograma visando a criar o cronograma do projeto;
- Controlar o cronograma: o processo de monitoramento do andamento do projeto para atualização do seu progresso e gerenciamento das mudanças feitas na linha de base do cronograma.

2.2.4.3 Gestão de qualidade

A gerência de qualidade é um processo necessário para garantir que as necessidades que deram origem ao projeto sejam cumpridas corretamente (SBROCCO, 2012).

O PMBoK (PMI, 2008, p.159), define a gestão de qualidade em três processos, são eles:

- Planejar a qualidade: o processo de identificar os requisitos e/ou padrões de qualidade do projeto e do produto, bem como documentar de que modo o projeto demonstrará a conformidade;

- Realizar a garantia da qualidade: o processo de auditoria dos requisitos de qualidade e dos resultados das medições de controle de qualidade para garantir que sejam usados os padrões de qualidade e as definições operacionais apropriadas;
- Realizar o controle da qualidade: o processo de monitoramento e registro dos resultados da execução das atividades de qualidade para avaliar o desempenho e recomendar as mudanças necessárias.

2.3 Processos de software

Os modelos prescritivos de processos foram criados com o objetivo de fornecer estabilidade, controle e organização ao desenvolvimento de software que até então era um processo caótico. Fornece um roteiro razoavelmente efetivo para as equipes (PRESSMAN, 2006).

Pressman (2006, p.37) define modelos prescritivos de processo como "Um conjunto distinto de atividades, ações, tarefas, marcos e produtos de trabalho que são necessários para fazer a Engenharia de Software com alta qualidade". Sendo que esses modelos genéricos não são perfeitos, fornecem um roteiro e o fluxo a ser seguido para a equipe que irá desenvolver o projeto.

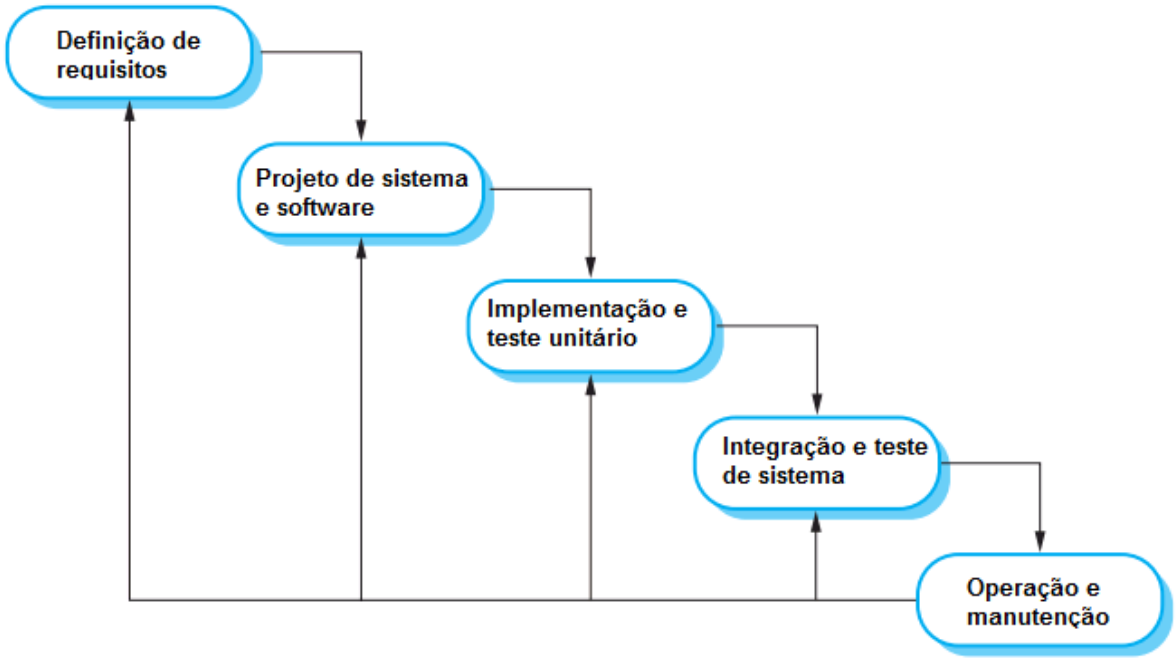
Podemos citar os seguintes modelos prescritivos: o modelo em cascata, o modelo iterativo e o modelo em espiral. Os três modelos citados serão descritos nesta seção pelo fato de serem os principais, quando nos referimos a métodos clássicos de desenvolvimento de software. Podemos ainda citar o modelo RAD (Rapid Application Development) e o modelo de desenvolvimento concorrente, porém esses não serão descritos nesta seção.

2.3.1 Modelo em cascata

O modelo cascata, também conhecido como modelo clássico, foi o primeiro processo de desenvolvimento de software publicado. É considerado um modelo de fácil entendimento, pois se baseia em uma sequência de etapas que devem ser seguidas. Cada etapa resulta em um documento e, para dar sequência à próxima etapa, o documento da anterior deve ser aprovado (PRESSMAN, 2006). É um processo dirigido a planos, pois as atividades do processo devem

ser planejadas antes do início dos trabalhos (SOMMERVILLE, 2011). A Figura 5 apresenta a visão das etapas do modelo em cascata, o formato desta figura deu origem ao nome do modelo “Modelo em cascata”.

Figura 5 – Modelo em cascata



Fonte: SOMMERVILLE(2011, p.20).

As principais etapas do modelo em cascata são análise e definição de requisitos, projeto de sistema e software, a implementação de teste unitário e a operação e manutenção. A definição de requisitos é responsável por definir as metas do sistema, definir os detalhes e funcionamento do sistema por meio de entrevista com usuários. No projeto de software, é definida a arquitetura do mesmo. Na implementação e teste unitário, o sistema é realmente desenvolvido e são criados os testes unitários necessários. Na integração e teste de sistema, os módulos do sistema são integrados e testados como um todo. Já, a operação e manutenção, que normalmente tende a ser o estágio mais longo, envolve a correção de erros que não foram descobertos em etapas anteriores e também a descoberta de novos requisitos (SOMMERVILLE, 2011).

Segundo Sbrocco (2012), o modelo em cascata é um dos modelos mais utilizados na engenharia de software e também é um dos mais criticados. Ainda segundo o autor, há alguns problemas com esse modelo. Um dos problemas observados é que dificilmente um projeto irá seguir o fluxo, devido ao fato do desenvolvimento de software sofrer várias mudanças de escopo ao decorrer de sua vida. Outro fato é do cliente, inicialmente, não conseguir

especificar todos os requisitos que serão necessários ao projeto, os quais somente virão à tona no decorrer do mesmo. Em virtude da necessidade da produção seguir o fluxo, o cliente somente verá o produto no final do processo, dessa forma, os erros ou mal-entendidos deverão ser detectados no início do projeto, caso contrário, o resultado final poderá ser desastroso.

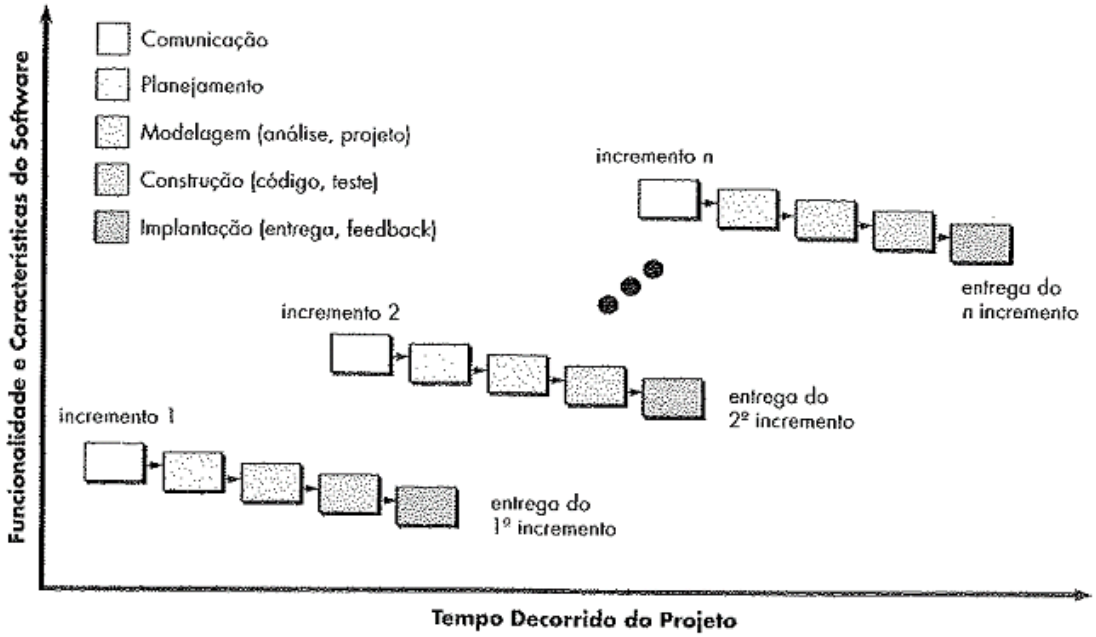
Segundo Pressman (2006), em uma análise realizada em projetos reais pela Bradac (1994) foi descoberto que o uso do "modelo em cascata" leva a estados de bloqueio, pois os membros do projeto, em vários momentos, precisarão aguardar outros membros concluírem as tarefas dependentes. Esse período de bloqueio ocorre normalmente no início e no fim do projeto, causando uma perda considerável de produtividade aos membros da equipe.

Ainda, segundo Pressman (2006), o modelo em cascata não é adequado aos dias atuais devido ao mercado de desenvolvimento de software atuar em um ritmo acelerado devido à grande quantidade de modificações que tende a ocorrer no projeto. Porém, ele pode ser útil em projetos em que os requisitos são fixos e o trabalho prossiga de forma linear.

2.3.2 Modelo incremental

O modelo incremental utiliza-se do modelo em cascata, porém o aplica de forma iterativa. Como mostra a Figura 6, as sequências do modelo em cascata são utilizadas de forma racional com o passar do tempo. Dessa forma, cada sequência produz um incremento do produto que pode ser entregue ao cliente. Por exemplo, ao desenvolver um editor de textos, utilizando a forma incremental, a primeira entrega poderia ser o produto com a capacidade básica de edição de textos. A segunda entrega seria um incremento com a capacidade de edição avançada do texto e a terceira poderia incorporar um corretor ortográfico (PRESSMAN, 2006).

Figura 6 - Modelo incremental



Fonte: PRESSMAN (2006, p.40).

Sommerville (2011, pg. 21) define o modelo incremental como "[...] baseado na ideia de desenvolver uma implementação inicial, expô-la ao usuário e continuar por meio de várias versões até que o sistema adequado seja desenvolvido.". Sommerville (2011) ainda afirma que o modelo incremental é parte fundamental das metodologias ágeis e, para os sistemas de negócios, *e-commerce* e sistemas pessoais, ele é melhor que o modelo em cascata. Ele reflete a maneira como resolvemos os problemas, pois resolvemos em partes e não de uma forma única. Além de o desenvolvimento incremental ser mais barato e garantir que mudanças possam ser realizadas no decorrer de seu desenvolvimento.

O primeiro incremento, quando o modelo incremental é utilizado, é denominado de núcleo do produto. Nele são abordados os requisitos básicos do sistema, deixando de lado os requisitos suplementares, ou seja, focando-se apenas no principal. Esse núcleo passará por uma revisão detalhada do cliente. Com base na avaliação do cliente, o próximo plano será desenvolvido com o intuito de melhorar o núcleo com as observações do cliente, satisfazendo as necessidades do mesmo. Esse processo é repetitivo, sendo realizado antes de cada incremento, até que o cliente esteja satisfeito com o produto final (PRESSMAN, 2006).

O principal objetivo do modelo incremental é de apresentar um produto operacional ao final de cada incremento. Esse produto operacional pode ou não ser entregue para o cliente para a análise. Os primeiros incrementos serão versões simplificadas do produto que, por

possuírem algumas capacidades, poderão ser utilizadas pelos usuários, assim como servir de plataforma de avaliação (PRESSMAN, 2006).

Ao comparar o modelo incremental com o modelo em cascata, podemos identificar três vantagens importantes. A primeira delas é o custo reduzido de acomodar mudanças, pois o modelo incremental já prevê que irão ocorrer mudanças no escopo. A segunda é o *feedback* constante recebido dos clientes, pois, por receber periodicamente uma versão funcional do produto, a análise é facilitada. Ao contrário dos documentos que dificultam muito avaliar a evolução do projeto, com o produto o usuário poderá interagir e verificar como o mesmo vem sendo desenvolvido. A terceira e última vantagem é que há a possibilidade do cliente obter ganhos a partir do produto antes que o mesmo esteja completo. Se comparado ao modelo em cascata, quando o usuário terá de esperar o final do processo, o modelo incremental é bem melhor (SOMMERVILLE, 2011).

Da visão de gerenciamento podemos identificar dois problemas principais. O primeiro deles é o processo que não é visível, ou seja, os gerentes necessitam de entregas regulares para medir o progresso. Se a entrega é realizada com rapidez não será viável economicamente o desenvolvimento dos documentos que descrevem cada versão. Outro problema é a degradação da estrutura do sistema em decorrência de novos incrementos. É necessário realizar constantes refatorações para melhoria do software, o que irá demandar tempo e dinheiro (SOMMERVILLE, 2011).

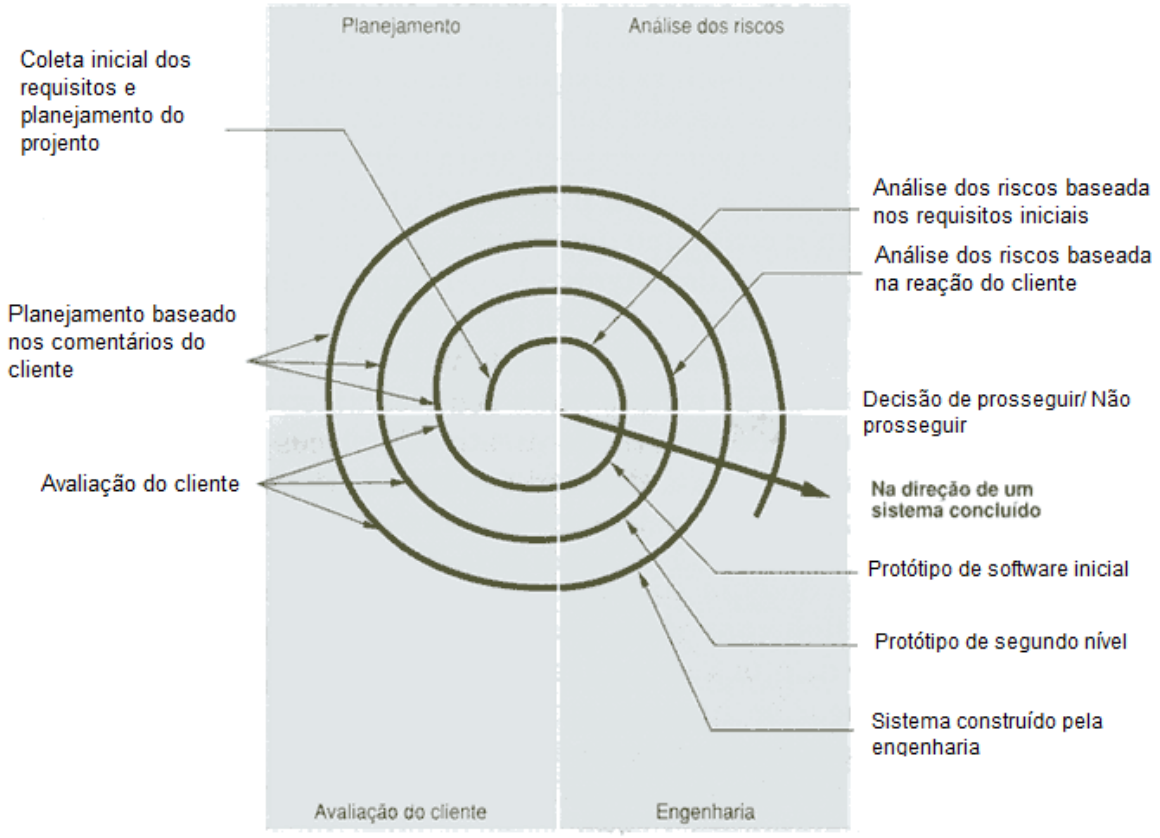
Esse processo de software vem a ser útil quando não há mão de obra para uma implementação completa dentro do prazo de entrega estabelecido. Os primeiros incrementos do processo podem ser realizados com recursos humanos reduzidos, e o aumento dos recursos humanos pode ocorrer ao longo da produção dos próximos incrementos e a aceitação dos mesmos por parte do usuário (SBROCCO, 2012).

2.3.3 Modelo espiral

O modelo espiral é definido por ser uma combinação entre a natureza da prototipagem com os aspectos controlados e sistemáticos do modelo em cascata (PRESSMAN, 2006). Esse modelo é dirigido ao risco e, como mostra a Figura 7, seu processo é representado como uma espiral e não como uma sequência. Cada volta da espiral representa uma fase do processo de

software. Dessa forma, a volta mais interna representa a viabilidade do sistema, a volta seguinte representa a definição dos requisitos, a seguinte o projeto do sistema e, assim, segue até chegarmos ao produto final (SOMMERVILLE, 2011).

Figura 7 - Modelo espiral



Fonte: SBROCCO (2012, p.67).

Cada volta do espiral é dividida em quatro partes. A primeira parte é a definição dos objetivos. Nessa parte, além da definição dos objetivos, também são identificadas as restrições ao processo do produto e os riscos do projeto, elaborando-se um plano de gerenciamento. A segunda parte é a avaliação e redução de riscos. Para cada risco identificado é realizada uma análise detalhada do mesmo e as medidas visando à redução do risco são tomadas. A terceira é o desenvolvimento e a validação. Após a avaliação dos riscos, deverá ser escolhido o modelo mais adequado para o desenvolvimento da solução nessa etapa. A quarta e última parte é o planejamento, é a etapa em que o projeto é revisado e é tomada a decisão se ele continuará, ou seja, serão elaborados os planos para uma nova volta, ou se o mesmo foi finalizado (SOMMERVILLE, 2011).

De acordo com Pressman (2006), a primeira volta no espiral poderia ser utilizada para o desenvolvimento das especificações de um produto, as próximas poderiam ser utilizadas no

desenvolvimento de um protótipo e depois poderiam resultar em versões mais sofisticadas do produto. Dessa forma, cada passagem pelo setor de planejamento acarretará em ajustes no projeto do produto, assim o custo e o cronograma poderão ser ajustados de acordo com o *feedback* recebido dos clientes após o uso dos protótipos.

Ainda, segundo Pressman (2006), o modelo espiral é uma abordagem de desenvolvimento de sistemas de grande porte. O software evolui juntamente com o processo, dessa forma o cliente e usuário passam a entender melhor o processo e conseguem reagir melhor aos riscos descobertos. Esse modelo mantém a sistemática passo a passo sugerida pelo modelo em cascata e incorpora uma estrutura iterativa, a qual se adapta mais à atualidade. O modelo espiral se baseia diretamente na consideração de todos os riscos técnicos em todas as etapas. Se o modelo for aplicado corretamente, deve reduzir os riscos antes que se tornem problemas.

2.4 Metodologias ágeis

Nos tempos atuais, prever a evolução de um sistema baseado em computador se tornou praticamente impossível. Isso ocorre devido às condições do mercado estarem mudando rapidamente, a evolução das necessidades dos usuários finais evoluírem, seguida das constantes ameaças de competição que emergem sem alerta. Dessa forma, torna-se inviável definir os requisitos completamente antes do início do projeto, o que força os engenheiros de software a se tornarem ágeis para responder prontamente a um ambiente mutante (PRESSMAN, 2006).

A metodologia ágil pode ser definida como métodos de desenvolvimento incremental, em que serão desenvolvidos pequenos incrementos. Esses incrementos, normalmente, resultam em pequenas novas versões que servem para os clientes realizarem os testes, o que pode ocorrer entre duas ou três semanas. O objetivo desses curtos *releases* é o de obter a opinião do cliente, assim como a demonstração da evolução dos requisitos do produto. Dessa forma, é possível minimizar as necessidades da documentação através do uso da comunicação informal (SOMMERVILLE, 2011).

As metodologias ágeis têm sido muito bem-sucedidas em dois tipos de desenvolvimento de sistemas. Pode-se citar o desenvolvimento de pequenos e médios

produtos para a venda e que não necessitem de uma grande alocação de recursos. E, também, é possível citar o desenvolvimento de sistema personalizado dentro de uma organização, ou para público externo, desde que haja um comprometimento de ambas as partes, com o *feedback* constante para guiar a evolução do produto (SOMMERVILLE, 2011).

Segundo Pressman (2006), a agilidade não é apenas uma resposta às modificações que ocorrem no decorrer de um projeto. Ela engloba, também, a filosofia apresentada no manifesto ágil, o qual encoraja as atitudes da equipe, a fim de facilitar a comunicação entre os membros da equipe, torna o cliente uma parte da equipe de desenvolvimento com o objetivo de tornar o projeto muito mais flexível.

Nesta seção, será apresentado o conceito da metodologia ágil (Manifesto Ágil), assim como os principais métodos de desenvolvimento ágeis de software.

2.4.1 Manifesto Ágil

Devido à insatisfação com os modelos de software clássicos, em 2001, ocorreu uma reunião com um grupo notável de 17 desenvolvedores. Esse grupo era formado por produtores e desenvolvedores de software que trocaram experiências, teorias e práticas referentes ao desenvolvimento de software. Neste encontro, essas pessoas reuniram um conjunto de princípios básicos necessários para o sucesso de um projeto e assinaram o "Manifesto para o Desenvolvimento Ágil de Software" (SBROCCO, 2012).

Nesse manifesto eles declararam:

> Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Através deste trabalho, passamos a valorizar:
>
> - Indivíduos e interações 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 (AGILLE MANIFESTO, 2013).

Por trás do manifesto ágil, existem os doze princípios, os quais foram criados pela *agile aliance*[1], e são listados a seguir (AGILLE MANIFESTO, 2013):

[1] Organização não lucrativa que promove o desenvolvimento ágil.

- Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado;
- Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando à vantagem competitiva para o cliente;
- Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo;
- Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto;
- Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho;
- O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face;
- Software funcionando é a medida primária de progresso;
- Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente;
- Contínua atenção à excelência técnica e bom design aumentam a agilidade;
- Simplicidade (a arte de maximizar a quantidade de trabalho não realizado) é essencial;
- As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis;
- Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Baseados nos doze princípios criados pela *agile alliance,* é que cada um dos métodos ágeis criados busca oferecer um conjunto de boas práticas e atividades para serem utilizados durante o desenvolvimento de projetos. A constante redefinição de prioridade e requisitos passa a ser a ideia principal para que o projeto possa se adaptar às constantes mudanças que ocorrem.

O uso das metodologias ágeis, atualmente, vem se tornando comum no desenvolvimento de software. A aproximação do cliente ao processo de desenvolvimento de software vem se tornando algo muito eficiente, pois, dessa forma, juntamente com o uso de processo de desenvolvimento iterativo, as equipes recebem o *feedback* logo após a

apresentação do produto, o que acaba se tornando um excelente aliado para lidar com as constantes mudanças de funcionalidade

2.4.2 *Extreme Programming* (XP)

XP é um dos mais conhecidos e mais utilizados dos métodos ágeis. Criado em 2000 por Kent Beck, foi desenvolvido com o objetivo de impulsionar as práticas de desenvolvimento iterativo. No XP, por exemplo, várias versões de um mesmo sistema podem ser desenvolvidas, testadas e integradas em um mesmo dia (SOMMERVILLE, 2011).

Segundo Sbrocco (2012), o XP possui três características marcantes, sendo elas o f*eedback* constante, a abordagem incremental e o encorajamento da comunicação entre as pessoas. Ele também é conhecido por ser a metodologia de desenvolvimento de software menos formal. O sucesso do XP ocorreu devido ao fato da metodologia ter causado uma revolução no conceito de desenvolvimento de software com a ideia básica do desenvolvimento rápido, garantia de satisfação do cliente e a ideia de favorecer o cumprimento das estimativas.

O XP é uma metodologia que preza seus valores. Ou seja, um projeto de software que pretende utilizar essa metodologia terá de seguir à risca um conjunto de valores sob a forma de diretrizes. Essas diretrizes devem ser encaradas como atitudes e prioridades de desenvolvimento, mesmo que haja adaptações. Conforme Sbrocco (2012), os cinco valores são:

- Comunicação: os desenvolvedores devem realizar a comunicação direta, de preferência face a face, para resolver os problemas relativos ao projeto para que não haja mal entendido. Essa prática reduz drasticamente as chances do projeto ficar com pendências;
- *Feedback*: à medida que o projeto for implementado, deverá ser utilizado imediatamente pelo usuário, criando, dessa forma, um processo de *feedback* contínuo. Deve envolver todos os membros interessados no projeto durante o ciclo de desenvolvimento;
- Simplicidade: durante a concepção do projeto, os desenvolvedores devem ser orientados a sempre resolver os problemas pela forma mais fácil. Para o *feedback* e

comunicação funcionarem corretamente, a equipe deverá utilizar o caminho da simplicidade. A metodologia aposta em realizar uma coisa simples hoje, que poderá ser melhorada amanhã, do que desenvolver algo complexo que no futuro poderá não ser utilizado;

- Respeito: visa ao respeito mútuo entre clientes e desenvolvedores, assim como os próprios membros da equipe. Aceitar críticas, dar ouvido às colocações de outros membros da equipe;
- Coragem: os desenvolvedores devem estar preparados para tomar decisões difíceis que suportem os outros valores.

2.4.3 FDD

A metodologia ágil *Feature Driven Development* (FDD) foi criada em Singapura, entre os anos de 1997 e 1998. A primeira utilização dessa metodologia se deu para a construção de um sistema bancário internacional, o qual havia sido considerado inviável de ser desenvolvido em um prazo determinado. Criado por Jeff De Luca e Peter Coad, é considerada uma metodologia ágil, robusta e muito utilizada atualmente (SBROCCO, 2012).

Segundo Pressman (2006), podemos traduzir FDD como desenvolvimento guiado por características. No contexto do FDD, uma característica deve ser interpretada como "uma função valorizada pelo cliente que poderá ser implementada em até duas semanas". Podemos observar os seguintes benefícios de seu uso:

- Pelas características serem pequenos blocos passíveis de entrega, os clientes podem descrevê-lo com mais facilidade, além de visualizar e revisar a implementação desses requisitos;
- Essas características podem ser organizadas de forma a facilitar a sua visualização, com a utilização de um agrupamento por hierarquia de relacionamento ao negócio;
- Entrega a cada duas semanas;
- É possível identificar a facilidade quanto à inspeção dessas características devido ao fato das mesmas serem pequenas;
- Facilidade na criação do cronograma e monitoramento do mesmo devido ao fato da utilização da organização por hierarquia.

O FDD, diferente de muitos outros métodos ágeis, enfatiza mais as diretrizes e técnicas de gestão de projeto. Isso se deve ao fato de, à medida que os projetos crescem em tamanho e complexidade, é extremamente importante que os desenvolvedores, seus gerentes e os clientes possam visualizar o estado do projeto, ou seja, quais avanços foram realizados e também quais problemas foram descobertos. Para conseguir isso, o FDD define, ao todo, seis marcos de referência durante o projeto, sendo eles: travessia do projeto, projeto, inspeção de projeto, código, inspeção de código e promoção para construção (PRESSMAN, 2006).

2.4.4 DSDM

Dynamic Systems Development Methodology (DSDM), no português, pode ser denominado de Método de Desenvolvimento Dinâmico de Sistema. Segundo Pressman (2006), DSDM é uma abordagem ágil de desenvolvimento de software que fornece uma estrutura para a construção de projetos que possuam um prazo apertado através do desenvolvimento iterativo. Segundo Sbrocco (2012), o DSDM tem foco em estabelecer os recursos e o tempo fixo para o desenvolvimento de um projeto, através do ajuste das funcionalidades, objetivando atender aos prazos estipulados.

