UNIVERSIDADE DO SUL DE SANTA CATARINA

DANIELLE MOREIRA ALEXANDRE

QUALIDADE DE SOFTWARE E O FRAMEWORK SCRUM: UMA PROPOSTA PARA MELHORAR A INTEGRAÇÃO DOS TESTES NO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE DA EMPRESA PP LTDA

Florianópolis

2016

DANIELLE MOREIRA ALEXANDRE

QUALIDADE DE SOFTWARE E O FRAMEWORK SCRUM: UMA PROPOSTA PARA MELHORAR A INTEGRAÇÃO DOS TESTES NO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE DA EMPRESA PP LTDA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação da Universidade do Sul de Santa Catarina, como requisito parcial à obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Flávio Ceci, Dr.

Florianópolis

2016

DANIELLE MOREIRA ALEXANDRE

QUALIDADE DE SOFTWARE E O FRAMEWORK SCRUM: UMA PROPOSTA PARA MELHORAR A INTEGRAÇÃO DOS TESTES NO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE DA EMPRESA PP LTDA

Este Trabalho de Conclusão de Curso foi julgado adequado à obtenção do título de Bacharel em Sistemas de Informação e aprovado em sua forma final pelo Curso de Graduação em Sistemas de Informação da Universidade do Sul de Santa Catarina.

Florianópolis, 18 de Novembro de 2016.

Professor e orientador Flávio Ceci, Dr.
Universidade do Sul de Santa Catarina

Profa. Maria Inés Castiñeira, Dra.
Universidade do Sul de Santa Catarina

Nélio Herzmann Junior, Esp.
Universidade do Sul de Santa Catarina

Dedico essa monografia a minha mãe, Lucia Moreira de Aquino, ao meu esposo, Guere Elton da Silva, aos meus amigos Clarissa Grando e Patrick Varela e ao professor Flávio Ceci, os quais ajudaram de alguma forma, esse sonho a se tornar realidade. Muito obrigada!

AGRADECIMENTOS

Inicialmente agradeço a Deus por guiar o meu caminho e me proporcionar à chance de cursar uma graduação e concretizar um sonho.

Ao professor Flavio Ceci, meu orientador, pelo auxílio, contribuições, orientações e amizade.

A professora Maria Inés pela ajuda, atenção e apoio durante toda a confecção da monografia.

Aos meus amigos, Clarissa Grando e Patrick Varela, os quais fizeram meus dias melhores, durantes esses quatro anos e meio de graduação.

A todas as pessoas da empresa, PP LTDA, que contribuíram e possibilitaram a execução deste trabalho.

Aos convidados da banca por aceitarem o convite e, por dispor o tempo para contribuir com o seu conhecimento e experiência.

A todos os professores do curso de sistemas de informação da Universidade do Sul de Santa Catarina, que contribuíram para a formação acadêmica.

Por fim, agradeço a minha mãe, Lucia Moreira de Aquino, e ao meu esposo Guere Elton da Silva por me apoiarem durante os momentos difíceis. Obrigada pela compreensão, carinho e o amor de vocês, essa vitória é nossa!

“Qualidade não é um ato, é um hábito”. (Aristóteles)

RESUMO

O *software* é uma ferramenta que busca atender as necessidades de processamento de informações dos usuários de um determinado sistema. Durante o processo de desenvolvimento do mesmo, podem ocorrer diversas falhas e para que estas não sejam percebidas pelos clientes, é necessário ter os testes integrados em todo processo, garantindo assim, a qualidade do produto. Além de encontrar *bugs*, o teste principalmente previne-os. Contudo, toda a equipe deve estar engajada para atender as necessidades dos clientes da melhor forma possível e assim gerar valor para a solução. Com o objetivo de aplicar e avaliar as melhorias propostas para integrar os testes ao processo de desenvolvimento, e também para o processo de forma geral, esta monografia realizou um estudo de caso dentro de uma empresa desenvolvedora de *software* que utiliza o *framework Scrum*. O processo atual foi modelado e as melhorias foram identificadas. Os resultados obtidos após a implantação das melhorias nos projetos da empresa e a avaliação dos colaboradores da mesma, demonstram que a melhor integração dos testes ao *Scrum* gerou benefícios satisfatórios para o processo de desenvolvimento de *software* como um todo.

Palavras-chave: Teste de *software.* Processo de *software*. *Scrum*.

ABSTRACT

The software is a tool that seeks to meet the information processing needs of users of a given system. During the development process of the same, several failures can occur and that they are not perceived by customers, it is necessary to have the integrated testing throughout the process, thus ensuring the product quality. In addition to finding bugs, the test mainly prevents them. However, all staff should be engaged to meet customer needs in the best possible way and thus generate value for the solution. In order to implement and evaluate the proposed improvements to integrate testing into the development process, and also to the process in general, this monograph conducted a case study in a software development company using the Scrum framework. The current process was modeled and improvements were identified. The results obtained after the implementation of improvements in the company's projects and the evaluation of the employees of the same, show that better integration of testing the Scrum generated satisfactory benefits to the software development process as a whole.

Keywords: Software testing. Software process. Scrum.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Na economia moderna, as condições do mercado mudam rapidamente. Dessa forma, é difícil ou impossível prever como um sistema computacional vai evoluir com o passar do tempo, uma vez que, as necessidades dos usuários se alteram. (PRESSMAN; MAXIM, 2016).

As empresas operam em um ambiente com mudanças rápidas e assim, se torna difícil possuir um conjunto completo de requisitos de *software* estável. (SOMMERVILLE, 2011). Visto que, é necessário que os processos possam se adaptar, sejam manipuláveis e sem excesso, de modo que contenham apenas o conteúdo essencial para se adequar ás necessidades do mundo dos negócios modernos. (PRESSMAN; MAXIM, 2016).

Logo, já se reconhecia a necessidade de desenvolvimento rápido de processos que fossem capazes de atender as mudanças de requisitos. Contudo, a ideia decolou somente no final da década de 90, com o desenvolvimento da noção de abordagens ágeis. (SOMMERVILLE, 2011). Em fevereiro de 2001, um grupo de profissionais de desenvolvimento de *software* se reuniu em uma estação de Ski para discutir melhores maneiras de se desenvolver *softwares*. Esse encontro deu origem ao manifesto ágil, uma declaração com os princípios que regem o desenvolvimento ágil. (GOMES, 2014).

Existem diversos métodos e processos ágeis. Por sua vez, a 10ª edição da pesquisa realizada em 2015 pela VERSIONONE aponta que o *Scrum* é seguido por 58% das empresas participantes. Segundo Rubin (2012, tradução nossa), o *framework Scrum* foi desenvolvido em 1993 por Ken Schwaber e Jeff Sutherland e “ele substitui o planejamento abrangente com a capacidade de reagir rapidamente, de forma flexível e adequada para eventos de projeto”. (LINZ, 2014, p.42). Desse modo, o *Scrum* foi escolhido para ser abordado nessa monografia.

No *Scrum*, a equipe deve integrar os testes ao seu trabalho diário, ao invés de testar a qualidade do produto no final. (COHN, 2011).

Segundo Cohn (2011, p. 343):

> A ênfase na qualidade pode ter resultados drásticos. Nove meses após adotar o Scrum, Steve Greene, da Salesforce.com, diz que sua empresa já atingiu a marca de “mais de 300 homens-hora economiza- dos a cada grande release e centenas mais no tempo de vida de todos os releases de patches. “É muito tempo que agora gastamos em requisito, automação, design, basicamente todas as coisas boas e produtivas”.

Desse modo, o investimento em qualidade é essencial, pois a redução de falhas no sistema em produção é refletida diretamente na percepção de valor, satisfação do cliente, redução de custos e aumento da produtividade.

1.1 PROBLEMÁTICA

Os modelos de processos prescritivos, também conhecidos como tradicionais, tem o objetivo de estruturar e ordenar o desenvolvimento de *software*, uma vez que, as atividades e tarefas ocorrem sequencialmente e com diretrizes de processos bem definidas. (PRESSMAN; MAXIM, 2016).

Nesse contexto, os testes realizados por meios manuais e automáticos ocorrem após o desenvolvimento de *software* por equipes especialistas e independes, com enfoque em testes de caixa preta na perspectiva da interface gráfica. Os testes são usados como roteiro para a execução manual realizada por um testador com ênfase em testes planejados detalhadamente e execução baseada em roteiro. (CAETANO, 2016)

Diante do cenário anterior, é difícil avaliar e melhorar a qualidade do produto existente, os erros continuam não sendo notados e a oportunidade de *feedback* são perdidas, visto que, os testes ocorrem apenas no final do processo de desenvolvimento de *software* e o tempo destinado para os mesmos fica muito apertado ou não existe devido as pressões do cronograma de entrega. (COHN, 2011).

Tendo em vista essa situação, chega-se a seguinte pergunta de pesquisa: é possível melhorar a integração dos testes ao processo de desenvolvimento de uma empresa desenvolvedora de *software* que utiliza o *framework Scrum*?

1.2 OBJETIVOS

São apresentados, a seguir, os objetivos desse trabalho.

1.2.1 Objetivo Geral

Propor e avaliar as melhorias na integração dos testes ao processo de desenvolvimento de *software* em projetos, de uma empresa, que utilizam o *framework Scrum*.

1.2.2 Objetivos Específicos

Como objetivos específicos, são apresentados os seguintes:

- Identificar como os testes de *software* podem se integrar melhor ao *Scrum*;
- Modelar o processo de desenvolvimento de *software* atual da empresa.
- Propor melhorias para o processo de desenvolvimento de *software* da empresa.
- Modelar o processo de desenvolvimento de *software* da empresa com a inclusão das melhorias;
- Apresentar um conjunto de ferramentas para auxiliar os testes no *Scrum*.
- Implantar as melhorias propostas em uma empresa de *software* a partir dos processos e ferramentas identificadas;
- Avaliar junto aos profissionais da área de tecnologia da informação da empresa as melhorias implantadas.

1.3 JUSTIFICATIVA

Os métodos ágeis surgiram como uma maneira diferente de desenvolver *softwares*, "a maioria deles se utiliza de ciclos curtos, que são chamados de iterações e normalmente têm duração de poucas semanas, dessa forma garantindo *feedback* frequentes e respostas rápidas às mudanças". (GOMES, 2014, p. 4).

Na abordagem ágil os testes são realizados preferencialmente por meio automático por todos os membros do time durante o desenvolvimento do *software* em ciclos frequentes e contínuos. Os testes são usados como complemento dos requisitos e documentação do código com ênfase em testes exploratórios. (CAETANO, 2016)

Um dos maiores benefícios em integrar o teste durante todas as etapas do processo de desenvolvimento de *software* é a detecção de defeitos na fase inicial, diminuindo o tempo e esforço para corrigi-lo, o que aumenta a produtividade do time. (COHN, 2011).

Segundo Cohn (2011, p. 328):

> As equipes Scrum tornam o teste uma prática e uma parte centrais do processo de desenvolvimento em vez de algo que ocorre após os desenvolvedores terem "acabado". Em vez de testarmos a qualidade depois de um produto ter sido construído, ela deve ser incorporada ao processo e ao produto à medida que ele é desenvolvido.

No *Scrum, framework* que será abordada nesta monografia, o testador ágil realiza tarefas no planejamento, execução, revisão e retrospectiva da *Sprint* e participa das reuniões diárias. Logo, colabora com pessoas técnicas e do negócio, assim como entende a importância de usar os testes para guiar o desenvolvimento e ajudar a documentar os requisitos. (CAETANO, 2016).

Sendo assim, à medida que mais organizações transitam para algumas metodologias ágeis de desenvolvimento de *software*, é essencial o entendimento de como os testes se encaixam no processo e ajudam o time a melhorar a qualidade do produto. Além disso, o *Scrum* não fornece orientação sobre como os testes devem ser realizados em um projeto. (BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD, 2014).

1.4 ESTRUTURA DA MONOGRAFIA

Este trabalho está dividido em cinco capítulos. A seguir, será apresentada a sua estrutura:

- Capítulo 1 – Introdução: apresenta a introdução, problema, os objetivos e a justificativa do trabalho.

● Capítulo 2 – Revisão bibliográfica: este capítulo apresenta temas relacionados à processo de desenvolvimento de *software*, modelo de processo prescritivo, o processo unificado, desenvolvimento ágil de *software*, *Scrum* e teste de *software*.

● Capítulo 3 – Método: caracterização do tipo de pesquisa, etapas metodológicas e delimitações do projeto.

● Capítulo 4 – Desenvolvimento: este capítulo apresenta o desenvolvimento de um estudo de caso. Desse modo, o mesmo contém a modelagem do processo atual de desenvolvimento de *software* da empresa do estudo de caso. Além disso, são apresentadas as melhorias identificadas e uma proposta de novos processos. Por fim, são apresentados os resultados obtidos com a implantação das melhorias e, a avaliação das mesmas.

● Capítulo 5 – Conclusões e trabalhos futuros: este capítulo apresenta as conclusões deste trabalho e sugestões para trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo de revisão bibliográfica, são abordados os assuntos como: processos de desenvolvimento *software*, modelos de processo prescritivo, o processo unificado, desenvolvimento ágil de *software*, *Scrum* e teste de *software*. Todos esses temas estão fundamentados na literatura científica.

Dentre os temas, o foco principal é a integração dos testes em um processo de desenvolvimento de *software*, o qual utiliza o *Scrum*, seguindo os conceitos dos autores utilizados como referência.

2.1 PROCESSO DE SOFTWARE

"Um processo de *software* é um conjunto de atividades relacionadas que levam à produção de um produto de *software*". (SOMMERVILLE, 2011, p. 18). Por sua vez, para Sommerville (2011, p.19), "um modelo de processo de *software* é uma representação simplificada de um processo de *software*. Cada modelo representa uma perspectiva particular de um processo e, portanto, fornece informações parciais sobre ele". Segundo Pressman e Maxim (2016, p. 40) "originalmente, os modelos de processo foram propostos para trazer ordem ao caos existente na área de desenvolvimento de *software*".

Os modelos de desenvolvimento de *software* são importantes para definir, por meio de suas notações, um canal de comunicação uniforme entre os membros da equipe de desenvolvimento. Além disso, os mesmos estabelecem produtos de trabalho padronizados que facilitam as atividades de manutenção de *software* e permitem que novos colaboradores sejam treinados, melhorando a qualidade do *software*. (HIRAMA, 2012).

Existem muitos processos de *software* diferentes, mas todos devem incluir quatro atividades fundamentais para a engenharia de *software*: especificação de *software*, projeto e implementação de *software*, validação de *software* e evolução de *software*. (SOMMERVILLE, 2011).

Desse modo, Sommerville (2011), nos mostra que na especificação de *software* a funcionalidade e as restrições devem ser definidas. Por conseguinte, o *software* deve ser

produzido atendendo às especificações na atividade de projeto e implementação. A validação garante que o *software* atende à demanda do cliente e por fim, o *software* deve evoluir para atender as necessidades de mudanças advindas dos clientes.

Sommerville (2011, p.19) explica que:

> Os processos de software, às vezes, são categorizados como dirigidos a planos ou processos ágeis. Processos dirigidos a planos são aqueles em que todas as atividades são planejadas com antecedência, e o progresso é avaliado por comparação com o planejamento inicial. Em processos ágeis, o planejamento é gradativo, e é mais fácil alterar o processo de maneira a refletir as necessidades de mudança dos clientes.

Nesse sentido, para Pressman e Maxim (2016, p. 40) "cada modelo de processo tenta encontrar um equilíbrio entre as necessidades de pôr ordem em um mundo caótico e a de ser adaptável quando as coisas mudam constantemente".

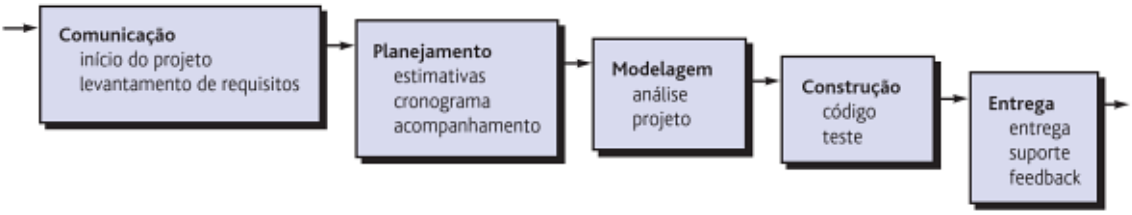
Para Wazlawick (2013), também se pode chamar de "ciclo de vida" os modelos de processos aplicados ao desenvolvimento de *software*. Assim, o autor afirma que, existem duas grandes famílias de modelos: os prescritivos e os ágeis. A seguir, os principais modelos de processos prescritivos e o processo unificado são explicados brevemente, uma vez que, este não é o foco do presente trabalho.

2.1.1 Modelos de processos prescritivos

O modelo de processo prescritivo, também conhecidos como modelos de processos tradicionais, tem o objetivo de estruturar e ordenar o desenvolvimento de *software*, visto que, as atividades e tarefas ocorrem sequencialmente, com diretrizes de processo definidas. (PRESSMAN; MAXIM, 2016). Assim, os modelos abordados serão: o modelo *Waterfall* ou Cascata, o desenvolvimento incremental e os modelos evolucionários.

O modelo *Waterfall* ou Cascata é o mais conhecido entre os modelos de processo prescritivos, cuja característica é a existência de fases bem definidas e sequências como podem verificar na figura 1. Por sua vez, o modelo V, variação do modelo Cascata, enfatiza a importância dos testes em seus vários níveis. (WAZLAWICK, 2013).

Figura 1 - O modelo cascata



Fonte: Pressman e Maxim, (2016, p.42).

O processo Cascata foi muito usado nas décadas de 1970 e 1980 em projetos de grande porte. No entanto, devido às demandas cada vez mais exigentes por produtividade e qualidade, este processo e outros similares foram questionados na década de 1990. Uma das críticas recorrentes é que o processo não é realista, visto que, os requisitos levantados no início do desenvolvimento não são os mesmos do término, pois estes são alterados pelos clientes durante o desenvolvimento de *software*. Logo, o levantamento de requisitos não é completo em muitos projetos e a participação do cliente fica limitada ao início e ao fim do processo. (HIRAMA, 2012).

Pressman e Maxin (2016) explicam que modelo cascata é frequentemente inadequado para trabalhar com *software* que tem um ritmo acelerado e está sujeito a uma cadeia de mudanças intermináveis. Contudo, ele pode servir como modelo de processo útil em situações nas quais os requisitos são fixos e o trabalho é realizado de forma linear até o final.

Há casos em que os requisitos são razoavelmente bem definidos, porém o escopo geral do trabalho não possibilita o uso de um processo puramente linear. (PRESSMAN; MAXIN (2016). Sendo assim, o desenvolvimento incremental baseia-se na ideia de desenvolver um incremento inicial e mostrá-lo ao usuário com objetivo de obter *feedback* dos usuários e continuar criando várias versões até que o sistema adequado seja desenvolvido. (SOMMERVILLE, 2011).

Já, os modelos evolucionários são interativos e apresentam características que permitem desenvolver versões mais completas de *software*. Este modelo foi feito para desenvolver um produto que cresce e muda, uma vez que, conforme o desenvolvimento do projeto avança, os requisitos de negócio são alterados, tornando inadequado seguir um planejamento em linha reta de um produto final. Os prazos apertados determinados pelo mercado, tornam impossível concluir um produto de *software* abrangente, mas uma versão limitada deve ser entregue para atender as pressões comerciais ou da concorrência. (PRESSMAN; MAXIM, 2016).

Segundo Sommerville (2011, p. 30), o protótipo, utilizado no modelo comum prototipação, de processos evolucionários:

> [...] é uma versão inicial de um sistema de software, usado para demonstrar conceitos, experimentar opções de projeto e descobrir mais sobre o problema e suas possíveis soluções. O desenvolvimento rápido e iterativo do protótipo é essencial para que os custos sejam controlados e os *stakeholders* do sistema possam experimentá-lo no início do processo de software.

A prototipação é comumente utilizada nos outros modelos de processo, apesar de poder ser utilizada de forma isolada, pois este paradigma auxilia os envolvidos a entender melhor o que deverá ser construído, quando os requisitos estão obscuros. (PRESSMAN; MAXIM, 2016). "[...] todos os envolvidos devem concordar que o protótipo é construído para servir como mecanismo para definição de requisitos e depois é descartado: o *software* final será arquitetado visando à qualidade". (PRESSMAN; MAXIM, 2016, p.47).

Por sua vez, o modelo Espiral, é uma variação do modelo proposto por Barry Boehm em 1998, no qual as atividades do processo evoluem no sentido horário, do centro para fora. Cada rodada representa um tipo de projeto (HIRAMA, 2012). Segundo Pressman e Maxim (2016, p.48) "nas primeiras interações, a versão pode consistir em um modelo ou em um protótipo. Já nas interações posteriores, são produzidas versões cada vez mais completas do sistema que passa pelo processo de engenharia".

Nesse sentido, segundo Hirama (2012), o modelo espiral é mais realista para o desenvolvimento de *software* de grande porte, pois suas características progressivas possibilitam ao desenvolvedor e o cliente entenderem e reagirem melhor aos riscos em cada nível evolucionário.

Por fim, os modelos de processo prescritivo sugerem fluxos de processos um pouco diferentes, mas com o mesmo objetivo de tentar organizar a estrutura de desenvolvimento de *software*. Todos realizam o mesmo conjunto de atividades metodológicas genéricas: comunicação, planejamento, modelagem, construção e disponibilização. (PRESSMAN; MAXIM, 2016).

2.1.2 O processo unificado

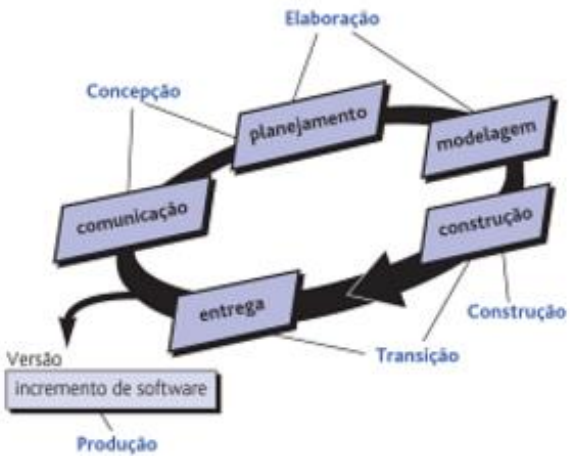
O processo unificado foi desenvolvido para unificar as abordagens existentes e servir como guia para o desenvolvimento de *software*. (HIRAMA, 2012). O mesmo é uma

tentativa de aproveitar as características dos modelos prescritivos, porém implementado muitos dos melhores princípios de desenvolvimento ágil de *software*. (PRESSMAN; MAXIN, 2016).

Um dos exemplos derivados do trabalho de UP e UML é o processo RUP, Processo Unificado Racional, o qual reconhece a importância da comunicação com o cliente e de métodos racionalizados para descrever a visão do mesmo sobre um sistema. (PRESSMAN; MAXIM, 2016).

Segundo Sommerville (2011), o RUP reconhece que os modelos convencionais apresentam uma visão única do processo. Para Pressman e Maxim (2016), o RUP identifica quatro fases distintas no processo, mostradas na figura 2. Contudo, suas fases são estreitamente relacionadas ao negócio, e não a assuntos técnicos.

Figura 2 - O processo unificado



Fonte: Pressman e Maxim (2016, p. 57).

Uma das vantagens do UP/RUP em relação aos demais processos é a desvinculação das fases com o fluxo de trabalho, mesmo porque, eles poderão ser ativados em todos os estágios do processo. Além disso, o modelo reconhece que a implantação do *software* no ambiente do usuário é parte do processo. (HIRAMA, 2012).

2.2 DESENVOLVIMENTO ÁGIL DE SOFTWARE

Para Sommerville (2011), atualmente as empresas operam em um ambiente global com mudanças rápidas. Por conseguinte, é necessário responder a novas oportunidades e

mercados, a mudanças nas condições econômicas e ao surgimento de produtos e serviços concorrentes.

Desse modo, o desenvolvimento e entrega rápida são os requisitos mais críticos para o desenvolvimento de *software*. Uma vez que, o *software* faz parte de todas as operações de negócio, assim novos *softwares* são desenvolvidos rapidamente para obterem proveito de novas oportunidades e responder as pressões competitivas. (SOMMERVILLE, 2011).

Na década de 1980 e início de 1990, acreditava-se que para conseguir o melhor *software* era necessário um planejamento cuidadoso do projeto, qualidade da segurança formalizada, do uso de métodos de análise e projeto apoiado por ferramentas CASE e do processo controlado e rigoroso de desenvolvimento de *software*. (SOMMERVILLE, 2011).

Entretanto, quando essa abordagem tradicional de desenvolvimento de *software* é aplicada aos sistemas corporativos de pequeno e médio porte, o tempo gasto em análise para definir como o processo deve ser desenvolvido é maior do que no desenvolvimento de programas e teste. Naturalmente, os requisitos alteram e o retrabalho torna-se essencial, pois a princípio a especificação e o projeto devem mudam com o problema. (SOMMERVILLE, 2011).

A insatisfação com os modelos tradicionais de desenvolvimento de *software* levou alguns desenvolvedores a propor novas abordagens na década de 1990, as quais resultaram no Manifesto Ágil. (HIRAMA, 212). “Um manifesto normalmente é associado a um movimento político emergente: ataca a velha guarda e sugere uma mudança revolucionária. De certa forma, é exatamente disso que trata o desenvolvimento ágil”. (PRESSMAN; MAXIM, 2016, p.67).

Segundo Sommerville (2011), o manifesto ágil foi acordado por muitos dos principais desenvolvedores desse método, refletindo a sua filosofia. O mesmo afirma que: indivíduos e interação entre eles mais que processos e ferramentas; *software* em funcionamento mais que documentação abrangente; colaboração com o cliente mais que negociação de contratos; responder a mudanças mais que seguir um plano. De modo que, os itens da direita são importantes, mas os itens da esquerda são mais valorizados.

Logo, a filosofia ágil defende a satisfação do cliente e a entrega incremental antecipada. Equipes de projetos pequenas e motivadas, métodos informais, artefatos de engenharia de *software* mínimos e simplicidade no desenvolvimento em geral. Assim, são priorizadas a entrega do projeto e a comunicação ativa e contínua entre os desenvolvedores. (PRESSMAN; MAXIM, 2016).

Apesar de há muitos anos as ideais fundamentais que norteiam o desenvolvimento ágil já estarem conosco, apenas há menos de duas décadas, se consolidaram como um movimento. (PRESSMAN; MAXIM, 2016).

2.2.1 Conceito de agilidade

"Agilidade é um conjunto de métodos e metodologias que ajudam sua equipe a pensar mais efetivamente, trabalhar de forma mais eficiente, e tomar melhores decisões". (STELLMAN; GREENE, 2014, p.2, tradução nossa).

Segundo Pressman e Maxim (2016), nos dias atuais, quando o assunto é processo de *software* moderno, a palavra da moda é agilidade. Uma equipe ágil é aquela capaz de responder de modo adequado às mudanças rapidamente e mudanças está diretamente ligada a desenvolvimento de *software*.

Mudanças de todos os tipos poderão influenciar o produto que está em construção ou no projeto que cria o produto, como por exemplo: mudanças no *software*, nos membros da equipe e mudanças devido a novas tecnologias. Assim, resposta à mudança deve ser parte de tudo que fazemos no *software*, pois é o coração e a alma do mesmo. (PRESSMAN; MAXIM, 2016).

"Fluidez implica mudança, e mudança é cara, particularmente se for sem controle e mal gerenciada. Uma das características mais convincentes da metodologia ágil é sua habilidade de reduzir os custos da mudança no processo de software". (PRESSMAN; MAXIM, 2016, p.67).

"Uma equipe ágil reconhece que o *software* é desenvolvido por indivíduos trabalhando em equipes e que as habilidades dessas pessoas, suas capacidades em colaborar no cerne do sucesso do projeto". (PRESSMAN; MAXIN, 2016, p.68).

Segundo Pressman e Maxim (2016), a *Agile Alliance* estabelece 12 princípios para alcançar a agilidade como pode-se visualizar no quadro 1.

Quadro 1 - Princípios de Agilidade

Princípios de Agilidade

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

Fonte: Pressman e Maxim (2016, p.70).

Desse modo, os princípios listados, definem um espirito ágil mantido nos modelos de processo. No entanto, nem todos os modelos de processos ágeis aplicam todos os princípios, ou atribuem o mesmo peso para os doze. (PRESSMAN; MAXIN, 2016).

Para Pressman e Maxim (2016), os métodos ágeis objetivam sanar as fraquezas existentes na engenharia de *software* tradicionais. O desenvolvimento ágil oferece muitos benefícios, mas não é indicado para todos os projetos, produtos, pessoas e situações. “Pode-se ganhar muito considerando o que há de melhor nas duas escolas e praticamente nada denegrindo uma ou outra abordagem”. (PRESSMAN; MAXIM, 2016, p.72).

2.2.2 Processo ágil

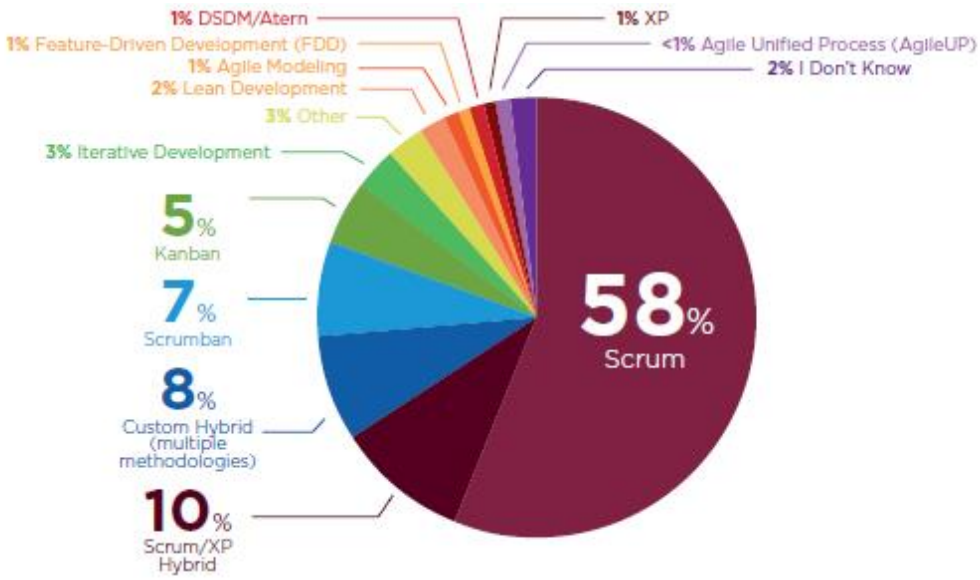
Segundo Pressman e Maxim (2016), um processo ágil deve ser adaptável, uma vez que, é difícil prever quais requisitos de *software* vão ser alterados, quanto de trabalho de projeto será necessário antes que a sua construção seja implantada para avaliar o projeto e a análise, projeto, desenvolvimento e testes não são previsíveis.

Há vários modelos utilizados de processos ágeis no setor, como: *Extreme Programming* (XP), *Scrum*, Métodos de Desenvolvimento de Sistema Dinâmico (DSSD), Modelagem Ágil (AM) e Processo Unificado Ágil (AUP). (PRESSMAN; MAXIM, 2016).

A empresa de *software* VERSIONONE, a qual desenvolve ferramentas para a gestão do ciclo de vida ágil, realiza anualmente uma pesquisa referente ao estado da agilidade. A pesquisa mostra que, o desenvolvimento ágil de *software* tem crescido cada vez mais ao longo da última década. Logo, a participação na pesquisa cresceu mais do que três vezes. Em 2006, havia menos de mil entrevistados na pesquisa, enquanto na última pesquisa havia 3.880.

Na 10ª edição da pesquisa, realizada em 2015, o resultado mostrou que a metodologia ágil mais seguida pelas empresas é o *Scrum*, visto que, quase 70% dos que responderam praticam *Scrum*, (58%) ou híbrida *Scrum* / XP (10%), como podemos visualizar na figura 3.

Figura 3 - Método ágil e práticas



Fonte: VERSIONONE (2015, p. 9).

Desse modo, nesta monográfica o *framework* escolhido foi o *Scrum*.

2.2.3 Scrum

O *Scrum* é "um *framework* dentro do qual, pessoas podem tratar e resolver problemas complexos e adaptativos, enquanto produtiva e criativamente entregam produtos com o mais alto valor possível". (SCHWABER; SUTHERLAND, 2013, p.3).

O *Scrum* compreende que um problema pode não ser totalmente entendido na análise e os requisitos poderão mudam com o passar do tempo. O objetivo é manter o foco na maximização da habilidade da equipe em responder de forma ágil aos desafios emergentes. (PRIKLANDNICKI et al, 2014). Segundo LINZ (2014), as técnicas de desenvolvimento de *software* não são determinadas no *Scrum*, uma vez que, as decisões devem sem tomadas pela própria equipe de desenvolvimento.

Segundo Schwaber e Sutherland (2013, p.4):

> Scrum é fundamentado nas teorias empíricas de controle de processo, ou empirismo. O empirismo afirma que o conhecimento vem da experiência e de tomada de decisões baseadas no que é conhecido. O Scrum emprega uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e o controle de riscos.

