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

IGOR ALEXANDRO SIQUEIRA SCHERER

ESTUDO DE CASO SOBRE OS BENEFÍCIOS DA PRÁTICA DE TDD: DESENVOLVIMENTO GUIADO POR TESTE EM UMA EMPRESA DE PEQUENO PORTE

Prof. Me. André Fernando Rollwagen

PASSO FUNDO
2016

IGOR ALEXANDRO SIQUEIRA SCHERER

ESTUDO DE CASO SOBRE OS BENEFÍCIOS DA PRÁTICA DE TDD: DESENVOLVIMENTO GUIADO POR TESTE EM UMA EMPRESA DE PEQUENO PORTE

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-rio-grandense, Câmpus Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador : Prof. Me. André Fernando Rollwagen

PASSO FUNDO

2016

IGOR ALEXANDRO SIQUEIRA SCHERER

ESTUDO DE CASO SOBRE OS BENEFÍCIOS DA PRÁTICA DE TDD: DESENVOLVIMENTO GUIADO POR TESTE EM UMA EMPRESA DE PEQUENO PORTE

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

Banca Examinadora:

_____

Prof. Me. André Fernando Rollwagen

_____

Prof. Me. Carmen Vera Scorsatto

_____

Prof. Me. Rafael Marisco Bertei

_____

Coordenação do Curso

PASSO FUNDO

2016

Aos meus pais,
pela compreensão e o estímulo
em todos os momentos . A minha
esposa que sempre me apoiou
nos momentos difíceis.

AGRADECIMENTOS

Primeiramente agradeço a Deus por sempre estar ao meu lado me dando forças para seguir em frente e saúde para concluir este trabalho. A minha esposa Ana Micheli Bagolin que sempre me incentivou e me apoiou nos momentos de dificuldades que enfrentei durante o curso e ao longo da minha vida. Aos meus pais Jorge Fernando Ramos Scherer e Jane de Mello Siqueira, por me ajudarem na realização da graduação e pelos seus ensinamentos e estímulos para eu sempre correr atrás dos meus objetivos. Aos meus irmãos Fernando Siqueira Scherer, Fernanda Siqueira Scherer e Sabrina Siqueira Scherer e ao meu tio Sergio Rodrigo de Mello, pelos pensamentos positivos e as palavras de incentivo e conforto e por estarem sempre ao meu lado.

Agradeço ainda ao meu orientador André Fernando Rollwagen, pela paciência e pelos ensinamentos passados, além da amizade e disposição de me ajudar sempre que precisei. Aos meus amigos em geral, que sempre me apoiaram e sempre me incentivaram a seguir em frente. Por fim agradeço a todos os professores do curso pelos ensinamentos e contribuição que fizeram de mim um melhor profissional e ser humano.

“Cada sonho que você deixa para trás,
é um pedaço do seu futuro que deixa de existir”
Steve Jobs

RESUMO

Atualmente no mundo em que vivemos o software ocupa papel fundamental nas tarefas realizadas diariamente, processos que antigamente levavam dias para serem concluídos hoje levam alguns minutos. Esse benefício trazido pelo software tornou sua disseminação no ambiente coorporativo inevitável, empresas de pequeno, médio e grande porte, possuem sua própria equipe de TI responsável por desenvolver suas necessidades computacionais. Entretanto a grande maioria das empresas não dão a devida importância para os quesitos de qualidade de seus softwares, o que acaba prejudicando seus usuários finais que frequentemente se deparam com erros de sistema o que abala a confiabilidade do software e gera retrabalho para a empresa. Este trabalho tem o objetivo de realizar a comparação de dois projetos reais, sendo um deles utilizando a metodologia TDD e o outro seguindo o processo tradicional de desenvolvimento dentro da empresa, visando através desta comparação evidenciar se a metodologia de desenvolvimento guiado a testes TDD apresenta um ganho de qualidade ao software.

Palavras-chave: TDD, Desenvolvimento guiado a testes, Engenharia de software.

ABSTRACT

Nowadays, in the world we live in, software plays a fundamental role in everyday tasks, processes that used to take days to complete today take a few minutes. This benefit brought by the software has made its dissemination in the corporate environment inevitable, small, medium and large companies have their own IT staff responsible for developing their computing needs. However the majority of companies do not give due importance to the quality requirements of their software, which ends up hurting their end users who often come across system errors that undermine the reliability of the software and generate rework for the company. This work has the objective of comparing two real projects, one of them using the TDD methodology and the other following the traditional process of development within the company, aiming at this comparison to show if the development methodology guided to TDD tests presents a Quality gain to software.

Keywords: TDD, Test-driven development, Software engineering.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACBR – Automação Comercial Brasil

DSDM – *Dynamic Systems Development Method*

FDD – *Feature Driven Development*

GQM – *Goal Question Metric*

IDE – *Integrated Development Environment*

ISO – *International Organization for Standardization*

JOB – Trabalho

NFCe – Nota Fiscal de Cupom Eletrônico

NFe – Nota Fiscal Eletrônica

RAD – *Rapid Application Development*

SEFAZ – Secretaria de Estado da Fazenda

TDD – *Test Driven Development*

TI – Tecnologia da Informação

XP – *Extreme Programming*

SUMÁRIO

1 INTRODUÇÃO

Atualmente os avanços tecnológicos e científicos surgem com maior frequência comparado há anos atrás, um exemplo disso são as versões dos sistemas operacionais Windows e Linux que a cada nova versão lançada no mercado, surpreendem seus usuários com as suas novas funcionalidades implementadas e seu desempenho computacional. Essa evolução tecnológica constante é positiva tanto para o meio coorporativo quanto para o ambiente acadêmico, porém juntamente com esses avanços surgem questões relevantes como a preocupação cada vez maior com a qualidade desses softwares lançados (PRESSMAN, 2011).

Na elaboração de um software diversos aspectos devem ser considerados, entre eles a qualidade, tópico esse vital no sucesso de qualquer sistema que preze pela satisfação do usuário final. Para atingir um alto nível de qualidade computacional testes sistemáticos e adaptáveis são inseridos no contexto da engenharia de software.

Quando a questão é qualidade de software diversos fatores que influenciam no seu insucesso podem ser medidos e até mesmo evitados, como o número de defeitos encontrados em uma ou mais fases específicas do projeto.

Na busca de evitar que os softwares apresentem problemas técnicos na sua utilização e manuseio, diversas metodologias ágeis de testes de software estão sendo utilizadas nos mais diversos segmentos tecnológicos, para obter um maior nível de excelência no quesito qualidade e confiabilidade do software.

Utilizando a metodologia de desenvolvimento guiado a testes (TDD), pode-se aumentar a qualidade do software oferecido ao cliente final sem que seja necessário alterar a estrutura organizacional da empresa, também é possível através do TDD alcançar uma redução no tempo operacional na fase de validação do software.

Diante deste contexto apresentado, o presente estudo foi realizado aplicando a metodologia de desenvolvimento guiado a testes TDD em projeto real de uma empresa do ramo varejista, buscando desta forma comparar um projeto tradicional da empresa com o projeto desenvolvido com o uso do TDD, tendo em vista demonstrar seus prós e contras e evidenciar se essa metodologia apresenta ganhos de qualidade de software.

1.1 MOTIVAÇÃO

O mercado de desenvolvimento de softwares está em crescimento constante no mundo todo e deverá seguir assim por muito tempo, segundo Pressman (2011). Entretanto a maioria das empresas atualmente não investe e muito menos dão a devida atenção para os aspectos da qualidade e testes dos seus sistemas, essa situação é causada na grande maioria das vezes pela questão financeira que se faz necessária para o ganho da qualidade.

Soluções alternativas e com investimentos mínimos ou até mesmo zero estão presentes no mercado de TI, como é o caso do TDD que possui ferramentas *open source* e necessita apenas da capacitação profissional do desenvolvedor de software que lhe fará uso. Deste modo, se torna justificável a implementação da metodologia TDD no presente estudo, como meio de buscar a qualidade do software em uma empresa real de pequeno porte que deseja investir o mínimo possível no quesito de qualidade do seu sistema.

1.2 PROBLEMA

Atualmente no mercado de desenvolvimento de software a complexidade das aplicações exigidas por organizações e empresas privadas cresce a cada ano, equipes numericamente grandes desenvolvem programas de computadores que antigamente eram desenvolvidos por um único indivíduo. A tecnologia de engenharia de software é indispensável para que se possa garantir um nível adequado de qualidade no produto final, Além disso, é preciso ouvir todos os interessados no desenvolvimento das funcionalidades do software (PRESSMAN, 2011).

Tendo em vista a ligeira importância exercida pela engenharia de software na construção de sistemas robustos, deve-se encontrar as melhores metodologias disponíveis em cada situação e para cada organização com características únicas. Segundo Pressman (2011, p.42), “O processo de engenharia de software não é rígido nem deve ser seguido a risca. Mais que isso, ele deve ser ágil e adaptável (ao problema, ao projeto, a equipe e a cultura organizacional) ”.

Entretanto adequar a melhor metodologia para cada situação não é o suficiente para garantir qualidade ao software, faz-se necessário também estar ciente de que o levantamento indevido de requisitos de um sistema pode acarretar

problemas sérios ao projeto, pois uma definição indevida ou não interpretada corretamente dos requisitos compromete todo o ciclo da engenharia de software, desde a fase de desenvolvimento até a entrega final ao cliente. Filho cita em seu livro que um dos problemas básicos da engenharia de software está no levantamento de requisitos e a documentação do mesmo:

> Um dos problemas básicos da engenharia de software é o levantamento e a documentação dos requisitos dos produtos de software. Quando esse levantamento é bem feito, os requisitos implícitos são minimizados. Quando a documentação é bem feita, os requisitos documentados tem maiores chances de ser corretamente entendidos pelos desenvolvedores. Algumas técnicas de análise dos requisitos ajudam a produzir especificações mais precisas e inteligíveis. O conjunto das técnicas de levantamento, documentação e análise forma a engenharia dos requisitos, que é uma das disciplinas da Engenharia de Software (FILHO, 2005, p. 08).

Por essa razão, fica notória a importância de um correto levantamento de requisitos e de uma precisa validação, para que as funcionalidades do software sejam correspondidas acertadamente. Outro fator relevante quando se trata de qualidade de software está no processo de teste do produto, ponto crucial para o sucesso de um sistema de boa qualidade, uma vez que o teste é responsável por garantir que as funcionalidades estão de acordo com aquelas exigidas no levantamento dos requisitos.

A prática de teste de software é indispensável até mesmo para sistemas extremamente simples, pois através dele é possível obter a resposta de que o software se comporta conforme o esperado. Para Schach um teste bem-sucedido é aquele que encontra falhas:

> Um teste bem-sucedido encontra falhas, e isto também representa uma dificuldade. Ou seja, se o artefato de código passar pelo teste, então o teste falhou. Ao contrário, se o artefato de código não atuar de acordo com as especificações, o teste foi bem-sucedido. É solicitado a um programador que teste seu próprio artefato de código onde esse deve ser executado, de tal forma que surja, em seguida, um defeito (ou um comportamento incorreto). Isso vai contra o instinto criativo dos programadores (2010, p. 170).

Os projetos em que o teste de software é falho ou tratado em segundo plano possuem maior probabilidade de apresentar algum defeito grave para o usuário, ou apresentar um comportamento diferente do esperado no momento da entrega de um protótipo ou versão final. Permitir que o cliente se depare com uma destas situações

degradam a confiabilidade do software, fazendo com que a imagem de qualidade produto consequentemente também seja afetada (PRESSMAN, 2011).

Portanto, o grande desafio presente nisso tudo é adequar uma melhor metodologia de desenvolvimento de software que atenda as necessidades de eficiência no levantamento de requisitos combinado com um bom processo de testes, respeitando as características únicas e uma cultura organizacional própria da essência corporativa da empresa.

1.3 OBJETIVOS

Neste capítulo é apresentado o objetivo geral e os objetivos específicos do presente trabalho.

1.3.1 Objetivo Geral

Aplicar a metodologia de desenvolvimento guiado por teste (TDD) em uma empresa do ramo varejista que não possui equipe de teste especializada, com o intuito de demonstrar se a metodologia apresenta ganhos de qualidade e redução do tempo operacional.

1.3.2 Objetivos Específicos

- Pesquisar as principais metodologias de teste de software disponíveis atualmente no mercado;
- Estudar a metodologia de desenvolvimento guiado por teste (TDD), seus conceitos, requisitos para implantação e ferramentas existentes;
- Modelar um projeto da empresa com a criação de cenários de testes para o TDD e aplicar a metodologia em um projeto real da empresa acompanhando o processo de desenvolvimento e teste juntamente com a equipe de programadores;
- Comparar os resultados obtidos por meio da implementação desenvolvida através do TDD, com dos dados de um processo antigo de desenvolvimento de software da empresa.

1.4 JUSTIFICATIVA

A prática de engenharia de software tem o forte compromisso de entregar os projetos no prazo estipulado e com alta qualidade, além de garantir todas as funcionalidades exigidas para o produto. Os benefícios proporcionados pela prática de engenharia de software já são comprovados e sua adoção é de suma importância quando se busca o cumprimento de prazos e a garantia de um nível elevado de qualidade (PRESSMAN, 2011).

Mesmo ciente de todos os prós que a utilização da engenharia de software agrega ao projeto, inúmeras empresas mantém uma posição de resistência contra o mesmo, justificando que sua aplicação é algo muito custoso e demorado até que se atinja um nível maduro no processo. Por outro lado, quando a aplicabilidade de uma determinada metodologia garante uma diminuição considerável no retrabalho e na prevenção de falhas, isso faz com que a utilização da mesma seja vista com outros olhos por parte das companhias (SOMMERVILLE, 2007).

Através de um processo eficaz e adaptável de teste de software chega-se a um nível avançado de prevenção a falhas, oferecendo um ganho financeiro devido ao tempo operacional que deixa de ser gasto para a correção de futuros erros ou funcionalidades indevidas. Um processo bem definido e planejado de testes, que seja adequado para realidade da empresa pode oferecer inúmeros benefícios em curto e longo prazo, Pressman ainda destaca:

> No contexto da engenharia de software, um processo não é uma prescrição rígida de como desenvolver um software. Ao contrário, é uma abordagem adaptável que possibilita às pessoas (a equipe de software) realizar o trabalho de selecionar e escolher o conjunto apropriado de ações e tarefas. A intenção é a de sempre entregar software dentro do prazo e com qualidade suficiente para satisfazer àqueles que patrocinaram sua criação e àqueles que irão utilizá-lo (2011, p. 40).

.

Aprimorar constantemente a forma com que o software é submetido a testes e validações é fundamental para a evolução da gestão de qualidade. O processo de testes deve estar presente em todos os momentos do projeto quando tratamos de metodologias ágeis, as técnicas como a revisão de software são indispensáveis para revelar erros e defeitos que podem ser eliminados antes que cheguem ao contato com cliente (SOMMERVILLE, 2007).

No sentido de buscar o processo ágil de desenvolvimento e a prevenção de falhas Pressman enfatiza que:

> A Extreme Programming (XP) é o processo ágil mais amplamente utilizado. Organizada em quatro atividades metodológicas, planejamento, projeto, codificação e teste a XP sugere um número de técnicas poderosas e inovadoras que possibilitam a uma equipe ágil criar versões de software frequentemente, propiciando recursos e funcionalidade estabelecidos anteriormente, e, então, priorizando os envolvidos (PRESSMAN, 2011, p. 103)

Dessa forma seguindo no objetivo de alcançar técnicas poderosas e inovadoras chega-se ao encontro da metodologia TDD, que para Cohn (2011, p.179) é uma abordagem inestimável, “Uma das principais razões é que ele garante que nenhum código não testado entre no sistema”. Essa garantia dá ao projeto segurança e maturidade suficiente para que se chegue a um alto nível de qualidade.

O desenvolvimento guiado por testes enfatiza a produção mais eficaz de sistemas computacionais, sendo uma poderosa ferramenta de qualidade e validação funcional. Desenvolver softwares orientados a testes garante ao projeto que todo o código que entre no sistema foi previamente testado e validado, garantindo sua funcionalidade e aprimorando o levantamento de requisitos em cada teste unitário.

