UNIVERSIDADE DO SUL DE SANTA CATARINA

MESSIAS FLOR SANTOS

QUALIDADE DE SOFTWARE ATRAVÉS DE TESTES UNITÁRIOS

Araranguá

2011

UNIVERSIDADE DO SUL DE SANTA CATARINA

MESSIAS FLOR SANTOS

QUALIDADE DE SOFTWARE ATRAVÉS DE TESTES UNITÁRIOS

Monografia apresentada ao Curso de... Pós Graduação em Engenharia de Projetos de Software da Universidade do Sul de Santa Catarina, como requisito parcial para obtenção do título de Especialista.

Orientador: Carlos André de Sousa Rocha, Msc.

Araranguá

2011

MESSIAS FLOR SANTOS

QUALIDADE DE SOFTWARE ATRAVÉS DE TESTES UNITÁRIOS

Esta Monografia foi julgada adequada à obtenção do título de Especialista em Engenharia de Projetos de Software e aprovada em sua forma final pelo Curso de Sistemas de Informação da Universidade do Sul de Santa Catarina.

Araranguá, 23 de Dezembro de 2011.

_____

Orientador: Carlos André de Sousa Rocha, Msc.

Universidade do Sul de Santa Catarina

Dedico este trabalho a minha família, meus amigos, colegas de trabalho e professores que me incentivaram e colaboraram para que conseguisse concluir esta monografia com sucesso.

AGRADECIMENTOS

Primeiramente agradeço a Deus por possibilitar mais esta conquista e por fazer acreditar que tudo é possível quando temos fé.

Agradeço também aos meus familiares, amigos e noiva, pela compreensão e paciência nos momentos de falta e pelos incentivos concedidos para que este trabalho se concretizasse.

Aos professores do curso, que muito qualificados, passaram além do conteúdo programado também conhecimentos práticos de sua experiência profissional para que pudesse entender melhor e adquirir estes conhecimentos passados.

Ao meu orientador e professor Carlos André de Sousa Rocha e ao coordenador do curso, professor Marco Antônio pelo apoio, paciência e dedicação, e também pelo empenho da organização do curso que foi excelente.

“Eu acredito demais na sorte. E tenho constatado que, quanto mais duro eu trabalho, mais sorte eu tenho.” (Thomas Jefferson)

RESUMO

Devido à qualidade ser um fator diferencial para permanência da empresas de software no mercado tão competitivo da Tecnologia da Informação, o presente trabalho mostrará as diversas maneiras que estas empresas podem utilizar para garantir a qualidade em seus produtos e serviços através de testes unitários de software. Conjuntamente os elementos sobre metodologias ágeis, que atendem rápido as mudanças que ocorrem em ambientes de desenvolvimento, serão abordados com foco na metodologia *Extreme Programming*, que utiliza testes unitários para garantir a qualidade nos softwares. Modelos de referência de melhoria de processos e produtos serão também apresentados, pois além de fornecer maior qualidade para a empresa e seus produtos, permite que ela obtenha maior reconhecimento no mercado. O principal foco do trabalho, qualidade de software e testes serão mostrados nas últimas seções, apresentando as diferentes formas de testes e centralizando nos testes unitários, que garantem que cada parte do software possa ficar livre de falhas, permitindo que a qualidade englobe maior parte dele e gerando um produto diferenciado para o mercado. A aplicação dos testes unitários será realizada aplicando o desenvolvimento guiado por testes juntamente com o framework JUnit.

Palavras-chave: Qualidade. Testes. Testes unitários. JUnit.

ABSTRACT

Due to the fact that quality is a differential feature to the permanence of software companies in competitive Information Technology market, this study will show the several ways those companies can guarantee the quality of its products and services through unit tests of software. Together the elements of agile methodologies which quickly answers to the changes that are occurring in developing environments will be addressed with a focus on Extreme Programming methodology which uses unit tests to guarantee the quality of the software. It is also presented models of reference to improve processes and products that besides it gives higher quality to the company and its products, it also offers more acknowledgment in the market.

The main focus of this work, quality of software and tests, will be in the last sections, where it will be shown the several ways of tests, centralizing the unit tests which guarantee that every part of the software is free of failures, giving higher quality and generating a differential product to the market. The performance of unit tests will use the development guided by tests together with the framework JUnit.

Key words: Quality. Tests. Unit tests. JUnit.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Atualmente ocorrem muitos problemas relacionados com projetos de desenvolvimento de software, que geralmente são recorrentes em várias empresas de desenvolvimento de sistemas de informação: prazos estourados, custos extrapolados, cancelamento por falta de planejamento, falhas por falta de funcionalidades, tudo isso gerando a insatisfação do cliente.

Ao longo da história da engenharia de software, muitas instituições e profissionais da área vêm procurando soluções para acabar com estes problemas e agregar mais qualidade aos projetos de software.

Surgiram modelos de referência para melhoria dos processos da organização, possibilitando a padronização e integração dos processos, como o CMMI (*Capability Maturity Model Integration*) e o MPS.BR (Melhoria de Processo do Software Brasileiro), de forma que a empresa possa adquirir maturidade e desenvolver software de maneira mais eficaz e com mais qualidade. Além disso, eles agregam um reconhecimento internacional para empresa em melhoria de processos através de uma avaliação, oferecendo um diferencial competitivo.

Existem diversas metodologias de desenvolvimento de software que também contribuem para que surjam os problemas anteriormente apresentados. Metodologias conhecidas como tradicionais, que já estão desgastadas e vêm sendo muito criticadas, por serem antigas e ainda utilizadas. Uma destas metodologias é o cascata, que é o paradigma mais antigo da engenharia de software, e determina que todos os requisitos devem ser levantados no início do projeto, fato este que é praticamente impossível pois é difícil para o cliente lembrar de tudo no início do projeto, e assim quando o projeto está pronto, não atende muitas de suas expectativas (PRESSMAN, 2006)

Como alternativa para estes problemas, surgiram os métodos ágeis de desenvolvimento, que são apoiados num conjunto de práticas, conhecidas como boas práticas, que podem ser aplicadas à maioria dos projetos de software, de acordo com a necessidade de cada empresa, pois não necessitam serem todas aplicadas num determinado projeto. Dentro destas práticas, encontramos uma que se utiliza de testes unitários para fornecer melhor qualidade para o software, o TDD (*Test-Driven Development)* ou em português, desenvolvimento guiado por testes.

Com a utilização de testes unitários, o código será quase que totalmente testado muito antes da etapa de testes normais que ocorre na maioria dos projetos, e devido a sua cobertura, diminui com as falhas e erros que podem ocorrer quando estes não são aplicados.

1.1 JUSTIFICATIVA

A grande competitividade no mercado atual atrelada ao avanço da tecnologia obriga as empresas desenvolvedoras de software a buscar soluções para atender melhor seus clientes e desenvolver softwares de melhor qualidade, buscando a maior satisfação deles para que ela possa permanecer no mercado.

De acordo com Bartié (2002, p.6), “mais de 70% dos projetos falham nas entregas das funcionalidades esperadas”, o que mostra que muitas empresas não estão conseguindo entender bem os requisitos solicitados pelos clientes. Isto está abrindo espaço para os concorrentes, e neste mundo globalizado, deve-se também levar em conta a qualidade e mão de obra oferecida por outros países, como por exemplo, a Índia, que se destaca pela competência em desenvolvimento de software.

Segundo o relatório do Standish Group (2010), conhecido como *chaos report,* realizado em 2009, 32% dos projetos de software pesquisados tiveram sucesso, 44% mudaram e os outros 24% falharam, o que demonstra que ainda existem muitos problemas acontecendo nos projetos de desenvolvimento.

A melhoria de processos através dos modelos de referência oferece resultados positivos para as empresas, pois além de contribuir com a diminuição dos problemas que ocorrem com os projetos através da padronização dos processos, abre novas fronteiras de comércio para a empresa por meio do reconhecimento de qualidade de seus processos, servindo como diferencial para avaliação de seus produtos e serviços.

A aplicação dos testes unitários nos projetos de software faz com que os requisitos possam ser mais bem compreendidos pelos desenvolvedores, além de que dificilmente alguma falha passará despercebida, pois cada método do código

será testado pelo próprio desenvolvedor, gerando um bom código, de fácil manutenção e ainda sem falhas.

Portanto, a geração de um produto final que atenda as expectativas do cliente, que mostre que não possui falhas, dessa forma atingindo a sua satisfação, é resultado de um produto com qualidade que alavanca o reconhecimento da empresa e torna fácil a sua permanência por muito tempo no mercado.

1.2 OBJETIVO GERAL

Apresentar práticas, modelos e testes de software, capazes de garantir a qualidade para empresa e seus produtos, focando nos testes unitários como forma de englobar grande parte do sistema e atingir a satisfação do cliente final.

1.3 OBJETIVOS ESPECÍFICOS

- Apresentar as características dos modelos ágeis de desenvolvimento enfatizando um modelo que prioriza os testes;
- Mostrar modelos que garantem a qualidade e reconhecimento da empresa bem como de seus produtos e serviços;
- Apresentar os benefícios para um projeto de software quando aplicado os testes unitários;
- Utilizar o framework JUnit para aplicação de testes unitários.

1.4 ESTRUTURA DO TRABALHO

O presente trabalho está dividido em 7 capítulos. O primeiro capítulo contém a introdução, justificativa, objetivo geral, objetivos específicos e a estrutura do trabalho.

No segundo capítulo será tratado sobre os métodos ágeis de desenvolvimento, dando ênfase ao que foca mais em testes, que é o XP.

O terceiro capítulo versará sobre modelos de melhoria de processos, conhecidos como modelos de referência, como o CMMI e o MPS.BR. Também mostrará o MPT.BR (Melhoria do Processo de Teste Brasileiro), e as normas de qualidade para software.

O quarto capítulo apresentará o tema central do trabalho, qualidade e testes de software, mostrando os diferentes tipos de testes.

No quinto capítulo, os testes unitários serão abordados, apresentando também o TDD e detalhando o framework JUnit.

O sexto capítulo trará o problema em que os testes unitários foram aplicados juntamente com um exemplo de sua utilização e a análise dos resultados dos testes..

Por fim, o sétimo e último capítulo tratará sobre a conclusão do trabalho e de sugestão para trabalhos futuros.

2 METODOLOGIAS ÁGEIS

Devido aos problemas encontrados com os processos de desenvolvimento mais tradicionais, surgiram as metodologias ágeis para desenvolvimento de software. Os ágeis pregam que a codificação é mais importante que a extensa documentação que ocorre com os outros modelos, e também através de suas boas práticas, mostra que pode diminuir e em certos casos até acabar com os freqüentes problemas que ocorrem. (TELES, 2006)

As metodologias ágeis de desenvolvimento vêm ganhando adeptos pelo mundo inteiro. Elas possuem valores e princípios que prometem dar fim aos problemas que ocorrem freqüentemente e que atormentam a vida da equipe desenvolvedora e também do cliente. Estas metodologias, lançados já a algum tempo mas que começaram a ganhar força a partir de 2001, quando um grupo de 17 grandes profissionais de software, reuniu-se com a finalidade de consolidar os seus conhecimentos com as boas práticas de agilidade, e resolveram criar o manifesto ágil. (BECK et al, 2010a)

O grupo, conhecido como a Aliança Ágil resolveu criar uma maneira diferenciada de desenvolver, atacando os modelos mais tradicionais e sugerindo mudanças revolucionárias, através da construção de um produto que atendesse realmente as necessidades do cliente em todos os aspectos mencionados nos quais havia problemas. No desenvolvimento ágil, a principal prioridade é a satisfação do cliente. (BECK et al., 2010b)

Os métodos ágeis são apoiados nos princípios do manifesto para desenvolvimento ágil de software, e possui os seguintes valores:

> Indivíduos e interações mais que processos e ferramentas.
> Softwares funcionais mais que documentação abrangente.
> Colaboração do cliente mais que negociação de contratos.
> Responder a mudanças mais que seguir um plano. (BECK et al, 2010a)

Analisando os valores do manifesto ágil pode-se perceber as características dos métodos ágeis. A interação de toda equipe envolvida no projeto incluindo o cliente, profissionais capacitados trabalhando juntos, é mais importante do que possuir ferramentas e processos revolucionários e não ter alguém capacitado para usufruir destas ferramentas. Às vezes as empresas gastam dinheiro investindo

em novas ferramentas, enquanto o que realmente irá agregar mais valor para ela são seus profissionais.

A documentação de software é importante para projetos de software. Porém, muitas vezes muito tempo é perdido criando a documentação do sistema ao invés do próprio sistema. Por isso, os métodos ágeis focam mais na codificação, porque se o cliente for consultado sobre o que ele mais precisa, se uma documentação extensa descrevendo o que será desenvolvido ou o software real, quase todos irão optar pelo software. Além do mais, os usuários darão um melhor retorno através do software do que de diagramas gráficos, por exemplo, os diagramas da UML (Linguagem de Modelagem Unificada). (AMBLER, 2004)

A comunicação com o cliente durante o processo de desenvolvimento fornece um melhor entendimento das necessidades deste e ainda facilita a validação do software. Desta forma, é mais interessante a comunicação direta e contínua com o cliente do que um contrato realizado no início do projeto que não possua todas as necessidades do cliente.

A agilidade no desenvolvimento de software deve ser uma característica adaptável às necessidades do cliente que mudam conforme o andamento do projeto. Por isso, o desenvolvimento é realizado em ciclos curtos, com a presença do cliente, capaz de responder rapidamente às mudanças de prioridades feitas por eles.

É claro que os métodos ágeis também acham que devam existir processos e ferramentas para desenvolvimento, documentação, contratos e planos, mas não acreditam ser tão importante quanto as características mencionadas anteriormente. (AMBLER, 2004)

Nos modelos mais tradicionais, a equipe do projeto tem como referência para desenvolvimento do sistema, um documento de requisitos que é criado depois de ser realizada a fase de levantamento de requisitos, fase esta que é a única em que há contato direto com o cliente antes da entrega final do produto.

É justamente neste ponto que um dos maiores problemas é encontrado: os detalhes, características do sistema a ser projetado que o cliente tem que expor. Como tudo muda muito rapidamente, assim as necessidades aumentam e as ameaças da competitividade também, logo, modificações precisam ser feitas com muita agilidade. (PRESSMAN, 2006)

Por isso, nos modelos ágeis todos os detalhes não são especificados no início do projeto, porque sempre o cliente quer acrescentar mais alguma

funcionalidade no sistema. Sendo assim a interação e comunicação com o cliente é um ponto fundamental para que seja realmente desenvolvido o que o cliente necessita e, por isso, durante todo o projeto existe a participação do cliente para verificação da validade dos requisitos e acréscimo de novos que achar necessário, e desta forma uma dos problemas que ocorre nos tradicionais, que é quando o sistema não atende as expectativas do cliente, pode ser resolvido. (BECK, 2004a)

Eles têm como referência o desenvolvimento iterativo e incremental, onde o sistema é desenvolvido em pequenos módulos, que a cada iteração vai crescendo acrescentando-se mais funcionalidades, de modo que o cliente acompanhe o desenvolvimento do projeto, aprenda e possa dar seu feedback. Cada iteração é um executável pronto, mas é claro, não com todas as funcionalidades. (TELES, 2006)

Para um processo ser ágil, as entregas dos incrementos de software devem ser contínuas e em períodos curtos, de duas semanas a no máximo dois meses, para que o cliente possa verificar, avaliar e fornecer o feedback necessário de modo que a equipe de desenvolvimento possa realizar as alterações a cada retorno do cliente. (PRESSMAN, 2006)

Dessa forma, os problemas podem ser encontrados precocemente, pois só se migrará para o próximo ciclo se o anterior estiver pronto, então, problemas devem ser corrigidos para depois dar continuação à próxima iteração. Sendo assim, os custos para realização de alterações e também de retrabalho serão reduzidos, fato este que não ocorre nos modelos mais tradicionais, pois estes dois indicadores serão em grande parte altos.

Algumas abordagens de desenvolvimento ágil possuem uma forma de teste mais informal, já que não existe uma especificação que possa ser usada para realização dos testes pela equipe de qualidade. Porém, em um dos modelos, no Extreme Programming (XP), que será visto no item 2.2, para evitar problemas de validação, os testes são mais abordados do que em outros modelos. (SOMMERVILLE, 2007)

Existem diferentes tipos de modelos ágeis de processo, mas todos eles atendem aos princípios do manifesto ágil. Alguns dos modelos que podem ser citados são: Extreme Programming, Scrum, Desenvolvimento Adaptativo de Software (DAS), Método de Desenvolvimento Dinâmico de Sistemas (DSDM), Crystal e Desenvolvimento Guiado por Características (FDD). Segundo Koscianski

(et al., 2007), as mais conhecidas e utilizadas são os dois primeiros citados, XP e Scrum.