O DSDM sugere, assim como o XP, um processo iterativo para o desenvolvimento de software. Na abordagem de cada iteração, é utilizada a regra dos 80%. Essa regra sugere que 80% de uma aplicação podem ser entregues em 20% do tempo que levaria para entregar uma aplicação completa, ou seja, apenas é necessário certo trabalho para que cada incremento facilite o avanço para o incremento seguinte (PRESSMAN, 2006).

Segundo Sbrocco (2012), o DSDM possui um ciclo de vida composto por cinco fases para aplicação de sua metodologia. São as fases: estudo da viabilidade, estudo de negócio, modelo de iteração funcional, projeto e construção de iteração e implementação. As fases serão detalhadas a seguir.

O estudo de viabilidade consiste em um processo que deve ser realizado uma vez durante o projeto. O seu propósito é o de analisar a viabilidade do projeto, ou seja, verificar se será possível a utilização do DSDM para o projeto. Nessa fase, é verificada a organização do projeto, são listadas as pessoas envolvidas e são mapeados os riscos que podem vir a ocorrer.

Esse estudo deve durar em torno de duas a três semanas e, como produto final, serão criados relatórios referentes à viabilidade e um plano geral para desenvolvimento.

O estudo de negócio consiste em captar as características do negócio. A melhor forma de execução vem a ser a proposta de *workshops* e reuniões com representantes da empresa para discutir os requisitos a serem implantados. Esses representantes devem chegar a um acordo das prioridades dos requisitos levantados. Como resultado, será criada uma lista de prioridades com o seu respectivo responsável.

O modelo de iteração funcional consiste em um planejamento de conteúdo e abordagem a ser aplicado após cada iteração para análise do que foi produzido. Tem como objetivo a criação de protótipos que serão disponibilizados aos clientes com a finalidade de receber um *feedback*. O final do processo resultará na atualização da lista de prioridades, com base no *feedback* dos clientes, assim como comentários dos clientes para a iteração seguinte e a lista dos requisitos não funcionais e documento de análise de risco.

O projeto e a construção da iteração consistem no processo que resultará na criação do sistema testado de acordo com as necessidades dos clientes.

A implementação consiste na passagem do sistema do ambiente de desenvolvimento para o sistema de produção. Nesse processo, os usuários são treinados para utilizar o sistema. Nessa fase, também são listados os *feedbacks* dos usuários para futuras iterações, quando um novo modelo funcional deverá ser planejado, assim como novos prazos e recursos deverão ser definidos.

2.4.5 Crystal Methods

Criado em 1998, por Alistair Cockburn, consiste em uma família de metodologias que possuem o objetivo de suprir as necessidades da época. De acordo com Sbrocco (2012), trata-se de uma família de metodologias com um código genético para atender diferentes tipos de projetos e de vários tamanhos. Já Pressman (2006), define o *Crystal* como:

> [...] uma abordagem de desenvolvimento de software que premia a "manobrabilidade" durante o que Cockburn caracteriza como 'um jogo cooperativo de invenção e comunicação de recursos humanos limitados, com o principal objetivo de entregar softwares úteis funcionando e com o objetivo secundário de preparar-se para o jogo seguinte'(PRESSMAN, 2006, p.71).

Essa metodologia foi dividida em cores, sendo que, quanto mais escura a cor, mais crítico o sistema seria. Cada cor tem um objetivo e o sistema é separado de acordo com o nível crítico, ou seja, projetos com menos desenvolvedores e problemas, o prejuízo tende a ser menor. Já projetos maiores, com um maior número de pessoal e um nível crítico de segurança tende a ter um prejuízo muito maior (SBROCCO, 2012). Pode-se observar, na Tabela 1, a organização de cores e nível crítico.

Tabela 1 - Divisão da família *Crystal*

Cores	Número de desenvolvedores	Em caso de falha
Clear	1-6	Perdem dinheiro, mas recuperam facilmente.
Yellow	7-20	Perdem dinheiro discretamente.
Orange	21-40	Perdem dinheiro substancialmente.
Red	41-100	Há perda substancial de dinheiro e, possivelmente, vidas humanas.

Fonte: SBROCCO (2012).

Além das cores, Sbrocco (2012, pg. 134), descreve os sete princípios básicos como:

a) Envolver o cliente nas iterações e nas decisões do projeto;

b) Quanto maior a complexidade do projeto, maior será o custo do mesmo;

c) Uso de metodologias diferenciadas para equipes maiores;

d) Maior número de cerimônias, visando a aumentar o número de diálogo entre os envolvidos;

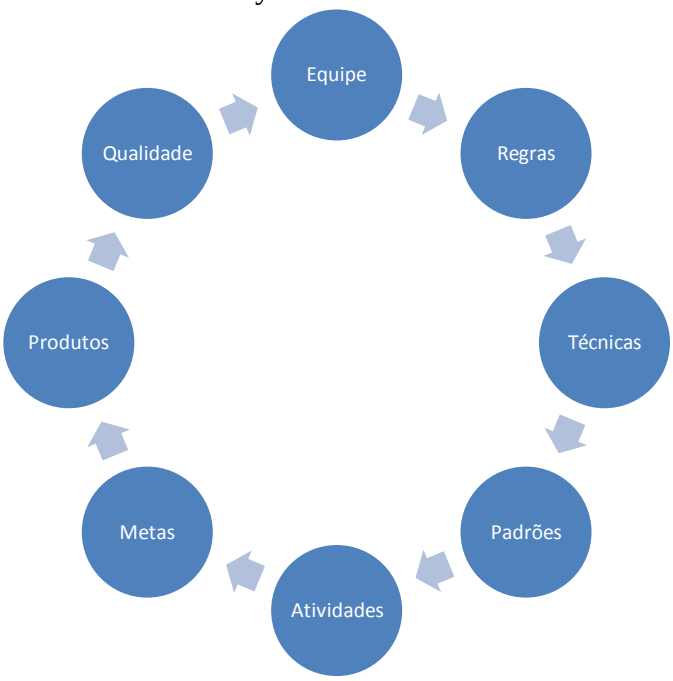
e) Comunicação eficiente;

f) Tolerância ao lidar com seres humanos;

g) Eficiência no desenvolvimento.

Sbrocco (2012) ainda define o ciclo de vida do *Crystal* baseado em integração e o mesmo se assemelha a um relógio. Na Figura 8, pode ser observado o ciclo de vida da metodologia *crystal.*

Figura 8 - Ciclo de vida Crystal



Fonte: SBROCCO (2012, p.136)

2.5 Scrum

O Scrum é uma metodologia de desenvolvimento ágil desenvolvida por Jeff Sutherland e equipe em 1990 (PRESSMAN, 2006). Como muitas das metodologias consideradas ágeis, o Scrum foi fortemente influenciado pela cultura japonesa e suas boas práticas, em especial pelos princípios da manufatura enxuta utilizada pelas companhias Honda e Toyota. Em 1986, Takeuchi e Nonaka escreveram um famoso artigo intitulado "*The new product game*" e descreveram que projetos que usam equipes pequenas e multidisciplinares produzem melhores resultados (SBROCCO, 2012).

O nome Scrum se deu devido à associação das equipes altamente eficazes que utilizavam Scrum com uma típica formação do rúgbi. No rúgbi, essa formação é utilizada quando há a necessidade de reiniciar o jogo, reunindo todos os jogadores. De acordo com Sbrocco (2012, p. 159), "o uso dessa terminologia pareceu adequado, porque, no rúgbi, cada time age em conjunto, como uma unidade integrada, cada membro desempenha um papel específico e todos se ajudam em busca de um benefício comum".

De acordo com Pressman (2006), os princípios do Scrum são:

- Pequenas equipes de trabalho organizadas com objetivo de maximizar a comunicação, minimizar a supervisão e maximizar o compartilhamento de informações;
- Com objetivo de garantir que o produto seja produzido da melhor forma possível é preciso que tanto as modificações técnicas quanto a de negócios seja adaptável;
- O processo produz frequentes incrementos de software que, no final de cada ciclo, podem ser testados, ajustados, documentados e expandidos;
- Documentação e testes devem ser realizados no decorrer da construção do produto;
- O processo fornece a habilidade de declarar o produto pronto sempre que necessário;
- O trabalho do desenvolvimento, assim como o pessoal que irá realizá-lo, deve ser dividido em pequenas partes para fácil gerenciamento.

Não há uma solução mágica para a resolução de problemas complexos, mas há um consenso de que o Scrum pode ser utilizado na resolução destes problemas (SBROCCO, 2012), sendo eles:

- Desenvolvimentos complexos em que os requisitos mudam rapidamente e constantemente;
- Gerenciar e controlar o desenvolvimento do trabalho;
- Tornar a equipe autogerenciável e funcional;
- Implementar o conceito iterativo e incremental no desenvolvimento de software e/ou produtos;
- Identificar causas de problemas e remover impedimentos;
- Valorizar indivíduos.

Segundo Schwaber (2004), o *Scrum* combinado com o XP possui práticas de melhoria contínua que aumentam consideravelmente a produtividade da equipe. Uma prática que os membros da equipe adquirem com o *Scrum* vem a ser o comprometimento, pois, a partir do momento em que a equipe decide o que será feito, ela fará de tudo para concluir. Promovendo, também, a união da equipe, pois fará com que os membros se ajudem para alcançar o objetivo comum.

De acordo com Pressman (2006), os princípios listados anteriormente são utilizados com o objetivo de guiar as atividades de desenvolvimento. Cada atividade, para desenvolver um requisito, ocorre dentro de um padrão chamado de *Sprint*. A complexidade do requisito irá interferir na quantidade de S*prints* necessários para desenvolver a solução. A equipe tem o

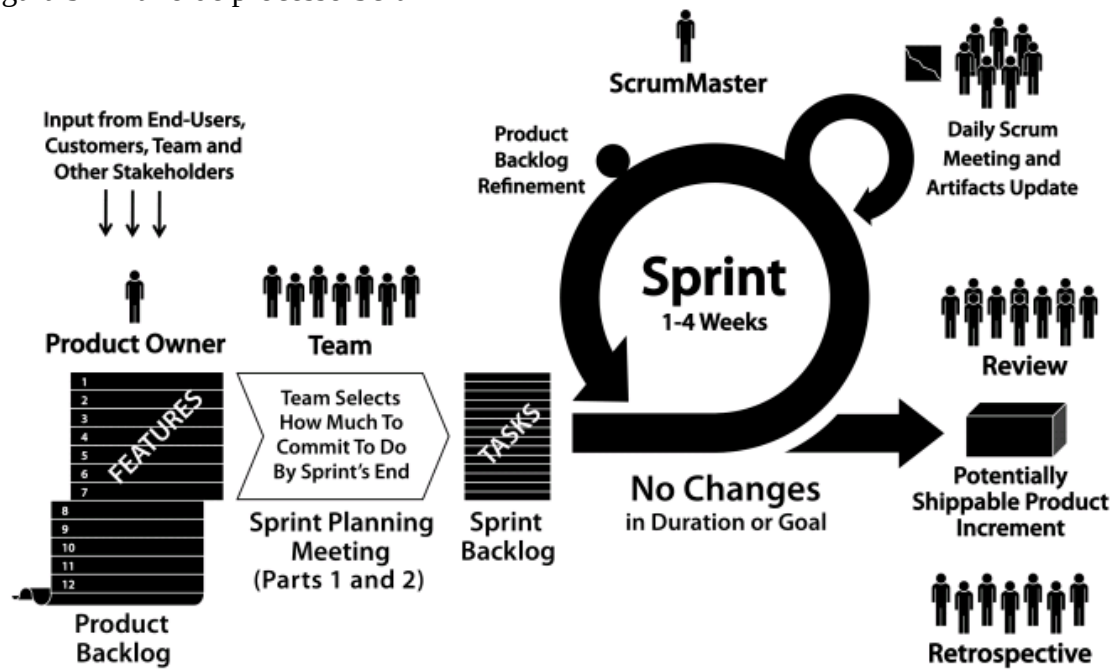
princípio básico de que o cliente pode mudar de ideia no decorrer do ciclo de vida, dessa forma, a equipe deverá ser flexível para incorporar essas mudanças no clico de vida.

2.5.1 Ciclo de desenvolvimento

Sutherland (2012) define o Scrum como um *framework* iterativo e incremental, tendo sua estrutura de desenvolvimento baseada em ciclos de trabalho, denominado *Sprint*, e em reuniões que ocorrem no decorrer do projeto. O ciclo de desenvolvimento tem como início a criação do *Product Backlog* (PB) pelo *Product Owner* (PO), o qual transcreve as necessidades do cliente e as prioriza para o processo do Scrum. Também são levantados os custos do projeto, assim como os seus riscos e também é definido quem será o *Scrum Master*. Após as definições iniciais, o próximo passo vem a ser a elaboração do *Sprint Backlog*. Esse processo é realizado no *Sprint Planning Meeting*, que é dividida em duas partes, e nelas são definidos os itens que irão fazer parte do *Sprint Backlog*, a qual dá origem ao *Sprint*.

Cada iteração não tem mais de um mês e, após o término, outra se inicia sem pausa e são denominadas de *Sprint*. No início de cada *Sprint*, a equipe e o cliente se reúnem, escolhem os itens com prioridade a serem desenvolvidos e definem um prazo para entrega. Esses itens não poderão ser alterados durante o *Sprint*. Diariamente é verificado o progresso do trabalho e realizada uma pequena reunião para verificar o andamento do projeto denominado *Daily Scrum*. No final do *Sprint*, a equipe se reúne com o cliente e demonstra o que foi desenvolvido, sendo realizado o *Sprint Review* e, o por fim, o *Sprint Retrospective*. Com o S*print* finalizado, o ciclo é iniciado novamente (SUTHERLAND, 2012). O fluxo descrito pode ser verificado na Figura 9.

Figura 9 - Fluxo de processo Scrum



Fonte: SUTHERLAND (2012, p.26).

Ainda, segundo Sutherland (2012), o principal foco do Scrum é ter o produto pronto no final do *Sprint*. No caso de software, significa ter o código integrado, totalmente testado e passível de entrega. O Scrum tem como tema principal a inspeção e a adaptação. Dado que desenvolvimento de software inevitavelmente envolve o processo de aprendizado, a inovação e as surpresas, o Scrum frisa a curta etapa de desenvolvimento, inspecionando tanto o produto resultante como as práticas utilizadas para chegar ao objetivo, para, então, adaptar os objetivos do produto, assim como os processos praticados, repetindo o processo até o cliente estar satisfeito com a entrega.

2.5.2 *Sprints*

Os projetos que fazem uso de Scrum são divididos em ciclos chamados de *Sprint*, os quais correspondem às iterações. Esses ciclos têm em torno de duas a quatro semanas, tendo o principal objetivo de gerar um produto de valor ao cliente e dentro dos requisitos previamente estabelecidos com o cliente para aquele ciclo. Tanto o cliente como a equipe deve ter ciência de que nenhuma mudança deve ocorrer durante um *Sprint* (SBROCCO, 2012).

Sbrocco (2012, pg. 163) define o *Sprint* como "[...] a unidade básica de desenvolvimento do Scrum, entendida como um esforço dentro de uma "caixa de tempo"; em

outras palavras, um esforço restrito de duração específica". Já Sutherland (2012), define o *Sprint* como:

> Uma iteração, ou um ciclo repetitivo de trabalho, que produz um incremento de produto ou sistema. Não mais que um mês e, geralmente, mais de uma semana de ciclo. A duração é fixada ao longo do trabalho total e todas as equipes trabalhando no mesmo sistema ou produto utilizam o mesmo tempo de ciclo (SUTHERLAND, 2012, pg. 30).

O cliclo do *Sprint* é iniciado pelo *Sprint Planning Meeting*, que pode ser denominado como Reunião de Planejamento do *Sprint*. Essa reunião é realizada entre *Product Owner* e a equipe e tem como principal objetivo identificar itens de maior prioridade do *Backlog* que futuramente irão compor o *Sprint*. Com o objetivo traçado, diariamente, durante o *Sprint*, a equipe deverá se reunir por até quinze minutos para atualizar o progresso referente aos itens do *Backlog* e trocar informações entre os membros. Ao final do *Sprint*, é realizado o *Sprint Review*, com o objetivo de formalizar o final do ciclo, reunião esta com objetivo de apresentar as funcionalidades concluídas no decorrer do *Sprint* para o cliente. Entre o *Sprint Review* e o novo *Sprint* é realizado o *Sprint Retrospective*, com o objetivo de rever todos os processos e as práticas realizadas no último *Sprint* para que melhorias possam ser definidas ao processo. Reunião esta realizada juntamente com o *Scrum Master* e com toda a equipe. Após essa reunião, o ciclo do *Sprint* é iniciado novamente (SUTHERLAND, 2012).

Segundo Schwaber (2004), o *Scrum Master* tem um papel importante durante o *Sprint*, que vem a ser de manter a equipe focada no objetivo do *Sprint* e afastada de interferências externas. Dessa forma, nenhuma alteração deve ser realizada no *Sprint Backlog* até o final do *Sprint*. Mudanças são bem vindas, desde que sejam realizadas no *Product Backlog* e devem ser controladas pelo *Product Owner*. Os novos itens a serem incluídos no *Product Backlog* deverão ser priorizados para que futuramente possam ser desenvolvidos através de um *Sprint*.

O Scrum define que o tempo de um *Sprint* não deve ser alterado após o seu início ou estendido. Mesmo que em um *Sprint* a equipe veja que não conseguirá realizar todas as tarefas o *Sprint* deverá ser concluído. Isso se deve ao fato de que para uma equipe Scrum amadurecer em suas estimativas, ela leva em torno de três a quatro *Sprints* consecutivos. O simples fato de um *Sprint* não ser alterado deixa a equipe mais confiante e ajuda a encontrar seu ritmo, ritmo este denominado de "*heartbeat*" no Scrum.

2.5.3 Papéis

O Scrum se divide em três papéis principais: o *Product Owner*, o *Scrum Master* e o *Team*. Juntos, esses três papéis formam o *The Scrum Team* (SUTHERLAND, 2012). Ainda, segundo Sbrocco (2012), é considerado prudente, por alguns, incluir também a figura do cliente nos papéis do Scrum, isso se deve ao fato do cliente possuir um papel importante no processo de desenvolvimento.

2.5.3.1 *Product owner*

O *Product Owner*, que pode ser traduzido para o português como "dono" do produto, é responsável por maximizar o *Return on Investment* (ROI) através da identificação de funcionalidades do produto, as quais devem ser transformadas em uma lista de prioridades, decidindo qual item será mantido no topo para o próximo *Sprint*. A cada início de novo ciclo, essa lista de prioridades deverá ser refinada e repriorizada de acordo com a necessidade. Quanto ao ROI, sobre o qual o *Product Owner* tem responsabilidade, não é referente ao sentido comercial, e sim no sentido de realizar a escolha certa para cada *Sprint* e referente aos itens priorizados no *Backlog* a serem desenvolvidos. Em alguns casos, o *Product Owner* e o cliente podem ser a mesma pessoa. Isso é comum para o desenvolvimento interno. Enquanto que em outros casos o *Product Owner* possa representar milhões de pessoas com uma variedade de necessidades (SUTHERLAND, 2012).

Segundo Sbrocco (2012), o *Product Owner* é o representante do cliente e responsável por garantir que a equipe agregue valor ao negócio. Desempenha o papel de mediador entre o cliente e a equipe, tendo a responsabilidade de manter a equipe funcional. É responsável por definir a visão e as funcionalidades do produto, elaborar e manter o *Product Backlog*, definir as datas de lançamento do produto, representar o cliente e aceitar ou rejeitar os resultados dos trabalhos.

2.5.3.2 *Scrum Master*

O *Scrum Master* é responsável por aplicar e ensinar o *Scrum* à equipe de desenvolvimento com o objetivo de alcançar o valor do negócio. Deve fazer tudo que estiver

ao seu alcance para ajudar a equipe e o *Product Owner* a alcançar seus objetivos. Como o *Scrum Master* não exerce o papel de gerente da equipe ou gerente do projeto, ele deve se dedicar à equipe, educando e guiando o *Product Owner* e a equipe no uso do *Scrum,* além de proteger a equipe de interferências externas. Conforme o *Scrum* for tornando visíveis os impedimentos e as ameaças para a equipe ou *Product Owner*, o *Scrum Master* deve se dedicar em tempo integral para resolver esses problemas, focando sempre no sucesso do projeto (SUTHERLAND, 2012).

O *Scrum Master* também atua juntamente com o *Product Owner* nas tarefas referentes à definição de funcionalidades com valor ao cliente, além de planejar e elaborar em conjunto com o *Product Owner* o *Backlog* (SBROCO, 2012).

Em suma, o *Scrum Master* é responsável pelo processo *Scrum*, por ensiná-lo a todos os envolvidos no projeto, por introduzi-lo na cultura da empresa, além de garantir que todos os envolvidos sigam as regras e as práticas do *Scrum*.

2.5.3.3 Equipe *Scrum*

A equipe é responsável por construir o produto. A equipe, no *Scrum,* deve ser multifuncional, abrangendo todas as áreas do conhecimento necessárias para entregar o produto utilizável no final de cada *Sprint*. A equipe também deverá ser auto-organizável, com um alto nível de autonomia e responsabilidade. Com essas duas características citadas, a equipe poderá decidir com quais tarefas se comprometerá e qual será a melhor forma de alcançar esses objetivos (SUTHERLAND, 2012).

Sbrocco (2012) define a equipe *Scrum* como sendo composta por um grupo de cinco a nove integrantes com características multifuncionais. Levando em conta o desenvolvimento de software, uma equipe deveria ser composta, por exemplo, de um analista, testadores e programadores em tempo integral e, em alguns casos, como no caso do banco de dados, um DBA poderia ser contratado para trabalhar apenas algumas horas no projeto. O integrante da equipe também não deve possuir títulos que diferencie uns dos outros, assim como a retirada de um membro da equipe somente poderá ser realizada ao término do *Sprint*. Isso se deve ao fato do *Scrum,* além de incentivar a auto-organização, também preza pela comunicação. A equipe com um membro que possua um título maior do que os demais poderá causar falhas na

comunicação, devido ao medo dos membros de título inferior divergirem da ideia do membro com maior título.

Ainda, segundo Sbrocco (2012), a equipe é responsável em realizar as estimativas referentes ao *Sprint Backlog*, por definir as tarefas a serem realizadas, por garantir a qualidade do produto e, ao fim, por apresentar o resultado ao cliente.

2.5.3.4 Cliente

O cliente é a parte interessada do projeto. Deve participar das tarefas relacionadas à implantação da lista de funcionalidades e, ao longo do desenvolvimento do projeto, deve realizar as análises dos protótipos (SBROCCO, 2012).

2.5.4 Cerimônias

O *Scrum* possui algumas cerimônias que ocorrem em momentos distintos. Essas cerimônias, formais ou não, têm como objetivo principal prover a comunicação entre a equipe. As principais cerimônias são o *Sprint Planning*, o *Daily Scrum Metting*, o *Sprint Review* e o *Sprint Retrospective*.

O *Sprint Planing* é uma reunião inicial, de preparação ao *Sprint* que deve ter, no máximo, oito horas de duração, pois o processo tem o objetivo de realizar o trabalho e não de pensar sobre o trabalho. Nessa reunião, o *Product Owner* deve elaborar a lista de prioridades que devem ser cumpridas pelo projeto. Essa reunião é dividida em duas partes com objetivos distintos (SBROCCO, 2012).

A primeira parte do *Sprint Planning*, que deve durar até quatro horas, é a parte quando o *Product Owner* e a equipe, intermediados pelo *Scrum Master*, verificam os itens de alta prioridade do *Backlog* que deverão ser implementados no *Sprint*. Nessa reunião, o *Product Owner* irá repassar à equipe a sua visão de produto para o *Sprint*. Ao final da primeira parte, o *Product Owner* poderá deixar a reunião, contudo deverá ficar disponível, nem que por telefone, para a segunda parte da reunião (SUTHERLAND, 2012).

Segundo Schwaber (2004), na primeira parte da reunião, é importante que a equipe questione o *Product Owner* quanto aos itens priorizados do *Backlog* apresentados por ele. As questões realizadas pela equipe devem ser referentes ao conteúdo, à finalidade, ao significado e às intenções de todos os itens apresentados com o objetivo de que a equipe realmente entenda o que o *Product Owner* busca para o *Sprint*. Após o entendimento de todos os itens apresentados, a equipe deverá selecionar desses itens, o quanto eles irão conseguir produzir no próximo *Sprint*.

A saída dessa primeira parte de reunião é o comprometimento da equipe em realizar as tarefas prometidas que compõem o *Sprint Backlog*. O *Product Owner* nem o *Scrum Master* devem interferir no comprometimento da equipe. Caso haja descontentamento por parte do *Product Owner*, ele poderá repriorizar as tarefas para atender a sua necessidade.

A segunda parte dessa reunião deverá ser realizada por quem realmente irá realizar o trabalho. Como citado anteriormente, o *Product Owner* poderá se ausentar da reunião, mas deverá ficar disponível para ser consultado pela equipe para esclarecer as dúvidas que possam surgir durante essa etapa. Essa reunião é focada em detalhar a implementação dos itens selecionados na primeira parte da reunião. Como, no *Scrum*, a equipe é autogerenciável, ela escolhe os itens com os quais irá se comprometer para o fim do *Sprint*. A equipe estima os itens que irão compor o *Sprint Backlog* de acordo com a prioridade estipulada pelo *Product Owner* e pelo espaço de tempo do *Sprint*, que é uma prática fundamental no *Scrum* (SUTHERLAND, 2012).

Para que o *Sprint* tenha sucesso, é necessário que, na segunda parte do *Sprint Planning Meeting*, cada membro tenha conhecimento sobre sua disponibilidade para realizar as tarefas. Ou seja, cada membro da equipe deverá calcular sua jornada de trabalho. Esse cálculo se dá basicamente pela jornada de trabalho subtraído o tempo gasto em atividades como participar de reuniões, ler e-mails, pausas, dentre outras atividades que possam ser realizadas. Em média, o tempo de trabalho deve ficar entre quatro e seis horas. A Figura 10 apresenta a estimativa de trabalho de uma equipe de quatro pessoas em um *Sprint* com duração de oito semanas. A equipe, conhecendo sua capacidade, consegue planejar, de forma eficiente, o *Sprint*, podendo sugerir ao *Product Owner* a inclusão de itens de menor prioridade dentro do *Sprint* para complementar um item de maior prioridade para uma entrega mais qualificada do processo (SUTHERLAND, 2012).

Figura 10 - Estimativa de horas de trabalho

Duração do Sprint	2 semanas
Dias de trabalho durante o Sprint	8 dias

Membro da Equipe	Dias de trabalho no Sprint	Horas disponíveis por dia	Total de horas disponíveis
Tracy	8	4	32
Sanjay	7	5	35
Philip	8	4	32
Jing	6	5	30

Fonte: SUTHERLAND (2012, p.19).

Diariamente, a partir do início do *Sprint*, deve ser realizada uma reunião de, no máximo, quinze minutos com todos os membros da equipe chamada de *Daily Scrum*. A reunião tem como principal objetivo sincronizar os trabalhos realizados e reportar a todos os membros da equipe os obstáculos encontrados, sendo aconselhável que seja realizado em pé, próximo ao quadro de tarefas. Essa reunião deve ser realizada com cada membro da equipe respondendo às seguintes questões (SUTHERLAND, 2012):

- O que eu fiz desde a última reunião?
- O que vou fazer até a próxima reunião?
- Quais os problemas que estão impedindo a realização do meu trabalho?

Segundo Sbrocco (2012), é importante que todos os membros da equipe participem da reunião e deve ser focado ao máximo nas três questões e não deve haver discussão de problemas. Já, Sutherland (2012), explica que essa reunião não pode ser considerada uma reunião de atualização de status do projeto para o gerente e, sim, considerado um pequeno espaço de tempo para a equipe se sincronizar e poder ajudar uns aos outros. Cabe ao *Scrum Master* intermediar a resolução dos problemas mencionados pelos membros da equipe. É recomendado que Gerentes não participem dessa reunião para que a equipe não se sinta vigiada e omita questões importantes.