O uso de TDD não está restrito apenas para desenvolvedores, esse conceito segundo Beck (2010, p.19) pode ser seguido por qualquer engenheiro de software, “Desenvolvimento guiado por testes é um conjunto de técnicas que qualquer engenheiro de software pode seguir, que encoraja projetos simples e conjuntos de testes que inspiram confiança”.

Nesse contexto, busca-se demonstrar que através da aplicação do desenvolvimento guiado por testes (TDD), metodologia que faz parte do conceito de programação extrema (XP), há um ganho expressivo na qualidade do software oferecido para o cliente, já que essa é uma das premissas da engenharia de software, e, além disso, espera-se também uma redução no retrabalho do projeto através de melhores levantamentos de requisitos e prevenção a falhas.

2 REFERENCIAL TEÓRICO

As seções a seguir visam dar embasamento teórico a esse projeto de pesquisa. Para isso, serão abordados os temas de engenharia de software, metodologias tradicionais, metodologias ágeis de desenvolvimento, qualidade de software, casos de testes, testes unitários, testes automatizados, desenvolvimento guiado por testes, linguagem de programação *Pascal*, linguagem de programação *Delphi* e *IDE Delphi 7*, *framework DUnit*, estudos relacionados e *goal question metric*.

2.1 ENGENHARIA DE SOFTWARE

A engenharia de software está presente nas tarefas diárias contribuindo para tornar a vida mais confortável, além de tornar as atividades exercidas rotineiramente mais eficientes. Seu objetivo é tornar a vida do usuário o mais fácil e prazerosa possível ao realizar tarefas computacionais que estão presentes em suas atividades.

Muitas vezes sua presença passa despercebida no cotidiano das pessoas, um exemplo disso é uma tarefa frequente de preparar o café, onde o código inserido na torradeira controla o grau em que o pão estará torrado e quando o produto final saltará da torradeira. Esse exemplo já reflete o quão disseminada a engenharia de software encontra-se no mundo atual.

O software está presente em todos os aspectos da vida, explicitamente ou mesmo sem se fazer notar, inclusive em sistemas críticos que afetam a saúde e o bem-estar de todos (PFLEEGER, 2004).

A engenharia de software define um conjunto de procedimentos que visam auxiliar na resolução de problemas computacionais decorrentes dos mais diversos softwares inseridos ao usuário. Segundo Pfleeger (2004) os engenheiros de software utilizam seu conhecimento sobre computadores e computação para ajudar na resolução de problemas:

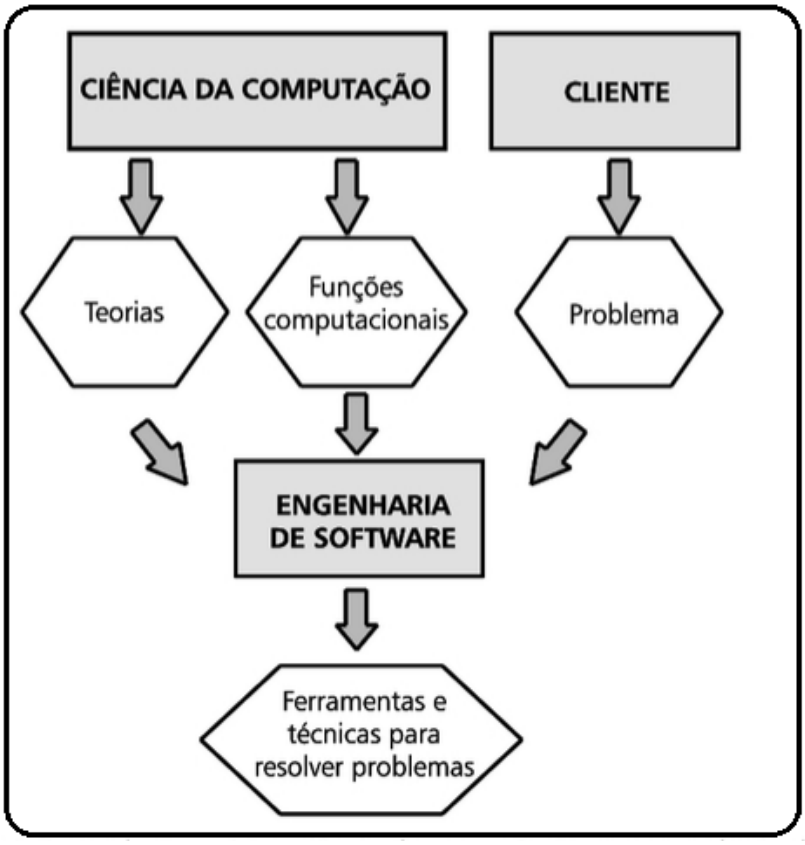
> Como engenheiros de software, utilizamos nosso conhecimento sobre computadores e computação para ajudar a resolver problemas. Frequentemente, o problema com o qual estamos lidando está relacionado a um computador ou a um sistema computacional já existente, mas, algumas vezes, as dificuldades que são apresentadas não têm relação com computadores. Portanto, é essencial entender primeiro a natureza do

> problema. Devemos ser muito cautelosos para não impor máquinas e técnicas computacionais a toda a questão que aparecer em nosso caminho (2004, p. 02).

Portanto deve-se ter a ciência de analisar, antes de qualquer coisa, o problema em questão para depois com base no conhecimento em computação aplicar o melhor procedimento para sua resolução.

A figura 1 ilustra a relação entre o engenheiro de software que detém o conhecimento necessário e o cliente que apresenta o problema que lhe afeta diretamente, esses dois fatores englobam uma integração pela engenharia de software resultando em ferramentas e técnicas para a resolução do problema do cliente.

Figura 1 – Relação entre ciência da computação e a engenharia de software.



Fonte: PFLEEGER 2004, p 04.

Como visto na figura 1, a engenharia de software busca relacionar a pessoa que possui um problema e o engenheiro de software com o intuito de obter êxito na solução do problema, criando um vínculo de comunicação entre as duas partes envolvidas. Na produção de um novo software não é diferente, pois os sistemas

computacionais só surgem decorrentes de alguma dificuldade ou necessidade do cliente em realizar um determinado processo (PRESSMAN, 2011).

Por essa razão Pfleeger (2004, p.11) ressalta que a comunicação entre o cliente e desenvolvedores é um ponto chave para o sucesso de um software: “Um componente chave do desenvolvimento de software é a comunicação entre os clientes e os desenvolvedores; caso ele falhe, o sistema também falhará”, portanto deve-se buscar ao máximo metodologias e técnicas que visem o sucesso na comunicação entre clientes e desenvolvedores.

A engenharia de software não deve ser algo estagnado, rígido e sistemático, muito pelo contrário, deve ser algo adaptável e em constante evolução visando sempre aprimorar a comunicação entre os envolvidos no processo como um todo.

2.2 METODOLOGIAS TRADICIONAIS

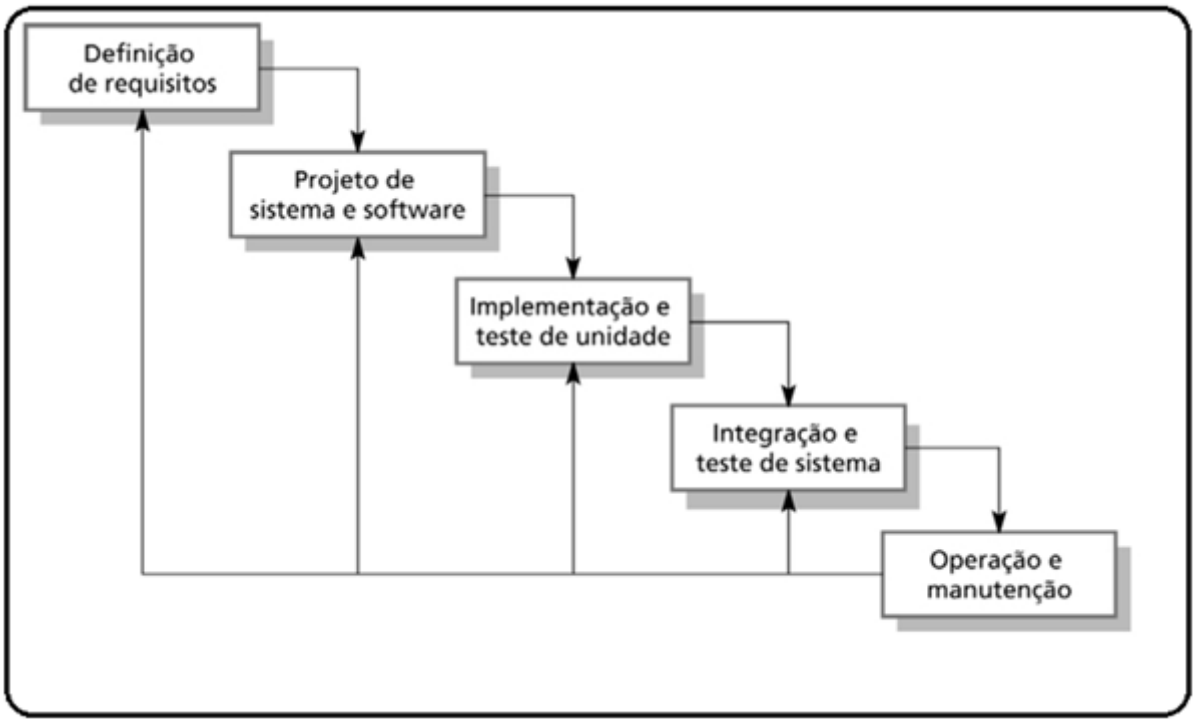
As metodologias de desenvolvimento tradicionais ou “pesadas” como são conhecidas no meio acadêmico foram muito utilizadas no passado. Esse modelo foi planejado em um ambiente onde faziam parte terminais burros e *mainframes*, como o custo de realizar alterações e manutenções nesses sistemas era altíssimo a metodologia propôs que o software fosse bem planejado e documentado para depois ser desenvolvido.

2.2.1 Modelo linear ou modelo cascata

Uma das metodologias tradicionais mais utilizadas no mundo todo e que ainda se mantém presente nos dias de hoje é o modelo cascata. Esse modelo foi idealizado em 1970 e sua característica principal é a sequência das atividades, sugerindo um tratamento ordenado e sistemático. O modelo considera as atividades fundamentais do processo, compreendendo especificações, desenvolvimento, validação e evolução e as representa como fases separadas (SOMMERVILLE, 2007).

A figura 2 demonstra os principais estágios do desenvolvimento em cascata, nesse modelo clássico a próxima fase se inicia apenas quando a anterior for aprovada.

Figura 2 – Ciclo de vida de software modelo cascata.



Fonte: SOMMERVILLE 2007, p.44

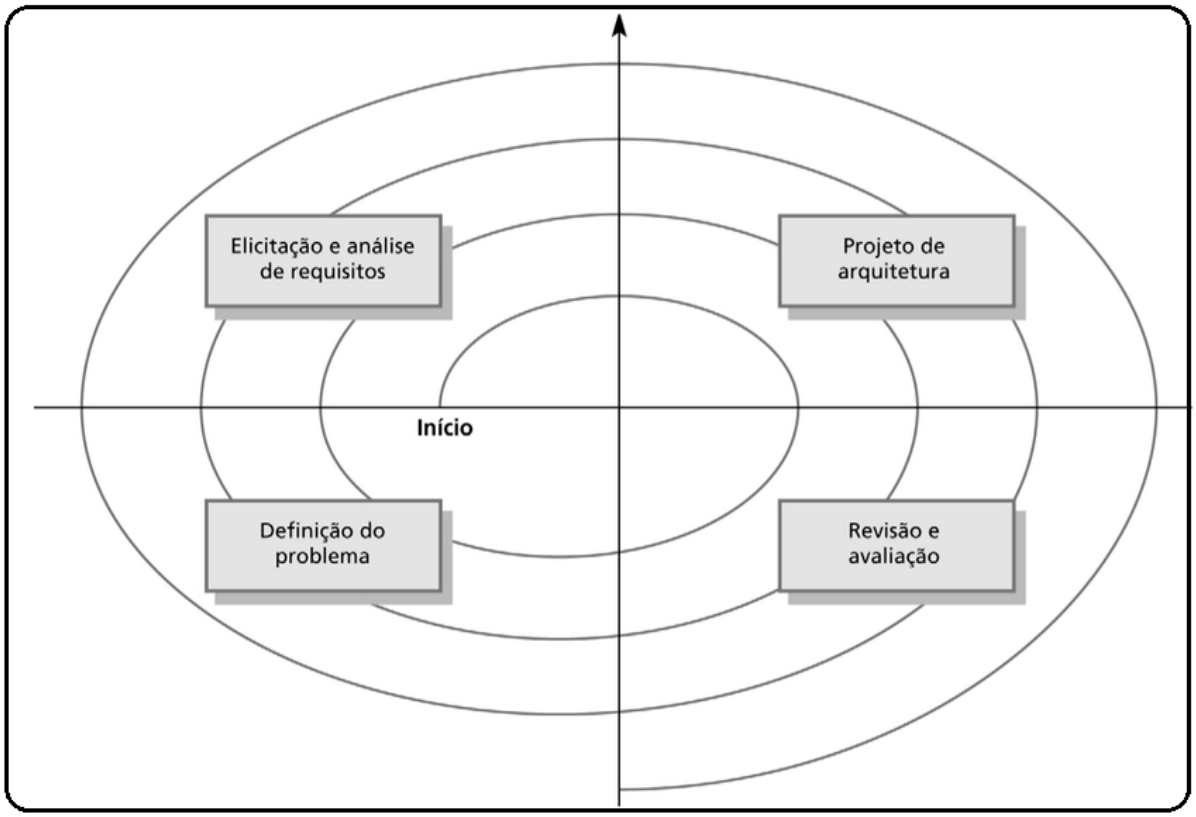
Nesse modelo, cada fase transcorre de forma separada e sequencial, o sistema é desenvolvido em um processo longo e entregue ao final do mesmo. Nessa metodologia o cliente precisa de muita paciência, pois uma versão do software só estará disponível próximo do encerramento do projeto. Outro fator negativo é no caso de ocorrer um erro nas fases iniciais do processo, isso irá comprometer todo o projeto.

2.2.2 Modelo espiral

Esse modelo é uma evolução do modelo cascata, visto que é baseado em uma nova sequência de fases que resultam a cada fase espiral em uma nova versão incremental do programa, podendo gerar em determinadas circunstâncias um protótipo.

Segundo Sommerville (2007, p.20) “O processo espiral reflete o fato de que os requisitos afetam as decisões de projeto e vice-versa e, assim, faz sentido interligar esses processos”. É possível notar que no modelo exibido na figura 3, as fases não estão estagnadas como visto no cascata, elas se interligam influenciando as tomadas de decisões a cada passada da espiral.

Figura 3 – Ciclo de vida de software modelo espiral.



Fonte: SOMMERVILLE 2007, p.20.

No modelo apresentado na figura 3, é demonstrado o funcionamento do processo espiral, em que o mesmo age de forma circular em que cada giro do espiral agregando novas definições de requisitos e funcionalidades. A ideia de gerar protótipos de software em determinados ciclos do modelo visam minimizar os riscos envolvendo o projeto oferecendo aos clientes prévias funcionais das soluções necessárias para o seu problema.