Sendo assim, há três pilares que apoiam o controle de processo empírico: transparência, inspeção e adaptação. Por sua vez, a transparência define que os aspectos importantes do processo devem estar visíveis aos responsáveis pelos resultados. A inspeção, deve ser realizada frequentemente pelos usuários *Scrum* com o objetivo de inspecionar os artefatos e o progresso em direção a detectar variações. Por fim, a adaptação deve ser feita o mais rápido possível, caso um ou mais aspectos de um processo sejam desviados para fora dos limites aceitáveis. (SCHWABER; SUTHERLAND, 2013).

2.2.3.1 Artefatos Scrum

Segundo Schwaber e Sutherland (2013, p.13), "os artefatos *Scrum* representam o trabalho ou valor para o fornecimento de transparência e oportunidade para inspeção e

adaptação". Para Prikladnicki et al (2014) há três artefatos, sendo estes, o *Backlog* do Produto, que lista a visão atualizada dos requisitos para o produto, o Incremento do Produto, representado pelas funcionalidades de *software* pronto, e o *Backlog* da Sprint, que representa o planejamento estratégico e tático da próxima *Sprint* em um nível mais macro.

De acordo com Cohn (2011, p. 255):

> Uma equipe Scrum deixa de lado a longa fase inicial de requisitos e favorece uma abordagem just-in-time. As descrições de alto nível dos requisitos podem ser coletadas no início, mas são minimamente descritas nesse momento e progressivamente refinadas à medida que o projeto avança.

Desse como, elas são documentadas em um *backlog* do produto, que corresponde a uma lista de funcionalidades que ainda não fazem parte do produto. Essa lista é mantida pelo dono do produto e colocada em ordem de prioridade. O *backlog* do produto difere de um documento de requisito tradicional por ser altamente dinâmico, ou seja, itens são adicionados, removidos e repriorizados a cada *Sprint*, à medida que se conhece melhor o produto, os clientes, o time e assim por diante. (COHN, 2011).

Contudo, segundo Linz (2014, tradução nossa), um projeto de desenvolvimento de *software* não pode ser controlado apenas por uma lista de requisitos priorizados. Logo, no início de uma *Sprint*, o time leva as tarefas prioritárias do todo da lista do *backlog* do produto para compor o *backlog* da *Sprint*. "Uma vez que a equipe *Scrum*, e não o proprietário do produto seleciona os itens a serem realizados no *Sprint*, a seleção é referida como sendo o princípio da atração e não o princípio do impulso". (BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD, 2014, p.12).

Por fim, "o incremento é a soma de todos os itens do *Backlog* do Produto completados durante a *Sprint* e o valor dos incrementos de todas as *Sprints* anteriores. Ao final da Sprint um novo incremento deve estar Pronto". (SCHWABER; SUTHERLAND, 2013, p.15). Sendo assim, o mesmo deve estar em condição para ser usado, independente do dono do produto decidir por liberá-lo ou não, e atender a definição de "Pronto" do time *Scrum*. (SCHWABER; SUTHERLAND, 2013).

2.2.3.2 O time Scrum

O modelo de time no *Scrum* é projetado para aperfeiçoar a flexibilidade, criatividade e produtividade. Os times de *Scrum* são auto organizáveis e multifuncionais, uma

vez que, escolhem a melhor forma para realizarem o seu trabalho e não são dirigidos por outros de fora do time. Eles possuem todas as competências necessárias para completar o trabalho sem depender de outros que não fazem parte da equipe. (SCHWABER; SUTHERLAND, 2013).

"O *Scrum Master*, garante que as práticas e regras do *Scrum* sejam implementadas e seguidas, e resolve quaisquer violações, questões de funcionalidades, ou outros impedimentos que possam impedir a equipe de seguir as práticas e regras". (BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD, 2014, p.12). Logo, ele existe para ajudar a equipe em seu uso no *Scrum*.

O *Product Owner*, também conhecido como o dono do produto, decide quais são os recursos a serem implementados, ou seja, determina a natureza do produto. (LINZ, 2014). Além disso, representa o cliente, e gera, mantém e prioriza o *backlog* do produto. (BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD, 2014). Desse modo, Cohn (2011), considera que o dono do produto é a pessoa que garante que a equipe se dedique ao objetivo correto, direcionando a equipe para o alvo correto.

Por fim, o time de desenvolvimento, segundo Schwaber e Sutherland (2013, p. 6) "consiste de profissionais que realizam o trabalho de entregar uma versão usável que potencialmente incrementa o produto "Pronto" ao final de cada *Sprint*. Somente integrantes do Time de Desenvolvimento criam incrementos". Os mesmos desenvolvem e testam o produto. O time não possui um líder, pois a equipe toma as decisões. (BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD, 2014).

Segundo Schwaber e Sutherland (2013, p. 6) "O tamanho ideal do Time de Desenvolvimento é pequeno o suficiente para se manter ágil e grande o suficiente para completar uma parcela significativa do trabalho dentro dos limites da Sprint".

A mudança no time *Scrum*, vai além da inclusão de dois novos papeis. Tendo em vista que, as pessoas precisam ter uma visão que vai além das suas especialidades com o objetivo de ajudar de todas as maneiras possíveis a sua equipe. Elas são solicitadas a produzir algo tangível no final de cada *Sprint*. Assim, essas mudanças alterar papeis e relacionamentos dentro de equipe e empresas. (COHN, 2011).

2.2.3.3 Os eventos Scrum

Segundo Schwaber e Sutherland (2013), os eventos *Scrum*, são utilizados para criar uma rotina. De modo que, todos são eventos *time-boxed*[1], ou seja, possuem duração máxima. Assim, quando a *Sprint* começa, sua duração é fixa e não pode ser diminuída ou aumentada. Os outros eventos terminam quando o propósito do mesmo é alcançado, garantindo que seja utilizado um tempo adequado sem que haja perdas no processo.

"O coração do *Scrum* é a *Sprint*, um *time-boxed* de um mês ou menos, durante o qual um "Pronto", versão incremental potencialmente utilizável do produto, é criado. *Sprints* tem durações coerentes em todo o esforço de desenvolvimento". (SCHWABER; SUTHERLAND, 2013, p.8).

A *Sprint* consiste em uma unidade de trabalho solicitada para atingir um objetivo pré-estabelecido com um prazo determinado. (PRESSMAN; MAXIN, 2016). "Aprender como entregar *software* funcionando a cada *Sprint* é um dos maiores desafios que uma nova equipe *Scrum* deve superar. E fazer isso é crucial para se tornar ágil". (CONH, 2011, p.279).

Na reunião de planejamento da *Sprint* é criado um plano de trabalho pelo Time *Scrum*, o qual deve ser executado durante a *Sprint*. O *Scrum Master* garante que o evento ocorrerá e que todos os integrantes entendem o seu objetivo. Esta reunião busca responder o que pode ser entregue como resultado do incremento da próxima *Sprint* e como o trabalho necessário para entregar o incremento será realizado. (SCHWABER; SUTHERLAND, 2013).

As reuniões diárias são curtas, normalmente com duração de 15 minutos, realizadas diariamente pelo Time *Scrum*. Nesta reunião são respondidas três perguntas chaves pelos integras do time, sendo estão: "O que você realizou desde a última reunião da equipe? ", "Quais obstáculos está encontrando? ", "O que planeja realizar até a próxima reunião da equipe? ". (PRESSMAN; MAXIM, 2016). "A Reunião Diária aumenta a probabilidade de o Time de Desenvolvimento atingir o objetivo da *Sprint*". (SCHWABER; SUTHERLAND, 2013, p.11).

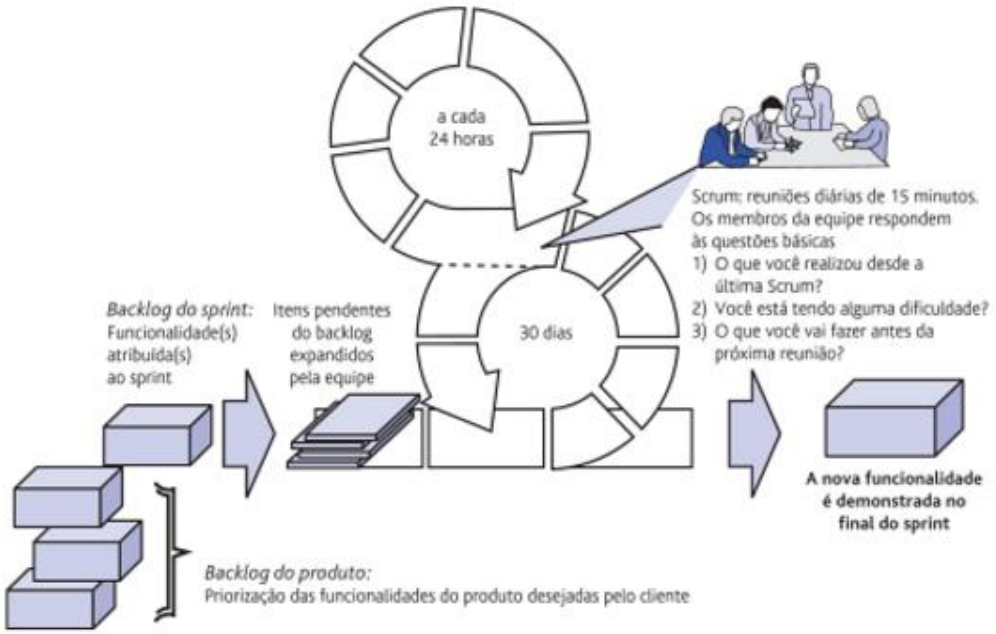
A revisão da *Sprint* é executada no final da mesma para verificar o incremento e adaptar o *backlog* do produto caso seja preciso. Nesta reunião o Time *Scrum* e as partes interessadas discutem sobre o trabalho realizado na *Sprint*. Com base nisso e em qualquer

[1] A tradução literal de "time-box" é "caixa de tempo" que deve ser entendido como um intervalo de tempo para realização de uma atividade.

mudança no *backlog* do Produto durante a *Sprint*, os participantes colaboram nas próximas coisas que podem ser feitas para aperfeiçoar valor. O objetivo é motivar e obter comentários e promover colaborações. (SCHWABER; SUTHERLAND, 2013).

Por fim, a retrospectiva da *Sprint* "[...] é uma reunião realizada no final de cada iteração para discutir o que foi bem-sucedido, o que poderia ser melhorado e como incorporar as melhorias e preservar os êxitos em iterações futuras. (BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD, 2014). De modo que, segundo Schwaber e Sutherland (2013, p.13), "a implementação destas melhorias na próxima Sprint é a forma de adaptação à inspeção que o Time Scrum faz a si própria". O fluxo geral do processo *Scrum* é ilustrado na figura 4.

Figura 4 - Fluxo de processo Scrum



Fonte: Pressman e Maxim (2016, p.78).

A figura acima demonstra o fluxo do processo *Scrum*, o qual inicia com o *backlog* do produto, contendo as funcionalidades priorizadas do projeto desejadas pelo cliente. Por sua vez, essas funcionalidades irão compor o *backlog* da *Sprint* e posteriormente os itens pendentes do *backlog* são expandidos pela equipe. Desse modo, a cada 24 horas ocorrem reuniões diárias de 15 minutos, onde os membros da equipe respondem questões básicas em relação ao andamento da *Sprint* e no final de 30 dias uma nova funcionalidade é demonstrada.

2.2.3.4 Definição de “pronto”

“Para verificar se há produto potencialmente liberável no final de cada *Sprint*, a equipe *Scrum* discute e define critérios adequados para a conclusão do *Sprint*. A discussão aprofunda a compreensão da equipe dos itens do *backlog* e os requisitos do produto”. (BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD, 2014, p.12).

Desse modo, todos do time devem entender o que é “Pronto”, mesmo que isto varie para cada Time *Scrum*, uma vez que, os integrantes devem ter um entendimento compartilhado do que significa o trabalho estar completo, com o objetivo de assegurar a transparência. (SCHWABER; SUTHERLAND, 2013).

2.3 TESTES EM ABORDAGEM ÁGIL

As equipes de desenvolvimento são capazes de adotar os valores ágeis, princípios e práticas. Contudo, pode demorar para a empresa compreender os benefícios de agilidade e assim demorarem mais tempo para se adaptarem a mudança. (CRISPIN; GREGORY, 2015, tradução nossa)

Desse modo, é necessário que os testadores estendam as diferenças entre os testes em modelos de ciclo de vida tradicionais e ciclo de vida ágil, com o objetivo de trabalhar de forma eficaz e eficiente. (BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD, 2014).

Em equipes tradicionais o foco está em garantir que todos os requisitos especificados estão sendo entregues no produto final, se tudo não estiver pronto para o lançamento, o mesmo é adiado. Os Testadores estudam documentos de requisitos para escreverem os planos de teste. (CRISPIN; GREGORY, 2009, tradução nossa).

Esse modelo, segundo Cohn (2011), fez com que muitos testadores tentassem gerar um documento de requisitos perfeito para verificarem se o sistema está funcionando corretamente. Contudo, mesmo que a conformidade com os requisitos seja interessante, a conformidade com as necessidades dos usuários é ainda melhor.

“Uma das principais diferenças entre os ciclos de vida tradicionais e os ciclos de vida Ágil é a ideia de iterações muito curtas, cada iteração resultando em um software que oferece funcionalidades de valor para as partes interessadas”. (BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD, 2014, p.19). Sendo assim, as atividades de teste ocorrem durante cada interação, e não como uma atividade no final. (BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD, 2014).

As equipes ágeis tornam o teste uma prática e uma parte central do processo de desenvolvimento, ao invés de algo que ocorre após o desenvolvimento ter acabado. Logo, a qualidade é incorporada ao processo e ao produto, à medida que ele é desenvolvido, em vez de testar a qualidade após o produto ser construído. (COHN, 2011).

Quando se trabalha dessa forma, segunda Cohn (2011), o programador e o testador conversam sobre quais requisitos serão adicionados ao produto e posteriormente o testador cria os testes automatizados e o programador programa. Assim, quando os dois terminam, os resultados são integrados.

Segundo a BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD (2014), os testadores podem servir como treinadores de teste e qualidade dentro dos times. Uma vez que, os mesmos compartilham o seu conhecimento, ajudando o trabalho de garantia de qualidade dentro da equipe. Desse modo, o senso de responsabilidade coletiva da qualidade do produto é compartilhado por todos. Logo, uma equipe ágil é aquela que realiza o seu trabalho da melhor forma e entrega o melhor produto possível. (CRISPIN; GREGORY, 2009, tradução nossa).

Testadores ágeis compreendem os aspectos técnicos e limitações relacionados à implementação e ainda conseguem ver cada história do ponto de vista do cliente. Assim, podem ajudar os desenvolvedores e os clientes a chegarem a uma linguagem comum. (CRISPIN; GREGORY, 2009, tradução nossa).

Por sua vez, o quadro 2 representa os cinco princípios fundamentais presentes no manifesto de testes que explicam por que testes ágeis é fundamentalmente diferente ao tradicional. (LAING; GREAVES, 2015).

Quadro 2 - O Manifesto dos Testes

O Manifesto dos Testes

1.	Testar por todas as etapas **versus** no final
2.	Prevenir bugs **versus** encontrar bugs
3.	Testar o entendimento **versus** checar funcionalidades
4.	Construir o melhor sistema **versus** quebrar o sistema
5.	Time responsável pela qualidade **versus** responsabilidade dos testadores

Fonte: Laing e Greaves (2015, p. 25, tradução nossa).

Tendo em vista, as diferenças apresentadas entre os testes em modelos tradicionais e no ágil, serão apresentadas a seguir, as habilidades necessárias para um testador ágil.

2.3.1 Habilidades e Funções do Testador Ágil

Segundo a Brazilian Software Testing Qualifications Board (2014), em equipes ágeis, o testador deve colaborar com todos os membros do time e as partes interessadas. Logo, há uma série de habilidades que um testador deve ter para desempenhar as atividades. Desse modo, o testador é um profissional que abraça a mudança, auxilia as pessoas técnicas e do negócio, e entende o conceito de usar testes para documentar requisitos e impulsionar o desenvolvimento. (CRISPIN; GREGORY, 2009, tradução nossa).

Testadores ágeis tendem a ter boas habilidades técnicas, sabem como colaborar com outros para automatizar testes, e também são experientes em testes exploratórios. Eles estão dispostos a aprender o que os clientes fazem para que eles possam entender melhor os requisitos de software. (CRISPIN; GREGORY, 2009, tradução nossa).

Segundo Crispin e Gregory (2015, tradução nossa), é difícil dominar habilidades como comunicação, colaboração, facilitação, solução de problemas e definição de prioridades. No entanto, essas são as habilidades mais importantes para o sucesso em testes ágeis.

A metodologia ágil depende da colaboração, comunicação e interação entre os integrantes do time. Desse modo, os testadores devem ter boas habilidades interpessoais. Entre estas podemos citar: ser positivo e orientar para solução de problemas, ativamente adquirir informações das partes interessadas, avaliar e relatar os resultados dos testes,

progresso e qualidade do produto, responder rapidamente a mudanças, planejar e organizar o seu trabalho e trabalhar para definir estórias de usuários testáveis, especialmente os critérios de aceitação, com representantes dos clientes e partes interessadas. (BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD, 2014).

"Crescimento contínuo de competências, incluindo o crescimento de habilidades interpessoais, é essencial para todos os testadores, incluindo aqueles das equipes do Ágil". (BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD, 2014, p.26). Assim, o testador ágil, está pronto para reunir e compartilhar informações, trabalhar com o cliente do produto para ajudá-lo a expressar seus requisitos de forma adequada para obter os recursos que necessitam, e fornecer *feedback* sobre o andamento do projeto a todos. (CRISPIN; GREGORY, 2009, tradução nossa).

"A função de um testador em uma equipe Ágil inclui atividades que geram e fornecem *feedback*, não só no status de teste, progresso de teste e qualidade do produto, mas também na qualidade do processo". (BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD, 2014, p.26).

Assim, as atividades do testador são: compreender, implementar e atualizar estratégias de teste, medir e informar a cobertura do teste em todas as dimensões de cobertura aplicáveis, compreender, implementar e atualizar a estratégia de teste, medir e informar a cobertura do teste em todas as dimensões de cobertura aplicáveis, garantir o uso adequado de ferramentas de teste, configurar, utilizar e gerenciar ambientes de teste e dados de teste, relatar defeitos e trabalhar com a equipe para resolvê-los, treinar outros membros da equipe em aspectos relevantes de testes, assegurar que as tarefas de teste adequadas sejam programadas durante lançamento e planejamento de iteração, colaborar ativamente com desenvolvedores e partes interessadas para esclarecer requisitos, especialmente em termos de testabilidade, consistência e completude e participar ativamente de retrospectivas da equipe, sugerindo e implementando melhorias. (BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD, 2014).

Por fim, para as autoras Crispin e Gregory (2009, p.20), "habilidades são importantes, mas a atitude conta mais". Assim, todos em uma equipe ágil devem ter a oportunidade para crescer de desenvolver suas habilidades. Equipes ágeis trabalham em um ritmo sustentável, que lhes permite seguir as práticas com disciplina. Como está escrito no Manifesto Ágil, nós valorizamos os indivíduos e interações sobre processos e ferramentas. (CRISPIN; GREGORY, 2009, tradução nossa).

2.3.2 Quadrantes de Teste e Níveis de Teste

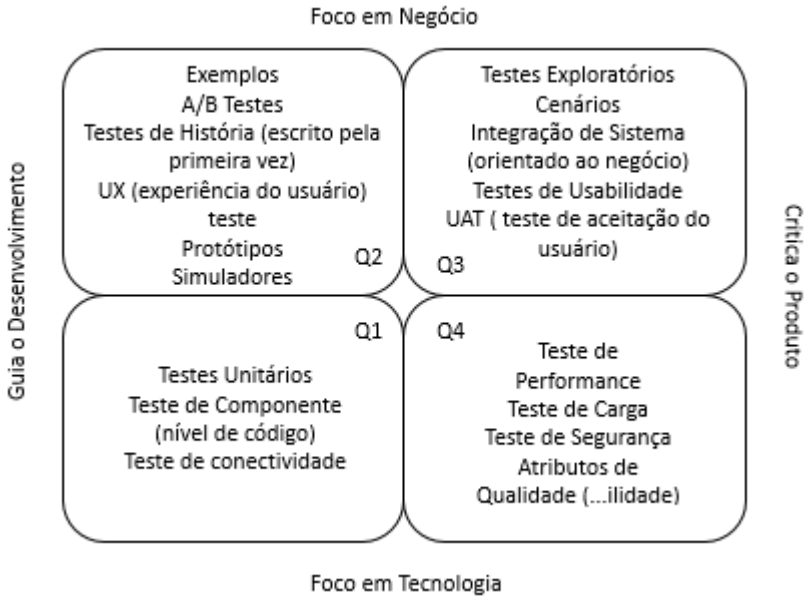
Os quadrantes de teste são baseados em uma matriz desenvolvida por Brian Marick em 2003 para descrever os tipos de teste utilizados em Extreme Programming. (CRISPIN; GREGORY, 2009, tradução nossa). A mesma alinha os níveis de teste com os tipos de testes apropriados na metodologia ágil. (BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD, 2014).

Segundo a Brazilian Software Testing Qualifications Board (2014, p. 29):

> O modelo de quadrantes de teste e suas variantes ajuda a assegurar que todos os tipos de testes importantes e os níveis de teste sejam incluídos no ciclo de vida de desenvolvimento. Este modelo também fornece uma maneira de diferenciar e descrever os tipos de testes a todas as partes interessadas, incluindo desenvolvedores, testadores e representantes de negócio.

Assim, Crispin e Gregory (2015, tradução nossa) explicam conforme a figura 5 que, o quadrante 1 contém os testes voltados para a tecnologia e orientam o desenvolvimento. No 2, temos os testes voltados para o negócio e também orientam o desenvolvimento. Por sua vez, o quadrante 3, possui os testes voltados para o negócio, os quais avaliam o produto e no quarto quadrante, os testes de tecnologia avaliam o produto.

Figura 5 - Quadrantes de teste ágil



Fonte: Crispin e Gregory (2015, p.148, tradução nossa).

O lado esquerdo da matriz quadrante é referente a prevenção de defeitos antes da implementação e o lado direito é sobre encontrar defeitos e descobrir características ausentes, mas com o entendimento de que queremos encontra-los o mais rápido possível. A metade

superior diz respeito aos testes que tem relação com o negócio e na metade inferior os testes são voltados para a equipe interna, mas igualmente importante para o produto de software. (CRISPIN, GREGORY, 2015, tradução nossa).

Desse modo, alguns testes apoiam o trabalho realizado pelo time ágil e confirmar o comportamento do *software*. Enquanto, outros verificam o produto. Os mesmos podem ser feitos manualmente, automaticamente ou uma combinação entre os dois, mas apoiado por ferramentas. (BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD, 2014).

Em relação aos níveis de teste, segundo a Brazilian Software Testing Qualifications Board (2014), os mesmos são atividades de teste que são logicamente relacionadas pela maturidade ou integridade do item em teste. Assim, durante uma iteração, qualquer estória de usuário geralmente progride sequencialmente através das seguintes atividades de teste: teste de unidade e teste de aceitação.

Os testes de unidade são normalmente realizados pelos desenvolvedores. Por sua vez, os testes de aceitação da funcionalidade são normalmente divididos em duas atividades. O teste de verificação de funcionalidade, muitas vezes é automatizado e podem ser feitos por desenvolvedores ou testadores e envolve testes contra os critérios de aceitação da estória de usuário. Já o teste de validação de funcionalidade, normalmente é manual e pode envolver desenvolvedores, testadores e partes interessadas que trabalham de forma colaborativa para determinar se a funcionalidade está apta para uso, para melhorar a visibilidade dos progressos realizados, e receber *feedback* real das partes interessadas. (BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD, 2014).

Segundo a Brazilian Software Testing Qualifications Board (2014, p. 21):

> Além disso, há muitas vezes um processo paralelo de testes de regressão que ocorre em toda a iteração. Trata-se de reexecutar os testes de unidade automatizados e testes de verificação da funcionalidade da iteração atual e iterações anteriores, geralmente através de uma estrutura de integração contínua.

Por fim, os times ágeis podem ter um nível de teste de sistema, uma vez que se inicia a primeira estória de usuário pronto para tais testes. Isso pode envolver a execução de testes funcionais, bem como testes não funcionais de desempenho, confiabilidade, usabilidade, e outros tipos de teste pertinentes. (BRAZILIAN SOFTWARE TESTING QUALIFICATIONS BOARD, 2014).

2.4 ESTRATÉGIAS DE TESTES

Segundo Pressman e Maxim (2016) muitas estratégias podem ser utilizadas para testar um *software*. No entanto, a escolhida por muitas equipes de *software* assume uma visão incremental de teste, iniciando com os testes das unidades individuais do sistema, passando para os testes que facilitar a integração das unidades e por fim os testes que usam o sistema concluído.

Por sua vez, podemos separar as técnicas de teste em caixa preta, branca e cinza. Segundo os autores Emerson e Moreira (2013), os testes de caixa preta buscam verificar as funcionalidades do sistema segundo uma ótima externa ou do usuário, sem qualquer conhecimento referente ao código ou lógica interna do *software*. Testes de caixa branca avaliam o código fonte, a lógica interna do componente codificado, as configurações e outros elementos técnicos.

Por fim, os testes de caixa cinza combinam as estratégias dos testes de caixa branca e preta, no qual analisa a parte lógica através do conhecimento interno do *software* e as saídas esperadas, comparando o que foi especificado com o que está sendo realizado. BEQUE (2000 apud LEWIS, 2009). Desse modo, os tipos de teste que fazem parte do processo de desenvolvimento de *software* da empresa do estudo de caso serão descritos a seguir.

2.4.1 Testes unitários

Os testes unitários são aplicados nas menores unidades do código com o objetivo de verificar o funcionamento de um pedaço do sistema de forma isolada, podendo assim, ser um programa ou componente. Por sua vez, este é o estágio mais baixo da escala de testes e na maioria das vezes, esses testes são realizados pelos desenvolvedores. (RIOS; MOREIRA, 2013).

Normalmente o teste de unidade é usado para auxiliar a etapa de implementação, visto que, os mesmos podem ocorrer antes do início da implementação ou após a geração do código fonte. Desse modo, um exame das informações de projeto fornece instruções para

estabelecer os casos de teste que devem ser acoplados a um conjunto de resultados esperados. (PRESSMAN; MAXIM, 2016).

2.4.2 Testes de integração

O termo teste de integração se refere aos testes que garantem que cada parte independente do sistema funciona corretamente com os serviços dos quais depende. Logo, os mesmos devem ser realizados em aplicações que se comunicam com sistemas externos ou em aplicações que consistem em uma série de módulos de baixo acoplamento e com interações complexas entre eles. (HUMBLE; FARLEY, 2014).

"O objetivo do teste de integração é validar as interfaces entre componentes da mesma arquitetura tecnológica. Essas interfaces representam as trocas dinâmicas de informação que ocorrem entre os componentes durante a execução de um software". (BARTIÉ, 2002, p.149).

Desse modo, essa estratégia de teste, é uma técnica sistemática para construir arquitetura de software, além de realizar teste para descobrir erros relacionados com as interfaces. (PRESSMAN; MAXIM, 2016).

2.4.3 Testes de sistema

Os testes de sistema visam à execução do sistema como um todo ou uma parte do sistema, dentro de um ambiente controlado para que seja possível validar a exatidão da execução de suas funções. Por sua vez, geralmente são realizados pela equipe de teste, a qual a operação normal do sistema deve ser simulada sendo testadas todas as suas funções de forma mais próxima ao ambiente de produção. (RIOS; MOREIRA, 2013).

Rios e Moreira (2013, p. 172) também explicam que "o teste *end-to-end* é um tipo de Teste de Sistema que visa colocar o sistema à prova de uma forma mais completa a partir da simulação de um ambiente real".

2.4.4 Testes de exploratórios

Segundo Bach (2003, tradução nossa), o teste exploratório é qualquer teste na medida em que o testador controla ativamente o design dos testes, como os mesmos são realizados e usa as informações obtidas durante o teste para projetas novos testes melhores.

Logo, o que torna os testes exploratórios eficazes e eficientes são as habilidades do testador de observar, ouvir, pensar e relatar os testes sem o subsídio de instruções detalhadas. (BACH, 2016, tradução nossa).

2.4.5 Testes de configuração

Os testes de configuração verificar se o *software* funciona em diferentes versões e configurações de ambientes, como por exemplo, diversos navegadores. (RIOS, MOREIRA, 2013).

2.4.6 Testes de usabilidade

Para Rios e Moreira (2013) os testes de usabilidade devem ser realizados principalmente em aplicações web, pois ocorre muita navegação entre páginas. Este tipo de teste busca verificar o nível de facilidade de uso do *software* pelo usuário através da avaliação das telas quanto ao seu conteúdo e clareza de linguagem.

2.4.7 Testes de regressão

Os testes de regressão são executados após a realização de uma melhoria funcional ou reparação em um *software*. O mesmo tem como objetivo determinar se a alteração regrediu outros aspectos do sistema, visto que, mudanças e correções são mais propensas a causar erros, do que o código do sistema original. (MYERS, 2004, tradução nossa).

Assim, um conjunto de dados e *scripts* deve ser mantido como *baseline* para serem executadas para verificar se s mudanças feitas no *software* não danificaram os códigos já validados anteriormente. Visto que, os resultados esperados do *baseline* devem ser comparados aos resultados após a mudança e se houver discrepância, a mesma deve ser resolvida. (RIOS, MOREIRA, 2013).

3 MÉTODO

Segundo Nola e Sankey (2014, tradução, nossa), método vem da palavra grega *methodos*, que tem suas raízes na ideia de ser esperto ou astuto. No entanto, mais tarde, passou a significar “a busca do conhecimento” ou “uma forma de investigação”. Para Cervo, Bervian e Silva (2007), de modo geral, o método é a ordem que se impõem aos diferentes processos para atingir um determinado objetivo. Por sua vez, nas ciências, o método é o conjunto de processos empregados na investigação e na demonstração da verdade.

Desse modo, para Marconi e Lakatos (2003), o uso dos métodos científicos não é exclusivo da competência da ciência, porém não há ciência sem o emprego dos métodos científicos. “O método é o conjunto das atividades sistemáticas e racionais que, com maior segurança e economia, permite alcançar o objetivo - conhecimentos válidos e verdadeiros -, traçando o caminho a ser seguido, detectando erros e auxiliando as decisões do cientista.” (MARCONI; LAKATOS, 2003, p.83).

De acordo com Cervo, Bervian e Silva (2007, p.28)

> O método não é um modelo, fórmula ou receita que, uma vez aplicada, colhe, sem margem de erro, os resultados previstos ou desejados. É apenas um conjunto ordenado de procedimentos que se mostrou eficiente, ao longo da história, na busca do saber. O método científico é, pois, um instrumento de trabalho. O resultado depende de seu usuário.

Assim, são apresentadas a seguir, a caracterização do tipo de pesquisa dessa monografia, as etapas metodológicas e a delimitação da mesma.

3.1 CARACTERIZAÇÃO DO TIPO DE PESQUISA

Para Barros e Lehfeld (2007), a pesquisa tornou-se uma atividade comum para todas as pessoas atuantes na sociedade, além dos cientistas. Logo, pesquisar é um fato natural e necessário a todos os indivíduos. Contudo, para ser qualificada como científica, a mesma deve ser “[...] uma atividade voltada para a investigação de problema teórica ou prática por meio do emprego de processos científicos. Ela parte, pois de uma dúvida ou problema e, com o uso do método científico, busca uma resposta ou solução”. (CERVO; BERVIAN; SILVA, 2007, p. 56).

Para Cervo, Bervian e Silva (2007), segunda a natureza da pesquisa existe uma distinção entre a pura ou básica e a aplica. Assim, na pesquisa pura, o pesquisador busca satisfazer uma necessidade intelectual por meio do conhecimento, ou seja, tem como meta o saber.

"Já na pesquisa aplicada, o investigador é motivado pela necessidade de contribuir para fins práticos mais ou menos imediatos, buscando soluções para problemas concretos". (CERVO; BERVIAN; SILVA, 2007, p.60). Por sua vez, a natureza aplicada foi utilizada na elaboração dessa monografia.

Em relação a abordagem, para Mascarenhas (2012), há dois jeitos de construir conhecimento, sendo estes, a pesquisa quantitativa e a pesquisa qualitativa. Desse modo, a pesquisa quantitativa baseia-se na coleta por quantificação para tratar os dados obtidos e utiliza técnicas estatísticas para evitar que o pesquisador influencie os resultados.

Já na pesquisa qualitativa, os dados são levantados e analisados ao mesmo tempo, os estudos são descritivos, voltados para a compressão do objetivo e o pesquisador deve influenciar na pesquisa. Visto que, o pesquisador deve apresentar uma pesquisa sólida e coerente. (MASCARENHAS, 2012). Neste sentido, a presente monografia apresenta uma abordagem qualitativa.

Cervo, Bervian e Silva (2007) afirmam que, segundo o critério de procedimento em geral que é utilizado, há três importantes tipos de pesquisa: a bibliográfica, a descritiva e a experimental.

A pesquisa bibliográfica busca explicar um problema com base em referências teóricas publicados em artigos, livros, dissertações e teses. Poder ser feita de forma independente ou como parte da pesquisa descritiva ou experimental. Por fim, busca conhecer e analisar as contribuições culturais ou cientificas do passado sobre determinado tema, assunto ou problema. (CERVO; BERVIAN; SILVA, 2007).