É importante que, diariamente, os membros da equipe atualizem o *Sprint Backlog*. A atualização desse artefato tem como objetivo estimar o tempo restante para completar o item em desenvolvimento. No final de cada dia, o membro da equipe deve calcular o número de

horas restante e atualizar o *Burndown Chart*, o qual é uma forma rápida para os membros do *Scrum* identificarem o progresso do *Sprint* (SUTHERLAND, 2012).

Ao final de cada *Sprint*, é realizado o *Sprint Review*. Nessa reunião, o *Product Owner*, a equipe e demais interessados revisam o *Sprint*. Trata-se de uma reunião informal, um bate papo entre as partes e não deve durar mais de quatro horas. O objetivo principal dessa reunião não é demonstrar o produto do *Sprint*, mas sim o momento em que o *Product Owner* passa a saber o que ocorreu com o produto e a equipe. A equipe é informada sobre o que ocorre com o *Product Owner* e o mercado. Sendo o ponto mais importante da reunião uma conversa mais profunda entre a equipe e o *Product Owner*, a fim de obter conselhos, saber da situação, saber do contexto atual. Se, por algum motivo, o foco da reunião for a demonstração ao invés da conversa, algum desequilíbrio pode estar ocorrendo (SUTHERLAND, 2012).

A demonstração do produto não deve durar mais de trinta minutos, devendo ser superficial, demonstrando as principais funcionalidades desenvolvidas. Sendo que somente as tarefas 100% concluídas deverão ser apresentadas, as outras, não concluídas no *Sprint* deverão ser incluídas no próximo *Sprint* (SBROCCO, 2012).

A fim de complementar o ciclo do *Scrum*, é realizado o *Sprint Retrospective*. Essa reunião é realizada entre o *Scrum Master* e a equipe, após o *Sprint Review* e antes do *Sprint Planning Meeting*. Segundo Sbrocco (2012), o principal objetivo dessa reunião é verificar o que houve de bom durante o processo e deverá continuar, e os pontos negativos que deverão ser melhorados. Devem participar dessa reunião o *Scrum Master* e a equipe. O *Product Owner* poderá participar, desde que seja convidado. Todos os membros da equipe deverão responder a duas questões básicas: o que foi bom durante o *Sprint* e o que se pode fazer para melhorar a próxima. Fica ao encargo do *Scrum Master* tomar nota de tudo e o time deve priorizar os itens apontados na reunião para que seja criada uma ordem ideal de mudança. Se comparar o *Sprint Review* e o *Sprint Retrospective*, pode-se identificar que o *Sprint Review* envolve inspeção e adaptação com relação ao produto enquanto *Sprint Review* tem o foco no processo *Scrum* (SBROCCO, 2012).

2.5.5 Artefatos

A metodologia *Scrum* propõe a produção de três artefatos, sendo eles o *Product Backlog*, o *Sprint Backlog* e o *Burndown Chart*. Estes artefatos são produzidos em momentos distintos de um determinado *Sprint* (SBROCCO, 2012).

O *Product Backlog* é uma lista refinada e priorizada dos itens que deverão ser desenvolvidos ao longo do projeto, com o objetivo de esclarecer e especificar o que deve ser entregue ao cliente. Esta lista de requisitos é desenvolvida pelo *Product Owner*, o qual tem liberdade para alterar qualquer item da lista ou prioridade do item, desde que este não seja parte de um *Sprint* em andamento. Além das novas funcionalidades, o *Product Backlog* será composto também por itens de melhoria e por defeitos descobertos ao longo do projeto. A prioridade dos itens do *Product Backlog* é criada por interesse dos *stakeholders*[2] e influenciada pela equipe. A Figura 11 demonstra a forma de construção do *Product Backlog* (SBROCCO, 2012; SUTHERLAND, 2012).

Figura 11 - *Product Backlog*

Nível de Prioridade (ROI)	Ator	Requisitos Funcionais	Descrição do Item	Tamanho	Release	Sprint	Status
Essencial	Usuário	RF001	Cadastro de Usuário				To Do
Essencial	Usuário	RF002	Cadastro de curriculum				To Do
Essencial	Usuário	RF003	Cadastro de Interesse				To Do
Essencial	Usuário	RF004	Busca de oportunidade de emprego				To Do
Essencial	Empresas	RF005	Cadastro de empresas				To Do
Essencial	Empresas	RF006	Busca de candidato a emprego				To Do

Fonte: SBROCCO (2012, p.168).

Segundo Sbrocco (2012), o *Product Backlog* deve ser priorizado em relação ao ROI, ou seja, os itens que representam maior valor para o negócio devem ser desenvolvidos primeiro e o tamanho de cada item deve ser estimado pela equipe de desenvolvimento. Para Schwaber (2004), as mudanças que ocorrem no *Product Backlog*, além de esperadas, são bem vindas, já que esse é um dos princípios do *Scrum*. Visto que o *Product Backlog* nunca está completo e deverá ser refinado e priorizado ao longo do projeto.

[2]*Stakeholder* significa parte interessada. É uma palavra em inglês muito utilizada nas áreas de comunicação, administração e tecnologia da informação cujo objetivo é designar as pessoas e grupos mais importantes para um planejamento estratégico ou plano de negócios, ou seja, as partes interessadas.

O *Sprint Backlog* tem origem na segunda parte do *Sprint Planning Meeting* e representa todas as tarefas selecionadas que devem ser desenvolvidas durante um *Sprint*. A Figura 12 apresenta um exemplo de *Sprint Backlog*. Assim como o *Product Backlog*, o *Sprint Backlog* também deve ser representado em função do ROI, ou seja, de acordo com sua prioridade. Cada item do *Backlog* deve ser quebrado em tarefas menores e cada uma dessas tarefas deve ser estimada em horas (SUTHERLAND, 2012; SBROCCO, 2012).

De acordo com Schwaber (2004), somente a equipe pode realizar mudanças no *Sprint Backlog*. Essas mudanças podem ser identificadas ao decorrer do *Sprint*, quando a equipe identifica tarefas que não haviam sido planejadas inicialmente para concluir o item do *Backlog*. Como mostra a Figura 12, cada item possui suas tarefas, as quais foram priorizadas pela equipe e, ao lado de cada tarefa, há colunas com os dias de trabalho. Essa tabela deve ser atualizada diariamente com o tempo de trabalho. De acordo com Sbrocco (2012), cada tarefa deve ser realizada em sequência e é importante que a equipe somente comece a trabalhar na tarefa seguinte quando a anterior for completada. Esse comportamento evita que o time coloque a meta em risco, pois, como as tarefas são realizadas com base no ROI, as que estão no topo são as que possuem maior valor de negócio ao cliente.

Figura 12 - *Sprint Backlog*

Item	Tarefa	1	2	3	4	5	6	7	8	9	10	Total
ES006	Criar Base de Dados	4	0	0	0	0	0	0	0	0	0	4
	Desenvolver Modelo	8	8	0	0	0	0	0	0	0	0	16
	Desenvolver Controle	0	0	4	4	0	0	0	0	0	0	8
	Desenvolver View	0	0	0	4	4	0	0	0	0	0	8
	Teste de Unidade	0	0	0	0	4	2	2	0	0	0	8
	Teste Funcional	0	0	0	0	0	4	4	2	0	0	10
	Documentação Técnica	0	0	0	0	0	0	4	4	2	2	12

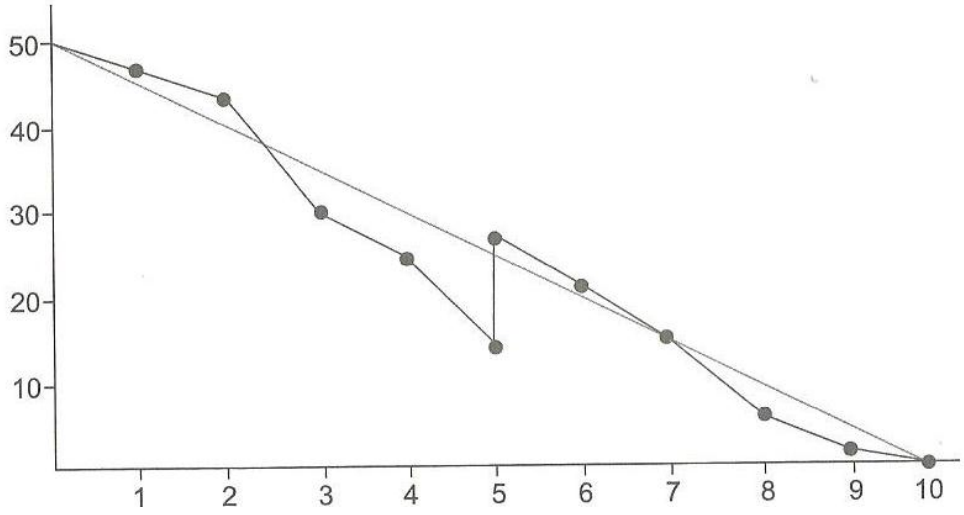
Fonte: SBROCCO (2012, p.169).

Segundo Sutherland (2012), um dos princípios do *Scrum* é incentivar as equipes multidisciplinares. Dessa forma, as tarefas criadas pela equipe no *Sprint Backlog* não devem ser nomeadas para um membro específico, e os membros devem buscar o que desenvolver e ajudar uns aos outros no que for possível. Nesse caso, se a equipe possui membros especialistas em determinados assuntos, esses membros poderão trabalhar com outros membros da equipe e ensinar novas habilidades aos colegas.

O *Burndown Chart* tem como objetivo mostrar à equipe o tempo restante para a conclusão do *Sprint*, assim como mostrar ao time o quão longe ou perto estão de atingir a

meta (SBROCCO, 2012). A Figura 13 mostra um exemplo deste gráfico. O eixo vertical representa a quantidade de esforço e o eixo horizontal representa o tempo. A linha contínua representa o fluxo ideal de trabalho, enquanto a linha pontilhada demonstra o fluxo de trabalho real. Se a linha pontilhada estiver abaixo da linha contínua, ou seja, perto do eixo horizontal, indica que a equipe está superando as expectativas e deverá acabar o trabalho antes do esperado. Porém, se a linha pontilhada estiver acima da linha contínua, ou seja, mais longe do eixo horizontal, indica que a equipe está passando por algumas dificuldades e poderá não finalizar no tempo estimado.

Figura 13 - *Burndown Chart*



Fonte: SBROCCO (2012, p.170).

Com essa forma de visualização, a equipe consegue identificar, diariamente, o seu ritmo de trabalho e consegue verificar seu progresso em direção ao objetivo. O mais importante desse gráfico não é apresentar o tempo que foi gasto no passado, e sim apresentar o progresso até o objetivo e quanto tempo ainda falta para alcançá-lo. Caso a linha pontilhada esteja tendendo para baixo, a equipe necessitará ajustar seu trabalho. Esse ajuste pode ser pela redução do escopo ou buscar por uma forma de trabalho mais eficiente e que mantenha um ritmo sustentável (SUTHERLAND, 2012).

3 METODOLOGIA

Este trabalho buscou implementar o *Framework Scrum* em uma empresa de desenvolvimento de Software, documentando as atividades de implantação do *Scrum* e suas etapas. Dessa forma, é caracterizado como um estudo de caso. No estudo de caso, o pesquisador interage com os sujeitos de uma forma semiformal, utilizando-se de entrevistas e conversas programadas, possui acesso também a dados, documentos e outros materiais da organização. Tendo como objetivo descobrir as práticas formais de uma organização, assim como opiniões e atitudes dos sujeitos (WAINER, 2007).

Com o acesso aos dados, documentos e outros materiais da organização, o pesquisador pode analisar e identificar os procedimentos formais da organização, assim como o tempo de duração de cada atividade, quem realiza o procedimento dentre outros vários fatores (WAINER, 2007). Dessa forma, o pesquisador, visualizando o ambiente externamente, pode identificar possíveis problemas nos processos e sugerir melhorias.

Conforme Yin (2010), o estudo de caso nos permite que sejam investigadas situações da vida real e que, delas, sejam absorvidas todas as características significativas dos eventos cotidianos. Nesses eventos, podemos incluir eventos de pequenos grupos, os processos organizacionais e administrativos. Eventos esses importantes para o desenvolvimento deste trabalho sendo que o foco deste trabalho será a mudança cultural da empresa.

O uso do estudo de caso como estratégia de pesquisa é composto por um método que engloba o todo, utilizando-se de uma lógica de planejamento na qual se utilizam abordagens específicas à coleta e análise de dados Yin (2010). Dessa forma, o estudo de caso não vem a

ser uma tática para coleta de dados ou uma característica de planejamento, e sim uma estratégia de pesquisa mais abrangente.

O estudo de caso pode ser definido como:

> A essência de um estudo de caso é tentar esclarecer uma decisão ou um conjunto de decisões: o motivo pelo qual foram tomadas, como foram implementadas e com quais resultados (SCHRAMM, 1971).

Yin também cita (2010, p.40) que "A investigação do estudo de caso enfrenta a situação tecnicamente diferenciada em que existirão muito mais variáveis de interesse do que pontos de dados e, como resultado conta com múltiplas fontes de evidência, com os dados precisando convergir de maneira triangular e, como outro resultado beneficia-se do desenvolvimento anterior das proposições teóricas para orientar a coleta e análise de dados.".

Quanto à natureza da pesquisa, o trabalho se caracteriza pela pesquisa exploratória. Conforme Wainer (2007, p. 29), a natureza da pesquisa se caracteriza como exploratória quando "além de descrever o fenômeno, faz propostas para novas teorias, ou novas observações, novas métricas para medir o fenômeno. Finalmente, a pesquisa é explanatória se ela busca provar ou desaprovar uma teoria particular."

O estudo de caso, neste trabalho, foi usado como forma de exploração sobre o tema definido, não gerando dados para amostragem, e sim como uma abordagem qualitativa quanto à própria utilização do *Framework Scrum* em sua capacidade de possibilitar mudanças positivas a uma empresa que não possui um padrão de trabalho definido. Segundo Wainer (2007, p. 5), "a pesquisa qualitativa baseia-se na observação cuidadosa dos ambientes nos quais o sistema está sendo usado ou nos quais será usado, do entendimento das várias perspectivas dos usuários ou potenciais usuários do sistema, etc.". A pesquisa bibliográfica tem por objetivo a exploração inicial dos assuntos, a fim de guiar o estudo de caso a ser realizado, provendo informações necessárias sobre o assunto. Sendo fontes da pesquisa bibliográfica os livros, as publicações periódicas, fitas gravadas de áudio e vídeos, *websites*, simpósios (WAINER, 2007).

Também foram levantados todos os procedimentos utilizados pela empresa antes do processo de implantação do *Framework Scrum*. Sendo que todo o processo de implantação do *Framework* será descrito.

Por fim, foi realizada uma avaliação da implantação do *Framework* visando a identificar e avaliar os impactos causados pelo procedimento. A avaliação foi desenvolvida, utilizando a comparação entre o processo antes da implantação e o processo após a implantação, considerando-se os aspectos de gestão de escopo, tempo e qualidade. Para dar maior ênfase aos resultados também foi realizada uma entrevista com as pessoas envolvidas no processo de implantação do *Framework Scrum*.

Sendo assim, esta pesquisa se define como qualitativa, de caráter exploratório, utilizando, como estratégia de investigação, o estudo de caso.

4 O ESTUDO

Este capítulo apresenta uma caracterização do ambiente de estudo no qual será desenvolvido o projeto em questão. Com o objetivo de implantar a metodologia Scrum em uma empresa de desenvolvimento de software optou-se por desenvolver tal experimento aplicado à RSData. Nesse sentido, procura-se apresentar uma introdução geral da empresa, bem como a identificação e comparação do processo anterior a implementação da metodologia e do processo posterior.

4.1 Caracterização

Nesta seção, a empresa RSData será caracterizada, apresentando o seu foco de trabalho, assim como os principais produtos e a divisão da equipe.

4.1.1 A empresa

A RSData começou a ser constituída em março de 2004, com objetivo de continuar o desenvolvimento de um software para a confecção e manutenção do Perfil Profissiográfico Previdenciário, o PPP, que existia desde de 2002.

A partir de março de 2004, o novo foco da empresa passou a ser o desenvolvimento de uma solução completa para gestão da Saúde e Segurança do Trabalho. Todo o trabalho é

desenvolvido com acompanhamento técnico do Engenheiro de Segurança do Trabalho Rogério Luiz Balbinot (consultor em SST há mais de 24 anos), além de outros profissionais da área técnica e médica.

A solução completa foi desenvolvida inicialmente para atender as demandas da empresa CONSETRA, pela qual o Engenheiro de Segurança do Trabalho Rogério Luiz Balbinot também é responsável. Formando-se, assim, uma parceria para o desenvolvimento da solução.

A CONSETRA é uma empresa prestadora de serviços na área de Segurança do Trabalho presente há mais de 24 anos no mercado, atendendo desde empresas de pequeno porte até multinacionais. Pela grande diversidade da área de atuação e tamanho das empresas atendidas pela CONSETRA, o dataSEESMT, software desenvolvido pela RSData, tornou-se um software configurável atendendo às necessidades da maioria das empresas do Brasil.

Em 2007, com a solução estável e atendendo a todas as necessidades da empresa CONSETRA, a solução começou a ser comercializada tanto para empresas com Serviço Especializado em Engenharia de Segurança e em Medicina do Trabalho (SESMT) próprio como para empresas que prestam consultoria de Saúde e Segurança do Trabalho. Atualmente, a RSData possui mais de duzentos clientes em todo o Brasil nas mais diversas áreas de atuação.

4.1.2 Os produtos

O foco das soluções da RSData é a Saúde e a Segurança do trabalho. A solução é dividida em cinco módulos. O dataSEESMT é o módulo responsável pela gestão da Saúde e Segurança do Trabalho, ele é um sistema que permite o gerenciamento das rotinas de saúde e segurança do trabalho, tanto de empresas com SESMT próprio como de empresas de consultoria/assessoria em SST. Sua principal característica é a facilidade de lançamento dos dados para a geração e controle de vencimentos de documentos como:

- Programa de Prevenção de Riscos Ambientais (PPRA);
- Laudo Técnico das Condições Ambientais do Trabalho (LTCAT);
- Programa de Controle Médico de Saúde Ocupacional (PCMSO);

- Programa de Condições e Meio Ambiente de Trabalho na Indústria da Construção (PCMAT).

O módulo RSÁudio foi elaborado para emissão e armazenamento das audiometrias ocupacionais de empresas ou de clínicas de fonoaudiologia. O sistema armazena e emite os exames audiométricos através de um cadastro simples e de fácil operação. Pode ser utilizado de forma individual ou totalmente integrado com os lançamentos do dataSEESMT e RSFinanceiro. O software foi desenvolvido seguindo o que está estipulado na Portaria 19, de 09/04/1998, que estabelece as diretrizes e parâmetros mínimos para a avaliação e o acompanhamento da audição dos trabalhadores expostos a níveis de pressão sonora elevada.

O módulo RSFinanceiro foi elaborado para gerir a parte financeira e é totalmente integrado com os demais softwares da RSData. Com o RSFinanceiro é possível realizar os movimentos, agendamentos, faturamento e o processamento das contas.

O módulo RSEpi foi elaborado para controlar o estoque de equipamentos de proteção Individual (EPI) da empresa, além de gerenciar a entrega dos mesmos. O RSEpi utiliza de biometria para realizar a confirmação de entrega, em que, através do uso de digitais ou veias da palma da mão, o empregado poderá confirmar o recebimento do EPI eletronicamente, sem a necessidade da utilização de papel para armazenar as fichas de entrega. A solução também possui a entrega off-line, em que é possível, utilizando um notebook, realizar a entrega em locais sem conexão com a internet ou com a rede. Depois de finalizada a entrega, o software sincroniza as bases.

O módulo RSWeb foi elaborado para estender as funcionalidades do dataSEESMT na WEB. Inicialmente, o RSWeb havia sido concebido apenas para que os clientes das empresas de consultoria pudessem gerenciar o empregado diretamente pelo módulo. Como a tendência, atualmente, vem a ser a computação nas nuvens, foi decidido que o módulo dataSEESMT seria totalmente na internet e possível de ser acessado por qualquer navegador. Dessa forma, o RSWeb foi reprojetado e será o substituto do dataSEESMT.

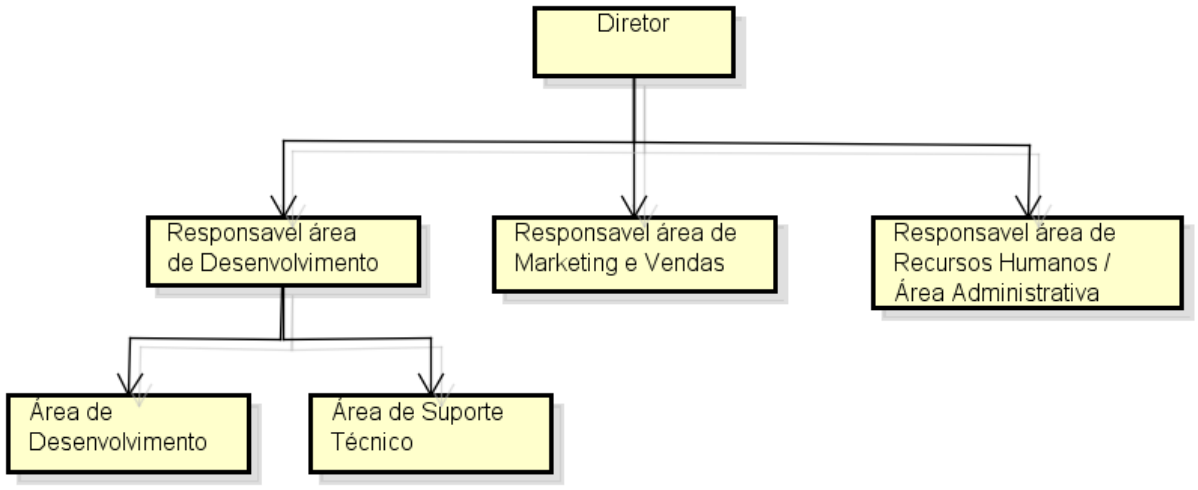
4.1.3 Divisão das equipes

Atualmente a RSData é composta por quatro setores. O setor de desenvolvimento, o setor de suporte técnico, o setor de vendas e o setor de recursos humanos/financeiro. O setor

de desenvolvimento é composto por sete pessoas e suas atividades são a de desenvolvimento de sistemas, manutenção e implantação. O setor de suporte técnico é responsável pelo suporte dos técnicos dos sistemas e é composto por uma pessoa. O setor de vendas é responsável pelas vendas e pelo marketing e é composto por duas pessoas. O setor de recursos humanos/financeiro é responsável pela contratação de novos empregados e cobrança dos títulos dos clientes e é composto por uma pessoa.

A hierarquia da RSData é composta pelo Diretor, seguido pelos responsáveis por cada setor. A Figura 14 apresenta o organograma da empresa.

Figura 14 - Diagrama de hierarquia da empresa



Fonte: Elaborado pelo autor.

Atualmente a RSData possui basicamente duas equipes de desenvolvimento as quais são divididas por tecnologias, sendo elas Delphi e Java. Na Tabela 2, podemos verificar os papéis existentes atualmente nos projetos e suas respectivas responsabilidades.

Tabela 2 – Papéis da equipe

Papel	Qtde	Responsabilidades
Analista	1	Elicitar de requisitos, reunião com clientes, arquitetura dos sistemas. Conhecer modelagem de Banco de dados, UML. Conhecer base de dados Oracle, SQLServer e Firebird.
Desenvolvedor Delphi	3	Conhecer a linguagem Delphi. Conhecer base de dados Oracle, SQLServer e Firebird.
Desenvolvedor Java	3	Conhecer a linguagem JAVA, base de dados Oracle, SQLServer e Firebird. Conhecer os *Frameworks* Hibernate, Spring e Richfaces.
Suporte Técnico	1	Conhecer informática, instalação de sistemas, configuração de rede, compartilhamento de arquivos e comunicativo
Área técnica	1	Conhecer a saúde e segurança do trabalho

Fonte: Elaborado pelo autor.

Na Tabela 3, são apresentados os membros da equipe e seus respectivos papéis. Dessa forma, ficam identificadas as responsabilidades de cada membro da equipe. Como é possível identificar pela Tabela 3, a RSData possui uma equipe enxuta e, por esse motivo, alguns membros possuem mais de um papel.

Tabela 3 - Membros da equipe

Quem	Papéis
Rômulo	Analista, Desenvolvedor Delphi, Desenvolvedor Java.
Dener	Desenvolvedor Java.
Guilherme W.	Desenvolvedor Java.
Ulf	Desenvolvedor Delphi.
Oscar	Desenvolvedor Delphi.
Guilherme S.	Suporte Técnico.
Rodrigo	Área Técnica, Suporte Técnico.

Fonte: Elaborado pelo autor.

A equipe Delphi desenvolve aplicações desktop e cliente servidor, utilizando a linguagem de programação Delphi. São aplicações que trabalham com base de dados Oracle, *SQLServer* e *Firebird*. Essas aplicações foram projetadas para trabalhar com múltiplos usuários em rede, tendo suas principais desvantagens a compatibilidade apenas com o Windows e trabalhar pela internet apenas com o uso de Windows Terminal Server. As aplicações desenvolvidas em Delphi são o dataSEESMT, RSÁudio, RSFinanceiro, RSEpi e a entrega *offline* de EPI. Essas aplicações Delphi serão gradativamente substituídas por aplicações JAVA.

A equipe JAVA, atualmente, trabalha com os *Frameworks Hibernate*, *Spring* e *Richfaces* e está focada em migrar a aplicação Delphi dataSEESMT para Java. A aplicação JAVA também possui suporte na base de dados *Oracle*, *SQLServer* e *Firebird*. Suas principais vantagens em relação aos sistemas desenvolvidos em Delphi são a compatibilidade com o Linux e a possibilidade de serem utilizados via internet sem a necessidade de Windows Terminal Server.

As duas equipes são auxiliadas pela equipe de suporte e área técnica. A equipe de suporte realiza o filtro referente aos chamados. Com esse filtro, somente os chamados que não podem ser resolvidos pelo suporte são encaminhados à equipe de desenvolvimento.

Como a RSData possui parceria com a empresa de consultoria em segurança do trabalho CONSETRA, conta com uma equipe de profissionais que, diariamente, está lidando com as necessidades do mercado e, principalmente, por utilizar o software para realizar a gestão de seus clientes. Dessa forma, é gerado um benefício mútuo no qual a CONSETRA

possui uma ferramenta que atende às suas necessidades e a RSData pode identificar os principais pontos a serem melhorados no sistema. Dessa parceria surgiu a equipe técnica, composta por um Técnico em Segurança da CONSETRA que passou a trabalhar diretamente com a RSData para identificar os principais pontos a serem desenvolvidos no sistema, validar as implementações e centralizar as necessidades da empresa em uma única pessoa que conversa diretamente com o analista.

Como os sistemas em Delphi serão descontinuados, o foco deste trabalho foi o RSWeb, o qual irá substituir aqueles sistemas. Dessa forma, por ser um projeto relativamente novo, assim como uma equipe nova e ainda não possuir um padrão de desenvolvimento definido, espera-se que seja mais fácil realizar a implantação do *Scrum*. Na Tabela 4, podemos verificar as equipes e suas responsabilidades.

Tabela 4 - Equipes

Projeto	Qtde	Responsabilidades
dataSEESMT e demais módulos Delphi	3	Desenvolver e manter as soluções desenvolvidas em Delphi
RSWeb	3	Migrar e desenvolver as novas funcionalidades a solução JAVA

Fonte: Elaborado pelo autor.

4.2 Processo de desenvolvimento

Atualmente, a RSData não possui um padrão de desenvolvimento e delegação de tarefas organizado. A empresa desenvolve um software único para todos os clientes, ou seja, todas as melhorias desenvolvidas para o software são distribuídas a todos os clientes, os quais poderão optar quanto ao uso desta nova funcionalidade.