2.2.3 Modelo concorrente

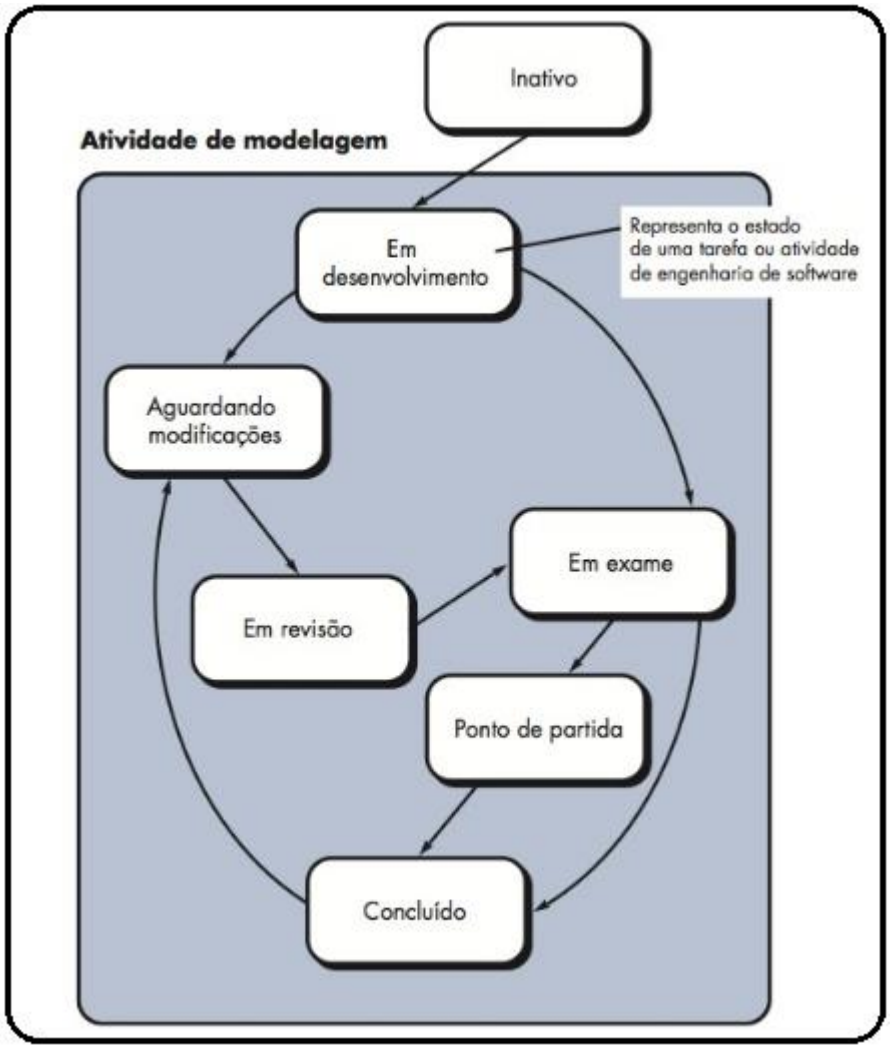
No modelo concorrente é levado em consideração que as fases do desenvolvimento do software não devem ocorrer sequencialmente, mas sim concorrentemente exercendo, portanto atividades simultâneas. Esse modelo de desenvolvimento de software é baseado em projetos de curto prazo, em que além do prazo reduzido há uma ligeira preocupação com a satisfação do cliente no resultado final.

Nesse modelo se fazem necessárias modificações constantes no projeto, possibilitando à equipe de software representar elementos concorrentes e iterativos. A modelagem concorrente pode ser aplicada a qualquer tipo de desenvolvimento como afirma Pressman:

> A modelagem concorrente se aplica a todos os tipos de desenvolvimento de software e fornece uma imagem precisa do estado atual de um projeto. Em vez de limitar as atividades, ações e tarefas da engenharia de software a uma sequencia de eventos, ela define uma rede de processos. Cada atividade, ação ou tarefa na rede existe simultaneamente com outras atividades, ações ou tarefas. Eventos gerados em um ponto de processos disparam transições entre os estados (2011, p. 68).

Perante isso, pode-se observar na figura 4 as transições de estados da modelagem concorrente.

Figura 4 – Modelagem concorrente.



Fonte: PRESSMAN 2011, p.67.

Com base na figura 4, pode-se observar que os processos não seguem sequencialmente suas fases de evolução do projeto, pelo contrário, as fases fluem de forma independente da espera umas das outras. Por essa razão, o modelo concorrente é indicado para projetos de curto prazo, onde há a necessidade de um processo rápido e eficaz.

2.3 METODOLOGIAS ÁGEIS DE DESENVOLVIMENTO

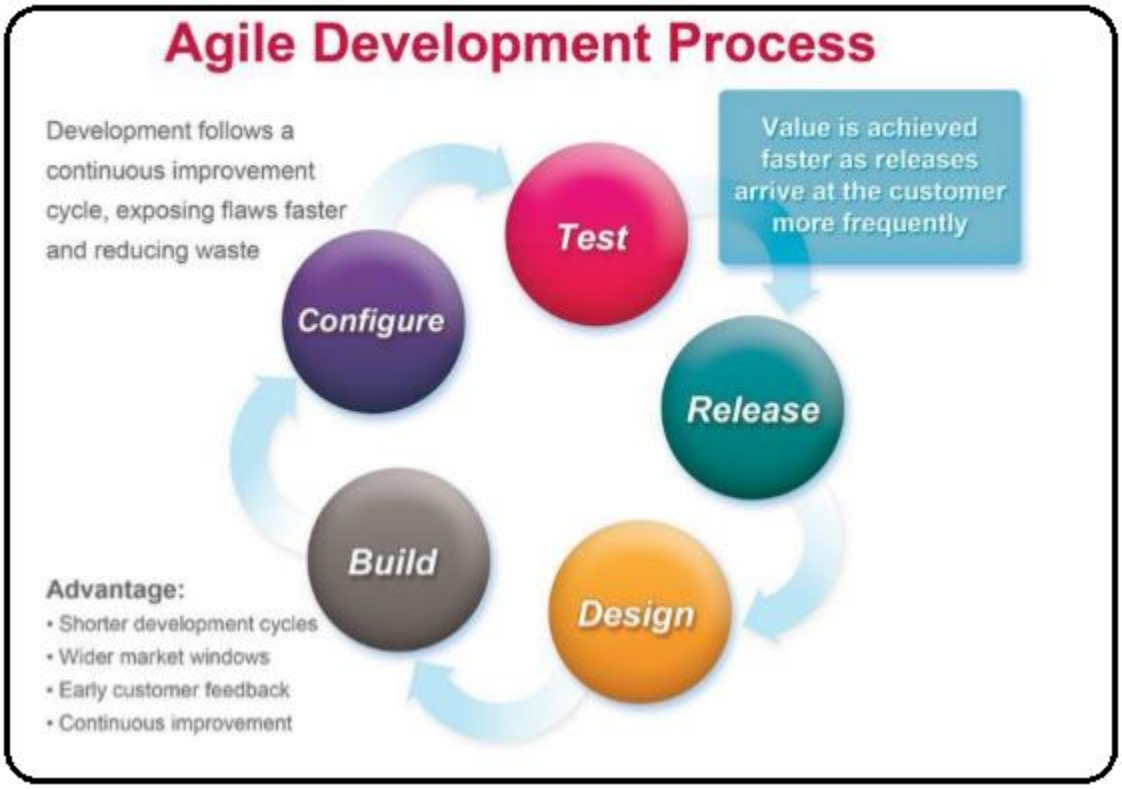
Metodologias de desenvolvimento ágeis estão sendo utilizadas para solucionar os desafios presentes na atualidade, que são os desafios de produzir softwares no menor tempo possível e com um nível de qualidade e confiabilidade aceitável para o cliente final.

Pressman cita que a engenharia de software ágil é uma combinação: “A engenharia de software ágil combina filosofia com um conjunto de princípios de desenvolvimento” (2011, p.81), em que a filosofia é a parte que defende a satisfação do cliente e um processo de entrega parcial para o cliente de modo a garantir que o projeto está seguindo o rumo correto em seu ciclo de vida.

Os princípios de desenvolvimento ágil zelam pela entrega dentro dos prazos estipulados nos projetos da empresa em questão, como também pelos processos de comunicação entre a equipe envolvida no projeto (PRESSMAN, 2011).

A metodologia ágil de desenvolvimento fornece a integração constante na codificação e nos testes, essas duas atividades ocorrem de forma síncrona e interagem em todo o processo, como é demonstrado na figura 5.

Figura 5 – Modelo ágil de Desenvolvimento.



Fonte: AGILE (s.d)

Dentre os métodos ágeis mais populares no mercado podemos citar o *Scrum*, *Crystal Clear*, *Adaptive Software Development*, *Feature Driven Development* (FDD), *Dynamic Systems Development Method* (DSDM) e principalmente a Programação Extrema (*Extreme Programming – XP*) metodologia mais difundida atualmente.

Todos esses métodos de desenvolvimento buscam o refinamento de qualidade na elaboração de sistemas e sua entrega dentro dos prazos estipulados junto ao cliente (PRESSMAN, 2011).

A abordagem de programação extrema é baseada em um conjunto de princípios que visam a agilidade na produção de software. A XP em sua essência defende cinco princípios básicos (SOMMERVILLE, 2007):

- Comunicação: Toda a equipe envolvida no *job* deve interagir constantemente com a finalidade de todo o grupo possuir a maior troca de informações possível no processo;
- Simplicidade: Procurar fazer o básico em cada fase do desenvolvimento, sem querer revolucionar a cada projeto, e ao invés disso, agregar sempre melhorias;

- *Feedback*: Buscar o maior número de *feedbacks* possível juntamente com o usuário, a modo de garantir o rumo correto do *job*.
- Coragem: Coragem para sanar as dúvidas juntamente com a equipe e com o cliente;
- Respeito: Respeito pelas etapas do XP e por todos os membros envolvidos.

Na XP todas as fases do projeto requerem comunicação com o cliente, de preferência de forma presencial no decorrer do desenvolvimento do sistema, ou seja, busca-se desta forma manter um elo constante na comunicação entre o cliente e os desenvolvedores para que sejam minimizados ao extremo os erros recorrentes nos levantamentos de requisitos.

2.4 QUALIDADE DE SOFTWARE

Atualmente no mercado do mundo todo o software ocupa uma posição importante ao apoio dos negócios das empresas, sendo em algumas organizações parte intrínseca. Negócios que priorizem a internet não conseguem sobreviver sem o apoio de um software focado no seu nicho de mercado, sendo que devido ao seu caráter decisivo sua disseminação tornou-se inevitável. Com essa popularização do software, surgiu também um número expressivo de defeitos que afetam a usabilidade e funcionalidade impactando fortemente a confiabilidade do programa (RIOS, 2013).

O aspecto de confiabilidade de um software para Pressman (2011, p.395) é um elemento importante na qualidade global do sistema, isso se deve ao fato de que: “Se um programa falhar frequentemente e repetidas vezes, pouco importa se outros fatores de qualidade de software sejam aceitáveis”. O sistema deve passar para o cliente a segurança necessária para que o mesmo não tenha nenhum tipo de receio ao utilizar todas as suas funcionalidades, para que sua experiência de uso seja a mais satisfatória possível.

Para alcançar a melhoria constante na produção de sistemas, a engenharia de software tem elaborado e aprimorado técnicas para garantir tal evolução. Nos últimos anos a área de TI teve contato com metodologias de desenvolvimento tradicionais, onde o sistema é, antes de qualquer coisa, planejado para depois ser construído e testado (BECK, 2000).

Porém, segundo Pressman (2011, p.387), essa forma de pensar no que se refere à qualidade de software está completamente defasada:

> Alguns desenvolvedores de softwares continuam a acreditar que a qualidade de software é algo sobre o qual começamos a nos preocupar depois que o código é gerado. Nada poderia estar tão distante da verdade! A garantia da qualidade de software, SQA (software quality assurance, muitas vezes denominada gestão da qualidade) é uma atividade universal aplicada em toda a gestão de qualidade.

A engenharia de software definiu padrões e procedimentos que devem ser seguidos no processo de produção de um software, mas a realidade da maioria das empresas é distinta do que orienta a engenharia. No cenário atual a estrutura organizacional na fase de testes se manteve a mesma de anos atrás, onde os programadores eram responsáveis por desenvolver o sistema e testá-lo, constatação essa feita por Rios (2013, p.55):

> A Estrutura organizacional para testes não tem se modificado. Quase todos os estágios de testes ainda são feitos pelos desenvolvedores. Nem todos eles gostam de testar o software. Outros não possuem o perfil de testador e/ou não são qualificados para executar as atividades de testes. A maioria dos desenvolvedores é muito bem treinada para desenvolver software, mas pouca atenção é dada para a capacitação desses profissionais no processo e nas técnicas de testes. Por outro lado, poucas organizações adotam equipes/ organizações de testes independentes.

Com base nesta constatação, os desenvolvedores de software exercem papel fundamental no que diz respeito à função de teste de um produto de software computacional, e ignorar essa realidade pode comprometer drasticamente o objetivo de alcançar um alto nível de qualidade. Perante isso, é extremamente importante para o sucesso de um projeto aperfeiçoar a participação dos mesmos no que diz respeito ao processo de elaboração, construção e validação dos requisitos funcionais.

2.5 CASOS DE TESTE

Os casos de testes podem ser definidos como seleções de entrada de dados para os testes na qual é esperada uma determinada saída pelo software. Os dados de entrada dos casos de testes serão os planejados para testar o sistema que podem ser feitos de forma automatizada. Casos de testes só podem ser definidos

por pessoas que tenham plena noção das funcionalidades do sistema e de como o mesmo deve se comportar em cada nível de atividade (SOMMERVILLE, 2007).

Casos de testes podem servir como documentação do sistema para os desenvolvedores no caso da metodologia de desenvolvimento guiado a testes. Através deles fica simplificada a busca por informação de como o sistema foi planejado para funcionar, sem ser necessário para os desenvolvedores acessarem toda a documentação do sistema (BECK, 2010).

2.6 TESTES UNITÁRIOS

Testes unitários é uma modalidade de teste de software que se baseia em testar unidades lógicas do sistema, seu objetivo é dividir todos os testes necessários em pequenos processos (unidades), onde cada unidade será testada e validada. Os testes unitários ocorrem na fase de desenvolvimento do software, pois são os testes codificados que são responsáveis por realizar a verificação da funcionalidade exigida.

As unidades de testes podem ainda ser refatoradas para garantir uma eliminação de defeitos de programação e duplicidade de código, conforme citado por Filho:

> As unidades de código fonte produzidas podem ser submetidas a uma revisão informal de código, para eliminar os defeitos que são introduzidos por erros de digitação ou de uso da linguagem. Essa revisão também deve ser feita pelos autores, possivelmente com a ajuda de outros programadores. As unidades de código fonte são transformadas em código objeto por meio da compilação. As revisões informais podem ser substituídas pela prática da programação em pares, na qual cada unidade é produzida por um par de programadores, que se revezam nas funções de digitar e verificar em tempo real o que está sendo programado (2008, p. 411).

Os testes unitários precisam de um planejamento prévio à sua execução, para que os desenvolvedores que irão executá-los tenham domínio pleno sobre as funcionalidades que determinada unidade deve cobrir, e tenha ciência de como aplicar as entradas e saídas de dados para o teste. Portanto antes da aplicação do conceito de testes unitários devemos elaborar os seus respectivos casos de teste para garantir a eficiência dos testes de unidades (SOMMERVILLE, 2007).

Por meio dos testes unitários podemos validar as funcionalidades em tempo de desenvolvimento do produto, sem ser necessário esperar uma versão final ou protótipo. Essa validação na fase de desenvolvimento auxilia no quesito de qualidade de software, devido ao fato de um *bug* ou requisito incorreto for descoberto, sua correção será mais facilmente aplicada sem ser necessário chegar ao contato com o cliente.

2.7 TESTES AUTOMATIZADOS

Testes automatizados como o nome já sugere é o conceito de utilizar um software para realizar o processo de testes. As automatizações de testes agregam inúmeros benefícios ao projeto e para a empresa que faz o uso da mesma, pelo simples fato de que uma vez escrito o teste automatizado, o mesmo pode ser reutilizado em outros projetos futuros sem ser necessária sua nova formulação.

Um exemplo da reutilização de um teste automatizado pode ser o teste de uma unidade de código que calcula a idade de uma pessoa, nos testes tradicionais toda a vez que essa funcionalidade fosse implementada, a mesma deve ser submetida a um novo teste, no teste automatizado bastaria escrever o teste apenas uma vez e reaproveitá-la em necessidades futuras.

Para realizar os testes automatizados é necessário o uso de alguma ferramenta que faça o acionamento dos testes de código fornecendo as suas entradas e que analise suas respectivas saídas, Filho define os testes automatizados como:

> Testes automatizados são aqueles que são executados por acionamento do item de teste por uma ferramenta, que gera as entradas de teste por meio de controladores de teste. Um controlador de teste (test driver ou test harness) é um módulo de software que invoca um módulo sob teste, frequentemente provendo entradas de teste, controlando e monitorando a execução e relatando os resultados. A geração das entradas pode ser feita por algoritmos ou por recuperação de arquivos de dados de teste, derivados das respectivas especificações. Testes automatizados podem ser considerados um caso específico de testes formais (FILHO, 2008, p.353).