Devido às práticas do XP estarem mais voltadas ao foco deste trabalho, como a prática de desenvolvimento guiado por testes, na qual trabalha com testes unitários, este será detalhado no item a seguir.

2.2 EXTREME PROGRAMMING – XP

O XP começou a ser criado no final dos anos 80, com a utilização de algumas idéias características dessa abordagem, mas seu trabalho pioneiro só foi lançado em 1999 pelo seu criador, Kent Beck. (PRESSMAN, 2006)

O termo, eXtreme programming pode ser traduzido para programação eXtrema, nome sugestivo pois leva os princípios e práticas aos extremos, sugerindo revisão de código através de programação em pares, testes unitários, refatoração, simplicidade, integração contínua, entre outros que serão mostrados nos itens seguintes. (BECK, 2004a)

XP é uma metodologia para ser aplicada em pequenos a médios projetos, com equipes reduzidas, de dois a dez programadores. Ela se baseia em idéias e técnicas que vêm sendo aplicadas por décadas, mas a inovação da XP se deu quando essas práticas foram colocadas juntas, praticadas a fundo e apoiando-se umas às outras. (BECK, 2004a)

O XP talvez seja o mais conhecido e utilizado dos métodos ágeis. Além disso, ele considera os testes fundamentais, e seus pontos fortes são o desenvolvimento *test-first*, onde se deve escrever primeiro o teste para depois desenvolver a funcionalidade, e as ferramentas automatizadas de teste. (SOMMERVILLE, 2007)

Existem alguns profissionais da área de tecnologia da informação que criticam os métodos ágeis e principalmente o XP, o mais utilizado, alegando que não existe modelagem e documentação, e se existe alguma modelagem ocorre com artefatos da UML. Estas informações são equivocadas, pois o XP ainda é uma metodologia nova se comparada às tradicionais, e então o que acontece é a troca de informações erradas, talvez de fontes duvidosas, falta de conhecimento mais sólido

sobre o assunto, pois existe modelagem e documentação no XP, só que é conhecida como modelagem ágil. (AMBLER, 2004)

A modelagem faz parte da XP, podendo ser citado como exemplo, as histórias dos usuários, que serão abordadas no item 2.2.2.1, e os cartões CRC (Classe responsabilidade colaborador), que são utilizados para entender o domínio do problema ou para o desenho, e são modelos utilizados pela abordagem do XP. (AMBLER, 2004)

Como foi apresentado anteriormente no item 2, de acordo com o manifesto ágil, os métodos ágeis não devem produzir muita documentação, focando mais no código. Sabe-se que a documentação é importante para projetos de software, por isso ela também é parte fundamental do XP. Mas o que acontece é que suas práticas reduzem a necessidade de documentação, e a metodologia não especifica quais documentos devem ser produzidos, deixando o cliente determinar quais são os documentos importantes para ele, sendo que muitas vezes ele prefere um incremento do sistema funcionando a um documento descrevendo tal funcionalidade. (AMBLER, 2004)

O XP é baseado em valores e práticas que o tornam um modelo ágil de desenvolvimento. Nos itens a seguir serão descritos estes dois, valores e práticas, que dão as características necessárias para o modelo.

2.2.1 Valores

O XP é organizado através de valores e práticas para garantir que o cliente receba o máximo de atenção a cada dia e para que ele receba o devido retorno do investimento. Segundo Beck (2004a) e Teles (2006), os quatro valores do XP são: comunicação, simplicidade, feedback e coragem.

2.2.1.1 Comunicação

A comunicação é um dos problemas que ocorrem com freqüência em projetos de desenvolvimento de software. Alguns problemas que ocorrem dizem respeito à comunicação com o cliente, onde as perguntas certas não são feitas, fazendo com que uma decisão seja tomada de forma equivocada. Outros porém, podem ocorrer dentro da própria equipe, quando um programador não informa os colegas de uma mudança importante no projeto. (BECK, 2004a)

Uma boa comunicação é um ponto fundamental na construção de um software. Todos devem participar e saber o que está acontecendo no decorrer do desenvolvimento para poder expor suas idéias, sugerir modificações e soluções, para que os detalhes não sejam esquecidos, todos devem estar em sintonia e em constante comunicação para juntos resolverem os problemas, desde a equipe da empresa ao cliente final.

No desenvolvimento tradicional a comunicação é feita de várias formas que são consideradas ineficientes pelo XP. A comunicação através de telefone e a escrita são dois destes exemplos que dificultam o trabalho da equipe na medida em que as idéias do cliente podem ser interpretadas de diferentes formas por um membro da equipe, podendo ser interpretada de forma incorreta ou incompleta, gerando problemas futuros. Por isso o XP utiliza uma comunicação que pode ser mais bem interpretada e as idéias melhor esclarecidas de forma a evitar futuros retrabalhos e conseqüentemente aumento de custo no projeto: a comunicação face a face, direta entre toda equipe e o cliente. (TELES, 2006)

> “Fora do projeto XP, você provavelmente precisará de documentação. Produza documentação. No projeto, há tanta comunicação verbal que você pode precisar de pouca coisa a mais. Confie em si próprio para perceber esta diferença.” (JEFFRIES 2001 apud AMBLER, 2004 p.175)

Analisando a citação realizada por Ron Jeffries, um dos responsáveis pelo manifesto ágil, pode-se perceber que a comunicação no XP já serve como documentação, não é necessário produzir vários documentos, alguns são substituídos pela comunicação. Não que os documentos não precisem ser produzidos, mas que sejam produzidos somente os que a equipe e o cliente acharem realmente importantes.

A XP procura estabelecer que a comunicação seja feita de forma direta e contínua, através de algumas práticas que não podem ser feitas sem comunicação.

Dentre estas, pode-se destacar os testes unitários, a programação em pares e a estimativa de tarefas, buscando aproximar todos envolvidos no projeto para uma constante comunicação, mais rica e melhor aproveitada. (TELES, 2006)

2.2.1.2 Simplicidade

Para que o cliente possa aprender suas necessidades ao longo do projeto e fornecer o correto feedback, é necessário também que a equipe trabalhe de forma simples, implementando somente o necessário para atender a determinada funcionalidade, corrigindo somente as falhas conforme vão sendo descobertas e se preocupando somente com os problemas atuais, para que a entrega possa ser feita o mais rápido possível ao cliente. (TELES, 2006)

Em alguns projetos o que acontece é que uma funcionalidade às vezes é implementada com mais detalhes do que o necessário, funcionalidades que poderão nem ser utilizadas, aumentando tempo e custo em algo do qual não se tem certeza que será utilizado. Logo deve-se evitar o retrabalho que poderá acarretar dessas novas funcionalidades e também estas atividades desnecessárias. (TELES, 2006)

A simplicidade está diretamente relacionada com a comunicação. Quando a comunicação está fluindo de maneira correta no projeto, os objetivos e deveres ficam mais claros e simples, e assim, quanto mais simples, menor a comunicação sobre o desenvolvimento. Com a simplicidade, a XP aposta que é melhor construir algo simples hoje para no futuro gastar um pouco mais em uma modificação, do que fazer algo mais complicado hoje e nunca ser utilizado. (BECK, 2004a)

2.2.1.3 Feedback

O feedback é bom tanto para equipe quanto para o cliente. Para a equipe fará com que ela fique focada nas reais necessidades e naquilo que será mais importante para o cliente. Para ele, porque poderá aprender e rever suas

necessidades, de forma que não sejam informadas funcionalidades que não serão utilizadas no momento.

O XP busca acabar com o problema de realimentação que acontece com os métodos tradicionais, onde o cliente é contatado somente no início e final do projeto, pois os ciclos completos consomem muito tempo, gerando um feedback demorado e ocasionando vários problemas, tais como retrabalho, aumento de custos e estouro de cronograma. Assim como acontece com os métodos ágeis, o feedback no XP é rápido, contínuo, em períodos curtos de tempo, onde o cliente está em constante realimentação do sistema. (TELES, 2006)

O feedback acontece em diferentes escalas de tempo. Pode ocorrer em períodos curtos, para os programadores, por exemplo, através dos testes unitários gerando um feedback concreto sobre o estado do sistema; ou para os clientes, através da avaliação imediata pelos desenvolvedores de novos detalhes para o sistema. Em períodos mais longos, quando os testes de funcionalidades são feitos para o que foi desenvolvido. (BECK, 2004a)

O feedback faz com que o cliente possa conduzir o trabalho da equipe, e também é um valor importante pois irá tornar mais fácil a comunicação e conseqüentemente, deixar o projeto mais simples.

2.2.1.4 Coragem

Toda a equipe envolvida num projeto XP precisa ser corajosa. Considerando que o XP se baseia em valores e práticas que contrariam o desenvolvimento tradicional e que desenvolve de forma incremental, corrigindo falhas e adicionando novas funcionalidades a cada entrega conforme os modelos ágeis ensinam, em alguma parte do projeto algo pode ser esquecido e gerar uma falha que pode afetar vários módulos do sistema. Então a equipe deve enfrentar o medo de gerar novas falhas e mais alguns problemas, como estouro de cronograma, por exemplo, e acreditar que será capaz de contornar a situação com agilidade através dos outros valores fundamentais da XP. (TELES, 2006)

2.2.2 Práticas do XP

O XP também possui um conjunto de práticas que as equipes de desenvolvimento devem usar para obter os resultados esperados. Elas podem ser utilizadas separadamente que já oferecerão um resultado diferenciado no projeto, mas para obter o máximo de benefício que elas fornecem, é aconselhável que sejam aplicadas em conjunto, pois uma complementa outra, e a interação entre elas aumentam os resultados positivos.

Muitas destas práticas também são das metodologias ágeis, por isso alguns dos problemas citados que ocorrem com as tradicionais, podem ser contornados através delas.

Nos itens a seguir veremos as práticas do XP sugeridas por Beck (2004a).

2.2.2.1 O jogo do planejamento

Como foi citado no item 2.2.1.3, feedback, é muito importante que o projeto seja conduzido atendendo as necessidades mais importantes do cliente e que sejam implementadas somente as funcionalidades definidas por ele, sem detalhes que poderão nunca ser utilizados no futuro, gerando somente esforço desnecessário, aumento de custos e de tempo.

Para que a equipe de desenvolvimento fique focada no que é mais importante para o cliente, o XP possui diversas formas de planejamento. O jogo de planejamento acontece no início de cada release e iteração, onde pequenos módulos do sistema são desenvolvidos em períodos curtos, para que sejam realizadas reuniões com o cliente a fim de definir mais alguma funcionalidade e priorizar o que é mais importante para ele, para ser implementado o mais cedo possível. (TELES, 2006)

No decorrer do projeto, através da interação de toda equipe e do cliente, várias mudanças acontecem, sobretudo em duas áreas: negócios e técnica.

Na área de negócios, formada pelos tomadores de decisões sobre o que o sistema deve fazer, a cada nova reunião devem decidir sobre escopo, prioridade, versões e datas de entrega; na área técnica, formada por todos responsáveis por implementar o sistema, devem decidir sobre estimativas, conseqüências, processo e cronograma. (BECK, 2004a)

As funcionalidades são descritas pelos clientes em pequenos cartões que são chamados de histórias, e estas escritas em cartões próprios. As histórias são estimadas pelos desenvolvedores através de uma unidade de tempo denominada ponto, para que o cliente conheça o custo de implementação de cada história e para facilitar a priorização delas. (TELES, 2006)

Assim o planejamento vai sendo adaptado ao longo de todo o projeto, priorizando as necessidades mais importantes para o cliente e sendo estimados os prazos e custos de acordo com os novos releases e iterações.

Desta forma não ocorrerá custos e prazos estourados, porque através deste planejamento e constante presença do cliente no projeto, tudo vai sendo negociado, e o cliente pode perceber como está sendo produzido seu software, acompanhando o seu andamento para poder entender os problemas que surgem e o tempo gasto nas tarefas.

2.2.2.2 Pequenas entregas

No XP as entregas dos incrementos do sistema devem ser freqüentes, e por isso, devem ser feitas em pequenos releases. Os requisitos do cliente devem ser implementados observando os de maior importância, de forma que cada entrega tenha um conjunto reduzido de funcionalidades implementadas. (BECK, 2004a)

Desta forma o cliente obterá um retorno mais rápido de seu investimento através de cada parte do sistema que é entregue, pois se comparado com o desenvolvimento tradicional, onde o cliente recebe o sistema só depois de um longo período, o retorno do investimento no XP é bem mais rápido.

Outra vantagem das pequenas entregas pode ser a avaliação do cliente mais cedo, a cada entrega realizada, possibilitando um retorno mais rápido para que

os desenvolvedores possam fazer os ajustes em menores trechos de código e as falhas possam ser resolvidas também mais rapidamente.

2.2.2.3 Metáfora

A utilização de metáforas facilita a comunicação entre o cliente e equipe de desenvolvimento. Para o cliente fica difícil entender termos técnicos aplicados na tecnologia da informação, e por isso, a utilização de metáforas para explicação do sistema ao cliente facilita o seu entendimento e melhora a comunicação. (TELES, 2006)

Para melhor compreensão dos assuntos, as metáforas devem estar ligadas a fatos do dia a dia para que o cliente possa raciocinar melhor e entender o que está sendo passado.

2.2.2.4 Projeto simples

Como já foi abordado nos valores do XP, a codificação deve estar focada somente em atender as necessidades do cliente, em implementar um conjunto de funcionalidades definidas pelo cliente, da maneira mais simples possível. Como Beck (2004a) afirma, não se deve prever o futuro, incrementar funcionalidades que poderão nem ser utilizadas somente por achá-las interessantes.

O projeto deve ser simples, a codificação das funcionalidades deve atender os testes unitários, e deve ser realizada uma refatoração para deixar o mínimo de classes e métodos possíveis, de forma a tornar o projeto mais simples e mais fácil de manter. (TELES, 2006)

Com a implementação de funcionalidades adicionais, não essenciais, além de tomar mais tempo na codificação, irá tornar a manutenção mais complexa, e conseqüentemente, poderá aumentar os seus custos. Isto é o que acontece com as metodologias tradicionais quando os requisitos não são bem compreendidos, além de gerar ainda a insatisfação do cliente.

Por isso no XP o projeto deve ser simples, para produzir um incremento mais rápido, mais fácil e barato de manter.

2.2.2.5 Testes

O XP é a metodologia ágil que está mais focada na realização de testes. Por isso tem como uma de suas práticas o desenvolvimento guiado por testes (TDD), que é realizado através de testes unitários feitos pelos próprios desenvolvedores antes da codificação, para garantir melhor qualidade no produto.

Este desenvolvimento faz com que o desenvolvedor imagine como o código que ele está implementando irá funcionar, para que ele primeiramente faça um teste que gere o resultado esperado, para depois implementar a funcionalidade baseada neste teste.

No XP os testes que precisam ser feitos são isolados e automáticos. Os testes unitários são feitos de formas isoladas, não interagem, para evitar que um erro em um teste cause falhas nos outros. Devem ser automatizados para não tomar muito tempo dos desenvolvedores em repetidos processos de teste. (BECK, 2004a)

Outra forma de teste no XP é realizada pelos clientes, através do teste de aceitação. Eles também devem ser criados antes da codificação das funcionalidades, e são aplicados para cada história do cliente, que contém um conjunto de funcionalidades. (TELES, 2006)

Caso alguma falha seja descoberta no processo de teste do XP, a tarefa mais importante para a equipe passa a ser encontrar uma solução para esta falha. Todos os esforços da equipe ficam focados no conserto do problema pois não se sabe o tempo que será gasto para resolução deste.

Devido à importância desta prática de teste, e do seu relacionamento com este trabalho, o TDD será tratado mais detalhadamente no item 5.3**.**

2.2.2.6 Refatoração

Para que um software seja fácil de manter, possibilitando rapidez nas alterações que vierem a ser necessárias, o código deve ser simples, organizado e legível. Desta forma, qualquer equipe que trabalhar no projeto não terá dificuldades para realizar a manutenção e acrescentar novas funcionalidades a pedido do cliente.