O processo tem início com o cliente requisitando a melhoria de uma funcionalidade existente ou requisitando o desenvolvimento de uma nova funcionalidade através dos canais de atendimento oferecidos, sendo os canais de atendimento: o e-mail, o telefone ou o contato pessoal com algum membro da equipe.

Todas as requisições, obrigatoriamente, devem passar pelo analista. O analista recebe as requisições e as organiza em uma lista. Cada item da lista é analisado pelo membro da equipe técnica. O analista verifica a relevância da funcionalidade e o impacto que a mesma irá causar, se implementada. Já, o membro da equipe técnica verifica o grau de utilização por

parte dos clientes (a funcionalidade requisitada será útil para outros clientes, além do requisitante) e se a implementação está de acordo com a legislação trabalhista vigente.

Com base nas análises realizadas, o analista decide se a requisição será desenvolvida. Caso não seja desenvolvida, o analista entra em contato com o requisitante e informa os motivos para a recusa. Caso seja desenvolvida, o analista elicita os requisitos com base na requisição e consulta a equipe técnica para eventuais dúvidas, caso o analista necessite de maiores informações, ele pode buscá-las com o requisitante.

Com os requisitos necessários, o analista estima o tempo para o desenvolvimento com base no seu grau de conhecimento da equipe e pelo nível de dificuldade da implementação. Após, o analista pode delegar as tarefas aos desenvolvedores disponíveis. Cada tarefa é delegada a um respectivo desenvolvedor e o analista apresenta a tarefa e seus requisitos a cada desenvolvedor. Neste momento, o desenvolvedor analisa tudo o que lhe foi apresentado e pode sanar eventuais dúvidas que podem vir a bloquear o desenvolvimento do trabalho.

Com as tarefas e seus respectivos requisitos apresentados, o desenvolvedor inicia a implementação. Neste processo, o desenvolvedor pode encontrar dúvidas, que poderão ser sanadas a qualquer momento com o analista. Finalizada a implantação, a tarefa é verificada pelo analista, o qual verifica se os requisitos foram implementados corretamente e se o padrão de telas e de codificação foi seguido pelo desenvolvedor.

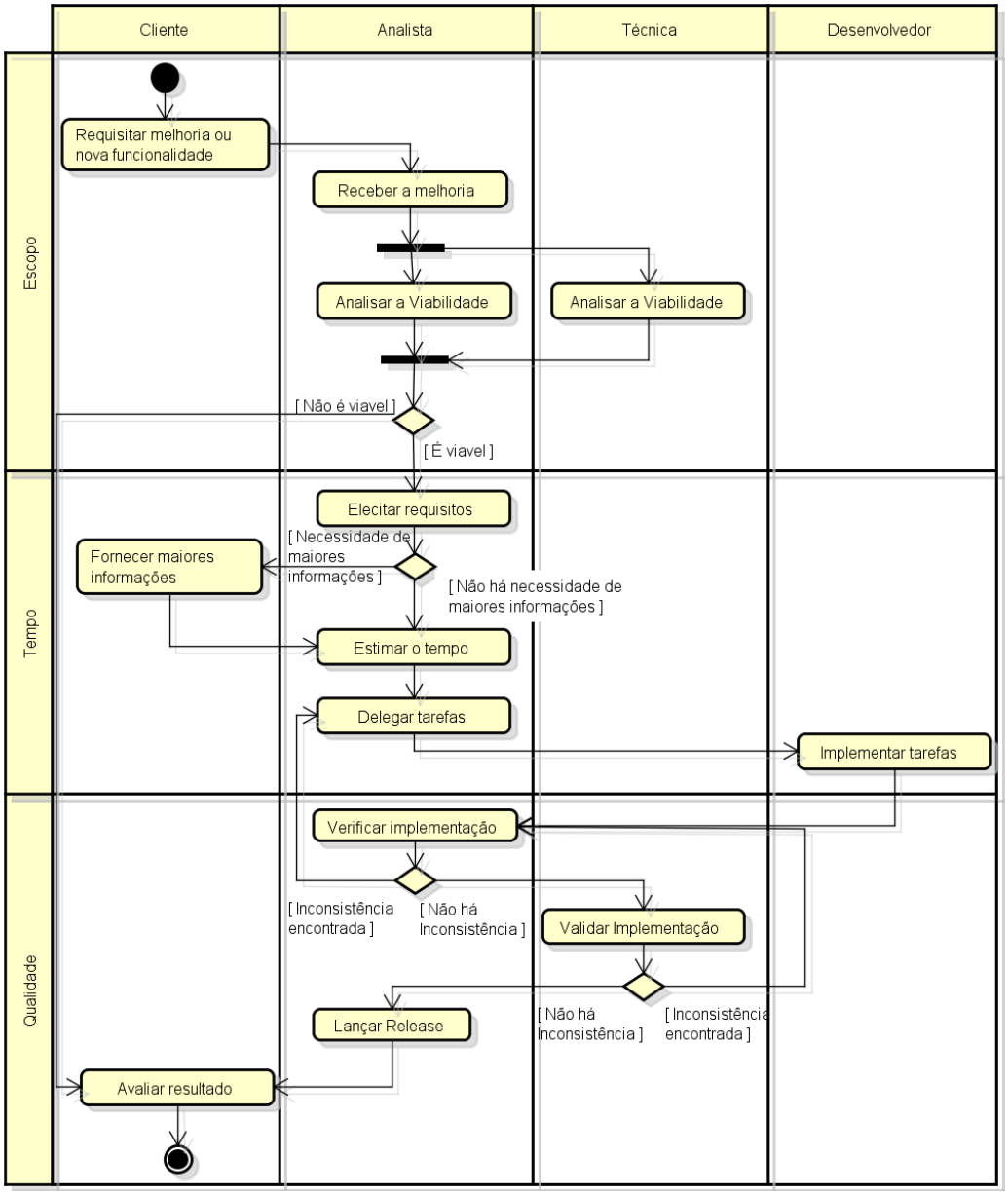
Com a tarefa implementada e verificada, o próximo passo é a validação. Esse passo, normalmente, ocorre um mês antes do lançamento do *release*. A equipe técnica realiza a validação da funcionalidade implementada, verificando se a mesma está de acordo com o que o usuário realmente requisitou.

Após todas as tarefas do *release* serem validadas, o mesmo é lançado. Todos os clientes que requisitaram melhorias ou funcionalidades são comunicados e o analista aguarda que os mesmos analisem o que foi implementado.

O processo descrito nessa seção pode ser visualizado na Figura 15, em forma de diagrama de atividade[3].

[3] É um diagrama que representa os fluxos conduzidos por processamentos, mostrando o fluxo de controle de uma atividade paraoutra.

Figura 15 - Diagrama de atividade do processo de desenvolvimento de software



Fonte: Elaborado pelo autor.

Os processos da gestão de escopo, gestão de tempo e gestão de qualidade são descritos nas próximas subseções deste capítulo.

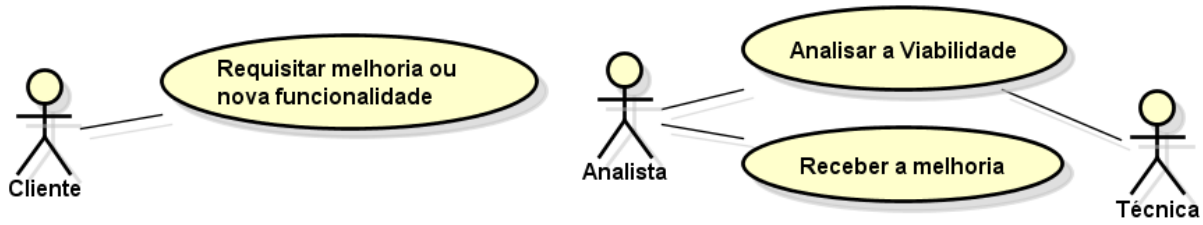
4.2.1 Gestão de escopo

No atual padrão organizacional utilizado pela empresa, o membro responsável por gerenciar o escopo é o analista. O analista realiza o planejamento do escopo para *releases*, os

quais podem ser lançados aos clientes em um período de seis meses a um ano, dependendo da quantidade de requisitos a serem desenvolvidos para o *release*.

A Figura 16 apresenta os atores e suas respectivas tarefas no processo de gestão de escopo realizado atualmente pela empresa. O processo de gestão de escopo, resumidamente, é representado pelo cliente, o qual inicia o processo requisitando uma nova funcionalidade a qual é recebida pelo analista. Por fim, esta demanda tem sua viabilidade analisada pelo analista e pela técnica.

Figura 16 - Atores e suas respectivas atividades na gestão de escopo



Fonte: Elaborado pelo autor.

Nas subseções seguintes, são apresentados, detalhadamente, os processos de elicitação, planejamento, decomposição, documentação e gerência referentes à gestão de escopo que vêm sendo praticados atualmente pela empresa.

4.2.1.1 Elicitação

As demandas podem ser originadas por necessidades de clientes, necessidades da equipe técnica ou por demandas originadas pela equipe de desenvolvimento. As demandas dos clientes, requisição de melhorias ou novas funcionalidades são encaminhadas pelos mesmos através de e-mail, telefone ou contato com o suporte técnico. Tanto as demandas dos clientes como as demandas internas (equipe técnica ou equipe de desenvolvimento) são encaminhadas ao analista.

4.2.1.2 Planejamento

Todas as demandas são centralizadas no analista. O analista organiza as demandas em uma lista de tarefas a serem desenvolvidas. Cada nova tarefa é analisada ao decorrer do tempo

e então são identificadas quais serão desenvolvidas para o próximo *release* e quais não. O processo de seleção consiste basicamente em o analista verificar a relevância da funcionalidade e o impacto que a mesma irá causar se implementada. Ficando ao membro da equipe técnica verificar o grau de utilização por parte dos clientes (a funcionalidade requisitada será útil para outros clientes, além do requisitante) e se a implementação está de acordo com a legislação trabalhista vigente.

Após o início do desenvolvimento do novo *release*, com o escopo já em desenvolvimento, novas funcionalidades podem ser requisitadas. Durante o processo, tanto o analista quanto a equipe técnica verificam o grau de importância dessa requisição e então identificam se há possibilidade de incluí-la no *release* em andamento. Em virtude do nível de importância da funcionalidade, a mesma poderá ser inclusa no *release* e, dependendo do nível de impacto dessa implementação, a data de lançamento do *release* poderá ser alterada.

Quanto aos requisitos da funcionalidade, inicialmente, quando ela é analisada, os requisitos necessários são levantados, caso falte algum ou ele não esteja bem especificado, o analista entra em contato com o solicitante para esclarecer as questões. Os requisitos ficam registrados junto ao descritivo da implementação e serão passados ao desenvolvedor, quando o mesmo iniciar o desenvolvimento da tarefa.

4.2.1.3 Decomposição

Devido ao fato do desenvolvimento de um *release* durar entre seis meses a um ano, a maioria das tarefas não necessitam de decomposição. O desenvolvedor alocado para desenvolver uma tarefa específica, irá desenvolvê-la até concluí-la. Isso se deve ao fato de, no processo anterior a implantação, o qual não era utilizado o controle de verões, forçava os desenvolvedores a utilizarem o mesmo arquivo de código fonte para trabalhar, dessa forma, os mesmos não podem trabalhar com os mesmos arquivos. E, como o tempo para o lançamento de *release* é relativamente extenso, é possível trabalhar sem que haja a necessidade de decomposição.

O analista abre uma exceção para tarefas importantes que não poderão ser implementadas em um *release* único e que possam ser fracionadas. Por exemplo, a elaboração de um relatório complexo que necessite também de um cadastro complexo: o analista poderá

fracioná-lo, desenvolvendo o cadastro do relatório em *release* e a implementação do relatório ficará para o próximo *release*.

4.2.1.4 Documentação

Nenhum dos processos executados na gestão de escopo gera artefatos.

4.2.1.5 Gerência

As tarefas do escopo são cadastradas e gerenciadas em um sistema de controle de tarefas e projetos chamados *dotProject*[4]. O *dotProject* é utilizado para registro das tarefas. Nesse sistema, não são mantidos históricos de alterações de requisitos, apenas são mantidos os requisitos finais, quem os implementou e o tempo necessário para a implementação ser concluída.

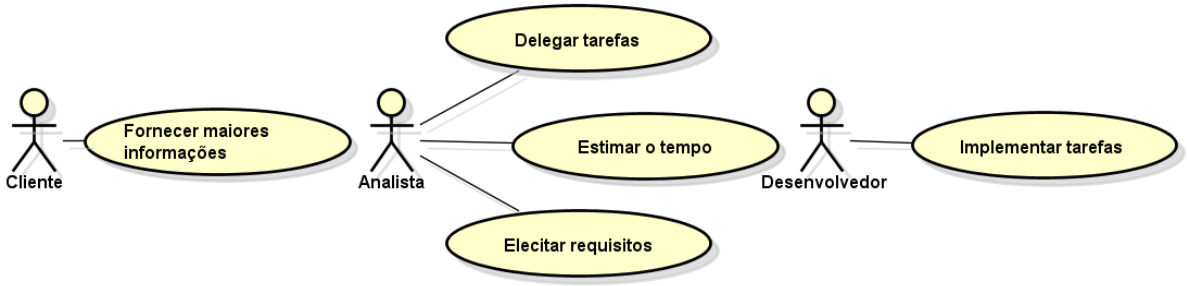
4.2.2 Gestão de tempo

No atual padrão organizacional utilizado pela empresa, o membro responsável pelo gerenciamento de tempo é o analista, contudo ele não realiza um planejamento e controle de cronograma rigoroso, pois seu principal objetivo é realizar *releases* a cada seis meses ou anualmente. Dessa forma, ele possui uma segurança maior para garantir que o produto lançado tenha qualidade, pois irá contar com um período estendido para desenvolvimento e testes.

A Figura 17 apresenta os atores e suas respectivas tarefas no processo de gestão de tempo realizado atualmente pela empresa. O processo de gestão de tempo, resumidamente, é representado pelo analista, o qual elicita os requisitos da tarefa. Sendo necessárias maiores informações, o analista contata o cliente, o qual fornece as informações necessárias. Com os requisitos, o analista estima o tempo necessário para a realização da tarefa e então a delega. Por fim, o desenvolvedor implementa a tarefa.

[4] Sistema de gerência de projetos. Site oficial http://www.dotproject.net/.

Figura 17 - Atores e suas respectivas atividades na gestão de tempo



Fonte: Elaborado pelo autor.

Nas subseções seguintes, são apresentados, detalhadamente, os processos de definição de atividades, estimativa de tempo, sequenciamento de atividades, alocação e controle e execução referentes à gestão de tempo que vêm sendo praticados atualmente pela empresa.

4.2.2.1 Definição das atividades

As tarefas foram definidas no planejamento da gestão de escopo.

4.2.2.2 Estimativas de tempo

A estimativa de tempo para a realização da tarefa também é definida pelo analista. O analista estima o tempo necessário para o desenvolvimento de cada tarefa com base na experiência e pelo histórico de tarefas desenvolvidas pelo desenvolvedor.

Todas as tarefas realizadas pelos desenvolvedores são armazenados no sistema de gerenciamento de projetos. Cada tarefa possui um breve histórico do que foi implementado, assim como quem a realizou e o tempo real necessário para implementá-la. Com base nestes históricos o analista é capaz de estimar com segurança o tempo para conclusão de cada tarefa por cada desenvolvedor.

4.2.2.3 Sequenciamento das atividades

O analista realiza o processo de sequenciamento das atividades de acordo com o nível de importância para o projeto, levando em conta a dependência para a realização das demais.

Dessa forma, as atividades que venham a bloquear o desenvolvimento são desenvolvidas com antecedência.

4.2.2.4 Alocação

A alocação das tarefas é realizada pelo analista. O desenvolvedor, ao finalizar uma atividade, dirige-se ao analista o qual seleciona uma nova atividade a ser desenvolvida, informa ao desenvolvedor e ao mesmo tempo lhe explica os requisitos desta tarefa para a implementação. Ao finalizar a atividade, o desenvolvedor se dirige ao analista novamente, dando novo início ao ciclo.

4.2.2.5 Controle e execução

O cronograma não é controlado, pois as tarefas não possuem uma sequência pré-definida e nem um período de tempo pré-estabelecido para serem desenvolvidas. Como citado na seção anterior, as atividades são delegadas conforme os desenvolvedores terminam as atividades que estão desenvolvendo.

O controle realizado é sobre as tarefas desenvolvidas. Com o *dotProject*, o analista define a tarefa que o desenvolvedor deve realizar. Ao concluir a tarefa, o desenvolvedor define o tempo médio que levou para desenvolver a atividade, descreve resumidamente a atividade realizada e o tempo necessário para a conclusão da tarefa. Dessa forma, o analista possui um histórico de atividades completas e o tempo que cada desenvolvedor levou para desenvolver a sua tarefa.

Nenhum dos processos executados na gestão de tempo gera artefatos. Apenas há o registro resumido das atividades desenvolvidas, quem as desenvolveu, o tempo levado para a conclusão e um resumo do que foi desenvolvido.

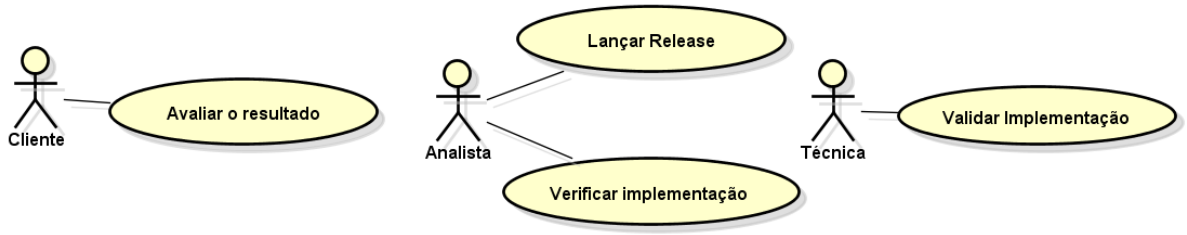
4.2.3 Gestão de qualidade

No atual padrão organizacional utilizado pela empresa, os membros responsáveis pelo gerenciamento de qualidade são o analista e a equipe técnica.

O controle de mudanças é realizado pelo *dotProject* apenas para manter um histórico de tarefas que foram realizadas, quem as realizou e o tempo necessário para o desenvolvimento, assim como as tarefas a serem desenvolvidas. Não é feito uso de integração contínua, assim como não é feito uso de controle de versão. O controle de versão é realizado dentro de um diretório compartilhado, no qual são encontradas as versões do software. Quando há uma nova versão, a pasta é copiada e renomeada. Caso haja inconsistências a serem corrigidas na versão em produção, as correções deverão ser copiadas para a versão em desenvolvimento, evitando, dessa forma, que o problema corrigido na versão anterior não seja corrigido novamente na versão em desenvolvimento.

A Figura 18 apresenta os atores e suas respectivas tarefas no processo de gestão de qualidade realizado atualmente pela empresa. O processo de gestão de qualidade resumidamente é representado pelo analista, o qual verifica a tarefa que foi implantada pelo desenvolvedor. A equipe técnica se responsabiliza por validar as tarefas realizadas antes do lançamento do *release*. Com as tarefas validadas, o analista lança o *release*, comunica o cliente e o cliente avalia o resultado da implementação.

Figura 18 - Atores e suas respectivas atividades na gestão de qualidade



Fonte: Elaborado pelo autor

Nas subseções seguintes, são apresentados, detalhadamente, os processos de verificação do produto, verificação do processo, validação e teste referentes à gestão de qualidade que vêm sendo praticados atualmente pela empresa.

4.2.3.1 Verificações do produto

Os testes de verificação do produto, os quais verificam se os requisitos foram implementados com sucesso, são realizados pelo analista após o desenvolvedor finalizar o desenvolvimento da tarefa. O analista testa as funcionalidades para identificar se o mesmo implementou-as com sucesso, caso haja alguma inconsistência na implementação, o desenvolvedor é avisado para realizar a correção.

4.2.3.2 Verificações do processo

As inspeções no código fonte do desenvolvedor apenas são realizadas nos primeiros meses de trabalho, com a finalidade de identificar se o mesmo vem seguindo o padrão de desenvolvimento do projeto. Após o analista identificar que o desenvolvedor se adaptou ao padrão, esses testes não são mais realizados.

4.2.3.3 Validação

No mês que antecede ao lançamento do *release*, a equipe técnica realiza os testes exploratórios pelas funcionalidades do software para verificar se as atividades desenvolvidas no escopo estão corretas e para validar as atividades. Neste período, também são corrigidas as falhas encontradas para preparar o software para o lançamento.

Cada inconsistência detectada pela equipe técnica é encaminhada ao analista. O mesmo fica encarregado de verificar o requisito inconsistente e identificar se realmente é um problema. O analista, identificando o problema, aloca um desenvolvedor para ser responsável pela correção.

4.2.3.4 Testes

Todos os testes realizados são de forma manual. No atual padrão utilizado pela empresa, não é feito o uso de testes unitários ou automatizados. Também não são utilizados técnicas para mensurar a evolução da equipe.

4.3 Equipe após adoção

No processo de desenvolvimento, após a implantação do Scrum, foi mantida a hierarquia da empresa composta pelo diretor da empresa, seguido pelos responsáveis de cada setor. Sendo que a principal mudança ocorreu dentro da equipe do produto RSWeb.

Na Tabela 5, são listados os papéis que foram criados após a implantação do *Scrum*. Nessa tabela, também é informado o número de membros que participa em cada papel, além de conter uma breve descrição das atividades realizadas em cada papel.

Tabela 5 - Papéis da equipe

Papel	Qtde	Responsabilidades
Project Owner	1	Elicitar de requisitos, reunião com clientes, Priorizar e gerencia o *Product Backlog*.
Scrum Master	1	Assegurar que a equipe respeite e siga os valores e as práticas do Scrum. Evitar que problemas externos afetem a produtividade. Evitar que a equipe se comprometa com mais do que possa realizar no Sprint.
Team	4	Conhecer a linguagem JAVA, base de dados Oracle, SQLServer e Firebird. Conhecer os *Frameworks* Hibernate, Spring e Richfaces.
Suporte Técnico	1	Conhecer informática, instalação de sistemas, configuração de rede, compartilhamento de arquivos e comunicativo
Qualidade	1	Conhecer técnicas para teste de software. Conhececer o sistema.

Fonte: Elaborado pelo autor.

Na Tabela 6 podemos identificar o papel que cada membro da equipe desenvolvia antes do processo Scrum e o papel que passou a desempenhar após o processo.

Tabela 6 - Papéis antes e depois do Scrum

Quem	Papéis antes Scrum	Papel após Scrum
Rômulo	Analista, Desenvolvedor Delphi e Desenvolvedor Java.	*Scrum Master*
Dener	Desenvolvedor Java.	*Team*
Guilherme W.	Desenvolvedor Java.	*Team*
Ulf	Desenvolvedor Delphi.	Desenvolvedor Delphi.
Oscar	Desenvolvedor Delphi.	Desenvolvedor Delphi.
Guilherme S.	Suporte Técnico.	QA
Rodrigo	Área Técnica e Suporte Técnico.	*Product Owner*

Fonte: Elaborado pelo autor.

Após a adoção do *Scrum*, a equipe conquistou maior liberdade para implementar as soluções. O que antes era papel unicamente do analista, passa a fazer parte de toda a equipe. Durante a primeira parte da reunião de planejamento dos *Sprints*, todos os membros presentes

realizam apontamentos referentes às demandas e, na segunda parte, os membros do *Team* analisam a melhor forma para realizar a implementação das tarefas e a documentam.

4.4 Processo após adoção

O processo tem início com o cliente requisitando a melhoria de uma funcionalidade existente ou requisitando o desenvolvimento de uma nova funcionalidade através dos canais de atendimento oferecidos:

- e-mail;
- telefone;
- contato pessoal com algum membro da equipe.

Todas as requisições, obrigatoriamente, devem passar pelo *Product Owner*, o qual é responsável pelo *Product Backlog*. O *Product Backlog* é refinado e priorizado diariamente com base no ROI de cada item. Os itens que possuem um ROI maior serão priorizados.

Com os itens do *Product Backlog* priorizados, é realizado o *Sprint Planning*. Esta reunião é dividida em duas partes. Na primeira parte, o *Product Owner*, *Scrum Master* e *Team* se reúnem com objetivo de identificar os itens de maior prioridade do *Backlog*, ou seja, o *Product Owner* passa para a equipe a visão do produto, informando maiores detalhes de cada demanda. Nessa reunião, é importante que o *Team* tire todas as dúvidas referentes às demandas com maior prioridade e o *Product Owner* detalha as demandas mais importantes, quebrando-as em partes menores.

Na segunda parte da reunião, o *Scrum Master* e a equipe se reúnem para detalhar as demandas priorizadas na primeira parte da reunião e o *Team* verifica o que poderá ser realizado durante o *Sprint* e se compromete a realizá-las. Nessa reunião, o *Team* quebra as *User Stories*[5] em partes menores e as detalha em conjunto, a fim de implementá-las com detalhes.

Ao final do *Sprint Planning*, o *Sprint Backlog* está composto e a equipe poderá começar a realizar a implementação das demandas no *Sprint*. No processo utilizado pela empresa estudada, o *Sprint* é realizado em duas semanas. Durante o *Sprint*, diariamente, é

[5]Uma descrição da funcionalidade desejada, pelo ponto de vista do *stakeholder*.

realizado o *Daily Meeting*, quando o *Team* expõe as principais dificuldades encontradas no processo ao *Scrum Master*. Sendo o *Scrum Master* responsável por resolver os problemas descritos pelo *Team*.

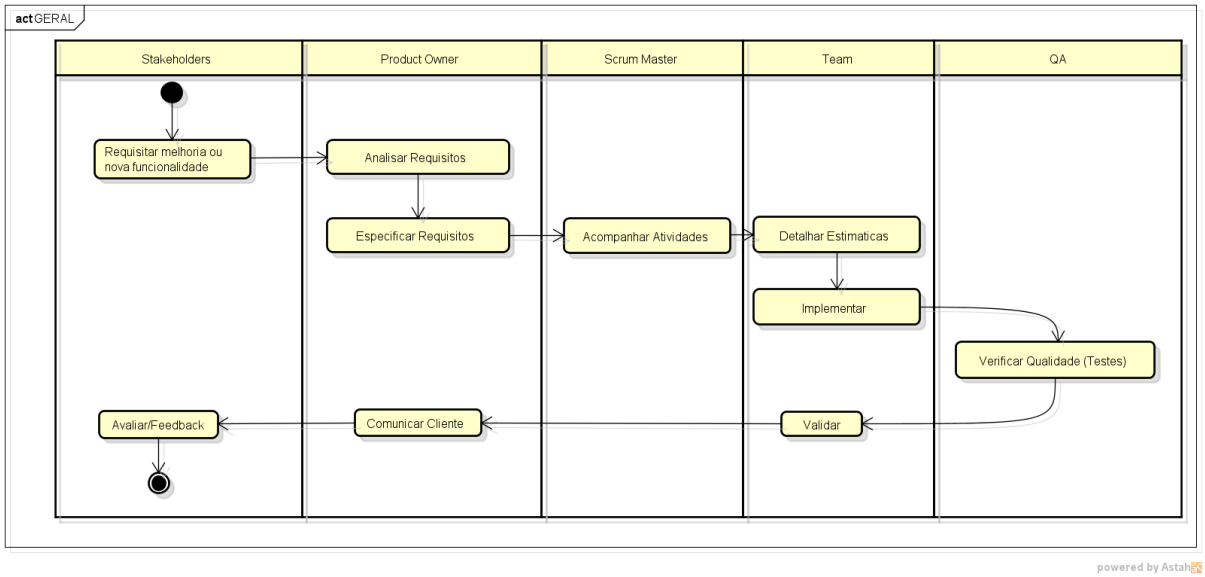
Cada membro do *Team* verifica o *Sprint Backlog*, identifica uma demanda priorizada e, de comum acordo, às implementam. Após finalizar, o membro do *Team* responsável, atualiza o status do *Sprint Backlog*, informando que a demanda foi realizada. Havendo itens pendentes, o membro do *Team* verifica uma nova a ser realizada e reinicia o processo.