Os testes automatizados surgiram juntamente com o conceito das metodologias ágeis que identificaram que os testes também necessitavam de maior agilidade no seu processo. Ao contrário dos testes automatizados, existem os

manuais que dependem de agentes humanos e são executados de maneira formal ou informal, o que para a empresa tem um custo elevado.

O conceito para os testes automatizados defende que não faz sentido manter recursos humanos com o intuito apenas de rodar testes de forma manual, por isso os processos desses que podem ser realizados de forma automática sem a intervenção humana deve sempre ser aplicado.

Atualmente no mercado existem diversas ferramentas para o uso de testes automatizados, sendo que no presente trabalho, será aplicado o *framework DUnit* específico para a linguagem de programação *Pascal*. Todos os *frameworks* disponíveis têm o conceito de formar suítes de testes, que são pacotes utilizados para a automatização dos processos de teste. Através dessas suítes de testes podemos gerar relatórios de problemas encontrados e execuções bem sucedias do pacote de testes.

2.8 DESENVOLVIMENTO GUIADO POR TESTES

A metodologia *Test Driven Development* faz parte das metodologias ágeis de desenvolvimento, visando a produção mais rápida e eficiente na qualidade do software. O Desenvolvimento guiado por testes é uma cultura de desenvolvimento de software que surge para mudar o paradigma tradicional conhecido por desenvolvedores e engenheiros de software onde primeiramente se desenvolve o software para posteriormente realizar o teste do mesmo (BECK, 2010).

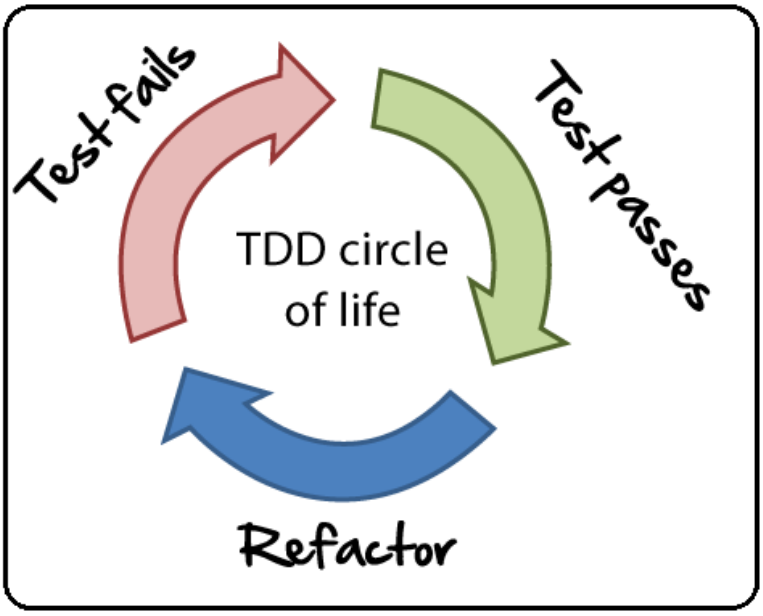
O conceito de TDD é extremamente simples, primeiramente testa-se para depois desenvolver, nessa metodologia o processo é invertido primeiramente o desenvolvedor escreve um caso de teste para depois desenvolver o código (KOSKELA, 2008).

Cohn (2011, p.179) define que o desenvolvimento guiado a testes não é restrito apenas a uma prática de desenvolvimento: “É apropriado considerar o TDD tanto como uma prática de projeto quanto de programação. Afinal, os testes que um programador escreve e a ordem em que eles são escritos guiam o projeto e o desenvolvimento de um requisito”. Nessa definição fica evidente a importância e participação que a metodologia exerce em todo o projeto.

A figura 6 demonstra o ciclo de desenvolvimento guiado a testes, onde primeiramente o teste é escrito, em segundo plano faz-se com que ele seja bem

sucedido (passar no teste) e por último o código é refatorado, melhorando-o e eliminando as redundâncias encontradas.

Figura 6 – Ciclo de vida Test Driven Development.



Fonte: TDD (s.d)

No conceito de TDD a qualidade de um sistema computacional de alto nível não pode ser alcançada no modo tradicional de desenvolver e depois testar, pois a qualidade deve estar presente em todas as fases do projeto. O desenvolvimento guiado a teste é uma técnica que visa o aumento da qualidade interna do software que geralmente é responsável por grande parte dos erros com maior severidade (KOSKELA, 2008).

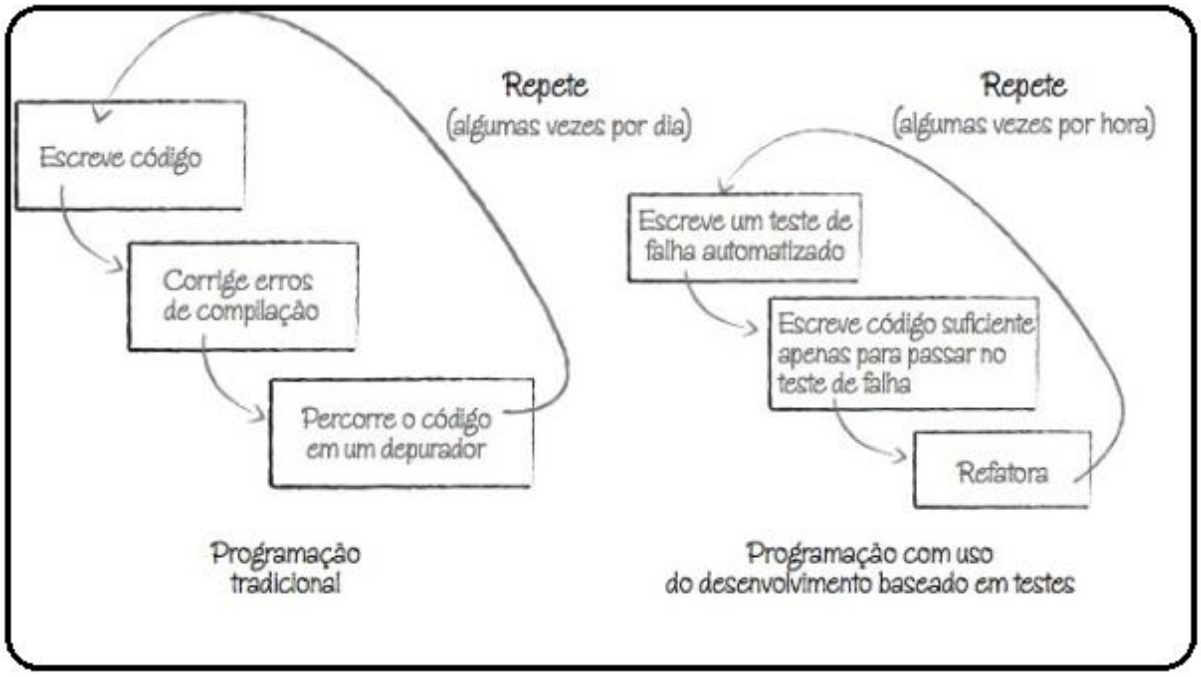
O TDD traz para os programadores o benefício de um código mais limpo e robusto, pois através dele é possível automatizar testes, aprimorá-los e eliminar códigos redundantes ou em duplicidade. Outra vantagem é o *feedback* constante no projeto sendo possível fazer ajustes de funcionalidades em tempo de desenvolvimento. Esses benefícios oferecidos tornam essa metodologia muito importante conforme cita Cohn:

> Acho o desenvolvimento baseado em testes (TDD, teste-driven development) inestimável. Uma das principais razões é que ele garante que nenhum código não testado entre no sistema. Se todo o código deve ser escrito em resposta a um teste de falha, então, mesmo se não fizermos nada mais, pelo menos conseguiremos uma cobertura completa do código com o TDD. Você pode achar que uma abordagem de “teste imediatamente

> após" obteria o mesmo resultado. No entanto, descobri que, quando os programadores se comprometem a escrever seus testes de unidade "logo após" terminarem de implementar um requisito, frequentemente não fazem isso. A pressão para começar a programar o próximo requisito pode ser tremenda. Logo, os programadores tendem a escrever testes apenas para um subconjunto da nova funcionalidade ou a colocar o teste em uma lista de coisas para fazer depois e descobrir que o depois não chega nunca (2011, p.179).

A figura 7 mostra um comparativo entre a prática de programação tradicional e a programação baseada em testes, onde se tornam evidentes as diferenças no ciclo de programação, visto que no TDD devido ao processo de refatoração torna a unidade de código programada mais robusta.

Figura 7 – Comparativo Programação Tradicional e TDD.



Fonte: COHN 2011, p.178.

O TDD "força" o desenvolvedor a realizar testes constantes em suas linhas de código para validar e aprimorar cada unidade, e para aplicar essa metodologia é obrigatório o uso de um *framework* que permita realizar os testes unitários. No presente trabalho foi utilizado o *framework DUnit* voltado para a linguagem de programação *Delphi*, através desse *framework* é possível gerar relatórios sobre os testes de uma suíte específica ou unidade. Outro benefício deixado pelo TDD aos programadores é poder usar esses testes unitários como uma documentação do software, visto que para um desenvolvedor com base em seus conhecimentos

técnicos fica extremamente simples analisar esses testes e interpretar as funcionalidades do sistema.

O desenvolvimento orientado a testes oferece robustez ao projeto e uma melhor qualidade no momento dos testes na fase de desenvolvimento, além disso, pode ser aplicado isoladamente para os programadores ou combinado com outras metodologias como o *Scrun,* agregando ainda mais benefícios ao processo como um todo.

2.9 LINGUAGEM DE PROGRAMAÇÃO PASCAL

A linguagem de programação *Pascal* foi criada por volta de 1970 para ser uma linguagem estruturada, visto que existem muitos dialetos sobre a linguagem. Porém seu objetivo inicial era o de ensinar a programação de forma estruturada e criar softwares de qualidade para a realidade da época (GHEZZI, 1997).

A linguagem *Pascal* utiliza os módulos providos como procedimentos e funções que podem ser utilizadas para implementar a modularização procedimental, pois nessa estrutura a linguagem suporta apenas programação de forma sequencial. A figura 8 demonstra a sintaxe de uma função estruturada da linguagem *Pascal*:

Figura 8 – Sintaxe da Linguagem Pascal.

```
function sonumeros( val : string ):string;
var ind : integer;
    mat : set of char;
    aresult : string;
begin
    aresult := '';
    mat:=['0','1','2','3','4','5','6','7','8','9'];
    for ind := 1 to length(val) do
    begin
        if (val[ind] in mat)then
         aresult := aresult + val[ind];
    end;
    result := aresult;
end;
```

Fonte: Autor, 2016.

O *Pascal* foi muito utilizado nos anos 80 pela IDE *Delphi* para a construção de softwares comerciais, contribuindo fortemente para sua disseminação no mundo inteiro. Anos depois foi necessária a evolução desta linguagem, que se tornou uma linguagem orientada a objetos (LAUREANO, 2010).

A linguagem então ficou comercialmente conhecida pela IDE *Delphi*, que implementava seus recursos. Após esta se tornar orientada a objetos o *Delphi* que até então era apenas uma IDE de programação, realizou modificações na estrutura do *Object Pascal*, passando a ser considerada também como uma linguagem de programação *Delphi Programming Languagem* (LEÃO, 2003).

2.10 LINGUAGEM DE PROGRAMAÇÃO DELPHI E IDE DELPHI 7

No presente estudo foi utilizada a linguagem de programação *Delphi*, devido ao fato de que a empresa analisada no experimento utiliza o *Delphi* na grande maioria de seus projetos e propor a mudança para outra linguagem, além de ser muito trabalhoso iria contra a proposta desta pesquisa.

O *Delphi* surgiu em 1993 com a *Borland*, empresa responsável pela marca na época que desenvolveu um projeto para a criação de um ambiente visual, onde seria possível criar aplicações para o sistema operacional *Windows* utilizando a linguagem *Object Pascal*, esse projeto foi denominado *Delphi* (LEÃO, 2003).

No ano de 1995 foi disponibilizado no mercado a primeira versão da IDE, o *Delphi* 1 que nessa época representou grande avanço no desenvolvimento de software por possuir tecnologias pioneiras, como componente para conexão com banco de dados, programação orientada a objetos e ambiente de desenvolvimento baseado no conceito RAD (*Rapid Application Development*), que é um conceito de desenvolvimento de software interativo e incremental que busca o ciclo de desenvolvimento em curto prazo (LEÃO, 2003).

A tabela 1 demonstra a evolução das IDEs *Delphi* e seus respectivos anos de lançamento e ganhos tecnológicos até a versão do *Delphi* 7 que será foco do presente estudo:

Tabela 1 – Versões da IDE Delphi até o Delphi 7

Versão da IDE	Ano Lançamento	Principais Avanços Tecnológicos
Delphi 1	1995	Suporte ao Windows 3.1 16 bits, compilação native, orientação a objetos, acesso aos principais bancos de dados do mercado na época
Delphi 2	1996	Suporte a 32 bits, Windows 95, Grids baseados em DataBases,Ole Automation e Herança Visual.
Delphi 3	1997	Debug de Dlls e pacotes, Tecnologia WebBroker, Interfaces COM.
Delphi 4	1998	Suporte ao Windows 98, Arrays Dinâmicos e Method Overloading.
Delphi 5	1999	Suporte profissional para o desenvolvimento WEB, XML, ADO, Melhorias significativas no Debug
Delphi 6	2000	Suporte total a Webservices, DBExpress e muitos outros novos componentes. Nesse ano foi apresentado também a versão Kylix, versão do Delphi pra Linux.
Delphi 7	2001	Ide Chamada de Delphi 7 Studio, aprimorou o suporte a WebServices e aplicações distribuídas com DataSnap, Integração com .Net e Linux.

Fonte: Autor, 2016.

Foi na versão 7 do *Delphi* que a ferramenta ficou mais conhecida e disseminada comercialmente no mundo inteiro, tendo diversas empresas que ainda fazem uso da mesma para criação e manutenção de seus softwares. O principal atrativo dessa *IDE* foi a agilidade com que se conseguia criar sistemas robustos e com um bom design para o cliente.

Nessa versão o *Delphi* passou ainda, a ser considerado também uma linguagem de programação por não fazer uso direto do *Object Pascal* e sim de modificações com melhorias feitas em cima do mesmo. Nesta pesquisa será utilizada a *IDE* de desenvolvimento *Delphi* 7 e, portanto a linguagem de programação *Delphi* (*Delphi Programming Languagem*) para o desenvolvimento de softwares baseado na metodologia de *Test Driven Development*.

2.11 FRAMEWORK DUNIT

O *DUnit* é um *framework* para testes unitários voltado para a linguagem de programação *Delphi*, e através dele é possível realizar testes de unidades em projetos programados de forma orientado a objetos. O *DUnit* não funciona em sistemas programados sequencialmente, ou seja, a unidade que será testada pelo *DUnit* deve obrigatoriamente ter sido escrita orientada a objetos.

O *Framework DUnit* funciona com base em uma classe a ser testada, onde indicamos os parâmetros de entrada para o teste e o resultado esperado com a execução da função da classe. Para cada classe que se deseja submeter o teste unitário, deve ser criada uma nova classe de testes.

Por meio do *DUnit* esse trabalho aplicou a metodologia de TDD em um projeto real da empresa estudada, gerando relatório dos testes efetuados e criando uma suíte de testes automatizados para projetos futuros que desejem fazer o seu uso.

2.12 ESTUDOS RELACIONADOS

No trabalho de pesquisa de Rassweiler (2015) foi realizado um estudo voltado para a melhoria no processo de qualidade de software. O trabalho acadêmico apresenta ainda um estudo de caso comparando três projetos de portais de *e-commerce* com um escopo semelhante, sendo todos portais de grande porte. Para comparar esses três projetos à acadêmica realizou um estudo comparando os resultados com base na quantidade erros (*Bugs*) encontrados em cada um. Como resultado da utilização de um ambiente ágil de prevenção a falhas foi evidenciado uma redução no tempo de elaboração dos casos de testes, como também uma redução no número de *Issues* e retrabalho do projeto que utilizou a metodologia ágil.

Outro trabalho relacionado ao presente estudo foi desenvolvido pelo acadêmico Amaral (2013), onde foi demonstrado que projetos que fizeram o uso de TDD no desenvolvimento de software diminuíram consideravelmente a quantidade de falhas no projeto final. Para chegar a essa constatação Amaral (2013) comparou projetos da *IBM* e da *Microsoft* utilizando a métrica *ISO* 9126, que define padrões de qualidade de software em sistemas computacionais.