A refatoração possui esta idéia e para isso faz com que os desenvolvedores, ao analisar um código que não esteja legível, reescrevam o código de forma a torná-lo legível, não alterando a funcionalidade, somente tornando ele mais simples de entender, processo este que é fundamental no XP. (BECK, 2004a)

Essa prática do XP obriga os desenvolvedores a realizarem a refatoração em código não legível antes de realizar qualquer alteração no sistema. Desse modo, o sistema terá sempre um código mais organizado e claro, tornando fácil a sua manutenção por qualquer outro desenvolvedor que futuramente precise implementar mais alguma funcionalidade. (TELES, 2006)

Porém um problema que pode acontecer com a refatoração é o código parar de funcionar. Por isso existem técnicas que podem ser seguidas que possuem caminhos para aplicação desta prática para que ela seja feita de forma mais segura. Mas além disso, o XP possui a qualidade proporcionada pelos testes unitários, e assim, os testes poderão ser aplicados após a refatoração para fornecer maior segurança no processo. (TELES, 2006)

2.2.2.7 Programação em pares

A programação em par talvez seja uma das práticas mais conhecidas do XP. Existem muitas pessoas que criticam esta técnica, alegando que é uma perda de tempo um profissional ficar observando o outro trabalhar, o que na realidade não é verdade, e o que acontece é uma interpretação incorreta, e os que criticam não sabem realmente como funciona esta prática.

Na programação em pares, dois desenvolvedores trabalham em um mesmo computador, onde um fica codificando e o outro avaliando o que o codificador está digitando. Dessa maneira, pequenos erros que poderiam gerar grande perda de tempo para descoberta futura, podem ser instantaneamente

corrigidos através da interferência do profissional que está observando o outro codificar. (BECK, 2004a)

Outro benefício que pode ser citado, é que quando surge uma dúvida ou incerteza em uma implementação de um método, por exemplo, o colega expõe suas idéias e sugere uma solução, agregando o conhecimento dos dois profissionais na implementação. Há uma comunicação constante entre o par, de forma a garantir que as melhores idéias dos dois sejam implementadas e escolhidas as melhores soluções para os problemas. (TELES, 2006)

2.2.2.8 Código coletivo

A programação em par favorece que o código seja compartilhado por toda equipe através da técnica e também possibilitando que os pares possam se revezar para que todos compartilhem o conhecimento.

A prática do código coletivo no XP faz com que cada desenvolvedor conheça todas as partes do sistema e se sinta capaz de realizar qualquer alteração. O conhecimento não fica concentrado em determinados profissionais, objetivando a agilidade quando precisar fazer uma refatoração no sistema. (TELES, 2006)

Com os tradicionais, este conhecimento muito vezes é centralizado, fazendo com que caso ocorra algum problema na equipe e um novo desenvolvedor precise alterar o código, perderá muito tempo até entender como ele funciona, gerando mais tempo perdido no projeto. Isto quando existe tempo, pois caso receba uma tarefa, um erro para corrigir e seu tempo estiver muito escasso, ele irá acabar codificando algo muito errado, simplesmente para atender aquele problema, não importando o que foi codificado e talvez sem realizar os testes, que poderá futuramente acarretar em um grande problema.

2.2.2.9 Integração contínua

Em muitos projetos de software, o sistema é divido em módulos de modo que cada programador fique responsável por desenvolver um determinado módulo. Os desenvolvedores implementam baseados em interfaces para que depois o software possa ser integrado facilmente. Mas desta forma além do tempo de espera para acontecer a integração, muito problemas ocorrem porque os códigos nem sempre são implementados de acordo com as interfaces estipuladas.

Por isso no XP isso não ocorre, como já foi visto, o código é coletivo e a integração é contínua. Segundo Jeffries (et al. 2001 apud TELES, 2006) a equipe deve integrar com a maior freqüência possível, normalmente integrar e testar várias vezes por dia.

2.2.2.10 Ritmo sustentável

O XP defende a idéia de que a semana de trabalho deve ter 40 horas de trabalho. Quando o prazo está se esgotando e o projeto não está pronto, algumas empresas fazem com que seus funcionários passem a fazer horas extras, trabalhar até tarde da noite ou talvez em finais de semana. Isto não irá agregar muito valor ao projeto, pois talvez o sistema fique pronto no prazo estipulado, mas muitos problemas futuros poderão acontecer. (BECK, 2004a)

A área de desenvolvimento, exige muita criatividade dos profissionais para que possam implementar as melhores soluções a fim de atender as necessidades do cliente, e quando estes são submetidos a jornadas longas de trabalho, a produtividade e a capacidade de raciocínio diminuem, e possivelmente irão afetar o projeto.

Alguns gerentes de projeto acham que o trabalho na empresa de software é como na indústria, mas sabe-se que é totalmente diferente, pois enquanto uma exige atividades repetidas possibilitando que maquina e pessoa trabalhem mais horas por dia, a outra exigem mais agilidade principalmente da pessoa, que não pode trabalhar como uma máquina. (TELES, 2006)

No XP o ritmo deve ser sustentável, não se deve exigir demais dos profissionais, pois estes também possuem uma vida fora da empresa. A empresa

deve exigir que ele trabalhe 8 horas diárias para que este se sinta motivado, possa produzir mais, e agregar mais resultados positivos para a empresa.

2.2.2.11 Cliente presente

No XP é fundamental a presença do cliente durante o desenvolvimento do projeto, para que assim as dúvidas possam ser esclarecidas mais rapidamente, o cliente possa fornecer constantes feedbacks, expressar suas necessidades através de histórias, tudo para facilitar o desenvolvimento e concluir o projeto com sucesso.

O ideal é que o cliente presente seja aquele que trabalhe diretamente no negócio, que irá utilizar o sistema, como por exemplo, num sistema de vendas, o vendedor seja esta pessoa, de forma a garantir que suas idéias foram realmente implementadas e para que pequenos ajustes possam ser feitos durante o desenvolvimento. (BECK, 2004a)

2.2.2.12 Padrões de codificação

Para que a equipe possa aplicar as práticas do código coletivo, da refatoração, dos testes unitários, deve haver uma padronização no código aplicada por cada desenvolvedor para que o trabalho possa ser realizado. Imagine se cada desenvolvedor implementar de acordo com seu estilo de codificar. Isso poderia não ter efeito em projetos em que as partes do sistema são construídas separadamente, mas no XP, onde toda equipe deve conhecer o código, isso não é válido.

Por isso, no início do projeto a equipe deve optar por um padrão na codificação para simplificar o trabalho posterior e permitir que todas possam navegar pelo código e entender de forma clara o que o sistema está realizando. Pode-se construir um padrão em consenso da equipe ou utilizar um já existente que esteja documentado. (TELES, 2006)

3 MELHORIA DE PROCESSOS E PADRÕES DE QUALIDADE

A qualidade dos softwares é fator determinante para o sucesso da empresa e permanência no mercado competitivo. Qualidade esta que não se apresenta somente nos produtos ou serviços, mas também deve ser apresentada sobre os processos de produção do software. Desta maneira, possuir processos produtivos eficientes e eficazes baseados em padrões internacionais de qualidade é fator diferencial para uma empresa. (SOFTEX, 2010b)

Pode-se então salientar dois modelos de referência para padronização e melhoria dos processos da organização: o CMMI, modelo internacional, e o MPS.BR, modelo criado no Brasil mas que começa a ser reconhecido internacionalmente a partir de alguns países da América latina.

3.1 CMMI - CAPABILITY MATURITY MODEL INTEGRATION

Em 1984, o departamento de defesa americano criou a SEI (Software Engineering Institute), instituto no qual criou o CMM (Capability Maturity Model), que era a versão anterior do CMMI. O CMMI é uma evolução do CMM, e seu modelo original foi publicado no ano de 2000. (SEI, 2010)

CMMI se tornou um modelo de referência mundial em melhoria de processos de empresas com foco principal na qualidade de desenvolvimento de software. Seu objetivo é padronizar todas as tarefas que ocorrem num processo de desenvolvimento para que a equipe saiba quais atividades desempenhar, e que cada membro tenha clareza das suas tarefas que estão ligadas ao atendimento das metas da empresa. Além disso, objetiva integrar todos os setores da empresa de forma que as atividades também sejam executadas em integração. (SEI, 2010)

Segundo relatório de Goldenson (et tal, 2010), realizado com 35 empresas de software, alguns dos benefícios que o CMMI traz para a empresa são: redução de custo em 34%, diminuição do tempo em 50%, aumento de produtividade em 61%, da qualidade em 48% e da satisfação do cliente em 14%.

O CMMI traz um enorme reconhecimento de organização e maturidade para empresa. Existem algumas empresas que já estão exigindo que a contratada para desenvolver um software tenha ao menos o CMMI nível 2. E como já foi citado algumas vezes neste trabalho, com a globalização atingindo as empresas de software, uma empresa que conseguir evoluir seus processos e obter avaliação positiva para atingir os níveis do CMMI, acabará abrindo as portas de exportação para esta empresa, já que o CMMI é reconhecido internacionalmente.

São vários os benefícios que ele pode trazer para a empresa. Mas é claro, tudo tem seu preço. Hoje, um dos impedimentos que acabam gerando um obstáculo para as empresas são os altos custos para implantação e adequação do CMMI, além do tempo e esforço que devem ser empregados.

O CMMI possui duas formas de representação, que são dois caminhos diferentes que a empresa pode adotar para conseguir melhorar seus processos: a representação contínua e a por estágios. A primeira, que utiliza níveis de capacidade, a evolução é realizada continuamente através da melhoria de processos relacionados com áreas de processo individuais ou conjunto de áreas escolhidas pela empresa; a segunda utiliza níveis de maturidade, onde cada nível possui um conjunto já estabelecido de áreas de processo. (KOSCIANSKI et al., 2007)

Neste trabalho será abordada somente a representação por estágios, que é a mesma utilizada no CMM.

A representação por estágios é dividida em 5 níveis de maturidade, onde o mais baixo é o menos maduro. Cada nível de maturidade possui um conjunto de áreas de processo, no total elas são 25, como pode ser observado na figura 1.

NÍVEIS	ÁREAS DE PROCESSO
5 – OTIMIZADO	Inovação e implantação organizacional Análise e resolução de causas
4 – GERENCIADO QUANTITATIVAMENTE	Gerência quantitativa do projeto Desempenho do Processo organizacional
3 - DEFINIDO	Desenvolvimento de requisitos Solução técnica Verificação Validação Integração do Produto Foco no Processo organizacional Definição do processo organizacional Treinamento organizacional Gerência de Projeto Integrada Gerência integrada de fornecedores Gerência de riscos Análise de decisão e resolução Integração da equipe Ambiente organizacional para integração
2 - GERENCIADO	Gerência de requisitos Planejamento de projeto Monitoração e controle do projeto Gerência de acordo com fornecedores Medição e análise Garantia da qualidade do processo e do produto Gerência de configuração
1 - INICIAL	

Figura 1: Níveis de maturidade e área de processo do CMMI
Fonte: Elaborado pelo Autor

3.1.1 Os níveis do CMMI por estágio

Como apresentado na figura 1, a representação por estágio possui 5 níveis de maturidade. A empresa sempre começa do nível 1, e vai adquirindo maturidade através da contemplação dos requisitos de cada nível sucessivamente. Não existe possibilidade de pular mais de um nível a cada vez, como por exemplo, passar do nível 1 para o 3, deve-se contemplar cada nível que é um pré requisito para o nível superior. A mudança de nível não é uma tarefa fácil, e exige muitas vezes uma grande mudança na organização, em todos processos e rotina. É um processo lento e que custa bastante dinheiro. (BARTIÉ, 2002)

O nível mais baixo do CMMI, o um, chamado de inicial, é quando não existe controle nos processos da empresa, os processos são imaturos, onde o software é produzido de maneira desordenada e sem controle.

Comumente ocorrem problemas nos projetos destas empresas, relacionados a prazos, custos e funcionalidades, mas isto não significa que os seus produtos sejam ruins.

Estas empresas dependem de talentos individuais que praticamente vivem dentro da empresa carregados de horas extras podendo desta forma oferecer um bom produto. Isto ocorre devido à falta de planejamento, e desta forma os poucos processos estabelecidos acabam sendo atropelados quando os prazos estão apertados. Uma das etapas importantes que pode ser até deixada pra trás neste caso é a de testes, gerando um grande problema futuro. (KOSCIANSKI et al., 2007)

Cerca de 85% das empresas ainda estão neste nível, e a tendência é que com o tempo estas acabem condenadas se não evoluírem e gerenciarem melhor os seus processos. (BARTIÉ, 2002)

No segundo nível, o gerenciado, o foco está na gestão básica do projeto e possui sete áreas de processo. Um dos objetivos é tornar corporativos os processos de gestão de software para desta forma poderem ser realizadas estimativas a partir da experiência de projetos anteriores. Os projetos são executados de acordo com o que é planejado, sendo organizados e disciplinados com planos realistas. Neste nível, começa-se a medir e monitorar o desempenho dos produtos de software gerados para ajudar no processo de tomada de decisão. Caso surja algum problema relacionado com o prazo, com a gestão básica do projeto, o gerente poderá ser capaz de identificar este problema e encontrar uma solução em um tempo hábil. (KOSCIANSKI et al., 2007)

O nível três, o definido, possui 21 áreas de processo, onde sete pertencem ao nível anterior e outras 14 a este nível, então para contemplar o terceiro nível, a empresa precisa estar realizando todas as áreas de processo do nível dois e mais as do nível três, assim como irá acontecer também nos próximos níveis.

O foco no nível três é a padronização de processo, onde estes agora estão documentados e integrados entre as áreas da empresa. É possível observar a evolução de cada projeto, pois neste nível já se conhece bem os processos, os produtos de software são controlados e a qualidade é medida, e ainda, as atividades da equipe estão claras para cada membro.

No nível quatro, gerenciado quantitativamente, com 23 áreas de processo, o foco é o gerenciamento quantitativo, onde os processos são

gerenciados, definidos e controlados com dados estatísticos, para que o desempenho dos processos possa ser avaliado e analisado quais estão influenciando no atendimento dos objetivos da empresa. Os dados quantitativos que servem para medir a qualidade e desempenho dos processos são armazenados para auxiliar numa futura tomada de decisão, e desta forma as decisões corretas podem ser repetidas em outros projetos. (KOSCIANSKI et al., 2007)

O nível máximo de maturidade do CMMI é o nível cinco, otimizado, com 25 áreas de processo e foco no aperfeiçoamento contínuo dos processos. Neste ponto, deve-se conhecer detalhadamente todos os processos que acontecem na empresa, com dados estatísticos, desempenhos, tempo, e analisado o que pode ser melhorado para obter maior qualidade no atendimento dos objetivos.

3.2 MPS.BR - MELHORIA DE PROCESSO DO SOFTWARE BRASILEIRO

Com a grande competitividade no mercado de software, adotar um modelo de referência para obter melhoria de capacidade de desenvolvimento é um fator determinante. Como apresentado no item 3.1, o CMMI, é um modelo de referência conhecido internacionalmente, mas para adequação da empresa, exige um processo muito demorado e custos um pouco fora da realidade da maioria das pequenas e médias empresas brasileiras.

Esses dois fatores apresentados, tempo e custos, acabam gerando pouca demanda para adequação ao CMM, com 29 avaliações, e CMMI com 10 avaliações, até o ano de 2006 no Brasil, segundo informações do MCT (2010).

