INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE - IFSUL, *CAMPUS* PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

CAROLINE RASSWEILER

MELHORIA NO PROCESSO DE QUALIDADE: UM ESTUDO DE CASO

Prof. Me. André Fernando Rollwagen

PASSO FUNDO, 2015

CAROLINE RASSWEILER

MELHORIA NO PROCESSO DE QUALIDADE: UM ESTUDO DE CASO

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-Rio-Grandense, *Campus* Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador: André Fernando Rollwagen

PASSO FUNDO, 2015

CAROLINE RASSWEILER

MELHORIA NO PROCESSO DE QUALIDADE: UM ESTUDO DE CASO

Trabalho de Conclusão de Curso aprovado em ____/____/____ como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet

Banca Examinadora:

_____

Orientador Prof. Me. André Fernando Rollwagen

_____

Prof. Esp. Carmen Vera Scorsatto

_____

Prof. Wilian Bouviér

_____

Prof. Dr. Alexandre Tagliari Lazzaretti

Coordenador do Curso

PASSO FUNDO, 2015

À minha família,
pela compreensão e o incentivo
em todos os momentos.

“Prometa menos, entregue mais.”

Tom Peters

RESUMO

A comunidade de engenharia de software tem implantando alguns mesmos métodos que têm prejudicado a indústria com o insucesso frequente. Neste trabalho, é apresentada a tentativa de remediar esta questão, implantando alguns métodos que fornecem uma visão mais ampla a fim de projetar sistemas de software complexos. Dessa forma, foram introduzidas algumas práticas de modelagem de regras de negócios para as fases de análise e teste, mais eficientes, sendo o suficiente para captar a realidade dos sistemas de negócios como promulgada no comportamento e interações de usuários. O método proposto é baseado no modelo híbrido.

Palavras-chave: Qualidade de software; Teste de software; Análise de sistemas; Modelo Cascata; Modelo Ágil; Modelo Híbrido.

ABSTRACT

The software engineering community has deployed some of the same methods that have harmed the industry with the frequent failure. In this paper, we will show the attempt to remedy this issue by deploying a few methods that provide a broader view in order to design complex software systems. In this way, a number more efficient of practices of business rules modeling were introduced for the phases of analysis and testing,being enough to capture the reality of business systems like enacted in behavior and user interactions. The proposed method is based on the hybrid model.

Palavras-chave: Software Quality; Software Testing; Systems Analysis; Waterfall Model; Agile Model; Hybrid Model.

LISTA DE TABELAS

Tabela 1 - Modelo Ágil x Modelo Cascata.....23
Tabela 2 - Tempo gasto para escrita de 1 caso de teste.....36
Tabela 3 - Tempo gasto para escrita de 490 casos de teste 36
Tabela 4 - Funcionalidades ajustadas 37
Tabela 5 - Total de issues dos projetos.....40

LISTA DE QUADROS

Quadro 1- Exemplo de História de Usuário 20
Quadro 2 - Exemplo de História de Usuário 22
Quadro 3 - Cenários de teste..... 23
Quadro 4 - Exemplo de requisito..... 30
Quadro 5 – Requisito da documentação..... 33
Quadro 6 – Esquema de detalhamento da funcionalidade..... 33

LISTA DE FIGURAS

Figura 1 - Métrica dos erros encontrados 15
Figura 2 - Fases do Modelo Cascata 16
Figura 3 - Fases do modelo ágil 19
Figura 4 - Custo relativo para corrigir um defeito 25
Figura 5 - Modelo Híbrido 28
Figura 6 - Passos do Caso de Teste 31
Figura 7 - Caso de teste de cenário principal 34
Figura 8 - Caso de teste de cenário alternativo 35
Figura 9 - Tempo gasto para escrita de 490 casos de teste 37
Figura 10 - Funcionalidades ajustadas, conforme entregas. 37
Figura 11 - Formato dos projetos 39
Figura 12 - Métricas 40

SUMÁRIO

1 INTRODUÇÃO 11
1.1 MOTIVAÇÃO 12
1.2 OBJETIVOS 12
1.2.1 Objetivo Geral 12
1.2.2 Objetivos específicos 12

2 REFERENCIAL TEÓRICO 13
2.1 QUALIDADE DE SOFTWARE 13
2.2 ASPECTOS A SEREM ABORADANDOS QUANTO A QUALIDADE DE SOFTWARE 14
2.2.1 Análise de Software 14
2.2.2 Teste de software 15
2.3 MODELO CASCATA 16
2.3.1 Análise de software no modelo cascata 17
2.3.2 Teste de software no modelo cascata 17
2.4 MODELO ÁGIL 19
2.4.1 Análise de software no modelo ágil 20
2.4.2 Teste de software no modelo ágil 21
2.5 MODELO ÁGIL E MODELO CASCATA 23
2.6 TRANSIÇÃO DE MODELO CASCATA PARA MODELO ÁGIL 25
2.6.1 Educação 26
2.6.2 Documentação 26
2.6.3 Métricas de teste 27
2.6.4 Mudança de atitudes 27
2.7 MODELO HÍBRIDO 27

3 ESTUDO DE CASO 28
3.1 CONTEXTO EMPRESARIAL 29
3.1.1 O processo atual 29

4 APLICAÇÃO 32
4.1 ESCOPO DO PROJETO 32

4.2 ESCRITA DOS CENÁRIOS 32
4.3 ESCRITA DOS CASOS DE TESTE 34
4.4 TESTE DE ACEITAÇÃO 35
4.5 MÉTRICAS E RESULTADOS 36
4.5.1 Métrica da economia de tempo na escrita dos casos de teste 36
4.5.2 Resultado do teste de aceite 37
4.5.3 Resultado Geral 38

5 CONSIDERAÇÕES FINAIS 42

6 TRABALHOS FUTUROS 43

1 INTRODUÇÃO

Com a expansão da área de Tecnologia da Informação (TI), as empresas dependem cada vez mais da construção de softwares. Devido a esse crescimento, corporações que estão adquirindo um novo sistema, ou solicitando modificações em um existente, tem urgência na finalização desse processo. Os solicitantes desconhecem a complexidade de algumas funcionalidades requeridas para serem implementadas. O cliente não considera a fase de levantamento de requisitos importante, sendo assim, o nível de detalhamento dessa fase não é o ideal. Depois de finalizada essa etapa, esse cliente deve ler muitas páginas de especificação, onde se espera que ele tenha o completo entendimento da documentação.

Com base nisso, as empresas desenvolvedoras de software propõem uma implementação em curto prazo para conquistar o cliente. Em virtude desse cenário, é destinado pouco tempo para as etapas que devem ser cumpridas em um projeto desta natureza. Consequentemente, algumas etapas importantes são deixadas de lado para atender a estimativa de tempo conforme destacado por Pressman (2011). Esses tipos de negociações refletem como baixa qualidade do sistema.

Nas etapas posteriores do projeto, percebe-se que os principais defeitos apontados pelo teste, são justamente por falta de definição e problemas no levantamento de requisitos. Muitas vezes, as regras de negócio do cliente acabam dependendo do trabalho e conhecimento do desenvolvedor para implementação de uma funcionalidade.

O desafio do desenvolvimento de um software é tornar uma ideia real. Geralmente a ideia original reflete a realidade das regras de negocio da empresa. No momento de mapeamento dessas necessidades, deve-se garantir que o que foi definido está sendo apresentado conforme especificado. Quanto mais investir na disciplina de requisitos, menor será o tempo e custo de construção.

Na realização do presente trabalho mostrou-se necessário o estudo de duas etapas que são necessárias para entregar um produto mais confiável. A primeira é adaptar a forma de trabalho, utilizando conceitos de métodos ágeis. A segunda é evitar a introdução de erros na etapa de levantamento de requisitos, utilizando conceitos de teste de aceitação.

1.1 MOTIVAÇÃO

A internet é aliada de milhões de empresas que cada vez mais estão elevando o nível de exigência quanto à qualidade de seus websites. A maior parte dos clientes solicitam como obrigatória a fase de teste no projeto, mas em termos de prazo, geralmente o cronograma estipulado prevê um tempo x para a execução dos testes e esta é uma variável que dificilmente tem margem para alteração.

Cada vez mais, é necessário ter um processo ágil e eficiente de teste, inclusive na fase de análise. Para projetos com um grande número de funcionalidades, que utilizam modelo cascata, o processo de teste que faz uso de casos de teste no formato passo-a-passo, aonde a escrita e manutenção são demoradas, a cobertura de validações é baixa e o processo é incessante. Portanto, esse cenário não está suprindo a necessidade imposta, que é um alto nível de qualidade. A proposta é mostrar a possibilidade de adaptar e mesclar esse processo com alguns conceitos de qualidade que são mapeados para métodos ágeis. Assim, é necessária uma forma de trabalho que, através da necessidade de resultados rápidos, satisfaça os pontos a serem melhorados, conforme mencionados acima.

1.2 OBJETIVOS

Nesse capítulo é apresentado o objetivo geral e objetivos específicos do trabalho.

1.2.1 Objetivo Geral

Implementar técnicas que contribuam para qualidade, utilizando abordagem de métodos ágeis.

1.2.2 Objetivos específicos

- Pesquisar sobre os tipos de estratégias e mapeamentos utilizados para qualidade.
- Estudar a influência das estratégias adotadas pela empresa do estudo de caso e seu impacto no custo do projeto.
- Realizar estudo da qualidade em metodologias ágeis para utilizá-los posteriormente na fase de implementação.
- Desenvolver e aplicar conceitos de métodos ágeis.
- Estimar e analisar os resultados do plano desenvolvido.

2 REFERENCIAL TEÓRICO

As seções a seguir visam dar embasamento teórico ao presente trabalho. Serão abordados os assuntos qualidade de software, aspectos a serem abordados na qualidade de software, análise de software, teste de software, modelo cascata, análise no modelo cascata, teste no modelo cascata, modelo ágil, análise no modelo ágil, teste no modelo ágil, modelo ágil e modelo cascata, transição de cascata para ágil e modelo híbrido.