A cada demanda realizada, o membro do *Team* atualiza o status no quadro para "Revisão". Cada uma atualizada para "Revisão" entra na fila de testes. Com base na descrição detalhada, a pessoa responsável realiza os testes de interface e de negócio. Caso haja alguma falha nos testes, o encarregado do teste cria uma nova demanda com status "*Bug*" fora do *Sprint Backlog* para que o usuário responsável possa corrigir, assim que possível. O processo é realizado até o final do *Sprint*.

Ao final do *Sprint*, com todas as demandas realizadas, ou ao final da *dead line*, o *Team* e o *Scrum Master* se reúnem e realizam o *Sprint Review* e *Sprint Retrospective*. Sendo uma reunião, cujo principal objetivo é analisar o que foi realizado no *Sprint* e indicar melhorias aos processos. Nesta última reunião, é verificado o que foi implantado e decidido se será ou não disponibilizado um *release* aos clientes. Caso seja realizado o *release* ao cliente, o mesmo é comunicado para avaliar a implementação.

O processo descrito nesta seção pode ser visualizado na Figura 19, em forma de diagrama de atividade.

Figura 19 - Novo processo de desenvolvimento



Fonte: Elaborado pelo autor.

4.4.1 Gestão de configuração

Embora o *Scrum* não necessite da gestão de mudanças para funcionar corretamente, é imprescindível que, atualmente, as equipes ágeis façam uso de tais ferramentas devido a sua necessidade de agilizar os processos e também poder evoluir de maneira controlada em um ambiente em constante mudança.

A Tabela 7 apresenta os papéis dos atores envolvidos na gestão de configuração e as responsabilidades executadas por cada um deles.

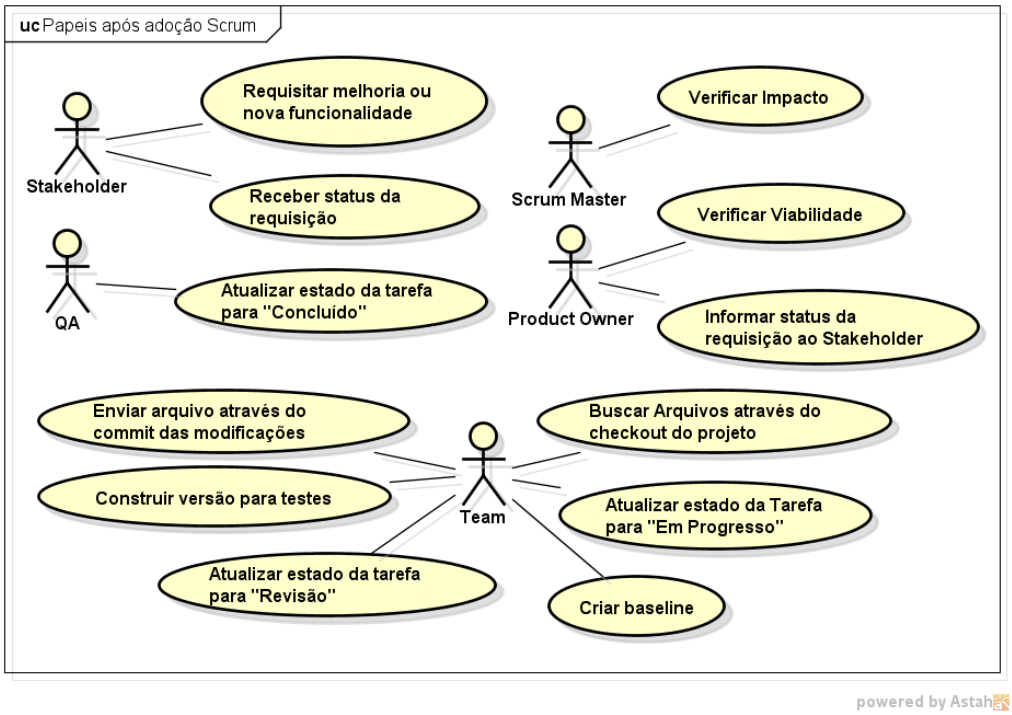
Tabela 7 - Responsabilidades dos papéis na gestão de configuração

Papel	Responsabilidades
Product Owner	Verificar viabilidade;
Scrum Master	Verificar Impacto;
Stakeholder	Requisitar nova melhoria ou funcionalidade;
QA	Atualizar o estado da tarefa para “Concluído”;
Team	Enviar arquivo através do *commit* das modificações, construir versão para testes, atualizar o estado da tarefa para “Em progresso” e “Revisão”, criar TAG e buscar arquivos através do *checkout* do projeto.

Fonte: Elaborado pelo autor.

A Figura 20 apresenta os atores e suas respectivas demandas no processo de gestão de configuração. O processo de gestão de escopo, resumidamente, é representado pelo *stakeholder*, o qual inicia o processo requisitando uma nova funcionalidade a qual é recebida pelo *Product Owner*, o qual verifica a viabilidade da requisição. Nesse processo, o *Team* possui responsabilidades apenas de manutenção realizada com apoio do *Subversion*.

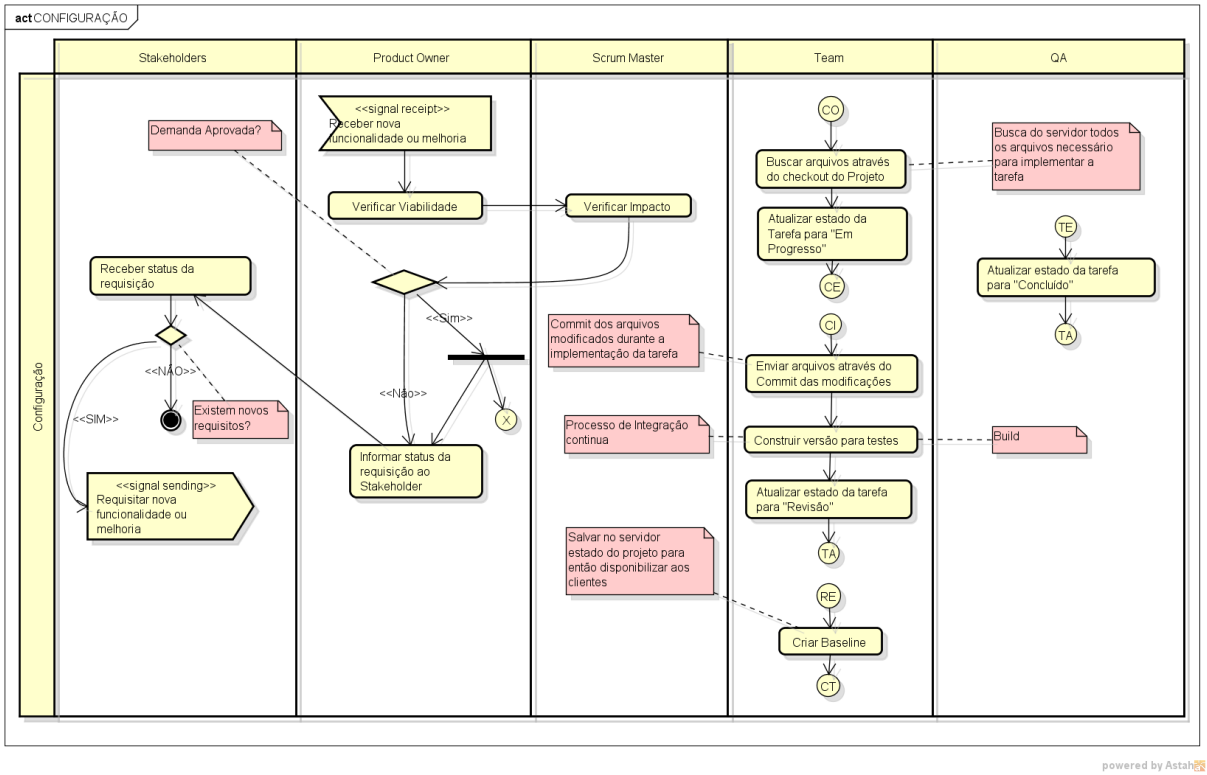
Figura 20 - Papéis envolvidos na gestão de configuração



Fonte: Elaborado pelo autor.

A Figura 21 apresenta o diagrama de atividades do processo de gestão de configuração.

Figura 21 - Diagrama de atividade da Gestão de Configuração



Fonte: Elaborado pelo autor.

Atualmente, grande parte das equipes que faz uso de metodologias ágeis, utiliza alguma ferramenta para gestão de mudanças. Podemos citar o controle de mudanças, o controle de versão e a integração contínua como sendo as principais ferramentas de gestão de mudanças utilizadas pela empresa do estudo.

Com o controle de versões, podemos saber quais áreas do *software* foram alteradas, quando as alterações ocorreram e quem as realizou. Já, com o controle de mudanças, podemos rastrear as necessidades que originaram essas mudanças e quem as solicitou. E, com a integração contínua, podemos realizar a construção do projeto a cada mudança que algum desenvolvedor realizar, atualizar o ambiente de testes com a versão mais recente do *software* e descobrir possíveis erros no desenvolvimento.

4.4.1.1 Controle de mudanças

A ferramenta para controle de mudanças escolhido para o projeto foi o JIRA[6]. O JIRA foi escolhido, pois, em comparação com as demais ferramentas do mercado, foi a que se

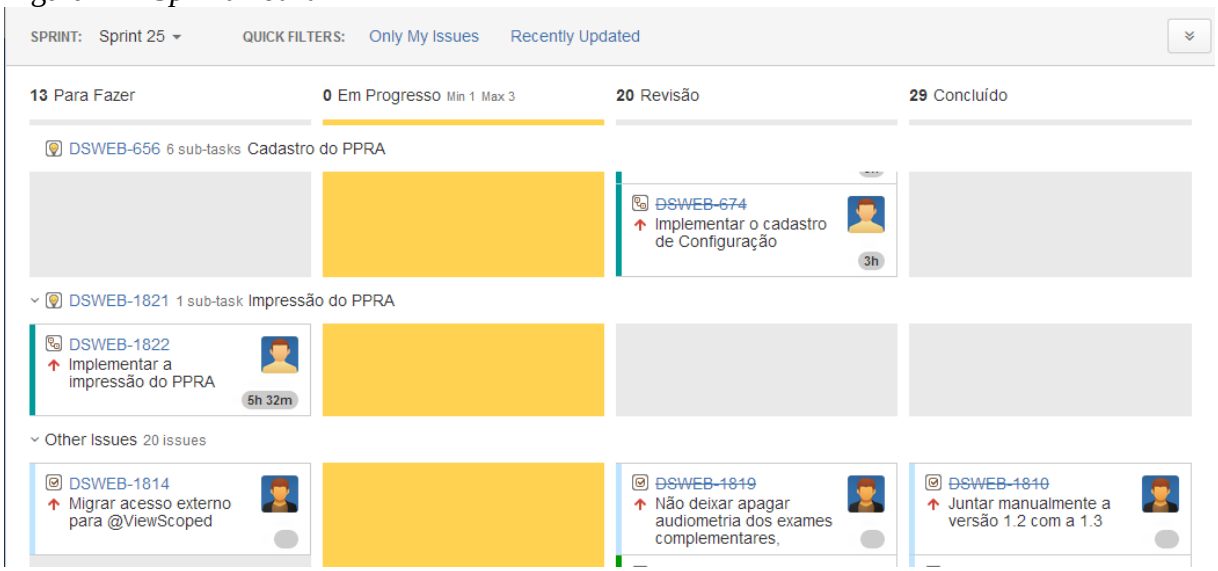
[6]É um programa de gestão de projetos e acompanhamento de projetos, demandas e erros. Mais informações https://www.atlassian.com/.

adaptou da melhor forma possível com o processo de desenvolvimento da empresa após o processo de implantação do *Scrum*.

Isso se deve ao fato do JIRA possuir um *plugin* para desenvolvimento ágil chamado de JIRA Agile[7], o qual permite o gerenciamento de vários projetos ágeis pela ferramenta desde o processo de gerenciamento do *Product Backlog* até o processo de acompanhamento do *Sprint Board*.

A Figura 22 apresenta o *Sprint Board* utilizado pela empresa foco do estudo. Na primeira coluna da esquerda, "Para Fazer", são apresentadas todas as tarefas que deverão ser realizadas durante o *Sprint*. Na coluna ao lado, "Em progresso", são apresentadas as tarefas que estão sendo implementadas e também é apresentada uma imagem referente ao desenvolvedor que a esta implementando, possibilitando identificar rapidamente em quais tarefas cada desenvolvedor está trabalhando. Na coluna "Revisão" são apresentadas todas as tarefas já implementadas pelos desenvolvedores e que estão aguardando a Equipe de testes para testá-los. Com esta coluna, a equipe de testes identifica rapidamente quais tarefas estão aptas a serem testadas, ou seja, quais tarefas foram implementadas e testas pelos desenvolvedores. Já a última coluna, "Concluído" apresenta todas as tarefas implementadas no Scrum e quem a implementou. Possibilitando que toda a equipe identifique rapidamente quais tarefas foram implementas e passaram por todos os testes necessários.

Figura 22 - *Sprint Board*



Fonte: Elaborado pelo autor.

[7] Software para planejamento de projetos ágeis da Atlassian.Mais informações https://www.atlassian.com/.

O JIRA se tornou uma ferramenta essencial após o processo de implantação do *Scrum*. Todos os membros da equipe possuem um cadastro no JIRA. O PO realiza o cadastramento de todas as demandas no *Product Backlog* do JIRA para o projeto em questão. A própria ferramenta possibilita a decomposição das demandas, assim o PO pode cadastrar as tarefas entre *EPIC*[8], *User Story* ou tarefa.

Todas as demandas cadastradas no sistema, depois de decompostas, podem ser priorizadas de acordo com o nível de importância. No JIRA, o *backlog* é representado por uma lista de demandas, das quais, as que estão no topo são as de maior importância. Dessa forma, o PO as organiza pela importância.

Na reunião do *Sprint Planning*, o PO, juntamente com o *Scrum Master* e o *Team*, consultam as tarefas que estão cadastradas no *Product Backlog* do JIRA e identificam as tarefas que farão parte do *Sprint*. O *Team* as decompõe, sendo adicionadas as estimativas de tempo para as tarefas e então é criado o *Sprint*.

Após a criação do *Sprint*, todos os membros do *Team* têm acesso ao *Sprint Board* e podem começar a trabalhar nas tarefas.

O acompanhamento das tarefas é realizado pelo *Sprint Board*, o qual é oferecido pela ferramenta de controle de mudanças. Todos os membros tem o endereço do *Sprint Board* como página inicial em seus navegadores. Por este motivo não é utilizado o quadro tradicional na parede.

Cada linha do *Sprint Board* representa uma história e, nela, devem ser incluídas todas as tarefas necessárias para que aquela história esteja pronta. Inicialmente, as tarefas são inseridas na primeira coluna do quadro e, à medida que vão sendo desenvolvidas, vão avançando até chegarem à última coluna.

As raias do quadro, que são as colunas traçadas, são os estados em que uma determinada tarefa pode estar. A quantidade e a descrição de cada uma podem variar de acordo com seu processo de desenvolvimento. No processo de implantação, a equipe optou por utilizar 4 colunas:

- "Para fazer" (*TO DO*);

[8]Uma *user story* muito grande, normalmente envolvendo mais de um *Sprint*.

- “Em Progresso” (*DOING*);
- “Revisão” (*TEST*);
- “Concluído” (*DONE*).

Dessa forma, ao olhar o quadro, a pessoa é capaz de, facilmente, perceber como está o andamento do trabalho, sendo acessível de qualquer lugar pela internet, através do JIRA.

O JIRA possui uma funcionalidade de integração das tarefas com o Eclipse. O Eclipse é a IDE de desenvolvimento utilizado pelo *Team* e, através dessa ferramenta, os membros do *Team* passam a ter acesso a todas as demandas que estão no JIRA dentro da IDE.

Com essa funcionalidade, os membros do *Team* podem iniciar o trabalho nas tarefas sem a necessidade de consultar o JIRA. Ao iniciar o trabalho de uma tarefa pelo Eclipse, o status da mesma é alterado no JIRA.

Além de possibilitar o gerenciamento das tarefas pela própria IDE, a ferramenta possibilita que, ao atualizar o status da tarefa, o membro da equipe possa salvar o seu estado. Ou seja, salvar uma lista de todos os arquivos que foram trabalhados durante uma tarefa. Havendo a necessidade de outra pessoa abrir a tarefa, a IDE irá apresentar todos os arquivos que foram afetados, facilitando a manutenção de tarefas que possam ser trabalhados por mais de uma pessoa em períodos de tempo distintos.

4.4.1.2 Controle de versão

O controle de versão é utilizado principalmente pela necessidade de atualização simultânea de código fonte por mais de um desenvolvedor, para que todos os desenvolvedores do projeto sejam alertados de mudanças no código fonte e também para evitar que existam várias versões do software sendo desenvolvidos em simultâneo. Problemas como defeitos que já foram corrigidos e voltam a acontecer, funcionalidades implementadas e testadas que param de funcionar ou indefinição sobre o que deve ou não ser disponibilizado em produção são comuns em um ambiente sem um controle adequado.

O controle de versão possibilita a gerência das diferentes versões do projeto. Além disso, durante o processo de desenvolvimento do software, é formado um histórico completo de todas as alterações efetuadas, possibilitando ainda a comparação entre versões,

identificação dos responsáveis pelas alterações, marcação de versões específicas e ramificações do projeto.

Com o uso do controle de versão é possível trabalhar com duas versões em paralelo, uma versão em produção e outra versão de desenvolvimento. A correção de erros encontrados na versão de produção pode ser aplicada em forma de *patch* na versão em desenvolvimento.

No processo pós *Scrum*, foi adotada a ferramenta *Subversion* para realizar o controle de versões juntamente com o software de Gerenciamento de Código Fonte da Atlassian, o *Fisheye*[9].

O *Subversion* foi escolhido, pois a maioria dos membros da equipe já estava familiarizada com a ferramenta e ela permite controlar as versões do código fonte da forma desejada para o projeto.

Atualmente, a equipe utiliza o *Subversion* principalmente para manter o histórico de alterações do código fonte, além de trabalhar em duas versões ao mesmo tempo. A versão de produção, que todos os clientes estão utilizando e a versão de desenvolvimento que é a versão na qual as novas funcionalidades vêm sendo implantadas.

O fluxo de trabalho do *Subversion* adotado pela equipe consiste em, basicamente, antes de iniciar o processo de implementação, realizar o *checkout*[10] do projeto, para então iniciar o processo de implementação. Finalizado o processo de implementação, o desenvolvedor deve realizar o processo de *commit*[11].

Para controlar o repositório é utilizado o *FishEye*. O *Fisheye* é uma ferramenta que possui integração com o JIRA e com ela é possível identificar todas as mudanças realizadas pelos membros do *Team* e, identificando em quais tarefas as mudanças estão associadas e receber, por e-mail, notificações de mudanças no código fonte.

[9] Software para gerenciamento de código fonte. Mais informações https://www.atlassian.com/fisheye.
[10] Carrega uma cópia de trabalho a partir do repositório.
[11] Envia as alterações de sua cópia de trabalho para o repositório.

4.4.1.3 Integração contínua

Integração Contínua é uma prática de desenvolvimento de software em que os membros de um time integram seu trabalho frequentemente, geralmente cada pessoa integra pelo menos diariamente – podendo haver múltiplas integrações por dia. Cada integração é verificada por um *build* automatizado (incluindo testes) para detectar erros de integração o mais rápido possível.

A grande vantagem da integração contínua está no *feedback* instantâneo. O processo funciona da seguinte forma: a cada *commit* no repositório, o *build* é realizado automaticamente com todos os testes sendo executados de forma automática e falhas sendo detectadas. Se algum *commit* não compilar ou quebrar qualquer um dos testes, a equipe toma conhecimento instantaneamente (através de e-mail, por exemplo, indicando as falhas e o *commit* causador das mesmas). A equipe pode, então, corrigir o problema o mais rápido possível, o que é fundamental para não introduzir erros ao criar novas funcionalidades, refatorar, etc. Integração contínua é mais uma forma de trazer segurança em relação a mudanças: você pode fazer modificações sem medo, pois será avisado caso algo esteja errado.

A ferramenta de integração contínua escolhida para o projeto é o *bamboo*[12]. A escolha se deu, devido ao fato do *bamboo* trabalhar juntamente com o JIRA. O que acaba trazendo algumas vantagens. Sendo elas a criação de *Branches* automáticos a cada nova versão criada no JIRA e a criação de uma *Tag*, no *Subversion*, cada vez que uma versão é lançada através do JIRA.

4.4.2 Gestão de escopo

No processo de desenvolvimento após a implantação do *Scrum*, o membro responsável por gerenciar o escopo é o *Product Owner*. O PO é responsável por gerenciar o escopo do produto e priorizar as demandas de acordo com as necessidades dos *stakeholders*.

A Tabela 8 apresenta os papéis dos atores envolvidos na gestão de escopo e as responsabilidades executadas por cada um deles.

[12] Software para integração contínua. Mais informações https://www.atlassian.com/bamboo.

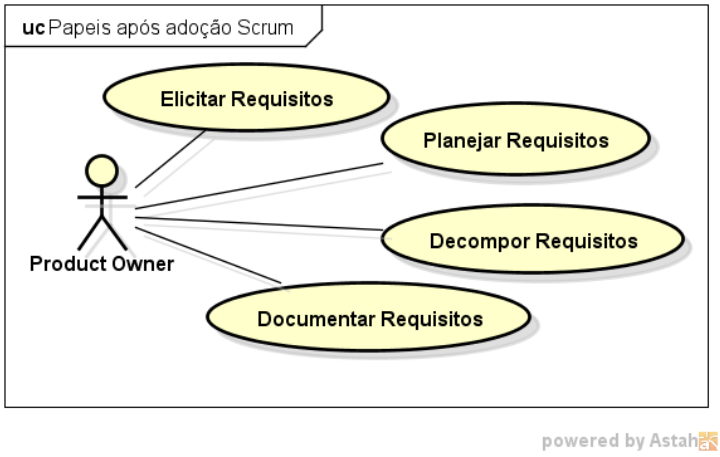
Tabela 8 - Responsabilidades dos papéis na gestão de escopo

Papel	Responsabilidades
Product Owner	Elicitar os requisitos das demandas, planejar o escopo, decompor as demandas, documentar e gerenciar.

Fonte: Elaborado pelo autor.

A Figura 23 apresenta os atores e suas respectivas demandas no processo de gestão de escopo. O processo de gestão de escopo, resumidamente, é representado pelas requisições aprovadas na gerência de configuração. Essas funcionalidades são recebidas pelo *Product Owner*, o qual prioriza a demanda e reorganiza o *Product Backlog*.

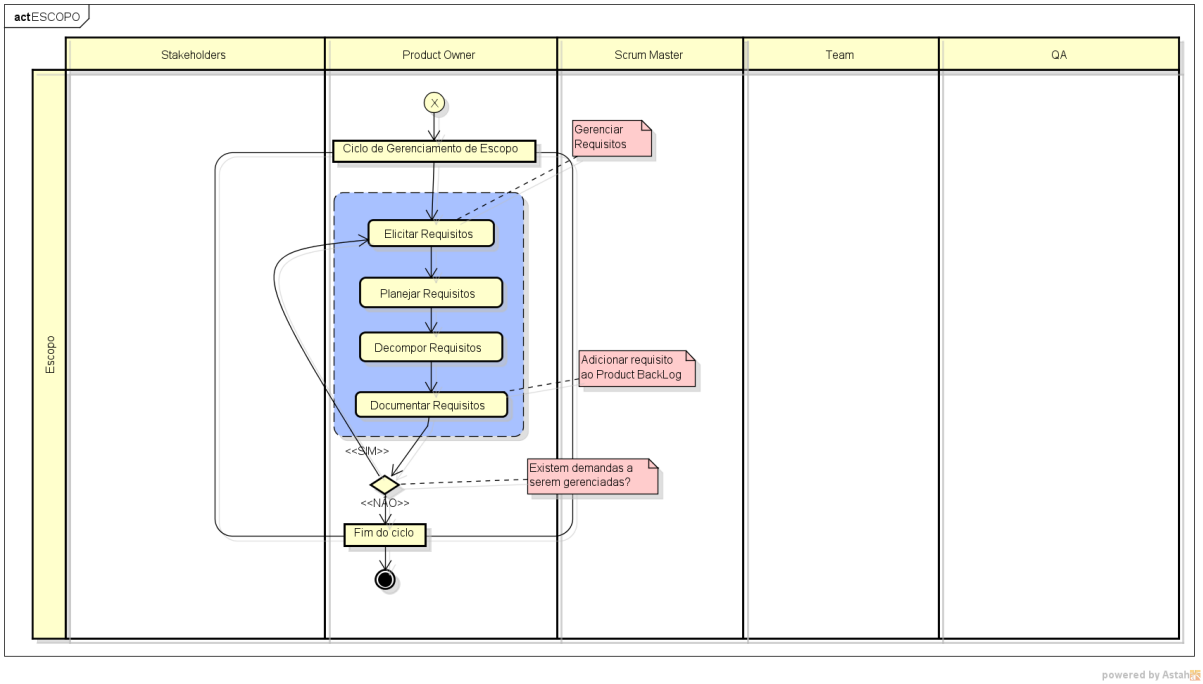
Figura 23 - Papéis envolvidos na gestão de escopo



Fonte: Elaborado pelo autor.

A Figura 24 apresenta o diagrama de atividades do processo de gestão de escopo. Nas subseções seguintes, são apresentados, detalhadamente, os processos de elicitação, planejamento, decomposição, documentação e gerência referentes à gestão de escopo que vem sendo praticado após a implantação.

Figura 24 - Diagrama de atividade da Gestão de Escopo



Fonte: Elaborado pelo autor.

4.4.2.1 Elicitação

As demandas podem ser originadas por necessidades do *stakeholder* ou necessidades do *Team*. Essas demandas são resultado da gestão de configuração, a qual fica responsável em realizar a triagem das tarefas.

As demandas dos *stakeholders*, requisição de melhorias ou novas funcionalidades, são encaminhadas através de e-mail, telefone ou contato com o suporte técnico. Tanto as demandas dos clientes quanto as demandas internas (*Team*) são encaminhadas ao Product Owner. O PO, por sua vez, analisa cada uma e a cadastra no JIRA. Elas são cadastradas na forma de *Epics* ou *User Stories* para posterior análise. O PO deve possuir todos os detalhes da demanda para repassar ao *Team* durante a primeira parte da reunião do *Sprint Planning*. No caso do PO, por representar uma série de clientes, como na empresa base para esse estudo, ele deve entrar em contato com cada cliente que requisitou a demanda e identificar todos os detalhes necessários para que o *Team* consiga implementá-la sem maiores problemas.

4.4.2.2 Planejamento

Com todas as demandas devidamente cadastradas no JIRA, o PO passa a analisar cada uma delas e a identificar quais serão implementadas. Pela visão de produto concebida pelo PO, o mesmo identifica as demandas viáveis ao projeto, sendo as com alto ROI priorizadas para serem desenvolvidas com antecedência, enquanto as demais são armazenadas para serem desenvolvidas futuramente.