Segundo Cervo, Bervian e Silva (2007, p.60):

> A pesquisa descritiva observa, registra, analisa e correlaciona fatos ou fenômenos (variáveis) se, manipulá-los. Procura descobrir, com a maior precisão possível, a frequência com que um fenômeno ocorre, sua relação e conexão com outros, sua natureza e suas características.

A pesquisa descritiva pode assumir diversas formas, sendo umas delas, o estudo de caso. A mesmo, pesquisa aspectos variados da vida de determinados individuo, família, grupo ou comunidade que seja representativo do seu universo. (CERVO, BERVIAN; SILVA, 2007).

Por sua vez, a pesquisa experimental, de acordo com Barros e Lehfeld (2007), ocorre a manipulação de uma ou mais variáveis sob determinado controle com o objetivo de interpretar as reações e as modificações ocorridas no objeto da pesquisa. Sendo assim, o investigador interfere na realidade, fato ou situação estudada, através da manipulação direta das variáveis.

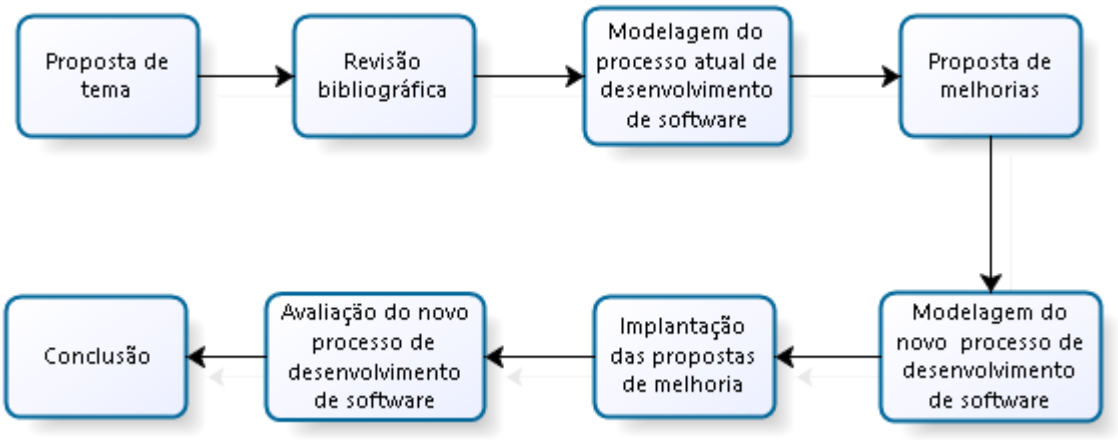
Desse modo, essa monografia é do tipo bibliográfico, de natureza aplicada e apresenta uma abordagem qualitativa. Tendo em vista os tipos de pesquisa, um estudo de caso será aplicado sobre o assunto teste em um projeto ágil utilizando a metodologia *Scrum* na empresa considerada.

Por fim, segundo Yin (2015), o estudo de caso é usado em muitas situações, para contribuir com o nosso conhecimento dos fenômenos individuas, grupais e organizacionais. O mesmo permite que os investigadores foquem em um “caso” e retenham uma perspectiva holística do mundo real, como nos estudos relacionais a processos organizacionais, por exemplo.

3.2 ETAPAS DA PESQUISA

O processo de desenvolvimento desse projeto foi subdividido nas seguintes etapas metodológicas ilustrados na figura 6.

Figura 6 - Etapas metodológicas do projeto



Fonte: Elaboração da autora, 2016.

- **Proposta de tema:**

Inicialmente, há a necessidade de elaboração de um tema, que se baseia no processo de desenvolvimento da empresa do estudo de caso, com o objetivo de propor melhorias para o processo e para a integração dos testes ao *Scrum*.

- **Revisão bibliográfica:**

A revisão bibliográfica é a base que sustenta essa monografia, possibilitando a análise e modelagem das etapas da área de qualidade de *software* para se integrarem aos eventos da metodologia ágil *Scrum*, objeto de estudo nesse projeto. Os seguintes assuntos foram abordados na revisão bibliográfica: modelos de processo de *software*, prescritivos e ágeis e qualidade de *software* no modelo tradicional e ágil.

- **Modelagem do processo atual de desenvolvimento de software:**

O modelo é uma simplificação da realidade. Desse modo, foram levantadas informações sobre o processo de desenvolvimento de *software* com o objetivo de compreender melhor a forma de organização das atividades na empresa em questão.

- **Proposta de melhorias:**

Foi realizada uma análise dos processos atuais de desenvolvimento de *software* com o objetivo de propor melhorias ao processo e para a integração dos testes ao *Scrum*.

- **Modelagem do novo processo de desenvolvimento de software:**

Esta etapa do projeto busca modelar o novo processo de desenvolvimento de *software*, com as atividades já integradas aos eventos do *Scrum*, o qual será montado com os subsídios fornecidos pelas etapas anteriores.

- **Implantação das melhorias propostas:**

O objetivo dessa etapa é descrever a implantação das melhorias propostas para o processo de desenvolvimento de *software* da empresa.

- **Avaliação do novo processo de desenvolvimento de software:**

O novo processo de desenvolvimento de *software*, contendo as melhorias sugeridas, foi avaliado pelos colaboradores da empresa do estudo de caso.

- **Conclusão:**

Por fim, será realizada a conclusão dessa monografia através dos conhecimentos advindos da revisão da bibliografia e da aplicação do estudo de caso. Feito isso, será identificado se os objetivos propostos foram alcançados.

3.3 DELIMITAÇÕES

Dentro das delimitações desse projeto, não foram especificadas as várias metodologias ágeis, e sim, somente o *Scrum*, pois o trabalho se propõe a apresentar uma proposta para melhorar a integração dos testes ao processo de desenvolvimento de *software* de uma empresa que utiliza o *framework Scrum*.

Os processos atuais da empresa em questão foram ajustados com base em modelos e técnicas existentes e não foram mostradas novas técnicas ou modelos no projeto.

Por sua vez, o BPMN (Notação de Modelagem de Processos de Negócio), foi utilizado somente para representar os processos da empresa. Logo não é o foco desse trabalho, descrever profundamente a sua notação.

Além disso, não foi realizada uma análise aprofundada das ferramentas propostas na seção 4.4.1 dessa monografia.

Por fim, não foi possível acompanhar a implantação das melhorias propostas em todos os projetos da empresa PP LTDA, somente a realizada em um projeto específico, o Rastreador.

4 MODELAGEM DO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

O presente capítulo contém a descrição do cenário de aplicação e a apresentação do processo de desenvolvimento de *software* da empresa do estudo de caso, o qual foi dividido em duas partes: ciclo de vida de gestão de funcionalidades e ciclo de vida de gestão de bugs de produção. Por sua vez o ciclo de vida de gestão de funcionalidades conta com sete etapas: verificação de solicitação, planejamento, implementação, teste de funcionalidade, correção de *bug* de desenvolvimento, teste de *bug* de desenvolvimento e publicação de funcionalidade e o ciclo de vida de gestão de bugs possui quatro etapas: verificação de *bug*, correção, teste de *bug* e publicação de *bug*.

Além disso, é mostrado como a empresa PP LTDA utiliza o *Scrum* durante o processo de desenvolvimento, as propostas de melhorias para o processo e a modelagem do novo processo, e pôr fim, a implantação e a avaliação da proposta.

4.1 CENÁRIO DE APLICAÇÃO

A empresa do estudo de caso, PP LTDA, é uma organização voltada ao desenvolvimento de aplicativos especializados (*softwares*) que atendem às necessidades da cadeia produtiva de alimentos e bebidas do setor do agronegócio. Fundada em 2005 na cidade de Florianópolis, SC, a empresa atualmente conta com 70 funcionários e 2 sócios. A mesma caminha lado a lado com seus clientes, desenvolvendo ferramentas que favoreçam resultados mensuráveis e impacto econômico positivo nos indicadores de seus clientes e da sociedade

Em relação ao seu portfólio principal de produtos, a empresa oferece as seguintes soluções:

a) **Caderno de Campo:** é um aplicativo para dispositivos móveis, o qual auxilia a organização da informação no nível de produção, realizando a gestão integrada do campo. Por sua vez, o mesmo também possui versão *web*.
b) **Rastreador:** é um *software* que roda em ambiente *web*, o qual permite conectar as diversas empresas ao longo da cadeia produtiva e também os diversos sistemas de

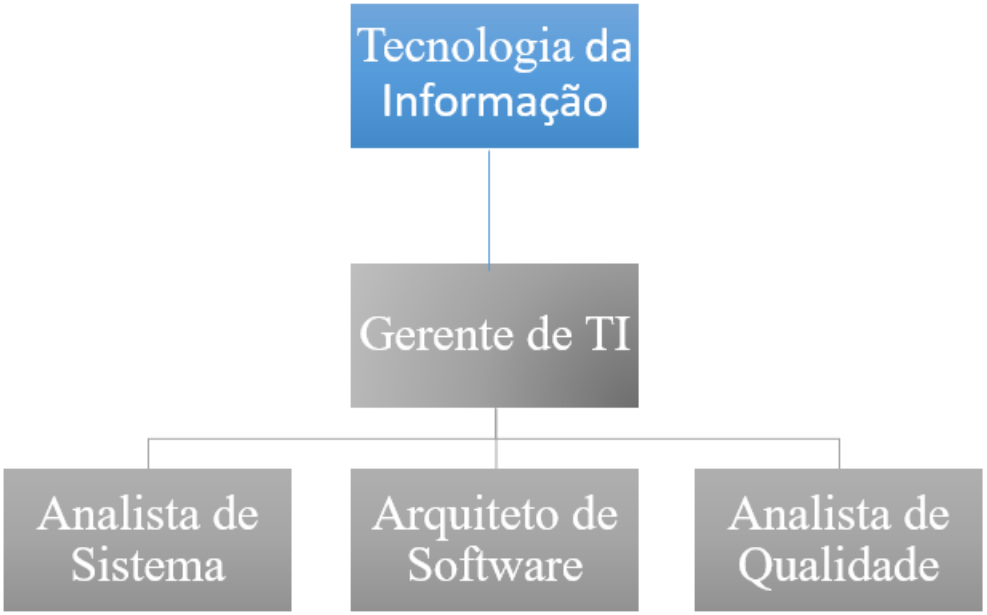
qualidade e controle interno de cada participante. O mesmo também possui uma versão *desktop*.

c) **Clicq:** é um aplicativo para dispositivos móveis que busca controlar as atividades associadas à inspeção de qualidade de produtos e processos, na produção, distribuição e ponto de venda. O mesmo também possui versão *web*.

d) **Panorama:** é um *software*, o qual roda em ambiente web, de gestão de informação (*Business Inteligence*) dos dados do ecossistema PariPassu (produção, rastreabilidade, controle de qualidade, monitoramento de análise de resíduos e avaliação feitas pelo consumidor final).

A PP LTDA está fundamentada em quatro departamentos, sendo estes: administrativo, comercial (pré-venda), operação (pós-venda) e tecnologia da informação. Visto que, no estudo de caso, será considerado apenas o ultimo departamento citado acima, composto por dezesseis analistas de sistema, cinco analistas de qualidade, um arquiteto de *software* e um gerente de TI, como mostra a figura 7.

Figura 7 - Organograma departamento de TI



Fonte: Elaboração da autora, 2016.

A empresa do estudo de caso passou por mudanças em relação ao processo de desenvolvimento de *software*, pois precisava ganhar agilidade nas entregas de novas funcionalidades. Desse modo, escolheu o *framework Scrum* como base para organizar as suas atividades. Por sua vez, o time de desenvolvimento de *software* foi dividido entre os quatro principais projetos da empresa e um analista de sistema de cada time foi nomeado como líder

do produto, exceto no Caderno de Campo. Desse modo, cada projeto ficou em média com 5 integrantes. Nesse momento, a empresa não seguia todos os princípios do *Scrum* e alguns eram aplicados incorretamente por falta de conhecimento dos membros da equipe. Logo, as lições eram aprendidas conforme ia se fazendo.

Além disso, a área de qualidade ficou desorientada devido as mudanças ocorridas, pois o novo modelo não fornecia orientação de como realizar os testes do *software*. O número de *bugs*[2] aumentava e as reclamações dos clientes eram frequentes. Assim, a preocupação em não perder qualidade fez com que a acadêmica, autora desse trabalho, a qual faz parte da equipe de qualidade da empresa PP LTDA, resolvesse realizar uma proposta de melhorias para o processo de desenvolvimento de *software* com foco na qualidade do produto.

4.2 PROCESSO ATUAL DE DESENVOVIMENTO DE SOFTWARE DA EMPRESA DO ESTUDO DE CASO

A empresa do estudo de caso, atualmente não possui o processo de desenvolvimento de *software* modelado e documentado. Não havendo um padrão definido, cada equipe de projeto segue um modelo que acredita ser correto, conforme seus conhecimentos e experiências. Desse modo, novos integrantes da equipe têm dificuldade de compreender e se adequar ao processo. Além disso, a gerência não consegue acompanhar corretamente todas as equipes e medir sua produtividade.

Sendo assim, para que seja possível melhorar o processo e, por conseguinte a qualidade do produto é necessário, inicialmente, modelar e documentar o processo atual da empresa do estudo de caso. A modelagem do processo de desenvolvimento de *software* será dividida em ciclo de vida de gestão de funcionalidade e ciclo de vida de gestão de *bugs* de produção, os quais serão apresentados a seguir. Além disso, também são mostradas as ferramentas que apoiam os testes dentro do processo atual da empresa.

[2] Bug pode ser definido como um erro comum no funcionando de um software.

4.2.1 Ferramentas de apoio aos testes do processo atual da empresa

Nessa seção, são apresentadas as ferramentas utilizadas pela empresa do estudo de caso, PP LTDA, para apoiar os testes no processo de desenvolvimento de *software*.

4.2.1.1 Amazon Web Services (AWS)

A AWS oferece um amplo conjunto de soluções com base na nuvem para auxiliar a disponibilização das aplicações em produção ou desenvolvimento com maior rapidez, flexibilidade, escalabilidade e confiabilidade. (AMAZON WEB SERVICES, 2016).

4.2.1.2 Bitbucket

Segundo a Atlassian (2016), o Bitbucket é um sistema de controle de versão distribuída que tem como objetivo gerenciar diferentes versões de um mesmo projeto. É possível obter históricos das modificações, permite que vários desenvolvedores trabalhem no mesmo projeto e também um comparativo entre as várias versões do projeto. O mesmo conta com um serviço *open source* e um comercial.

4.2.1.3 Confluence

Segundo a Atlassian (2016), o Confluence é uma ferramenta comercial que tem como objetivo a geração de conteúdo para o compartilhamento do conhecimento entre toda a

equipe. O mesmo pode ser utilizado para criar notas de reuniões, compartilhar arquivos, definir requisitos e tomar decisão.

4.2.1.4 HipChat

Segundo a Atlassian (2016), o HipChat é uma ferramenta comercial ideal para troca de mensagens no âmbito empresarial. É possível conversar com as pessoas em mensagens privadas ou criar grupos.

4.2.1.5 Jenkins

O Jenkins é uma ferramenta *open source*, multiplaforma para a integração continua. Nele é possível criar Jobs para diversos projetos e integrar as mesmas. (JENKINS, 2016).

4.2.1.6 Jira

Segundo a Atlassian (2016), o Jira é uma ferramenta comercial para gestão de projetos e tarefas, bem como para acompanhamento e reporte de bugs utilizada por equipes ágeis. A mesma cobre todas as etapas do ciclo de vida de desenvolvimento e manutenção de sistemas e possui integração com sistemas de gerência de versão como o Bitbucket e de gestão do conhecimento, como o Confluence.

4.2.2 Ciclo de vida de gestão de funcionalidade

São realizadas solicitações por e-mail de novas funcionalidades ou ajustes das que já existem por parte dos colaboradores da PP LTDA. Visto que, as mesmas podem ou não serem solicitações de clientes. Essas solicitações são cadastradas automaticamente no Jira.

Desse modo, as solicitações são verificadas pelo diretor comercial, representado por um dos sócios da empresa e o gerente de TI, em reuniões que ocorrem um dia antes do início da *sprint* de cada projeto. Por sua vez, o diretor comercial exerce o papel de *product owner* no *Scrum,* como foi abordado na seção 2.2.3.2 dessa monografia. O objetivo é analisar e decidir quais solicitações serão atendidas. Como resultado, se tem uma lista de *stories*[3] que irão compor o *backlog* do produto. Visto que, as *stories* que foram priorizadas e são candidatas a entrar na *sprint* permanecem em uma *sprint* fixa chamada de prioridades.

Após essa etapa, nomeada como verificação de solicitações, o gerente de TI inicia o planejamento e realiza os eventos de refino do backlog e *planning*, descrevendo melhor as funcionalidades que estavam no *backlog* de produtos, agrupa as mesmas em *epics*[4] e atribui os pontos de acordo com suas experiências anteriores no que diz respeito ao esforço de implementação. Essas atividades irão integrar o *backlog* da s*print*. Uma vez que, o tempo de duração da *sprint* de todos os projetos da empresa do estudo é de duas semanas.

Assim é iniciada a etapa de implementação, a qual diz respeito à implementação da nova funcionalidade ou ajuste da que já existe pelo analista de sistema, de acordo com o seu projeto. Após ter implementado a funcionalidade de acordo com a descrição do caso e testado de acordo com os seus critérios, o responsável pela atividade informa ao analista de qualidade do projeto que os testes já podem ser realizados.

Logo, o analista de qualidade inicia a etapa de teste de funcionalidade, a qual tem como objetivo validar se a funcionalidade está de acordo com o solicitado. Se todos os testes não forem concluídos com sucesso, o analista de qualidade relata os *bugs* de desenvolvimento encontrado no Jira para que o analista de sistema responsável pela implementação possa realizar a atividade na etapa de correção de *bug* de desenvolvimento.

---

[3] Stories são as menores unidades de trabalho. O objetivo de uma história de usuário é entregar um determinado valor de volta para o cliente.

[4] Epics é o trabalho em nível de recurso que engloba uma ou mais stories.

Após a correção, a etapa de teste de *bug* de desenvolvimento é realizada. Caso for verificado que o *bug* não foi corrigido, o mesmo é retornado para correção. No entanto, se o *bug* foi corrigido, a etapa de teste de funcionalidade é realizada novamente. Assim, se a funcionalidade está de acordo com o solicitado e não possui *bugs* de desenvolvimento, a mesma será aprovada.

O analista de qualidade é responsável por comunicar ao líder do projeto pelo mensageiro HipChat as funcionalidades aprovadas, as quais podem ser disponibilizadas no ambiente de produção[5]. Assim na etapa de publicação de funcionalidades é concretizada. Visto que, essa etapa é realizada de forma contínua, após o término dos testes de cada *epic*. Além disso, a publicação pode ser realizada em qualquer dia da semana até às quatro horas da tarde, exceto na sexta-feira. Se for necessário fugir dessa regra, é preciso pedir autorização ao gerente de TI.

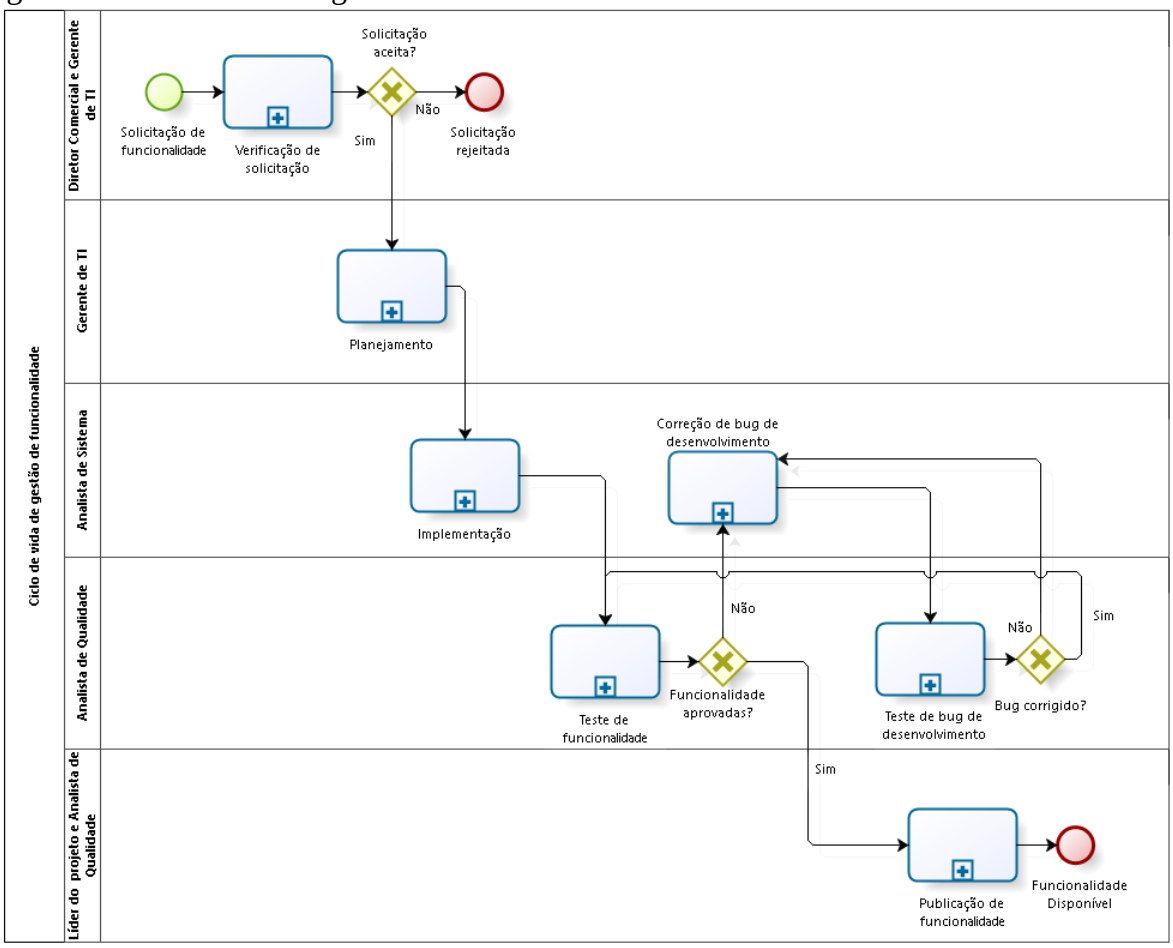
Dessa forma, a comunicação de mudanças é feita através de um e-mail enviado pelo analista de qualidade a todos os colaboradores da empresa informando que a nova versão está disponível para uso dos clientes.

Além disso, o time de cada projeto realiza reuniões diárias de quinze minutos. Cada membro da equipe discursa brevemente sobre a atividade que está fazendo, focando em responder se a atividade está no prazo ou se está atrasada e qual será a próxima atividade a ser feita.

A figura 8 representa de forma geral o ciclo de vida de gestão de funcionalidades da empresa PP LTDA e os papeis envolvidos em cada atividade. Logo, serão detalhados a seguir as etapas de verificação de solicitações, planejamento, implementação, teste de funcionalidade, correção de *bug* de desenvolvimento, teste de *bug* de desenvolvimento e publicação de funcionalidade.

[5] O ambiente de produção é onde os usuários finais acessarão o software.

Figura 8 - Ciclo de vida de gestão de funcionalidade



Fonte: Elaboração da autora, 2016.

Desse modo, a primeira etapa do processo de desenvolvimento de *software*, verificação de solicitações, realizada pelo diretor comercial e o gerente de TI da empresa PP LTDA será descrita com detalhes na próxima seção. A mesma tem como critério de entrada uma solicitação de funcionalidade e os critérios de saída são a aprovação ou rejeição da solicitação.

4.2.2.1 Etapa verificação de solicitação

As solicitações enviadas via e-mail pelos colaboradores da PP LTDA devem seguir o padrão apresentado no quadro 3. As mesmas são cadastradas automaticamente no projeto triagem da ferramenta Jira com status “Novo” e tipo *task*. Contudo, se as solicitações forem enviadas pelos colaboradores do setor de TI, as mesmas devem ser cadastradas diretamente no Jira, como tipo *story* e no projeto correspondente.

Quadro 3 - Padrão de solicitação de funcionalidade

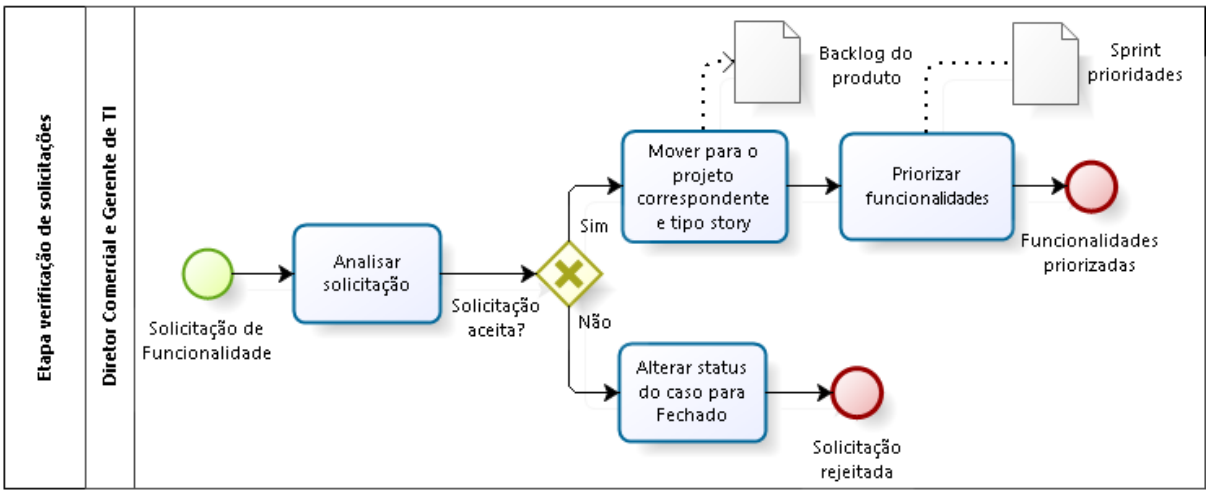
Remetente	E-mail exclusivo para solicitação de funcionalidades
Assunto	[Sistema] – Descrição breve da necessidade (Nome da empresa)
Corpo do e-mail	a. Descrição detalhada da necessidade de uma nova funcionalidade ou ajuste de uma que já existe; b. Motivo pelo qual é necessária essa funcionalidade; c. Nome da empresa; d. Usuário (e-mail); e. Anexo de imagens; f. Documentos que auxiliem na identificação da necessidade.

Fonte: Elaboração da autora, 2016.

Posteriormente, o diretor Comercial e o gerente de TI decidem quais solicitações irão compor o *backlog* de cada produto, visto que as mesmas são movidas para os seus devidos projetos, com o tipo s*tory*, e a lista de funcionalidades é priorizada. Dessa forma, as *stories* que foram priorizadas e são candidatas a entrar na *sprint* permanecem em uma *sprint* fixa chamada de prioridades.

A figura 9 representa a etapa de verificação de solicitação da empresa PP LTDA e os responsáveis pelas tarefas.

Figura 9 - Etapa verificação de solicitação



Fonte: Elaboração da autora, 2016.

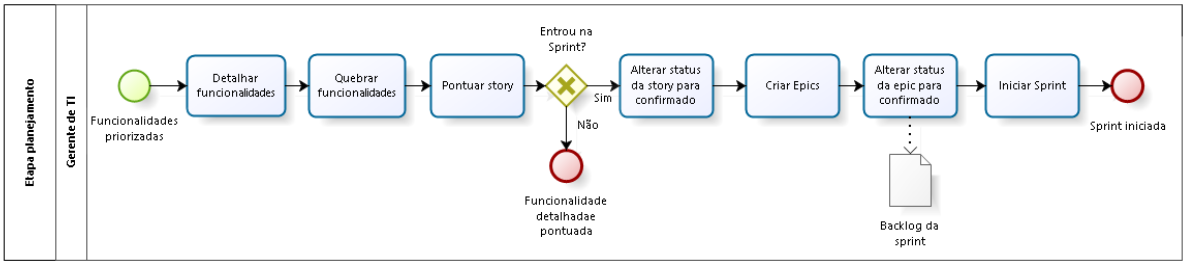
Na próxima seção dessa monografia, é detalhada a etapa de planejamento, a qual é realizada pelo gerente de TI. Por sua vez, o critério de entrada é a funcionalidade priorizada e os de saída o início da *sprint*.

4.2.2.2 Etapa planejamento

Por sua vez, o gerente de TI descreve com maiores detalhes as funcionalidades que irão compor o *backlog* da s*print*, a qual tem duração de duas semanas. As *stories* são quebradas entre as camadas *backend* e *frontend*, se necessário, e pontuadas. Se a funcionalidade entrar na *sprint*, a mesma é confirmada. As *epics* com o nome das atividades e a data prevista de término são criadas, confirmadas e atribuídas para cada *story* que compõem a *sprint*. Por fim, a s*print* é iniciada.

A figura 10 representa a etapa de planejamento da empresa PP LTDA e o responsável pela atividade.

Figura 10 - Etapa planejamento



Fonte: Elaboração da autora, 2016.

Após essa etapa, é possível iniciar a implementação da funcionalidade, a qual seja descrita na próxima seção. A mesma tem como critério de entrada, o início da *sprint*, e o critério de saída à funcionalidade implementada.

4.2.2.3 Etapa implementação

Com a s*print* iniciada, os analistas de sistemas de cada projeto devem iniciar a implementação de acordo com a ordem estipulada no *backlog* s*print*. Desse modo, o analista atribui a atividade para si e troca o status da *story* no Jira para “Em implementação” e o status da *epic* para “Em execução”.

Por sua vez, a *branch* é criada a partir da *master*, com o formato padrão*:* dev-epic, onde serão feitas as implementações. Assim, o *branch* correspondente à funcionalidade leva o

nome da *epic* e é obrigatória a inclusão do id do caso no texto dos *commits* para o Bitbucket. A funcionalidade é implementada conforme as definições pré-estabelecidas, assim como os testes unitários citados na seção 2.4.1 dessa monografia.

Feito isso, a funcionalidade é testada pelo analista de sistema em seu ambiente local. Caso os testes não passarem, o analista de sistema corrige o problema e testa novamente. No entanto, se os testes passarem, é necessário executar os testes unitários existentes no projeto. Se os mesmos falharem devem ser ajustados até que passem.

A *branch* de desenvolvimento deve ser atualizada com a *master*, uma *tag* deve ser criada, para realizar o *deploy* via Jenkins, seguindo o mesmo padrão da *branch,* e o *pull request* deve ser feito, com o objetivo de lançar a nova versão da aplicação com as implementações.

Sendo assim, o analista de sistema encerra a implementação e preencher o formulário apresentado no quadro 4 na ferramenta Jira e caso a funcionalidade corresponda a um *crud* [6], o formulário representado no quadro 5 também deve ser preenchido.

Quadro 4 - Formulário geral de encerramento de implementação

Componente do sistema que foi afetado com a nota funcionalidade. Exemplo: *backend*, *frontend*, aplicativo, banco de dados, etc.
Pré teste geral de implementação: [Caminho feliz] Todas as funcionalidades descritas na issue foram implementadas? [Testes unitários] Os testes unitários do projeto estão funcionando? [Visibilidade dos dados] Os dados das listagens e dos combos aparecem apenas para a empresa corrente?
Serviços adicionados ou modificados e um exemplo de requisição.
SQL necessário para a funcionalidade.
Configurações adicionadas no *json* ou variáveis do Java.
Configurações que foram removidas do projeto.

Fonte: Elaboração da autora, 2016.

Ambos os formulários foram elaborados para ajudar os analistas de sistema da empresa PP LTDA a realizar as validações das funcionalidades implementadas e para fornecer os subsídios necessários ao analista de qualidade configurar sozinho o ambiente de teste.

[6] Refere-se as quatro operações básicas de uma interface: inserção, edição, listagem e exclusão.

Quadro 5 - Formulário de pré-teste de crud

[Inserção] Na inserção de um registro o usuário pode preencher os campos e salvar sem ocorrer erro (caminho feliz).
[Edição] Na edição de um registro o usuário pode preencher os campos e salvar sem ocorrer erro (caminho feliz).
[Exclusão de itens] Os registros que não têm nenhuma relação de dependência com outra unidade informacional são excluídos corretamente (ex.: um registro de envio pode ser excluído) e os registros que têm relação de dependência com outra unidade.
[Preenchimento obrigatório de campos] Na inserção ou edição de um registro, o sistema não permite que o usuário salve os dados sem preencher todos os campos obrigatórios e todos os campos obrigatórios têm uma marcação diferente (asterisco).
[Preenchimento inválido de campos] Os campos das telas aceitam apenas caracteres válidos para seu propósito (ex.: os campos de data aceitam apenas caracteres numéricos; campos de quantidade aceitam apenas valores numéricos inteiros, etc.)
[Duplicidade de dados] Na inserção ou edição de um registro, o sistema não permite que o usuário salve os dados preenchendo a identificação principal desse registro com o mesmo dado de um outro registro existente (ex.: nome do elo; nome do usuário).
[Alinhamento de campos] Os campos das telas aparecem corretamente alinhados tanto no Firefox, no IE e no Chrome, incluindo modais e campos de data.
[Ordenação de itens] Os itens listados nos combos são ordenados alfabeticamente.

Fonte: Elaboração da autora, 2016.

Além disso, antes de enviar efetivamente a funcionalidade para teste, a analista de sistema precisa preencher o ultimo formulário, apresentado no quadro 6, para que a etapa seja concluída.