Devido a isso, em 2003 a Associação para Promoção da Excelência do Software Brasileiro (SOFTEX), com o apoio do MCT (Ministério da Ciência e Tecnologia), Financiadora de Estudos e Projetos (FINEP), Serviço Brasileiro de Apoio às Micro e Pequenas Empresas (SEBRAE) e Banco Interamericano de Desenvolvimento (BID), lançou um modelo para melhoria da capacidade de desenvolvimento do software brasileiro, o MPS.BR, que foi baseado na norma

internacional ISO/IEC 12207:2008[1], na ISO/IEC 15504[2] e no modelo CMMI, além de ser constituídos de boas práticas da engenharia de software, tudo voltado para os negócios das empresas brasileiras e com custos mais acessíveis. (SOFTEX, 2010b)

O MPS.BR está dividido em três componentes: método de avaliação (MA-MPS), que visa orientar os profissionais para realização da avaliação, modelo de negócio (MN-MPS), que possui as regras para implementação do modelo de referência, e modelo de referência (MR-MPS), no qual possui os requisitos a serem atingidos pelas empresas que desejam obter a avaliação do MPS. (KOSCIANSKI et al., 2007)

O MPS.BR está organizado em 4 guias: o guia geral, que possui informações gerais sobre o MPS e apresenta em detalhes o modelo de referência; o de implementação, no qual possui informações para implantação do MR-MPS e é dividido em 10 guias, onde 7 delas são referentes ao níveis do MR que vão do ‘G’ ao ‘A’, e mais 3 guias para diferentes tipos de organizações; o de aquisição, que possui informações para empresas que queiram adquirir software; e o de avaliação, que contém informações sobre os métodos de avaliação. (SOFTEX, 2010a)

O MR possui níveis de maturidade como no modelo CMMI. Estes níveis podem ser comparados com o do modelo internacional, onde os níveis ‘G’ e ‘F’ do MPS equivalem ao nível 2 do CMMI, os níveis ‘E’, ‘D’ e ‘C’ ao 3, o ‘B’ ao 4 e o ‘A’ ao 5. São sete níveis no modelo, ao invés de cinco, possibilitando uma implantação mais gradual e adequada para realidade das pequenas e médias empresas brasileiras. (KOSCIANSKI et al., 2007)

Para cada nível de maturidade do MPS.BR existe um ou mais processos e para cada processo um conjunto de atributos de processos, que é uma característica da capacidade do processo aplicável a qualquer um destes e são divididos em 9 para formar a capacidade do processo. A capacidade do processo representa o grau de refinamento com que o processo é executado na organização, relacionando-se com o atendimento aos atributos de processo de cada nível de maturidade, sendo que quanto maior o nível que a organização deseja chegar, maior o nível de capacidade deverá possuir. (KALINOWSKI et al., 2010)

[1] ISO/IEC 12207:2008 - Contêm processos, atividades e tarefas a serem aplicadas durante o fornecimento, aquisição, desenvolvimento, operação, manutenção e descarte de produtos de software, bem como partes de software de um sistema. (SOFTEX, 2010b p.14)

[2] ISO/IEC 15504 – possui normas para avaliações de processos de software com dois objetivos: a melhoria de processos e a determinação da capacidade de processos de uma unidade organizacional. (SOFTEX, 2010b p.15)

Para se conseguir atingir um determinado nível de maturidade, todos os resultados esperados do processo e dos atributos de processo (RAP), no total de 46 RAP, deverão ser atendidos. Os níveis são cumulativos, então significa que para obter, por exemplo, o nível 'F', é necessário atender também os processos e os AP do nível 'G'. (SOFTEX, 2010b)

O conjunto de atributos de processo (AP) são nove:

- AP 1.1 O processo é executado;
- AP 2.1 O processo é gerenciado;
- AP 2.2 Os produtos de trabalho do processo são gerenciados;
- AP 3.1. O processo é definido;
- AP 3.2 O processo está implementado;
- AP 4.1 O processo é medido;
- AP 4.2 O processo é controlado;
- AP 5.1 O processo é objeto de melhorias e inovações;
- AP 5.2 O processo é otimizado continuamente.

Na figura 2, podem-se observar os níveis de maturidade do MR-MPS, os processos e atributos de processo para cada nível.

Nível	Processos	Capacidades (AP)
A	*(sem processos adicionais)*	*1.1, 2.1, 2.2, 3.1, 3.2, 4.1*, 4.2*, 5.1*, 5.2**
B	*Gerência de Projetos (evolução)*	*1.1, 2.1, 2.2, 3.1, 3.2, 4.1*, 4.2**
C	*Gerência de Riscos, Desenvolvimento para Reutilização, Gerência de Decisões*	*1.1, 2.1, 2.2, 3.1, 3.2*
D	*Desenvolvimento de Requisitos, Integração do Produto, Projeto e Construção do Produto, Validação, Verificação*	*1.1, 2.1, 2.2, 3.1, 3.2*
E	*Avaliação e Melhoria do Processo Organizacional, Gerência de Projetos (evolução), Gerência de Recursos Humanos, Gerência de Reutilização, Definição do Processo Organizacional*	*1.1, 2.1, 2.2, 3.1, 3.2*
F	*Aquisição, Garantia da Qualidade, Gerência de Configuração, Gerência de Portfólio de Projetos, Medição*	*1.1, 2.1, 2.2*
G	*Gerência de Projetos, Gerência de Requisitos*	*1.1, 2.1*

** Estes APs capacitam apenas um conjunto de processos selecionado pela organização de acordo com seus objetivos de melhoria. Os demais APs precisam capacitar todos os processos do nível pretendido*

Figura 2: Níveis e atributos de processo do MPS.BR
Fonte: Kalinowski et al., 2010

Segundo Kalinowski (et al., 2010) até setembro de 2009, 174 organizações foram avaliadas para obtenção de algum nível do MPS.BR, o que

mostra um futuro promissor para o modelo, devido as várias empresas estarem buscando a avaliação.

Destas empresas avaliadas, a maioria delas ficou muito satisfeita com os resultados apresentados pela imposição da melhoria dos processos na organização, e salientaram os resultados positivos que obtiveram, como aumento no faturamento, número de clientes no país, retorno de investimento, produtividade e satisfação do cliente. (TRAVASSOS et al.,2010)

Além destes resultados citados, pode-se observar a adoção de boas práticas da engenharia de software, já que algumas constam nos resultados de processos do modelo de referência, como por exemplo, revisão por pares, testes e reutilização. Para Nogueira (2006 apud Kalinowski et al., 2010 p.9), “as organizações só implementam as boas práticas da engenharia de software quando estas são exigidas para certificações”, e sendo assim, o modelo está contribuindo para adoção destas práticas trazendo mais benefícios para as empresas.

3.2.1 Os níveis do MPS.BR

Começando pelo nível G (Parcialmente gerenciado), o nível mais baixo de maturidade envolve os processos mais críticos para gerenciamento, e neste nível a empresa deve focar seus esforços no planejamento, monitoramento e controle dos projetos e também no controle dos requisitos. (SOFTEX, 2010b)

O nível F (Gerenciado) é onde a empresa deve melhorar o controle sobre os processos e projeto, voltando sua atenção para a qualidade destes, podendo avaliar o desempenho através de dados quantitativos. Estes dois primeiros níveis são o passo inicial que a empresa deve dar na busca da melhoria da qualidade de seus processos e maior reconhecimento. Mas ainda o que ocorre até o nível F é que existem muitas responsabilidades e conhecimentos importantes concentrados em algumas pessoas, fato este que não é nada bom, pois em algum momento pode acontecer do profissional ter que sair da empresa e assim o projeto em desenvolvimento ficar comprometido. (KALINOWSKI et al., 2010)

Por isso, no nível E (Parcialmente definido), institui-se processos padrão na empresa, e foca-se na melhoria destes processos para que toda a equipe possa

se basear neles e não ficar desorientada se algum membro sair, e assim poderá continuar o projeto sem problemas. (KOSCIANSKI et al., 2007)

No nível D (Largamente definido), como os processos já estão estabelecidos, agora outros processos começam a ser detalhados, como os processos de engenharia de software mais técnicos, associados a requisitos, arquitetura, verificação e validação, por exemplo. (SOFTEX, 2010b)

O nível C (Definido) possui processos complementares à gestão de projetos, ligados a gestão de riscos e ao apoio da tomada de decisão. Para possibilitar a reutilização de artefatos de software, foi adicionado neste nível o processo de desenvolvimento para reutilização. (KALINOWSKI et al., 2010)

O nível B (Gerenciado quantitativamente), visa controlar os dados quantitativos referentes aos processos e as variações que podem ocorrer com estes. Após este nível, temos o A (Em otimização) que não possui processos, somente atributos de processo voltados para inovações e resolução de problemas, ou seja, focados na melhoria contínua. (SOFTEX, 2010b)

3.3 MPT.BR – MELHORIA DO PROCESSO DE TESTE BRASILEIRO

Os testes de software são de extrema importância no desenvolvimento do software, pois através deles se conseguirá atingir uma melhor qualidade no produto final.

Os modelos de referência para melhoria da qualidade dos processos MPS.BR e CMMI, trazem muitos benefícios para as empresas, como pôde ser observado nos itens 3.1 e 3.2**,** mas para a Associação Latino Americana de Teste de Software (ALATS), a implementação destes modelos e seus níveis de maturidade são insuficientes para que os resultados tenham uma melhora significativa na área de testes. Por isso, a ALATS e a RIOSOFT (Sociedade Núcleo de Apoio à Produção e a Exportação de Software), estão desenvolvendo um modelo de maturidade para ser aplicado especificamente na área de testes de software, conhecido como modelo de melhoria do processo de teste brasileiro, MPT.BR. (ALATS, 2010)

O MPT está sendo desenvolvido para ser aplicado em organizações que implementaram o MPS ou o CMMI para que consigam melhorar também a qualidade

dos processos da área de testes, já que este modelo é compatível com os modelos citados. O objetivo do modelo de testes é possibilitar que pequenas equipes da área de testes possam adquirir maior maturidade e conseqüentemente mais qualidade, sem ter que realizar altos investimentos. (ALATS, 2010)

Assim como no MPS.BR, o MPT.BR é voltado para realidade brasileira e possui 7 níveis de maturidade, mas com a diferença de que neste modelo eles são apresentados por números, enquanto que no outro os níveis são representados por letras. O nível mais baixo é o 1, e foi adotado dessa maneira pois sabe-se que existem menos processos a serem observados na área de testes do que no desenvolvimento do software, e este primeiro nível contempla somente uma área de processo, tornando mais simples e mais rápida sua implementação do que o primeiro nível dos modelos de maturidade para o desenvolvimento.

Os níveis de maturidade são avaliados através de áreas de processo, e estas por sua vez são compostas de práticas específicas, genéricas e objetivos, que deverão ser atendidos pela área de testes para que esta consiga adquirir mais qualidade. Os níveis e suas áreas de processo, bem como a relação com os outros modelos de processo podem ser observados na figura 3.

Nível MPT	Áreas de Processo	MPS.BR	CMMI
5	Verificação Validação	D	Sem relação
4	Gerência de Recursos humanos Gerência de Reutilização Gerência de Riscos	E	Sem relação
3	Aquisição Gerência de configuração Garantia da qualidade Medição	F	2
2	Gerência de requisitos de teste	G	Sem relação
1	Gerência de Projetos de teste	Sem relação	Sem relação

Figura 3: Níveis e áreas de processo do MPT e relação com MPS.BR e CMMI
Fonte: Elaborado pelo Autor

Como o modelo ainda está em desenvolvimento, atualmente só existem avaliações para o nível 1 e 2, sendo que algumas empresas já estão em processos de avaliação, e os outros níveis ainda estão em processo de desenvolvimento.

A respeito do nível 2, como pode ser observado na figura 3 e comparando com a figura 2 do modelo MPS, é possível ver que é equivalente ao nível G, então se a empresa estiver implementando o MPS, por exemplo para atender o atributo de processo Gerência de Projetos, com um esforço pode expandir este AP até a área de testes e assim irá facilitar a avaliação para o nível 1 do MPT, podendo acontecer isso também com o nível 2.

Os níveis são equivalentes, mas existem mais algumas práticas e objetivos para se cumprir, mas a empresa já estará encaminhando a área de testes para avaliação. Como pode ser observado também, as áreas de processo são específicas para a área de testes, então a gerência de requisitos de testes, se empenhará em controlar os requisitos desta área, mas poderá também controlar os requisitos do desenvolvimento.

O MPT.BR é um modelo novo mas que está apoiado nas idéias do MPS.BR. Tanto que uma das criadoras, a Riosoft, é agente da Softex, coordenadora do projeto do MPS. Ele parece ser um modelo promissor, pois está voltado para a realidade brasileira, possui processos equivalentes ao MPS, está focado diretamente na área de qualidade da empresa, é leve, e o custo de implantação é baixo relacionado com outros modelos internacionais. (ALATS, 2010)

3.4 NORMA SQUARE: ISO/IEC 25000: 2005

A norma ISO/IEC 25000 é uma das mais importantes em relação à caracterização e medição de qualidade de produtos de software. Esta norma engloba as anteriores que tratavam sobre este tema, as normas ISO/IEC 9126 e a ISO/IEC 14598. Conhecida como norma Square (Software Product Quality Requirements and Evaluation), ou em português, requisitos de qualidade e avaliação de produtos de software.

A ISO/IEC 25000 reorganizou os documentos contidos nas outras duas normas, sendo que os documentos destas anteriores foram aproveitados, tornando-se uma só norma, mais completa, focada na qualidade dos produtos de software. Diante desta junção, a norma foi então dividida em cinco partes essenciais:

gerenciamento, modelo de qualidade, medição, requisitos de qualidade e avaliação. (KOSCIANSKI et al., 2007)

Na parte referente ao modelo de qualidade, onde ficaram contidos alguns documentos da norma 9126, podem ser encontradas informações sobre qualidade externa, a qual não se preocupa com a estrutura interna do software, mas sim com o seu funcionamento, e sobre a qualidade interna que diz respeito à estrutura interna do software, focando na qualidade e organização do código fonte, e sobre a qualidade em uso, aquela que é verificada a partir do usuário, quando o software está em produção.

De acordo com a norma, para se garantir a qualidade do software, deve-se estabelecer objetivos de qualidade juntamente com os requisitos, para que se consiga medir os dados no decorrer do projeto e verificar se estão atingindo a qualidade esperada.

3.5 OUTROS MODELOS DE MELHORIA E NORMAS

Existem diversos outros modelos de melhoria de processos de testes de software e normas de qualidade, algumas utilizadas no Brasil, outras somente conhecidas.

A norma ISO 9000 é mundialmente conhecida e referência em termos de qualidade. Na verdade ela é composta por um conjunto de normas, e pode ser utilizada por qualquer organização, independente do ramo de atividade, para realizar o controle de qualidade dos produtos ou serviços oferecidos. (KOSCIANSKI et al., 2007)

Existe também a versão brasileira desta norma, a NBR ISO 9000, que está dividida em três partes, e assim como a versão original, trata a respeito da qualidade dos processos das empresas e do controle para garantia da qualidade.

Em relação à melhoria de processos de teste, existem vários modelos, mas que não são aplicados no Brasil. Alguns deles são: Testability Suport Model (TSM), Test Process Improvement (TPI), e o mais conhecido destes, o Testing Maturity Model integration (TMMi), que é baseado no CMMI.

O TMMi possui 5 níveis de maturidade, e assim como o MPT.BR, busca qualificar melhor os processos da área de teste e tornar os testadores mais qualificados para realizar a função. Porém, assim como os outros modelos de melhoria internacionais, o custo de implantação do TMMi é altíssimo e trabalhoso, e além disso, no Brasil ainda não existe uma entidade responsável pela implantação e avaliação do modelo, fator este que contribui para que o MPT.BR tenha sucesso. (ALATS, 2010)

4 QUALIDADE E TESTE DE SOFTWARE

Apesar dos primeiros indícios de padrões de qualidade serem datados do Egito antigo, com a implantação de unidades de medida, o marco principal na história da qualidade ocorreu com a revolução industrial, assim como mudanças na economia e sociedade, onde com o grande crescimento das indústrias surgiu a competitividade entre elas obrigando-as a buscar um processo de melhoria contínua e eficiência para poder garantir sua permanência no mercado. (KOSCIANSKI et al., 2007)

Ao longo dos anos, na história da qualidade, ela tem se tornado cada vez mais importante. Nos anos 40, começaram a surgir os primeiros órgãos de controle de qualidade como a ABNT (Associação Brasileira de normas técnicas) e a ISO (International Standarts Organization). (KOSCIANSKI et al., 2007)

No início do desenvolvimento de software, não havia nenhum controle de qualidade, ninguém tinha compromisso do que estava sendo desenvolvido. Já nos anos 80, o compromisso era descobrir os erros, e na década de 90 o foco foi o negócio, sendo que no final da década, a qualidade de software passou a ser importante e vem evoluindo no decorrer destes anos. (MOLINARI, 2003)

Durante estes anos foi que surgiram mais algumas organizações e padrões que se tornaram referência no mundo até hoje. Dentre estes pode-se destacar o IEEE (Institute of Eletrical and Eletronics Enginneers), ANSI (American National Standarts Institute), CMM (Capability Maturity Model) e mais tarde o brasileiro MPS.BR (Melhoria de Processos de Software Brasileiro). (BARTIE, 2002)

Como já citado no item 1.1, o constante avanço da tecnologia no mundo globalizado obriga as empresas desenvolvedoras de softwares a buscar soluções para garantir a permanência no mercado, produzindo softwares com eficiência e qualidade. Ao longo dos anos, várias metodologias de desenvolvimento estão surgindo, introduzindo novas práticas e tentando buscar a melhor maneira de produzir software com maior qualidade.