2.1 QUALIDADE DE SOFTWARE

Há um aumento do uso de software, em diversos aspectos do cotidiano. A partir de dispositivos eletrônicos, como relógios e telefones celulares, para aplicações como comércio eletrônico, serviços bancários, serviços médicos. Os sistemas de computação são onipresentes e todos os computadores executam algum software. Devido à ampla aceitação e uso de sistemas de software, em diversas áreas, erros de software estão provando ter custos elevados, e às vezes um erro pode ser fatal. Defeitos podem afetar sistemas bancários, bolsas de valores, instituições médicas. A maioria das falhas, encontradas durante o desenvolvimento de software, pode ser evitada, tendo um controle de qualidade de software.

Existem casos que marcaram a história. Um exemplo ocorreu na década de 80, o Therac-25 era uma máquina utilizada para o tratamento com radiação contra o câncer. Devido a erros de programação concorrente, várias pessoas receberam dosagem extremamente elevada de radiação, resultando em ferimentos graves, além do registro de 6 mortes. Outro exemplo ocorreu nos Estados Unidos, em 2003, onde houve um apagão nas regiões nordeste e centro-oeste do país. Esse episódio ocorreu em função de uma falha no sistema de alarme e deixou mais de 50 milhões de pessoas sem energia. O *blackout* começou perto do horário de pico, atrasando milhões de passageiros, prendendo mais de 800 mil pessoas nos metrôs de Nova York. Dez mil homens da Guarda Nacional e 5.000 policiais foram chamados ao serviço para evitar saques. O prejuízo chegou a US$6 bilhões para o governo americano (ZHIVICH, 2009).

Assim, a qualidade é orientada para a prevenção de defeitos no software, tendo em vista o desenvolvimento de um processo que impede a geração de erros, resultando na produção de software com maior credibilidade. Garantir a qualidade de software é um atributo desafiador no mundo, onde o mercado de negócios está cada vez mais competitivo, uma

empresa tem que se certificar de que seus usuários ou clientes estão recebendo produtos e serviços eficazes o tempo todo. Empresas estão voltando a atenção para a importância da garantia de qualidade utilizando soluções com testes rápidos, que concedam maior estabilidade e desempenho em seus sistemas (SOMMERVILLE, 2011).

2.2 ASPECTOS A SEREM ABORADANDOS QUANTO A QUALIDADE DE SOFTWARE

A seguir serão abordados dois aspectos que influenciam diretamente na qualidade do produto de software.

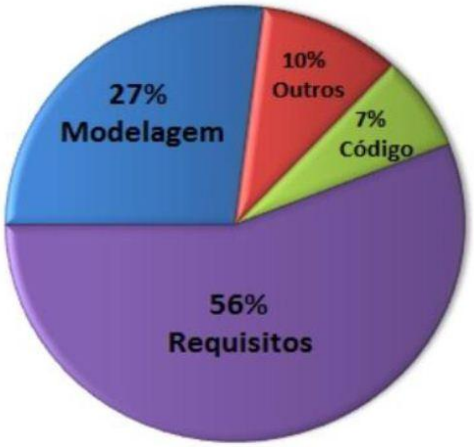
2.2.1 Análise de Software

Análise de requisitos, também chamado de engenharia de requisitos, é o processo de determinar as expectativas do usuário para um produto novo ou modificado. Na engenharia de software, as exigências do cliente são geralmente chamadas de especificação funcional. A análise de requisitos envolve a comunicação frequente com os usuários do sistema a fim de determinar as expectativas específicas da melhor forma possível, pois essa documentação será utilizada do início ao fim do projeto. Essa atividade deve ser dirigida no sentido de assegurar que o sistema final ou produto está em conformidade com as necessidades do cliente (SOMMERVILLE, 2011).

Sendo assim, o principal objetivo desta fase é definir os requisitos. Esta é a etapa mais importante do projeto, pois é necessário entender os requisitos do novo sistema e desenvolver um sistema que atenda a esses requisitos. É preciso ter comunicação clara e habilidades gerenciais para essa etapa ser bem sucedida. A partir disso, é possível obter uma especificação e características que o sistema deve apresentar. O foco é mapear as necessidades e garantir que o cliente entenda as implicações do novo sistema (SOMMERVILLE, 2011).

Análise de um projeto é a primeira etapa para qualquer novo desenvolvimento de software, e fazê-lo corretamente não é o suficiente. É preciso de muita clareza na definição dos requisitos e funcionalidades, além do detalhamento de qual é a melhor forma de implementá-las. Este último ponto geralmente é negligenciado ou ignorado (PRESSMAN, 2011). Esse cenário pode ser observado na figura 1:

Figura 1 - Métrica dos erros encontrados



Fonte: PRESSMAN, 2011

Conforme demonstrado no gráfico, na finalização dos projetos de software percebe-se que boa parte das divergências e falhas são oriundas da fase de análise onde há o levantamento de requisitos (PRESSMAN, 2011).

2.2.2 Teste de software

O teste de software é um termo amplo que rodeia uma variedade de atividades ao longo do ciclo de desenvolvimento e está ganhando cada vez mais importância (RIOS, 2013). Esse trabalho deve ser realizado com cautela, conforme explanação de Pressman:

> O teste muitas vezes requer mais trabalho de projeto do que qualquer outra ação da engenharia de software. Se for feito casualmente, perde-se tempo, fazem-se esforços desnecessários, e ainda pior, erros passam sem ser detectados. Portanto é razoável estabelecer uma estratégia sistemática para teste de software (2011, p. 366).

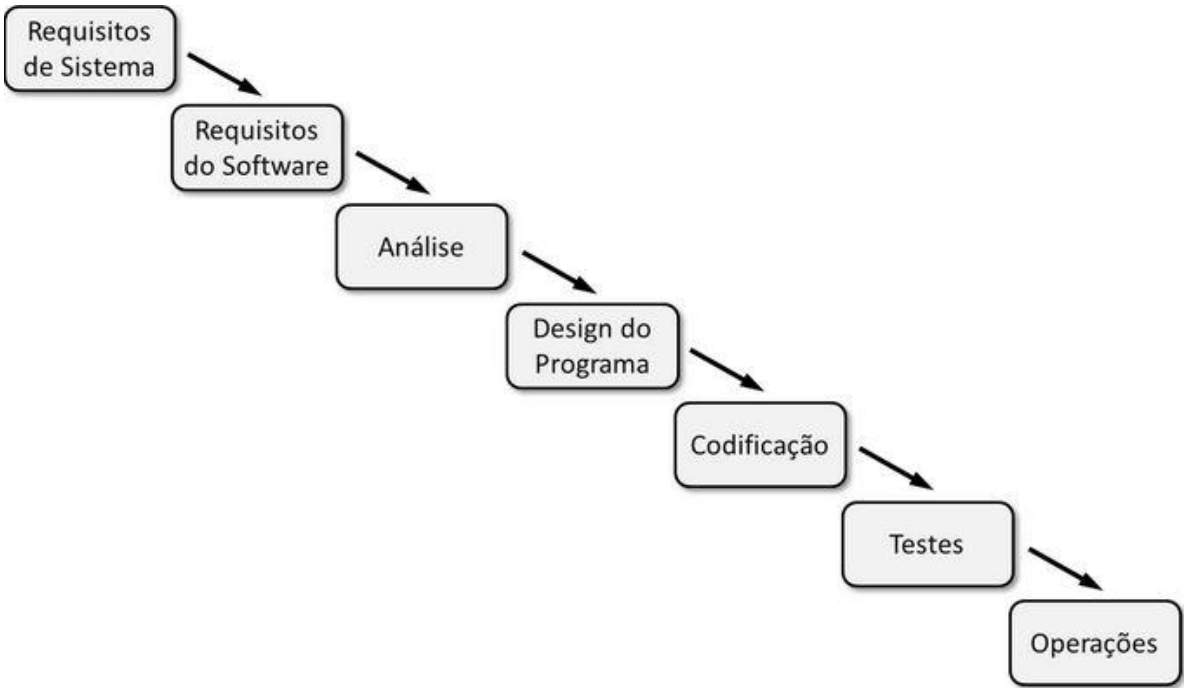
Com base no autor supracitado, percebe-se a importância da realização de um planejamento de teste eficiente, que atenda as expectativas do usuário. A qualidade do software não é testável, mas para desenvolver qualquer coisa de qualidade, é necessário ter testes efetivos.

O teste é um processo que ajuda a determinar a qualidade do software desenvolvido. É um estudo metodológico destinado a avaliar as informações do produto relacionadas com a qualidade. O teste, basicamente, ajuda a rastrear defeitos, avaliar a segurança e o desempenho do software. Esse processo tem muitos benefícios. Assim, o teste é uma etapa importante no processo de desenvolvimento de software que envolve a análise e validação de cada pequeno módulo do sistema (SOMMERVILLE, 2011).

2.3 MODELO CASCATA

O modelo Cascata foi o primeiro a ser utilizado na indústria de software. Trata-se de uma abordagem para o desenvolvimento de software sequencial, que divide um projeto em diferentes fases. Deve-se passar para a próxima fase apenas quando sua fase anterior é finalizada (FOSTER, 2010). A figura 2 mostra as etapas do modelo:

Figura 2 - Fases do Modelo Cascata



Fonte: WAZLAWICK, 2013

Com base na figura é possível observar que as fases não se sobrepõem. Devido a esta natureza, cada fase do modelo em cascata é precisa e com objetivos bem definidos.

Esse modelo é indicado para projetos que são aplicações pequenas, possuem requisitos estáveis, bem compreendidos e dificilmente são modificados. Sendo assim, essa abordagem se torna inadequada para projetos complexos e logos em que, geralmente, os requisitos estão em risco de sofrer constantes modificações (STEPHENS, 2015).