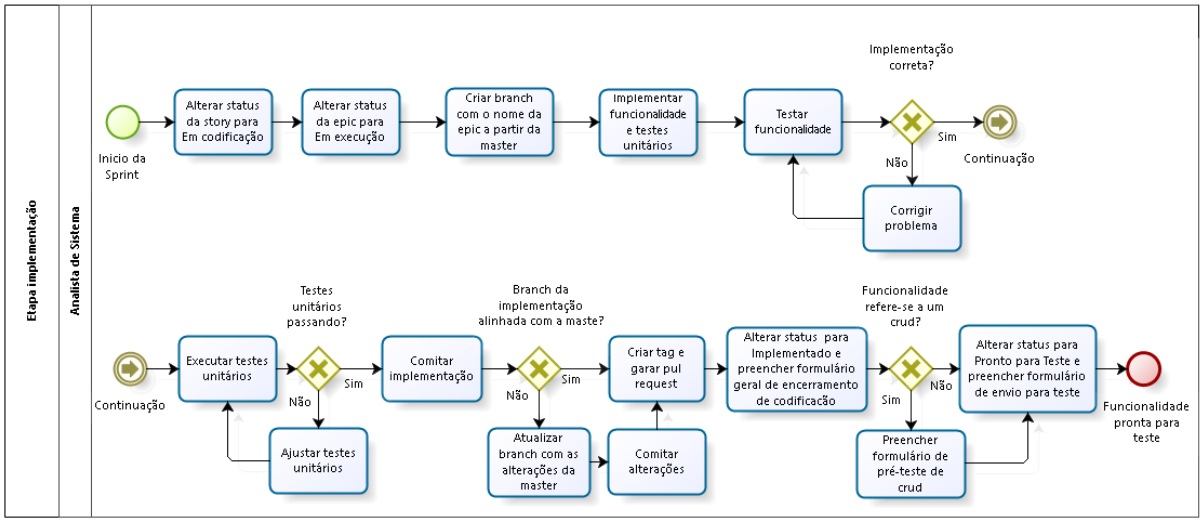
Quadro 6 - Formulário de envio para teste

Pré-Teste [Código fonte] O código fonte está atualizado com a Master? [Tag de implementação] A mesma tag de implementação foi gerada no backend e no frontend? [Dependências nexus] Todas as dependências estão no Nexus?
Tag de implementação

Fonte: Elaboração da autora, 2016.

Por fim, a figura 11 representa a etapa de implementação da empresa PP LTDA e o responsável envolvido em nas tarefas.

Figura 11 - Etapa implementação



Fonte: Elaboração da autora, 2016.

Após finalizar a terceira etapa do processo de desenvolvimento de *software* da empresa do estudo de caso, é iniciada a etapa de testes de funcionalidades, a qual é realizada pelo analista de qualidade. Seu critério de entrada são funcionalidades prontas para teste e o de saída são as funcionalidades testadas.

4.2.2.4 Etapa testes de funcionalidade

A etapa de testes de funcionalidade é iniciada pelo analista de qualidade quando as funcionalidades implementadas estão com status “Pronto para Teste” na ferramenta Jira. Assim a tarefa é atribuída para si quando o status é alterado para “Em Teste”.

No que diz respeito à infra e o *deploy* dos projetos, sejam estes na camada *backend* ou *frontend*, a PP LTDA conta com as ferramentas da Amazon Web Services (AWS) e o Jenkins para auxiliar na automação desses processos. Assim, quando é necessário subir um novo ambiente ou apenas trocar a versão que está sendo executada, basta abrir a aba INFRA (sistema não provisionado) ou Qualidade (sistema provisionado), clicar na j*ob* do sistema, acessar *Build with Parameters*, utilizar a *tag* gerada na etapa anterior, passar os outros parâmetros requeridos e clicar no botão *Build*.

Além disso, a empresa possui um banco de desenvolvimento, o qual foi triplicado, de modo que, um é usado pelos analistas de sistema durante o desenvolvimento, o outro pelos

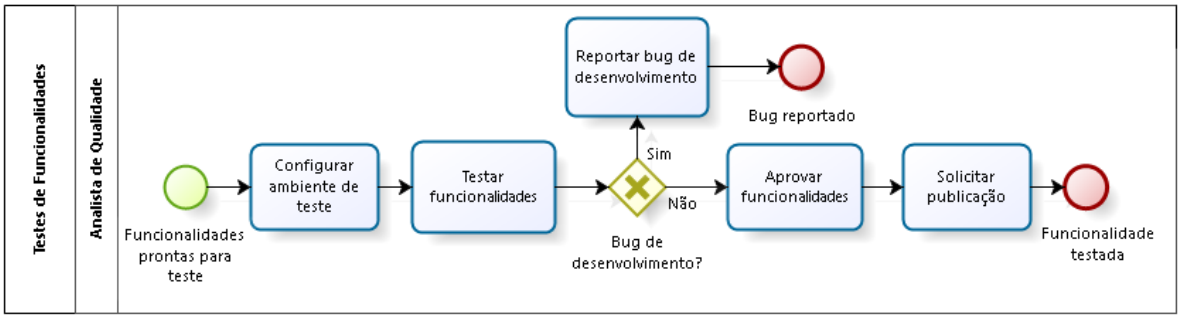
analistas de qualidade durante o teste e o último é usado somente para executar a integração continua (CI). Por sua vez, o banco é restaurado diariamente no período noturno, pois o mesmo precisa manter-se atualizado para que o ambiente seja o mais próximo do ambiente de produção.

As configurações adicionadas ou removidas também são ajustadas nos arquivos *json* correspondentes. Terminado a configuração do ambiente para teste, o analista de qualidade realiza os testes de sistema, citados anteriormente na seção 2.4.3 para verificar se a funcionalidade foi implementada de acordo com o que foi solicitado. Além disso, também são realizados os testes exploratórios, cujo seu conceito pode ser encontrado na seção 2.4.4 dessa monografia.

Caso todos os testes não forem concluídos com sucesso, são relatados os *bugs* de desenvolvimento encontrados, para que o analista de sistema possa corrigi-los. Para criar um *bug* no Jira é necessário selecionar o projeto, a categoria *Bug*, descrever o problema identificado, informar os passos para reproduzir e o resultado esperado. Além disso, para confirmar o problema é necessário adicionar a *label* desenvolvimento, informar o analista de sistema que causou o *bug*, a s*print* que o *bug* pertence relacionar a *issue* do *bug* com a da *story* que o gerou e adicionar arquivos que auxiliem a correção do *bug*.

No entanto, se após os testes de sistema e exploratórios da funcionalidade for verificado que a implementação está de acordo com o solicitado e não possui *bugs* de desenvolvimento, a mesma será aprovada. Para isto, o analista de qualidade alterar o status “Testado” e solicita a publicação em ambiente de produção. Por fim, a figura 12, ilustra a etapa de testes de funcionalidades e o papel envolvido nas tarefas.

Figura 12 - Etapa testes de funcionalidade



Fonte: Elaboração da autora, 2016.

Na próxima seção é apresentada a etapa de correção de *bug* de desenvolvimento, a qual é executada pelo analista de sistema. A mesma tem como critério de entrada o reporte de um *bug* de desenvolvimento e como critério de saída o *bug* corrigido.

4.2.2.5 Etapa correção de bug de desenvolvimento

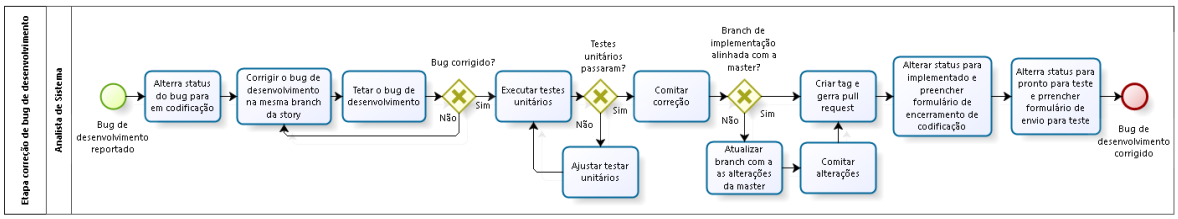
A etapa de correção de bug de desenvolvimento é iniciada quando o analista de qualidade reporta um *bug* de desenvolvimento encontrado durante o teste de funcionalidade.

Por sua vez, o analista de sistema altera o status para “Em codificação” no Jira e corrige o *bug* na mesma *branch* de implementação da *story*. Por conseguinte, ele verifica se a correção está correta. Se o *bug* não for corrigido, o mesmo realiza o ajuste na correção e testa novamente. Desse modo, se os testes passarem, é necessário executar os testes unitários do projeto. Visto que, se os testes não passarem é feito o ajuste e se passarem o analista de sistema comita o código fonte da correção.

Logo, se a *branch* de implementação não estiver atualizada com a *master*, é necessário atualizá-la para posteriormente criar a *tag* e realizar o *pull request*. Por fim, o status do caso é alterado para implementado e o formulário de encerramento de codificação é preenchido e, por conseguinte o status é alterado para pronto para teste e o formulário de envio para teste é preenchido.

A figura 13 representa a etapa de correção de *bug* de desenvolvimento e o responsável envolvido nas tarefas.

Figura 13 - Etapa correção de bug de desenvolvimento



Fonte: Elaboração da autora, 2016.

Na próxima seção é apresentada a etapa de teste de bug de desenvolvimento, a qual é executada pelo analista de qualidade. A mesma tem como critério de entrada um *bug* de desenvolvimento corrigido e como saída, o *bug* testado ou o bug retornado para correção.

4.2.2.6 Etapa teste de bug de desenvolvimento

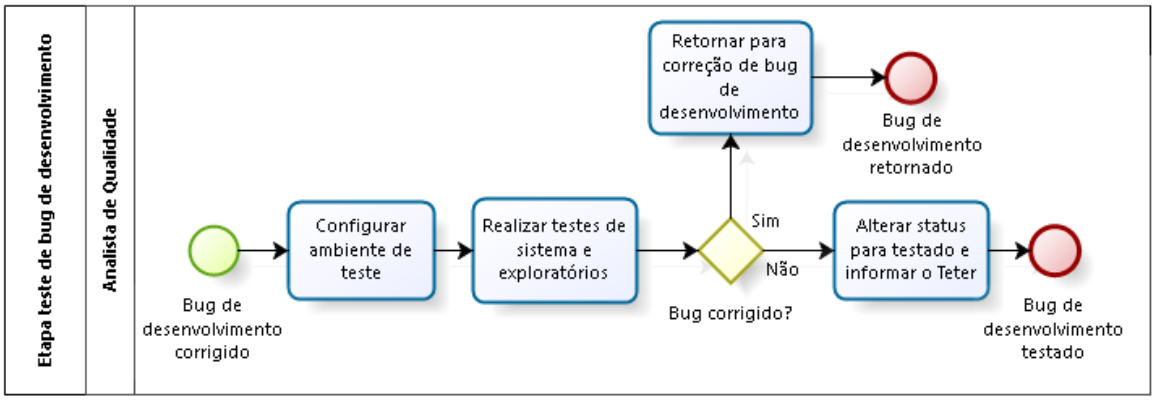
A etapa de teste de *bug* de desenvolvimento inicia quando o analista de sistema corrige um bug encontrado em desenvolvimento na etapa de teste de implementação.

Dessa forma, o analista de qualidade altera o status para em teste, atribuindo a tarefa para si. O mesmo configura o ambiente de teste executando *sql*, alterando configurações *json*, entre outros e com o auxílio do Jenkins e da Amazom realiza o provisionamento e o *deploy* do sistema.

Posteriormente são realizados testes de sistema e os testes exploratórios. Se todos os testes passarem o status é alterado para testado. Contudo, se os testes referentes ao *bug* de desenvolvimento não passarem, o mesmo é retornado para a etapa de correção de *bug* de desenvolvimento, descrita na seção anterior.

A figura 14 ilustra a etapa de teste de *bug* de desenvolvimento e o responsável envolvido nas tarefas.

Figura 14 - Etapa teste de bug de desenvolvimento



Fonte: Elaboração da autora, 2016.

Na próxima seção é apresentada a etapa de publicação de funcionalidades, a qual é executada pelo líder do projeto com o auxílio do analista de qualidade. A mesma tem como critério de entrada as funcionalidades testadas e como saída as funcionalidades publicadas.

4.2.2.7 Etapa publicação de funcionalidade

A última etapa do processo de desenvolvimento de *software* da empresa PP LTDA, publicação de funcionalidades, inicia com a solicitação de publicação de funcionalidades pelo analista de qualidade. Sendo assim, o líder do projeto realiza o *merge* com a *branch master* das alterações de cada projeto e gera uma *tag* de *release* no padrão: PRO-0.0.0. Visto que, PRO é a sigla do projeto que está sendo alterado (*backend* e/ou *frontend*) e 0.0.0 refere-se à versão, que é a *fix-version* da *issue* do Jira, ou a próxima da sequência.

Após essa tarefa, o analista de qualidade executa a *job* do Jenkins, selecionando a *tag* de *relese* para que o *war* do projeto *backend* fique disponível na AWS. Após finalizar a *job* do Jenkins com sucesso, o líder do projeto aplica as modificações necessárias para a nova versão no ambiente de produção, como execução de *sql*, alteração em arquivos *json*, etc.

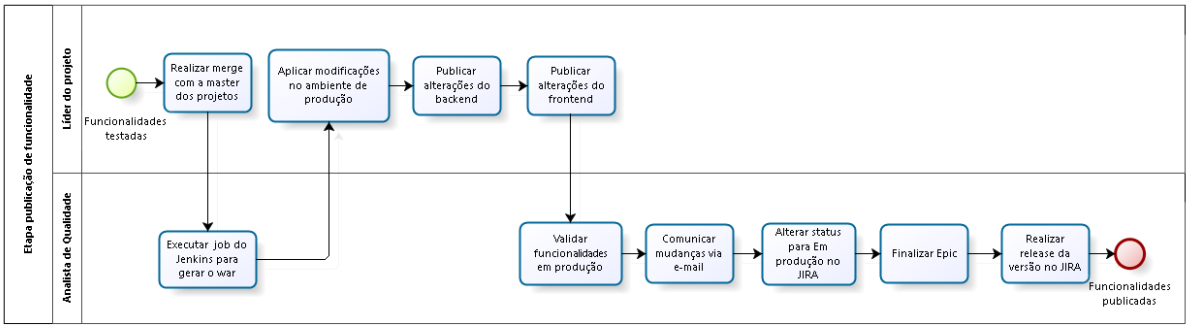
Feito isso, é possível disponibilizar as mudanças no ambiente de produção efetivamente. Primeiro deve ser feito o *deploy* das alterações do projeto *backend* em produção através da ferramenta AWS e após essa tarefa ter sido finalizado, a *job* do Jenkins de produção referente ao *frontend* é executada para que nova versão fique disponível aos usuários do sistema.

Para garantir que todas as funcionalidades foram publicadas, o analista de qualidade acessa o ambiente de produção com um usuário utilizado somente para testes e valida se as mesmas estão disponíveis e funcionando corretamente. Obtendo sucesso nessa tarefa, um e-mail de comunicação de mudanças é enviado para todos os colaboradores da empresa contendo a versão do sistema que foi publicada, as funcionalidades disponíveis e os clientes que solicitaram cada uma delas.

Por fim, o status da *story* no Jira é alterado para “Em Produção” e o analista de qualidade informa a data de publicação da funcionalidade e marca a opção do e-mail de comunicação de versão enviado. A *epic* é finalizada, visto que se a data da entrega for superior à data prevista, a mesma é finalizada em atraso e a release da versão é feita, informando a data da publicação.

A figura 15 ilustra as tarefas envolvidas durante essa última etapa e os papeis envolvidos.

Figura 15 - Etapa publicação de funcionalidade



Fonte: Elaboração da autora, 2016.

O ciclo de vida de gestão de funcionalidades atual da empresa do estudo de caso foi descrito e representado com figuras nessa seção. A seguir é apresentado o ciclo de vida de gestão de *bugs* de produção.

4.2.3 Ciclo de vida de gestão de bugs de produção

São encontrados erros no funcionamento comum de um *software* em produção por parte dos clientes ou dos colaboradores da empresa, os quais, posteriormente são enviados por e-mail e reportados automaticamente na ferramenta Jira.

Desse modo, na etapa verificação de *bugs*, os possíveis defeitos são verificados diariamente pelo analista de qualidade de cada projeto com o objetivo de tentar reproduzi-los. Caso o *bug* não persistir o mesmo é fechado e o motivo da ação é informado. No entanto, se o *bug* for confirmado, o caso é atualizado através da inserção das informações faltantes, de modo que fique mais claro para o entendimento do analista de sistema.

Além disso, mensalmente um calendário com o nome do *firefighter* [7] de cada projeto, por semana, é montado pelo gerente de TI na ferramenta Confluence, de modo que a divisão de correção de *bugs* seja justa entre todos os analistas de sistema da equipe.

Com a descrição detalhada do *bug* de produção confirmado, o analista de sistema responsável por corrigir os *bugs* reportados na semana, inicia a etapa de correção de bugs. Após ter encontrado a causa do *bug*, efetuado a correção e testado a mesma, o responsável pela tarefa a envia para teste.

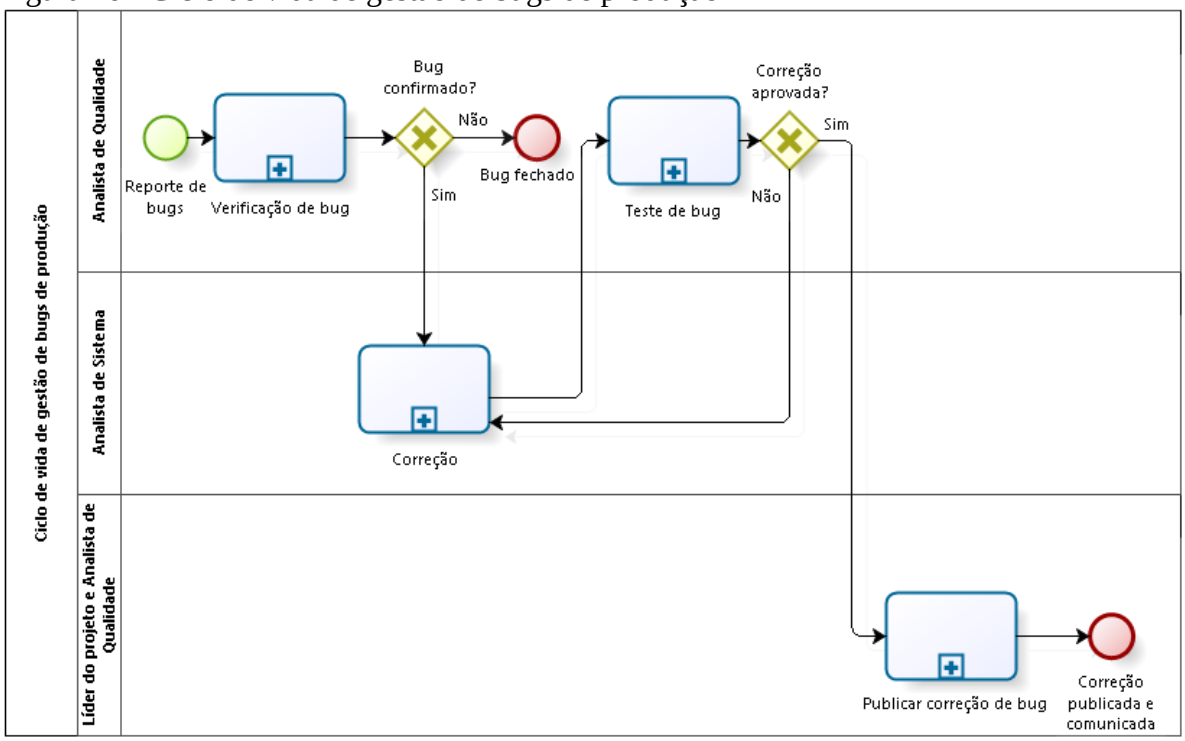
[7] A denominação *firefighter* refere-se ao Analista de Sistema responsável pela correção de bugs no ambiente de produção na semana.

Assim, o analista de qualidade do projeto inicia a etapa de teste de *bug*, a qual tem como objetivo validar se a correção resolveu efetivamente o problema e não gerou outros. Se os testes não tiverem sucesso, o analista de qualidade retorna o *bug* para o analista de sistema ajustar a correção. Após a nova correção, a etapa de teste de funcionalidade é realizada novamente. Caso seja verificado que a o *bug* foi corrigido e a solução proposta não gerou efeitos colaterais no sistema, o mesmo é aprovado.

Logo, inicia-se a etapa de publicação de *bug*, a qual o analista de qualidade informa ao líder o projeto que a correção foi aprovada e pode ser disponibilizada no ambiente de produção. Por fim, a comunicação da correção do *bug* é feita através de um e-mail enviado a todos os colaboradores da empresa.

A figura 16 representa de forma geral o ciclo de vida de gestão de *bugs* de produção da empresa PP LTDA e os papeis envolvidos em cada atividade. Logo, são detalhadas a seguir as etapas de verificação de *bug*, correção, teste de *bug* e publicação de *bug*.

Figura 16 - Ciclo de vida de gestão de bugs de produção



Fonte: Elaboração da autora, 2016.

Desse modo, a primeira etapa do ciclo de vida de gestão de *bugs* de produção, verificação de *bug*, realizada pelos analistas de qualidade de cada projeto da empresa PP LTDA, é descrita com detalhes na próxima seção. A mesma tem como critério de entrada o reporte de *bug* e como critérios de saída o *bug* fechado ou confirmado.

4.2.3.1 Etapa verificação de bug

Os *bugs* encontrados no ambiente de produção enviadas via e-mail pelos usuários das soluções da PP LTDA devem seguir o padrão apresentado no quadro 7. Os mesmos são cadastrados automaticamente no projeto triagem da ferramenta Jira com status “Novo” e tipo *Bug*.

Quadro 7 - Padrão de reporte de bug

Remente	E-mail exclusivo para envio de bugs
Assunto	[Sistema] – Descrição breve do problema (Nome da empresa)
Corpo do e-mail	a) Descrição detalhada do problema; b) Nome da empresa; c) Usuário (e-mail); d) Passos para reproduzir e) Anexo de imagens; f) Documentos que auxiliem na identificação do problema.

Fonte: Elaboração da autora, 2016.

No entanto, se o *bug* de produção for encontrado pelos colaboradores do setor de TI, o mesmo deve ser reportado no projeto correspondente na ferramenta Jira contendo todas as informações descritas no quadro 7, padrão de reporte de *bug*, e também o comportamento esperado do sistema caso não houvesse o *bug*.

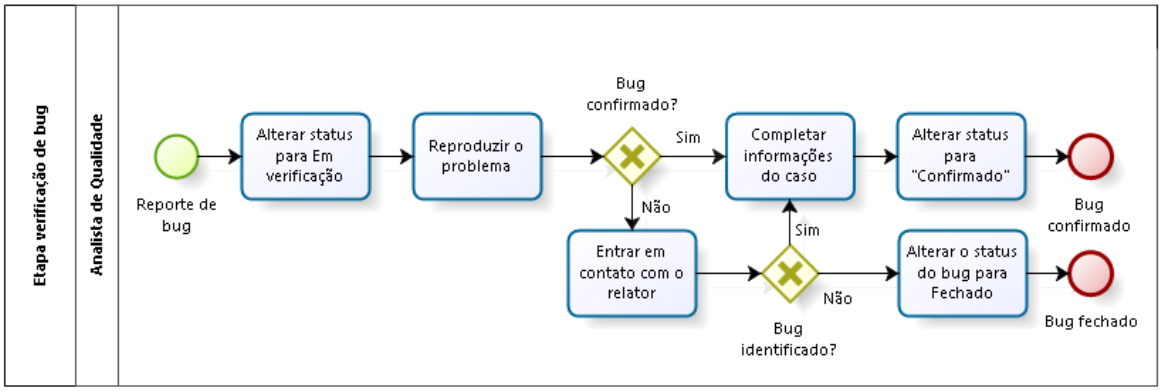
Feito isso, é necessário tentar reproduzir o problema. Logo, o analista de qualidade altera o status para “Em verificação”, atribuindo o caso para si, visto que, a tentativa deve ser realizada primeiramente no ambiente de desenvolvimento e se não for possível, no ambiente de produção. Caso o problema não seja confirmado, o analista de qualidade entra em contado com o relator do mesmo para que ele realize uma demonstração. Se ainda assim, não for possível reproduzir o problema, o caso é fechado e o motivo da ação é informado.

Por fim, se for possível reproduzir o *bug*, o mesmo será confirmado através dos seguintes passos: mover o caso para o projeto correspondente, completar informações

faltantes na descrição, informar a *label* interno[8] ou produção[9], versão afetada pelo bug, versão a qual o bug será corrigido, informar a *sprint*, a classificação do *bug* pela criticidade (Bloqueio, Crítico, Maior e Menor), atribuir o caso para o *firefighter* da semana e alterar *status* para “Confirmado”.

A figura 17 ilustra a etapa de verificação de *bug* de produção assim como o papel envolvido.

Figura 17 - Etapa verificação de bug



Fonte: Elaboração da autora, 2016.

Na próxima sessão, é detalhada a etapa de correção de *bug*, a qual é realizada pelos analistas de sistema. Por sua vez, o critério de entrada é o *bug* confirmado e o de saída, o *bug* pronto para teste.

4.2.3.2 Etapa correção de bug

Com o *bug* confirmado, o analista de sistema responsável pelas correções da semana inicia a tarefa pelos *bugs* de produção considerando a classificação pela criticidade e posteriormente os internos. Assim, o status é alterado para “Em implementação” e o nome do desenvolvedor é informado no caso.

Por sua vez, a *branch* é criada a partir da *master*, com o formato padrão *Hotfix*: HF-dia-mes-ano, onde serão feitas as correções. Assim, o *branch* correspondente aos *bugs*

[8] Interno refere-se à bugs que foram encontrados no ambiente de produção por um dos integrantes da TI.

[9] Produção refere-se à bugs que foram encontrados no ambiente de produção por um dos clientes da empresa.

resolvidos no dia leva como nome a data atual, e é obrigatória a inclusão do id do caso no texto dos *commits* para o Bitbucket.

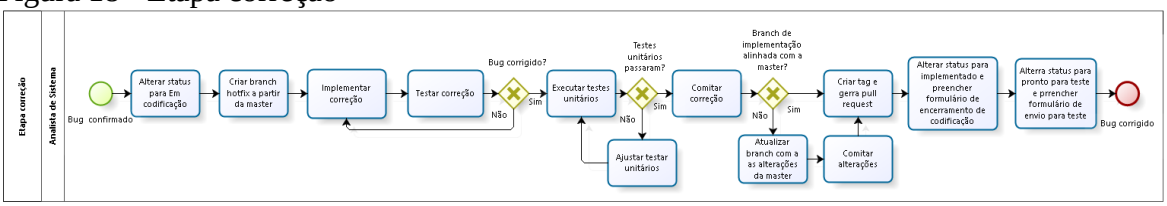
Após as correções, é necessário testar a mesma. Caso o *bug* não será corrigido efetivamente, a correção é iniciada novamente. No entanto, se o for corrigido, os testes unitários do projeto devem ser executados para validar se continuam funcionando. Visto que, se não estiverem passando, é necessário ajustá-los.

Feito isso, é realizado os *commmits*, sendo que, se a *branch* de correção não estiver atualizada com a *master*, será necessário ajusta-la. Assim, é aplicada uma *tag*, a qual será utilizada posteriormente pelo analista de qualidade, para realizar o *deploy* via Jenkins, seguindo o mesmo padrão da *branch Hotfix* e o *pull-request* para a *master* é feito, com o objetivo de lançar a nova versão da aplicação com os *bugs* do dia corrigidos.

Desse modo, o analista de sistema altera o status do caso para “Implementado” e preencher o formulário geral de encerramento de implementação apresentado na 4.2.1.2 Etapa Implementação. Posteriormente o status é alterado para “Pronto para teste” e o formulário de envio para teste, também apresentado na mesma seção do anterior, é preenchido.

Por fim, a figura 18 ilustra o a etapa de correção de *bugs* e o papel envolvido na atividade.

Figura 18 - Etapa correção



Fonte: Elaboração da autora, 2016.

Após finalizar essa etapa, se inicia a etapa de teste de *bug*, a qual é realizada pelo analista de qualidade. Seu critério de entrada são os *bugs* prontos para teste e o de saída são os *bugs* testados ou retornados para correção.

4.2.3.3 Etapa teste de bug

A etapa de testes de *bug* é iniciada pelo analista de qualidade quando os *bugs* corrigidos estão com status “Pronto para Teste” na ferramenta Jira. Assim a tarefa é atribuída para si quando o status é alterado para “Em Teste”.

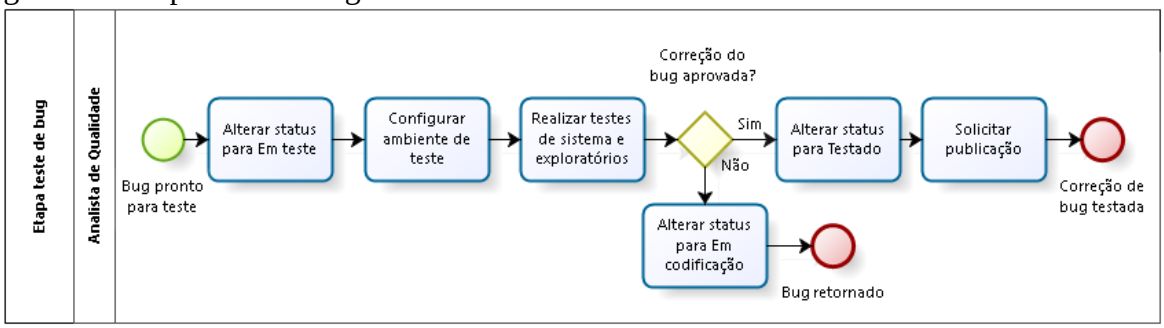
Desse modo, o ambiente para teste é configurado conforme as necessidades, através do provisionamento de um novo ambiente ou alteração da versão de um já existente utilizando a *tag* gerada pelo analista de sistema. Além disso, podem ser feitas alterações no j*son* e a execução do *sql* no banco de dados destinados a equipe de qualidade da empresa.

Terminado a configuração do ambiente para teste, o analista de qualidade verifica através dos testes de sistema e exploratórios se a correção resolve efetivamente o problema e se não causou nenhum efeito colateral. Caso todos os testes não forem concluídos com sucesso, o caso é retornado para correção e o status é alterado para “Em correção de bug”, o motivo da reprovação é informado no campo comentário e o *bug* é atribuído para o responsável.

Contudo, se após os testes da correção do *bug*, for verificado que o problema foi sanado, o mesmo será aprovado. Para isto, o analista de qualidade alterar o status “Testado” e solicita a publicação em ambiente de produção.

Por fim, a figura 19, ilustra a etapa de testes de *bug* e o papel envolvido na tarefa.

Figura 19 - Etapa teste de bug



Fonte: Elaboração da autora, 2016.

Na próxima seção é apresentada a etapa de publicação de *bug*, a qual é executada pelo líder do projeto com o auxílio do analista de qualidade. A mesma tem como critério de entrada os *bugs* testados e a saída é a disponibilização das correções no ambiente de produção.

4.2.3.4 Etapa publicação de bug

A última etapa do ciclo de vida de gestão de *bugs* de produção, publicação de *bug*, inicia com a solicitação de publicação das correções pelo analista de qualidade. Sendo assim, o líder do projeto realiza o *merge* com a *branch master* das alterações de cada projeto e gera uma *tag* de *release* no padrão: PRO-0.0.0. Visto que, PRO é a sigla do projeto que está sendo alterado (*backend* e/ou *frontend*) e 0.0.0 refere-se à versão, que é a *fix-version* da *issue* do Jira, ou a próxima da sequência.

Após essa tarefa, o analista de qualidade executa a *job* do Jenkins, selecionando a *tag* de *relese* para que o *war* do projeto *backend* fique disponível na AWS. Após finalizar a *job* do Jenkins com sucesso, líder aplica as modificações necessárias para a nova versão no ambiente de produção, como execução de *sql*, alteração em arquivos *json*, etc.

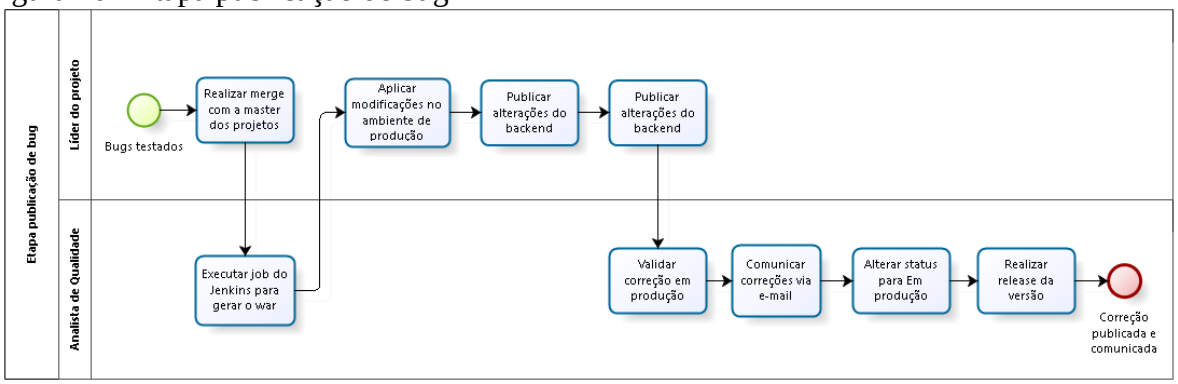
Feito isso, é possível disponibilizar as correções no ambiente de produção efetivamente. Primeiro deve ser feito o *deploy* das alterações do projeto *backend* em produção através da ferramenta AWS e após essa tarefa ter sido finalizado, a *job* do Jenkins de produção referente ao *frontend* é executada para que nova versão fique disponível aos usuários do sistema.