Atualmente, com a busca da maior informatização dentro das empresas e com o surgimento dos grandes sistemas prontos, chamados ERP (Enterprise Resource Planning), que integram todos os dados de uma organização em um só sistema, a complexidade de desenvolvimento está também cada vez maior, e conseqüentemente, torna-se mais complexo desenvolver estes novos softwares com uma boa qualidade. (BARTIÉ, 2002)

Ainda existem metodologias utilizadas pelas empresas que podem ser consideradas ultrapassadas, onde a atividade de testes é realizada pelos próprios desenvolvedores navegando pelas linhas de código e corrigindo problemas, e sendo realizada somente no final do desenvolvimento, quando todas as funcionalidades tiverem sido implementadas. (BARTIE, 2002)

Empresas que ainda trabalham desta forma, que não possuem uma equipe específica na área de testes, ou que não aplicam os testes no decorrer do desenvolvimento, estão ficando para trás, não estão acompanhando o avanço da tecnologia e com o passar do tempo poderão perder seu espaço no mercado.

Manter a qualidade nos softwares é uma questão de permanência neste mercado cada vez mais competitivo. Um dos problemas que comumente ocorrem hoje são os pequenos prazos negociados pelo setor comercial das empresas de software, que acabam estourando na equipe de desenvolvimento, que sem tempo para produzir um bom produto e trabalhando sobre pressão, acaba produzindo o produto sem realizar todos os testes, causando grandes problemas futuramente e gerando conseqüências desagradáveis.

Por isso, com o passar dos anos, tem-se observado e analisado melhor a área da garantia da qualidade no desenvolvimento de software, como Bartié (2002) mostra na tabela abaixo.

Evolução das organizações desenvolvedoras de software			
Características	1960	1980	2000
Tamanho do software	Pequeno	Médio	Muito Grande
Complexidade do software	Baixa	Média	Alta
Tamanho da equipe de desenvolvimento	Pequeno	Médio	Grande
Padrões e Metodologias de desenvolvimento	Interno	Moderado	Sofisticado
Padrões e Metodologias de Qualidade e testes	Interno	Emergente	Sofisticado
Organizações de Qualidade e Testes	Bem poucas	Algumas	Muitas
Reconhecimento da importância da qualidade	Pequeno	Algum	Significante
Tamanho da equipe de Qualidade e Testes	Pequeno	Pequeno	Grande

Tabela 1: Evolução do processo de qualidade e de testes de software
Fonte: Bartié 2002 p.5

Para Bartié (2002, p.16), “Qualidade de software é um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos”.

Para produzir software com qualidade, deve-se observar e avaliar cada atividade no desenvolvimento do projeto, e para isso, uma organização deve possuir um processo mínimo de desenvolvimento, seja ágil ou tradicional, porque o que acontece é que muitas empresas possuem um processo somente no papel, quando o projeto está em desenvolvimento, cada equipe trabalha de uma forma, e assim fica difícil estabelecer um padrão de qualidade para o projeto. (KOSCIANSKI et al., 2007)

> “O desenvolvimento de software com qualidade é um assunto amplo, complexo e ainda muito discutido. São vários os fatores que precisam ser considerados para obtermos um resultado satisfatório... o principal indicador de qualidade no desenvolvimento de qualquer produto, inclusive o software, é a satisfação do cliente.” (INTHURN, 2001 p. 21)

Segundo Pressman (2006), a qualidade de um sistema também está diretamente ligada ao atendimento das necessidades do cliente, se o produto entregue a ele atendeu suas expectativas, supriu as necessidades e ainda trouxe benefícios para o seu negócio, o cliente até tolera que eventuais problemas no

sistema ocorram, por isso que para completar o quesito qualidade, deve-se obter a satisfação do cliente.

Uma questão que também vem sendo avaliada pelas empresas hoje em dia, e que influencia na qualidade do produto, diz respeito aos seus recursos humanos. Algumas empresas começam a se preocupar em oferecer melhor qualidade de vida para seus funcionários. O projeto tem que ser bem planejado porque quanto mais pressão existir, menor a qualidade. O funcionário tem família, e uma vida fora da empresa. Quando este é submetido a rotinas pesadas, com cronograma estourado e muitas horas extras, a produtividade com certeza irá cair e com ela a qualidade. Uma empresa que acredita e que aposta em seus recursos humanos terá um bom retorno, por isso deve-se investir em promoções, gratificações, valorizar o seu maior bem para que este possa dar o seu melhor para produzir produtos com mais qualidade.

Para que não haja pressão referente a prazos, o que comumente acontece em empresas de software, é imprescindível que se tenha interação entre o comercial e o desenvolvimento, uma sintonia para que ambos possam trabalhar normalmente e para que não surjam atritos. Esse também é um dos fatores que acaba gerando problemas dentro de uma empresa, quando acontece algum desentendimento entre departamentos, isso afeta também na produtividade e na qualidade. Por isso, uma saída interessante seria criar uma área intermediária, formada por desenvolvedores e vendedores, para que os prazos pudessem ser melhor negociados de acordo com a dificuldade técnica envolvida. (KOSCIANSKI et al., 2007)

4.1 OS PROBLEMAS DE QUALIDADE

Os defeitos que ocorrem nos sistemas são conhecidos por vários nomes. De uma maneira geral, bugs, falhas, erros e defeitos, são iguais para muitas pessoas. Para Molinari (2003), existe uma distinção entre bug e defeito, onde o bug ou falha é aquele problema que ocorre quando o software está sendo usado e ninguém havia percebido tal erro antes, e defeito acontece quando o software é implementado para fazer uma determinada função e não faz.

Esta definição não é confirmada pela maioria da comunidade da engenharia de software, que considera todos esses problemas como um só, mas o que se deve ter em consenso são os impactos causados por estes problemas dependendo do tempo ou fase em que são descobertos.

A qualidade deve estar presente durante todo o projeto, pois como pode ser observado na figura 4, segundo Bartié (2002) os problemas ocorrem em todo o projeto, e principalmente nas fases iniciais.

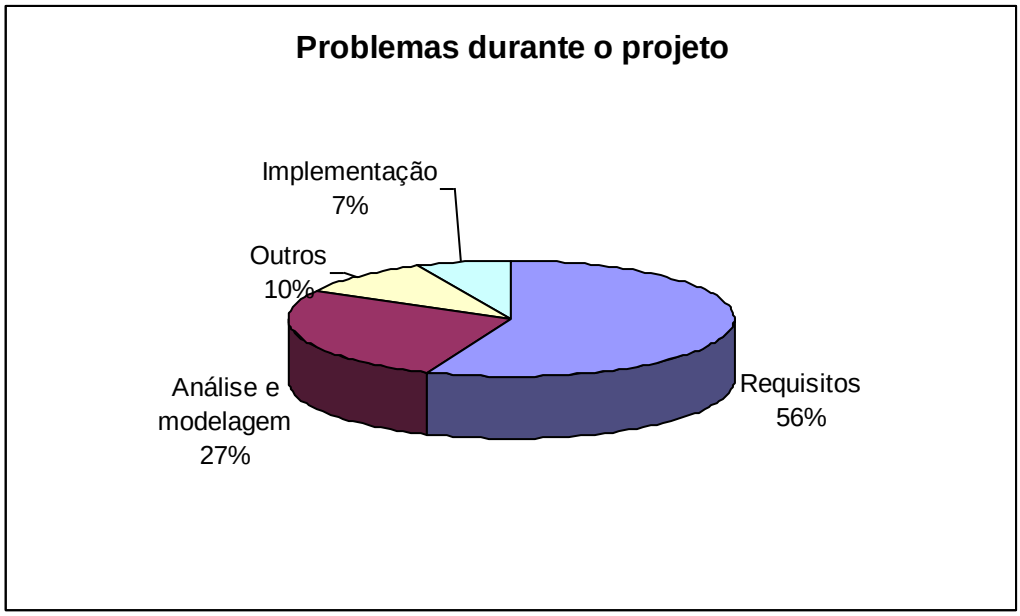

Figura 4: Os problemas em cada fase do projeto
Fonte: Elaborado pelo Autor

Segundo Myers[3] (1979 apud Bartié, 2002), quando mais tarde o erro é descoberto, mais cara se tornará a sua correção, envolvendo o processo de identificação, novo código para correção e testes, e ele afirma que quando um erro não é identificado, os custos multiplicam-se por dez a cada fase seguinte. Este fato é chamado de regra de 10, sugerida por Myers e apresentada na figura 5**.**

[3] Glenford Myers – Respeitável estudioso no assunto de qualidade de software possuindo vários estudos e publicações sobre testes de software.

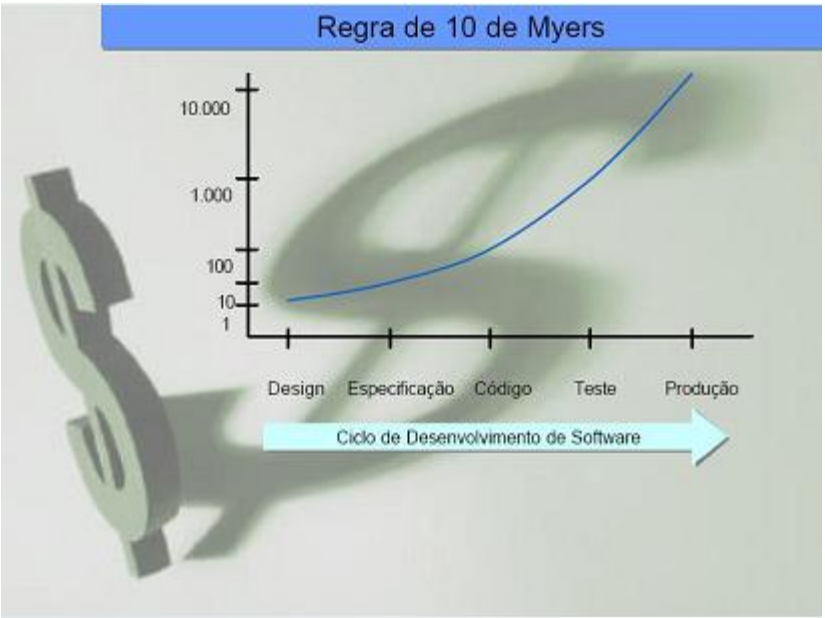

Figura 5: Regra de 10 de Myers
Fonte: Alats, 2010.

Este é um dos motivos que culminou no surgimento de novas metodologias de desenvolvimento e práticas ágeis, pois quando esta é aplicada, os problemas podem ser descobertos precocemente, pois as fases são curtas, e pequenas entregas são feitas ao cliente, ao invés de fases longas e entrega do produto somente na versão completa como é praticada por algumas metodologias.

4.2 TESTES DE SOFTWARE

Os testes de software são fundamentais para se adquirir a qualidade do produto final. Sua principal função é diminuir o risco de ocorrem erros, falhas, para garantir a qualidade quando o software estiver em produção com o cliente. Eles são feitos para buscar erros no software, de forma a encontrar e corrigir o maior número de erros possíveis, para aumentar a confiabilidade e qualidade do produto final. É uma tarefa difícil, assegurar uma melhor qualidade corrigindo erros através da aplicação de atividades de teste, pois sempre poderão ser descobertas falhas pelo usuário final, e não existe possibilidade de encontrar todas as falhas do sistema na aplicação dos testes, não existe software com zero defeito. (MOLINARI, 2003)

É uma tarefa de muita responsabilidade também, pois alguns sistemas exigem que certos erros nunca aconteçam. Por exemplo, em um sistema de banco, caso possua uma falha numa determinada área que calcule errado o valor de uma operação, ou um software de um avião que apresente uma informação errada fazendo com que o piloto tome uma decisão equivocada, isso iria gerar enormes problemas ou até tragédias. Então existem alguns erros que podem acontecer, são permissíveis, mas também os que de maneira alguma podem acontecer.

Erroneamente muitas pessoas entendem que os testes são somente os realizados no final do desenvolvimento, onde uma equipe especializada simula vários dados no sistema na busca de erros no código. Estes testes realmente existem e são conhecidos como testes de validação. Mas os testes também devem ocorrer em todas as etapas de desenvolvimento, analisando-se documentos, planos, atividades, e deve haver o comprometimento de toda a equipe na busca do nível adequado de qualidade, para que assim se alcance este objetivo e seja entregue um bom produto ao cliente final, sendo este conhecido como teste de verificação. (INTHURN, 2001)

Apesar de existirem atividades para garantir a qualidade durante o processo de desenvolvimento de software, a etapa de teste é fundamental para identificação e correção de falhas que ainda existirem, principalmente no código do sistema, para assim garantir um melhor produto ao cliente e conseguir atingir sua satisfação.

Segundo Fowler (2004 p.83) “um conjunto de testes é um detector poderoso de falhas que diminui o tempo que se leva para encontrá-las”.

Para Inthurn (2001), os testes devem ser bem planejados para remover o maior número de erros possíveis, e em média 60% destes são removidos. Em relação ao custo de remoção, como já foi visto na figura 5, eles vão aumentando no decorrer do projeto, e assim fica claro que a qualidade bem como os testes de verificação devem ser aplicados durante todo o processo de desenvolvimento, de forma a descobrir os erros o quanto antes e reduzir os custos de reparação.

Os testes são fundamentais no projeto e nos últimos tempos tem-se colocado a atividade como parte do processo de desenvolvimento, não como uma tarefa isolada a fim de buscar somente erros, mas também como parte fundamental para garantir a qualidade do produto, assumindo essa tarefa como um novo projeto,

afinal os testes são fundamentais para fazer a diferença na entrega do software final. (MOLINARI, 2003)

4.2.1 Testes de verificação

Os testes de verificação às vezes são pouco praticados pelas empresas ou quando são praticados, não são feitos da maneira correta. Como foi abordado no item anterior, eles têm papel importante na garantia da qualidade durante todo o processo de desenvolvimento, já que a maioria dos erros são criados no início do desenvolvimento, e quanto antes os erros forem descobertos, menores serão os custos para a solução.

Os testes de verificação devem englobar dois aspectos para que aconteça a garantia da qualidade no decorrer do processo de desenvolvimento: as revisões, que estão focadas nas documentações elaboradas em cada fase do desenvolvimento, e as auditorias, que são responsáveis por avaliar as atividades do processo.

As revisões podem acontecer de diferentes formas. Para Bartié (2002), existem 3 tipos que podem ser aplicadas: a revisão isolada, quando se deseja analisar a documentação na fase inicial de criação, para fazer uma validação parcial do documento, a revisão formal, para ser aplicada no documento finalizado, e reuniões de acompanhamento, para garantir a leitura do documento por todos envolvidos no projeto.

As revisões isoladas são feitas individualmente por um revisor, e possibilitam que sejam descobertos problemas precocemente, pois os documentos analisados estarão em fase de elaboração, permitindo que as pessoas envolvidas no processo de revisão possam buscar estes erros e também possam adquirir maior conhecimento sobre a organização das regras, facilitando para a revisão formal. (BARTIÉ, 2002)

As revisões formais são mais bem estruturadas e realizadas na forma de reuniões, onde estão presentes revisores, que devem ter conhecimento sobre o documento a ser avaliado, e autores dos documentos, para que os erros possam ser debatidos e as dúvidas esclarecidas. Na reunião de revisão são apontadas

mudanças que devem ser realizadas na documentação, sendo que para conclusão do documento, os problemas deverão ser corrigidos para ser feita a versão final do documento. (BARTIÉ, 2002)

As reuniões de acompanhamento, outra forma de revisão, não têm por objetivo principal a detecção de erros, como as outras duas formas apresentadas anteriormente, mas sim a apresentação do documento para as pessoas envolvidas no processo de desenvolvimento a fim de que elas fiquem interadas e possam confirmar o que foi escrito no documento. (MOLINARI, 2003)

Pesquisas apontam que com a realização de revisões, muitos defeitos são descobertos, mas ainda falta um trabalho nas empresas de conscientização e planejamento para a verificação, de forma que as revisões possam ser mais bem organizadas e estruturadas para atingir uma maior detecção de erros precocemente e assim melhorar a qualidade. (BARTIÉ, 2002)

Em relação às auditorias, elas são complementares as revisões e também devem ser aplicadas para conclusão do processo de verificação. O objetivo das auditorias é analisar a fim de garantir que os documentos estão sendo criados, os defeitos ao longo do processo sendo registrados, reuniões de revisão sendo realizadas, enfim, garantir que o processo de desenvolvimento esteja sendo realizado de maneira correta para que os problemas possam ser tratados em cada fase do projeto.