Para atender a grande demanda de projetos de complexidade alta, as empresas estão investindo no modelo ágil para o processo de desenvolvimento, mas a área de teste dificilmente acompanha essa evolução, permanecendo com os conceitos do modelo cascata (WAZLAWICK, 2013). Em vista disso, algumas práticas precisam progredir. Os maiores

problemas são justamente os conceitos que regem essa área. Há quantidade grande de retrabalho, processos ineficientes, que também deixam muito a desejar em questão de cobertura, ou seja, vários defeitos podem passar despercebidos. Todo esse processo demanda valores altos de tempo e custo (WAZLAWICK, 2013).

2.3.1 Análise de software no modelo cascata

O papel principal do analista é o de estabelecer a comunicação com o cliente e produzir um documento que é assinado e será usado como um guia do que deve ser projetado por arquitetos, codificados por desenvolvedores e utilizado por toda a equipe. Nesse documento geralmente não consta a solução técnica final, apresentando somente uma descrição das necessidades do usuário. Devido às fases do modelo cascata, a comunicação é limitada e tende a ser baseada principalmente na revisão de documentos já produzidos. Normalmente nessa documentação estão os requisitos, que são estruturados em um modelo de especificação de requisitos. Para verificar os requisitos envolvidos é de praxe utilizar alguns métodos tradicionais, tais como entrevistas a usuários, observação e estudo de documentos e sistemas de software (SOMMERVILLE, 2011).

A análise em forma de requisitos continua sendo útil para certos tipos de projetos e pode, quando devidamente implementado, produzir economias de custo e de tempo significativos. Estes métodos de coleta de informações são simples e de baixo custo, porém são eficazes quando os objetivos do projeto são claros, pois há pouco ou nenhum risco envolvido. Portanto, é preciso entender claramente as necessidades do cliente (PRESSMAN, 2011).

2.3.2 Teste de software no modelo cascata

Uma vez que o desenvolvimento é concluído, começa a fase de testes na qual será testado cada unidade ou componente afim de garantir que o desenvolvimento está funcionando como esperado. Todas as atividades de teste são realizadas nesta fase. Assim que a equipe de teste inicia seu trabalho, é realizada a análise de requisitos de software utilizando a mesma documentação que foi formulada no início do projeto. Os objetivos da equipe de teste e a equipe de desenvolvimento são distintos, porém, ambas as equipes precisam de uma especificação clara e inequívoca. A equipe de desenvolvimento quer um conjunto completo de requisitos que podem ser usados para gerar uma especificação técnica do sistema, e que lhes

permita projetar e codificar o software. A equipe de teste, por outro lado, precisa de um conjunto de requisitos que irão permitir-lhes escrever um plano de teste[1], desenvolver os casos de teste[2] e executar os seus testes (PRESSMAN, 2011).

Posteriormente é realizado o planejamento dos testes, que deve preparar o ambiente para as etapas seguintes, e compreende as atividades:

- Definição do que vai ser testado e a abordagem que vai ser utilizada;
- Mapeamento de testes para os requisitos;
- Definição dos critérios de entrada e saída para cada fase de testes;
- Avaliação, pelo conjunto de habilidades e disponibilidade, das pessoas necessárias para o esforço de teste;
- Estimativa do tempo necessário para o esforço de teste;
- Calendário das principais etapas;
- Definição do sistema de teste (hardware e software) necessário para o ensaio;
- Definição dos produtos de trabalho para cada fase de testes;
- Uma avaliação dos riscos relacionados com o ensaio e um plano para a sua mitigação.

Os produtos de trabalho ou saídas que resultam destas atividades podem ser combinados em um plano de teste, que pode consistir de um ou mais documentos.

Uma vez que um procedimento de teste é escrito, ele precisa ser testado em ambiente apropriado. A execução de testes funcionais consiste de um conjunto de testes que determina se o sistema faz e o que é suposto fazer do ponto de vista do usuário. Quando a execução de teste da equipe interna estiver concluída, o produto pode ser enviado para o usuário ou cliente para teste de homologação. Nessa etapa o cliente irá realizar o teste de aceite do sistema (SOMMERVILLE, 2011).

Posteriormente há a fase de Manutenção onde são realizados correções e ajustes no sistema, além disso, pode haver melhorias para a funcionalidade do produto para atender as necessidades dos clientes. Para a equipe de teste, manutenção significa a verificação de

---

[1] Um Plano de Teste de Software é um documento que descreve o alcance e as atividades de testes. É a base para testar formalmente qualquer software / produto em um projeto (PAULA FILHO, 2009).

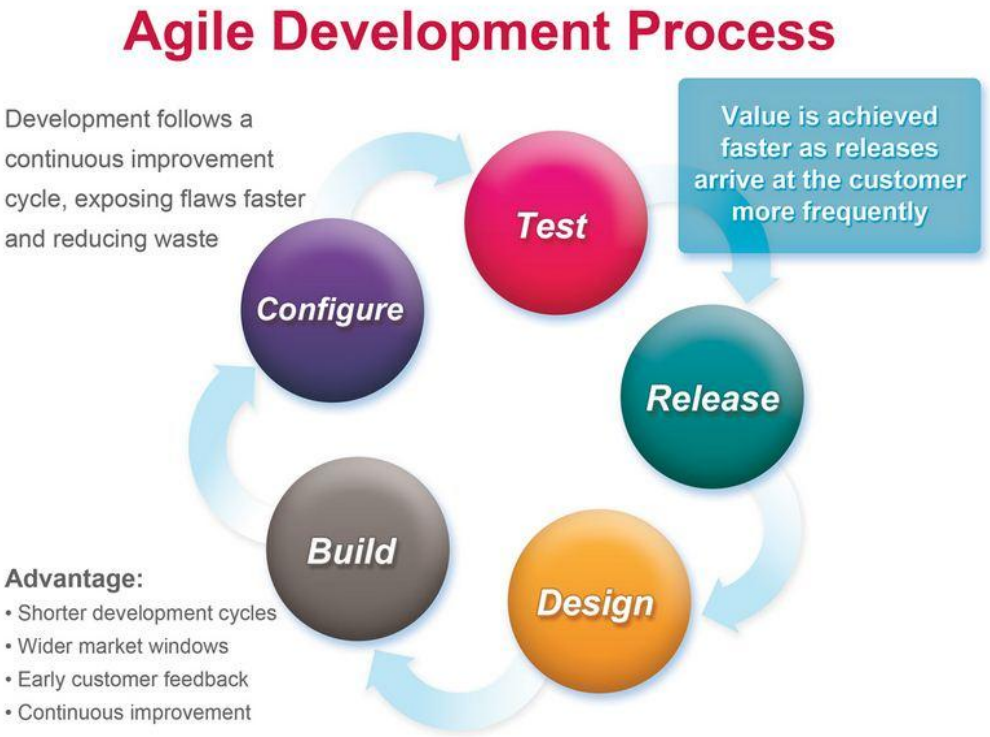
[2] Um caso de teste é um conjunto de condições ou variáveis em que um testador irá determinar se um sistema não satisfaz os requisitos de teste ou funciona corretamente (PAULA FILHO, 2009).

correções de bugs, testar a funcionalidade aprimorada e realizar testes de regressão[3] em novas versões do produto para garantir que a funcionalidade de trabalho não tenha sido prejudicada pelas novas mudanças (SOMMERVILLE, 2011).

2.4 MODELO ÁGIL

Agile é uma metodologia que promove a interação contínua de codificação e testes em todo o ciclo de vida de desenvolvimento de software do projeto. Ambas as atividades de desenvolvimento e testes são concorrentes ao contrário do modelo Cascata (SHORE, 2007). Como pode ser observado na figura 3.

Figura 3 - Fases do modelo ágil



Fonte: AGILE.(s.d.)

Sendo assim, esse modelo prioriza a satisfação do cliente pela entrega rápida e melhoria contínua. Porém, como há falta de ênfase na concepção e documentação necessária, esse processo depende muito de uma comunicação de alta qualidade entre todas as pessoas

[3] Sempre que os desenvolvedores modificarem o software, mesmo um pequeno ajuste, pode ter consequências inesperadas. Esse tipo de teste é realizado para garantir que uma mudança ou adição não quebrou nenhuma funcionalidade existente (PAULA FILHO, 2009).

envolvidas no projeto. E a chave para o sucesso desse modelo é ter uma equipe qualificada, pois as equipes devem trabalhar em colaboração em todos os momentos (CRISPIN, 2008).

2.4.1 Análise de software no modelo ágil

Os clientes, muitas vezes, têm dificuldade em explicar suas exigências a um nível abstrato de uma especificação funcional e só vai entender plenamente o que é necessário quando o aplicativo for entregue. Devido a este tipo de problema, analistas de sistemas começaram a procurar métodos alternativos de sistemas de concepção (COHN, 2010).

Agile é uma metodologia de desenvolvimento de software que depende do estabelecimento de uma relação direta entre toda a equipe envolvida no projeto. Essa metodologia baseia-se na premissa de que os requisitos irão ser modificados e, como tal, não há necessidade de investir na produção de requisitos longos ou documentos de especificação (FINK, 2014).

2.4.1.1 Histórias de usuário

A história do usuário é uma ferramenta utilizada no desenvolvimento *agile* para capturar uma descrição de um recurso de software a partir de uma perspectiva do usuário final. As histórias de usuário são simples o suficiente para que as pessoas possam aprender a escrevê-los facilmente (COHN, 2009).

Histórias de usuários são muitas vezes escritas em cartões de índice ou notas, normalmente são dispostos em paredes, mesas ou quadros para facilitar o planejamento e discussão. O foco é discutir todos os itens, assim os desenvolvedores possuem visão do todo do projeto e não somente das funcionalidades que estão implementando. Assim, essas discussões são mais importantes do que qualquer texto que está escrito (COHN, 2009).