O terceiro estudo semelhante foi apresentado em um artigo de autoria de Filho et all, (2012) evidenciando um estudo de caso sobre o aumento de qualidade de software com a utilização de TDD. Neste artigo foram comparados dois projetos, projeto A e projeto B, onde o projeto A fez uso da metodologia de desenvolvimento guiado a testes, ao contrário do projeto B não teve nenhum contato com essa metodologia.

Para realizar o estudo de caso comparativo entre o projeto A e B fez-se uso da métrica *Goal Question Metric*, a qual visa especificar primeiramente os objetivos para os projetos, e por último fornecer um quadro para a interpretação dos dados obtidos em relação aos objetos comparados. Como resultado foi comprovado que o projeto de engenharia de software que utilizou a metodologia de TDD obteve um índice consideravelmente menor de erros severos sobre o projeto tradicional.

2.13 GOAL QUESTION METRIC

A qualidade de software atualmente é um dos principais fatores para o sucesso de um produto computacional, devido a sua grande importância exercida na elaboração de sistemas, o tema qualidade de software tornou-se cada vez mais presente no mundo da tecnologia. Para aprimorar as técnicas de medição de qualidade surgiram diversas abordagens focadas no objetivo comum de avaliar de forma qualitativa os processos de desenvolvimento de software.

Independentemente da abordagem utilizada para avaliar qualitativamente um projeto de software, a técnica de avaliação deve ter a capacidade de ser adaptável aos objetivos e as particularidades do mesmo. Pressman destaca que centenas de métricas já foram criadas, sendo algumas muito complexas para implementar-se:

> Centenas de métricas já foram propostas para programas de computadores, mas nem todas proporcionam suporte prático ao engenheiro de software. Algumas demandam medições muito complexas, outras são tão esotéricas que poucos profissionais do mundo real têm qualquer esperança de entende-las, e outras ainda violam as noções intuitivas básicas do que é realmente um software de alta qualidade (PRESSMAN, 2011, p.542)

Com base nesta afirmação o presente trabalho fez uso de uma métrica de conceitos simples e de fácil adaptação para a proposta de pesquisa realizada.

Nesse sentido, foi definida a utilização da métrica *Goal Question Metric*, um paradigma baseado em definir os objetivos para o projeto, ou seja, traçar as metas

para os dados que serão aplicados nos objetivos operacionais do projeto, para que posteriormente esses dados sejam utilizados para a criação de um quadro para analisar as informações obtidas e compará-las com os objetivos traçados.

O GQM é norteado pelo conceito de que para uma empresa poder organizar e analisar um determinado tema como a qualidade de um software, ela deve antes de tudo especificar os objetivos pretendidos para os seus projetos (FERNANDES, 1995).

O modelo de avaliação do *Goal Question Metric* pode ser resumido em três importantes etapas:

- Objetivo: um objetivo é definido para um projeto em questão;
- Questão: define caminhos para alcançar o objetivo determinado, nessa etapa é definido como chegar ao objetivo final da avaliação;
- Métrica: um conjunto de dados é relacionado a cada pergunta para respondê-la de forma qualitativa.

2.13.1 Objetivo

Evidenciar que o projeto da empresa que utilizou o TDD apresentou um ganho de qualidade através da redução de erros e de funcionalidades incorretas.

2.13.2 Questões

Para alcançar o objetivo de questões de analises, sobre quais propósitos são relevantes para o sucesso do projeto, as seguintes questões foram propostas:

- O projeto que utilizou a metodologia de TDD obteve uma qualidade de software superior ao projeto que não utilizou TDD.
- O projeto que fez uso do TDD apresentou menor quantidade de erros sobre o projeto que não fez uso.
- O projeto que utilizou TDD obteve um melhor levantamento de requisitos e consequentemente uma redução de retrabalho.

2.13.3 Métricas

Para responder as questões pertinentes ao objetivo da avaliação qualitativa surgem as métricas, no *Goal Question Metric* as métricas têm a finalidade de mostrar como serão respondidas as questões para atingir o objetivo da pesquisa.

As métricas fornecem as ferramentas necessárias para aplicar o estudo, como por exemplo, tecnologias materiais e métodos utilizados no decorrer do projeto. Para que a comparação qualitativa dos softwares seja o mais preciso possível, o GQM sugere a escolha de métricas semelhantes e a utilização de prazos próximos.

Grandes empresas quando utilizam essa metodologia procuram estipular um prazo máximo de tolerância de tempo entre os projetos, além de buscar a máxima similaridade com as tecnologias usadas no desenvolvimento.

3 METODOLOGIA

O estudo proposto pode ser considerado como uma pesquisa bibliográfica e um estudo de caso, isso devido ao fato de que para a elaboração do mesmo se fez necessário buscar referências bibliográficas sobre o tema do projeto, e os conceitos e estudos realizados anteriormente na área. Andrade (2010, p.25) cita que a pesquisa bibliográfica é fundamental em um trabalho de pesquisa: “Ela é obrigatória nas pesquisas exploratórias, na delimitação do tema de um trabalho ou pesquisa, no desenvolvimento do assunto, nas citações, na apresentação das conclusões”.

O referencial teórico utilizado para a construção do trabalho serviu de embasamento para a implementação dos conceitos e ferramentas da metodologia de desenvolvimento guiado a testes. Essa pesquisa bibliográfica realizou-se por meio de livros, artigos e apresentações de palestras que foram ministradas sobre o tema de *Test Driven Development*.

O estudo de caso se fez aplicando a metodologia de desenvolvimento guiado a testes em um ambiente real de produção de software em uma empresa do varejo que não possui nenhuma metodologia ágil de desenvolvimento. A metodologia de TDD foi inserida diretamente á equipe de desenvolvedores da empresa, que nesse processo fez uso dos conceitos e ferramentas pesquisadas anteriormente no referencial teórico. O objetivo do uso do TDD é realizar um comparativo entre dois projetos da empresa, um com a sua utilização e outro seguindo o ciclo tradicional de desenvolvimento.

Com a conclusão desse estudo de caso conseguiu-se obter dados sobre o projeto testado, para se tornar possível a demonstração através do mesmo quais foram os reais ganhos que essa metodologia proporcionou ao produto final. Os dados do experimento foram gerados e analisados com base na métrica de comparação *GOAL QUESTION METRIC*, um paradigma orientado a metas que define normas e padrões de analises comparativas entre projetos para que sejam evidenciadas suas diferenças.

4 MÉTRICAS E RESULTADOS

4.1 CENÁRIO ATUAL

O presente estudo realizou um comparativo entre dois projetos reais da mesma empresa com o intuito de demonstrar os reais ganhos de qualidade de software do projeto que utiliza *Test Driven Development* sobre o projeto que não utiliza nenhuma metodologia ágil de desenvolvimento.

Atualmente na empresa estudada que não terá seu nome divulgado por pedido da mesma, em sua estrutura de TI não existe equipe especializada em testes de softwares, como também não se faz utilização de nenhum tipo metodologia ágil de desenvolvimento de software. Os softwares produzidos pela empresa visam atender as necessidades internas dos colaboradores, ou seja, sistemas de frente de caixa para as lojas e demais programas computacionais para rotinas administrativas.

Os projetos na área de tecnologia da informação da empresa são definidos por meio de projetos descritos e assinados em reuniões na sede administrativa, em que fazem parte gerentes das áreas interessadas na solução computacional, um ou mais analistas de sistemas que preenchem um formulário fazendo os levantamentos de requisitos das funcionalidades do sistema e o gerente de TI. O modelo de projeto pode ser visualizado no Anexo I.

Depois de feita a reunião o projeto é assinado e repassado para o gerente de TI que realiza uma reunião com os programadores que terão a responsabilidade de projetar, desenvolver e testar o sistema. Os programadores em caso de dúvidas possuem total acesso aos analistas que participaram da reunião e os gerentes das áreas interessadas.

Os prazos de entrega do software são definidos pelo gerente de TI, que com base em um histórico de projetos anteriores estipula o tempo necessário de desenvolvimento. Depois de pronto e testado, o sistema é colocado em produção em uma das lojas da companhia com o objetivo de obter um *feedback* dos colaboradores que preenchem um formulário avaliando o software, lhe atribuindo notas de qualidade e fazendo observações de melhorias. O formulário de avaliação do software pode ser visualizado no Anexo II.

Com base no formulário padrão já existente na empresa preenchido por uma ou mais lojas, se o resultado do *feedback* for satisfatório e o sistema não apresentar

nenhum tipo de erro severo ou *bug* de funcionamento, a versão é enviada para todas as lojas da rede.

A proposta deste estudo de caso é manter a estrutura atual de produção de software da empresa, aplicando juntamente com os programadores na fase de desenvolvimento os conceitos de TDD em um projeto específico. Juntamente com esse projeto que fará uso da metodologia de desenvolvimento guiado a testes, outro projeto ocorrerá simultaneamente na forma tradicional sem o uso do TDD.

Ambos os projetos tiveram seu nível de complexidade semelhante e recursos humanos e tecnológicos equivalentes para que se consiga avaliar qualitativamente os projetos com base na métrica *Goal Question Metric*.

4.2 PROJETOS ANALISADOS

Nesta seção serão apresentados os projetos analisados na empresa estudada, obtendo um melhor detalhamento do projeto tradicional e do projeto com o uso do desenvolvimento guiado a testes. Os projetos analisados foram selecionados juntamente com a empresa visando obter a máxima similaridade possível entre complexidade, equipe e prazo de entrega, como também foi levada em conta a necessidade da solução computacional para a empresa.

4.2.1 Projeto tradicional

O projeto tradicional da empresa estudada foi realizado de forma corriqueira por parte da equipe de desenvolvimento, onde habitualmente o gerente de TI entrega o projeto de software para as pessoas designadas para a tarefa mediante uma reunião com as partes envolvidas. O projeto proposto foi o desenvolvimento de um módulo para a emissão de NFe (nota fiscal eletrônica) esse módulo contemplou a emissão de notas de devolução, venda, divergência, transferência e demonstração.

O módulo desenvolvido de NFe funciona por meio de comunicação direta com a SEFAZ do respectivo estado da loja, ficando a cargo do módulo a realização de todos os tratamentos de comunicação com a receita, inclusive os tratamentos de falta de internet (*offline*). Para a criação desse software foi utilizado o componente

Open Source ACBR voltado para a linguagem *Delphi* com a finalidade de acelerar o processo de desenvolvimento.

Para esta tarefa foram designados dois programadores com o foco exclusivo no projeto e um analista de sistemas para acompanhar o desenvolvimento e auxiliar na resolução das possíveis dúvidas relevantes. Durante todo o projeto tradicional foram realizadas reuniões semanais na segunda-feira, onde participaram toda a equipe de TI integrante do projeto, nessas reuniões o gerente de TI atualizava-se do andamento das atividades e ficava aberto para perguntas e sugestões para modificações e melhorias ou até mesmo correções necessárias.

4.2.2 Projeto com o uso do TDD

O projeto com a utilização do desenvolvimento guiado a testes foi aplicado na empresa estudada de forma nova, onde foi designado um tempo inicial para explicar para a equipe de desenvolvimento como funciona o TDD e como os mesmos deveriam proceder na fase de desenvolvimento do software. Depois de realizado o primeiro contato com a metodologia e feito alguns exemplos práticos do seu uso, o gerente de TI realizou a entrega por escrito do projeto que foi trabalhado por esses desenvolvedores, juntamente com uma reunião em seguida onde estavam presentes as partes envolvidas no projeto.

O projeto proposto foi o desenvolvimento de um módulo interligado ao sistema de vendas das lojas, onde o mesmo teria a responsabilidade de emitir NFCe (nota fiscal de consumidor eletrônica), esse módulo contemplou a emissão de cupons de venda para pessoas físicas e jurídicas.

O módulo de NFCe funciona de forma interligada com as informações do cupom fiscal tradicional, seu papel é validar os dados do cupom e realizar a criação do XML para envio até o respectivo SEFAZ do estado da loja. Para realizar esse envio do cupom o sistema realiza a comunicação direta com a receita, ficando a cargo do mesmo gerenciar uma possível queda de internet e consequentemente gerar um XML no formato de contingência, onde a loja terá no máximo 24 horas para envio ao SEFAZ responsável. Para a criação deste software como no projeto tradicional, foi utilizado pela a equipe de desenvolvimento o componente *Open Source ACBR* voltado para a linguagem Delphi com o objetivo de acelerar o processo de desenvolvimento.

Neste projeto semelhante ao tradicional foram designados dois programadores com dedicação exclusiva ao projeto, também foi cedido um analista de sistemas para acompanhar o projeto e auxiliar os programadores em possíveis dificuldades ou dúvidas existentes. Durante toda a fase de desenvolvimento do software até sua entrega final, o gerente de TI realizou reuniões semanais na segunda-feira, que na ocasião estavam presentes todas as pessoas envolvidas no projeto, nessas reuniões o gerente atualizava-se do andamento do desenvolvimento e no final da reunião ficava aberto para perguntas e sugestões para possíveis modificações visando a melhoria do sistema.

4.3 MÉTRICAS

As métricas fornecem inúmeros benefícios para a analise e acompanhamento dos projetos de engenharia de software, um deles é a possibilidade de poder medir a qualidade de um software. Por essa razão, as métricas visam equipar os projetos analisados com o intuito de demonstrar suas similaridades. Para uma melhor visualização das características de cada projeto, ambos foram colocados lado a lado nas tabelas 2 e 3.

Tabela 2 – Métricas de comparação dos projetos.

Descrição	Projeto Tradicional	Projeto TDD
Nível de complexidade (nota de zero a dez)	5	6
Analistas de Sistemas	1	1
Programadores	2	2
Gerente de Projetos	1	1
Casos de Testes	15	22
Duração (meses)	3	3
Prazo máximo (dias)	100	100

Fonte: Autor, 2016.

A tabela 2 evidencia a similaridade entre os projetos por possuírem o mesmo número de programadores e analistas de sistema, foi destacado também o grau de complexidade de cada projeto, onde o gerente de TI atribuiu uma nota de zero a

dez. O prazo para a entrega de ambos os softwares foi definido em noventa dias podendo chegar no máximo a cem dias, esse prazo se refere a entrega final do sistema na fase de produção.

Na tabela 3 destacaram-se as tecnologias utilizadas em cada um dos projetos, onde novamente os dados demonstraram o seu grau de proximidade possuindo apenas a diferença do uso do TDD e do *framework DUnit*.

Tabela 3 – Tecnologias e metodologias utilizadas.

Descrição	Projeto Tradicional	Projeto TDD
Linguagem de Programação	Delphi 7	Delphi 7
Banco de Dados	Firebird 2.0	Firebird 2.0
IDE	Delphi7	Delphi7
Framework Teste Unitário	Nenhum	DUnit
Metodologia Ágil de Desenvolvimento	Nenhum	TDD

Fonte: Autor, 2016.

Com base no que foi apresentado nas tabelas 2 e 3 fica explícita as similaridades entre os projetos, pois ambos possuem o mesmo número de analistas, programadores e gerentes de projetos. Todas as demais características dos projetos são semelhantes, exceto a utilização do TDD, justificando, portanto, uma comparação qualitativa que gera resultados com base no fator diferenciado “TDD”.

Para que o GQM obtenha êxito no objetivo da comparação dos projetos é necessário definir como e quais dados serão analisados para gerar os resultados esperados para a pesquisa. Visando a definição dos dados relevantes no processo, foi definido que os resultados serão gerados com base no número de erros (Bugs) encontrados em ambos os projetos, como também o número de funcionalidades indevidas ou não aplicadas ao software.

O levantamento de dados do GQM foi realizado em três etapas (desenvolvimento, entrega e produção), isso devido ao fato do projeto de pesquisa possuir um projeto tradicional de desenvolvimento de software e um projeto ágil com o uso do TDD. Na primeira etapa de coleta de dados os programadores realizaram

casos de testes, que são cenários elaborados juntamente com os analistas de sistemas com a finalidade de encontrar falhas no software.

Nessa primeira etapa do levantamento de dados foi executada a aplicação dos cenários de testes em fase de desenvolvimento do sistema, e reportada as eventuais falhas ou funcionalidades incorretas na forma de anotações em uma tabela do Excel conforme demonstra a figura 9.