Pode-se perceber de acordo com o que foi apresentado sobre a verificação, que a documentação é muito importante no processo de desenvolvimento de software, pois é a base para esses testes serem realizados. Contudo sabe-se que existem metodologias de desenvolvimento, como as ágeis, que pregam que a codificação é mais importante que documentação. Neste caso, como uma das características destas metodologias é a flexibilidade, a equipe que utiliza as práticas ágeis irá decidir qual documentação é importante para o projeto. Caso isso aconteça, ficará difícil realizar as auditorias e revisões neste tipo de projeto por falta de documentação. Então, para que os testes de verificação possam ser aplicados com as metodologias ágeis, deverá ser estabelecido, por uma equipe especializada em processos, um conjunto mínimo de documentação para os projetos, de forma a garantir que se tenha base para realização dos devidos testes.

Os testes de verificação são muito úteis para a descoberta de erros no início do desenvolvimento do projeto, diminuindo o número de erros que chegará

para os testes de validação. Os dois tipos de testes são fundamentais, pois um complementa o outro de forma a garantir a qualidade do produto e aumentar os resultados positivos.

4.2.2 Testes de validação

Enquanto a verificação está mais voltada para avaliação dos processos, a validação fica encarregada de avaliar o produto, de forma a realizar os testes com o sistema implementado, simulando dados a fim de descobrir problemas no sistema.

A validação é uma confirmação de que o software atendeu aos pedidos do usuário. Existem diversos tipos de testes de validação, que visam identificar e corrigir o maior número de erros possíveis. Nos itens a seguir serão apresentados estes testes.

4.2.2.1 Teste de caixa branca

Os testes estruturais de caixa branca são realizados na parte interna do sistema visando verificar esta estrutura e também a lógica empregada na codificação. É um processo de teste complexo, pois o testador deverá ter conhecimento sobre a aplicação para poder verificar os caminhos críticos dos métodos no código fonte, dados do banco, arquitetura interna do sistema, para poder aplicar as devidas validações. (INTHURN, 2001)

Devido ao profissional ter que possuir estes conhecimentos, essa tarefa pode até ser realizada pelos desenvolvedores, mas isso gerará certa resistência, pois além dos desenvolvedores não aceitarem muito bem a validação, eles terão que trabalhar um pouco mais para realizar estas validações. Outro fator que prejudicaria esta situação é que se o cronograma estiver um pouco estourado, fato que acontece algumas vezes nos projetos de software, os desenvolvedores irão atropelar o teste da caixa branca. (BARTIÉ, 2002)

Então o ideal e recomendado é que este teste seja realizado por testadores especializados ou com conhecimento mais aprofundado sobre estas questões técnicas.

4.2.2.2 Teste de caixa preta

Os testes funcionais de caixa preta têm por objetivo principal verificar se os requisitos levantados pelo cliente foram realmente implementados. São realizados testes para verificar se o sistema retorna os resultados propostos, não se preocupando como o sistema retorna estes resultados, mas sim em descobrir erros de retorno de dados, de interface e de desempenho. (INTHURN, 2001)

Como os aspectos a serem analisados neste tipo de teste são atrelados a fatores externos, o conhecimento técnico sobre a tecnologia do sistema deixa de ser um requisito, e mais testadores estarão capacitados para realização destes testes, necessitando apenas de conhecimento sobre os requisitos funcionais para o sistema.

4.2.2.3 Testes de integração

Os testes de integração, como o nome já diz, são utilizados para a validação dos componentes do software, após ter sido realizado o teste unitário, que será estudado mais adiante, onde os componentes são integrados e é verificado se ainda estão funcionando corretamente.

A integração dos componentes pode ser realizada como um todo, juntando todos ao mesmo tempo para depois realizar os testes, ou de uma forma incremental, adicionando os componentes um a um, para testar partes integradas menores. Pode-se perceber que com a integração em incrementos, ficará mais fácil de descobrir erros, pois serão partes menores a serem verificadas, enquanto que na outra, será a integração completa. (INTHURN, 2001)

4.2.2.4 Testes de recuperação

Os testes de recuperação são utilizados para verificar como o sistema se comporta quando ocorrer uma falha. São feitos testes para forçar o sistema a falhar, e verificar se sua recuperação é automática ou se precisa de uma intervenção do usuário. As falhas que podem ocorrer são relacionadas à memória insuficiente, interface, disco, entre outros. (INTHURN, 2001)

O ideal é que a recuperação seja realizada pelo sistema, ao invés de o usuário precisar intervir e aconteça de forma rápida, para não desagradar o usuário e gerar problemas no seu trabalho, além disso, existem muitos sistemas que não podem ficar parados por muito tempo, então esta validação também é muito importante.

4.2.2.5 Testes de segurança

A segurança da informação é um fator importante e deve ser bem tratada para que não ocorram acessos indevidos ao sistema, ou que se estes acontecerem, que o sistema possua códigos que impeçam que se tenha acesso às informações.

Muitos acessos indevidos tendem a acontecer, dependendo do sistema ao qual se está trabalhando, pode ser alvo de pessoas mal intencionadas, com objetivo de coletar informações importantes da empresa, realizar algum desvio de valores bancários, subtrair senhas, dados pessoais, entre outros.

Estes testes de maneira alguma devem ser realizados pelos desenvolvedores, pois estes sabem os mecanismos de segurança do sistema que foram implementados. A equipe de testes é que deverá realizar, pois desta forma forçará e buscará falhas de segurança para tentar invadir o sistema de todas as formas possíveis.

4.2.2.6 Testes de estresse

Nos testes de estresse são realizadas tarefas que levam a situações anormais no sistema, com carga excessiva de dados para verificar se o sistema continua funcionando. Testa-se os limites do sistema e como ele se comporta com várias janelas abertas, vários acessos ao disco, testes para verificar quanto o sistema pode ser exigido. (BARTIÉ, 2002)

4.2.2.7 Teste de desempenho

A realização do teste de desempenho simula situações de vários acessos e concorrência para avaliar se a performance do sistema está de acordo com o que foi especificado. É analisado o tempo de resposta do sistema a estas situações, sendo que o teste pode ser combinado com o de estresse. (BARTIÉ, 2002)

A equipe de testes deverá conhecer os objetivos de desempenho do sistema para realizar os testes e verificar se as respostas do sistema estão de acordo com o desempenho esperado.

4.2.2.8 Teste de compatibilidade

Problemas de compatibilidade também podem acontecer com os sistemas, devido ao sistema operacional, outras versões do sistema, hardwares, causando mau funcionamento ou o não funcionamento do sistema.

Para isso são realizados os testes de compatibilidade, de forma a garantir que novas versões do sistema possam ser colocadas em produção sem danos de compatibilidade para com a versão anterior, pois esta poderá ainda utilizar, por exemplo, uma interface da versão antiga ou precisar trocar dados com a outra versão. (BARTIÉ, 2002)

4.2.2.9 Teste de usabilidade

O teste de usabilidade visa identificar a facilidade em se utilizar o sistema. São verificadas as características das telas, para ver a padronização dos botões, se estão na mesma ordem nas telas e possuem o mesmo design, cores da tela, mensagens de erro, opções para voltar para tela anterior e desfazer a operação, enfim, opções que facilitem a utilização do sistema.

Uma das operações que também pode ser realizada para analisar a usabilidade seria pedir para um usuário fazer uma determinada tarefa. Os passos que o usuário dará juntamente com o tempo gasto para realizar a tarefa, mostrarão para equipe se o sistema possui uma fácil navegação. Caso o usuário não consiga realizar a operação, existe um problema de usabilidade no sistema.

4.2.2.10 Teste de aceitação

Essa categoria de teste é a última fase de validação para garantir a qualidade do software. Nesta etapa o sistema deve estar pronto para utilização, mas ainda erros podem ser descobertos a tempo para realizar as devidas correções. Se muitos erros forem descobertos, é sinal de que os testes anteriores não foram corretamente aplicados. (BARTIÉ, 2002)

O sistema é disponibilizado para o cliente para que ele possa verificar se atendeu suas expectativas e também o que ele havia pedido. Essa disponibilização para o cliente é realizada de 2 formas: através do teste alpha e do teste beta.

O teste alpha é realizado num ambiente dentro da empresa que desenvolveu o software, onde alguns dos usuários do sistema simulam tarefas normais de trabalho acessando todas as partes do sistema, para que os responsáveis pela avaliação possam identificar os problemas encontrados através de situações reais dos usuários.

No teste beta, o sistema é colocado no ambiente real da empresa, para que uma maior parte dos usuários do sistema possam utilizar e verificar se o sistema está atendendo as necessidades da empresa. Ainda não é uma implantação definitiva do sistema, pois haverá um acompanhamento da equipe desenvolvedora até que os usuários tenham confiança em utilizar o novo sistema.

Quando houver a aceitação efetiva do sistema através do teste beta, a implantação do sistema é iniciada na empresa.

5 TESTES UNITÁRIOS

Os testes unitários, também conhecidos como testes de unidade, visam testar cada classe crítica do sistema com a finalidade de buscar erros, assim como os outros testes. É a primeira etapa dos testes de validação, e avaliam a estrutura interna do sistema, podendo assim ser considerado um teste de caixa branca.

O objetivo deste teste é percorrer o código de cada classe do sistema e verificar os desvios condicionais, caminhos alternativos de execução, e se ele responde conforme seus requisitos. Além disso, os testes unitários devem avaliar os requisitos funcionais, de usabilidade e de sistema relacionado ao componente. (BARTIÉ, 2002)

Segundo Massol (et al., 2005 p. 6) “uma descrição genérica de um teste de unidade típica: confirme que o método aceita a faixa de entradas esperada, e que o método retorne ao valor esperado para cada entrada de teste”.

A aplicação dos testes unitários não precisa ser feita em todas as classes. Mesmo porque existem classes simples que não necessitam ser testadas, seria perda de tempo testar uma classe que se tem certeza que não há possibilidade de ocorrer um erro, e assim o teste não iria agregar valor algum. Então o que deve ser testado são as classes consideradas críticas, que podem apresentar algum tipo de falha, e na dúvida, aplicar os testes. (BECK, 2004a)

> Um teste de unidade examina o comportamento de uma unidade de trabalho distinta. Dentro de um aplicativo Java, a “unidade de trabalho distinta” é frequentemente (porém, nem sempre), um método único. Por contraste, testes de integração e teste de aceitação examinam como os vários componentes estão integrados. Uma unidade de trabalho é uma tarefa que não é diretamente dependente da finalização de qualquer outra tarefa (MASSOL et al., 2005, p. 6).

Apesar dos testes unitários estarem focados internamente nas classes, eles podem ser realizados de diferentes formas e seguindo o modelo de desenvolvimento aplicado no projeto.

Quando os testes unitários são realizados na forma de teste de caixa branca, estes ocorrem com cada classe do sistema, avaliando o retorno de seus métodos e comparando com o retorno esperado de forma a testar todo o código fonte, até que as falhas não sejam mais encontradas.

Para que essa forma aconteça, o profissional deverá possuir avançado conhecimento técnico em programação, além de conhecer toda a estrutura do sistema, e sendo assim, o ideal é que o próprio programador realize este teste no seu código. Levando em consideração que muitas vezes os programadores estão trabalhando sob pressão com o cronograma extrapolado, esta tarefa não será considerada prioritária, então a sugestão é que a empresa aumente a força de trabalho desta área para que estes testes possam acontecer, pois como já foi visto, os custos de manutenção ficam muito altos quando são descobertos tardiamente.

Utilizando os testes de caixa preta, a validação dos componentes estará focada em identificar quais os requisitos de cada componente, para aplicar um teste de validação conforme os requisitos indicarem. Por exemplo, numa classe que fará uma transação com o banco de dados, poderá ser realizado um teste de desempenho para verificar o tempo de duração desta transação.

Em modelos de desenvolvimento estruturados, onde o desenvolvimento dos componentes acontece de forma hierárquica, do nível superior para o inferior, os testes unitários acontecem da mesma forma. Desta maneira, é necessário a construção de simuladores pois somente os componentes superiores ao que serão testados estarão disponíveis, os inferiores não, fazendo com que os testes tornem-se demorados e trabalhosos. (BARTIÉ, 2002)

Nos modelos orientados a objetos, a validação com testes unitários acontece de maneira mais eficaz, pois o desenvolvimento da aplicação acontece de forma crescente, e assim, cada classe do sistema é testada e depois o conjunto destas também, para verificar se todos os requisitos foram atendidos e os métodos devidamente implementados. Como os testes são realizados de acordo com o desenvolvimento, neste caso começando do nível inferior, todas as classes poderão ser testadas de forma contínua, potencializando a descoberta de erros.

Algumas pessoas pensam que os testes unitários tomam muito tempo dos desenvolvedores e assim aumentam os custos, já que mais serviço é agregado aos profissionais e conseqüentemente mais tempo será tomado deles. Um fator que pode minimizar este pensamento é relacionar o custo que é gasto quando um problema é encontrado meses depois no sistema e o desenvolvedor tem que focar seu trabalho na descoberta deste erro. Primeiro terá que lembrar ou entender a lógica aplicada para a funcionalidade, depois buscar em que classe do código está o problema, para depois pensar em uma solução. É claro que os testes unitários

geram um aumento de custo e tempo no desenvolvimento, um custo para disponibilizar mais profissionais ao setor, mas que serão menores e gerarão um retorno mais positivo e rápido do que utilizar todos os testes somente no final do desenvolvimento, mostrando uma das vantagens se sua utilização. (TELES, 2006)

É real que desenvolver e ter que manter um conjunto significativo de testes de unidade custa tempo e aumento de custos. Mas pode-se analisar por outro lado esta situação, considerando isto como um investimento e não como despesa, de forma que a cada nova funcionalidade incrementada e um novo conjunto de testes unitários rodar apresentando execuções bem sucedidas, aumentará a confiança da equipe e este começa a ser o retorno do investimento. (MARINESCU, 2002)

Sabe-se também que todo investimento possui um risco, que é o de não receber o retorno esperado. O investimento aplicado nos testes de unidade deve ser controlado, e para minimizar o risco, pode-se definir a quantidade de testes necessários a ser construído, quem deve fazê-los, e também definir estratégias para mantê-los organizados para permitir que sejam fáceis de entender e manter.

Como já foi tratado no início do capítulo, não se deve testar tudo, somente as classes e métodos mais críticos, como afirma Jeffries (et al. 2000 apud Marinescu 2002 p. 145) “teste tudo o que pode dar errado”, não interpretando esta afirmação literalmente, pois desta forma haverá desenvolvedores implementando testes de unidade para todos os tipos de métodos, gerando um enorme conjunto nada agradável.

Não há uma convenção ou uma regra que diga o quanto testar. Marinescu (2002) diz que a empresa deve concentrar seus esforços nos limites das camadas do software, concentrando os testes unitários nas camadas de serviço e aplicação, mas não negligenciando outras camadas, persistência e domínio.

Para Fowler (2004) o testes devem ser orientado pelo risco, não devendo testar cada método público que desenvolver, afinal a idéia dos testes unitários é encontrar falhas tanto no presente quanto no futuro, e então não deve-se testar métodos de acesso para leitura e gravação, por exemplo, pois dificilmente algum erro será encontrado, por serem tão simples. Então se deve concentrar os esforços onde há realmente risco e parar quando houver uma boa bateria detectora de erros.

5.1 BENEFÍCIOS DOS TESTES UNITÁRIOS

Um dos principais objetivos dos testes unitários é gerar código simples, claro, de fácil manutenção e que funcione corretamente. As vantagens em utilizá-los não está somente no fato deles estarem validando constantemente o projeto, mas também em auxiliar a análise e o design, e facilitar a codificação do sistema (BECK, 2004a).

Segundo Marinescu (2002 p.145) “o testes de unidade é uma daquelas coisas que todos sabemos que devemos fazer, mas tendemos a deixar para trás na corrida por faturamento”.