No quadro 1 é possível observar um exemplo de uma história de usuário:

Quadro 1- Exemplo de História de Usuário

Exemplo de uma história de usuário destacado por Helm, 2014:
SENDO um vendedor que realiza 50 visitas por dia
POSSO consultar as últimas compras de cada cliente
PARA QUE ao chegar no cliente eu possa consultar qual foi sua última compra, e assim conseguir negociar com ele estando melhor informado.

Fonte: HELM, 2014.

Conforme quadro 1, é possível verificar que a história de usuário utiliza as palavras chave para informar para quem e por que a funcionalidade está sendo desenvolvida. Essa estrutura auxilia o time de desenvolvimento a tomar decisões mais alinhadas com a necessidade do cliente.

2.4.2 Teste de software no modelo ágil

Teste *agile* é executado de uma forma muito diferente da norma tradicional. Esse método consiste basicamente de um processo contínuo durante todo o ciclo de vida de desenvolvimento de software, fornecendo informações sobre suas funcionalidades. Como resultado, não há tempo para o teste detalhado do software em um estágio posterior e muito menos fazer as correções necessárias, uma vez que é um processo iterativo. Por conseguinte, o primeiro *sprint*[4] é desenvolvido e testado. Em seguida, o segundo é desenvolvido e testado isoladamente, antes de ser integrado com o primeiro *sprint* e assim sucessivamente até que o software seja totalmente desenvolvido (COHN, 2010).

Em testes ágeis, um testador é uma pessoa proficiente, cujas habilidades são usadas em todo o processo de desenvolvimento de software. Enquanto realizam a criação de um plano de teste, eles são utilizados para esclarecer os requisitos, bem como sugerir alternativas (FOSTER, 2010).

Durante a fase de desenvolvimento, os testadores ágeis irão trabalhar com os promotores para escrever o código de tal modo que será mais fácil para testar mais tarde. Assim poupa-se tempo novamente (FOSTER, 2010).

No momento em que a fase de testes começa o testador está completamente ciente das necessidades dos utilizadores e como o sistema deve operar. Em função disso, eles podem fazer o seu trabalho com eficiência, mesmo sem uma longa especificação de requisitos (COHN, 2010).

2.4.2.1 Teste de Aceitação

Os testes de aceitação, extremamente utilizados no contexto ágil, expressam o que o software precisa fazer para que a parte interessada o torne aceitável. Dentro desse conceito, existem técnicas para o desenvolvimento de software de forma incremental, onde os cenários de teste são especificados antes do código funcional (HUMBLE, 2014).

---

[4] *Sprint* é um período de tempo durante o qual um trabalho específico tem de ser concluído e preparado para revisão (SHORE, 2007).

O foco dessa técnica é detectar os defeitos no início do projeto, a ideia de escrita de cenários de teste durante a fase de requisitos é a principal motivação do teste de aceite. Os cenários de teste são escritos em uma linguagem de alto nível simples e não técnica das regras de negócio, ou seja, são roteiros que especificam como o usuário vai utilizar seu produto final (WYNNE, 2012).

O teste de aceite é uma abordagem que utiliza exemplos concretos para descobrir, descrever e formalizar o comportamento de um sistema. O ponto principal é solicitar *feedback* para garantir que as regras de negócio foram compreendidas. É nesse formato que as equipes de software ágeis aprenderam a trabalhar, gerenciando pequenos incrementos com o *feedback* que confirma o resultado esperado pelo cliente. Essa abordagem envolve as equipes de forma colaborativa, assim é possível constituir uma comunicação de alta qualidade (FINK, 2014).

No quadro 2, é possível observar um exemplo de história de usuário:

Quadro 2 - Exemplo de História de Usuário

No quadro 2, é possível observar um exemplo de história de usuário: SENDO um cliente correntista do banco POSSO sacar dinheiro em caixas eletrônicos PARA poder comprar em estabelecimentos que não aceitam cartão de débito/crédito

Fonte: HELM, 2014.

Considerando a história de usuário apresentada no quadro 2, é apresentado no quadro 3 alguns exemplos do formato de suas validações:

Quadro 3 - Cenários de teste

Cenário 1: Horário limite
Dado que são 5h00min
E já estou autenticado no caixa eletrônico
Quando solicito sacar R$10,00
Então o sistema apresenta a mensagem "Os saques somente são permitidos entre 6h00min e 22h59min"
E o saque não é realizado

Cenário 2: Valor máximo de saque
Dado que a hora atual está entre 6h00min e 22h59min
E já estou autenticado no caixa eletrônico
Quando solicito sacar R$1.000,00
Então o sistema apresenta a mensagem "O valor de um único saque no caixa eletrônico está limitado a R$ 800,00"
E o saque não é realizado

Fonte: HELM, 2014.

Conforme exemplo demonstrado no quadro 3, são utilizados os termos "Dado que", para indicar o cenário atual, o "Quando", para indicar a ação do usuário, e o "Então", para indicar como o software vai responder. Para enriquecer os cenários de teste pode-se utilizar os operadores lógicos "E" e o "OU".

2.5 MODELO ÁGIL E MODELO CASCATA

Agile e modelo Cascata são dois métodos diferentes para o processo de desenvolvimento de software. Embora eles sejam diferentes na sua abordagem, ambos os métodos são úteis, dependendo da necessidade e o tipo de projeto, como pode ser visto na tabela 1:

Tabela 1 - Modelo Ágil x Modelo Cascata

Modelo Ágil	Modelo Cascata
O Método Ágil propõe abordagem incremental e iterativa.	O desenvolvimento do software flui sequencialmente a partir do ponto de partida ao ponto final.

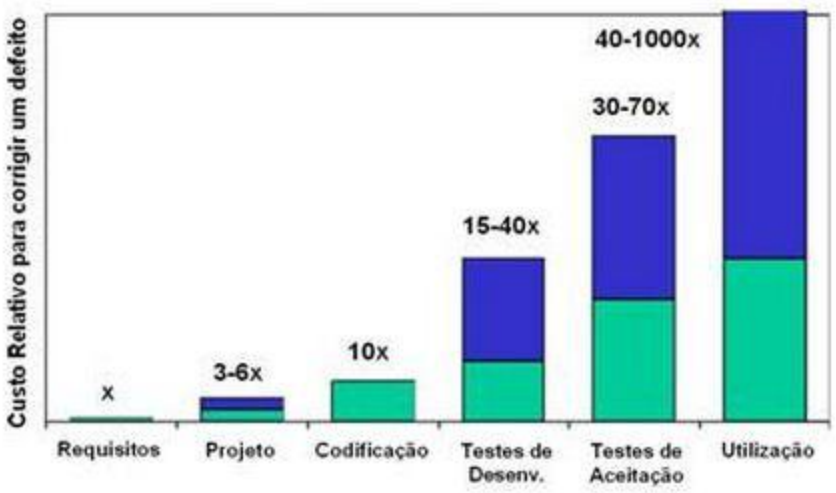
O cliente tem oportunidades frequentes para olhar o produto e sugerir alterações no projeto.	O cliente apenas pode ver o produto no final do projeto.
Erros podem ser corrigidos no meio do projeto.	Somente no final, a totalidade do produto é testada.
Processo de desenvolvimento é iterativo, e o projeto é executado em *sprints* de 2 a 4 semanas.	O processo de desenvolvimento é gradual. Cada fase termina com a descrição detalhada da fase seguinte.
Documentação tem prioridade menor do que o desenvolvimento de software.	A documentação é uma prioridade e pode até mesmo ser utilizada para treinamento de pessoal.
Cada iteração tem a sua própria fase de testes. Permitindo a implementação de testes de regressão cada vez que novas funcionalidades são liberadas.	Somente após a fase de desenvolvimento, a fase de testes é executada.
Testadores e desenvolvedores trabalham juntos.	Testadores trabalham separadamente dos desenvolvedores.
No final de cada *sprint*, a aceitação do usuário é realizada.	Aceitação do utilizador é efetuada no final do projeto.

Fonte: adaptado de (CRISPIN, 2008; FOSTER, 2010; SOMMERVILLE, 2011).

A partir da visão da tabela, há dois principais pontos de atenção no comparativo desses métodos. O primeiro é que, de acordo com a metodologia ágil, não é viável tratar o desenvolvimento e o teste como disciplinas tão distintas. Teste não é uma prática separada, ela deve ser parte integrante do processo de desenvolvimento. A qualidade é alcançada a partir do momento que teste e desenvolvimento, estão fundidos de tal forma que não se distinguem mais um do outro (HITTAKER, 2012).

O segundo trata da busca e correção de erros, somente no final do projeto. Esse ponto de atenção tem um custo elevado no total do desenvolvimento de software. À medida que o projeto vai chegando ao fim os custos de detecção e correções de erros aumentam drasticamente. Ou seja, um erro quanto mais cedo for diagnosticado, mais fácil e mais barato é sua retificação (PRESSMAN, 2011). Esse aspecto pode ser observado no gráfico apresentado na figura 4:

Figura 4 - Custo relativo para corrigir um defeito



Fonte: KALINOWSKI.(s.d.)

Como pode ser observado no gráfico exposto na figura 4 à medida que as fases do projeto avançam, maior é o custo para correção de falhas.

Assim, a prevenção de defeitos é um atributo imprescindível, mas habitualmente é negligenciado na qualidade de software. Se esse cuidado permanecer em todas as fases de um projeto, é possível condensar o tempo e as despesas gerais dando origem a um produto de alta qualidade. Dessa forma, o principal desafio de uma indústria de TI é a engenharia de um produto de software ter o mínimo de defeitos após a implantação (ZHIVICH, 2014).