Figura 9 – Figura cadastro de erros no desenvolvimento.

A	B	C	D	E	F
Projeto	**Erro**	**Descrição**	**Nivel do Erro**	**Programador**	**Data**
Tradicional	Erro Listagem de Produtos	Erro ao buscar produtos da Nfe	Medio	Kleber	08/08/2016
Tradicional	Erro login do colaborador	O sistema apresetou erro no momento	Baixo	Tiago	09/08/2016
Tradicional	Deletar nfe	O módulo de notas não comitava o dele	Medio	Tiago	09/08/2016
Tradicional	Código CFOP Errado	O sistema estava gerando o CFOP errad	Baixo	Tiago	09/08/2016
Tradicional	Erro ao excluir item da nota	O item excluido não atualizava no XML	Baixo	Tiago	09/08/2016

Fonte: Autor, 2016.

No projeto tradicional da empresa os erros foram reportados quando a equipe de desenvolvimento já possuía um protótipo do sistema para a realização dos testes propostos, enquanto que no projeto com o uso do TDD esses dados foram gerados diariamente na medida em que os programadores completavam o ciclo do TDD em cada unidade coberta do código.

A segunda etapa de coleta de dados foi realizada na fase de entrega do software depois de finalizado o desenvolvimento do software de ambos os projetos, as equipes apresentaram os módulos dos sistemas desenvolvidos para o gerente de TI e os analistas de sistema, e realizaram testes funcionais em cada um com o objetivo de validar e encontrar possíveis erros. Os erros e funcionalidades incorretas foram novamente registrados em uma planilha do Excel contendo ainda os retrabalhos necessários em cada um dos projetos.

Na última etapa da coleta dos dados o software foi colocado em produção em quatro filiais da empresa, sendo que em duas lojas com o módulo de NFCe desenvolvido com o TDD e as outras duas com o módulo de NFe desenvolvido no processo tradicional da empresa.

Nesse processo as filiais da empresa preencheram um formulário em que relataram os erros encontrados no sistema e também melhorias sugeridas para cada um dos módulos em questão, o modelo do formulário de relato de erros reportados pelas lojas pode ser visualizado no Anexo III, juntamente com o modelo de avaliação tradicional de implantação de novos softwares Anexo II.

Também foram obtidos dados cadastrados pela a equipe de suporte que quando identificava um erro cadastrava em uma planilha semelhante a da fase de desenvolvimento.

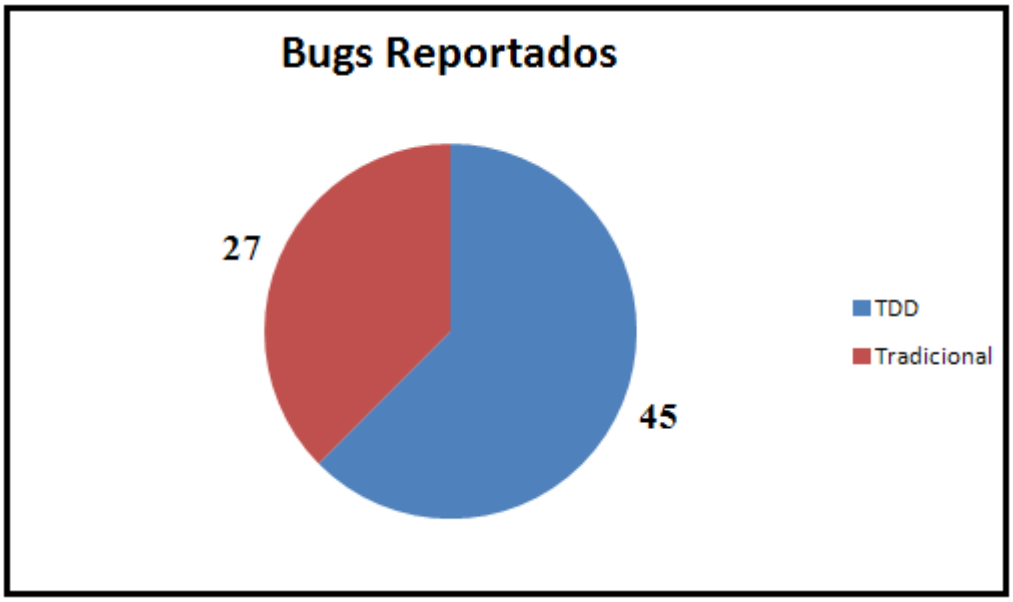
4.4 RESULTADOS

Neste capítulo estão apresentados os resultados obtidos com a comparação dos projetos. Os dados foram coletados em três fases de ambos os projetos, a primeira geração de resultados foi obtida no processo de desenvolvimento dos softwares, a segunda coleta foi realizada no momento de entrega dos sistemas e por último os dados foram coletados na fase de produção.

4.4.1 Bugs reportados no desenvolvimento dos softwares

Na fase de desenvolvimento dos softwares todos os erros encontrados pela a equipe de desenvolvimento foram registrados conforme pode ser visualizado no modelo presente na figura 9 do item 4.3. No levantamento realizado foi constatado que o número de erros identificados no projeto que fez uso do TDD foi 66,66% maior comparado ao projeto tradicional. No total foram reportados 45 *bugs* no projeto TDD e 27 *bugs* no projeto tradicional, a figura 10 demonstra os dados citados.

Figura 10 - Bugs Reportados no Desenvolvimento.



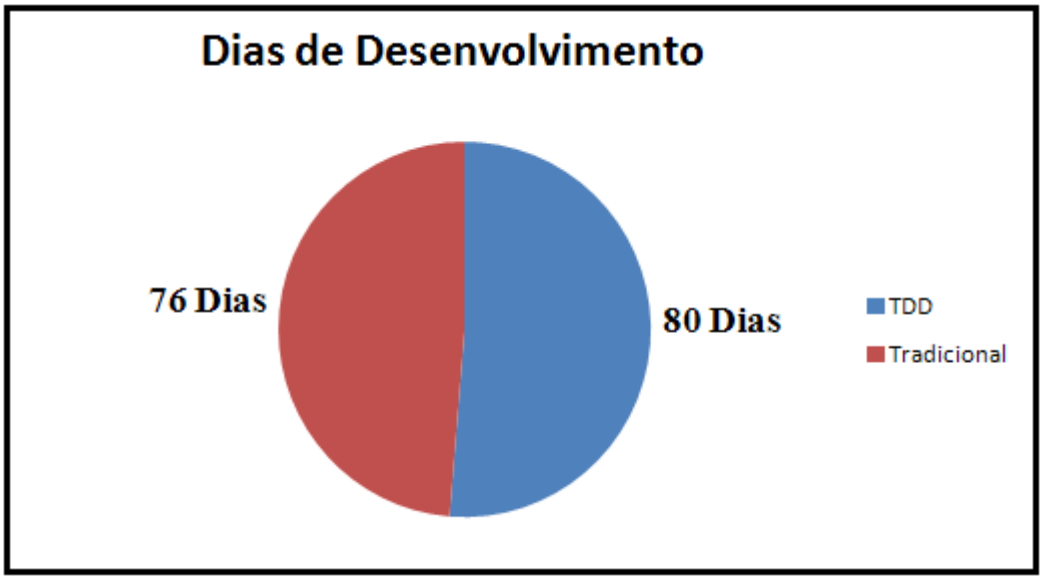
Fonte: Autor, 2016.

A quantidade maior de erros localizada no TDD na fase de desenvolvimento aponta uma maior eficiência nos testes aplicados pelo projeto com TDD em comparação com o tradicional. Essa afirmação se deve ao fato de que os erros filtrados e corrigidos na fase de elaboração do software, são falhas que deixarão de atingir o usuário final, ou seja, seu impacto e custo de correção é muito menor comparado aos erros descobertos na fase de produção do sistema.

Durante o desenvolvimento do projeto tradicional a equipe de programadores identificou três especificações funcionais incorretas e um erro de processo que não havia sido previsto, no total foram realizadas quatro mudanças no projeto. Já no projeto TDD os desenvolvedores localizaram sete especificações funcionais incorretas e dois processos não planejados, totalizando nove mudanças no sistema.

O desenvolvimento do módulo NFe no projeto tradicional foi concluído em setenta e seis dias de programação, quatro dias a menos que o projeto TDD que foi concluído em oitenta dias. A figura 11 esboça o tempo de desenvolvimento gasto em cada um dos projetos.

Figura 11 – Dias de Desenvolvimento.



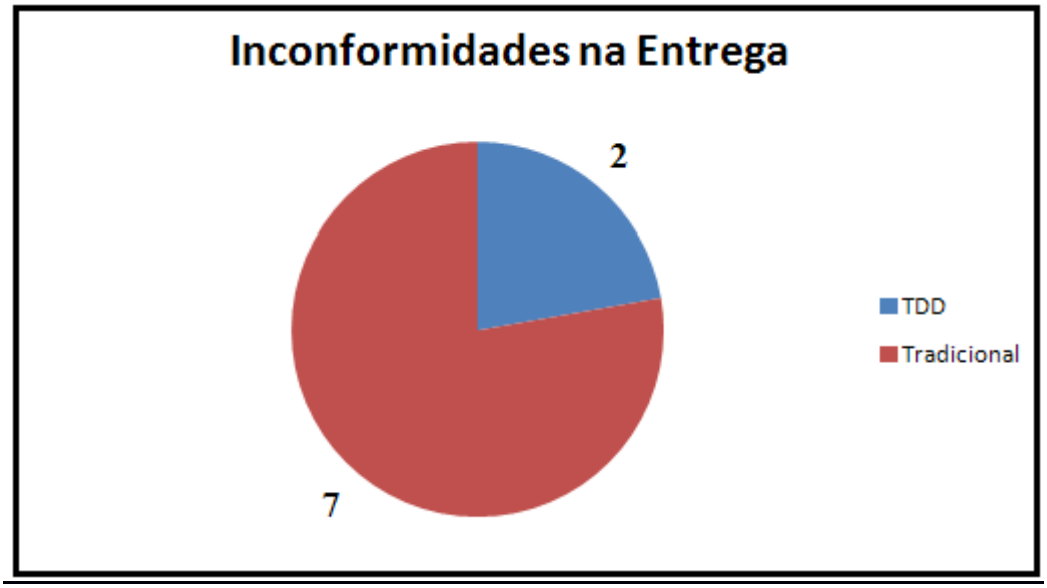
Fonte: Autor, 2016.

O prazo estipulado pelo gerente de projeto para a finalização do desenvolvimento foi no máximo 85 dias, ambos os projetos cumpriram o prazo estipulado. Entretanto o projeto desenvolvido com TDD apresentou uma perda de quatro dias comparado com o tradicional. Depois de finalizado o desenvolvimento e os testes desta fase, ambos os softwares disponibilizaram um protótipo para que o gerente de TI e os analistas pudessem testar o funcionamento dos módulos.

4.4.2 Avaliação da entrega dos softwares

Finalizado o desenvolvimento dos softwares as equipes de programadores do projeto tradicional e do projeto TDD, encaminharam um executável para o gerente de TI que analisou se os sistemas cumpriam todas as exigências funcionais. Nessa fase de teste, o gerente e os analistas submeteram o sistema há uma série de validações, toda a vez que uma inconformidade fosse percebida imediatamente era descrita para posteriormente ser corrigida pelo desenvolvimento. A figura 12 demonstra os dados obtidos na entrega de ambos os softwares.

Figura 12 – Resultados na Entrega.



Fonte: Autor, 2016.

No projeto TDD foi identificado um erro de sistema e uma funcionalidade incorreta que foi ajustada pela equipe de desenvolvimento, essas duas inconformidades levaram um dia para serem tratadas. No projeto tradicional foram localizados sete erros no total, sendo quatro erros de sistema e três funcionalidades incorretas apresentadas pelo módulo, ao todo foram necessários três dias para aplicar todas as mudanças devidas ao projeto.

Depois de feitos os ajustes nos dois projetos os softwares foram reavaliados pelo gerente de TI que não identificou problemas de funcionamento, e não foi necessário acrescentar mais dias nesta etapa.

4.4.3 Avaliação na fase de produção dos softwares

Após aprovado os softwares pelo gerente de TI na fase de entrega do sistema, os desenvolvedores geraram uma versão do módulo de NFe e NFCe. A versão do módulo NFe (projeto tradicional) foi enviado para duas lojas da empresa juntamente com o manual de funcionamento do sistema, no projeto com TDD o módulo de NFCe foi colocado em duas lojas com o manual. Na figura 13 são demonstrados os erros registrados pelo suporte da empresa na produção do software.

Figura 13 – Erros Reportados no Suporte.

Fonte: Autor, 2016.

A equipe de suporte registrou todos os chamados e ligações efetuadas pelas lojas onde os softwares foram implantados. Nesse processo foram registrados dois erros no módulo de NFCe (projeto TDD), um erro registrado pelo suporte interrompeu momentaneamente o funcionamento do sistema na loja, para solucionar esse problema a equipe do projeto TDD necessitou de 15 minutos. O segundo erro reportado no TDD não afetou o funcionamento do sistema para a loja, entretanto foi necessário gerar uma nova versão do software para realizar as correções, ao todo foram utilizadas 5 horas de desenvolvimento. No final da fase de produção o TDD apresentou um total de dois erros e o tempo gasto para as correções foi de um dia de desenvolvimento.

No projeto tradicional foram registrados ao todo dez erros, sendo que seis erros impactaram o funcionamento do módulo e quatro erros puderam ser resolvidos sem prejudicar a produção do software. Dois erros registrados pelo suporte tiveram seu nível de severidade alto e a loja não pode operar no módulo durante um dia inteiro, para solucionar os seis erros prioritários foi utilizado um dia de programação e mais um dia para resolver os outros quatro erros. Por fim, o tempo total gasto foi de dois dias para realização das alterações.

Os erros registrados no suporte podem ser visualizados no Anexo IV, onde consta cada problema encontrado nos projetos na fase de produção.

4.4.4 Avaliação das lojas

Terminada a fase de produção do software que ao todo ficou sete dias em avaliação no projeto TDD e no tradicional, as quatro lojas responderam os questionários de avaliação do sistema.

No formulário de relato de erros cujo modelo pode ser visualizado no Anexo III, tanto as duas lojas que utilizaram o software do projeto em TDD quanto às duas que o projeto tradicional foi aplicado, reportaram os mesmos erros já registrados pela equipe de suporte. Também foram registrados erros que os programadores e os analistas não conseguiram identificar ou replicar a ocorrência, esses registros foram investigados também pela equipe do suporte que entrou em contato com as lojas e as mesmas não evidenciaram mais a ocorrência do erro. Portanto o formulário de relato de erro das lojas não foi considerado para compor a quantidade total de erros em ambos os projetos.

Na avaliação geral do software conforme modelo presente no Anexo II, as duas lojas que utilizaram o módulo de NFCe (projeto TDD) atribuíram nota 9 e 10 para o quesito qualidade do sistema, as lojas ainda deram nota 10 para o tópico funcionalidade do software, concluído que o sistema cumpre todas as suas necessidades.

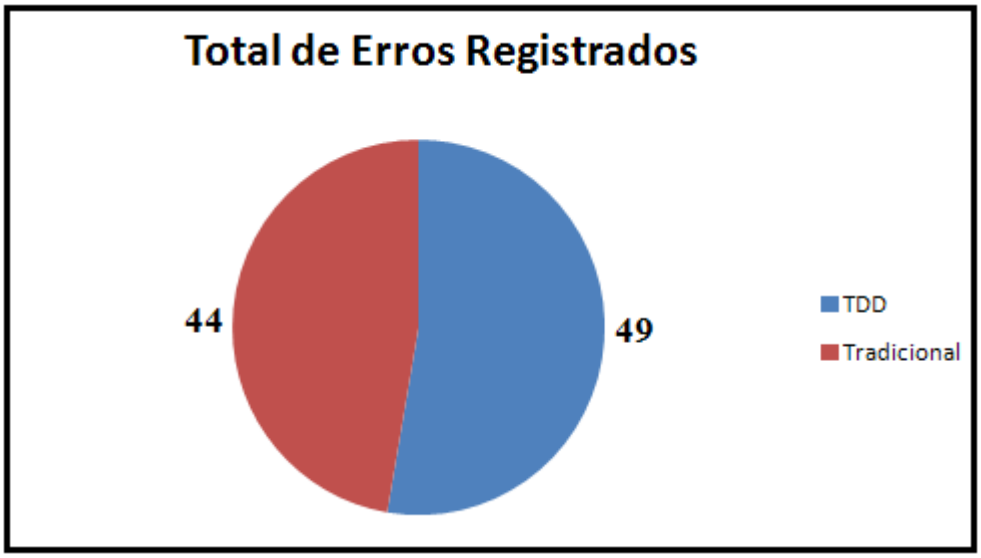
Nas duas unidades que utilizaram o módulo de NFe do projeto tradicional, as notas atribuídas para o quesito qualidade foram 7 e 8, na questão de funcionalidades as lojas deram 9 e 10.