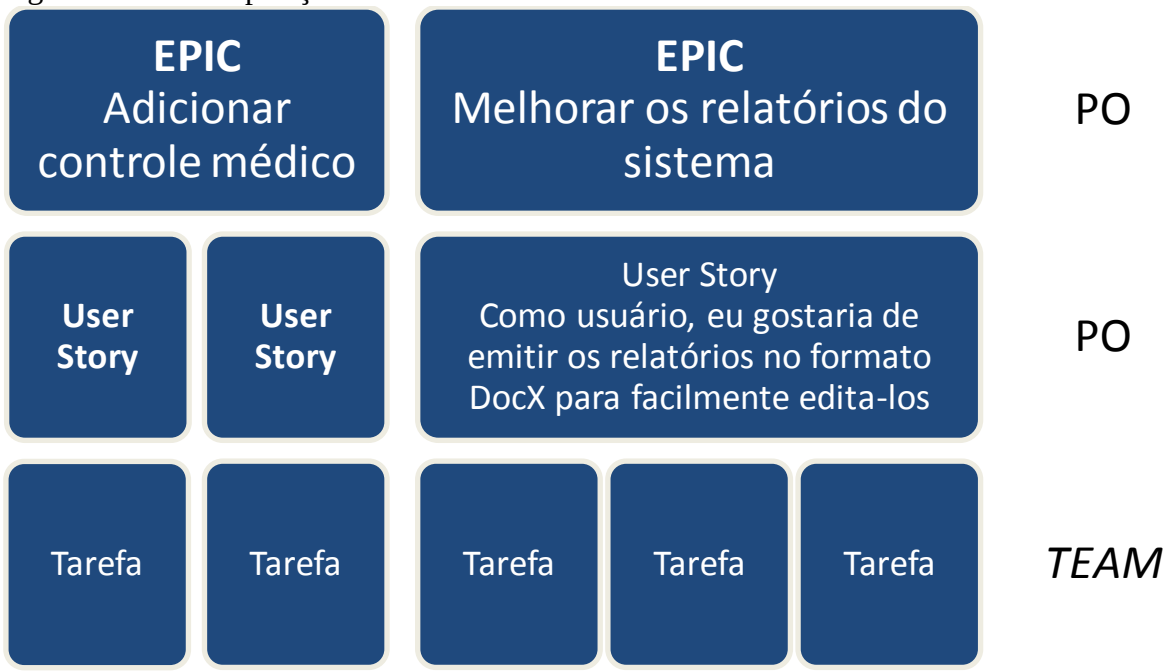
Como resultado desse processo, o PO tem um *Product Backlog* refinado e de acordo com a sua visão de produto.

4.4.2.3 Decomposição

A decomposição é realizada primeiramente pelo PO, o qual é responsável pelo *Product Backlog*. Ao receber a demanda, o PO levanta todas as informações necessárias referentes à demanda através do processo de elicitação, citado na subseção 4.4.2.1, e refinado através do processo de planejamento, citado na subseção 4.4.2.2.

Com todas as informações necessárias, o PO pode adicionar ao *Product Backlog* as demandas e decompô-las da forma desejada. Inicialmente, o PO pode criar, no JIRA, as demandas como *EPIC* ou como *User Story*. Se a demanda se estender por mais de um *Sprint*, o PO deverá criá-la como um *EPIC* e decompô-la em *User Stories,* como apresentado na Figura 25. Nesse processo inicial de decomposição desenvolvido pelo PO, as demandas são apresentadas em alto nível. O *EPIC* irá apresentar uma síntese do grupo e as *User Stories,* que compõem esse *EPIC,* irão apresentar as demandas na linguagem do usuário.

Figura 25 - Decomposição de demandas



Fonte: Elaborado pelo autor.

Na primeira parte da reunião do *Sprint Meeting*, o *Team* poderá sugerir ao PO mudanças nas demandas, e este, por sua vez, poderá aceitá-las ou não. No início da segunda parte da *Sprint Meeting*, o *Team*, com base nas *User Stories* selecionadas para serem implementadas no *Sprint*, deverá decompor essas *User Stories* em tarefas técnicas. Ou seja, as *User Stories*, descritas em alto nível, serão decompostas em tarefas técnicas, em que o *Team* e o *Scrum Master* irão detalhar o desenvolvimento das tarefas. Sendo que uma *User Story* poderá ser quebrada em mais de uma Tarefa Técnica, essa questão será escolhida pelos membros do *Team*, os quais não têm permissão para alterar os *EPICs* ou as *Users Stories*. As tarefas que necessitem de outra tarefa como pré-requisito devem ser identificadas para que o JIRA informe, ao desenvolvedor, que determinada tarefa somente poderá ser realizada quando o pré-requisito for desenvolvido.

4.4.2.4 Documentação

A documentação é realizada pelo JIRA. Nesse processo, as demandas são registradas no JIRA, conforme classificação resultante do processo de decomposição e são posicionadas de acordo com sua prioridade.

4.4.2.5 Gerência

Todas as demandas, desde seu cadastramento até a conclusão, são realizadas pelo JIRA. A ferramenta permite o acompanhamento do projeto por completo.

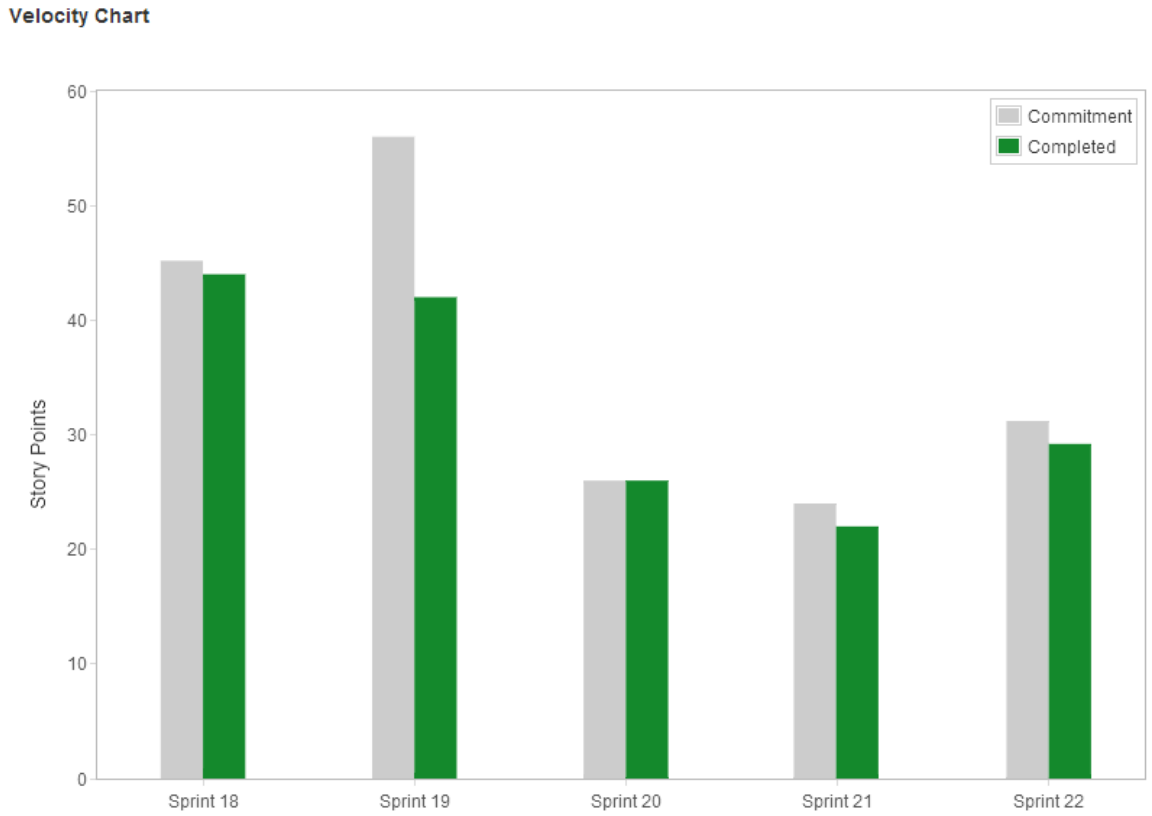
O PO pode realizar o gerenciamento do *Product Backlog* através do lançamento dos *EPICs* e *User Stories*, priorizá-lo e, por fim, apresentá-lo ao *Team* na reunião do *Sprint Planing Meeting*. Permite, também, acompanhar a evolução do projeto, assim como a evolução do *Sprint* através do *Sprint Board* em tempo real, assim como estudar gráficos de evolução.

O *Scrum Master* tem uma visão panorâmica e pode acompanhar todo o processo de perto através do JIRA. E, com o estudo dos Gráficos como *Velocity*[13], *Sprint Report*[14] e *Burndown* é possível detectar possíveis problemas que venham a ocorrer com a equipe ou com o processo. Além de poder detectar quais membros do *Team* estão ociosos.

A Figura 26 apresenta o *Velocity Chart* gerado pelo JIRA. Com ele é possível acompanhar a evolução da equipe, no decorrer dos *Sprints*, com base no comprometimento. É possível identificar que ao decorrer dos *Sprints* a equipe não tem completado todos os *Story Points* comprometidos durante a reunião do *Sprint Meeting*. Apenas no *Sprint* 20 que completaram todos os itens comprometidos, o que indica que há pontos a serem acertados no processo, os quais o *Scrum Master* terá de identificar durante as reuniões;

[13] Gráfico utilizado para analisar a velocidade dos *Sprints* já realizados.
[14] Gráfico com o detalhamento da evolução das Tarefas Técnicas realizados em cada Sprint. Utilizado para analisa nas reuniões de *Sprint Retrospective*.

Figura 26 – *Velocity Chart*



Fonte: Elaborado pelo autor.

O *Team* tem a visão, em tempo real, do *Sprint*, além de contar com a integração entre o JIRA e a IDE Eclipse. Com o *Sprint Board*, o membro do *Team* pode verificar quais tarefas foram desenvolvidas, quais estão aguardando testes e quais estão em aberto. Ao finalizar uma tarefa, o membro do *Team* vai em busca de outra tarefa em aberto no *Sprint Board* para implementá-la.

4.4.3 Gestão de tempo

No processo de desenvolvimento, após a implantação do *Scrum*, os membros responsáveis por gerenciar o escopo são o *Scrum Master* e o *Team*. Esses são responsáveis por realizar a gestão do tempo identificando, com base nas tarefas priorizadas pelo PO, quais delas poderão ser desenvolvidas durante um *Sprint* de duas semanas.

A Tabela 9 apresenta os papéis dos atores envolvidos na gestão de tempo e as responsabilidades executadas por cada um deles.

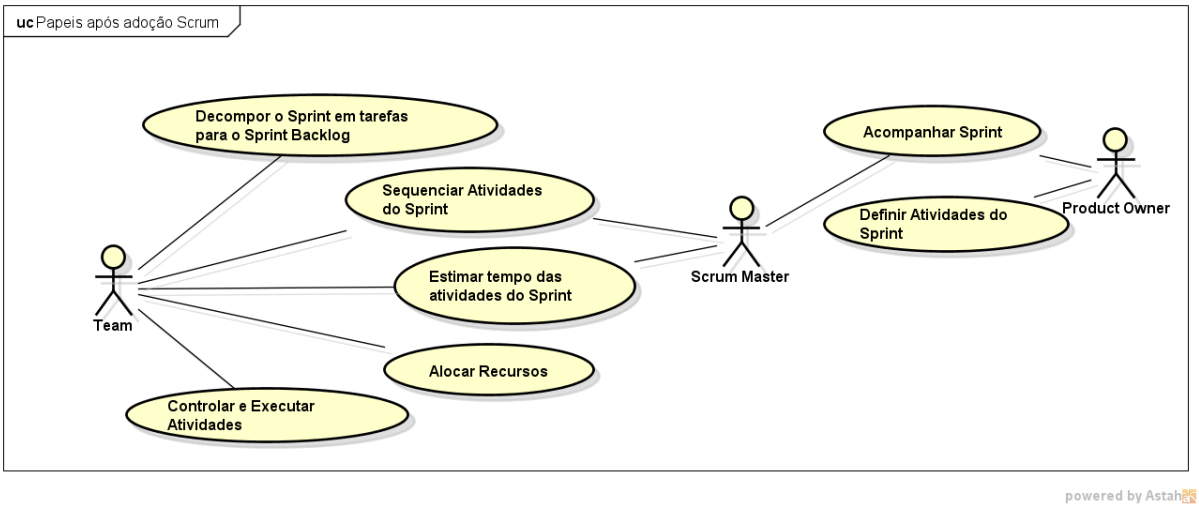
Tabela 9 - Responsabilidades dos papéis na gestão de tempo

Papel	Responsabilidades
Product Owner	Definir atividades do Sprint e acompanhar Sprint.
Scrum Master	Estimar tempo das atividades, Sequenciar atividades, Acompanhar o Sprint.
Team	Estimar Tempo, Sequenciar atividades, Controlar e Executar Atividades, Alocar Recursos, Acompanhar o Daily Meeting e participar do Sprint Planning Meeting.

Fonte: Elaborado pelo autor.

A Figura 27 apresenta os atores e suas respectivas demandas no processo de gestão de tempo. O processo de gestão de tempo, resumidamente, é representado pelo PO responsável por definir as atividades na primeira parte da reunião do *Sprint Planning Meeting*. *Scrum Master* e *Team* são responsáveis por realizar a *Sprint Planning Meeting* parte 2 (Reunião para Estimar o tempo das tarefas e sequenciar as atividades realizadas no Sprint) e realizar a *Daily Meeting*. Diariamente, o *Scrum Master* fica responsável por analisar os tópicos levantados durante a *Daily Meeting*. Caso algum problema seja levantado por um membro do *Team*, é necessário que ele seja resolvido o quanto antes, para evitar que isso prejudique o andamento dos trabalhos. E, por fim, o *Team* fica responsável por atualizar o andamento do *Sprint*.

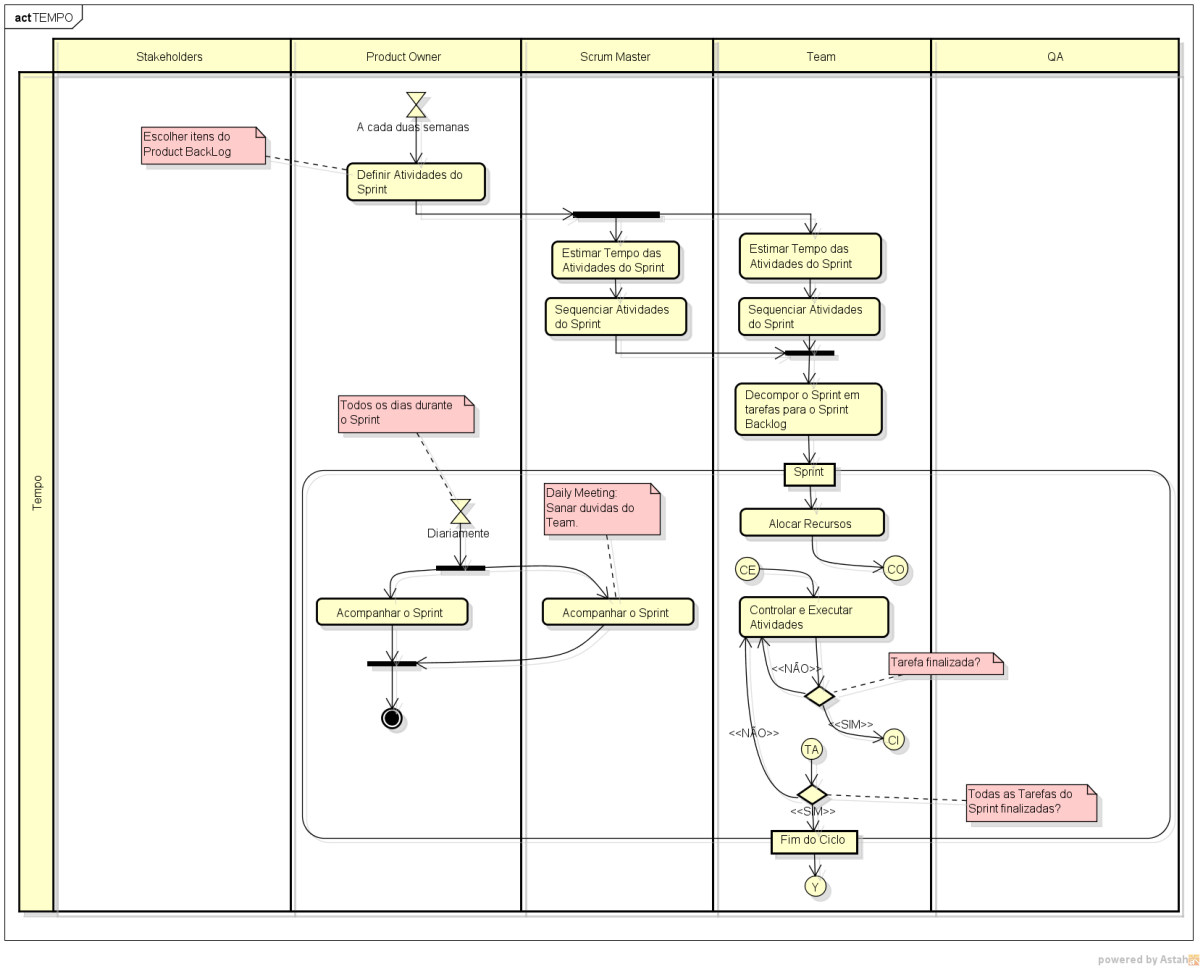
Figura 27 - Papéis envolvidos na gestão de Tempo



Fonte: Elaborado pelo autor.

A Figura 28 apresenta o diagrama de atividades do processo de gestão de tempo. Nas subseções seguintes, são apresentados, detalhadamente, os processos de definição de atividades, estimativa de tempo, sequenciamento de atividades, alocação e controle, e execução referente à gestão de tempo que vem sendo praticado atualmente pela empresa.

Figura 28 - Diagrama de atividade da Gestão de Tempo



Fonte: Elaborado pelo autor.

4.4.3.1 Definição das atividades

O *Product Backlog* é definido no processo de planejamento, citado na subseção 4.4.2.2. Dessa forma, todas as demandas que se encontram no PB devem ser implementadas de acordo com as suas prioridades. Como o *Scrum* trabalha com iterações, deverão ser identificadas as demandas que farão parte da iteração. Para isso o PO, *Scrum Master* e *Team* realizam a primeira parte da reunião de planejamento do *Sprint* (*Sprint Planning*).

Nessa reunião, o PO mostra ao *Scrum Master* e ao *Team* a sua visão do produto e apresenta as demandas de maior importância. Nessa reunião, é importante que o PO passe à equipe como ele realmente visualiza o produto e o *Team*, por sua vez, deve questionar o PO sobre as dúvidas que venham a surgir.

Ao final da primeira parte dessa reunião, o PO deverá atualizar o *Product Backlog* no JIRA com as demandas de maior importância, indicando ao *Team* quais demandas ele desejaria que fossem implementadas para o próximo *Sprint*.

Na segunda parte da reunião, o PO poderá se retirar e o *Team* e o *Scrum Master* deverão discutir o que será implementado no *Sprint*. Cada demanda é analisada e são identificadas as que poderão ser entregues em um *Sprint* de duas semanas. Com base nessa reunião, as demandas são decompostas pelo *Team* em tarefas técnicas e o *Sprint* é criado no JIRA.

Todas as demandas que foram selecionadas para o *Sprint* devem ser implementadas pelo *Team* o qual deve entrar em um acordo referente ao desenvolvedor que irá implementar cada tarefa. Dessa forma, por ser uma equipe autogerenciável, é possível que a equipe entre em comum acordo referente a qual tarefa será implementada e por quem, com atenção para as tarefas de maior prioridade.

4.4.3.2 Estimativas de tempo

A estimativa de tempo é realizada, inicialmente, pelo PO, o qual estima um *Story Point*[15] para cada demanda. Na reunião de *Sprint Planning*, os membros do *Team* podem contestar essa estimativa e negociar uma nova com o PO. Esse processo é acompanhado pelo *Scrum Master* que, se necessário, pode interferir e negociar entre as partes.

Para estimar o tempo, o PO pode utilizar o histórico de tarefas realizadas e que está cadastrado no sistema de gestão, enquanto o *Team* utiliza de sua experiência para estimar o tempo para implementação de cada tarefa. O *Team*, normalmente, após o segundo *Sprint* consecutivo sem interferência externa, passa a realizar estimativas próximas à exatidão.

4.4.3.3 Sequenciamento das atividades

O processo de sequenciamento de atividades é realizado pelo *Team* na segunda parte do *Sprint Planning Meeting*. As tarefas são sequenciadas durante o processo de

[15] É a combinação da complexidade, esforço e risco e utilizadas para mensurar a velocidade de uma equipe ágil.

decomposição, sendo o *Team* responsável pelo sequenciamento das atividades de acordo com a prioridade determinada pelo PO e pelos pré-requisitos para elaboração e pelo desenvolvimento de tarefas anteriores.

4.4.3.4 Alocação

A alocação para implementação das tarefas é realizada pelo próprio *Team*, com acompanhamento do *Scrum Master*. As tarefas são discutidas e criadas com base nas *User Stories* na segunda parte da *Sprint Planning Meeting*. Dessa forma, todos os membros do *Team* têm conhecimento das tarefas, podem se dirigir ao *Sprint Board* e selecionar a tarefa a ser desenvolvida, ao finalizar, o *Board* é atualizado e o mesmo pode buscar uma nova tarefa a ser desenvolvida. O processo é similar ao *Kanban*, tornando a equipe autogerenciável. Caso não tenha mais tarefas a serem desenvolvidas, o membro do *Team* pode se juntar a outro membro da equipe e ajudá-lo a concluir o desenvolvimento da tarefa.

4.4.3.5 Controle e execução

O controle é realizado diariamente pela *Daily Meeting* e pela atualização do *Sprint Board*. Todos os dias, pela manhã, é realizada uma rápida reunião de acompanhamento entre o *Team* e o *Scrum Master* com intuito de alinhar o progresso. Todos os membros do *Team* devem saber o que realizaram no dia anterior e o que deverão implementar no restante do dia, e, ao mesmo tempo, estarão atualizando os demais membros.

Cada tarefa concluída deve ter seu status atualizado no *Sprint Board*. Quando um membro do *Team* iniciar a implementação de uma tarefa, essa deve ter seu status atualizado no *Sprint Board* de “Em aberto” para “Em andamento”. Caso a tarefa tenha sido implementada e testada pelo desenvolvedor, ela deve ter seu status atualizado para “*Revisão*”. Tarefas em revisão são de responsabilidade dos membros do QA, os quais deverão realizar uma bateria de testes para assegurar que a tarefa foi implementada de forma correta. Após os testes do QA, essa tarefa deve ser atualizada para “Concluída”.

4.4.4 Gestão de qualidade

No processo de desenvolvimento, após a implantação do *Scrum*, todos os membros da equipe têm alguma participação no processo de qualidade.

A Tabela 10 representa os papéis dos atores envolvidos na gestão de tempo e as responsabilidades executadas por cada um deles.

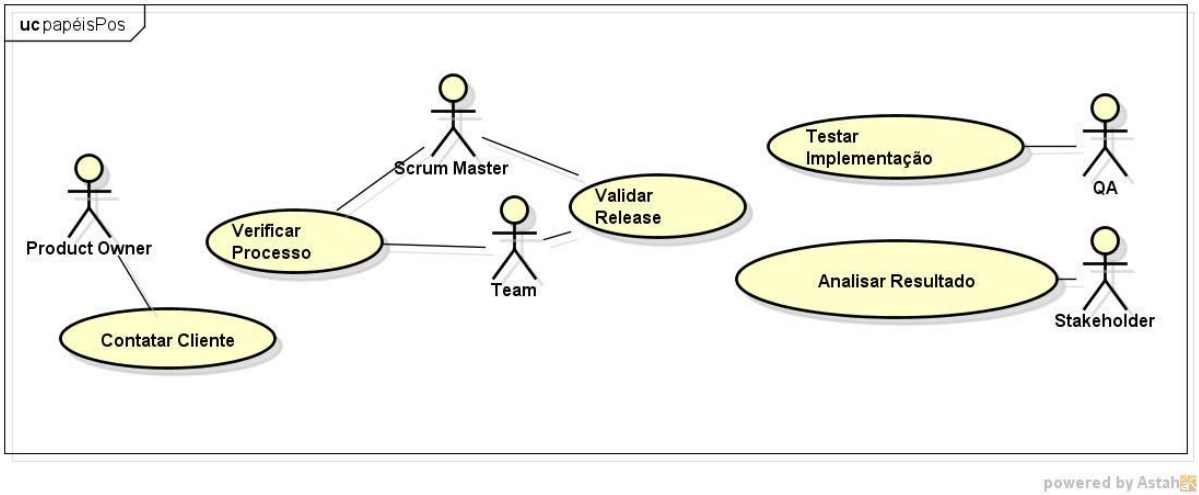
Tabela 10 - Responsabilidades dos papéis na gestão de qualidade

Papel	Responsabilidades
QA	Testar as implementações durante o *Sprint*;
StakeHolder	Avaliar o resultado da demanda implementada durante o Sprint;
Scrum Master	Verificar o Processo, Validar o *Release*, participar da *Sprint Review*, participar da *Sprint Restrospective*;
Product Owner	Contatar o cliente;
Team	Verificar o Processo, Validar o *Release*, participar da *Sprint Review*, participar da *Sprint Restrospective*.

Fonte: Elaborado pelo autor.

A Figura 29 representa os atores e suas respectivas tarefas no processo de gestão de qualidade. O processo de gestão de qualidade é resumidamente representado pelo *Scrum Master* e pelo *Team* que realizam a reunião de *Sprint Review* e *Sprint Retrospective*. Já, o QA realiza o teste de cada tarefa implementada pelo *Team* durante os *Sprints* e o *Stakeholder*, no final do processo, analisa a implementação e identifica se ela foi implementada da forma como discutida com o PO.

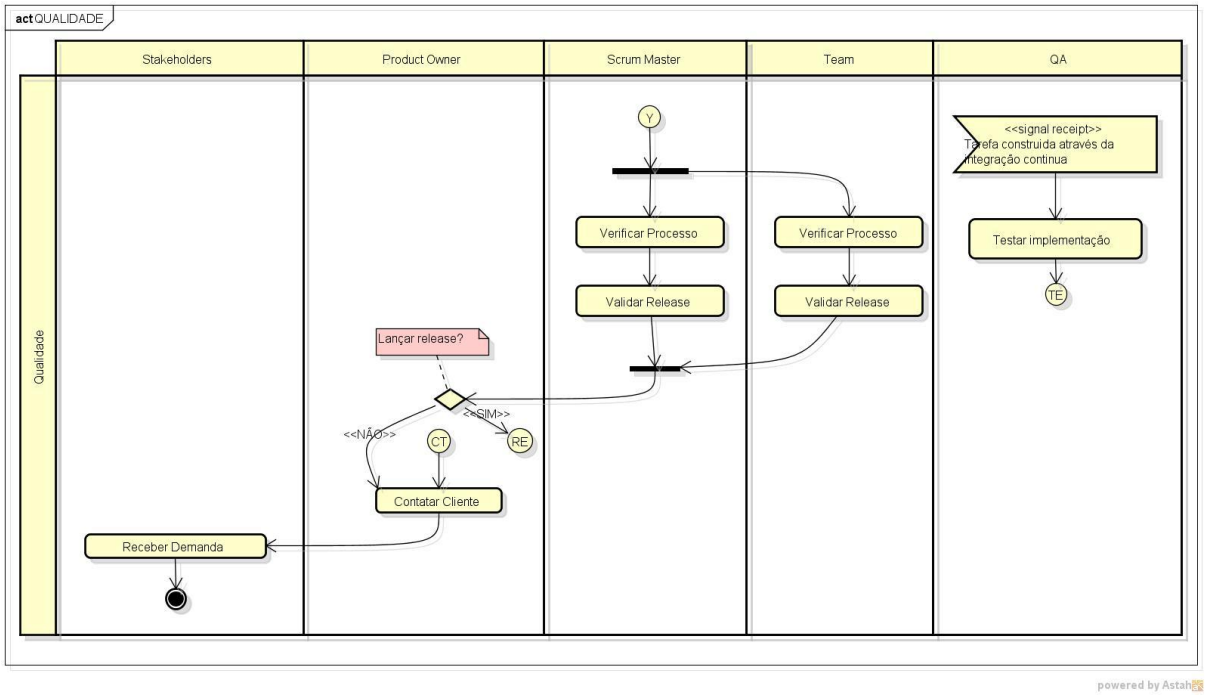
Figura 29 - Papéis envolvidos na gestão de Qualidade



Fonte: Elaborado pelo autor.

A Figura 30 apresenta o diagrama de atividades do processo de gestão de qualidade. Nas subseções seguintes são apresentados, detalhadamente, os processos de verificação do produto, verificação do processo, validação e testes referentes à gestão de qualidade que vem sendo praticado após a implantação do Scrum pela empresa.