A fase de requisitos é o componente do ciclo de vida mais instável de um projeto, pois é nesse momento que é realizado o maior número de suposições sobre o produto. No entanto, apenas em fases posteriores as funcionalidades tornam-se mais bem compreendidas. Este aspecto volátil é uma das principais causas de inconsistência na especificação. Assim, desenvolvedores programam regras com base nessas definições fracas e esse cenário propaga erros em fases posteriores (HUMBLE, 2014).

2.6 TRANSIÇÃO DE MODELO CASCATA PARA MODELO ÁGIL

O desenvolvimento ágil não é uma metodologia, mas sim uma mentalidade. Muitas empresas estão praticando o desenvolvimento ágil, e muitas estão lutando com a transformação, existem diversas falhas nesse processo, que geralmente são atribuídas à cultura e resistência à mudança. Essa transição deve começar modificando a mentalidade, antes de tentar abordar as mudanças no processo de trabalho, caso contrário a equipe dificilmente alcançará seus objetivos (CRISPIN, 2008).

A adoção de técnicas ágeis tem sido tradicionalmente atribuída a desenvolvedores, mas o teste ágil é um componente vital do processo e merece uma atenção especial. O desafio é que os métodos tradicionais de testes não se encaixam facilmente com metodologias ágeis. Passos importantes destacados por Stellman (2014), para a transição de cascata para ágil serão descritos a seguir.

2.6.1 Educação

A equipe deve adotar uma mentalidade ágil e se educar sobre o que os processos ágeis vão acarretar. Definir qual será o novo fluxo de trabalho e qual será o papel que deve ser desempenhado por cada membro da equipe. É importante, o grupo de teste ter um envolvimento maior no negócio. Para contribuírem na adoção do modelo ágil e sentir que as suas próprias necessidades estão sendo consideradas.

Todos da equipe devem ser proativos e estar aptos a compreender o valor do negócio e trabalhar para entender as necessidades do usuário final (STELLMAN, 2014).

2.6.2 Documentação

A tradicional dependência de documentação é um hábito que deve ser quebrado e as expectativas que cercam a sua criação devem ser redefinidos. Baseando-se em requisitos de documentos que foram escritos, antes de qualquer desenvolvimento ter sido iniciado, já não é mais válido.

Agile é tudo sobre a adaptação a um *loop* de *feedback* do usuário que desenha o produto final em iterações configurando um processo fluido. Isso significa que qualquer documentação deve evoluir juntamente com o desenvolvimento.

Nesse formato de trabalho os testadores têm mais tempo para passar a fazer a tarefa de documentar casos de teste, e menos tempo para realizar atividades que configuram custo alto para o projeto, como encontrar defeitos. Esse ambiente é propício para geração de *scripts* automáticos para testes de regressão (STELLMAN, 2014).

Em um ambiente ágil é necessário uma documentação inteligente, pois nem tudo pode ser documentado, então é necessário foco sobre o que é realmente preciso documentar para manter os processos em andamento. Encontrar um equilíbrio entre documentar o suficiente para permitir a transferência de conhecimentos futuro, e evitar um trabalho desnecessário, pode ser uma das partes mais difíceis de programar em um processo ágil (STELLMAN, 2014).

2.6.3 Métricas de teste

Talvez a maior mudança de atitude que é necessário para uma transição bem sucedida para o teste ágil vem ao tentar substituir a mentalidade do teste tradicional para métricas. Equipes de qualidade e testadores tradicionalmente utilizam métricas que rastreiam a conclusão de atividades de teste e criação de defeitos. Essas métricas não se alinham com a mentalidade ágil. A equipe deve pensar em métricas que estão alinhadas com o sucesso do negócio. Não concentrar esforços na contagem geral de defeitos, pois esse número não é uma medida adequada da eficácia do teste.

Muitas vezes, os testadores se sentem pressionados para atingir metas que se baseiam somente em números, então a equipe está mais propensa a criar defeitos que são questionáveis. Falhas de projeto e exigências pouco claras não são defeitos.

O testador é o maior defensor para o usuário final e deve estar preocupado em atender as expectativas do mesmo. Assim, o foco deve mudar para a satisfação do usuário final. Todos na empresa devem ter como objetivo entregar o melhor produto para o cliente e ouvir o seu *feedback*, então o sucesso seguirá naturalmente (STELLMAN, 2014).

2.6.4 Mudança de atitudes

O hábito de se comunicar e colaborar são fundamentais para o sucesso do processo ágil. No passado, os departamentos de controle de qualidade trabalhavam como uma unidade isolada, eles se posicionam como guardiões do produto, como se estivessem trabalhando em oposição aos desenvolvedores. No contexto ágil, eles estão inseridos no processo, e auxiliam a entregar maior valor aos negócios.

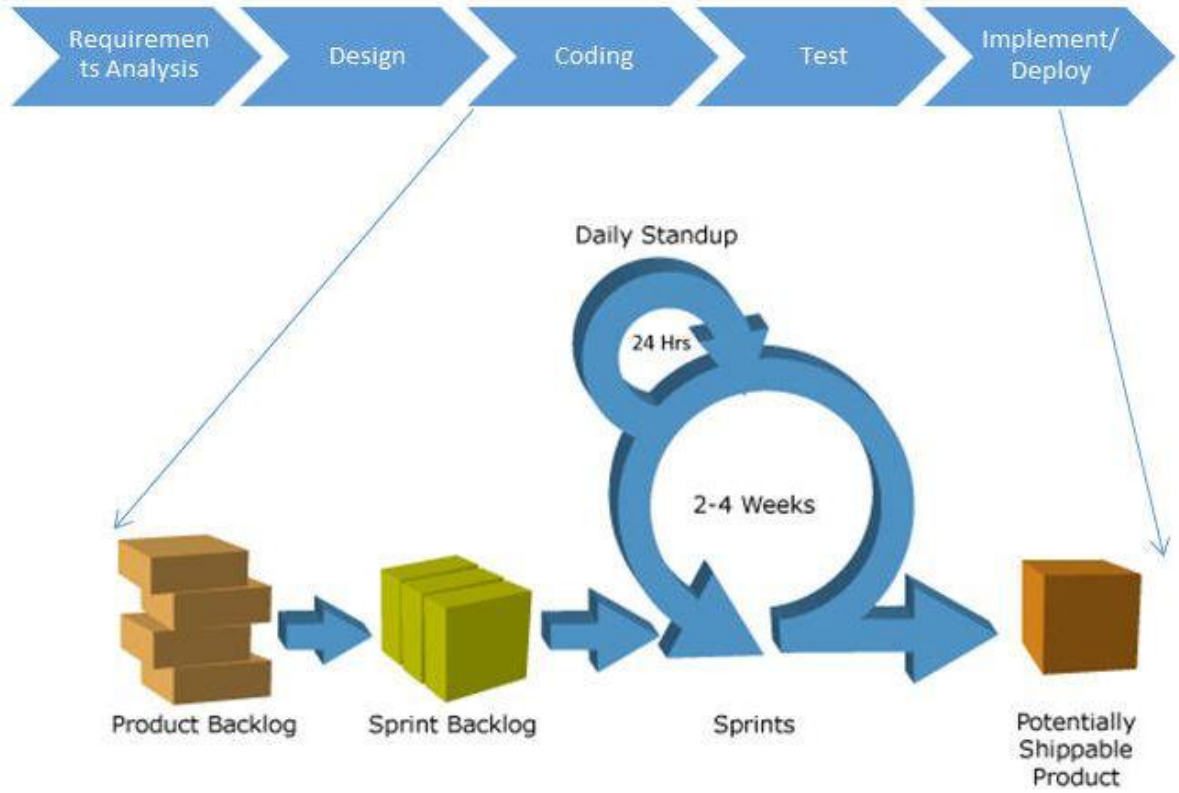
Trata-se de alinhar os objetivos através da empresa de modo que cada departamento e cada membro de sua equipe sigam os mesmo objetivos. Todos devem ser focados em criar a melhor experiência possível para o usuário final (STELLMAN, 2014).

2.7 MODELO HÍBRIDO

O desenvolvimento ágil de software evoluiu para eliminar os problemas que o modelo cascata tem. Quando as empresas que usavam o modelo cascata mudaram para *agile*, a transição trouxe muitos problemas com ele, em função da inadaptação para uma abordagem diferente para o desenvolvimento de software. E o produto final acabou por ser um desastre (CRISPIN, 2008).

Uma nova metodologia tem assim evoluído, que pode ser chamado de 'híbrido', para garantir um produto final robusto, visa combinar os modelos Ágil e Cascata. Essa metodologia remove os inconvenientes de ambos os modelos, enquanto que reúne as respectivas vantagens (LIU, 2013). Assim, esse modelo colaborativo pode ser representado esquematicamente conforme figura 5:

Figura 5 - Modelo Híbrido



Fonte: Liu, 2013.

O aspecto mais importante antes do início de qualquer projeto é decidir o modelo que o time vai adotar. Esse item requer muito planejamento. Fatores como orçamento, tempo, utilização de recursos, complexidade dos requisitos, devem ser considerados na adoção de um modelo de software. Uma alternativa é optar por um modelo híbrido, que é uma forma de alavancar o melhor dos dois mundos: Ágil e Cascata. Cabe aos clientes e gerentes de projeto decidir quais os aspectos a valorizar com prioridade (LIU, 2013).

3 ESTUDO DE CASO

Neste capítulo será abordado um pouco sobre o histórico da empresa, na qual está sendo realizado o estudo de caso, e os processos existentes atualmente.

3.1 CONTEXTO EMPRESARIAL

Empresa brasileira fundada em 1995, que trabalha com desenvolvimento e manutenção de softwares personalizados, implementação de aplicativos, testes e integração de sistemas, voltada para empresas de médio e grande porte. Trabalha com grande maioria de projetos de desenvolvimento de software para Web.