Os formulários com a avaliação de cada loja e seu respectivo projeto podem ser visualizados nos Anexos V e VI.

4.4.5 Resultado final

Concluído o levantamento dos dados obtidos nas três fases de testes, podemos comparar os projetos e avaliar a quantidade de erros de cada um juntamente com o tempo gasto no desenvolvimento até sua entrega final. A figura 14 demonstra o total de erros reportados ao fim de cada projeto.

Figura 14 – Total de Erros.

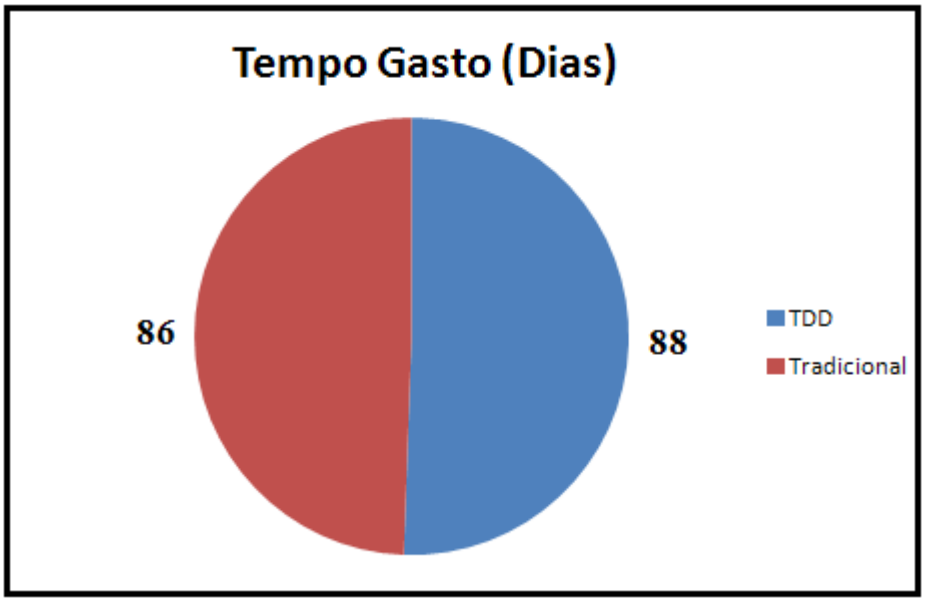


Fonte: Autor, 2016.

A quantidade de *bugs* encontrados no projeto TDD foi maior em comparação ao tradicional, contudo a maior parte dos erros reportados no TDD foram identificados antes de entrar em contato com o cliente final. Com base nisso, pode-se concluir que o TDD apresentou um nível mais acentuado de qualidade de software. Outro fator que ressalta essa constatação foi a avaliação realizada pelas lojas que atribuíram nota maior para o módulo de NFCe.

Ambos os projetos tiveram o prazo de noventa dias para a conclusão, os dois projetos conseguiram cumprir o tempo estipulado. Na figura 15 foi esboçado os dias utilizados para cada módulo.

Figura 15 - Total de Dias.



Fonte: Autor, 2016.

No projeto tradicional o tempo de conclusão do software foi dois dias menor que o projeto TDD. Nesse sentido, a metodologia de desenvolvimento guiado a testes não apresentou redução no tempo operacional, contudo o projeto cumpriu o tempo estipulado e ainda demonstrou um ganho significativo de qualidade.

Aplicando a métrica GQM constatou-se que o objetivo de evidenciar um ganho de qualidade através do uso do TDD foi alcançado, a quantidade de erros apresentado para a loja na fase de produção foi muito menor comparado ao tradicional. Na avaliação das lojas também foi constatado uma melhor avaliação na qualidade do software no projeto TDD, com o número de erros menor na produção do software a loja pode utilizar o sistema de maneira mais eficiente e confiável.

5 CONSIDERAÇÕES FINAIS

Finalizado o experimento, em relação ao objetivo inicial desse trabalho com a comparação dos projetos foi possível evidenciar um ganho de qualidade no projeto que utilizou a metodologia TDD, foi evidenciado também que o desenvolvimento guiado a testes destaca-se fortemente na fase de desenvolvimento do software, onde consegue filtrar uma grande quantidade de erros.

No aspecto do tempo operacional não foi obtido nenhuma comprovação do ganho de tempo com a metodologia TDD, os prazos em ambos os projetos foram cumpridos e os dias de desenvolvimento se equivaleram dentro da tolerância da empresa.

A equipe de desenvolvimento relatou que a metodologia agregou maior interação com os analistas e as áreas envolvidas, isso devido ao fato dos testes serem aplicados em toda a unidade e gerarem maior quantidade de dúvidas sobre as funcionalidades do software.

O gerente de TI se manifestou favorável pela utilização do TDD, e definiu sua adoção nos próximos projetos, destacou a maior qualidade do software percebida e o designer do código legado das refatorações do TDD. Os analistas destacaram os testes automatizados que foram legados da metodologia e que poderá ser facilmente adaptada e utilizada em projetos futuros.

Como pontos em que há a possibilidade de aprimoramentos e melhorias nesse estudo por meio de trabalhos futuros, pode-se citar:

- Realizar um estudo com maior tempo de projeto, com o intuito de analisar se através de longo prazo o TDD apresenta uma redução no tempo de entrega do software;
- Utilizar a metodologia TDD juntamente com o *Scrun* para obter maior eficiência no filtro de erros e processos incorretos;
- Aplicar o TDD em prazo mais longo analisando os testes automatizados e seus benefícios para a redução do tempo gasto;
- Acompanhar o processo pós-implantação do software, com o objetivo de analisar o surgimento de novos erros e custo de manutenção.

6 REFERÊNCIAS

AGILE. Disponível em: < http://www.gunnertech.com/2011/12/what-is-agile-development/>. Acesso em: 28 abr. 2016.

AMARAL, Nelson Senna do. **Minimizando as falhas na concepção dos sistemas com o Desenvolvimento Guiado por Testes**, São Paulo, Faculdade de Tecnologia de São Paulo. 2013.

ANDRADE, Maria Margarida de. **Introdução à metodologia do trabalho científico: elaboração de trabalhos na graduação**, 10ª edição. São Paulo: Atlas, 2010.

BECK, Kent. **Extreme Programming Explained**. Addison Wesley, Estados Unidos, 2000.

BECK, Kent. **TDD Desenvolvimento guiado por testes**. Porto Alegre: Artmed, 2010.

COHN, Mike. **Desenvolvimento de software com Scrum**: **aplicando métodos ágeis com sucesso**.Porto Alegre: Bookman, 2011.

FILHO, Wilson de Pádua Paula. **Engenharia de software**: fundamentos, métodos e padrões. 3ª ed. Rio de Janeiro: LTC, 2005.

FILHO, Maurício C. Vasconcelo, Juliherms L. Santos, Wylliams B. Silva, Ivonei F. **Um Estudo de Caso sobre o Aumento de Qualidade de Software em Projetos de Sistemas de Informação que Utilizam Test Driven Development** , São Paulo, VIII Simpósio Brasileiro de Sistemas de Informação. 2012.

FERNANDES, Aguinaldo Aragon. **Gerência de software através de métricas**. São Paulo: Atlas, 1995.

GHEZZI, Carlo e M. Jazayeri. **Programming Language Concepts**. John Wiley & Sons, New York, 1997.

KOSKELA, L.**Test Driven Practical TDD and Acceptance TDD for Java Developers**, Ed. Manning. 2008.

LAUREANO, Marcos. **Lógica de programação: Uma Abordagem em Pascal** 7ª edição. Ciência Moderna 2010.

LEÃO, M. **Borland Delphi 7 – Curso Completo**. Rio de Janeiro: Axcel Books 2003.

MCCLURE, Robert M. **The NATO Software Engineering Conferences.** Disponível em: <http://www.cs.ncl.ac.uk/old/people/brian.randell/home.formal/NATO/index.html>. Acesso em: 01/04/2016.

PFLEEGER, Shari Lawrence. **Engenharia de software: teoria e prática**. **2ª edição**. São Paulo: Pearson Education, 2004.

PRESSMAN, Roger S. **Engenharia de Software: Uma Abordagem Profissional** . 7ª Edição. São Paulo: McGraw-Hill, 2011.

PRESSMAN, Roger S. **Engenharia de software**. 6ª edição. São Paulo: McGrawHill/Nacional, 2006.

RASSWEILER, Caroline. **MELHORIA NO PROCESSO DE QUALIDADE: UM ESTUDO DE CASO** ,Passo Fundo, Instituto Federal Sul-Rio-Grandense. 2015.

RIOS, Emerson. MOREIRA, Trayahú. **Teste de Software**. *3*ª Edição. Rio de Janeiro: ALTA, 2013.

SCHACH, Stephen R. **Engenharia de Software: os paradigmas clássico e orientado a objetos**, 7ª edição: São Paulo: VitalSource Boolshelf Online, 2010.

SOMMERVILLE, Ian. **Engenharia de software**. 8ª ed. São Paulo: Pearson Addison-Wesley, 2007.

TDD. Disponível em: <http://www.agilenutshell.com/test_driven_development/>. Acesso em: 11 junh. 2016.

7 ANEXOS

ANEXO I – MODELO DESCRITO DE UM PROJETO REAL DA EMPRESA

Projeto: Planejamento financeiro de compra e venda

Objetivo: Projetar a necessidade de compras para as metas de vendas planejadas para os setores, grupos e subgrupos de produtos.
Atualizar os orçamentos automaticamente após aprovação do diretor da rede.

Objetivo a ser atingido:

FINANCEIRO:

- A partir do histórico de venda do mês é gerado a representatividade do grupo e do subgrupo na venda para o planejamento do ano seguinte;
- O percentual de impostos sobre os produtos do grupo e subgrupo é calculando conforme os valores dos impostos nas compras e nas vendas dos mesmos;
- O comprador poderá ajustar/informar os percentuais de representatividade do grupo e subgrupo. Também poderá incluir um grupo ou subgrupo novo e eliminar os que não irão mais trabalhar;
- O comprador irá informar a margem e o REV de cada subgrupo para o sistema calcular o restante dos valores (venda a custo, estoque final, valor das compras, cota financeira de compra);
- O comprador vai informar a meta geral de venda do setor e o sistema fará a distribuição dos valores nos grupos e subgrupos pela representatividade de cada um;
- Os valores da venda a custo, do estoque final, da compra e da cota financeira de compra, o sistema calcula automaticamente;
- O estoque inicial será atualizado a cada fechamento de mês. Os valores serão recalculados somente quando o comprador solicitar para recalcular;
- Os valores do grupo serão a soma dos valores calculados no subgrupo. Os valores do setor serão a soma dos valores do grupo;
- Os valores serão atualizados nos respectivos orçamentos quando o diretor da rede aprovar o planejamento de compra do comprador;

- As ordens de compra somente poderão ser realizadas quando tiver o valor de compra planejado e aprovado para o subgrupo;

Fórmulas:

Venda a custo = valor venda multiplica pela diferença entre 100 e a margem e multiplica pelo percentual de impostos na venda.

Venda custo = (venda * margem) * impostos.

Estoque final = venda a custo multiplica pelo VER multiplica por 2 e diminui o estoque inicial.

Estoque final = ((venda custo * REV) * 2) – estoque inicial

Valor compras = estoque inicial diminui a venda a custo e diminui o estoque final

Valor compras = (estoque inicial – venda custo) – estoque final.

Cota de compra financeira = valor da compra multiplicada pelos impostos sobre a compra.

Cota de compra financeira = valor compra * impostos.

Orçamentos a serem atualizados:

1) Orçamento 14 = valor planejado para a venda
2) Orçamento 44 = valor planejado para a venda a custo
3) Orçamento 54 = valor planejado para o estoque final
4) Orçamento 34 = valor planejado de compra.

Dúvidas:

- Está previsto para ser informado o REV para o calculo do estoque e da necessidade de compra. Deve ser informado o REV ou o valor da compra planejada?

- No encerramento do mês, deve ser atualizado o valor do estoque final e recalcular todos os valores planejados para os meses seguintes? OBS: o planejamento mudaria todos os meses.

_____

PRESIDÊNCIA

_____

GERENTE DE TI

_____

DIRETORIA COMERCIAL

ANEXO II – FORMULÁRIO DE AVALIAÇÃO DE SOFTWARE DA EMPRESA ESTUDADA.

QUESTIONÁRIO DE AVALIAÇÃO DO SISTEMA NOTA DE 0 – 10.

Tabela 4 – Formulário de avaliação de software.

CRITÉRIO	NOTA
1. FACILIDADE DE USO Avaliar se o entendimento do sistema e o seu uso são simples.	
2. TELAS DO SISTEMA Avaliar se as telas desenvolvidas no sistema são simples e apresentam os dados com clareza.	
3. MANUAL DO SISTEMA Avaliar se o manual do sistema é claro e esclarece as dúvidas sobre o seu funcionamento.	
4. QUALIDADE DO SISTEMA Avaliar se o sistema é eficaz e não possui nenhum erro na sua execução. Se for encontrado algum problema a nota deve ser reduzida.	
5. MELHORIA NA EXECUÇÃO DA TAREFA Avaliar se o novo sistema melhorou a atividade exercida pelo usuário.	
6. O SISTEMA ATENDE TODAS AS NECESSIDADES Avaliar se o sistema cumpre todas as funcionalidades que se propõe.	
7. AVALIAÇÃO GERAL DO SISTEMA Avaliar o sistema como um todo, em todas as questões pertinentes como funcionalidades, rapidez etc.	
TOTAL	

FORMUÁRIO DE AVALIAÇÃO DO SISTEMA, ONDE O CLIENTE DESCREVE OS PONTOS POSITIVOS E NEGATIVOS.

Tabela 5 – Formulário de ponderações sobre o software.

ANÁLISE DA LOJA
1. PONDERAÇÕES SOBRE O SISTEMA.
2. MELHORIAS SUGERIDAS
3. DEMAIS OBSERVAÇÕES

ANEXO III – FORMULÁRIO DE RELATO DE ERROS ENCONTRADOS PELA LOJA NA FASE DE PRODUÇÃO DO SOFTWARE.

FORMUÁRIO DE RELATO DE ERROS CADASTRADOS PELAS FILIAIS.

Tabela 6 - Formulário Relato de erros encontrados.

ANÁLISE DA LOJA
1. Erro encontrado no sistema.
2. Descrição do Erro.
3. Demais ponderações.

ANEXO IV – RELATÓRIO DOS ERROS REGISTRADOS PELO SUPORTE.

RELATÓRIO DOS ATENDIMENTOS.

Figura 16 – Atendimentos do Suporte.