Figura 30 - Diagrama de atividade da Gestão de Qualidade



Fonte: Elaborado pelo autor.

4.4.4.1 Verificações do produto

Os testes de verificação do produto os quais validam se os requisitos foram implementados com sucesso, são realizados inicialmente pelo próprio desenvolvedor após a conclusão de implementação da tarefa. O desenvolvedor testa as funcionalidades implementadas para identificar se as mesmas foram implementadas com sucesso, caso haja alguma inconsistência, o desenvolvedor realiza a correção na mesma hora.

Após o desenvolvedor realizar todas as validações necessárias, a tarefa é encaminhada ao QA que são responsáveis por validar a implementação. Todos os testes necessários são realizados, caso alguma inconsistência seja identificada um membro do QA abre uma nova tarefa identificando os problemas detectados no teste e fica no aguardo de uma nova tarefa para ser validada.

4.4.4.2 Verificações do processo

As inspeções no código fonte do desenvolvedor apenas são realizadas nos primeiros meses de trabalho, com a finalidade de identificar se o mesmo vem seguindo o padrão de desenvolvimento do projeto. Após identificar que o desenvolvedor se adaptou ao padrão, não são mais realizados esses testes. Quando há a contratação de um novo membro de equipe, um membro mais antigo é denominado para realizar o processo descrito.

Também são realizadas a *Daily Meeting* para identificar possíveis problemas no processo e, ao final de cada *Sprint*, é realizado o *Sprint Retrospective* com todos os membros, a fim de melhorar o processo. Essa é uma reunião na qual todos os membros podem sugerir melhorias ao processo utilizado.

4.4.4.3 Validação

O processo de validação não sofreu alterações no processo após a implantação do Scrum.

4.4.4.4 Testes

Todos os testes realizados são de forma manual. No processo após a implantação do Scrum padrão utilizado pela empresa não é feito uso de testes unitários ou automatizados. É realizado um acompanhamento por parte do QA, onde são mensuradas as quantidades de erros encontradas em cada *Sprint*. Esses dados são encaminhados para o *Scrum Master*, o qual fica responsável por identificar a causa e, juntamente com o *Team*, resolver a situação.

4.4.5 Ferramentas utilizadas

As ferramentas utilizadas para auxiliar no processo implantação do Scrum foram:

- JIRA – Gestão de mudanças;
- JIRA Agile – Módulo do JIRA para gestão Agile;
- JIRA Capture - Módulo do JIRA para gestão de testes Agile;

- Bamboo – Integração contínua e *deploy* de aplicações;
- SVN – Subversion.

O JIRA é uma ferramenta desenvolvida para gerenciamento de projetos. Ela é utilizada para conectar os profissionais, centralizando as tarefas a serem realizadas além de possibilitar o acompanhamento do trabalho que está sendo realizado. A ferramenta aumenta a produtividade, a qualidade de execução e o rastreamento das tarefas que envolvem um projeto. Isso é possível, pois ela oferece uma interface amigável para localizar os bugs e tarefas, vincula automaticamente as tarefas aos códigos-fonte implementados pelos desenvolvedores e possibilita o monitoramento das atividades.

O JIRA Agile é um módulo do JIRA que facilita o gerenciamento de projetos ágeis tanto para quem está começando quanto para quem já utiliza a metodologia há vários anos. A ferramenta possibilita a criação e a estimativa de *User Stories*, a criação de *Product Backlog* e *Sprint Backlog*, a criação e o gerenciamento do *Sprint Board*. Tendo como seu ponto positivo o trabalho em conjunto com o JIRA, centralizando a execução das tarefas em um única ferramenta, possibilitando a gestão e controle do progresso do trabalho, a identificação do nível de comprometimento da equipe. Trazendo ao meio eletrônico, o que antigamente precisava ser realizado de forma manual utilizando quadros e papéis.

O JIRA Capture é um módulo do JIRA que facilita a realização dos testes em ambientes WEB. Consiste em um plug-in, disponível para os navegadores Internet Explorer, Firefox, Chrome e Safari, que auxilia na realização de testes. Com o gerenciamento de sessões de teste, permite à equipe de testes gerenciar as tarefas testadas, mensurar o tempo dedicado aos testes e indicar em quais tarefas foram detectados os problemas e quais os problemas detectados em cada tarefa. Cada tarefa cadastrada pela ferramenta é armazenada com os dados do ambiente onde o teste foi realizado e também possibilita ao testar adicionar imagens ou arquivos aos testes.

O Bamboo é um software utilizado para gerenciamento de integração continua e *deploy* de aplicações. Por ser totalmente integrável com o JIRA é possível identificar as tarefas implementadas em cada *build*. Em caso de quebra de *build* é possível identificar qual usuário enviou o código responsável pela quebra e em qual tarefa estava trabalhando. A separação entre o *build* e o *deploy* trouxe maior organização ao processo. É possível realizar o *build* de um artefato e realizar o *deploy* do mesmo em vários servidores sem a necessidade de

realizar o *build* novamente. Possibilitando dessa forma entregar o produto em servidores de teste ou até mesmo atualizar o produto dos clientes.

Para realizar a implantação do JIRA e seus aplicativos a empresa realizou o investimento de $50 dólares para a licença de suporte referente a doze meses e até dez usuários. Esta licença garante a atualização das versões do sistema neste período e a não renovação impede apenas as atualizações. Este valor é baseado em uma tabela disponibilizada pela Atlassian de acordo com a quantidade de usuários que irão utilizar a ferramenta.

O SVN é um sistema de controle de versão, o qual gerencia arquivos e diretórios, e as modificações feitas neles ao longo do tempo. Permite que as versões antigas de seus dados sejam recuperados, ou que o histórico das alterações dos arquivos possam ser consultados. Permite que várias pessoas modifiquem e gerenciem o mesmo conjunto de dados de seus próprios locais, fomentando a colaboração. Possibilitando que a equipe trabalhe em duas versões de um mesmo projeto em simultâneo, além de controlar os *releases* de cada versão.

Para a implantação do SVN não houveram custos, pois o processo foi realizado pelos próprios membros da equipe.

4.5 O processo de implantação

O processo de implantação se deu inicialmente com a apresentação da metodologia para o diretor da empresa. Foram apresentados todos os problemas enfrentados diariamente pela equipe de desenvolvimento e suporte.

Problemas como falta de comunicação entre os membros da equipe, a necessidade de adaptação às frequentes mudanças de escopo. Os *releases,* contendo as novas funcionalidades, demoravam, no mínimo, seis meses para serem disponibilizados aos clientes.

Em contrapartida foram apresentados os benefícios que a metodologia traria para a empresa, como a solução para os problemas citados. Ao final, a equipe e o diretor chegaram ao consenso de realizar o piloto da implantação do *Scrum* em um projeto recente para migração do antigo sistema em Delphi para Java, devido ao fato de poucos clientes utilizarem o software e o mesmo ser desenvolvido praticamente do zero. O que possibilitaria uma comparação entre as duas metodologias: a antiga e a nova.

Com o apoio da diretoria, foi elaborado um plano para realizar a implantação da metodologia. Foi definido, inicialmente, que a implantação da metodologia iria ocorrer por partes, para evitar um choque, pois seriam muitas mudanças em comparação com o processo antigo. A primeira parte a ser implantada foi a gestão de configuração, a qual, de acordo com o modelo proposto, ficou definida como a base de todo o processo. Garantindo agilidade e eficácia à metodologia.

4.5.1 Gestão de Configuração

O primeiro passo para implantação da gestão de configuração foi a implantação do controle de versões *Subversion*. O mesmo foi apresentado à equipe e, gradativamente, todos os projetos foram migrados para o controle de versões. Os principais problemas encontrados durante o processo de implantação do *Subversion* foram problemas relativos à junção dos arquivos modificados pelos desenvolvedores, conhecido como *merge*. Esses problemas ocorriam devido à forma desorganizada utilizada pelos desenvolvedores para a codificação e manutenção do código fonte. Esse problema foi resolvido através da elaboração de um manual de boas práticas que foi distribuído a todos os membros da equipe, diminuindo efetivamente o problema relativo ao *merge* dos arquivos modificados levando menos de uma semana para entrar em produção.

O próximo passo foi a implantação de um sistema de gerenciamento de projetos em substituição ao antigo *dotProject*. Foram realizados testes com o *RedMine, BugZilla* e JIRA. O JIRA se destacou, pois possui um pacote com diversos softwares para realizar o controle do processo, principalmente, ao que diz respeito à metodologia de desenvolvimento ágil.

O período de comparação e teste entre as ferramentas levou aproximadamente um mês, pois o processo foi realizado fora do horário de trabalho para evitar interferência com as demandas internas. O JIRA se saiu superior em comparação com as outras ferramentas, pois ele possui uma ferramenta para o controle de projetos e rastreamento, que vincula tarefas e código fonte, além de contar com um *plugin* para trabalhar com projetos Agile.

O processo de implantação do JIRA também foi realizado em partes. Primeiramente foi realizada a implantação do software em um ambiente virtualizado e um membro do *Team* se ofereceu para realizar os testes durante o desenvolvimento das tarefas. Este processo levou

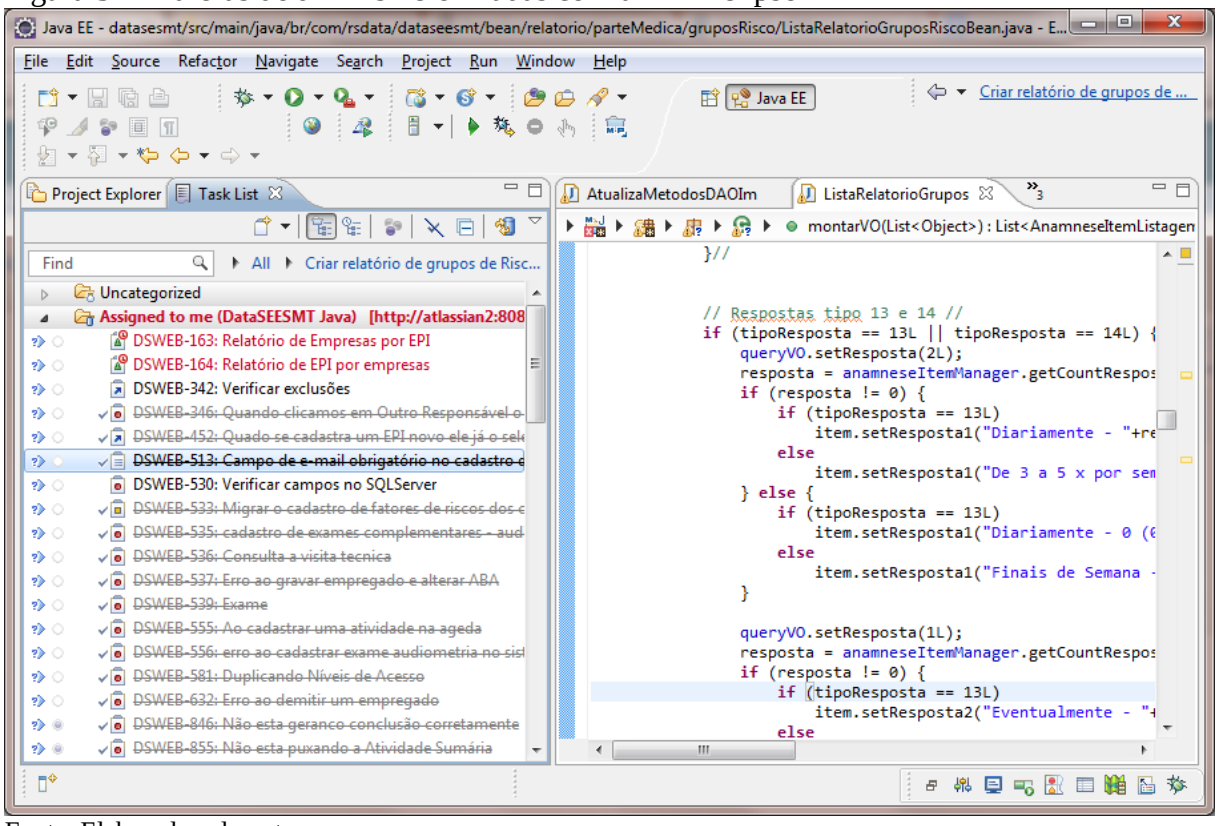
aproximadamente dois meses. Isso se deu, pois houveram problemas tanto no ambiente virtualizado quanto na integração dos módulos e aplicativos com o JIRA. Principalmente com o uso das opções para compartilhar os usuários do JIRA com as outras aplicações da Atlassian, opção esta, que não utilizamos mais. Outro problema constado foi o uso do MYSQL como base de dados, esta estava instalada no mesmo computador e como o JIRA é um software que demanda muitos recursos, tivemos de mover a base MYSQL para uma base SQLServer em outro computador.

Com o software implantado e testado, os membros da equipe foram reunidos e foram apresentadas a eles todas as melhorias que a implantação do software iria trazer e, então, foi realizado um piloto de uma semana, sendo que o membro que realizou os testes iniciais ficou como replicador e ajudou os outros membros com a nova ferramenta.

Após o término do piloto, foi realizada nova reunião na qual os membros da equipe decidiram pelo uso da ferramenta. Os principais motivos levantados foram a integração da ferramenta com a IDE Eclipse, a qual possibilitada que o desenvolvedor possa ter acesso a todas as demandas sem sair da IDE.

A Figura 31 apresenta a IDE Eclipse integrada com as tarefas do JIRA. Na aba *Task List* o desenvolvedor tem acesso a todas as tarefas vinculadas ao seu usuário do JIRA. O desenvolvedor também pode personalizar as *queries* para apresentar as tarefas na IDE da forma como preferir. Cada uma das tarefas listadas possui um contexto, no qual são armazenados todos os arquivos que foram trabalhados nesta tarefa.

Figura 31 - Tarefas do JIRA sincronizadas com a IDE Eclipse

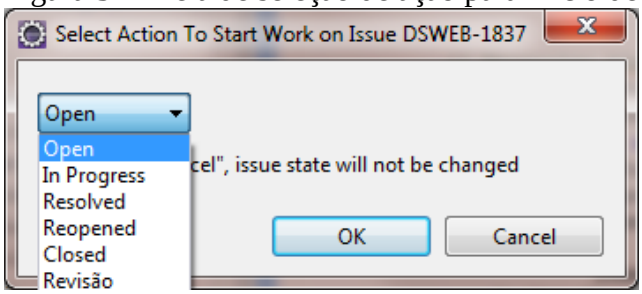


Fonte: Elaborado pelo autor.

A IDE Eclipse possibilita realizar o controle de tempo e do contexto de trabalho referente à uma tarefa. Ao iniciar o trabalho em uma tarefa pela IDE, ela armazena todos os arquivos modificados. Ao término do trabalho estes dados são sincronizados com o JIRA, que recebe o tempo de trabalho e a relação dos arquivos modificados e grava junto a tarefa. Dessa forma, se o usuário voltar a trabalhar nesta tarefa em outro momento, a IDE mostra a ele quais os arquivos que ele modificou até aquele instante, além de armazenar o tempo de trabalho na tarefa.

A Figura 32 apresenta a tela de seleção de ação visualizada pelo desenvolvedor ao iniciar o trabalho em uma tarefa. Ao iniciar uma tarefa, o desenvolvedor deve selecionar o status desta tarefa. Ao selecionar, a IDE irá sincronizar a tarefa com o JIRA, atualizando então o *Sprint Board*, ficando visível a todos os membros da equipe o status da tarefa.

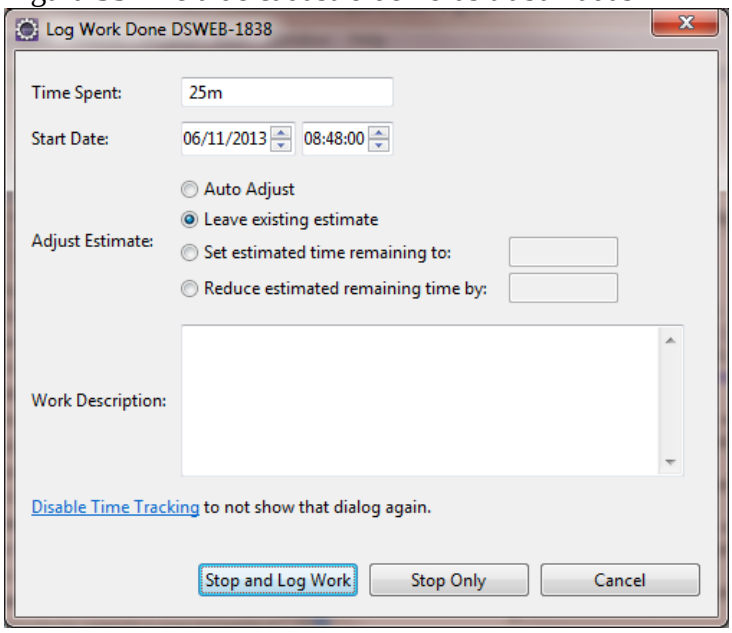
Figura 32 - Tela de seleção de ação para início de trabalho em tarefa



Fonte: Elaborado pelo autor.

A Figura 33 apresenta a tela de cadastro de horas trabalhadas apresentada ao desenvolvedor ao finalizar uma tarefa. Ao finalizar o trabalho em uma tarefa, o desenvolvedor visualiza a tela, a qual informa o tempo gasto na implantação da tarefa, a data de início e por fim ele pode informar observações sobre o trabalho realizado. Ao gravar, as informações são sincronizados com a base de dados do JIRA alimentando as estatísticas referentes ao trabalho do desenvolvedor.

Figura 33 - Tela de cadastro de horas trabalhadas

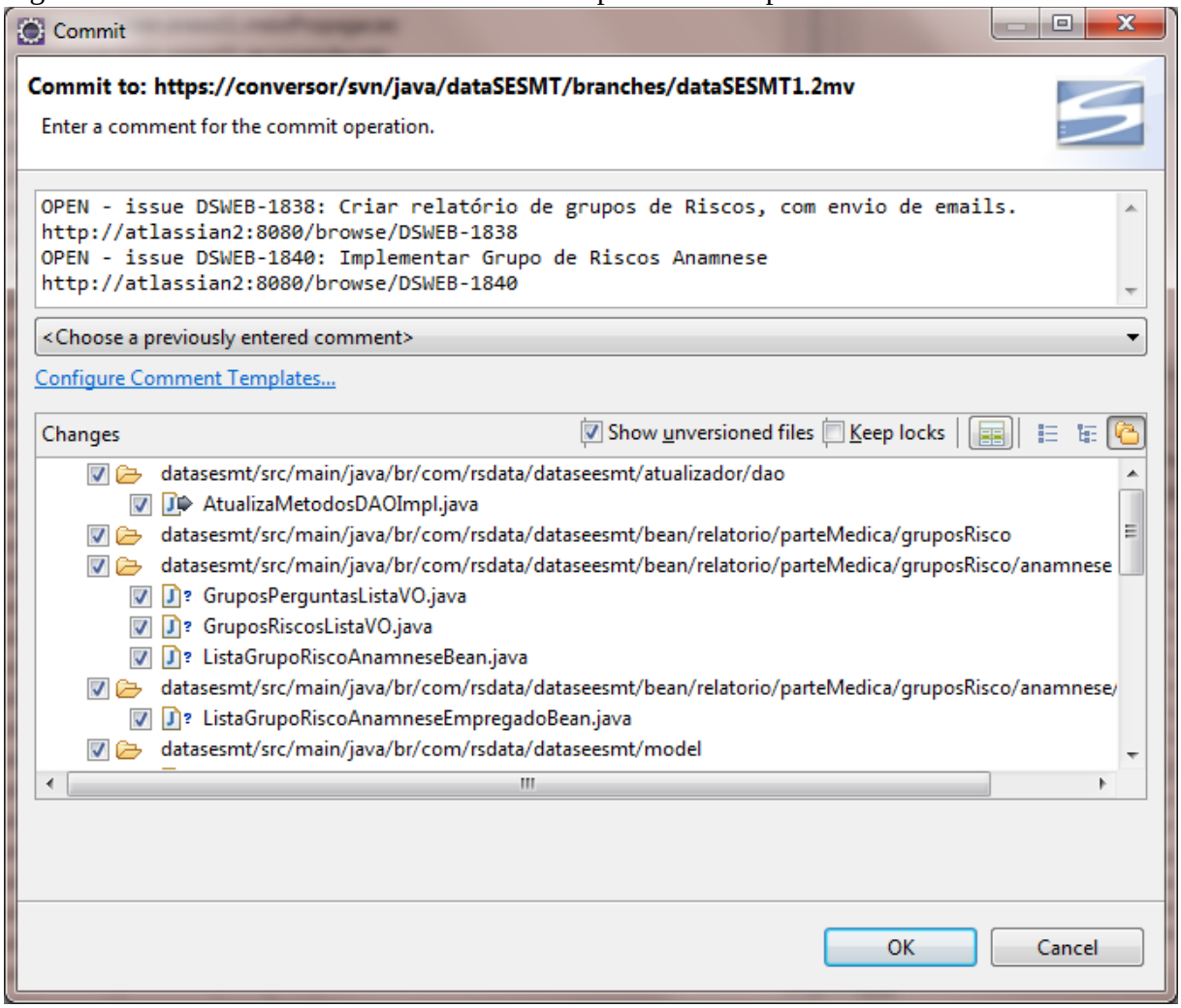


Fonte: Elaborado pelo autor.

Outro ponto positivo apontado pela equipe foi a possibilidade de realizar o vínculo entre o *commit* realizado no *Subversion* com a tarefa. Com a integração realizada entre a IDE e o JIRA, quando o programador realiza o *commit*, a IDE automaticamente informa, na descrição do *commit*, o código da tarefa que foi desenvolvida. Isso possibilita que toda a equipe possa identificar quais arquivos foram alterados para completar determinada tarefa, trazendo rastreabilidade e segurança para o projeto.

A Figura 34 apresenta a tela de *commit* apresentado pela IDE Eclipse ao enviar os arquivos para o servidor. Na descrição das alterações realizadas, a IDE apresenta os títulos das tarefas realizadas pelo desenvolvedor, seguidos pelo status (*OPEN*, *IN PROGRESS*) e o ID do JIRA referente a tarefa. Com a identificação do JIRA, o *fisheye* realiza o vínculo entre as tarefas e os *commits* apresentando os arquivos alterados em cada tarefa.

Figura 34 - Tela de *commit* ao finalizar a tarefa pela IDE Eclipse



Fonte: Elaborado pelo autor.

Em seguida, foi realizada a implantação do *bamboo*, o qual contribuiu para a realização dos testes através da geração de *builds* automáticos após cada *commit* realizado por um membro da equipe.

O *bamboo* resolveu o problema da necessidade de um membro da equipe de desenvolvimento atualizar o ambiente de testes manualmente a cada tarefa implantada. Como o processo antigo era muito manual, necessitava que, toda a vez em que o programador

terminasse de implantar uma tarefa no sistema, fosse criado um *build* e então preparasse o ambiente de testes para o encarregado testar. Isso, com o *bamboo*, ocorre automaticamente.

No processo antigo, muitas vezes, o programador acabava não colocando uma versão para testes, pois perdia muito tempo com a configuração. O que ele fazia, era esperar acumular uma quantidade de tarefas desenvolvidas, para, então, colocar a versão para testar, afetando, assim, todo o processo de testes e de desenvolvimento de software.

4.5.2 Gestão de escopo

Com o cenário constituído, favorável à implantação da metodologia, foi realizada uma apresentação aos membros da equipe referente à metodologia *Scrum* e a equipe se comprometeu com a nova metodologia de trabalho. A partir desse momento, o analista, que no processo antigo realizava várias atividades, passou a delegá-las aos membros do projeto.

Na gestão do escopo, o membro da equipe técnica com conhecimentos voltados à área de negócio do software ficou encarregado de realizar a gestão. Por se tratar de uma pessoa com pouco conhecimento em desenvolvimento de software, o *Scrum Master* realizou o acompanhamento durante o primeiro mês para a realização do levantamento de requisitos.

A principal dificuldade encontrada pelo *Product Owner* foi a forma como buscar as necessidades dos clientes e como apresentá-las na *Sprint Planning Meeting*. Isso ocorreu, pois o PO não possuía uma visão de arquitetura de software e, muitas vezes, deixava passar alguns itens importantes para implementação da requisição por parte da equipe de desenvolvimento.

A solução encontrada pelo *Scrum Master* foi, durante este período de um mês de acompanhamento, mostrar ao PO alguns diagramas referentes à estrutura do software. Durante este período, PO e o *Scrum Master* analisaram as demandas juntamente com o diagrama ER. Ao final do primeiro mês de acompanhamento, o PO já possuía uma noção de como as principais funcionalidades do software estavam estruturadas.

Após este período de acompanhamento intensivo, o PO já conseguia gerenciar o *Product Backlog* e contatar os clientes para esclarecer dúvidas referentes à elaboração dos requisitos das demandas, para, então, apresentar a sua visão de produto aos demais membros na *Sprint Planning Meeting*.

Na concepção inicial para implantação do *Scrum*, a gestão de escopo foi atribuída para um membro da equipe técnica que também realiza outras funções externas como apresentação do software e treinamentos. Ao contrário do que havia sido planejado inicialmente, o processo de gestão de escopo requer tempo e concentração, principalmente em um ambiente em que as leis estão em mudança constante. Por esse motivo, está sendo estudada a contratação de uma pessoa específica para realizar o processo de gestão de escopo.

4.5.3 Gestão de tempo

Na gestão de tempo, os membros da equipe de desenvolvimento ficaram encarregados pelo processo de execução do produto. Por se tratar de uma equipe nova e sem nenhum contato anterior com a metodologia *Scrum* o processo de adaptação torna-se mais lento. Isso se deve ao fato da equipe de desenvolvimento estar acostumada a implementar o que o analista havia planejado. O que não acontece no novo processo, pois a metodologia traz a autonomia para a equipe de desenvolvimento.

Nos primeiros *Sprints,* a equipe sentiu dificuldades em tomar decisões, principalmente, da forma como as demandas deveriam ser implementadas e quais tarefas poderiam ser desenvolvidas e entregues no final de duas semanas. Por esse motivo, nos primeiros dois *Sprints,* o *Scrum Master* indicou como as demandas deveriam ser realizadas e, a partir do terceiro, apenas acompanhou o processo, sanando dúvidas que eventualmente pudessem surgir.

Para dar maior segurança para os membros da equipe quanto ao tempo de desenvolvimento das tarefas, o JIRA possui um banco de dados com todas as tarefas realizadas por cada integrante da equipe e o tempo necessário para conclusão da tarefa. Esses dados podem ser consultados por qualquer integrante da equipe e servem como apoio ao processo de planejamento do *Sprint*. Através da consulta dos históricos, cada membro da equipe pode verificar o tempo médio para a implementação e utilizar como base para as novas demandas.

4.5.4 Gestão de qualidade

Na gestão de qualidade, o processo manual necessário para preparar o ambiente de testes não é mais necessário. Com o uso do *bamboo,* todas as alterações realizadas no software, pelos programadores, são compiladas e enviadas ao servidor de testes automaticamente. Isso é possível através do uso dos *scripts* pré-configurados no *bamboo*. Evitando, dessa forma, que os programadores se esqueçam de realizar o *build* e preparar o servidor de teste, pois o processo não depende mais deles, e sim da ferramenta.

Durante os primeiros *Sprints,* não houve detalhamento na decomposição das tarefas. Isso acabou ocasionando dificuldades para os responsáveis pela realização dos testes. Mesmo participando de todas as reuniões, faltavam detalhes que eram implementados pelos programadores e que não eram testados. Dessa forma, ficou combinado com a equipe de desenvolvimento que fossem realizados detalhamentos na decomposição das tarefas a serem implementadas.

Com o detalhamento realizado pelos programadores, os testadores utilizaram a ferramenta JIRA Capture para a realização dos testes. Essa ferramenta permite ao testador testar cada tarefa realizada e vincular o teste à mesma. Dessa forma, é possível acompanhar o progresso dos testes realizados no sistema e garantir que todas as tarefas realizadas sejam cobertas por testes.