Dessa forma, esta pressa em entregar logo o software irá acabar gerando problemas, e a falta de um bom conjunto de testes unitários, faz com que a empresa corra este risco e fique exposta a contrair os mesmos.

Os testes unitários nos projetos de software permitem uma maior cobertura de testes, se comparado com testes funcionais, que cobrem em torno de 70% do código do projeto. Eles permitem simular condições de erros muito facilmente, o que não é tão simples assim com os testes funcionais (MASSOL et al., 2005)

Como foi citado anteriormente no item 5, os testes de unidade podem ser aplicados como testes de caixa preta ou branca. Cabe ressaltar que quando aplicados na forma de caixa preta, estes não cobrem totalmente o código, já que não são aplicados verificando o código completo. Para uma maior cobertura, os testes de caixa branca são indicados, pois desta forma eles são escritos analisando-se cada método, cada detalhe nas classes, para consequentemente obter melhor qualidade.

O trabalho em equipe favorece muito o desempenho dos trabalhos e desenvolvimento dos projetos numa empresa. Uma equipe organizada, cooperativa e motivada, produz mais e com melhor qualidade. Com os testes unitários, cada classe do sistema pode ser testada separadamente, e como o código é trabalhado por toda equipe, é muito motivador receber um código já testado que está em perfeito funcionamento, diferente da utilização dos testes funcionais, que necessitam do sistema completo, ou grande parte dele, para sua aplicação (MASSOL et al., 2005).

Outro benefício que o teste de unidade proporciona está relacionado com o tempo gasto com depuração. Quando o desenvolvedor está munido com um conjunto significativo de testes unitários, as falhas são descobertas precocemente, garantindo que o tempo perdido com depuração, seja aproveitado para desenvolver outras atividades.

Os testes de unidade aumentam a produtividade do desenvolvedor, além de gerar um código já testado para a equipe de garantia de qualidade, que receberá o software já com uma boa qualidade, tornando o trabalho mais simples e muito mais motivador.

Uma das características das metodologias ágeis é desenvolver somente o necessário para atender uma determinada funcionalidade, ou seja, defendem a refatoração, é muito melhor desenvolver algo que certamente será usado, do que incrementar algum funcionalidade que poderá nunca ser utilizada, e com testes unitários esta forma de desenvolvimento fica mais protegida, pois o desenvolvedor contará com a segurança e confiança fornecida pela aplicação destes testes.

Os testes unitários também podem ser aplicados para gerar documentação para o projeto. Analisando-se a forma com que eles são aplicados e precisam ser atualizados de acordo com o desenvolvimento do projeto de um sistema, eles podem ser utilizados para o entendimento de suas funcionalidades. Suponha-se que exista uma grande documentação deste sistema, com vários diagramas, descrição de dados, manuais, entre outros, e um novo desenvolvedor deseja entender com o sistema funciona. Será muito mais rápido para ele analisar os testes do que passar horas lendo documentação para depois verificar o código.

5.2 TESTES UNITÁRIOS DURANTE O PROCESSO DE DESENVOLVIMENTO

Após os testes unitários serem apresentados e também seus benefícios, neste item será mostrado que apesar de suas enormes viabilidades, durante o processo de desenvolvimento de software eles ainda necessitam ser complementados com outros testes. É claro que sua aplicação já alavancará a qualidade do produto, mas os outros testes apresentados no item 4.2 e suas subseções também têm importância durando o ciclo de desenvolvimento.

Os testes unitários podem ser aplicados durante todo o desenvolvimento, enquanto que a maioria dos demais testes são aplicados no final do ciclo. Com isso, os testes unitários potencializam a qualidade do produto, e no final do desenvolvimento, alguns testes que podem ser aplicados, por exemplo, são o de estresse e de desempenho, apresentados nos itens 4.2.2.6 e 4.2.2.7, para garantir maior qualidade no software.

Os testes de unidade podem ser divididos em três modalidades: os lógicos, que tratam da lógica de negócio e que são realizados isoladamente; os de integração, que se focam na integração e interação entre as classes, provando por exemplo, em uma interação com um banco de dados, que tudo está funcionando; e os funcionais, que não são testes de unidades puros, pois estão envolvidos também com o ambiente externo, testando um fluxo de trabalho. Todos estes três são úteis, e um complementa o outro e ainda complementam os demais testes que devem ser aplicados pela equipe de testes.

5.3 O DESENVOLVIMENTO GUIADO PELOS TESTES (TDD)

Como pôde ser visto nos itens anteriores, a maioria dos testes são realizados no final do projeto, o que faz que estes possam ser atropelados quando o projeto está atrasado.

Muitos detalhes precisam ser testados no sistema, e por isso, existem diversos tipos de testes, o que causa mais um motivo para que estes não sejam todos realizados e gerem problemas de qualidade no produto final.

Apesar do pessoal do desenvolvimento conhecer os benefícios dos testes, esta atividade costuma ser considerada chata e às vezes não muito importante, e por isso, quando é realizada acontece no final do desenvolvimento. Para que eles ganhem a importância devida, devem ser aplicados de forma simples, sem muito consumo de tempo, de forma que os desenvolvedores passem a gostar da atividade, tornando-a natural à tarefa de codificação (TELES, 2006).

O TDD utiliza os testes unitários, que são realizados pelos próprios desenvolvedores, antes da implementação das funcionalidades do sistema, de forma

a garantir maior qualidade no produto final e para que a atividade torne-se uma tarefa tão importante que passe a fazer parte da codificação.

Segundo Beck (2004a) quando você programa com testes o trabalho fica mais divertido, e você ganha mais confiança, além de que programar e testar ao mesmo tempo é mais rápido do que só programar. A lógica para esta afirmação está no fato de que quando a codificação é realizada sem os testes, um ganho de produtividade inicial poderá ser obtido, mas quando a equipe se acostuma a testar, a produtividade aumentará se considerar a redução de tempo na depuração.

Sabe-se que os custos de manutenção do software aumentam relativamente de acordo com o andamento do projeto, acarretando mais tempo e mais trabalho para descobrir o erro conforme o projeto vai sendo desenvolvido. Com a aplicação dos testes unitários os erros são descobertos rapidamente, após cada execução dos testes e em cada classe aplicada, facilitando também a correção destes erros com mais facilidade e rapidez.

> O desenvolvimento orientado por testes é uma prática de programação que ensina os desenvolvedores a escreverem novos códigos, somente se um teste automatizado falhar, e a eliminar a duplicação. A meta do TDD é “código limpo que funciona” (BECK 2003 apud MASSOL et al., 2005 p. 88).

Para que seja melhor compreendido e entendida a sua importância e funcionalidade, toma-se como exemplo uma metáfora, comparando-o com uma doença na vida humana. Existe um ditado popular que diz que é melhor prevenir do que remediar. Quando uma pessoa está com uma doença e não se cuida, não procura um médico e não toma medicação, seu estado poderá se agravar, e conseqüentemente os custos para recuperação serão mais altos. (TELES, 2006)

Quando um problema no software é encontrado algum tempo depois da funcionalidade ter sido desenvolvida, muito tempo e trabalho serão gastos para encontrar a falha, aumentando os custos para manutenção, fato este que seria evitado caso o problema fosse descoberto em pouco tempo, através da aplicação dos testes unitários do desenvolvimento guiado pelos testes

Ao aplicar o desenvolvimento guiado por testes é como prevenir uma doença. Problemas podem surgir, mas com menor freqüência e mais simples de serem solucionados.

Para Teles (2006 p. 107), “a idéia principal da programação guiada pelos testes é fazer com que o tempo total dedicado a depuração em um projeto seja reduzido ao mínimo possível”.

O TDD pode ser considerado uma sequência evolucionária da utilização dos testes unitários. Os desenvolvedores começam fazendo testes para seus métodos e classes. Com isso, os testes o ajudarão a ir melhorando o projeto inicial, e o encorajando a cada vez escrever mais testes e mais cedo, a fim de melhorar cada vez mais o projeto. Com o tempo, o desenvolvedor vai percebendo que à medida que desenvolve seu projeto, já fica imaginando como fará os testes. Então, seguindo esta linha, cada vez mais desenvolvedores estão dando um salto do desenvolvimento amigável dos testes para o desenvolvimento guiado por testes, onde estes primeiramente são escritos, para depois a codificação das classes e implementação dos métodos (MASSOL et al., 2005).

Normalmente, o desenvolvimento de sistemas começa com um planejamento, depois parte para codificação, testes, caso haja erros estes são corrigidos, e após aprovação do sistema. No desenvolvimento tradicional geralmente é isto que ocorre: codificar, testar, corrigir e aprovar.

A aplicação do TDD faz com que esta sequência seja um pouco ajustada, porém prega que esta forma gera maior qualidade e traz mais eficiência para o desenvolvimento: testar, codificar, corrigir e aprovar. Nesta forma, o teste irá orientar o projeto. Ele virá primeiro que a codificação e fará com que o desenvolvedor pense melhor para poder produzir um código mais limpo, mais coeso e de melhor qualidade.

Quando se programa através do TDD, a implementação dos testes de unidade permite que o teste seja executado para provar que a implementação funciona. Caso o teste falhe, deve-se trabalhar na implementação até que este passe e prove que o código funciona como foi solicitado (MASSOL et al., 2005).

Para Martin (et al. 2003 apud AMBLER 2010) o TDD visa à especificação e não a validação. É uma maneira de pensar sobre o projeto antes de escrever o código funcional.

Para exemplificar o funcionamento do TDD e para que este seja melhor compreendido, no próximo item este assunto será tratado mostrando também um exemplo de sua utilização.

5.3.1 Funcionamento do TDD

A figura 6 mostra o funcionamento do TDD através de um diagrama de atividades da UML.

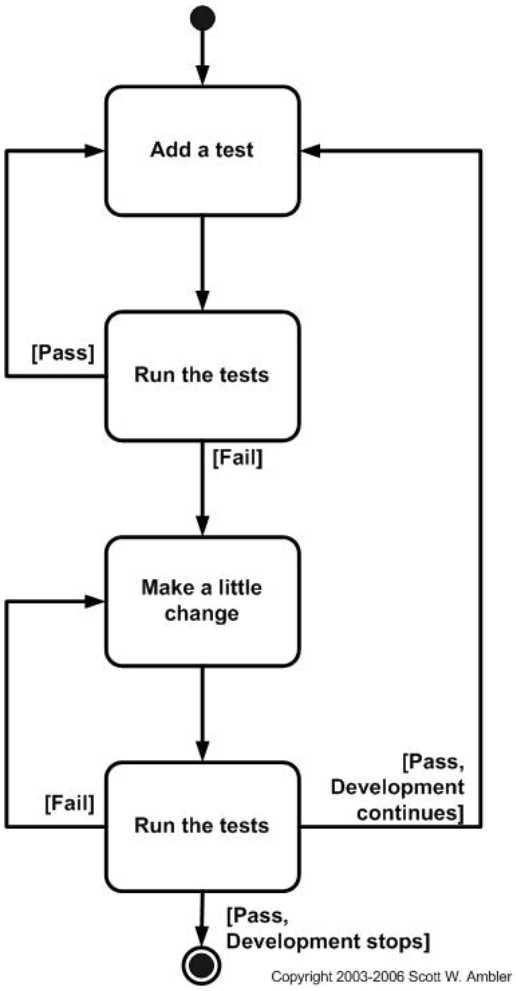

Figura: 6: Sequência de atividades do TDD
Fonte: AMBLER, 2010

Como já foi visto no item anterior, 5.2 sobre o desenvolvimento guiado pelos testes, nesta abordagem de aplicação de testes unitários, o primeiro passo é escrever os testes, e para isso, deve-se saber quais as respostas esperadas para poder aplicar os testes. Dessa forma, o desenvolvedor começa a entender o que precisa ser desenvolvido, e assim poderá validar precocemente seu código. Este teste deve ser simples, para que uma falha seja capturada, para garantir que ele possa capturar um erro. Neste ponto ainda não é necessário que a classe que contenha o método a ser testado exista, mas após o teste estar pronto, deve-se desenvolver somente o código necessário para passar na compilação e falhar após

execução.

Este é um ponto importante nos testes unitários através do TDD: antes de implementar qualquer código, deve-se implementar primeiro um teste de falha, já que o código para que ele seja bem sucedido ainda não foi escrito. Então, o que acontece normalmente é a primeira codificação ser bem simples, para fazer o teste passar, e assim, ir refatorando esta codificação para abordar por completo uma funcionalidade e desta forma o primeiro teste irá falhar (MASSOL et al., 2005).

Após rodar o teste e este falhar, deve-se ir aprimorando o teste aos poucos, fazendo pequenas mudanças até ele rodar e passar sem falhar, pois desta forma o teste irá garantir que está funcionando, capturando falhas e passando quando estas não existirem.

O próximo passo é melhorar os testes e permitir que eles consigam abordar as mais variadas situações de falha, e ir aprimorando o código real até que a funcionalidade seja toda abordada e todos os testes passem.

5.4 O FRAMEWORK JUNIT

Segundo Johnson (1988, apud MASSOL et al. 2005) um framework é uma aplicação reutilizável e semicompleta que pode ser especializada para produzir aplicações personalizadas.

O framework JUnit é o padrão para testes de unidade em Java. O Junit é simples, porém muito eficiente, e foi desenvolvido em 1997 por Eric Gamma e Kent Beck, devido à necessidade da automação de testes em Java, sendo um software de fonte aberta que pode ser distribuído livremente. (MASSOL, 2005)

Ele fornece uma interface gráfica e permite que você possa acompanhar facilmente o progresso dos testes, informando o tempo gasto na execução e ainda, o mais interessante, sinais de positivo e negativo simples para que o desenvolvedor possa se orientar na execução dos testes. Pode ser realizada uma série de testes ao mesmo tempo, e se a barra verde aparecer está tudo certo, os testes passaram, mas caso a barra vermelha aparece, algo está errado e deve ser corrigido.

Para Neto (2008 apud BIANCHINI 2009 p.26) este framework facilita a criação de código para automação de testes unitários, possibilitando a verificação de

cada método, para garantir que estes funcionem de forma esperada, podendo ser utilizado em bateria de testes ou extensão. O autor ainda cita algumas vantagens de utilização da ferramenta:

- Criação rápida de código de teste;
- Amplamente utilizado pela comunidade de código aberto;
- Uma vez escritos, os testes podem ser executados sem interromper o processo de desenvolvimento;
- Este framework verifica o resultado dos testes com resposta imediata;
- É livre e orientado a objetos;
- Faz com que o desenvolvedor perca menos tempo na depuração do código.

Segundo Beck (2004b), entre suas várias funcionalidades o Junit:

- Executa testes automaticamente;
- Executa vários testes simultaneamente e resume os resultados;
- Compara os resultados reais com os esperados e relata as diferenças.

A automação de teste com JUnit não deve ser considerada um trabalho extra pelo desenvolvedor, pois ele fornece mais segurança e confiança, e assim ele pode garantir que o sistema está funcionando perfeitamente.

Para Beck (2004b), os testes automatizados com JUNIT deixam o desenvolvedor mais confiante, mais seguro no seu trabalho, e quando não são realizados, causam a sensação de que algo está faltando, que foi esquecido de realizar algum teste, que algo pode ter ficado com falha, gerando uma sensação de insegurança. Sem contar que quando uma manutenção precisa ser realizada, um tempo depois o desenvolvedor acaba esquecendo-se do comportamento esperado do sistema, e caso precise consertar algo, muito tempo será gasto até encontrar o ponto correto para corrigir o problema. Se o sistema foi desenvolvido com o auxílio do JUnit, é só rodar os testes e ver se tudo está funcionando corretamente, ou em caso de falha, focar no ponto exato para realizar a devida correção.

Segundo Beck (2004b) todos os programadores testam seus códigos, e realizar os testes com JUnit não é uma atividade totalmente diferente do que já é realizado, é apenas uma maneira diferente de fazer o que já está sendo feito. A

diferença entre os testes é que com Junit pode-se ir testando se o programa está se comportando como esperado, e tendo uma bateria de testes, eles garantem que o programa está se comportando da maneira correta.