Possui hoje seis unidades localizadas nas cidades brasileiras de São Paulo, Porto Alegre, Passo Fundo, Fortaleza, Erechim e Rio Grande, através das quais os consultores oferecem serviços para grandes organizações no Brasil, América Latina e América do Norte. Dentre as seis unidades existentes, o centro de desenvolvimento de Passo Fundo concentra a maior parte dos funcionários de um total de trezentos colaboradores.

A empresa adota e domina o avanço de tecnologias reconhecidas mundialmente como as apresentadas pela Oracle, possuindo o título de *Oracle Platinum Partner* por conta de um grande time certificado em *Oracle Middleware*, tecnologias de desenvolvimento, integração e *Oracle E-Business Suite*.

Existem hoje na empresa os setores: administrativo, de recursos humanos, financeiro e de consultoria que estão ligados diretamente com a área administrativa. Para as áreas técnicas existem os setores de desenvolvimento e de testes de software.

3.1.1 O processo atual

Atualmente na empresa do estudo de caso, não existe um processo formal de testes e análise. Em cada projeto, existe um processo diferente. A diferenciação entre os modelos de trabalho ocorre naturalmente pela falta de padronização e algumas vezes por peculiaridades de cada cliente.

A análise é feita no padrão do modelo cascata, independente do tipo do projeto. Assim são elaboradas extensas documentações de levantamento de requisitos.

Exemplo de um trecho de um requisito pode ser analisado no quadro de número 4:

Quadro 4 - Exemplo de requisito

RF36 - Venda com retirada em loja

O sistema deverá prover um processo de compra de produtos no qual o cliente tenha disponível a opção de comprar e retirar em loja. Nessa forma de entrega o cliente poderá escolher sua loja de preferência para a retirada do pedido/produto.

O sistema também deverá exibir ao cliente o tempo para disponibilidade do produto para a retirada na loja escolhida, pois, de acordo com a disponibilidade do produto em determinadas lojas esse tempo pode variar.

Não existirá a opção de "retirada agendada" (equivalente à entrega agendada para o caso dos clientes que optaram por retirada em loja).

RF37 - Entrega Parcial

O sistema deverá permitir a entrega parcial dos itens de um mesmo pedido quando estes possuírem prazos distintos.

O layout/UX deverá prever este cenário e o sistema externo que controlará o "prazo x item" deverá informar o prazo de cada item individualmente para que possa ser apresentado corretamente ao cliente.

Na segunda fase será implementada a funcionalidade aonde o cliente poderá escolher se deseja receber o pedido em uma ou mais entregas. No caso da entrega única, o frete deverá obedecer a condição mais vantajosa, em termos de custo ou de promoção (frete grátis). Na condição do cliente escolher por entregas múltiplas, o "segundo pedido" deverá oferecer frete nas condições normais, podendo ou não ser frete grátis, dependendo do valor. Esta informação deverá ficar clara para o cliente na escolha feita.

Fonte: do autor, 2015.

Os testes são em alguns casos executados pela própria equipe de desenvolvimento, que após realizar a implementação, realizam a execução dos testes. Para a realização desta atividade por parte da equipe de desenvolvimento são realizados apenas testes exploratórios baseados no entendimento obtido através da especificação funcional. Normalmente o desenvolvedor que implementou o código é o mesmo que realiza os testes. Esta prática ocorre para que não se perca tempo de outro desenvolvedor para entender o funcionamento da demanda.

Em alguns projetos os testes são realizados por uma equipe de testes independente. Neste caso, normalmente, são utilizados casos de teste, criados e armazenados na ferramenta

de gerenciamento de teste chamada *Testlink*. Na figura 6 é possível verificar a sequencia de passos padrão que é utilizado, por exemplo, para um teste de login:

Figura 6 - Passos do Caso de Teste

#	Ações do Passo	Resultados Esperados:	Execução
1	Acessar o Site indicado para a execução do Teste;	Aplicação apresenta página inicial corretamente;	Manual
2	Acessar a opção "Faça o seu Login" que está no cabeçalho;	Aplicação apresenta tela de Login;	Manual
3	No campo "E-mail ou CPF/CNPJ", informar o e-mail válido do usuário cadastrado;	Aplicação aceita;	Manual
4	No campo "Senha", informar a senha válida do usuário cadastrado;	Aplicação aceita;	Manual
5	Clicar em [Login];	Aplicação aceita e direciona o usuário para última tela que foi visitada, mostrando o nome que o usuário definiu no cadastro: "Olá, <nome do usuário>";	Manual
6	Encerrar o Caso de Teste;	N/A.	Manual

Fonte: do autor, 2015.

Algumas vezes são utilizados *checklists*[5] de interface, mas isso depende também na necessidade e existência de padrão de interface em cada projeto.

Tanto o desenvolvimento quanto os testes são baseados em uma documentação de especificação funcional que prevê apenas os fluxos principais. Dessa forma, muitos fluxos não previstos anteriormente, não são desenvolvidos e deixam lacunas que só são descobertas na fase de teste ou até mesmo na homologação do cliente.

Devido à falta de planejamento na área de testes, em alguns casos os testes finais são executados, até mesmo no ambiente de desenvolvimento, e quando realizados pelos desenvolvedores dificilmente são realizados retestes[6] das correções ou testes de regressão, pois não foi destinado o tempo correto para os testes no momento da estimativa, dessa forma a realização dos retestes e regressões fariam com que o prazo de entrega fosse excedido.

[5] Checklist é uma lista de verificação que contém itens e/ou tarefas (PAULA FILHO, 2009).

[6] Esse tipo de teste é feito após a equipe de desenvolvimento fazer uma correção em uma falha, onde há verificação dessa alteração (SOMMERVILLE, 2011).

Para os colaboradores atuantes na área de testes, os cargos e as atividades que cabem a cada um deles não estão bem definidos. Com isso, ocorre confusão com relação as atividades que são de responsabilidades de cada um, e naturalmente atividades importantes deixam de ser realizadas.

Diante desse cenário, fica clara a necessidade da utilização de boas práticas e de padrões de processo para contribuir com a qualidade dos produtos desenvolvidos.

4 APLICAÇÃO

Através do estudo realizado sobre algumas práticas de métodos ágeis ficou clara a necessidade da utilização de boas práticas e de padrões de processo para contribuir com a qualidade dos produtos desenvolvidos. A pesquisa viabilizou perceber que é possível utilizar algumas práticas de métodos ágeis para trabalhar na prevenção de defeitos e ter melhores resultados, sem precisar trabalhar em um ambiente 100% ágil.

4.1 ESCOPO DO PROJETO

O projeto utilizado para a aplicação trata-se de um portal e-commerce, que em função da complexidade, foi divido em 3 entregas. A entrega 1, abrange funcionalidades da ferramenta administrativa, que é o publicador de conteúdo do site. A entrega 2, abrange fluxos de loja que são realizados com cliente não logado. A entrega 3, abrange as principais funcionalidades do cliente logado na aplicação.

4.2 ESCRITA DOS CENÁRIOS

No momento em que é definida a solução técnica pelo desenvolvedor é elaborado um cenário com informações que complementam a especificação. Assim, o comportamento do requisito é esclarecido e o entendimento é o mesmo para todos os membros da equipe. No quadro 5 é apresentado um requisito da documentação:

Quadro 5 – Requisito da documentação

4.3 Pricing e Promoções

4.3.1 Listas de Preços

A loja Eletro vai operar com 2 listas de preços: "De" e "Por". O preço "De" será o padrão e o preço "Por" será ativado/desativado para alguns itens.
Ainda, poderá ter listas de preços diferenciadas para segmentos de clientes específicos.

[REFERÊNCIA] RQ-5.2.1

Fonte: do autor, 2015.

Para o requisito do quadro 5 foi elaborado o esquema apresentado no quadro 6:

Quadro 6 – Esquema de detalhamento da funcionalidade

SE o produto não tiver estoque no ATG && o atributo **sellable** for IGUAL a "Não", então faz leitura do atributo **status** e exibe a imagem conforme DE-PARA e a **VISÃO 01**.

SENÃO executa o cálculo do menor preço para o cliente e inicia as seguintes comparações:

SE o produto tiver preço Y && este for MENOR que o preço cliente, então exibe **em cima o preço cliente** e **embaixo o preço Y** seguindo a **VISÃO 02**.
SENÃO
SE o cliente for Y && o preço Y for IGUAL ao preço cliente, então exibe **em cima o preço padrão** e **embaixo o preço Y** seguindo a **VISÃO 02**.
SENÃO
SE o preço padrão é MAIOR que o preço cliente, então exibe **em cima o preço padrão riscado** e **embaixo o preço cliente** seguindo a **VISÃO 03** (De x Por).
SENÃO exibe **somente o preço cliente** seguindo a **VISÃO 04**.
FIM SE
FIM SE
FIM SE

FIM SE

Fonte: do autor, 2015.

Nesse contexto, o esquema é validado pelo analista do projeto e os times de desenvolvimento e de teste trabalham com a mesma visão.

4.3 ESCRITA DOS CASOS DE TESTE

Após a revisão da documentação e correção dos problemas devem ser criados os casos de teste, que serviram como um roteiro dos testes.

Todos os casos de teste deverão ser criados e gerenciados através da ferramenta de gerenciamento de teste chamada *Testlink*.

A escrita dos casos de teste foi realizada de forma mais sucinta, porém tendo o cuidado para não ter prejuízo no quesito cobertura de validação. Primeiramente se utiliza de forma mais abrangente as pré-condições do caso de teste, evitando que tenha um passo a passo que é supérfluo. Em seguida, se realiza a escrita de um caso de teste que valida determinada funcionalidade, posteriormente é elaborada a escrita dos casos de teste que possuem validação de cenários alternativos com pré-condição de que o executor do teste já tenha executado o teste principal. Por exemplo: para executar os casos de teste 2 ao 10, é necessário ter o conhecimento que está no caso de teste 1.