Outro ponto de adaptação foram os testes realizados no software, pois, por não possuir uma equipe de testes, esses, inicialmente, ficaram com um membro do suporte. Porém, devido à falta de tempo disponível para testar o software, o *Scrum Master* também realizou os testes das tarefas, juntamente com o membro do suporte. Dessa forma, para não comprometer os *Sprints* pela falta de testes, será criado um setor de testes para a empresa e será contratada uma pessoa destinada para essa função.

As *Daily Meetings* têm sido realizadas com frequência durante quinze minutos antes do início dos trabalhos. A reunião de planejamento de *Sprint* é realizada de forma informal: o PO apresenta as demandas com maior ROI e os membros do *Team* identificam quais poderão ser implementadas e realizam a decomposição. A reunião tem demorado menos de duas horas e o processo foi adaptado para necessitar de apenas uma reunião, e não de duas, como apontado no *Scrum*.

Ao final de cada *Sprint* é realizada uma reunião e são identificadas todas as demandas implantadas. O ambiente de testes é acessado e o PO verifica cada demanda implantada no projeto. Ao final da reunião, se as funcionalidades implantadas no pacote estiverem de acordo com a visão do PO, o software é disponibilizado para os clientes. Caso contrário, o PO apresenta os pontos a serem melhorados na próxima reunião de planejamento.

4.6 Avaliação

Aplicou-se, na empresa, foco desse trabalho, um questionário referente ao processo de implantação da metodologia *Scrum*. O questionário visa identificar como os colaboradores receberam esse novo processo de desenvolvimento e quais processos poderiam ser melhorados.

Seguindo a metodologia proposta, esse trabalho foi submetido para avaliação, através de um questionário qualitativo, para conduzir à avaliação, o questionário constante no apêndice A foi utilizado. Esse está dividido em três partes: a primeira, com perguntas direcionadas ao contexto geral da implantação; a segunda, direcionada à implantação da metodologia *Scrum* e a terceira, direcionada às questões específicas referentes à implantação do novo processo.

Para responder ao questionário, foram escolhidos três programadores que participaram do processo de implantação do *Scrum* desde o início. Todos responderam ao mesmo questionário, validando os novos processos implantados na empresa e buscando o entendimento das melhorias e avaliação dos resultados do trabalho.

Para a elaboração das questões, foi realizada a adaptação da escala *Likert*, que é avaliada como de fácil entendimento e de muita clareza. A escala *Likert* é uma escala amplamente utilizada que exige que os entrevistados indiquem um grau de concordância ou discordância com cada uma da série de afirmações sobre objetos de estímulo (MALTRORA, 2004).

Para o questionário desse trabalho, a adaptação se deu pelo fato da escala *Likert* indicar opções de resposta que variam de “Totalmente Positiva” até “Totalmente Negativa”.

Outra característica importante aplicada ao questionário foram as categorias ímpares de escala, facilitando a verificação dos resultados para positivos, negativos ou neutros e vinculando os mesmos com o formato de resposta *Likert* que é uma escala balanceada de comparação com números ímpares de categorias e uma posição neutra (MALTRORA, 2004).

As questões utilizadas no questionário aplicado e as imagens do mesmo podem ser visualizadas no Apêndice A. Entre os avaliadores estão: Avaliador A, possui mais de 10 anos de experiência em análise e desenvolvimento de software e atua há mais de 5 anos na empresa avaliada; Avaliador B, possui mais de 2 anos de experiência e de atuação na empresa; Avaliador C, possui menos de 1 ano de experiência e de atuação na empresa.

Todos os avaliadores consideraram que a mudança de processo de desenvolvimento foi, no geral, totalmente positiva. Salientaram que o novo processo de desenvolvimento trouxe ao projeto uma organização inexistente até a implantação da metodologia, além de contribuir positivamente para as estimativas de tempo para desenvolvimento das tarefas, como proporcionar uma visão geral sobre o progresso do desenvolvimento das tarefas.

Também concordando que o novo processo de desenvolvimento trouxe uma melhor distribuição de tarefas, ou seja, a fragmentação do processo contribuiu para um melhor controle de produção, exposição de gargalos e também a possibilidade de se mensurar produtividade na programação.

Concordaram, também, que a mudança de processo, no contexto do gerenciamento de escopo (análise das novas requisições, divisão em tarefas, clareza nos requisitos, documentação), foi totalmente positiva. Salientaram principalmente a organização trazida pela nova metodologia para o processo de desenvolvimento, assim como a clareza quanto à evolução do produto. Isso se deve ao fato de a nova metodologia possibilitar que o programador acompanhe o seu processo de evolução, contribuindo para sua concepção, diferentemente da antiga metodologia, na qual essas questões eram centralizadas no analista.

Outro ponto apontado foi o *layout* para a visualização das tarefas trazido pelo *Scrum*. Através dele, todos os membros da equipe podem visualizar rapidamente todas as tarefas a serem desenvolvidas, quais estão sendo desenvolvidas e quais foram finalizadas. Possibilitando, dessa forma, que todos os envolvidos no projeto acompanhem o seu progresso, fato este que, na antiga metodologia, também era restrita ao analista.

Todos os avaliadores avaliaram como parcialmente positiva a mudança de processo no contexto do gerenciamento de qualidade (validação das implementações, gerenciamento dos *releases*). Isso se deve ao fato da realização dos testes nos softwares, realizados por uma pessoa específica, ser uma exigência dos programadores, os quais, muitas vezes, por estarem direcionados em uma tarefa, têm dificuldades em realizar os testes e abordar todas as possíveis ações de sua implementação.

Salientaram que a forma de como a realização dos testes vem sendo abordada atualmente, apesar de melhor que na metodologia antiga, ainda não é a ideal. Com a análise dos questionários foi possível identificar que os testes estão se tornando um gargalo para o novo processo. Inicialmente, quando o processo foi concebido, o responsável pelo suporte possuía tempo hábil para a realização dos testes, o que, devido ao aumento no número de clientes, não é mais possível.

Quanto à avaliação da mudança de processo no contexto do gerenciamento de configuração (controle de versões, registro das mudanças, integração contínua), todos os avaliadores concordaram que a mudança foi totalmente positiva. Salientaram que a rastreabilidade, tanto do *Subversion* quanto da gestão de mudanças (JIRA), é um ponto positivo, pois possibilita que os programadores trabalhem de forma segura e tranquila, possibilitando a rápida resposta a problemas que possam ocorrer.

Outro ponto positivo foi a integração contínua, a qual foi citada dentre as respostas como sendo um facilitador para a realização dos testes. Antes da implantação da integração contínua os programadores se negavam a realizar o processo de *build* e de entrega de versão de testes, pois era muito demorado. Também foi citado, nas respostas, que o processo já está ajudando a resolver problemas. Um dos avaliadores citou um problema ocorrido em um ambiente de produção, em que, depois de instalada a nova versão, ele foi detectado e facilmente corrigido com a utilização das ferramentas de gestão de configuração, sendo que, sem elas, os programadores perderiam algum tempo explorando as possíveis causas do problema.

Quanto ao processo de implantação do *Scrum* na empresa, resultante da proposta desse trabalho, dois dos avaliadores concordaram que o processo de implantação foi totalmente positivo e um deles indicou como parcialmente positivo. A indicação do parcialmente

positivo deu-se devido ao avaliador não concordar completamente com a necessidade das reuniões diárias e do tempo dos *Sprints*, os quais serão comentados nas próximas questões.

Todos os avaliadores concordaram que a adaptação sugerida pela abordagem das *Daily Meetings* foi parcialmente positiva. Essa questão foi elaborada, pois, durante o processo de implantação do *Scrum*, foi identificado que os membros da equipe não estavam se adaptando às rotinas de reuniões diárias, por outro lado, houve um aumento da comunicação dentre os membros da equipe, a qual era inexistente no processo antigo.

Com base no contexto da equipe, a mesma é composta por dois membros relativamente novos e outro mais experiente, sendo possível notar que os membros da equipe possuem dificuldades de expor seus problemas durante a reunião. Sendo compensado pela comunicação que ocorre durante o desenvolvimento das tarefas. Ou seja, os membros da equipe optam por discutir seus problemas com os outros membros da equipe fora das reuniões diárias.

Os avaliadores foram questionados quanto a sugestões para melhorar as *Daily Meeting*. Todos sugeriram a diminuição quanto ao número de reuniões realizadas. Indicando assim, a necessidade de reavaliação da forma de como as reuniões vem sendo aplicadas com o objetivo de melhoras e para que todos os membros possam tirar um melhor proveito dessa dinâmica.

Quanto ao tempo de duas semanas para a realização do *Sprint*, um dos avaliadores indicou como parcialmente positivo, outro indicou como parcialmente e outro como parcialmente negativo. Essa oscilação nas respostas dos avaliadores se deve ao fato de que os mesmos estavam acostumados a trabalhar sem um prazo definido e ainda não se sentem totalmente à vontade de, no final de duas semanas, terem um produto pronto para entrega. A cada *Sprint* realizado, a equipe está se adaptando à nova rotina e amadurecendo perante o novo processo.

Dois dos avaliadores concordaram que a avaliação da comunicação entre os membros da equipe foi totalmente positiva e um deles avaliou como parcialmente positiva. Como citado anteriormente, a comunicação entre os membros da equipe praticamente inexistente, agora ocorre de forma natural. Quanto à avaliação parcialmente positiva, deve-se ao fato de que os membros da equipe se comunicam durante o desenvolvimento das tarefas, e durante as reuniões não portam a mesma postura. Para reverter esta situação, e encorajar a comunicação

deverão ser realizadas dinâmicas durante as reuniões, para que os membros da equipe percam o medo de falar.

Os avaliadores, questionados quanto aos pontos positivos da antiga metodologia que poderiam ser aproveitados pela nova, concordaram que aquela não traz nenhum ponto.

5 CONCLUSÃO

Todos os objetivos estabelecidos na metodologia foram cumpridos, realizando-se a revisão do referencial teórico, a teorização dos conceitos abordados e a implantação das práticas do *Scrum*, juntamente com a gestão de configuração no processo de desenvolvimento de software da empresa.

Durante a revisão da bibliografia, destacou-se a importância da Engenharia de Software para o desenvolvimento de software, assim como os principais processos que visam a dar suporte e qualidade para a entrega do produto final. Esses processos auxiliam no aumento da produtividade e, consequentemente, na redução dos custos de desenvolvimento e na diminuição do tempo de retorno aos clientes.

Pode-se verificar que a implantação do *framework* apresentou resultados positivos para os processos de gerenciamento de configuração, escopo, tempo e qualidade com base na avaliação realizada entre os membros da equipe participantes.

Percebeu-se que a agilidade nos processos de desenvolvimento de software e a entrega constante de *builds* para os clientes contribuem positivamente para a qualidade do software, tendo em vista que o *feedback* é recebido pela equipe horas após o seu lançamento, contribuindo para a melhoria contínua imposta pela forma iterativa de implementação.

O *feedback* recebido pelas avaliações realizadas junto aos membros da equipe de desenvolvimento aponta a aceitação pela metodologia, os quais, interessados em contribuir e melhorar o processo, sugeriram mudanças no processo, como a diminuição nas reuniões realizadas e a necessidade de um membro focado na realização de testes. Pontos estes que serão discutidos com a equipe para verificar a melhor solução.

Quanto à comparação do antes e depois do processo de desenvolvimento de software, ficaram visíveis as melhoras trazidas pela nova metodologia como a melhora no acompanhamento do projeto, a realização dos testes realizados logo após o término do projeto e a melhora na comunicação entre os membros da equipe. Questões essas visíveis e que também foram citadas positivamente na avaliação.

Por se tornar uma equipe auto gerenciável, o processo de adaptação para novos funcionários deverá ser facilitado. Pois ao invés de somente o analista tirar duvidas e auxiliar no processo de adaptação, todos os membros da equipe terão liberdade para fazê-lo.

REFERÊNCIAS

AGILE MANIFESTO. ***Manifesto for Agile Software Development***. Disponível em: <http://www.agilemanifesto.org>. Acesso em Fevereiro de 2013.

BRADAC, M.; D. PERRY, L. VOTTA. ***Prototyping a Process Monitoring Experiment***. IEEE - Software Engineering. 1994.

COHN, Mike. **Desenvolvimento de software com scrum**: Aplicando métodos ágeis com sucesso. 1ªed. Porto Alegre: Bookan, 2011.

DALL'OGLIO, Pablo. **Uma Ferramenta para Gerenciamento de Requisito sem Projetos Baseados em Extreme Programming.** 2006.

HIGHSMITH, J.; COCKBURN, A. ***Agile Software Development****: The Business of Innovation*. IEEE Computer. 2001.

IEEE,***Institute of Electrical and Electronics Engineers****: Software Engineering*, IEEE Standard 610.12 – 1990, IEEE, 1993.

KERZNER, Harold. **Gestão de projetos**: as melhores práticas. 2.ed. Porto Alegre: Bookman, 2006.

KNIBERG, Henrik. **Scrum e XP direto das Trincheiras** – Como Fazemos Scrum. Publisher C4Media editor of InfoQ.com, 2007.

MALHOTRA, Naresh. K. **Pesquisa de Marketing uma orientação aplicada.** 4ªed. São Paulo. Artmed, 2004.

PMI (Project Management Institute). **A Guide to the Project Management Body of Knowledge** (PMBOK). 4ªed. PMI Standard, 2008.

PRESSMAN, Roger S. **Engenharia de Software**. 6ª Ed. São Paulo: McGraw-Hill, 2006.

SBROCCO, José Henrique Teixeira de Carvalho; MACEDO, Paulo Cesar. **Metodologias ágeis**: engenharia de software sob medida. 1ª ed. São Paulo: Érica, 2012.

SCHRAMM, Wilbur. ***Notes on case studies of instructional media projects***. Working paper, the Academy for Educational Development, Washington, DC, 1971.

SCHWABER, Ken. ***Agile Project Management with Scrum***. Redmond, Washington: Microsoft Press, 2004.

SOMMERVILLE, Ian. ***Software engineering***. 9ª ed. São Paulo: Pearson Education - BR, 2011.

SUTHERLAND, Jeff. ***The Scrum Papers****: Nut, Bolts, and Origins of an Agile Framework*. Version 1.1, Scrum, Inc., Cambridge, 2012. Disponível em: <http://jeffsutherland.com/scrumpapers.pdf>. Acesso em abril 2013.

SWEBOK. ***The Guide to the Software Engineering Body of Knowledge***. Disponível em: <http://www.swebok.org/>. Acesso em Abril 2013.

VALLE, André Bittencourt do; SOARES, Carlos Alberto Pereira; FINOCCHIO JR., José; SILVA, Lincoln de Souza Firmino da. **Fundamentos do Gerenciamento de Projetos**. 2ª Ed. Rio de Janeiro: Editora FGV, 2007.

WAINER, Jacques. **Métodos de pesquisa quantitativa e qualitativa para a Ciência da Computação**. In: KOWALTOWSKI, Tomasz; BREITMAN, Karin; organizadores. Atualizações em Informática 2007. Rio de Janeiro: Ed. PUC-Rio; Porto Alegre: SociedadeBrasileira de Computação, 2007.Rising, L.; Janoff, N. (2000) The Scrum software development process for small teams. In IEEE, v. 17, nº 4.7

YIN, Robert K. **Estudo de caso**: planejamento e métodos. 3ª ed. Porto Alegre: Bookman, 2005.

APÊNDICE A – Questionário aplicado ao avaliador A

Abaixo estão listadas as respostas do questionário aplicado ao avaliador A.

1. Geral

1.1 Você considera que a mudança de processo de desenvolvimento foi, no geral:

(x) Totalmente positiva
( ) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

1.2 Quais seriam os principais pontos (positivos/negativos) a se destacar, conforme seu posicionamento na questão anterior? (dissertativa)

A mudança no processo de desenvolvimento fez com que o trabalho ficasse melhor organizado e trouxe informações importantes como, por exemplo, o tempo necessário no desenvolvimento e aquilo que já foi desenvolvido.

2. *Scrum*

2.1 Como você avalia a mudança de processo no contexto do gerenciamento de escopo (análise das novas requisições, divisão em tarefas, clareza nos requisitos, documentação)?

(x) Totalmente positiva
( ) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

2.2 Quais seriam os principais pontos (positivos/negativos) a se destacar, conforme seu posicionamento na questão anterior? (dissertativa)

A divisão em tarefas, pois facilita o desenvolvimento e garante mais organização no trabalho.

2.3 Como você avalia a mudança de processo no contexto do gerenciamento de tempo (delegação de tarefas, estimativas de tempo, acompanhamento do desenvolvimento)?

(x) Totalmente positiva
( ) Parcialmente positiva

( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

2.4 Quais seriam os principais pontos (positivos/negativos) a se destacar, conforme seu posicionamento na questão anterior? (dissertativa)

A estimativa de tempo é uma informação muito importante para o planejamento do desenvolvimento. Com o controle do tempo é possível saber quanto tempo foi gasto em cada tarefa e assim saber qual a produtividade da empresa.

2.5 Como você avalia a mudança de processo no contexto do gerenciamento de qualidade (validação das implementações, gerenciamento dos releases)?

( ) Totalmente positiva
(X) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

2.6 Quais seriam os principais pontos (positivos/negativos) a se destacar, conforme seu posicionamento na questão anterior? (dissertativa)

Após o término de cada tarefa, aquilo que foi desenvolvido já é encaminhado diretamente para ser testado, o que garante que *bugs* sejam encontrados e corrigidos rapidamente, agilizando a produção da empresa. Sendo o principal ponto negativo o gargalo para realização de testes que muitas vezes atrasa o processo de desenvolvimento, ou devido a pressa para sua conclusão, acaba deixando escapar alguns *bugs* para o ambiente de produção.

2.7 Como você avalia a mudança de processo no contexto do gerenciamento de configuração (controle de versões, registro das mudanças, integração contínua)?

(x) Totalmente positiva
( ) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

2.8 Quais seriam os principais pontos (positivos/negativos) a se destacar, conforme seu posicionamento na questão anterior? (dissertativa)

É muito importante saber o que foi alterado, quando, e quem fez a alteração, pois através deste histórico fica mais fácil encontrar as causas dos *bugs* no sistema. Recentemente passamos por um *bug* em ambiente de produção do qual não possuíamos ideia do que o estava causando. O qual pode ser resolvido rapidamente após uma pesquisa pelo gerenciamento de mudanças, onde detectamos qual revisão estava sendo utilizada pelo cliente com base na sua versão e no SVN podemos identificar as alterações realizadas. Sem esta ferramenta teríamos de explorar as possíveis causa do problema, já com ela foi possível atacar o problema na fonte.

3. Específica

3.1 Como você avalia o processo de Implantação do *Scrum* na empresa?

(x) Totalmente positiva
( ) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

3.2 Como você avalia a adaptação sugerida pela abordagem das *Daily Meetings*?

( ) Totalmente positiva
(x) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

3.3 Qual sua sugestão para melhorar a *Daily Meeting*? (dissertativa)

Diminuir o número de reuniões. Acredito que não sejam necessárias tantas reuniões.

3.4 Como você avalia o tempo de duas semanas para realização dos *Sprints*?

(x) Totalmente positiva
( ) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

3.5 Como você avalia a comunicação entre os membros da equipe após a implantação do *Scrum?*

(x) Totalmente positiva
( ) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

3.6 Quais pontos positivos da antiga metodologia você traria para a nova? (dissertativa)

Nenhum.

APÊNDICE B – Questionário aplicado ao avaliador B

Abaixo estão listadas as respostas do questionário aplicadas ao avaliador B.

1. Geral

1.1 Você considera que a mudança de processo de desenvolvimento foi, no geral:

(x) Totalmente positiva
( ) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

1.2 Quais seriam os principais pontos (positivos/negativos) a se destacar, conforme seu posicionamento na questão anterior? (dissertativa)

A mudança no geral levou a um novo modo de trabalho, tanto em estrutura quanto em organização, tendo melhor controle do tempo e estimativas de tarefas.

2. *Scrum*

2.1 Como você avalia a mudança de processo no contexto do gerenciamento de escopo (análise das novas requisições, divisão em tarefas, clareza nos requisitos, documentação)?

(x) Totalmente positiva
( ) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

2.2 Quais seriam os principais pontos (positivos/negativos) a se destacar, conforme seu posicionamento na questão anterior? (dissertativa)

Os principais pontos positivos são os que foram destacados no próprio enuncia da questão anterior, como: análise das novas requisições, divisão de tarefas, clareza nos requisitos, ótimo layout para visualização das tarefas, entre outros. A utilização da ferramenta faz com que o trabalho seja desenvolvido com mais organização.

2.3 Como você avalia a mudança de processo no contexto do gerenciamento de tempo (delegação de tarefas, estimativas de tempo, acompanhamento do desenvolvimento)?

(x) Totalmente positiva
( ) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

2.4 Quais seriam os principais pontos (positivos/negativos) a se destacar, conforme seu posicionamento na questão anterior? (dissertativa)

O controle de tempo é muito bom, tendo um bom controle, principalmente, do tempo que levou para realizar cada tarefa. Sabe-se quem está fazendo o que, as tarefas que já foram realizadas, as que levaram mais tempo e as que ainda estão na fila de espera de desenvolvimento. Dados importantes para a melhor produtividade.

2.5 Como você avalia a mudança de processo no contexto do gerenciamento de qualidade (validação das implementações, gerenciamento dos releases)?

( ) Totalmente positiva
(X) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

2.6 Quais seriam os principais pontos (positivos/negativos) a se destacar, conforme seu posicionamento na questão anterior? (dissertativa)

Após cada implementação feita no sistema, já é automaticamente lançada uma versão de teste, o que facilita para quem vai testar, podendo dar um retorno positivo ou negativo o quanto antes.

2.7 Como você avalia a mudança de processo no contexto do gerenciamento de configuração (controle de versões, registro das mudanças, integração contínua)?

(x) Totalmente positiva
( ) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

2.8 Quais seriam os principais pontos (positivos/negativos) a se destacar, conforme seu posicionamento na questão anterior? (dissertativa)

O acompanhamento realizado pelo *Scrum*, juntamente com o gerenciamento realizado pelo JIRA foi um casamento perfeito. Tanto em mudanças quanto integração é realmente muito bom. Tem-se o controle de cada coisa que foi alterada, tendo comparações com registros antigos e novos, sabendo quem alterou o que, além de ter todo o histórico de versões antigas.

3. Específica

3.1 Como você avalia o processo de Implantação do *Scrum* na empresa?

(x) Totalmente positiva
( ) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

3.2 Como você avalia a adaptação sugerida pela abordagem das *Daily Meetings*?

( ) Totalmente positiva
(x) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

3.3 Qual sua sugestão para melhorar a *Daily Meeting*? (dissertativa)

As reuniões facilitam para o melhor desenvolvimento por serem rápidas e eficazes em sua maior quantidade, onde todos podem opinar e ver como vão realizar as tarefas, entretanto, alguns casos específicos que não necessitariam delas.

3.4 Como você avalia o tempo de duas semanas para realização dos *Sprints*?

( ) Totalmente positiva
(x) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

3.5 Como você avalia a comunicação entre os membros da equipe após a implantação do *Scrum*?

( ) Totalmente positiva
(x) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

3.6 Quais pontos positivos da antiga metodologia você traria para a nova? (dissertativa)

Nenhum, creio que esta metodologia está mais funcional em todos os aspectos do que a última.

APÊNDICE C – Questionário aplicado ao avaliador C

Abaixo estão listadas as respostas do questionário aplicadas ao avaliador C.

1. Geral

1.1 Você considera que a mudança de processo de desenvolvimento foi, no geral:

(X) Totalmente positiva
( ) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

1.2 Quais seriam os principais pontos (positivos/negativos) a se destacar, conforme seu posicionamento na questão anterior? (dissertativa)

Com o novo processo de desenvolvimento existe uma melhor distribuição de tarefas ou seja, a fragmentação do processo contribui para um melhor controle de produção, exposição de gargalos e também a possibilidade de se mensurar produtividade na programação. Isso, eu, como programador, vejo como uma ferramenta de qualidade no produto final.

2. *Scrum*

2.1 Como você avalia a mudança de processo no contexto do gerenciamento de escopo (análise das novas requisições, divisão em tarefas, clareza nos requisitos, documentação)?

(X) Totalmente positiva
( ) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

2.2 Quais seriam os principais pontos (positivos/negativos) a se destacar, conforme seu posicionamento na questão anterior? (dissertativa)

Delegação, responsabilidade e organização de produção. O processo é concebido e distribuído em tarefas de responsabilidade, o que favorece o controle do desenvolvimento com qualidade.

2.3 Como você avalia a mudança de processo no contexto do gerenciamento de tempo (delegação de tarefas, estimativas de tempo, acompanhamento do desenvolvimento)?

(X) Totalmente positiva
( ) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

2.4 Quais seriam os principais pontos (positivos/negativos) a se destacar, conforme seu posicionamento na questão anterior? (dissertativa)

Uma das principais benesses do novo processo é a estimativa de tempo, onde podemos, como já me referi, mensurar produtividade. A gerência pode executar um planejamento de programação com base em uma produtividade já definida.

2.5 Como você avalia a mudança de processo no contexto do gerenciamento de qualidade (validação das implementações, gerenciamento dos releases)?

( ) Totalmente positiva
(X) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

2.6 Quais seriam os principais pontos (positivos/negativos) a se destacar, conforme seu posicionamento na questão anterior? (dissertativa)

Os testes realizados visam ajudar os programadores a identificar problemas de codificação. É importante que os testes sejam realizado logo após o término do desenvolvimento, pois assim que identificados podem ser rapidamente corrigidos. O atual processo de teste deve ser revisto, pois acaba se tornando um gargalo, visto que a pessoa que realiza os teste esta encarregada de outros afazeres, atrasando assim a realização dos testes.

2.7 Como você avalia a mudança de processo no contexto do gerenciamento de configuração (controle de versões, registro das mudanças, integração contínua)?

(X) Totalmente positiva
( ) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

2.8 Quais seriam os principais pontos (positivos/negativos) a se destacar, conforme seu posicionamento na questão anterior? (dissertativa)

Um dos principais pontos positivos almejados com a implantação deste novo processo foram as ferramentas de controle que servem de alicerce para todo o processo. Com a implantação do controle de versão (SVN), gestão de mudanças (JIRA) e integração continua (Bamboo) a equipe pode trabalhar de forma sincronizada e sem preocupação. No antigo processo o versionamento ocorria de forma manual e não era possível identificar as alterações realizadas no sistema e nem quem as realizou. Com estas ferramentas podemos rastrear todas as mudanças realizadas no sistema, o que possibilita que a inclusão de novos membros na equipe de desenvolvimento ocorra de forma mais tranquila, diminuindo a preocupação quanto a rastreabilidade de seu serviço. Já a integração continua trouxe comodidade, visto que no antigo processo era necessário realizar o procedimento manualmente, hoje ocorre de forma automática.

3. Específica

3.1 Como você avalia o processo de Implantação do *Scrum* na empresa?

( ) Totalmente positiva
(X) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

3.2 Como você avalia a adaptação sugerida pela abordagem das *Daily Meetings*?

( ) Totalmente positiva
(X) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

3.3 Qual sua sugestão para melhorar a *Daily Meeting?* (dissertativa)

O número de reuniões podem se tornar perdas de tempo. O planejamento já estaria equacionado e quaisquer impasse ou dificuldade encontrada, ao meu ver, deve ser questionada ao supervisor. Um número de reuniões excessivas prejudicam.

3.4 Como você avalia o tempo de duas semanas para realização dos *Sprints?*

( ) Totalmente positiva
( ) Parcialmente positiva
( ) Indiferente
(X) Parcialmente negativa
( ) Totalmente negativa

3.5 Como você avalia a comunicação entre os membros da equipe após a implantação do *Scrum?*

(X) Totalmente positiva
( ) Parcialmente positiva
( ) Indiferente
( ) Parcialmente negativa
( ) Totalmente negativa

3.6 Quais pontos positivos da antiga metodologia você traria para a nova? (dissertativa)

Não traria nada.