Para garantir que todas as correções estão corretas no ambiente de produção, o analista de qualidade acessa o sistema com um usuário utilizado somente para testes e valida se as mesmas estão disponíveis e funcionando corretamente. Obtendo sucesso nessa tarefa, um e-mail de comunicação de correção é enviado para todos os colaboradores da empresa contendo a versão do sistema que foi publicada, as correções disponíveis e os clientes que reportaram cada uma delas.

Por fim, o status do *bug* no Jira é alterado para “Em Produção” e o analista de qualidade informa a data de correção do *bug* e marca a opção do e-mail de comunicação de versão enviado.

A figura 20 ilustra as tarefas envolvidas durante essa última etapa e os papeis envolvidos.

Figura 20 - Etapa publicação de bug



Fonte: Elaboração da autora, 2016.

Finalizado a modelagem do processo de desenvolvimento de *software* da empresa do estudo de caso, são apresentadas as melhorias propostas para o processo na próxima seção.

4.3 PROPOSTA DE MELHORIAS

A seguir são apresentadas as propostas de melhorias com foco na qualidade do processo e, por conseguinte do produto, para algumas etapas do ciclo de vida de gestão de funcionalidades e ciclo de vida de gestão de *bugs* de produção, os quais compõem o processo de desenvolvimento de *software* da empresa do estudo de caso, PP LTDA. Uma vez que, não serão apresentadas melhorias para as seguintes etapas: verificação de funcionalidade, correção de *bug* de desenvolvimento e teste de *bug* de desenvolvimento.

Após a descrição da proposta, para cada etapa, será incluído um quadro com as seguintes informações:

- Nº: indica um número sequencial para auxiliar na identificação.
- Descrição: possui a apresentação de cada melhoria proposta.
- Responsável (eis): contém os papeis necessários para a realização de cada melhoria proposta. Visto que, nesta análise, são utilizados os seguintes valores:
    - Time: quando todos os colaboradores do time são responsáveis, inclusive o líder;
    - Gerente de TI: quando a presença do gerente do setor for necessária;
    - Analista de Qualidade: quando a melhoria estiver relacionada com a etapa de testes.

- Analista de Sistema: quando a melhoria estiver relacionada com a etapa de implementação e correção.

- Objetivo: explica o objetivo de cada melhoria proposta.

A seguir, o quadro 8, representa o modelo utilizado para descrição das melhorias propostas.

Quadro 8 - Modelo para descrição das melhorias propostas

Nº	**Descrição**	**Responsável (eis)**	**Objetivo**

Fonte: Elaboração da autora, 2016.

4.3.1 Etapa planejamento

Inicialmente é necessário que a atividade de refinamento do *backlog* não seja mais somente de responsabilidade do gerente de TI, incluindo todo o time do projeto na mesma. Logo, quando os colaboradores participam da especificação da tarefa, podem surgir ideias melhores de solução e o entendimento do que deve ser feito fica mais claro a todos. Desse modo, o gerente de TI, apenas daria a diretriz inicial da reunião, explicitando as atividades candidatas a entrar na *sprint* e o objetivo de cada uma.

Feito isso, deve-se focar na descrição das funcionalidades a serem implementadas, pois esta não é padronizada, o que dificulta o entendimento do analista de sistema durante a implementação e do analista de qualidade na etapa de teste. Logo, a falta de informações leva o time, muitas vezes, a entregar um resultado diferente do que era esperado.

Sendo assim, é necessário padronizar a descrição das funcionalidades, dividindo-a nos módulos: “o que?“, “por que?”, “quem?” e “como?”. Uma vez que, “o que?”, refere-se à descrição de o que deve ser feito na atividade, contendo de forma objetiva os pontos que devem ser implementados. O “por que?”, deve conter a descrição da necessidade da atividade e também a do problema a ser resolvido. “Quem?”, diz respeito ao perfil do usuário que poderá acessar a funcionalidade, de modo que, deve haver informações suficientes para saber se é necessário criar um novo perfil ou utilizar um existente.

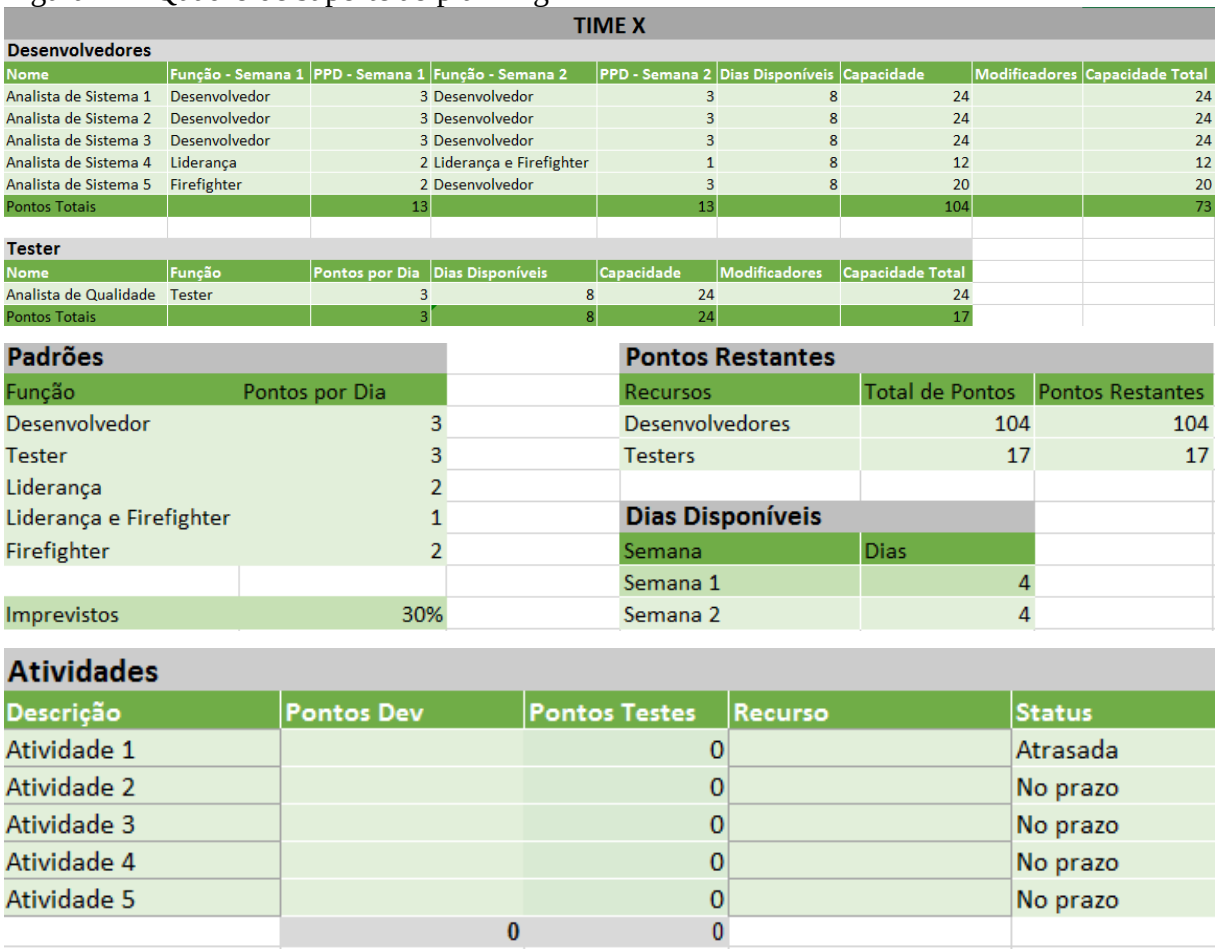
E por fim, o “como?”, precisa conter a descrição de como a funcionalidade será implementada, ou seja, a solução técnica e os elementos (telas, botões, campos, etc.) que serão criados durante a implementação. Também deve ser incluída, como a atividade será testada (tipos de teste, critérios de aceitação, etc.).

Com a descrição melhorada, agora é preciso dividir as responsabilidades dessa atividade, sendo que, o gerente de TI deve se encarregar de preencher “o que?”, “por que?” e “quem?” e o time o “como?”. Além disso, esses campos devem ser criados na ferramenta Jira como obrigatórios, e devem ser preenchidos antes de confirmar a atividade.

Por sua vez, as propostas de melhoria para o refino do *backlog* foram apresentadas, contudo o *planning* também precisa de mudanças. Logo, o time do projeto deve ter autonomia para decidir quantos pontos, referentes ao esforço de implementação são necessários para realizar determinada tarefa e também quantos pontos para que a mesma seja testada, de modo que, a etapa de teste pode ser um gargalo, por haver apenas um analista de qualidade em cada projeto. Sendo assim, se alguma atividade não couber na *sprint* devido aos testes, ela deve ser trocada por outra ou até mesmo retirada. Com o time pontuando as atividades, a tendência é aumentar a assertividade das estimativas, uma vez que, o mesmo conhece seus pontos fracos e fortes.

Para dar suporte ao *planning*, é necessário que uma tabela *excel*, representada pela figura 21, seja criada para cada projeto, com o objetivo de ajudar no cálculo das pontuações e também manter um histórico da capacidade do time.

Figura 21 - Quadro de suporte ao planning



TIME X

Desenvolvedores

Nome	Função - Semana 1	PPD - Semana 1	Função - Semana 2	PPD - Semana 2	Dias Disponíveis	Capacidade	Modificadores	Capacidade Total
Analista de Sistema 1	Desenvolvedor	3	Desenvolvedor	3	8	24		24
Analista de Sistema 2	Desenvolvedor	3	Desenvolvedor	3	8	24		24
Analista de Sistema 3	Desenvolvedor	3	Desenvolvedor	3	8	24		24
Analista de Sistema 4	Liderança	2	Liderança e Firefighter	1	8	12		12
Analista de Sistema 5	Firefighter	2	Desenvolvedor	3	8	20		20
Pontos Totais		13		13		104		73

Tester

Nome	Função	Pontos por Dia	Dias Disponíveis	Capacidade	Modificadores	Capacidade Total
Analista de Qualidade	Tester	3	8	24		24
Pontos Totais		3	8	24		17

Padrões

Função	Pontos por Dia
Desenvolvedor	3
Tester	3
Liderança	2
Liderança e Firefighter	1
Firefighter	2
Imprevistos	30%

Pontos Restantes

Recursos	Total de Pontos	Pontos Restantes
Desenvolvedores	104	104
Testers	17	17

Dias Disponíveis

Semana	Dias
Semana 1	4
Semana 2	4

Atividades

Descrição	Pontos Dev	Pontos Testes	Recurso	Status
Atividade 1		0		Atrasada
Atividade 2		0		No prazo
Atividade 3		0		No prazo
Atividade 4		0		No prazo
Atividade 5		0		No prazo
	0	0		

Fonte: Elaboração da autora, 2016.

Primeiramente foram definidos os padrões de pontos por dia para cada função, sendo estas: desenvolvedor, liderança, *firefighter*, *firefighter* e liderança ou *tester* e a quantidade de dias disponíveis por semana. Posteriormente, foram listados os colaboradores do time com suas respectivas funções na semana um e dois e feitos os cálculos automáticos da capacidade total de implementação e de teste.

Em relação a modificadores, eles se referem a acontecimentos que podem diminuir a quantidade de pontos de uma determinada pessoa por ela estar alocada em outra tarefa como correção de *bugs* acumulados, por exemplo. Além disso, também é subtraída da capacidade total, uma porcentagem de imprevistos, a qual pode variar de um time para outro.

Desse modo, são listadas as atividades candidatas a entrar na *sprint* e são atribuídos pontos de desenvolvimento e de teste, sendo que a soma não pode ultrapassar a capacidade total prevista. Além disso, as atividades devem ser divididas nas menores partes possíveis, não ultrapassando oito pontos, com o objetivo de facilitar a implementação e o teste. Enquanto que, as atividades que não cumpriram o prazo de entrega estabelecido na *sprint* anterior, entram na *sprint* atual com status de atrasada.

Para ser mais assertivo ao pontuar as *stories* é necessário levar em consideração o esforço necessário, o grau de complexidade e risco de impacto. Logo seria possível utilizar a tabela apresentada no quadro 9.

Quadro 9 - Tabela para pontuar stories

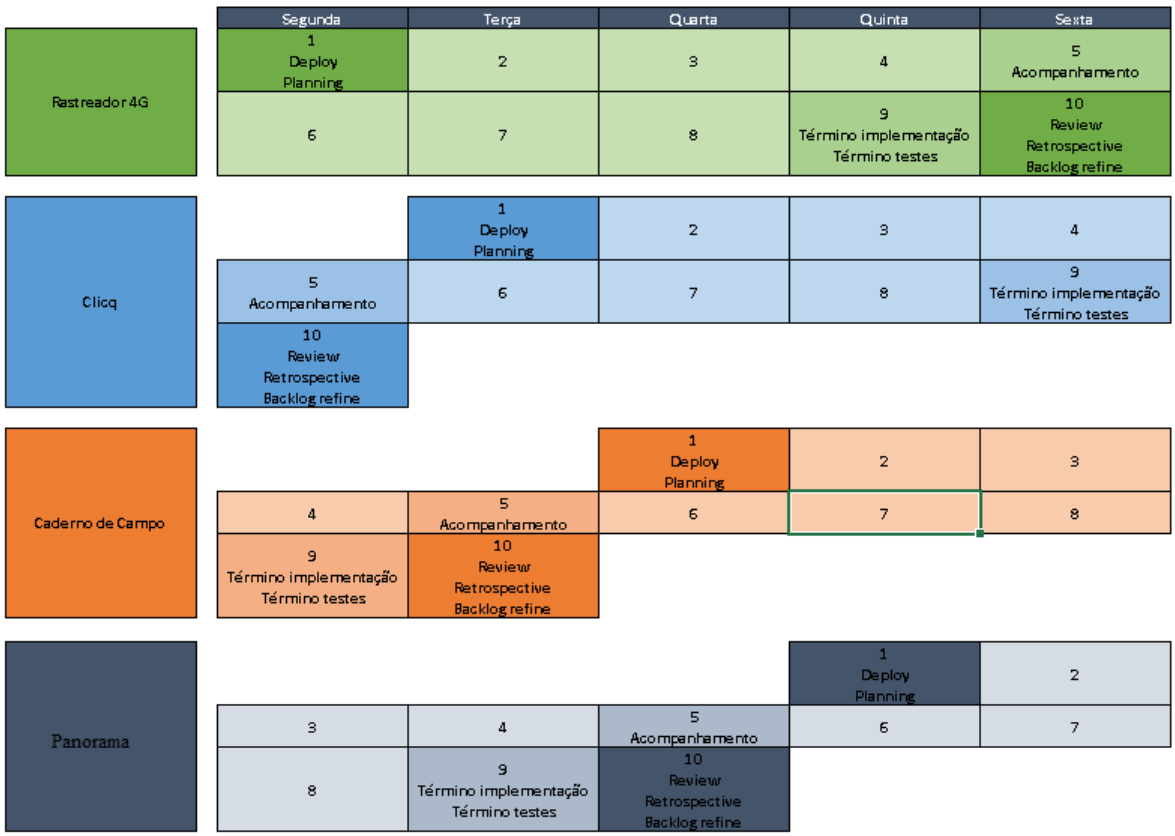
	Esforço	**Complexidade**	**Risco**	**Faixa de story point**
Baixo	0 a 1	0 a 1	0 a 1	1 - 2 - 3
Médio	2 a 3	2 a 3	2 a 3	5 a 8
Alto	Acima de 3	Acima de 3	Acima de 3	Desmembrar

Fonte: Elaboração da autora, 2016.

Terminado o planejamento da *sprint*, a empresa do estudo de caso, não tem como prática a criação dos casos de teste das atividades que serão implementadas pelo analista de sistema. Assim, ele testa de acordo com os seus critérios. Dessa forma, como melhoria é possível utilizar um *software* para gestão de casos de teste. Por sua vez o analista de qualidade deve criar os casos de teste de acordo com a descrição do caso para que o mesmo seja usado no início da implementação, após a mesma e na etapa de teste.

Outro ponto relevante é a criação de uma agenda da *sprint* para cada projeto. As *sprints* devem iniciar e terminar em dias diferentes para não haver choque de horários e também para o gerente de TI poder participar das atividades que lhe são cabíveis. Sendo assim, a figura 22 ilustra como deve ser a agenda de *sprints* e as atividades realizadas em cada dia da semana.

Figura 22 - Agenda de sprints



Fonte: Elaboração da autora, 2016

A *sprint* inicia com o refino do *backlog* do produto, para melhorar a descrição das atividades. Posteriormente, é feito o *planning* para revisar as atividades, atribuir as pontuações referentes ao esforço de desenvolvimento e teste e verificar quais *stories* irão compor a *sprint.* Feito isso, inicia-se a implementação e conforme as atividades vão ficando prontas, o teste também é iniciado.

Após o termino do teste, é realizado o *review* da atividade e se tudo estiver de acordo com o esperado, a mesma já é disponibilizada no ambiente de produção. No meio da *sprint* é feita uma reunião de acompanhamento com o líder do projeto para verificar o andamento do trabalho. Por fim, a implementação e os testes devem finalizar um dia antes do termino da *sprint* e a última entrega deve ser feita no segundo dia da próxima *sprint*.

O quadro 10 representa as melhorias propostas para a etapa de planejamento.

Quadro 10 - Melhorias na etapa planejamento

Nº	Descrição	Responsável (eis)	Objetivo
1.	Incluir todos os participantes do projeto no refino do *backlog*.	Time e gerente de TI	Possibilitar melhores propostas de solução e deixar mais claro o que deve ser feito.
2.	Padronizar a descrição das funcionalidades.	Time e gerente de TI	Melhorar a descrição das atividades.
3.	Dividir as responsabilidades da descrição da funcionalidade.	Time e gerente de TI	Dar autonomia ao time para que o mesmo chegue a melhor solução de implementação e testes.
4.	Dividir as atividades nas menores partes possíveis. A pontuação da *story* não deve ultrapassar oito.	Time	Facilitar a implementação e o teste.
5.	Dar autonomia para os participantes dos projetos realizarem o *planning*.	Time	Aumentar a assertividade das estimativas das atividades.
6.	Criar quadro de suporte ao *planning*.	Time	Ajudar no cálculo das pontuações e manter um histórico da capacidade do time.
7.	Considerar esforço necessário, grau de complexidade e risco de impacto ao pontuar as *stories*.	Time	Aumentar a assertividade ao pontuar as *stories*.
8.	Criar casos de teste para as atividades da *sprint* com o auxílio de um *software*.	Analistas de qualidade	Auxiliar o analista de sistema e o analista de qualidade a testar o que foi implementado.
9.	Criar agenda da *sprint* para cada projeto.	Gerente de TI	Evitar choque de horários e garantir a presença do gerente de TI quando necessário.

Fonte: Elaboração da autora, 2016.

Na próxima seção são apresentadas as melhorias propostas para a etapa de implementação do processo de desenvolvimento de *software*.

4.3.2 Etapa implementação

Na seção anterior, referente às melhorias na etapa de planejamento, foi proposta a criação dos casos de teste pelo analista de qualidade com o auxílio de um *software*. Dessa

forma, o analista de sistema, deve revisar os mesmos antes de iniciar a implementação com o objetivo de orientar o desenvolvimento e sanar possíveis dúvidas. Além disso, é preciso relatar novos casos, se outros cenários forem encontrados durante a implementação.

Caso a funcionalidade a ser implementada precise de novos serviços ou modificação dos existentes, o analista de sistema deve criar os testes de serviço com o apoio de um *software*, exportar o arquivo com os testes e anexar no caso. Desse modo, pode ser feito a automação dos testes de serviço com o auxílio do analista de qualidade e assim, melhor a qualidade do sistema. Além disso, antes de enviar para teste, o analista de sistema deve executar todos os testes de serviço existentes para garantir que os mesmos continuam funcionando após as mudanças na camada *backend*.

A revisão do código, também chamado do *code review*, tem como proposito validar o código implementado por outros colaboradores do time e o seu objetivo final é gerar valor para o cliente, tendo como resultado um *software* melhor. Desse modo, para melhorar a etapa de implementação é necessário incluir essa atividade. Visto que, para realizar *o code review*, o analista de sistema deve jogar um dado ao terminar cada *story*, no qual cada número represente um dos colaboradores do projeto, exceto o analista de qualidade.

Uma prática que pode evitar problemas e diminuir o retrabalho é o acompanhamento do analista de qualidade antes de enviar atividades mais complexas para teste. Assim, o analista de sistema deve realizar uma demonstração para que seja verificado se o resultado da atividade está de acordo com o esperado. Visto que, o nível da dificuldade da atividade fica a critério do analista de sistema e o mesmo decide se é necessário realizar a demonstração.

Por fim, são enfrentados muitos problemas durante a implementação de um *software* e com o objetivo de compartilhar as soluções encontradas, poderia ser criada uma página referentes ao “o que eu aprendi hoje?”, a qual deve ser encaminhada no final da semana, para a lista de colaboradores do setor de tecnologia de informação.

O quadro 11 representa as melhorias propostas para a etapa de implementação.

Quadro 11 - Melhorias na etapa de implementação

Nº	Descrição	Responsável (eis)	Objetivo
10.	Revisar casos de teste antes de iniciar a implementação.	Analistas de sistema	Orientar o desenvolvimento e sanar possíveis dúvidas.
11.	Incluir novos casos de teste encontrados durante a implementação.	Analistas de sistema	Melhorar os casos de teste elaborados pelo analista de qualidade.
12.	Realizar testes de serviço com o auxílio de um *software*.	Analistas de sistema	Auxiliar o analista de qualidade na automação dos testes de serviço.
13.	Executar testes automatizados de serviços existentes antes de mandar para teste, quando a mudança for na camada *backend*.	Analistas de sistema	Garantir que os serviços existentes continuam funcionando após as mudanças.
14.	Realizar prática de revisão do código (*code review*).	Analistas de sistema	Validar o código implementado por outro colaborador do time.
15.	Solicitar ao analista de qualidade do projeto que acompanhe uma demonstração das atividades mais complexas antes de enviar para teste.	Analistas de sistema	Evitar problemas e retrabalho.
16.	Criar página com “o que aprendi hoje? ” e enviar semanalmente.	Analistas de sistema	Compartilhar soluções de problemas enfrentados no dia-a-dia.

Fonte: Elaboração da autora, 2016.

Na próxima seção são apresentadas as melhorias propostas para a etapa de teste de funcionalidade do processo de desenvolvimento de *software*.

4.3.3 Etapa teste de funcionalidade

No processo atual não há automação dos testes de *interface* das novas funcionalidades, logo é necessário realiza-los. Visto que, a automação deve ser feita no início da próxima *sprint*, pois é quando o analista de qualidade possui tempo livre.

Além disso, também é preciso automatizar os serviços já existentes no sistema com o auxílio de uma ferramenta com o objetivo de evitar *bugs* na camada *backend* das funcionalidades.

A empresa também não conta com testes de regressão automatizados, logo é preciso criar uma suíte de testes, inicialmente das principais funcionalidades com o auxílio de uma ferramenta e essa deve ser executada antes de disponibilizar cada nova versão do sistema.

Também não há um padrão documentado de testes de configuração, seja este para ambiente *web*, dispositivo móvel ou *desktop*. Sendo assim, é necessário definir quais os navegadores serão utilizados para teste, quais sistemas operacionais e versão dos mesmos serão testados, assim como a definição de alguns dispositivos e tamanhos de tela. E por fim, quais versões do sistema operações Windows a versão *desktop* do sistema irá suportar. O objetivo é evitar e/ou minimizar os *bugs* gerados através da falta de todos os testes de configuração necessários.

Como a empresa possui algumas das suas principais soluções para dispositivos móveis, faz-se necessário incluir a automação de testes mobile com o auxílio de uma ferramenta, na rotina dos analistas de qualidade.

Por sua vez, os analistas de qualidade estão distribuídos entre os projetos da empresa e acabam não mantendo muito contato. Assim é necessário realizar reuniões semanais para verificar e alinhar as atividades e também receber *feedbacks*, sendo estes positivos e negativos.

Outro ponto de melhoria, é a criação de uma lista de problemas básicos que são encontrados comumente na implementação para que esta seja colocada no Jira. Desse modo, quando um dos erros ocorrerem o analista de qualidade retornará o caso para o analista de sistema, marcando o problema identificado.

O quadro 12 representa as melhorias propostas para a etapa de teste.

Quadro 12 - Melhorias na etapa de teste

Nº	Descrição	Responsável (eis)	Objetivo
17.	Automatizar novas funcionalidades implementadas.	Analistas de qualidade	Evitar *bugs* em novas funcionalidades.
18.	Automatizar serviços já existentes no sistema com o auxílio de uma ferramenta.	Analistas de qualidade	Evitar bugs nas funcionalidades já existentes.
19.	Automatizar e executar testes de regressão.	Analistas de qualidade	Validar se as funcionalidades existentes estão funcionando corretamente.
20.	Padronizar e documentar testes de configuração para *web*.	Analistas de qualidade	Evitar *bugs* devido aos diferentes navegadores.
21.	Padronizar e documentar testes de configuração para dispositivos móveis.	Analistas de qualidade	Minimizar o número de *bugs* devido aos diferentes tipos de sistema operacionais e dispositivos.
22.	Padronizar e documentas testes de configuração para *desktop*.	Analistas de qualidade	Evitar *bugs* devido aos diferentes sistemas operacionais.
23.	Automatizar testes em dispositivos móveis com o auxílio de uma ferramenta.	Analistas de qualidade	Garantir que as funcionalidades estão funcionando corretamente.
24.	Criar lista de boas práticas da ferramenta utilizada para testes de interface	Analistas de qualidade	Padronizar o desenvolvimento dos testes de *interface*.
25.	Realizar reuniões semanais com os analistas de qualidade.	Analistas de qualidade	Alinhar atividades com os analistas de qualidade.
26.	Listar problemas básicos que são encontrados comumente na implementação.	Analistas de qualidade	Criar lista de problemas para inserir no Jira.

Fonte: Elaboração da autora, 2016.

Na próxima seção são apresentadas as melhorias propostas para a etapa de publicação de funcionalidade do processo de desenvolvimento de *software*.

4.3.4 Etapa publicação de funcionalidade

Antes de publicar uma mudança no ambiente de produção é necessário validar se o resultado entregue está de acordo com o esperado. Para melhor essa etapa, é necessário incluir o evento de *review*, no qual o analista de qualidade apresenta as funcionalidades que estão prontas para o gerente de TI e recebe o seu *feedback*. Desse modo, se precisar de alguma

alteração, esta já é realizada, e se houver alguma melhoria, a mesma é relatada e vai para o *backlog* do produto.

Os testes automatizados de serviço e de *interface* do projeto também devem ser executados, com o objetivo de garantir que as funcionalidades antigas continuam funcionando corretamente.

No final da primeira semana da *sprint*, deve ser realizada a reunião o acompanhamento da mesma entre o líder do projeto e o gerente de TI, com o objetivo de verificar o andamento das atividades.

Além disso, seria interessante criar um *blog* para apresentar as funcionalidades que foram disponibilizadas no ambiente de produção em cada semana do mês e de cada produto, com o objetivo de informar a todos os colaboradores da empresa as novidades do sistema.

Após o termino de uma *sprint*, poderia ser realizado um treinamento com o setor de operação (pós-venda) em relação às funcionalidades disponibilizadas. Por sua vez, deve-se montar um calendário com todos os colaboradores de cada projeto, para que de duas em duas semanas, uma pessoa seja responsável por montar a apresentar o treinamento. Essa melhoria resulta em um melhor atendimento aos clientes da empresa.

Também é preciso incluir o evento de retrospectiva na etapa de encerramento com o objetivo de debater e relatar as lições aprendidas durante cada *sprint*. Logo, todo o time deve anotar os acontecimentos para que posteriormente seja apontado o que foi bom e o que pode ser melhorado. Visto que, para a lista do que pode ser melhorado, devem ser feitos planos de ações, os quais são designados para um ou mais colaboradores do time. Na próxima *sprint*, os planos de ação são revisados, para verificar os que já foram concluídos e quais irão continuar na lista.

Por fim, outra mudança necessária, é a inclusão de uma nova etapa, chama encerramento, para contemplar as propostas dos itens 29, 30, 31 e 32.

O quadro 13 apresenta as melhorias propostas na etapa de publicação de funcionalidades e para a nova etapa de encerramento.

Quadro 13 - Melhorias na etapa de publicação de funcionalidades

Nº	Descrição	Responsável (eis)	Objetivo
27.	Realizar *review* antes de publicar a versão.	Analista de qualidade e Gerente de TI	Inspecionar o que foi feito e coletar *feedback*.
28.	Executar automação de testes antes de publicar a versão.	Analistas de qualidade	Verificar se as funcionalidades continuam funcionando.
29.	Criar um blog para apresentar as funcionalidades que foram disponíveis em produção na semana.	Analistas de qualidade	Informar a todos os colaboradores da empresa as novidades da semana.
30.	Fornecer treinamento para o setor de Operação (pós-venda) da empresa.	Time	Treinar o setor de operação em relação às funcionalidades do sistema para melhorar o atendimento ao cliente.
31.	Incluir o evento de retrospectiva do S*crum*.	Time	Relatar as lições aprendidas durante cada *sprint* e propor planos de ação para o que pode ser melhorado.
32.	Realizar reunião de Acompanhamento no final da primeira semana de *sprint*.	Líder do projeto e gerente de TI	Verificar andamento da *sprint*.

Fonte: Elaboração da autora, 2016.

Na próxima seção são apresentadas as melhorias propostas para a etapa de verificação de *bug* do processo de desenvolvimento de *software*.

4.3.5 Etapa verificação de bug

Durante a etapa de verificação de *bugs* reportados não é feito nenhuma identificação do módulo do sistema afetado, caso o problema seja confirmado. Logo, para melhorar esse ponto, é necessário incluir um campo na ferramenta Jira para que seja informada a sigla do projeto, o módulo e a operação afetada. Exemplo: RST-Novo Fornecedor, Inserção.

Feito a identificação é necessário criar um relatório mensal com os *bugs* de produção e agrupar em módulos para verificar em qual parte do sistema se concentra o maior número de *bugs* e se os mesmos se referem à cama *backend* ou *frontend.* Desse modo é possível criar planos de ação para resolver o problema.

O quadro 14 apresenta as melhorias propostas para o processo de verificação dos bugs encontrados no ambiente de produção.

Quadro 14 - Melhorias no processo de verificação

Nº	Descrição	Responsável (eis)	Objetivo
33.	Identificar módulos do sistema com maior ocorrência de *bugs* de produção.	Analistas de qualidade	Gerar dados para análise.
34.	Criar relatório com módulos do sistema com mais ocorrência de *bugs*.	Analistas de qualidade	Gerar planos de ação.

Fonte: Elaboração da autora, 2016.

Na próxima seção são apresentadas as melhorias propostas para a etapa de correção do processo de desenvolvimento de *software*.

4.3.6 Etapa correção

Após a correção dos *bugs*, também é importante implantar a prática do *code review* com o objetivo de validar o código modificado por outro colaborador do time. Visto que, algumas vezes as correções aplicadas podem causar problemas em outros pontos do sistema que estavam funcionando corretamente.

Como descrito na seção 4.2.3.2, cada semana há um analista de sistema responsável pela correção de *bugs*. Contudo, em alguns momentos os *bugs* de produção acumulam e apenas um colaborador não consegue dar conta da correção. Logo, é preciso que os outros membros do time auxiliem na resolução para que a *sprint* não seja afetada.

O quadro 15 apresenta as melhorias propostas para o processo de verificação dos *bugs* encontrados no ambiente de produção.

Quadro 15 - Melhorias na etapa de correção

Nº	Descrição	Responsável (eis)	Objetivo
35.	Realizar prática de revisão do código (*code review*).	Analistas de sistema	Validar o código modificado por outro colaborador do time.
36.	Colaboração na correção de *bugs* caso tenha muitos bugs de produção acumulados.	Analistas de sistema	Auxiliar o *firefigther* na correção dos bugs de produção para não atrasar a *sprint*.

Fonte: Elaboração da autora, 2016.

Na próxima seção são apresentadas as melhorias propostas para a etapa de teste de *bug* do processo de desenvolvimento de *software*.

4.3.7 Etapa teste de bug

Após a correção dos bugs de produção, os testes são realizados somente de forma manual. Assim, é necessário que seja feita a automação dos mesmos para que não voltem a se repetir. Visto que, se o *bug* for na camada *backend* do sistema devem ser feitos teste de serviço e se for *frontend*, teste de *interface*.

O quadro 16 apresenta as melhorias propostas na etapa de teste de *bugs* encontrados no ambiente de produção.

Quadro 16 - Melhorias na etapa de teste de bugs

Nº	Descrição	Responsável (eis)	Objetivo
37.	Automatizar *bugs* corrigidos	Analistas de qualidade	Evitar que o *bug* volte a se repetir.

Fonte: Elaboração da autora, 2016.

Na próxima seção são apresentadas as melhorias propostas para a etapa de publicação de *bug* do processo de desenvolvimento de *software*.