Na figura 7 é possível observar o caso de teste principal:

Figura 7 - Caso de teste de cenário principal

Versão 1

Criado em 13/01/2015 17:22:50 por caroline.rassweiler

Última modificação em 09/02/2015 15:35:07 por caroline.rassweiler

Objetivo do Teste:

1. Histórico:
- 13/01/2015 - Caroline Rassweiler - Criação do Caso de Teste;
- 19/01/2015 - Caroline Rassweiler - Adição dos passos do Caso de Teste;

2. Objetivo do teste:
Validar cadastro de enriquecimento em categorias.

3. Pré-condições:
- Estar logado com usuário admin no BCC ("Catálogos" > "[redacted]" > "[redacted]" > "Navegáveis" > "Categoria");
- Estar na tela de configurações de categoria pai;

4. Pós-condições:
Aplicação apresenta enriquecimento de categorias conforme cadastro.

Pré-condições

#	Ações do Passo	Resultados Esperados:	Execução
1	Acessar campo "Promoção da Categoria" e inserir conteúdo HTML.	Conteúdo inserido.	Manual
2	Realizar processos de aprovação do projeto.	Sistema permite e aprova o projeto.	Manual
3	Acessar site indicado para execução e verificar categoria enriquecida no menu principal.	Aplicação apresenta campo descrição e html conforme cadastro.	Manual

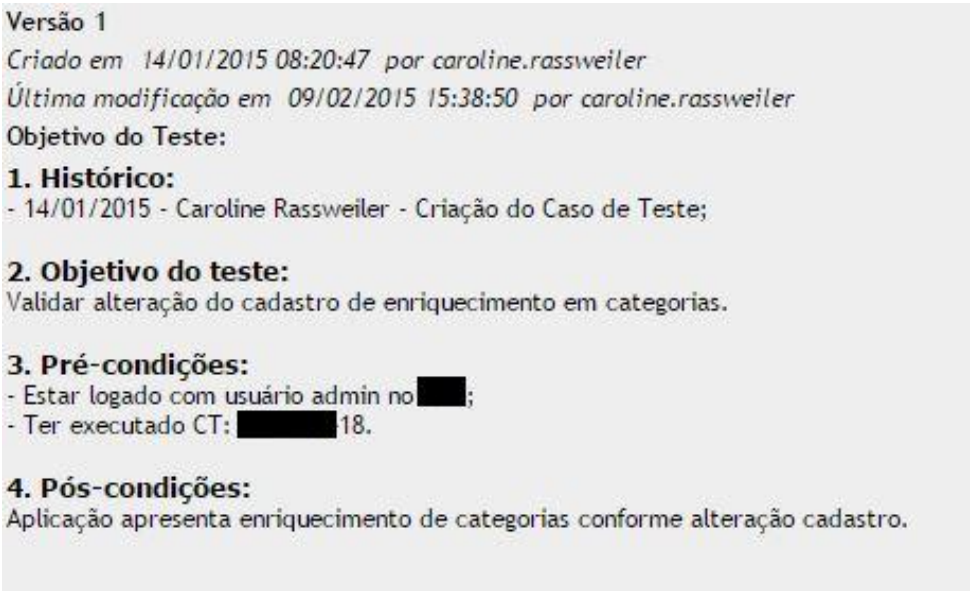
Fonte: do autor, 2015.

No caso de teste principal, é possível observar a utilização das pré-condições. A primeira pré-condição solicita o caminho que o testador deve seguir. A segunda pré-condição especifica que o executor deve estar em uma determinada tela, que é onde estará a funcionalidade a ser testada. Considerando que os testadores já possuem conhecimento da

ferramenta que está sendo utilizada, não é necessário especificar em um passo a passo todas as opções que deveriam ser acessadas para chegar ao destino do teste.

Na figura 8 é possível observar qual a estrutura utilizada para casos de teste que validam os cenários alternativos:

Figura 8 - Caso de teste de cenário alternativo



Fonte: do autor, 2015.

Nos casos de teste que são de cenários alternativos[7] se utiliza como premissa ter executado o caso de teste principal, pois a funcionalidade que sendo testada está no mesmo caminho já descrito no caso de teste principal. Assim, os casos de teste são escritos e executados de forma mais rápida, tendo uma economia de tempo considerável. Pois nesse formato, é possível evitar qualquer tipo de redundância.

4.4 TESTE DE ACEITAÇÃO

Após a finalização da escrita dos cenários da especificação e dos casos de teste, essa documentação foi enviada para o teste de aceite do cliente. Esse envio é realizado sempre antes do cliente começar a homologar a entrega. Assim, o cliente deve informar se está de

[7] Cenários alternativos são aqueles que validam possibilidades da ocorrência de condições que alteram o fluxo normal da execução (SOMMERVILLE, 2011).

acordo com os cenários descritos, e se for o caso, informar qualquer divergência na interpretação do requisito.

4.5 MÉTRICAS E RESULTADOS

4.5.1 Métrica da economia de tempo na escrita dos casos de teste

Com a realização da escrita mais sucinta dos casos de teste, foi possível ter uma redução no tempo da elaboração de um caso de teste, conforme tabela 2:

Tabela 2 - Tempo gasto para escrita de 1 caso de teste

Modelo tradicional	Modelo implementado
50 min	38 min

Fonte: do autor, 2015.

Com base nesse tempo aproximado, estima-se quanto tempo levaria para escrever todo o plano de teste do projeto do estudo de caso no modelo tradicional adotado pela empresa e, quanto tempo levou para escrever o mesmo plano de teste utilizando as práticas abordadas no item 4.3. O resultado é apresentado na tabela 3:

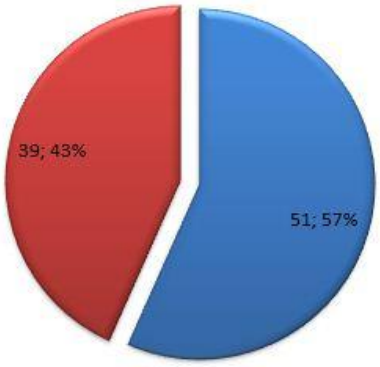
Tabela 3 - Tempo gasto para escrita de 490 casos de teste

Modelo tradicional	Modelo implementado
408 horas	310 horas
51 dias	39 dias

Fonte: do autor, 2015.

Na figura 9, é possível visualizar a diferença em porcentagem do tempo dos modelos demostrados na tabela 3.

Figura 9 - Tempo gasto para escrita de 490 casos de teste



Fonte: do autor, 2015.

Com essa estimativa aproximada, é possível ter uma economia de 14%, equivalente a, aproximadamente, 12 dias no plano de teste total no projeto do estudo de caso.

4.5.2 Resultado do teste de aceite

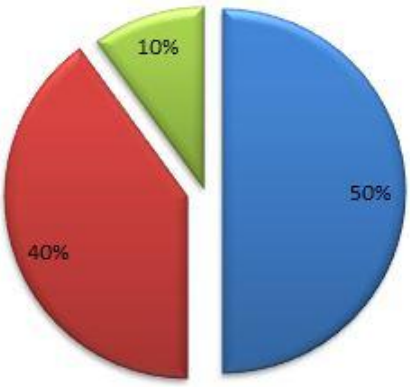
Após a revisão do teste de aceite, realizada pelo cliente, houve necessidade de modificação em algumas funcionalidades, conforme pode ser observado na tabela 4 e figura 10:

Tabela 4 - Funcionalidades ajustadas

Entregas:	Entrega 1	Entrega 2	Entrega 3	Total
Funcionalidades:	5	4	1	10
	50%	40%	10%	100%

Fonte: do autor, 2015.

Figura 10 - Funcionalidades ajustadas, conforme entregas.



Fonte: do autor, 2015.

No momento que o cliente encontra uma divergência nos cenários de teste, ele informa imediatamente ao gestor responsável pelo projeto. Em seguida o gestor sinaliza para as equipes de desenvolvimento e teste, bem como para o analista do projeto. Assim, o analista alinha com o cliente as especificações das modificações solicitadas e qualquer tipo de interpretação equivocada na especificação é ajustado na documentação do projeto. Posteriormente, as alterações são formalizadas com o time de desenvolvimento, que realiza ou ajusta a implementação, e com o time de teste que atualiza o plano de teste. Com esses ajustes realizados, com base em uma boa comunicação, é possível evitar falhas de comunicação e interpretações incorretas da especificação.

4.5.3 Resultado Geral

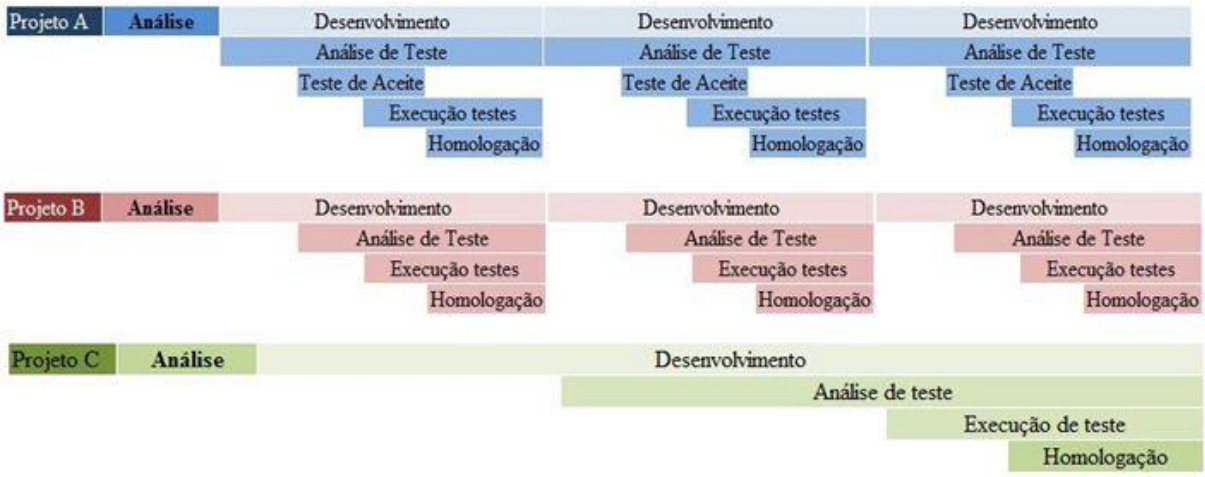
Para estimar o resultado do número de *issues*[8] do projeto da aplicação, serão analisados e comparados três projetos. Esses projetos possuem escopo semelhante, pois todos são portais *e-commerce* de grande porte. Para as estimativas, foram deixadas de lado as regras de negócio mais especificas de cada cliente e foram consideradas somente as funcionalidades em comum de um portal *e-commerce*, por exemplo: Cadastro de cliente, edição dos dados do cliente, fluxo de compras, catálogo de produtos. Os três projetos são:

- Projeto C: Realizado no tradicional modelo cascata.
- Projeto B: Realizado no modelo híbrido.
- Projeto A: Projeto utilizado no estudo de caso e especificado no item 4.1. Realizado no modelo híbrido, com implementação dos itens 4.2, 4.3 e 4.4.