Atendimentos Suporte

Data: 02/12/2016 15:20:45

Usuário	Data	Encerrado	Problema	Solução	Sistema
Gabriel	14/11/2016 09:20:16	14/11/2016 10:45:15	A loja044 não está conseguindo emitir notas de devolução e saída, quando dicamos em finalizar aparece uma mensagem de erro dizendo que deve ser informado a IE. O erro está acontecendo com cliente pessoa física e jurídica.	Passei a situação para o Kleber que me informou que o sistema estava com problema, retornei para a loja e disse que só de tarde iria ser resolvido.	Notas
Jussara	14/11/2016 15:22:56	14/11/2016 15:30:01	Loja 399 me ligou com o problema no módulo de notas, ela tentou selecionar um item que estava na lista 10 das promoções e o sistema não puxa.	Liguei para o Kleber que me disse que as notas estavam com problema e não daria certo hoje e me orientou a fazer as notas no sistema antigo	Notas
Jussara	15/11/2016 08:22:26	15/11/2016 09:52:11	Loja044 disse que atualizou o sistema de notas mas está dando erro no total de ICMS, acessei o servidor da loja e o erro está retornando no total do cálculo.	Falei com o Kleber que verificou que teve um novo problema, ele gerou uma nova versão e mandamos para a loja044 e 399.	Notas
Gabriel	15/11/2016 13:51:06	15/11/2016 14:10:08	A loja 044 ligou para mim me dizendo que estava dando uma mensagem de erro ao localizar XML da nota.	Passei para o desenvolvimento o Print do erro que geraram uma nova versão do notas.	Notas
Jussara	15/11/2016 17:10:06	15/11/2016 17:41:07	A loja 044 ligou e falou que estava com fila e não estava conseguindo imprimir a nota do cliente para realizar a devolução. Acessei e acompanhei juntamente com a gerente que me mostrou o erro "Telefone inválido".	Falei com o Kleber que me disse que tinha problema e que iria enviar uma nova versão na quarta.	Notas
Jussara	16/11/2016 08:40:10	16/11/2016 09:15:10	Atendi uma ligação da loja399 dizendo que estava dando erro para abrir o sistema de notas. Tirei um print do erro e levei para o Bruno que viu que era a DLL "Capicom.dll" que estava faltando na loja	Enviamos uma nova versão para as lojas 044 e 399 com a DLL.	Notas
Jussara	16/11/2016 10:34:26	16/11/2016 10:50:02	Loja 159 me ligou dizendo que quando foi finalizar uma venda, o sistema retorna erro de Serviço fora do ar.	Passei o problema para o Tiago que pediu pra eu avisar a loja que iria demorar uns minutos. Ele me disse que alterou uma rotina e que ia funcionar, falei com a loja e ela conseguiu imprimir	Vendas
Gabriel	16/11/2016 11:16:21	16/11/2016 11:28:14	A loja044 me ligou falando que não estava imprimindo de novo as notas. As notas ficam só comunicando com a receita e não imprime.	Passei para o desenvolvimento que verificou que a nota já estava como 100 receita, daí o programador alterou a base e imprimiu.	Notas
Jussara	16/11/2016 14:09:17	16/11/2016 14:25:01	A loja 025 ligou e disse que não estava conseguindo alterar o cadastro de endereço de um cliente que era de outra cidade. Acessei e tirei m print do erro.	Mandei o print para o Felipe que viu o erro e falou com os programadores que falaram que estava com problema. Foi enviado uma versão 2.0	Vendas
Gabriel	17/11/2016 10:01:36	17/11/2016 10:08:18	Loja044 ligo dizendo que esta fazendo uma nota de saída e não conseguiu selecionar a unidade do depósito.	Falei com o Kleber que me disse que estava faltando um cadastro para a loja. Ele rodou um script e funcionou	Notas
Jussara	18/11/2016 09:11:48	18/11/2016 09:35:48	Loja 399 ligou dizendo que estava dando erro na hora de fazer uma nota de demonstração. Tirei um print e mandei para o desenvolvimento.	O Kleber me disse que esta com problema e que ele iria compilar uma nova versão	Notas
Jussara	19/11/2016 16:12:48	19/11/2016 16:50:06	Loja 044 ligou para a Fran que transferiu para mim, a loja disse que tinha um cliente da argentina que não estava conseguindo emitir nota de devolução retornava erro de CPF.	Mandei um mensagem no Whats do Kleber que me orientou a mandar a loja fazer no sistema antigo que segunda ele iria ver o problema.	Notas

Fonte: Autor, 2016.

ANEXO V – TODOS OS FORMULÁRIOS DE AVALIAÇÃO DE SOFTWARE RESPONDIDOS.

QUESTIONÁRIO DE AVALIAÇÃO DO SISTEMA NOTA DE 0 – 10.

Tabela 7 – Formulário de avaliação de software TDD loja 01.

CRITÉRIO	NOTA
1. FACILIDADE DE USO Avaliar se o entendimento do sistema e o seu uso são simples.	7
2. TELAS DO SISTEMA Avaliar se as telas desenvolvidas no sistema são simples e apresentam os dados com clareza.	9
3. MANUAL DO SISTEMA Avaliar se o manual do sistema é claro e esclarece as dúvidas sobre o seu funcionamento.	8
4. QUALIDADE DO SISTEMA Avaliar se o sistema é eficaz e não possui nenhum erro na sua execução. Se for encontrado algum problema a nota deve ser reduzida.	9
5. MELHORIA NA EXECUÇÃO DA TAREFA Avaliar se o novo sistema melhorou a atividade exercida pelo usuário.	6
6. O SISTEMA ATENDE TODAS AS NECESSIDADES Avaliar se o sistema cumpre todas as funcionalidades que se propõe.	10
7. AVALIAÇÃO GERAL DO SISTEMA Avaliar o sistema como um todo, em todas as questões pertinentes como funcionalidades, rapidez etc.	10
TOTAL	59

Tabela 8 – Formulário de avaliação de software TDD loja 02.

CRITÉRIO	NOTA
1. FACILIDADE DE USO Avaliar se o entendimento do sistema e o seu uso são simples.	9
2. TELAS DO SISTEMA Avaliar se as telas desenvolvidas no sistema são simples e apresentam os dados com clareza.	10
3. MANUAL DO SISTEMA Avaliar se o manual do sistema é claro e esclarece as dúvidas sobre o seu funcionamento.	7
4. QUALIDADE DO SISTEMA Avaliar se o sistema é eficaz e não possui nenhum erro na sua execução. Se for encontrado algum problema a nota deve ser reduzida.	10
5. MELHORIA NA EXECUÇÃO DA TAREFA Avaliar se o novo sistema melhorou a atividade exercida pelo usuário.	8
6. O SISTEMA ATENDE TODAS AS NECESSIDADES Avaliar se o sistema cumpre todas as funcionalidades que se propõe.	10
7. AVALIAÇÃO GERAL DO SISTEMA Avaliar o sistema como um todo, em todas as questões pertinentes como funcionalidades, rapidez etc.	10
TOTAL	64

Tabela 9 – Formulário de avaliação de software Tradicional loja 01.

CRITÉRIO	NOTA
1. FACILIDADE DE USO Avaliar se o entendimento do sistema e o seu uso são simples.	5
2. TELAS DO SISTEMA Avaliar se as telas desenvolvidas no sistema são simples e apresentam os dados com clareza.	6
3. MANUAL DO SISTEMA Avaliar se o manual do sistema é claro e esclarece as dúvidas sobre o seu funcionamento.	8
4. QUALIDADE DO SISTEMA Avaliar se o sistema é eficaz e não possui nenhum erro na sua execução. Se for encontrado algum problema a nota deve ser reduzida.	7
5. MELHORIA NA EXECUÇÃO DA TAREFA Avaliar se o novo sistema melhorou a atividade exercida pelo usuário.	6
6. O SISTEMA ATENDE TODAS AS NECESSIDADES Avaliar se o sistema cumpre todas as funcionalidades que se propõe.	9
7. AVALIAÇÃO GERAL DO SISTEMA Avaliar o sistema como um todo, em todas as questões pertinentes como funcionalidades, rapidez etc.	8
TOTAL	49

Tabela 10 – Formulário de avaliação de software Tradicional loja 02.

CRITÉRIO	NOTA
1. FACILIDADE DE USO Avaliar se o entendimento do sistema e o seu uso são simples.	8
2. TELAS DO SISTEMA Avaliar se as telas desenvolvidas no sistema são simples e apresentam os dados com clareza.	9
3. MANUAL DO SISTEMA Avaliar se o manual do sistema é claro e esclarece as dúvidas sobre o seu funcionamento.	9
4. QUALIDADE DO SISTEMA Avaliar se o sistema é eficaz e não possui nenhum erro na sua execução. Se for encontrado algum problema a nota deve ser reduzida.	8
5. MELHORIA NA EXECUÇÃO DA TAREFA Avaliar se o novo sistema melhorou a atividade exercida pelo usuário.	7
6. O SISTEMA ATENDE TODAS AS NECESSIDADES Avaliar se o sistema cumpre todas as funcionalidades que se propõe.	10
7. AVALIAÇÃO GERAL DO SISTEMA Avaliar o sistema como um todo, em todas as questões pertinentes como funcionalidades, rapidez etc.	8
TOTAL	59

FORMUÁRIO DE AVALIAÇÃO DO SISTEMA, ONDE O CLIENTE DESCREVE OS PONTOS POSITIVOS E NEGATIVOS.

Tabela 11 – Formulário de ponderações sobre o software TDD loja 01.

ANÁLISE DA LOJA
1. PONDERAÇÕES SOBRE O SISTEMA. O sistema ficou bom porém a impressora ficou um pouco mais lenta ao emitir o cupom do cliente, cheguei a esperar 10 segundos para sair uma venda a vista.
2. MELHORIAS SUGERIDAS Deixar mais rápido o cupom dos clientes que já são da loja desde o inicio, teve uma senhora que sempre compra na loja que disse que estava demorando demais, ai tive que explicar que o sistema era novo e por isso estava mais devagar.
3. DEMAIS OBSERVAÇÕES No recebimento dos carnês a impressora é rápida emite mais rápido que a antiga, a colaboradora Cristina fez o recebimento de 7 parcelas na metade do tempo da antigo, esse foi um ponto que gostamos muito.

Tabela 12 – Formulário de ponderações sobre o software TDD loja 02.

ANÁLISE DA LOJA
1. PONDERAÇÕES SOBRE O SISTEMA. Gostei do novo sistema, a impressora não precisa ser desligada durante a noite, nem tive mais o problema de emitir a X-Forçada, a redução Z da impressora ficou bem menor que a antiga, porém não entendi por que está mostrando apenas o valor liquido das vendas? O cupom ficou melhor também, mas os clientes me perguntam sobre esse QR-Code como eles consultam ai não sei dizer por que meu celular não funciona. As vezes também a impressora emite dois cupons escrito contingencia, liguei para o suporte e eles informaram que era pela internet que caia.
2. MELHORIAS SUGERIDAS Colocar um espelho das vendas do dia como temos na antiga.
3. DEMAIS OBSERVAÇÕES Gostei bastante do novo sistema ficou mais rápido e fácil e deu poucos erros na versão.

Tabela 13 – Formulário de ponderações sobre o software Tradicional loja 01.

ANÁLISE DA LOJA
1. PONDERAÇÕES SOBRE O SISTEMA. O sistema no inicio funcionava só para clientes com CPF, tive bastante problemas com as empresas dava a mensagem de que a inscrição estadual era invalida Acho que o notas ficou mais lento que o antigo, as vezes vai mais rápido mas outra vez esperei 1 minuto e retornou erro do telefone que deveria conter o digito 9. Gostei das telas e do manual, mas deveria ser explicado melhor as notas de devolução que no outro sistema era mais simples que o atual.
2. MELHORIAS SUGERIDAS Integrar a nota de devolução com o sistema de vendas, para não ser preciso abrir o notas.
3. DEMAIS OBSERVAÇÕES Quinta-feira tive novamente o problema de não puxar um item da lista 10.

Tabela 14 – Formulário de ponderações sobre o software Tradicional loja 02.

ANÁLISE DA LOJA
1. PONDERAÇÕES SOBRE O SISTEMA. Tive bastante problema para usar o sistema é bem diferente do antigo, existe uma tela para cada tipo de nota, para as devoluções e para as saídas de mercadoria. Também deu bastante erro no primeiro dia em que fomos usar o sistema, não estava imprimindo nenhum tipo de nota fiscal, retornava erro de ICMS.
2. MELHORIAS SUGERIDAS Colocar todas as notas em uma única tela e diminuir o tempo para imprimir.
3. DEMAIS OBSERVAÇÕES Nenhuma.

ANEXO VI – FORMULÁRIO DE RELATO DE ERROS ENCONTRADOS PELA LOJA NA FASE DE PRODUÇÃO DO SOFTWARE.

FORMUÁRIO DE RELATO DE ERROS CADASTRADOS PELAS FILIAIS.

Tabela 15 - Formulário Relato de erros encontrados no projeto TDD na loja 01.

ANÁLISE DA LOJA
1. Erro encontrado no sistema. As gurias do caixa me informaram que anotaram 3 erros.
2. Descrição do Erro. Ocorreu um erro e não consegui realizar nenhuma venda, retornava a mensagem que o serviço estava indisponível. Liguei para o suporte e levou 15 minutos para eles resolverem o problema, depois não deu nenhum outro erro. Estava fazendo uma venda e o computador reiniciou ai tive que imprimir o cupom de novo, por que não tinha saído na impressora. Uma venda saiu escrito errado o nome do cliente, aparecia só o primeiro nome
3. Demais ponderações. Gostamos que os problemas foram resolvidos bem rápido e não voltou a dar. A impressora está um pouco lenta.

Tabela 16 - Formulário Relato de erros encontrados no projeto TDD na loja 02.

ANÁLISE DA LOJA
1. Erro encontrado no sistema. Tivemos dois erros: Não conseguimos alterar o CPF do cliente de outra unidade e o sistema deu erro em uma venda aparecia a mensagem de serviço indisponível.
2. Descrição do Erro. O cliente Edson que sempre compra na loja, tem o crediário na loja de Carazinho, ele informou que o endereço na era o mesmo, quando tentei mudar na tela de vendas o sistema não deixou dava a mensagem que não era possível alterar os dados do cliente. O Outro erro ocorreu com a Carla quando ela foi finalizar uma venda no crediário dava a mensagem de serviço fora, ligamos para o suporte e eles alteraram o sistema e a venda saiu na impressora.
3. Demais ponderações. Nenhum outro erro.

Tabela 17 - Formulário Relato de erros encontrados no projeto Tradicional na loja 01.

ANÁLISE DA LOJA
1. Erro encontrado no sistema. Foi anotado nove erros na loja.
2. Descrição do Erro. Teve muitos problemas no sistema quando fomos emitir a primeira nota dava um erro dizendo que a inscrição estadual era inválida, mas o cliente era uma pessoa jurídica e nos disse que nunca teve empresa. Ligamos para o suporte e eles demoraram 30 minutos para responder que esta com problema e teriam que enviar uma nova versão para a loja. Tivemos que liberar o cliente e fazer a devolução manual. No outro dia imprimiu a nota mas demorava 1 minuto para fazer 1 nota, o caixa estava com fila e tivemos que fazer 4 devoluções. Dava erro também no selecionar os produtos da lista 10, quando ia carregar o produto do cupom não apareciam os itens da venda. Outro Erro que aconteceu na hora de enviar a nota, estava retornando erro da UF autorizadora dizia que não era do Estado. Quando fomos fazer as notas para os colaboradores aparecia a mensagem de que o desconto estava zerado. Quando fomos passar a venda no caixa 101 o valor de devolução estava menor que o da nota e tivemos que ar desconto manual no cupom. A colaboradora Ana disse que também aparecia uma mensagem de erro quando ela deixava o sistema aberto, ai ela reiniciava o computador e funcionava.
3. Demais ponderações. As notas continuam demorando mais que o sistema de antes.

Tabela 18 - Formulário Relato de erros encontrados no projeto Tradicional na loja 02.

ANÁLISE DA LOJA
1. Erro encontrado no sistema. Pedi para a Eliza anotar os erros e ela me disse que teve 5.
2. Descrição do Erro. Na segunda não conseguimos imprimir as notas tivemos que fazer todas no sistema antigo. O Gabriel do suporte disse para a Eliza que só iria funcionar as 13 horas da tarde, mas só funcionou no outro dia. Na terça também aconteceu de imprimir umas notas e outras não. Quarta-feira o sistema mostrava uma mensagem dizendo que o total dos itens não era igual o ICMS. Foi ligado para o Gabriel e foi resolvido enviando uma nova versão, tivemos que para a loja para atualizar. Ficamos com bastante fila por causa do sistema de notas, os cliente queriam fazer devolução e os sistema não funcionava. Também perdi 4 cliente que eu ia abrir crediário e eles não quiseram esperar, falaram que estava muito demorado. Um cliente me disse que ia comprar em outra loja que é nosso concorrente. As notas também estão bastante demoradas, tentamos imprimir em 3 caixas e demora mais ainda. Outro erro que estava dando no caixa 103, quando finalizamos a nota da uma mensagem dizendo que o XML não foi encontrado, depois clicamos em OK e imprime. Os caixas 104, 105 e 106 não imprime as notas ligamos para o suporte e eles disseram que era problema de configuração da impressora.
3. Demais ponderações. Os outros erros ela esqueceu de anotar.