4.3.8 Etapa publicação de bug

Para facilitar a comunicação da correção dos *bugs* de produção que estão disponíveis deve ser criado um grupo de avisos no mensageiro Hipchat. Assim, o analista de qualidade pode informar o caso do Jira que foi corrigido logo após a publicação.

O quadro 16 apresenta as melhorias propostas na etapa de publicação de correções de *bugs* encontrados no ambiente de produção.

Quadro 17 - Melhorias na etapa de publicação de bugs

Nº	**Descrição**	**Responsável (eis)**	**Objetivo**
38.	Criar grupo de avisos	Analistas de Qualidade	Comunicar o setor de operação (pós-venda) os *bugs* corrigidos.

Fonte: Elaboração da autora, 2016.

Após a descrição de todas as propostas de melhoria para o processo de desenvolvimento de *software* da empresa do estudo de caso, é necessário realizar a modelagem o novo processo, a qual é apresentada a seguir.

4.4 NOVO PROCESSO DE DESENVOVIMENTO DE SOFTWARE DA EMPRESA DO ESTUDO DE CASO

A seguir é apresentado o novo modelo de processo de desenvolvimento de *software* da empresa do estudo de caso, levando em conta as melhorias propostas nas etapas do ciclo de vida de gestão de funcionalidade e do ciclo de vida de gestão de *bugs* de produção.

4.4.1 Ferramentas de apoio aos testes do novo processo da empresa

Além das ferramentas já utilizadas pela empresa do estudo de caso, PP LTDA, nesta seção serão propostas outras ferramentas necessários para implantação das melhorias e que irão apoiar o novo processo.

4.4.1.1 Appium

O Appium é um *framework open source* multi-plataforma de automação de testes de *software* para dispositivos móveis. O mesmo pode ser usado para automatizar aplicações nativas, híbridas e sites *mobile* para os principais sistemas operacionais do mercado: android e ios. (APPIUM, 2016).

4.4.1.2 IntelliJ IDEA

Segundo a Jetbrains (2016), o IntelliJ IDEA é um ambiente de desenvolvimento integrado, o qual que foi projetado para maximizar a produtividade do desenvolvedor. Além disso, este *software* comercial, possui um *design* ergonômico.

4.4.1.3 Protractor

O Protractor é um framework de testes *end-to-end* para aplicações AngularJS que permite executar testes automatizados contra a aplicação em execução no navegador, simulando o comportamento de um usuário real. (PROTRACTOR, 2016).

4.4.1.4 Postman

Segundo o Google (2016), o Postman é um aplicativo que permite realizar requisições HTTP a partir de uma interface facilitando a automação de testes de serviço REST.

4.4.1.5 TestRail

Segundo o Gurock (2016), o TestRail é um *software* comercial, *web* de gestão de casos de teste, o qual possui integração com o Jira. O mesmo é de fácil utilização e permite a troca de status do caso de teste conforme necessário.

A figura 23 apresenta as ferramentas de apoio aos testes para o processo de desenvolvimento de *software* da empresa PP LTDA.

Figura 23 - Ferramentas

Fonte: Elaboração da autora, 2016.

4.4.2 Ciclo de vida de gestão de funcionalidade

A seguir são apresentadas as melhorias propostas para cada etapa do ciclo de vida de gestão de funcionalidades, exceto para etapa verificação, correção de *bug* de desenvolvimento e teste de *bug* de desenvolvimento.

4.4.2.1 Etapa planejamento

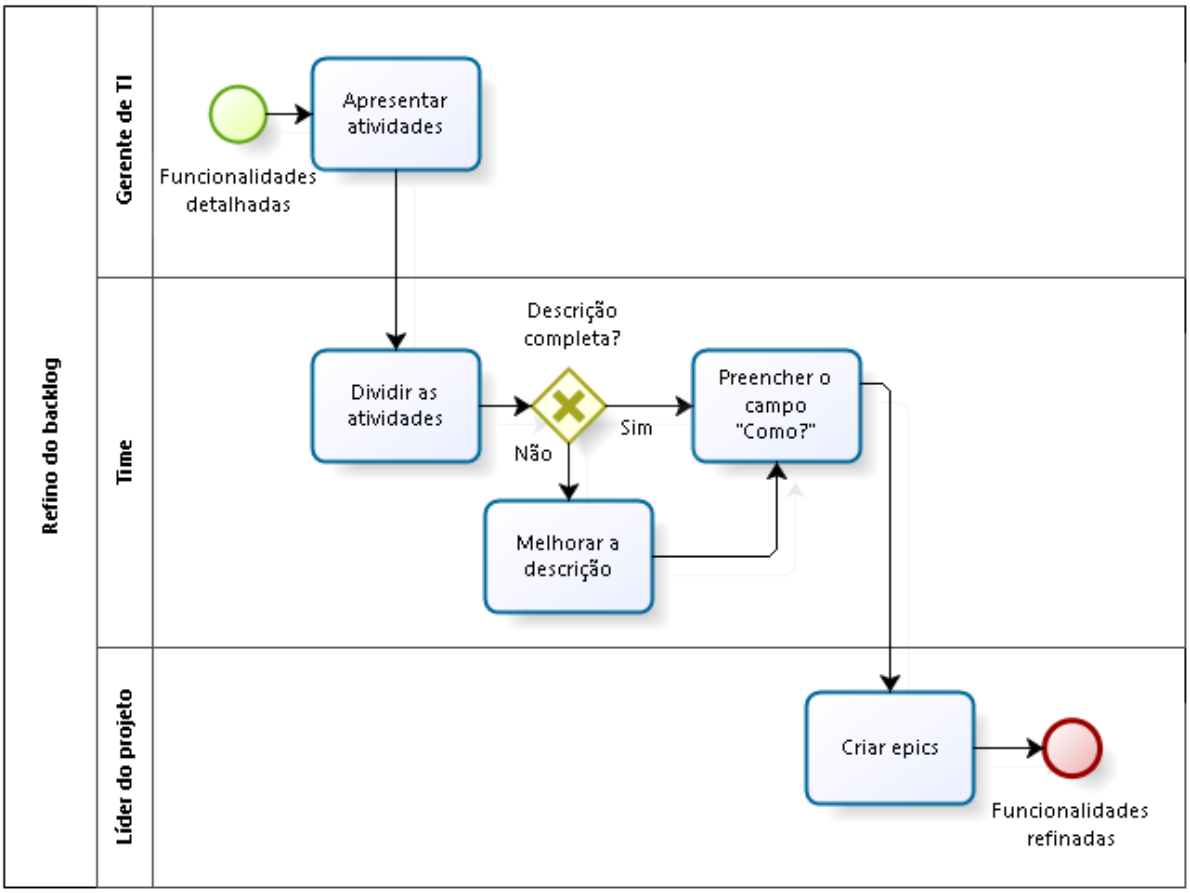
O processo de planejamento foi alterado para atender as melhorias propostas na seção 4.3.1.

Todos os colaboradores do projeto serão incluídos na reunião para refinar o *backlog*, sendo que no início da mesma, o gerente de TI apresenta as atividades previstas para entrar na *sprint*. Com a padronização da descrição das funcionalidades, a responsabilidade dessa tarefa será dividida entre o gerente de TI e o time. Sendo assim, o gerente de TI preenche na ferramenta Jira os campos destinados ao “o que?”, “quem?” e “por que?” e o time detalha o “como”. Dessa forma, no refino do *backlog* devem ser realizas as seguintes tarefas:

- Dividir as atividades nas menores partes possíveis que possibilitem a implementação e o teste;
- Melhorar a descrição das atividades quando necessário;
- Esclarecer as dúvidas;
- Preencher o campo “Como?” Descrevendo a solução técnica que será utilizada e os elementos (telas, botões, campos, etc.) que serão criados durante a implementação;
- Preencher o campo “Como?”, descrevendo também o modo que a atividade será testada (tipos de teste, critérios de aceitação, etc.).
- Criar *Epics* para agrupar as atividades.

A figura 24 apresenta as tarefas realizas no refino da *backlog*.

Figura 24 - Refino do backlog



Fonte: Elaboração da autora, 2016.

Além disso, os colaboradores do projeto também terão autonomia para realizarem a reunião de *planning*, a qual tem como objetivo estimar o esforço de desenvolvimento e de teste. Para isso o time deve-se utilizar a tabela de suporte ao *planning*, ilustrada na figura 20 da seção 4.3.1. Logo, devem ser realizadas as seguintes tarefas no *planning*:

- Calcular os pontos que devem entrar na *sprint* com o suporte da tabela, levando em consideração os feriados, férias, folgas e outros modificadores;
- Revisar as atividades no topo do *backlog* para que todos na equipe saibam o que deve ser implementado. Todas as dúvidas existentes devem ser sanadas;
- Melhorar a descrição das atividades quando necessário;
- Criar novas atividades, quando for necessário;
- Pontuar as atividades considerando o esforço necessário, grau de complexidade e risco de impacto. Garantir que as atividades estejam divididas o máximo possível, sem

atrapalhar a implementação ou os testes. O ideal é que nenhuma atividade tenha pontuação maior do que oito;

• Criar uma Sprint no Jira;

• Mover para a *sprint* as atividades que some o total de pontos equivalente a capacidade do time;

• Confirmar e Iniciar execução das Epics que estiverem na sprint;

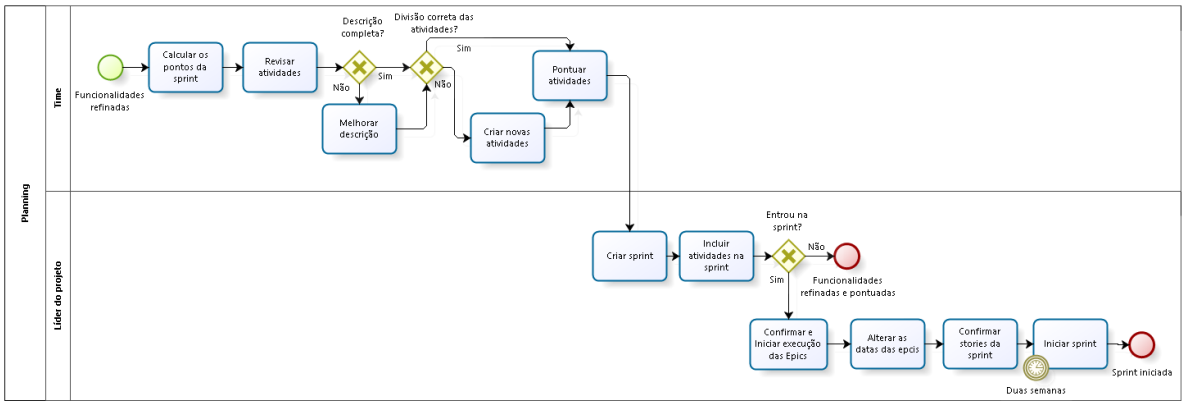
•Confirmar stories da sprint;

• Alterar as datas das *epcis* incluídas na *sprint*. O valor de "Original Due Date" e **"**Due Date" na ferramenta Jira deve receber como valor o dia do Deploy planejado para a funcionalidade, que deve ser o dia 1 da próxima *sprint*;

• Iniciar a *sprint* com as datas corretas, iniciando no dia 1 e terminando no dia 10.

A figura 25 apresenta as tarefas realizas no planning.

Figura 25 - Planning



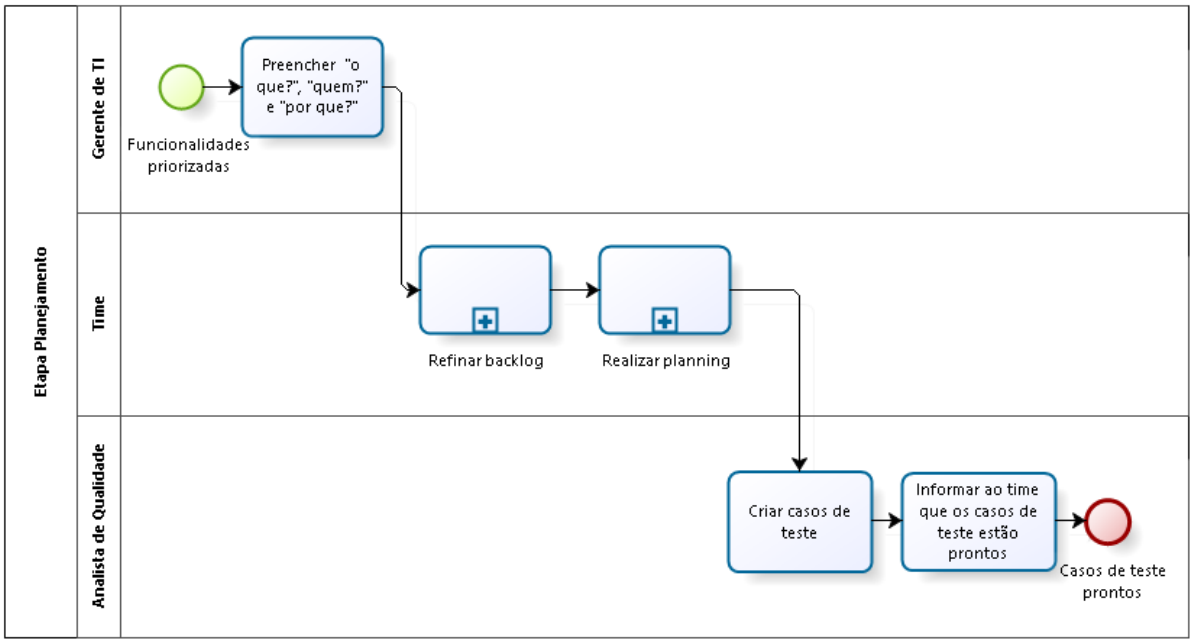
Fonte: Elaboração da autora, 2016.

Após a realização das reuniões de refino de *backlog* e *planning*, cabe ao analista de qualidade do projeto escrever os casos de teste com o auxílio da ferramenta TestRail. Para isso, o mesmo revisa as atividades da *sprint* e detalha todos os cenários possíveis, informando a referência de cada *storie* e *epic* no caso de teste. Além disso, o time deve ser aviso pelo mensageiro HipChat que os casos de testes estão prontos.

O último ponto é a criação de uma agenda contendo todos os eventos dos principais projetos, a qual pode ser impressa e exposta no setor de TI da empresa do estudo de caso. PP LTDA.

A figura 26 ilustra o novo processo de planejamento contendo as melhorias propostas.

Figura 26 - Novo processo de planejamento



Fonte: Elaboração da autora, 2016.

Após essa etapa, é possível iniciar a implementação da funcionalidade, a qual seja descrita na próxima seção. A mesma tem como critério de entrada, o início da *sprint*, e o critério de saída à funcionalidade implementada.

4.4.2.2 Etapa implementação

O novo processo de implementação foi modelado para atender as melhorias propostas na seção 4.3.2.

Com a adoção da ferramenta TestRail para gerenciar os casos de teste, o analista de sistema deverá revisar os casos de teste antes de iniciar a implementação e também incluir novos casos encontrados durante os testes se necessário.

Além disso, a ferramenta Postman irá auxiliar na realização dos testes de serviço e também na automação. Sendo assim após implementar as funcionalidades, o analista de sistema deve adicionar uma nova pasta chamada novos testes na ferramenta e criar os testes de serviço. Feito isso, é necessário exportar o arquivo com os testes e anexar no caso para que posteriormente o analista de qualidade valide os testes de serviço e insira-os na automação.

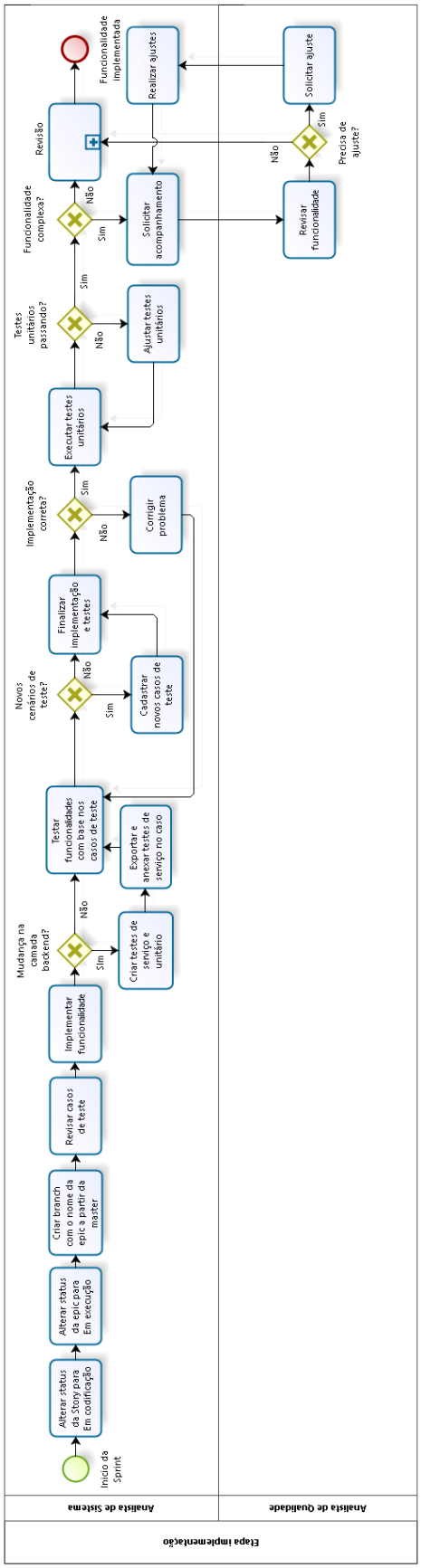
Outro ponto importante é solicitar ao analista de qualidade do projeto para que o mesmo acompanhe uma demonstração das atividades mais complexas antes de enviar para teste. A demonstração consiste em o analista de sistema mostrar a funcionalidade e ambos revisarem os casos de teste. Caso precise de alterações, o analista de sistema realiza as mesmas e verificar novamente com o analista de qualidade se há mais alterações, até que a demonstração seja aprovada.

Após finalizar a implementação, os testes e a demonstração, o analista de sistema deve enviar a *story* para *code review* e jogar o dado para verificar quem será o responsável. Desse modo, o analista de sistema escolhido, revisa o código fonte que está disponível no repositório do Bitbucket. Caso precise de alterações, o caso é retornando para o responsável pela implementação ajustar o que for necessário e posteriormente retorna para o *code review* até que não haja mais correções a serem aplicadas.

Por fim, quando ao analista de sistema encontrar algum problema e conseguir resolvê-lo, ele deve criar uma página no Confluence com “o que aprendi hoje?”, a qual será enviada semanalmente.

A figura 27 ilustra o novo processo de implementação contendo as melhorias propostas.

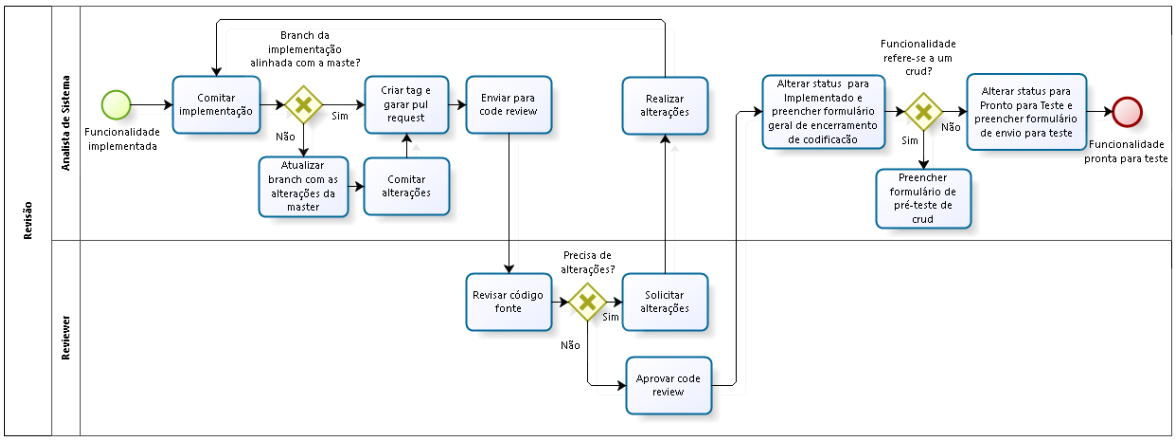
Figura 27 - Etapa implementação



Fonte: Elaboração da autora, 2016.

Por sua vez, a figura 28 representa o sub processo nomeada revisão, o qual foi criado no processo de implementação para dividir melhor as tarefas e facilitar a visualização.

Figura 28 - Revisão



Fonte: Elaboração da autora, 2016.

Após finalizar a terceira etapa do processo de desenvolvimento de *software* da empresa do estudo de caso, iniciaremos e etapa de testes de funcionalidades, a qual é realizada pelo analista de qualidade. Seu critério de entrada são funcionalidades prontas para teste e o de saída são as funcionalidades testadas.

4.4.2.3 Etapa teste de funcionalidade

O processo de teste de funcionalidades foi ajustado para atender as melhorias propostas na seção 4.3.3.

Desse modo, após os testes manuais da funcionalidade e a aprovação da implementação, o Analista de qualidade deve realizar os testes automatizados de serviço e interface com o auxílio das ferramentas Postman, Protractor e IntelliJ IDEA. Assim como, no tempo livre, deve ser realizada a automação dos testes de serviço já existentes na camada *backend* do sistema.

Além disso, também devem ser feitas reuniões semanais entre os analistas de qualidade de cada projeto para verificar se todos estão cumprindo o processo e se tem alguma dúvida ou sugestão.

Por sua vez, foi criada uma lista de problemas encontrados comumente na etapa de teste, representada no quadro 18.

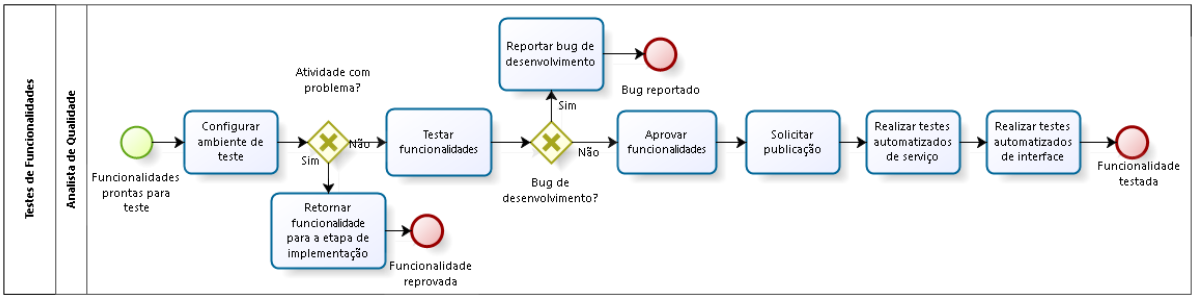
Quadro 18 - Problemas

Problema	**Descrição**
Caminho feliz	O cenário da funcionalidade básica implementada não está funcionando corretamente no sistema, uma vez que, os cenários estão especificados.
Campos obrigatórios	Os campos que deveriam ser obrigatórios não estão sinalizados ou a validação não ocorre no sistema.
Configuração Json	As configurações adicionadas/removidas não foram incluídas na tela de Encerrar codificação/correção no JIRA. Deve indicar o nome do arquivo ou as variáveis do java.
Dependência Nexus	As dependências necessárias não foram colocadas no repositório Nexus.
Erro de compilação	Problema no código fonte. Job do Jenkins não completou.
Ordenação dos itens	Os itens no sistema não estão ordenados como solicitado. Exemplo: data, ordem alfabética, etc.
Visibilidade dos dados	Os dados das listagens e dos combos não estão visíveis apenas para a empresa corrente no sistema.
Serviços adicionados	Há serviços que foram adicionados/modificados e não foram incluídos na tela de Encerrar codificação/correção no JIRA. Deve haver um exemplo de requisição.
SQL	O sql necessário para ser executado no banco de dados não consta no campo SQL da tela de Encerrar codificação/correção no Jira.
Tag de implementação	A tag de implementação não consta em um ou mais componentes do projeto (backend/frontend).
Testes unitários	Ao rodar a job do projeto no Jenkins, um ou mais testes unitários falham.
Código fonte desatualizado	O código fonte não está atualização em relação a master.

Fonte: Elaboração da autora, 2016.

A figura 29 ilustra o novo processo de publicação contendo as melhorias propostas.

Figura 29 - Nova etapa de teste de funcionalidade



Fonte: Elaboração da autora, 2016.

Na próxima seção é apresentada a etapa de publicação de funcionalidades, a qual é executada pelo líder do projeto com o auxílio do analista de qualidade. A mesma tem como critério de entrada as funcionalidades testadas e como saída as funcionalidades publicadas.

4.4.2.4 Etapa publicação de funcionalidades

O novo processo de publicação de funcionalidades foi modelado para atender as melhorias propostas na seção 4.3.4.

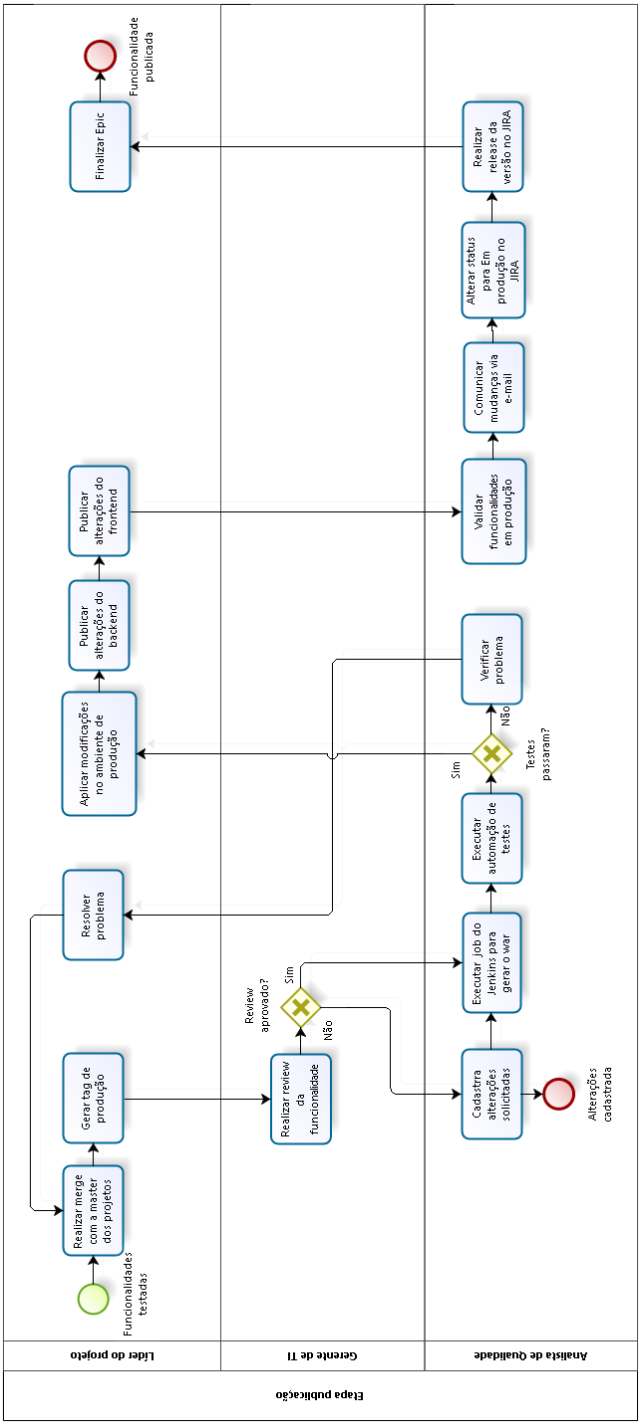
Sendo assim, após o líder do projeto gerar a *tag* de produção contendo uma ou mais *epics* testadas, o Analista de qualidade cria o ambiente de testes com a versão que deve ser aprovada e realiza a reunião de *review* com o Gerente de TI seguindo os passos a seguir:

- Apresentar as novas funcionalidades para o Gerente de TI;
- Gerente de TI deve aprovar a publicação da versão ou não;
- Cadastrar no JIRA as solicitações feitas pelo Gerente de TI.

Além disso, antes de aplicar as modificações em produção, os testes automatizados de serviço e interface devem ser executados para validar se as funcionalidades antigas continuam funcionando. Caso algum teste falhe, o Analista de qualidade deve verificar o problema para que o líder do projeto resolva o mesmo.

A figura 30 ilustra o novo processo de publicação contendo as melhorias propostas.

Figura 30 – Nova etapa de publicação



Fonte: Elaboração da autora, 2016.

Por sua vez, a próxima seção, nomeada de encerramento, contemplará as propostas de melhorias propostas na seção de 4.3.4 a partir do item 27.

4.4.2.5 Etapa encerramento

Após publicar as novas funcionalidades no ambiente de produção, o Analista de qualidade de cada projeto deve semanalmente criar um blog para explicar melhor as atividades que foram disponibilizadas. Além disso, no final da primeira semana da *sprint*, o Gerente de TI e o líder do projeto devem realizar a reunião de acompanhamento seguindo os passos abaixo:

- Apresentar para o Gerente de TI o andamento da *sprint*, apontando quais atividades está no prazo e quais não estão;
- Verificar se a *sprint* será entregue no prazo;
- Se houver algum problema, criar plano de ação para solucionar.

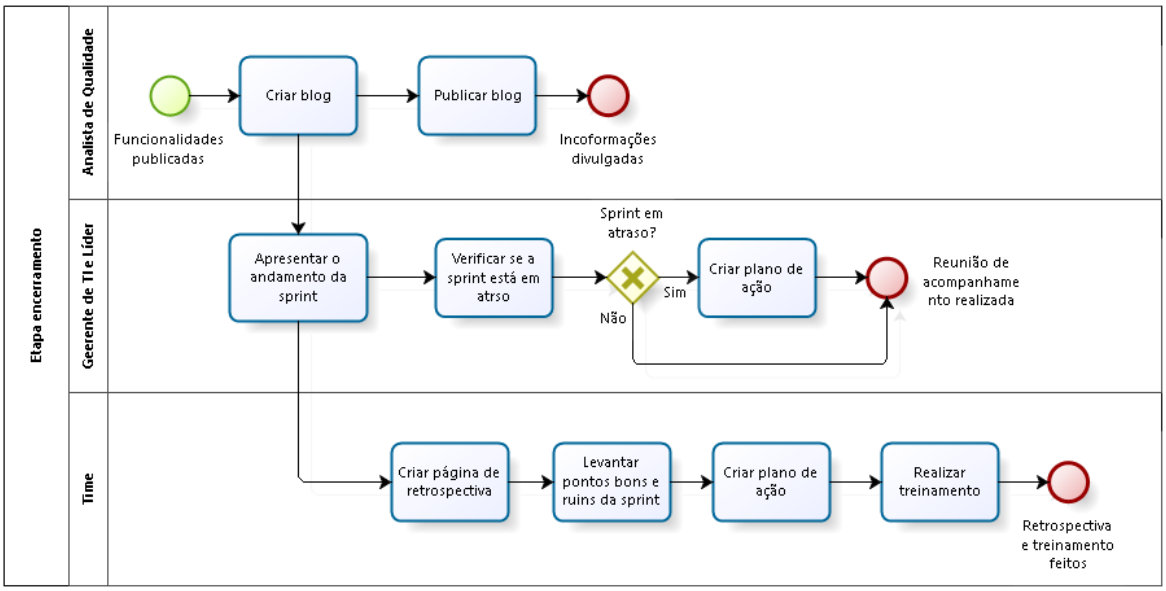
Além disso, no final de cada *sprint*, deve ser feita a reunião de retrospectiva com todo o time. A mesma deve ter como entrada, a retrospectiva da *sprint* anterior e também as anotações feitas durante a *sprint*. Desse modo, são feitos os seguintes passos:

- Criar uma página de retrospectiva no Confluence do projeto com o nome da *sprint*;
- Levantar os pontos bons e ruins da *sprint*;
- Levantar os pontos ruins da *sprint* anterior;
- Criar plano de ação para a *sprint* seguinte tentando resolver os problemas encontrados.

Por fim, deve ser criado um calendário com todos os membros do time para a realização do treinamento. O mesmo ocorre após o término de cada *sprint*, com o objetivo de apresentar as novas funcionalidades do sistema ao setor de operação (pós-venda) e sanar as dúvidas.

A figura 31 representa o novo processo de verificação de *bugs* contendo as melhorias propostas.

Figura 31 - Novo processo de encerramento



Fonte: Elaboração da autora, 2016.

Após a modelagem das etapas do ciclo de vida de gestão de funcionalidades, contemplando as melhorias propostas, também será modelado a seguir, as etapas do ciclo de vida de gestão de *bugs* de produção.

4.4.3 Ciclo de vida de gestão de bugs de produção

A seguir serão apresentadas as melhorias propostas para cada etapa do ciclo de vida de gestão de bugs de produção.

4.4.3.1 Etapa verificação de bug

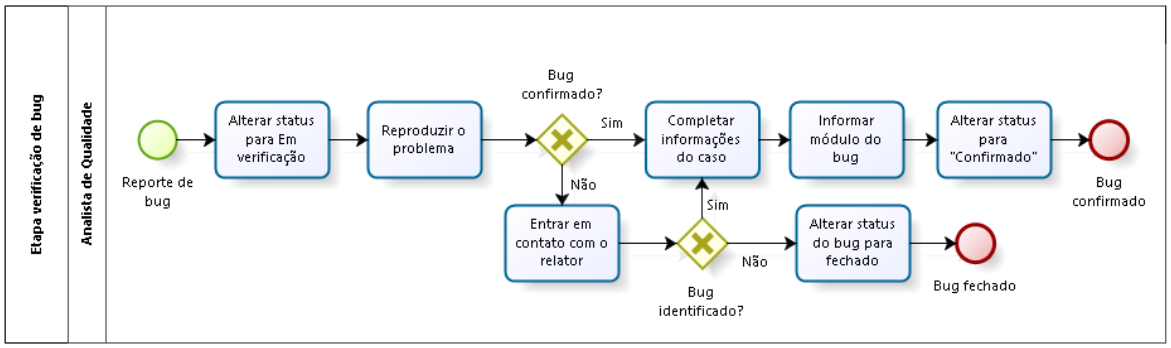
O processo de verificação de bugs foi ajustado para atender as melhorias propostas na seção 4.3.5.