  O formato dos projetos pode ser observado na figura 11:

[8] O termo "issue" é utilizado quando indica que há uma divergência no software que está sendo testado, e não necessariamente indicam que há um problema no código do desenvolvedor (PAULA FILHO, 2009).

Figura 11 - Formato dos projetos



Fonte: do autor, 2015.

De acordo com a ilustração 11, a estrutura dos projetos se apresenta da seguinte forma:

- Projeto C: Primeiramente é realizada toda a etapa de análise. Ao finalizar o fechamento da documentação é iniciado o desenvolvimento. Ao chegar perto de 40% do desenvolvimento, é iniciada a fase de análise de teste, com base na documentação criada no início do projeto. A execução de teste começa quando há mais de 70% do site desenvolvido, assim se inicia a execução pelas funcionalidades que já estão prontas, enquanto o restante é desenvolvido. Por fim, se inicia a homologação do cliente.
- Projeto B: Esse projeto segue os mesmos processos do projeto C, porém está divido em 3 blocos. A análise é realizada toda no início do projeto. A fase de análise de teste segue sendo iniciada somente depois que o desenvolvimento já está em andamento. Após essas etapas, há execução de teste e homologação do cliente. Porém, esse ciclo se repete 3 vezes.
- Projeto A: Projeto é dividido em 3 entregas. A fase de análise novamente é realizada toda no início do projeto. A grande diferença é que, nesse projeto, a análise de teste inicia junto ao desenvolvimento. Na etapa de análise de teste o cliente realiza o teste de aceitação. Em seguida começa a etapa de execução dos testes. A última etapa é a homologação do cliente.

A tabela 5 mostra as métricas de issues que são somente de codificação, ou seja, referentes a correção de funcionalidades.

Tabela 5 - Total de issues dos projetos.

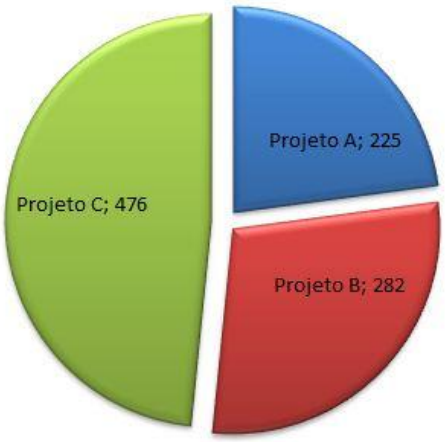
Projeto	**Total de *Issues***
Projeto A	225
Projeto B	282
Projeto C	476

Fonte: do autor, 2015.

Conforme detalhado na tabela 5, do projeto C para o B, já há um ganho alto em função do projeto ter sido dividido em etapas. Em relação ao projeto B para o A, também há um ganho considerável. Foram disponibilizados os cenários de teste para 40% da equipe de desenvolvimento, e os testes de aceite foram realizados pelo cliente em todas as entregas.

No gráfico da figura 12 é possível visualizar essa diferença:

Figura 12 - Métricas



Fonte: do autor, 2015.

O projeto C, teve uma quantidade alta de retrabalho para todos da equipe. Algumas funcionalidades tiveram que ser reimplementadas em função do número alto de issues, além de não atenderem as expectativas do cliente. Esse projeto demorou em torno de 3 meses a mais que os outros para ser encerrado. Quando o projeto foi finalizado, sua documentação, que possuía em torno de 100 páginas, já estava aproximadamente 80% desatualizada.

O projeto B, repete os problemas do projeto C, porém em menor escala. Em função de estar dividido em etapas, foi possível evitar diversos defeitos de codificação.

O projeto A, consegue evitar diversas falhas de comunicação e interpretações equivocadas, trabalhando na prevenção de erros.

5 CONSIDERAÇÕES FINAIS

Com a abordagem dos conceitos relacionados à qualidade do processo de análise e teste de software, realizada no item de Referencial Teórico, percebeu-se a importância da engenharia de software como principal meio para obtenção da qualidade e economia de tempo em um projeto de software. Algumas práticas ágeis foram fundamentais para a definição de alguns processos, o que consequentemente acarretou na qualidade do produto desenvolvido.

A partir dos modelos de análise e teste estudados e da análise do processo existente na empresa em estudo, foi possível definir um processo de teste, que foi constituído pelas boas práticas dos modelos de trabalho estudados, que se adequou melhor ao modelo de trabalho da empresa do estudo de caso.

O processo definido, conforme avaliação realizada no item Métricas e Resultados, obteve um nível de qualidade considerável. A percepção do enquadramento do processo neste nível foi possível através do levantamento de métricas e o comparativo realizado com projetos semelhantes. Este processo contribuiu para a qualidade do produto desenvolvido no projeto em que foi aplicado.

Além da qualidade obtida no produto se percebeu um ganho considerável na qualidade do processo, pois o mesmo trouxe números menores de divergências.

6 TRABALHOS FUTUROS

São destacados alguns temas derivados que podem ser aprofundados e avaliados futuramente:

- Um levantamento de requisitos baseado em modelagem utilizada em métodos ágeis;
- Estudo e comparativo dos tipos de modelos de métodos ágeis como Lean, XP, Scrum;
- Automação dos testes de aceitação com a utilização de ferramentas Jbehave ou Cucumber;
- Automação da fase de execução de testes com Selenium.

REFERÊNCIAS

AGILE. Disponível em: < http://www.gunnertech.com/2011/12/what-is-agile-development/>. Acesso em: 05 abr. 2015.

COHN, Mike. User Stories Applied: For Agile Software Development. 1. ed. Boston: Addison-Wesley, 2009.

COHN, Mike. Succeeding with Agile - Software development using Scrum. 1. ed. Boston: Addison-Wesley, 2010.

CRISPIN, Lisa. GREGORY, Janet. A Pratical Guide For Testers and Agile Teams. 1. ed. Crawfordsville: Addison-Wesley, 2008.

FINK, Gil. FLATOW, Ido. Pro Single Page Application Development: Using Backbone.js and ASP.NET. 1. ed. New York: Apress, 2014.

FOSTER, Elvis C. Software Engineering: A Methodical Approach. 1. ed. New York: Apress, 2014.

HELM, Rafael. WILDT, Daniel. Histórias de usuário: Por que e como escrever requisitos de forma ágil. 2. ed. São Paulo: Lucas Engel, 2014.

HUMBLE, Jez. FARLEY, David. Entrega Contínua Como Entregar software de Forma Rápida e Confiável. 1. ed. Porto Alegre: Editora Boockman, 2014.

KALINOWSKI, Marcos. Artigo Engenharia de Software - Introdução à Inspeção de Software. Disponível em: < http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-inspecao-de-software/8037>. Acesso em: 12 nov. 2014.

LIU, Zhiming. WOODCOCK , Jim. HUIBIAO, Zhu. Unifying Theories of Programming and Formal Engineering Methods. 1. ed. Shanghai: Springer, 2013.

PAULA FILHO, Wilson de Pádua. *Engenharia de Software Fundamentos Métodos e Padrões*. 3ed. Rio de Janeiro: Editora LTC – Livro Técnicos e Científicos Editora Ltda, Integrante GEN – Grupo Editorial Nacional, 2009.

PRESSMAN, Roger. S. *Engenharia de Software*. 7. ed. São Paulo: Editora McGraw-Hill, 2011.

RIOS, Emerson. MOREIRA, Trayahú. Teste de Software. 3. ed. Rio de Janeiro: Editora Alta Books, 2013.

SHORE, James. WARDEN, Shane. The Art of Agile Development. 1. ed. Sebastopol: O'Reilly, 2007.

STEPHENS, Rod. Beginning Software Engineering. 1. ed. Indianapolis: Wrox, 2015.

SOMMERVILLE, Ian. *Engenharia de Software*. 9ed. São Paulo: EditoraPrentice Hall, 2011.

STELLMAN, Andrew. GREENE, Jennifer. *Applied Software Project Mangement*. 1. ed. Sebastopol: O'Reilly, 2014.

WAZLAWICK, Raul Sidnei. Engenharia de Software Conceitos e Práticas. 1. ed. Rio de Janeiro: Elsevier, 2013.

HITTAKER, James. ARBON, Jason. CAROLLO, Jeff. How Google Tests Software. 1. ed. Westford: Addison-Wesley, 2012.

WYNNE, Matt. HELLESOY, Aslak. The Cucumber Book. 1. ed. Dallas, Texas: Raleigh, 2012.

ZHIVICH, Michael. CUNNINGHAM, Robert K. The Real Cost of Software Errors. Disponível em: < http://dspace.mit.edu/handle/1721.1/74607>. Acesso em: 16 nov. 2014.