Ao completar as informações do bug, o Analista de qualidade deve informar no Jira, o módulo em que o problema foi e encontrado e a operação (inserção, edição, listagem e exclusão). Esses dados servirão para que, no final do mês seja possível gerar um relatório

contendo os módulos afetados pelos bugs e assim podem direcionar os esforços da automação de testes.

A figura 32 representa o novo processo de verificação de *bugs* contendo as melhorias propostas.

Figura 32 - Novo processo de verificação de bug



Fonte: Elaboração da autora, 2016.

Na próxima seção, é detalhada a etapa de correção de *bug*, a qual é realizada pelos analistas de sistema. Por sua vez, o critério de entrada é o *bug* confirmado e o de saída, o *bug* pronto para teste.

4.4.3.2 Etapa correção

O processo de correção de bugs foi ajustado para atender as melhorias propostas na seção 4.3.6.

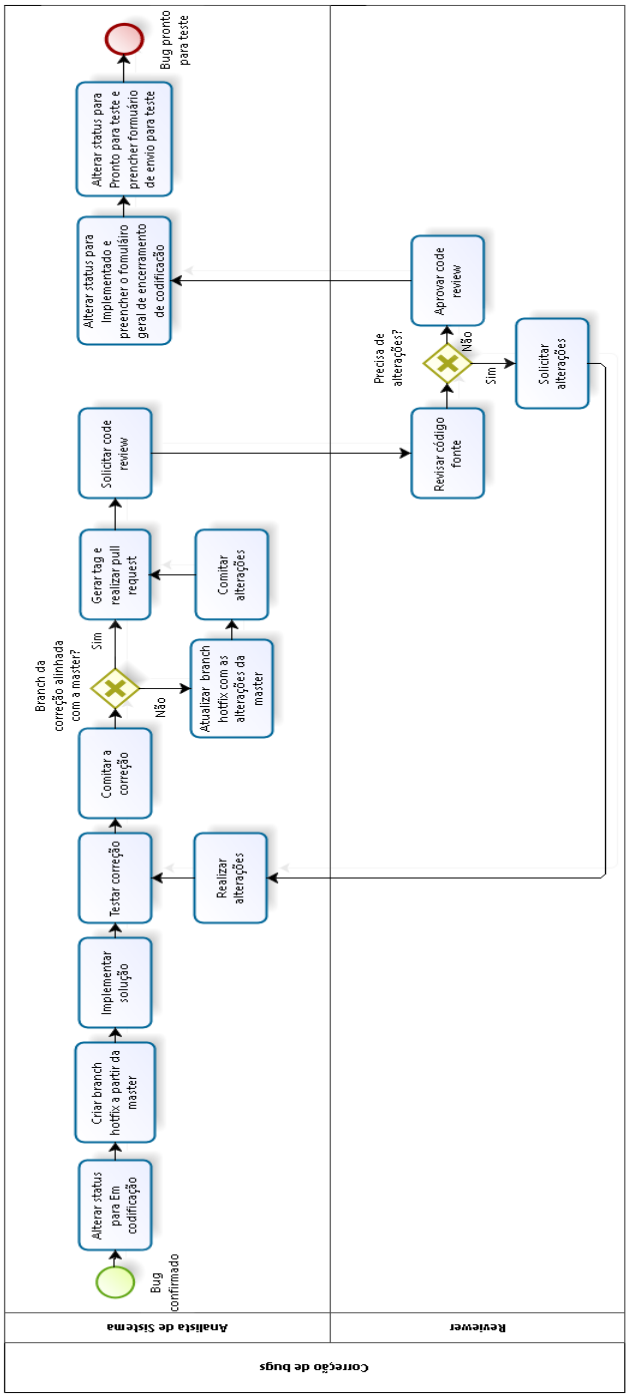
Após gerar a *tag* e realizar o *pull request*, o Analista de Sistema, deve sortear no dado um número, o qual corresponde a um dos Analistas de Sistema do time, e assim é solicitado o *code review*.

Por sua vez, o *reviewer* revisa o código fonte para verificar se a solução encontrada para o problema é a que melhor se aplica. Se estiver tudo certo, o *code review* é aprovada, e o Analista de Sistema, finaliza a implementação e preenche o formulário de geral de encerramento de codificação. No entanto, se forem necessárias alterações, o Analista de sistema realiza a mesma, e inicia o processo novamente a partir da tarefa de testar correção.

Além disso, se houverem muitos bugs durante a *sprint*, para não atrasar o término da mesma ou da próxima, os outros Analistas de Sistema auxiliam o *firefigther* na resolução dos bugs encontrados no ambiente de produção.

A figura 33 representa o novo processo de correção de bugs contanto as melhorias propostas.

Figura 33 - Novo processo de correção de bug



Fonte: Elaboração da autora, 2016.

Após finalizar essa etapa, iniciaremos e etapa de teste de *bug*, a qual é realizada pelo analista de qualidade. Seu critério de entrada são os *bugs* prontos para teste e o de saída são os *bugs* testados ou retornados para correção.

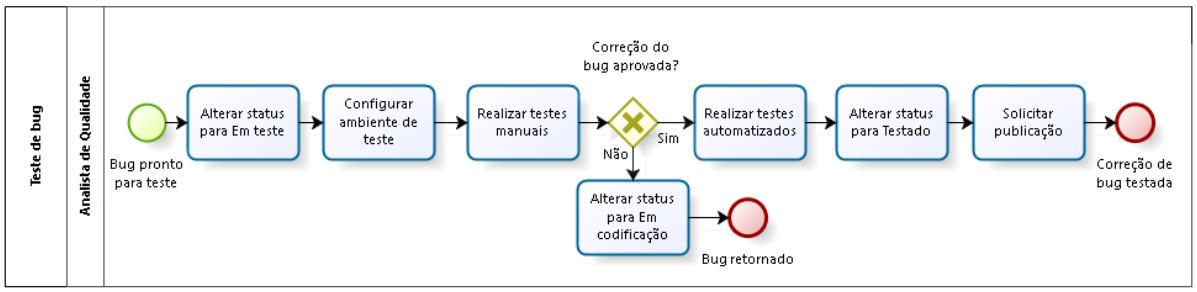
4.4.3.3 Etapa teste de bug

O novo processo de teste de *bugs* foi modelado para atender as melhorias propostas na seção 4.3.7.

Após todos os testes manuais serem realizados com sucesso pelo Analista de Qualidade, aprovando assim a correção do bug, devem ser feitos os testes automatizados, com o objetivo de garantir que o mesmo problema não vai se repetir.

A figura 34 ilustra o novo processo de testes de bugs contendo a proposta de melhoria.

Figura 34 - Novo processo de teste de bugs



Fonte: Elaboração da autora, 2016.

Na próxima seção é apresentada a etapa de publicação de *bug*, a qual é executada pelo líder do projeto com o auxílio do analista de qualidade. A mesma tem como critério de entrada os *bugs* testados e a saída é a disponibilização das correções no ambiente de produção.

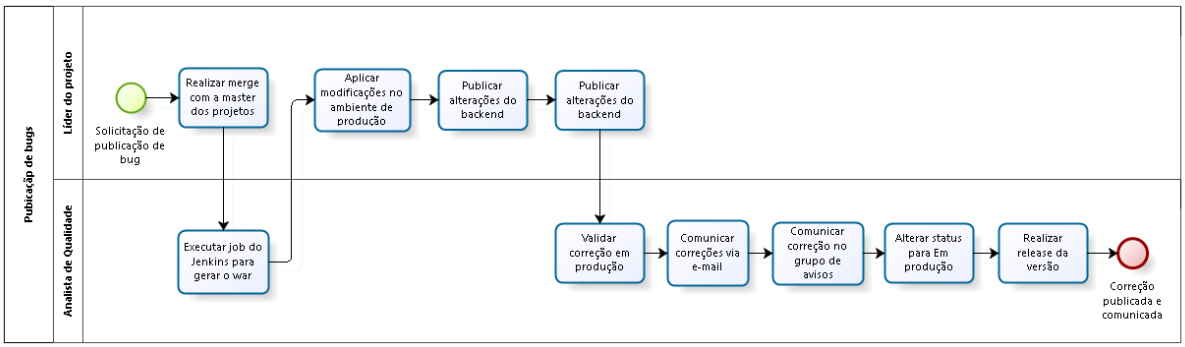
4.4.3.4 Etapa publicação de bug

O novo processo de publicação de *bugs* foi modelado para atender as melhorias propostas na seção 4.3.8.

Após o envio do e-mail referente à correção do *bug*, o analista de qualidade também deve realizar a comunicação no grupo referente a avisos criado no mensageiro HipChat.

A figura 35 ilustra o novo processo de publicação de *bug* com a melhoria proposta.

Figura 35 - Novo processo de publicação de bugs



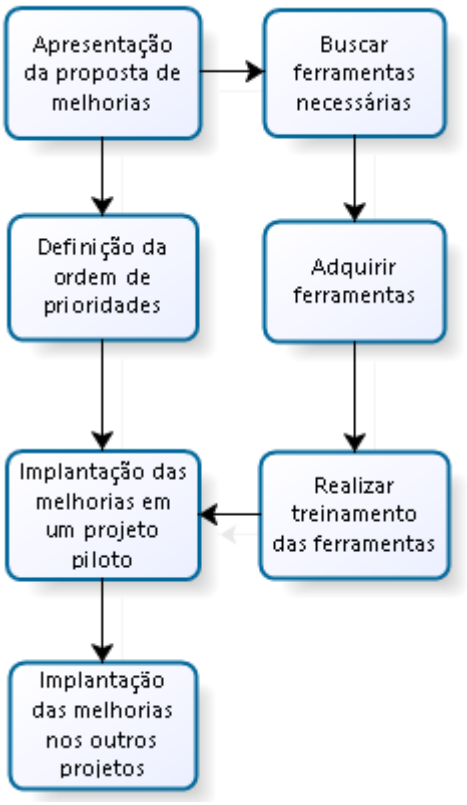
Fonte: Elaboração da autora, 2016.

Após a modelagem do novo processo de desenvolvimento de *software* para a empresa do estudo de caso, é necessário realizar a implantação das melhorias propostas. Sendo assim, essa fase é descrita na próxima seção.

4.5 IMPLANTAÇÃO DAS PROPOSTAS DE MELHORIA

Nessa seção é apresentada a implantação das melhorias propostas para o processo de desenvolvimento de *software*, na empresa PP LTDA. Para a melhor explicar a execução dessa etapa, foi criado um roteiro de implantação, sendo este apresentado na figura 36.

Figura 36 - Roteiro de implantação



Fonte: Elaboração da autora, 2016.

Nas próximas seções são detalhadas as etapas apresentadas na figura 35, roteiro de implementação, com o intuito de melhorar o entendimento.

4.5.1 Apresentação da proposta de melhorias

Inicialmente foi elaborada uma apresentação para o gerente de TI da empresa do estudo de caso, contento as melhorias propostas na seção 4.3 dessa monografia com o objetivo de explicar os benefícios que seriam obtidos ao implantar as mesmas, assim como, os colaboradores do setor de TI que seriam afetados com cada uma delas.

Após a apresentação que ocorreu no início do mês de maio de 2016, o gerente de TI respondeu de forma positiva e aprovou a implantação das melhorias no setor de tecnologia de informação. Por sua vez, o gerente de TI iniciou a elaboração da melhoria proposta de número 9, agenda da *sprint* para cada projeto e os analistas de qualidade também adotaram a

melhoria de número 25, realizar reuniões semanais com os analistas de qualidade, logo após a apresentação.

Além disso, a analista de qualidade dessa monografia iniciou a busca pelas ferramentas mais adequadas para auxiliar na implantação das melhorias. Após a escolha, a empresa adquiriu as ferramentas apresentadas na seção 4.4.1 desse trabalho e foram elaborados treinamentos para as ferramentas. Por conseguinte, foi necessário definir a ordem de implantação, a qual será apresentada na próxima seção.

4.5.2 Definição da ordem de prioridades

Com objetivo de orientar a implantação das melhorias, as mesmas foram inseridas em um quadro e, colocadas em ordem de implantação. A experiência da autora e do gerente de TI sobre o processo e a percepção sobre os benefícios foi considerada para definir a ordem de implantação no caso das melhorias. O quadro 19 apresenta cada melhoria, sua respectiva ordem de implantação, a etapa de processo de desenvolvimento de *software* e o projeto no qual as melhorias foram implantadas.

Quadro 19 - Ordem de implantação das melhorias

Nº	Prioridade	Etapa	Descrição	Projeto
1	Alta	Planejamento	Incluir todos os participantes do projeto no refino do *backlog*.	Todos
5	Alta	Planejamento	Dar autonomia para os participantes do projeto realizarem o *planning*.	Todos
8	Alta	Planejamento	Criar casos de teste para as atividades da *sprint* com o auxílio de um *software*.	Todos
14	Alta	Implementação	Realizar prática de revisão do código (*code review*).	Todos
17	Alta	Teste de funcionalidade	Automatizar novas funcionalidades implementadas.	Todos
18	Alta	Teste de funcionalidade	Automatizar serviços já existes no sistema com o auxílio de uma ferramenta.	Todos
19	Alta	Teste de funcionalidade	Automatizar e executar testes de regressão.	Todos
23	Alta	Teste de funcionalidade	Automatizar testes em dispositivos móveis com o auxílio de uma ferramenta	Caderno de Campo e Clicq

25	Alta	Teste de funcionalidade	Realizar reuniões semanais com os analistas de qualidade.	Todos
27	Alta	Publicação de funcionalidade	Realizar *review* antes de publicar a versão.	Todos
28	Alta	Publicação de funcionalidade	Executar automação de testes antes de publicar a versão	Todos
31	Alta	Publicação de funcionalidade	Incluir o evento de retrospectiva do *scrum*.	Todos
35	Alta	Correção	Realizar prática de revisão do código (*code review*).	Todos
37	Alta	Teste de bug	Automatizar bugs corrigidos	Todos
2	Média	Planejamento	Padronizar a descrição das funcionalidades.	Todos
3	Média	Planejamento	Dividir as responsabilidades da descrição da funcionalidade.	Todos
4	Média	Planejamento	Dividir as atividades nas menores partes possíveis. A pontuação da *story* não deve ultrapassar oito.	Todos
7	Média	Planejamento	Considerar esforço necessário, grau de complexidade e risco de impacto ao pontuar as *stories*.	Todos
10	Média	Implementação	Revisar casos de teste antes de iniciar a implementação.	Todos
11	Média	Implementação	Incluir novos casos de teste encontrados durante a implementação.	Todos
12	Média	Implementação	Realizar testes de serviço com o auxílio de um *software*.	Todos
13	Média	Implementação	Executar testes automatizados de serviços existentes antes de mandar para teste, quando a mudança for na cama *backend*.	Todos
24	Média	Teste de funcionalidade	Criar lista de boas práticas da ferramenta utilizada para testes de interface	Todos
26	Média	Teste de funcionalidade	Listar problemas básicos que são encontrados comumente na implementação.	Todos
30	Média	Publicação de funcionalidade	Fornecer treinamento para o setor de Operação (pós-venda) da empresa	Todos
32	Média	Publicação de funcionalidade	Realizar reunião de Acompanhamento no final da primeira semana de *sprint*.	Todos
33	Média	Verificação de bug	Identificar módulos do sistema com maior ocorrência de bugs de produção.	Todos
34	Média	Verificação de bug	Criar relatório com módulos do sistema com mais ocorrência de bugs	Todos
36	Média	Correção	Colaboração na correção de *bugs* caso tenha muitos bugs de produção acumulados.	Todos
6	Baixa	Planejamento	Criar tabela de suporte ao *planning*.	Todos
9	Baixa	Planejamento	Criar agenda da *sprint* para cada projeto.	Todos
15	Baixa	Implementação	Solicitar ao analista de qualidade do projeto que acompanhe uma demonstração das	Todos

			atividades mais complexas antes de enviar para teste.	
16	Baixa	Implementação	Criar página com “o que aprendi hoje? ” e enviar semanalmente.	Todos
20	Baixa	Teste de funcionalidade	Padronizar e documentar testes de configuração para *web*.	Todos
21	Baixa	Teste de funcionalidade	Padronizar e documentar testes de configuração para dispositivos móveis.	Caderno de Campo e Clicq
22	Baixa	Teste de funcionalidade	Padronizar e documentas testes de configuração para *desktop*.	Rastreador
29	Baixa	Publicação de funcionalidade	Criar um blog para apresentar as funcionalidades que foram disponíveis em produção na semana.	Todos
38	Baixa	Publicação de bug	Criar grupo de avisos	Todos

Fonte: Elaboração da autora, 2016.

Após definir a ordem da implementação das melhorias conforme a prioridade é necessário eleger um projeto piloto para colocá-la em prática. Dessa forma, essa etapa será apresentada na próxima seção.

4.5.3 Implantação das melhorias em um projeto piloto

Para implantar as melhorias propostas no processo de desenvolvimento de *software* da empresa, foi escolhido o projeto Rastreador, por seus integrantes serem mais experientes e adeptos a mudanças. Dessa forma, a mesma apresentação elaborada para o gerente de TI foi mostrada para os membros do projeto no final do mês de maio de 2016, assim como a ordem de prioridades para a implantação.

Assim, as melhorias com a prioridade alta foram implantadas durante o período de duas *sprints*, neste caso, em todo o mês de junho de 2016. Posteriormente, mais duas s*prints* foram disponibilizadas para implantar as melhorias propostas com prioridade média, neste caso, todo o mês de julho e por fim, a implantação das melhorias com prioridade baixa ocupou metade do mês de agosto de 2016, no caso uma *sprint*. O quadro abaixo apresenta o período para implantação das melhorias no Rastreador.

Quadro 20 - Período para implantação das melhorias no Rastreador

Projeto	Período	Prioridade
Rastreador	Junho de 2016	Alta
	Julho de 2016	Média
	Agosto de 2016	Baixa

Fonte: Elaboração da autora, 2016.

Dessa forma, o projeto Rastreador implantou todas as melhorias propostas para o processo de desenvolvimento de software, exceto as de número 21 e 25, uma vez que o sistema não possui versão para dispositivos móveis.

Após a implantação das melhorias no projeto piloto, é necessário implantar as melhorias para os demais projetos, neste caso, Caderno de Campo, Clicq e Panorama. Esta etapa será descrita na próxima seção.

4.5.4 Implantação das melhorias nos outros projetos

Após a implantação das melhorias de cada tipo de prioridade, alta, média e baixa, no projeto do Rastreador, é necessário inicia-la também nos outros projetos para que todos os times obtenham os benefícios o quanto antes. Sendo assim, após a implantação das melhorias com prioridade alta no Rastreador, é realizada uma apresentação para todos os colaboradores dos outros projetos contendo as melhorias com prioridade alta para que as mesmas sejam implantadas. O mesmo se repete para as melhorias com prioridade média e baixa.

Por sua vez, os outros projetos não implantaram a melhoria de número 22, pois a mesma destina-se apenas ao projeto do Rastreador. Além disso, as melhorias de número 21 e 23 foram implantadas somente nos projetos Caderno de Campo e Clicq.

O quadro 21 apresenta o período para implantação das melhorias nos outros projetos.

Quadro 21 - Período para implantação das melhorias nos outros projetos

Projeto	Período	Projeto	Período	Prioridade
Rastreador	Junho de 2016	Caderno de Campo, Clicq e Panorama	Julho de 2016	Alta
	Julho de 2016		Agosto de 2016	Média
	Agosto de 2016		Setembro de 2016	Baixa

Fonte: Elaboração da autora, 2016.

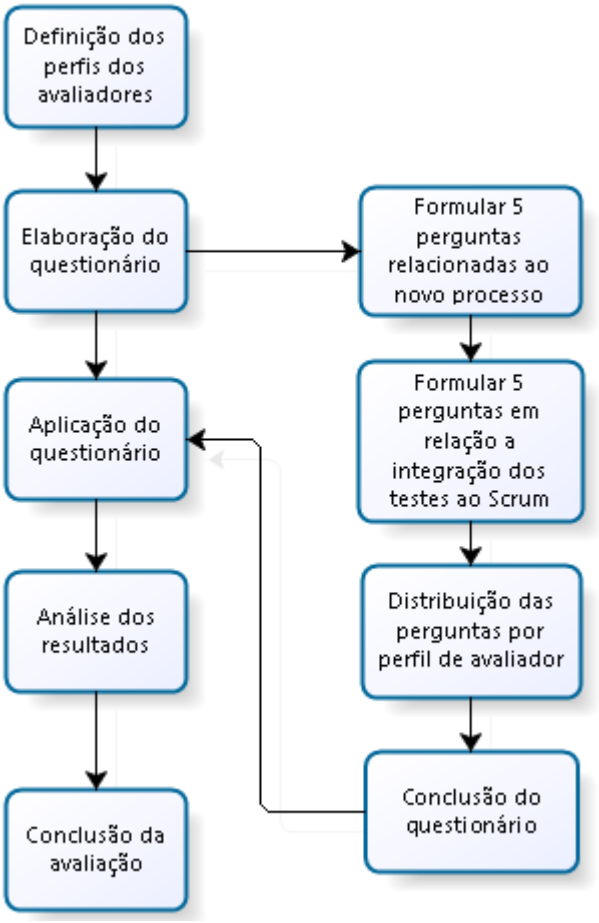
Por fim, é necessário salientar que a implantação das melhorias propostas nos projetos Caderno de Campo, Clicq e Panorama não foram acompanhados diretamente pela analista de qualidade desta monografia, uma vez que, a mesma participou somente da apresentação das mesmas.

4.6 AVALIAÇÃO DO NOVO PROCESSO DE DESENVOVIMENTO DE SOFTWARE

Nessa seção é apresentada a avaliação do novo processo de desenvolvimento de *software*, contento as melhorias propostas. A mesma consiste na aplicação de um questionário aos colaboradores do setor de tecnologia da informação, da empresa do estudo de caso, com o objetivo de verificar se as mudanças trazem benefícios para as atividades realizadas diariamente.

Após a aplicação de questionário proposto, será possível obter resultados qualitativos em relação ao novo processo. Para a elaboração do mesmo, foi criado um roteiro de avaliação, sendo as etapas apresentadas na figura 37.

Figura 37 - Roteiro de avaliação



Fonte: Elaboração da autora, 2016.

Nas próximas seções são detalhadas as etapas apresentadas na figura 36, roteiro de avaliação, com o intuito de melhorar o entendimento.

4.6.1 Definição dos perfis dos avaliadores

O departamento de tecnologia da informação da empresa PP LTDA, está dividido em quatro perfis. Os mesmos são: gerente de ti, analista de sistema, arquiteto de *software* e analista de qualidade. Visto que, o primeiro perfil citado possui uma relação hierárquica sob os demais.

Por sua vez, o analista de qualidade, tem como missão analisar o funcionamento dos sistemas, controlando e monitorando a gestão de defeitos, auxiliando no processo de

desenvolvimento através do planejamento, organização e controle das atividades referente qualidade de *software*, a fim de garantir o adequado funcionamento do produto.

O analista de sistema analisa, desenvolve e programa *softwares* e requisitos de tecnologia da informação, organizando as demandas recebidas, participando do processo de planejamento, seguindo as diretrizes estabelecidas pela empresa, para a entrega com qualidade dos serviços prestados e satisfação dos clientes.

Alguns analistas de sistema exercem o papel de líder do projeto. Dessa forma, além das atribuições do cargo, o mesmo também tem como funções: gerenciar os membros do projeto, facilitar as atividades, zelar pela manutenção da qualidade do *software* produzido e ser um canal de comunicação entre a gerência e a equipe.

Por fim, o gerente de TI, gerencia o setor de tecnologia de informação da empresa, prestando o suporte necessário para as pessoas e as atividades. Fazendo assim, conexões entre os sistemas desenvolvidos e a estratégia do negócio.

Todos os perfis detalhados anteriormente foram selecionados para responder a avaliação, exceto o do arquiteto de *software*, pois as melhorias propostas não impactam diretamente nas suas atividades.

4.6.2 Elaboração do questionário

A elaboração do questionário é feita com base nas melhorias propostas para o novo processo de desenvolvimento de *software*, apresentadas na seção 4.3 deste trabalho.

Para o desenvolvimento desta etapa, foi necessária a divisão em sub-estapas, sendo estas:

- Análise das melhorias propostas;
- Formular cinco perguntas relacionadas ao novo processo;
- Formular cinco perguntas em relação à integração dos testes ao *Scrum*;
- Distribuição das perguntas por perfil de avaliador;
- Conclusão do questionário.

Inicialmente foram analisadas as melhorias propostas para o processo de desenvolvimento de *software*, da empresa do estudo de caso, selecionando as mais relevantes para o novo processo, assim como, as ações sugeridas para melhorar a integrações dos testes

ao *Scrum*. Posteriormente foram elaboradas 10 perguntas, sendo oito fechadas e duas abertas, para que os avaliadores pudessem expressar sua opinião, sugerindo melhorias para o novo processo e também avaliando se os testes estão melhores integrados ao *Scrum*.

Dessa forma, as elaborações das questões fechadas foram baseadas na escala de Likert, na qual, segundo (GIL, 2008), solicita-se a certo número de pessoas que manifestem sua concordância ou discordância em relação a cada um dos enunciados, segundo a graduação:

1. Discordo totalmente;
2. Discordo parcialmente;
3. Indiferente;
4. Concordo parcialmente;
5. Concordo totalmente.

Quadro 22 - Questionário de avaliação

Questionário de avaliação			
Nº	Perfil do avaliador	Pergunta	Tipo de pergunta
1.	Analista de Qualidade e Analista de Sistema	A elaboração dos casos de teste na etapa de planejamento e antes de iniciar o desenvolvimento, ajudou a identificar especificações de atividades incorretas e/ou incompletas e a diminuir o retrabalho do time?	Fechada
2.	Analista de Qualidade e Analista de Sistema	A inclusão da automação dos testes de serviço durante a etapa de implementação ajudou a melhorar a as validações realizadas na camada *backend* dos sistemas e a diminuir os bugs de desenvolvimento?	Fechada
3.	Analista de Qualidade, Analista de Sistema e Gerente de TI	A automação e execução dos testes de interface e serviço ajudaram a diminuir o número de *bugs* de produção nos sistemas?	Fechada
4.	Analista de Qualidade e Analista de Sistema	A lista de problemas ajudou a evitar erros básicos encontrados na implementação?	Fechada
5.	Analista de Qualidade, Analista de Sistema e Gerente de TI	Houve uma percepção de melhoria da integração dos testes ao Scrum no novo processo de desenvolvimento de *software*?	Aberta

		Justifique sua resposta.	
6.	Analista de Sistema	A prática do *code review* nas etapas de implementação e correção ajudaram a identificar falhas e possíveis melhorias no código?	Fechada
7.	Analista de Qualidade, Analista de Sistema e Gerente de TI	A inclusão de todos os participantes do projeto nos eventos *Scrum*: refino do *backlog*, *planning* e retrospectiva, melhoraram o entendimento das atividades e a execução das *Sprints*?	Fechada
8.	Analista de Qualidade e Gerente de TI	O *review* permitiu verificar se a atividade desenvolvida está de acordo com o esperado, antes dela está disponível em produção?	Fechada
9.	Analistas de Sistema (líder) e Gerente de TI	As reuniões de acompanhamento na metade da *Sprint* ajudaram a melhorar as entregas no prazo?	Fechada
10.	Analista de Qualidade, Analista de Sistema e Gerente de TI	O novo processo de desenvolvimento de software da empresa está adequado? Descreva seus pontos positivos e o que poderia ser melhorado.	Aberta

Fonte: Elaboração da autora, 2016.

Conforme apresentado no quadro 22, as perguntas do questionário de avaliação estão baseadas no novo processo de desenvolvimento de *software*, o qual contém as melhorias propostas na seção 4.3. Além disso, as perguntas foram distribuídas de acordo com o perfil dos avaliadores.

4.6.3 Aplicação do questionário

A aplicação do questionário de avaliação tem como o objetivo obter *feedbacks* dos colaboradores do setor de TI da empresa PP LTDA em relação a novo processo de desenvolvimento de *software* proposto. Dessa forma, é possível verificar se o processo apresentado atende as necessidades do setor e traz benefícios para a empresa. Logo, a aplicação do questionário foi realizada através das seguintes etapas:

a. Apresentação das melhorias propostas;

b. Apresentação da modelagem do novo processo;

c. Aplicação do questionário.

Primeiramente foi realiza uma apresentação das melhorias propostas para que os colaboradores pudessem relembrar as mesmas e, por conseguinte a apresentação do novo processo de desenvolvimento de *software* para todos os colabores do setor de TI da empresa, PP LTDA. Logo, a cada perfil de avaliador, foi aplicado um questionário correspondente, para que os mesmos pudessem expressar sua opinião em relação as melhorias propostas no processo.

Por sua vez, a ferramenta *Google Forms* foi à escolhida para aplicação da pesquisa e o armazenamento das respostas. Visto que, a mesma é gratuita, de fácil utilização e *online*.

4.6.4 Análise dos resultados

Nesta seção são apresentados os quadros com os resultados dos questionários aplicados na empresa do estudo de caso.

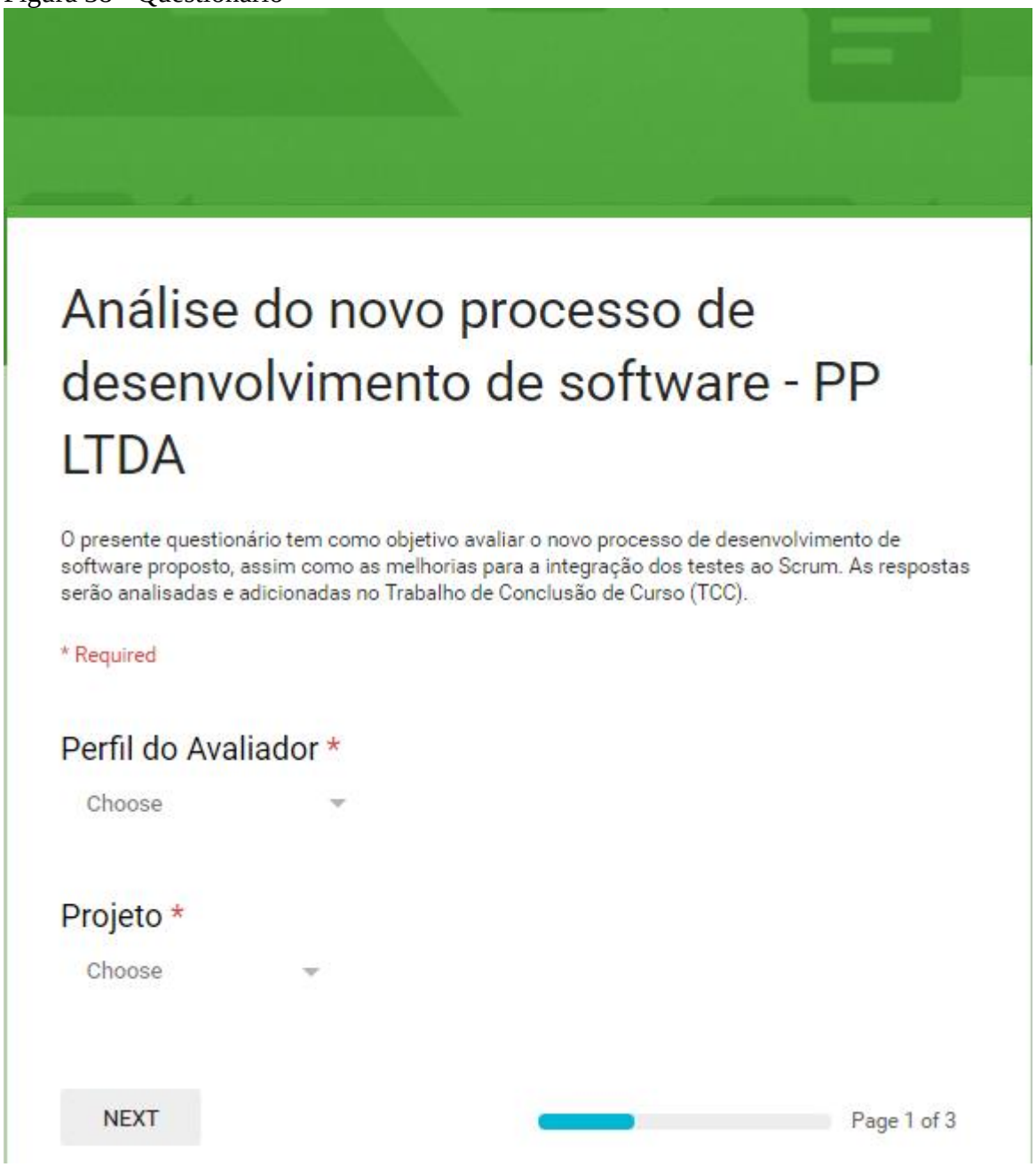
Tendo em vista os objetivos estabelecidos, está pesquisa pretende identificar se o novo processo de desenvolvimento de *software* atende as necessidades do setor de TI da empresa e se os testes estão mais integrados ao *Scrum*.

O resultado da validação foi baseado nas respostas de vinte pessoas. Após a apresentação do novo processo contento as melhorias foi disponibilizado de forma online o questionário direcionado para cada tipo de perfil de avaliadores: analista de qualidade, analista de sistema, líder e gerente de TI.

Para melhorar visualização das respostas do tipo fechada no quadro de avaliação, serão utilizadas as seguintes cores:

- Vermelho escuro: representa a resposta discordo totalmente (DT);
- Vermelho claro: representa a resposta discordo parcialmente (DP);
- Amarelo: representa a resposta indiferente (I);
- Verde claro: representa a resposta concordo parcialmente (CP);
- Verde escuro: representa a resposta concordo totalmente (CT).

Figura 38 - Questionário



Fonte: Elaboração da autora, 2016.

Na figura 39 é o questionário na visão do analista de qualidade e do analista de sistema. Visto que, no questionário do gerente de TI não é necessário selecionar o projeto. Sendo elaboradas duas perguntas abertas para relatar a sua opinião sobre o novo processo de desenvolvimento e a melhoria da integração dos testes ao *Scrum*. Os comentários serão utilizados para validação e aperfeiçoamento do processo.

A seguir são apresentadas as respostas dos entrevistados referentes ao processo.

Quadro 23 – Avaliação dos analistas de qualidade para as questões fechadas

Avaliador	**Projeto**	**P1**	**P2**	**P3**	**P4**	**P7**	**P8**
Analista de Qualidade 1	Caderno de Campo	CP	I	CP	DT	CT	CT
Analista de Qualidade 2	Clicq	CP	CT	CT	I	CP	CT
Analista de Qualidade 3	Clicq	CP	CT	CT	CT	CT	CT
Analista de Qualidade 4	Panorama	CT	CP	CP	CP	CT	CT

Fonte: Elaboração da autora, 2016.

Diante das quatro primeiras afirmações respondidas diretamente sobre a melhoria da integração dos testes ao *Scrum*, os analistas de qualidade dos projetos da empresa apresentam opiniões semelhantes em relação às perguntas um (1. A elaboração dos casos de teste na etapa de planejamento e antes de iniciar o desenvolvimento, ajudou a identificar especificações de atividades incorretas e/ou incompletas e a diminuir o retrabalho do time?) e três (3. A automação e execução dos testes de interface e serviço ajudaram a diminuir o número de bugs de produção nos sistemas?), contudo há discordâncias em relação as perguntas dois (2. A inclusão da automação dos testes de serviço durante a etapa de implementação ajudou a melhorar a as validações realizadas na camada backend dos sistemas e a diminuir os bugs de desenvolvimento?) e quatro (4. A lista de problemas ajudou a evitar erros básicos encontrados na implementação?).

A pergunta dois, obteve uma resposta indiferente, pois foi identificado que no projeto Caderno de Campo não estão sendo feitos testes de serviço durante a etapa de Implementação, visto que, os mesmos são realizados somente na etapa de Teste de funcionalidade. Além disso, a pergunta quatro recebeu uma resposta discordo totalmente e uma indiferente, pois a lista de problemas está ligada diretamente aos indicadores individuais do setor de TI da empresa PP LTDA. No entanto, na elaboração do questionário, a autora deste trabalho não deixou claro que a lista de problemas deveria ser considerada somente como um instrumento para lembrar sobre passos básicos do processo que devem ser executados na etapa implementação, o que permitiu dupla interpretação da questão.

Em relação às duas últimas afirmações, perguntas sete (7. A inclusão de todos os participantes do projeto nos eventos Scrum: refino do backlog, planning e retrospectiva, melhoraram o entendimento das atividades e a execução das Sprints?) e oito (8. O review permitiu verificar se a atividade desenvolvida está de acordo com o esperado, antes dela está disponível em produção?), respondidas sobre o novo processo, os analistas de qualidade apresentam opiniões semelhantes, conforme apresentado no quadro 23.

Por sua vez, as questões não foram respondidas pela analista de qualidade do projeto Rastreador, pois a mesma está redigindo esta monografia. Além disso, os comentários realizados pelos analistas de qualidade em relação a pergunta cinco, do tipo aberta, serão agrupados e apresentados a seguir.

Todos os analistas de qualidade que responderam ao questionário concordam que houve uma percepção de melhoria na integração dos testes ao *Scrum*. Os mesmos afirmam que, a disciplina do time foi favorecida pelas práticas dos testes de serviço e *interface*, uma vez que, a implantação dos testes de serviço pelo analista de qualidade ajuda a manter os testes atualizados.

Sobre os casos de teste, os analistas de qualidade comentaram que, ajudam a diminuir o índice de retrabalho, pois o analista de sistema tem a oportunidade de validar os testes que foram criados pelo analista de qualidade e também diminuiu a quantidade de problemas e *bug*s encontrados em desenvolvimento.

Para criar os casos de testes é necessário um bom entendimento das funcionalidades que estão entrando na *Sprint* e como elas serão desenvolvidas. Logo os analistas de qualidade explicam que é fundamental participar dos eventos do *Scrum*, o que gera uma maior percepção da inclusão do trabalho dos mesmos perante toda a equipe.

Por fim, atividades que demandam maior esforço são resolvidas com maior velocidade e a confiança nas entregas gera maior qualidade para o projeto. Também houve o aumento da percepção da participação dos analistas de qualidade na entrega do produto final.

Apresentado os devidos comentários dos analistas de qualidade referente à questão cinco, a seguir serão agrupadas as repostas da questão dez, do tipo aberta.

Todos os analistas de qualidade que responderam ao questionário concordam que o novo processo de desenvolvimento de *software* proposto está adequado. Os mesmos afirmar que o processo se enquadra para uma empresa que preza pela qualidade de *software*. O acompanhamento da gerência recebeu um gás com a formação de projetos independentes, mas que se ajudam dentro do necessário. A nomeação de um líder para os projetos reforçou o ambiente voltado a esse processo.

Em relação aos pontos positivos do processo, os analistas de qualidade destacaram as reuniões de retrospectiva, *review*, refino do *backlog* e *planning* com a participação de todos do time. Além disso, a atuação do analista de qualidade na organização dos casos também é um ponto positivo.

Os analistas de qualidade também comentaram como ponto positivo, a execução dos casos de testes pelos analistas de sistema antes de iniciar e de finalizar a codificação, pois assim é possível prevenir problemas em vez de somente resolvê-los.

Por fim, como ponto de melhoria, os analistas de qualidade, apontam o acompanhamento da execução do processo, pois alguns pontos estão escritos no processo, mas não são executados, e como não há acompanhamento do que é de fato executado, os pontos são simplesmente considerados como sem resultado, quando deveria ser investigado o porquê da não execução e o que ajustar para que seja executado de fato.

Quadro 24 – Avaliação dos analistas de sistema (líderes) para as questões fechadas

Avaliador	**Projeto**	**P1**	**P2**	**P3**	**P4**	**P6**	**P7**	**P9**
Líder 1	Clicq	CT	CP	I	CT	CT	CT	CT
Líder 2	Panorama	CP	I	I	CT	CT	CT	I
Líder 3	Rastreador	CT	CT	CP	CT	CT	CT	CP

Fonte: Elaboração da autora, 2016.

Diante das quatro primeiras afirmações respondidas diretamente sobre a melhoria da integração dos testes ao *Scrum*, os líderes dos projetos da empresa apresentam opiniões semelhantes em relação às perguntas um (1. A elaboração dos casos de teste na etapa de planejamento e antes de iniciar o desenvolvimento, ajudou a identificar especificações de atividades incorretas e/ou incompletas e a diminuir o retrabalho do time?) e quatro (4. A lista de problemas ajudou a evitar erros básicos encontrados na implementação?), contudo há discordâncias em relação às perguntas dois (2. A inclusão da automação dos testes de serviço durante a etapa de implementação ajudou a melhorar a as validações realizadas na camada backend dos sistemas e a diminuir os bugs de desenvolvimento?) e três (3. A automação e execução dos testes de interface e serviço ajudaram a diminuir o número de bugs de produção nos sistemas?).

As perguntas dois e três, obtiveram duas respostas indiferente para o projeto Panorama, pois foi identificado que não foi possível realizar os testes de serviço durante a etapa de Implementação e também não foram implementados os testes de interface. Além disso, a pergunta três obteve uma resposta indiferente do líder do projeto Clicq, pois apesar de os testes de serviço e interface das principais funcionalidades do sistema estarem implementados, os mesmos estão sendo executados a pouco tempo. Dessa forma, não é possível concluir se a automação ajudou a diminuir a quantidade de *bugs* de produção.

Em relação às três últimas afirmações respondidas sobre o novo processo, os analistas de sistema apresentam opiniões semelhantes para as perguntas seis (6. A prática do code review nas etapas de implementação e correção ajudaram a identificar falhas e possíveis melhorias no código?) e sete (7. A inclusão de todos os participantes do projeto nos eventos Scrum: refino do backlog, planning e retrospectiva, melhoraram o entendimento das atividades e a execução das Sprints?) e diferentes para a pergunta nove (9. O review permitiu verificar se a atividade desenvolvida está de acordo com o esperado, antes dela está disponível em produção?), conforme apresentado no quadro 24. Logo, a pergunta nove, obteve uma resposta indiferente por parte do líder do Panorama, pois a prática do review ainda não ocorre no projeto.

Por sua vez, o projeto Caderno de Campo não possui líder, atualmente quem está exercendo o papel é o gerente de TI. Além disso, os comentários realizados pelos líderes em relação a pergunta cinco, do tipo aberta, serão agrupados e apresentados a seguir.

Todos os líderes que responderam ao questionário concordam que houve uma percepção de melhoria na integração dos testes ao *Scrum*. Os mesmos afirmar que a criação dos casos de teste, a realização dos testes de serviço com o auxílio do Postman durante a implementação, os testes de interface e o acompanhamento do analista de qualidade nas atividades mais complexas ajudaram a revisar os requisitos do sistema de forma continua, evidenciando possíveis pontos não especificados ou não implementados corretamente. Além disso, as melhorias citadas reduzem o retrabalho durante o processo e garantem maior qualidade de *software*.

Apresentado os devidos comentários dos líderes referente a questão cinco, a seguir serão agrupadas as repostas da questão dez, do tipo aberta.

Todos os líderes que responderam ao questionário concordam que o novo processo de desenvolvimento de *software* proposto está adequado. Como pontos positivos, os mesmos destacam a entrega rápida e contínua das funcionalidades, especificação e estimativa colaborativas, casos de teste conhecidos desde o início da codificação e a prática do *code review*.

Em relação aos pontos a melhorar, os líderes citaram a especificação e priorização das funcionalidades. Dar mais espaço para débito técnico e automação de testes, também pelos desenvolvedores, preveniria erros conhecidos e prováveis gargalos, esforço demasiado, devido ao débito técnico daquele código nas implementações futuras e o versionamento com maior garantia de estabilidade nas novas versões.

Quadro 25 - Avaliação do gerente de TI para as questões fechadas

Avaliador	**P3**	**P7**	**P8**	**P9**
Gerente de TI	CP	CT	CT	CP

Fonte: Elaboração da autora, 2016.

Diante da primeira afirmação respondida diretamente sobre a melhoria da integração dos testes ao *Scrum*, o gerente de TI concorda parcialmente com a pergunta três. Em relação às três últimas afirmações respondidas sobre o novo processo, o gerente de TI, concorda totalmente com as perguntas sete e oito e concorda parcialmente com a pergunta nove, conforme apresentado no quadro 25. Além disso, os comentários realizados pelo gerente de TI em relação a pergunta cinco, do tipo aberta, serão apresentados a seguir.

O gerente de TI concorda que houve uma percepção de melhoria na integração dos testes ao *Scrum*. Este afirma que, com o novo processo foi possível diminuir o número de iterações entre a equipe de desenvolvimento e a área de qualidade, ajudou a elevar a velocidade de desenvolvimento e melhorou a qualidade dos projetos.

No que diz respeito a questão dez, do tipo aberta, o gerente de TI concorda que o novo processo está adequado. Ele explica que todos os integrantes da equipe estão incluídos nas decisões relacionadas ao projeto, existe uma cultura de melhoria contínua, retrospectivas das *sprints*, que ajuda no ajuste fino do processo, atacando os problemas do dia a dia.

Para ele os pontos a melhorar são a criação de um plano de médio prazo para o projeto, cronograma com as *features* que devem ser implementadas, deixando claro para a equipe, a definição da política de testes, inclusão de testes de carga, testes unitárias e de serviço durante o desenvolvimento.

Quadro 26 - Avaliação dos analistas de sistema para as questões fechadas

Avaliador	**Projeto**	**P1**	**P2**	**P3**	**P4**	**P6**	**P7**
Analista de Sistema 1	Caderno de Campo	CT	CT	I	I	I	CT
Analista de Sistema 2		CP	CT	CT	CP	I	CT
Analista de Sistema 3		CP	I	I	CP	I	CT
Analista de Sistema 4	Clicq	CP	CP	I	CP	CP	CT
Analista de Sistema 5		I	DP	I	CP	CT	CT
Analista de Sistema 6		CP	CT	CT	I	CT	CP
Analista de Sistema 7	Panorama	CP	I	I	DP	CT	CT
Analista de Sistema 8		CT	CP	CP	CP	CT	CT
Analista de Sistema 9	Rastreador	CT	CP	CP	CT	CP	CT
Analista de Sistema 10		CT	CT	CT	CT	CT	CT
Analista de Sistema 11		CP	CT	CT	CP	I	CT
Analista de Sistema 12		CT	CT	CT	CT	CT	CT

Fonte: Elaboração da autora, 2016.

Diante das quatro primeiras afirmações respondidas diretamente sobre a melhoria da integração dos testes ao *Scrum*, os analistas de sistema dos projetos da empresa apresentam opiniões diferentes em relação a todas as questões. Sendo assim, é possível verificar que algumas das melhorias propostas não foram implantadas em alguns projetos. E também há casos que ainda não é possível medir se a melhoria é válida, por questões de tempo.

Em relação às duas últimas afirmações respondidas sobre o novo processo, os analistas de sistema apresentam opiniões semelhantes para a pergunta sete (7. A inclusão de todos os participantes do projeto nos eventos Scrum: refino do backlog, planning e retrospectiva, melhoraram o entendimento das atividades e a execução das Sprints?) e diferentes para a pergunta seis (6. A prática do code review nas etapas de implementação e correção ajudaram a identificar falhas e possíveis melhorias no código?), conforme apresentado no quadro 26.

Dessa forma, foi verificado que a questão seis, obteve três respostas indiferente dos analistas de sistema do projeto Caderno de Campo, pois os mesmos não realizam a prática do *code review*.

Além disso, também podemos observar que no projeto Rastreador os analistas de sistema possuem visões iguais ou parecidas em relação as seis afirmações, o que não ocorre nos outros projetos. Visto que, no Rastreador foi possível acompanhar a implantação de todas

as melhorias, pois a desenvolvedora desse trabalho atua no projeto como analista de qualidade.

Por sua vez, os comentários realizados pelos analistas de sistema em relação à pergunta cinco, do tipo aberta, serão agrupados e apresentados a seguir.

Todos os analistas de sistema que responderam ao questionário concordam que houve uma percepção de melhoria na integração dos testes ao *Scrum*. Dessa forma, os mesmos afirmam que, com a integração dos testes, em geral, o desenvolvimento se torna muito mais simples, pois com isso temos a direção exata para seguir.

Os analistas de sistema também explicam que, quando se desenvolve com uma base de teste bem estruturada, o desenvolvimento fica com mais qualidade, pois é possível cobrir a maioria dos casos que poderiam resultar em *bugs*, evitando maiores problemas e retrabalho.

Tendo em vista os casos de testes, os analistas de sistema informaram que ajudam muito na etapa de implementação, principalmente no que diz respeito ao entendimento das atividades, pois os mesmos complementam as informações.

Por fim, os testes automatizados ajudam, e muito, no processo de desenvolvimento, pois garantem uma maior qualidade do software e os testes de serviço, depois de feitos, aumentam o desempenho do desenvolvimento, segundo os analistas de qualidade da empresa.

Apresentado os devidos comentários dos analistas de sistema referente a questão cinco, a seguir serão agrupadas as repostas da questão dez, do tipo aberta.

Todos os analistas de sistema que responderam ao questionário concordam que o novo processo de desenvolvimento de *software* proposto está adequado. Os mesmos destacam como ponto positivo, a participação de todos os colaboradores nos eventos *Scrum*. Além disso, também foi citada a prática dos Standup's diários para ajudar a manter informados todos os integrantes do projeto sobre o que está acontecendo e solucionar alguns problemas e sanar dúvidas.

Um ponto de melhoria citado seria a existência de documentação contendo a visão geral do sistema, como um grafo, para apresentar o que o mesmo faz em cada modulo. Auxiliando novos membros da equipe a entender tanto o sistema quanto as regras de negócio.

Outros pontos de melhoria citados pelos analistas de sistema são a adoção de práticas ágeis consagradas e eficientes, como TDD (*Test Driven Development*) e programação em pares (*Pair Programming*).

Por fim, para os analistas de sistema, o processo de desenvolvimento é bastante prático e pode melhorar conforme os envolvidos amadurecem a forma como se utiliza o método, já que a maior dificuldade do processo hoje é o fator humano.

Para concluir a análise dos resultados da aplicação do questionário, podemos citar que para as oito perguntas fechadas foram obtidas 121 respostas, advindas de 20 pessoas, sendo estas:

- 64 respostas tendo como opção selecionada, concorda totalmente;
- 34 respostas tendo como opção selecionada, concorda parcialmente;
- 20 respostas tendo como opção selecionada, indiferente;
- 2 respostas tendo como opção selecionada, discordo parcialmente;
- 1 resposta tendo como opção selecionada, discordo totalmente.

4.6.5 Conclusão da avaliação

Através das respostas dos questionários, é possível concluir que o processo proposto se mostra adequado para atender as necessidades da empresa PP LTDA, uma vez que, os colaboradores conseguiram perceber os seus pontos positivos. Contundo, há pontos de melhoria que precisam ser avaliados e aplicados. Além disso, foi observado que as melhorias propostas em relação à integração dos testes ao *Scrum* refletiram em benefícios satisfatórios para o processo de desenvolvimento de *software* como um todo.

5 CONCLUSÕES E TRABALHOS FUTUROS

Nesse capítulo, são abordados as conclusões e trabalhos futuros, relacionados ao processo proposto. Ainda, os resultados obtidos, desde a problemática inicial até a avaliação do processo. Também são apresentadas novas melhorias para o processo, sendo consideradas como trabalhos futuros.

5.1 CONCLUSÕES

A integração do teste do *software* em todo o processo de desenvolvimento é primordial para garantir a qualidade do produto. Além de encontrar *bugs*, o teste deve principalmente preveni-los. Contudo, toda a equipe deve estar engajada para atender as necessidades dos clientes da melhor forma possível e assim gerar valor para a solução. Neste contexto, este trabalho apresentou um estudo de caso do processo de desenvolvimento de *software* da empresa PP LTDA.

O processo de desenvolvimento de *software* da empresa foi mapeado e documentado. Durante essa etapa, foram encontrados vários pontos de falha e oportunidades de melhorias foram geradas. Por sua vez, os pontos levantados buscaram auxiliar a empresa a adequar-se melhor as boas práticas do *Scrum*, assim como a integração dos testes ao mesmo. Além disso, também foram propostas melhorias ao processo em geral.

Pesquisar e analisar os conceitos sobre metodologias ágeis com foco no *Scrum*, assim como qualidade e teste de software foi importante para relacionar a prática com a teoria. Além disso, é fundamental conhecer o processo para identificar os problemas na execução das etapas.

As melhorias identificadas para resolver os problemas do processo de desenvolvimento de *software* da empresa, foram agrupadas e classificadas conforma a ordem de prioridade. Sendo assim, no total, foram identificadas trinta e oito melhorias, sendo quatorze de prioridade alta, treze de prioridade média e nove de prioridade baixa.

Dessa forma, o novo processo de desenvolvimento de *software* foi modelado para atender as melhorias propostas. Novas atividades foram criadas para as etapas de

planejamento, implementação, teste de funcionalidade, publicação de funcionalidade, verificação de *bug*, correção, teste de *bug* e publicação de *bug*.

Por sua vez, as melhorias foram implantadas na empresa, inicialmente no projeto piloto Rastreador e posteriormente nos demais projetos: Caderno de Campo, Clicq e Panorama. Após essa etapa, foi realizada a avaliação do novo processo de desenvolvimento de *software* com o objetivo de verificar se houve percepção dos benefícios por parte dos colaboradores do setor de TI da empresa.

Um ponto importante com relação ao processo é a necessidade de acompanhar e revisar o mesmo constantemente com o objetivo de verificar se os colaboradores estão de fato executando o processo corretamente, e no caso de negativo, verificar o motivo e aplicar planos de ação. Uma vez que, as pessoas são os pontos críticos do processo.

Com os resultados obtidos com a avaliação, podemos concluir que o processo proposto se mostra adequado para atender as necessidades da empresa PP LTDA, uma vez que, os colaboradores conseguiram perceber os seus pontos positivos. Contundo, há pontos de melhoria que precisam ser revisados. Além disso, foi observado que as melhorias propostas em relação à integração dos testes ao *Scrum* refletiram em benefícios satisfatórios para o processo de desenvolvimento de *software* como um todo. Portando, após o término dessa monografia podemos concluir que o objetivo proposto para o trabalho foi alcançado.

5.2 TRABALHOS FUTUROS

Para trabalhos futuros, sugere-se documentar o novo processo de desenvolvimento de *software* descrito nessa monografia, utilizando a ferramenta de gestão de conhecimento da empresa, o Confluence, com o objetivo de disponibilizar de forma mais fácil à informação para todos os colaboradores do setor de TI.

Além disso, devem ser verificados os pontos a melhorar apresentados no quadro 27, os quais foram citados pelos colaboradores na seção 4.6.4 deste trabalho.

Quadro 27 - Pontos a melhorar

Pontos a melhorar
1.Acompanhamento do processo.
2.Versionamento com maior garantia de estabilidade nas novas versões.
3. Especificação e priorização das funcionalidades.
4. Criação de um plano de médio prazo para o projeto, cronograma com as *features* que devem ser implementadas e as políticas de testes.
5. Existência de documentação contendo a visão geral do sistema.
6. Adoção de práticas ágeis como TDD (*Test Driven Development*) e programação em pares (*Pair Programming*).

Fonte: Elaboração da autora, 2016.

Por fim, é necessário analisar melhor as questões aplicadas que tiveram como resposta as opções discordo totalmente, discordo parcialmente e indiferente e gerar planos de ação para melhorar estes pontos.

REFERÊNCIAS

AMAZON WEB SERVICES. **Site oficial da ferramenta**. Disponíveis em: <https://aws.amazon.com>. Acessado em: 11 set. 2016.

APPIUM. **Site oficial da ferramenta**. Disponível em: <http://appium.io/>. Acessado em: 26 set. 2016.

ATLASSIAN. **Site oficial das ferramentas Bitbucke, Confluence, Jira e Hipchat**. Disponível em: <https://www.atlassian.com/>. Acessado em: 11 set. 2016.

BACH, James. **Exploratory Testing Explained**. v.1.3.4. 2003. Disponível em: <http://www.satisfice.com/articles/et-article.pdf>. Acessado em: 25 jun. 2016.

BARROS, Aidil Jesus da Silveira Barros; LEHFELD, Neide Aparecida de Souza. **Fundamento de Metodologia Cientifica**. 3. ed. São Paulo: Pearson Prentice Hall, 2007.

BARTIÉ, Alexandre**. Garantia da Qualidade de Software**: as Melhores Práticas de Engenharia de Software Aplicadas à Sua Empresa. 5. ed. Rio de Janeiro: Elsevier, 2002.

BEQUE, Luciéli T. **Avaliação dos Requisitos para Teste de um Sistema Operacional Embarcado**. 2009. Disponível em: <http://www.lume.ufrgs.br/bitstream/handle/10183/17799/000725176.pdf?sequence=1>. Acessado em: 12 jul. 2016.

Brazilian Software Testing Qualifications Board**. Syllabys Agile Testing.** 2014**.** Disponível em: <http://www.bstqb.org.br/uploads/docs/syllabus_ctfl_at_2014br.pdf >. Acessado em: 02 abr. 2016.

CAETANO, Cristiano. **Testes Ágeis**. Material didático da Qualister Consultoria e Treinamento LTDA. 2016.

CERVO, Luiz Amado; BERVIAN, Pedro A.; SILVA, Roberto. **Metodologia científica**. 6. ed. São Paulo: Prentice Hall, 2007.

COHN, Mike. **Desenvolvimento de software com Scrum**: Aplicando Métodos Ágeis com Sucesso. 1. ed. Porto Alegre: Bookman, 2011.

CRISPIN, Lisa; GREGORY, Janet. **Agile Testing**: A Patrical Guide for Testers and Agile Teams. 1. ed. Upper Saddle River, NJ: Addison-Wesley Professional, 2008.

CRISPIN, Lisa, GREGORY, Janet. **More Agile Testing**: Learning Journeys for the Whole Team. 1. ed. Upper Saddle River, NJ: Addison-Wesley Professional, 2015.

FOGGETTI, Cristiano**. Gestão Ágil de Projetos**. São Paulo: Pearson, 2015.

GIL, Antonio Carlos. **Métodos e técnicas de pesquisa social**. 6. ed. São Paulo: Atlas, 2008.

GOOGLE. **Site oficial da ferramenta**. Disponíveis em: <https://www.getpostman.com> Acessado em: 18 set. de 2016.

GOMES, André Faria. **Agile**: Desenvolvimento de software com entregas frequentes e foco no valor de negócio. Casa do Código, 2014.

GUROCK. **Site oficial da ferramenta**. Disponíveis em: <http://www.gurock.com/testrail/>. Acessado em: 18 set. 2016.

HIRAMA, Kechi. **Engenharia de Software:** Qualidade e Produtividade com Tecnologia. Rio de Janeiro: Elsevier, 2012.

HUMBLE, Jez; FARLEY, David. **Entrega contínua**: como entregar software de forma rápida e confiável. 1. ed. Porto Alegre: Bookman, 2014.

JENKINS. **Site oficial da ferramenta**. Disponível em: <https://jenkins.io/>. Acessado em: 11 set. 2016.

JETBRAINS. **Site oficial da ferramenta**. Disponível em: <https://www.jetbrains.com/idea/>. Acessado em: 11 set. 2016.

LAING, Samantha; GRAVES, Karen. **Growing Agile**: A Coach's Guide to Agile Testing. 1. ed. Growing Agile, 2015.

LINZ, Tilo. **Testing in Scrum**: A Guide for Software Quality Assurance in the Agile World. 1. ed. Santa Barbara, CA: Rocky Nook, 2014.

MARCONI, Marina de Andrade; LAKATOS, Eva Maria. **Fundamentos de metodologia científica**. 5. ed. São Paulo: Atlas S.A, 2003.

MASCARENHAS, Sidnei Augusto. **Metodologia Cientifica**. São Paulo: Pearson Brasil, 2012.

MYERS, GLENFORD J. **The Art of Software Testing**. 2. ed. Hoboken, NJ: John Wiley & Sons, 2004.

NOLA, Robert; SANKEY, Howard. **Theories of Scientific Method**. New York, NY: Routledge, 2014.

PEZZÈ, Mauro; YOUNG, Michel. **Teste e Análise de Software**: Processos, Princípios e Técnicas. 1. ed. Porto Alegre: Bookman, 2008.

PRESSMAN, Roger S.; MAXIM, Bruce R.. **Engenharia de Software**: Uma abordagem profissional. 8. ed. Porto Alegre: Bookman, 2016.

PRIKLADNICKI, Rafael; WILLI, Renato; MILANI, Fabiano. **Métodos Ágeis para Desenvolvimento de Software**. Porto Alegre: Bookman, 2014.

PROTRACTOR. **Site oficial da ferramenta**. Disponível em: <http://www.protractortest.org/>. Acessado em: 18 set. 2016.

RUBIN, Kenneth S.. **Essential Scrum**: A Practical Guide to the Most Popular Agile Process. 1. ed. Upper Saddle River, NJ: Pearson Education, 2012.

SCHWABER, Ken; SUTHERLAND, Jeff. **Um Guia definitivo para Scrum**: As regras do jogo. 2013. Disponível em: <http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf>. Acessado em: 08 maio 2016.

STELLMAN, Andrew; GREENE, Jennifer. **Learning Agile**: Understanding Scrum, XP, Lean, and Kanban. 1. ed. Sebastopol, CA: O' Reilly Media, 2014.

SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Person Education, 2011.

RIOS, Emerson; MOREIRA, Filho; TRAYAHÚ, R.. **Teste de Software**. 3. ed. Rio de Janeiro: Alta Books, 2013.

VERSIONONE. **The 10th anual State of Agile Report**. 2015. Disponível em: <http://stateofagile.versionone.com/>. Acessado em: 29 maio 2016.

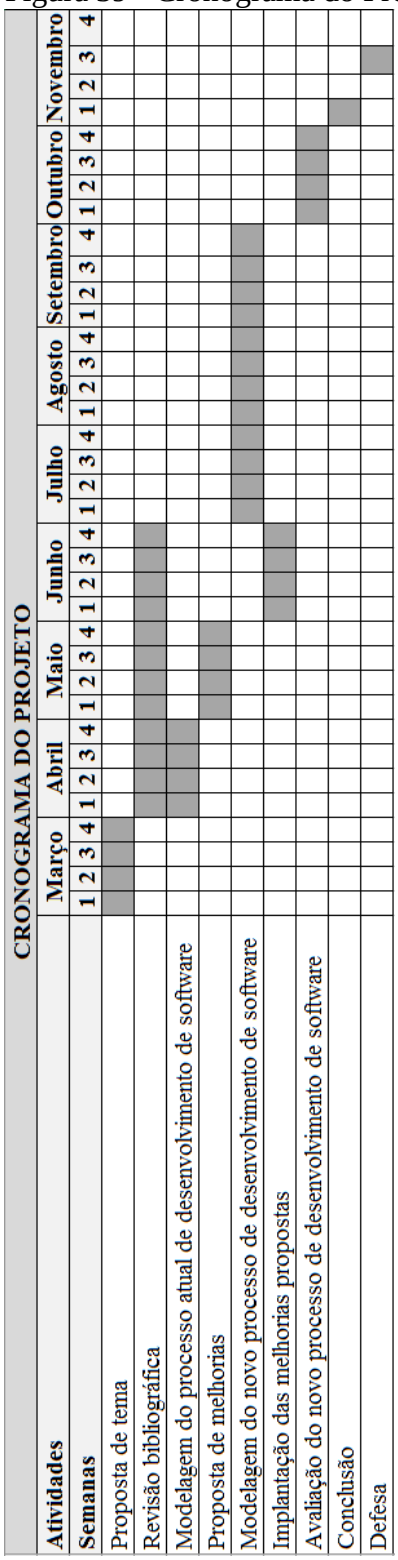
VERSIONONE, **What Is Scrum Methodology?**. Disponível em: <https://www.versionone.com/agile-101/what-is-scrum/>. Acessado em: 01 jun. 2016.

WAZLAWICK, Raul Sidnei. **Engenharia de software:** conceitos e práticas. 1. ed. Rio de Janeiro: Elsevier, 2013.

YIN, Robert K. **Estudo de caso:** planejamento e métodos. 5. ed. Porto Alegre: Bookman, 2015.

APÊNDICE A – CRONOGRAMA DO PROJETO

Figura 39 - Cronograma do Projeto



Fonte: Elaboração da autora, 2016.

UNIVERSIDADE DO SUL DE SANTA CATARINA
Sistemas de Informação
R. Antônio Dib Mussi, 366 - Centro
CEP 88015-110 – Florianópolis - SC

DECLARAÇÃO DE CIÊNCIA E CONCORDÂNCIA DAS INSTITUIÇÕES ENVOLVIDAS NO TRABALHO DE CONCLUSÃO DE CURSO

Local e data: Florianópolis, 14 de Outubro de 2016.

O representante legal da instituição Paripassu Aplicativos Especializados LTDA, objeto do estudo de caso da estudante Danielle Moreira Alexandre executora do **Trabalho de Conclusão do Curso Sistemas de Informação,** intitulado Qualidade de Software e o Framework Scrum: Uma proposta para melhorar a integração dos testes no processo de desenvolvimento de software da empresa PP LTDA, declara estar cientes e de acordo com o desenvolvimento desse trabalho.

O mesmo também autoriza [X], não autoriza [ ] o uso do nome da instituição no referido Trabalho de Conclusão de Curso.

Danielle Moreira Alexandre
Ass. da aluna executora do Trabalho de Conclusão de Curso (UNISUL)

UNISUL - Universidade do Sul de Santa Catarina
Congregação do Curso Sistemas de Informação
Profª Vera Schuhmacher
Coordenadora
Por Delegação do Reitor, através
Portaria nº 1753/2013 de 10 de Outubro de 2013
Ass. e carimbo do Coordenador do Curso

07 661 189/0001 - 02
Paripassu Aplicativos Especializados Ltda ME
Rua:Coronel Luiz Caldeira, nº 67 - Sl. 03 - Bl. 1
ITACORUBI - CEP 88034 - 110
FLORIANÓPOLIS - SC
Ass. e carimbo do responsável da Instituição envolvida no estudo