5.4.1 Arquitetura do JUnit

O Junit está organizado em pacotes, possui em torno de 75 classes, mais as internas e interfaces. A figura 7 representa a organização dos pacotes do JUNIT. (RAINSBERGER, 2004)

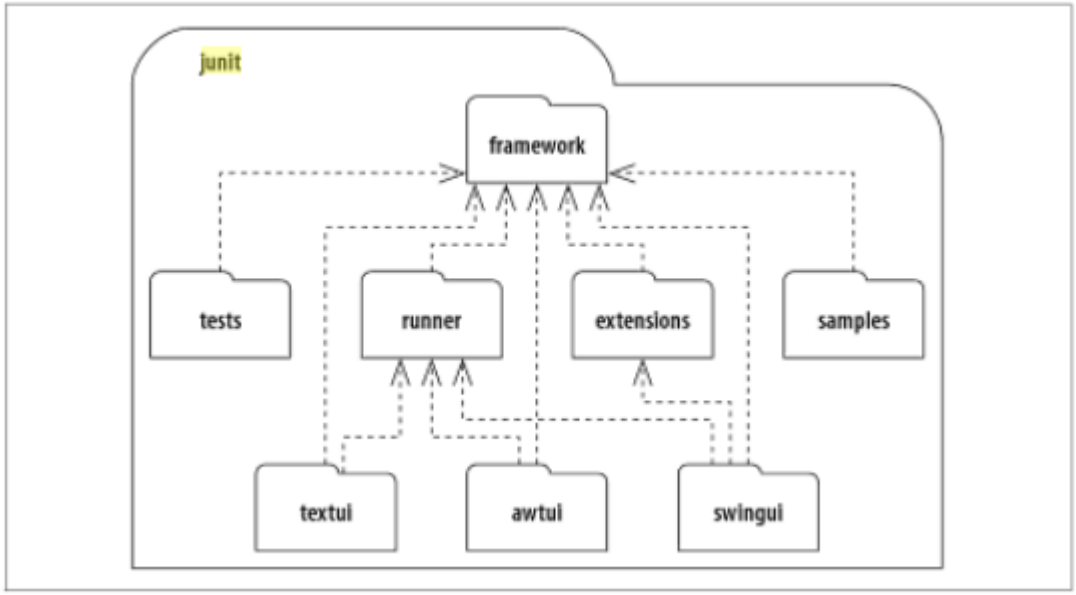

Figura 7: Os pacotes do Junit e suas dependências
Fonte: HAMILL, 2004

O pacote framework representa a principal funcionalidade, e a base na qual os testes unitários são desenvolvidos. Os pacotes de interface com usuário, awt, swing e text, são relativamente complexos. O samples, possui exemplos de testes unitários e o tests, possui testes unitários do próprio JUnit. (HAMILL, 2004)

Como é o pacote mais importante e funcional do Junit, o framework será detalhado através de suas classes na figura 8.

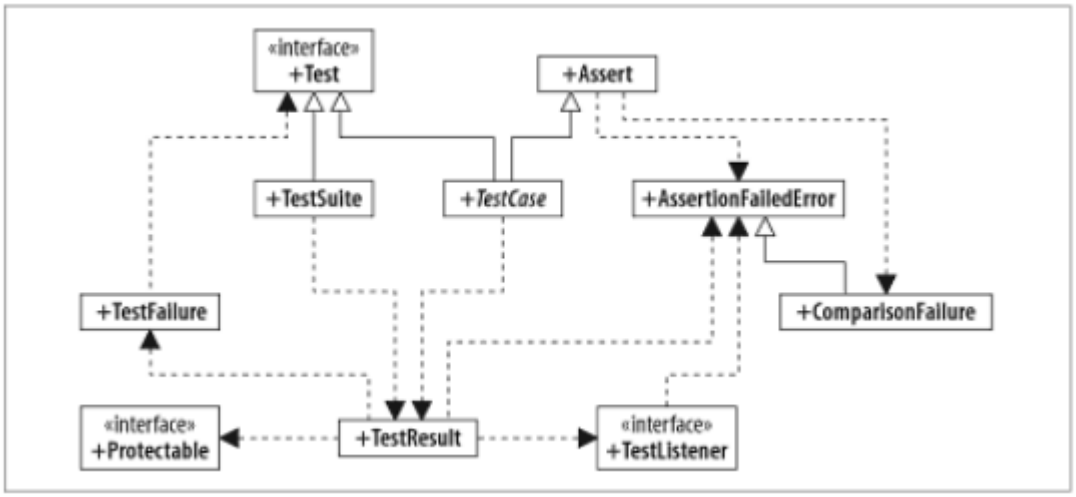

Figura 8 – Arquitetura de classes do pacote framework
Fonte: Hamill 2004

A interface Test, possui um método chamado runTest, reponsável pela execução de testes particulares.

Como pode ser observado na figura 8, as classes TestSuite e TestCase são filhas de Test, logo herdam desta. A primeira é utilizada para executar um teste em vários métodos, registrando os resultados na TestResult, e a segunda, testa os resultados de um método. A TestCase, que é a classe pai de todas as classes de testes unitários, ainda possui os métodos setUp, que indica o início do teste e é chamado antes de cada método, e o tearDown que faz o contrário, sinaliza o fim do teste e é chamado após cada método, desfazendo o que o setUp fez. (JUNIT, 2012)

Muitos casos de teste no JUnit são derivados indiretamente da classe Assert, que contém métodos para automaticamente checar valores e relatar diferenças.

Antigamente utilizava-se várias instruções *System.out.print()* para depurar o código em busca de erros. Logo, caso o software estivesse funcionando como esperado, todas estas instruções deviam ser removidas, gerando um processo demorado e propenso a erros.

Assert, que pode ser entendido com asserção, é um tipo Java que pode ser considerado uma instrução System.out.print() reforçada. A partir da versão 5 do Java, este tipo vem ativado por padrão, e caso não seja feito nada, estas instruções são ignoradas pelo Java, não ocasionarão problema algum. Mas caso sejam ativadas, elas ajudarão na depuração sem alterar uma linha de código sequer.

Existem diversos métodos para realizar a asserção, e podem ser feitos de 2 maneiras: com ou sem uma mensagem. A utilização do parâmetro String que representa a mensagem é opcional e pode ser utilizado quando é necessário que uma mensagem informando a falha seja lançada. A tabela 2 mostra algumas formas de realizar uma asserção, ou seja, de executar um teste e verificar o resultado esperado. (RAINSBERGER, 2004)

Assertion	Significado
void assertTrue(boolean)	Relata um erro caso o resultado seja falso.
void assertTrue(String, boolean)	Relata um erro representado pela String se o resultado for falso.
void assertEquals(double, double)	Relata um erro caso os 2 números decimais não sejam iguais.
void assertNull(Object)	Relata um erro caso o objeto não seja nulo.
void assertNotNull(Object)	Relata um erro caso o objeto seja nulo.

Tabela 2 – Assertions
Fonte: Elaborado pelo Autor

Todas asserções falhas invocam o método fail() que lança uma AssertionFailedError. (Beck, 2004b)

Para criação da classe de teste, existe uma convenção do JUnit, onde o nome da classe a ser testada deve ser complementada com o nome Test, como por exemplo, RendaTest.java, onde Renda é a classe que será aplicado os testes unitários neste trabalho. Para os métodos, pode ficar testCalcularRenda(), informando que esse é um método de teste. Estas convenções são consideradas boas práticas em testes. (HAMILL, 2004)

Para compreensão da funcionalidade do JUnit e suas asserções, pode-se exemplificar da seguinte maneira: suponha-se que exista uma classe denominada FuncoesMatematicas e nesta classe um método chamado somar(double, double) que recebe dois números para realizar a soma.

É preciso que exista uma classe de teste, que seguindo a convenção deve chamar-se FuncoesMatematicasTest, que conterá o método testSomar(). Neste método, existirá as asserções, como por exemplo, assertEquals(10.0, funcoesMatematicas.somar(5.0,5.0)), testando que se forem somados 5.0 e 5.0 o resultado tem que ser igual a 10.0. Se for executado desta maneira, será visto que o teste passou, o resultado foi positivo, como mostra a figura 9.

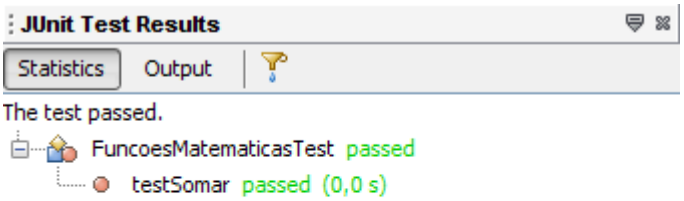

Figura 9: Teste OK no JUnitFonte: Elaborado pelo autor

Outro teste que pode ser realizado é um teste para capturar uma falha, escrevendo a asserção da seguinte maneira: assertEquals(10.0, funcoesMatematicas.somar(5.0,6.0)), e o JUnit apresentará a falha de acordo com a figura 10.

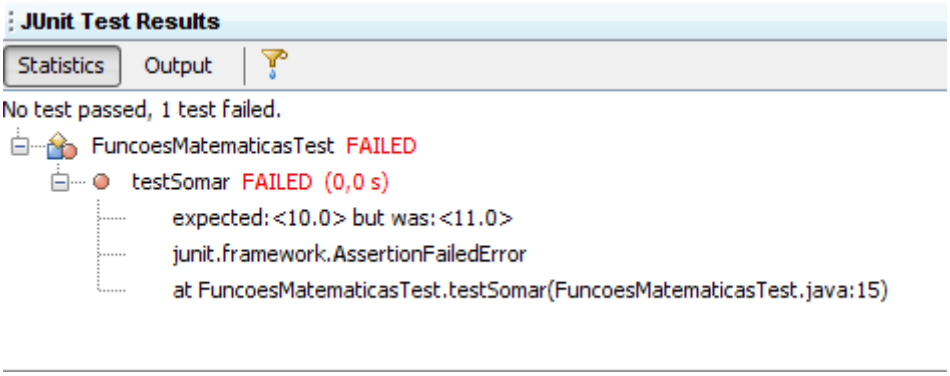

Figura 10: Falha no teste do JUnit
Fonte: Elaborado pelo autor

O resultado do teste de falha mostra que o método fail() foi invocado, lançando uma AssertionFailedError, como já abordado anteriormente neste item. Pode-se perceber também que o JUnit relatou o que está errado, era esperado 10.0 na asserção, mas o resultado foi 11.0.

Após a explicação sobre o JUnit e seu funcionamento, exemplificando um teste positivo e um teste com falha, no próximo capítulo será aplicado os testes unitários aplicados com o JUnit e utilizando a metodologia do TDD

6 TESTES UNITÁRIOS NA QUALIDADE DE SOFTWARE

Após mostradas as características e benefícios dos testes unitários, sua aplicação através do TDD e também o framework destes testes, neste item será feito a aplicação destes conceitos em alguns códigos utilizados pelo autor no seu Trabalho de Conclusão de Curso.

Para melhor entendimento sobre o que os códigos representam , será explicado do que se tratava o trabalho intitulado “Sistema Web para DAP - WEBDAP”.

> O sistema engloba o cadastro e envio das DAP[4] (como o cadastro de agricultores e disponibilização de algumas consultas, como a dos agricultores) utilizando um banco de dados relacional MySQL e JSP para o desenvolvimento do conteúdo dinâmico da Web, utilizando a arquitetura MVC, aplicando- se ao estado de Santa Catarina. (SANTOS, et al., 2007, p.19)

Os problemas nos quais o trabalho foi baseado estavam relacionados ao processo de envio das informações das DAP existentes, que estavam causando muita perda de dados, visto que os sistemas não realizavam uma consistência nos dados cadastrados enviados para o MDA[5] em Brasília, ou seja, não realizavam uma validação desses dados antes de enviá-los. (SANTOS, et al., 2007)

Os sistemas estavam um pouco defasados, e atrelados a eles, o processo de envio das informações desde os órgãos credenciados até o MDA estava inconsistente. (SANTOS, et al., 2007)

Os dados eram enviados por e-mail, até uma central onde eram importados para um banco de dados Oracle e após convertidos para XML para transmissão para Brasília. (SANTOS, et al., 2007)

A solução proposta para estes problemas abordou o desenvolvimento de um sistema Web com a tecnologia JEE, plataforma Java padrão para o desenvolvimento Web aplicando o MVC com a utilização do Web Framework Struts. (SANTOS, et al., 2007)

Na solução, o cadastramento do agricultor familiar é feito no sistema Web, onde o usuário realiza o cadastro do agricultor, e após salvar os dados, estes são

[4] Declaração de Aptidão ao Pronaf
[5] Ministério do Desenvolvimento Agrário

armazenados em uma base local do sistema gerenciador de banco de dados MySQL e, posteriormente, enviados diretamente para a base do MDA em Brasília. (SANTOS, et al., 2007)A figura 11 apresenta a arquitetura lógica da solução, onde a DAP é gerada nos escritórios municipais, após a coleta e cadastramento dos dados dos agricultores, em seguida é armazenada em uma base local MySQL, e é enviada através do sistema Web direto para o MDA em Brasília através de um arquivo XML que é validado pelo sistema para ser armazenado no banco de dados do MDA. (SANTOS, et al., 2007)

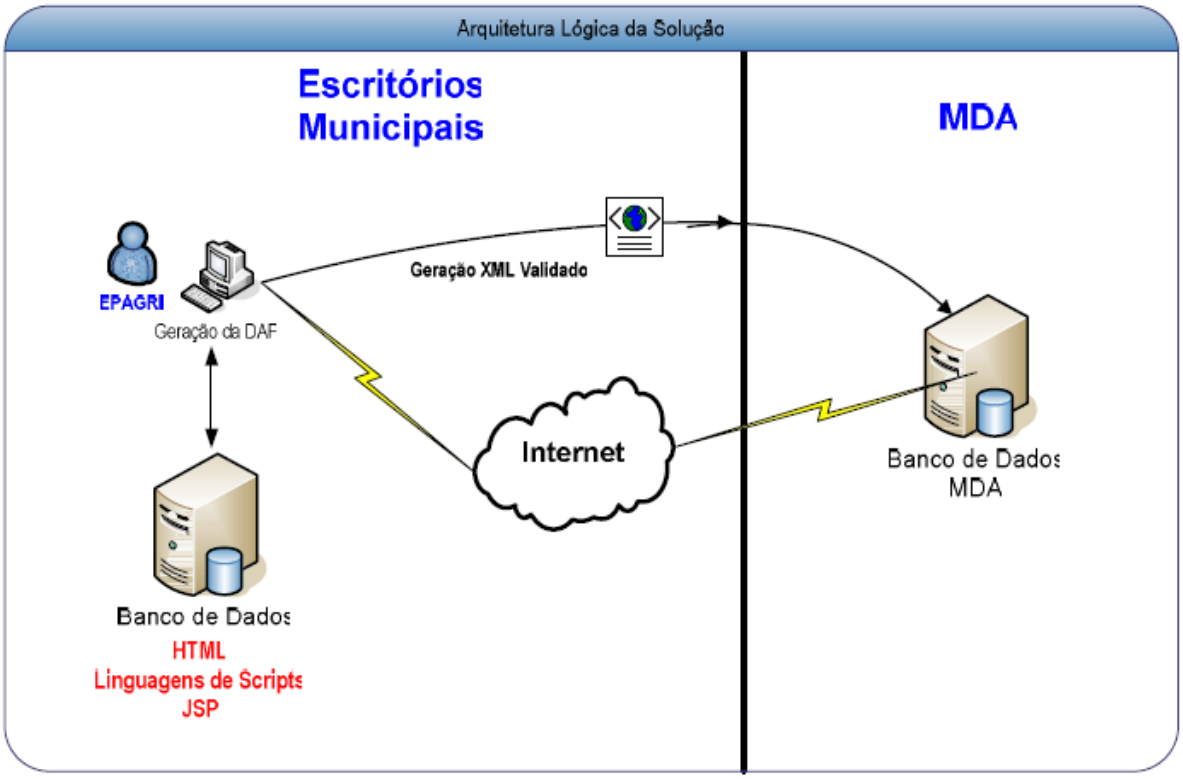

Figura 11: Arquitetura Lógica da solução WEBDAP
Fonte: Santos, et al., 2007, p.84

Para compreender o que foi realizado no trabalho, estes eram seus objetivos: (SANTOS, et al., 2007, p.18)

- Utilizar o processo unificado e a análise orientada a objetos para a modelagem e desenvolvimento da solução;
- Realizar a reengenharia dos dados através do projeto de banco de dados;
- Eliminar a perda de informações provocadas pelos sistemas, e modificar a forma de transmissão das informações;
- Aplicar a arquitetura MVC para o desenvolvimento da solução.

Para a composição da renda do agricultor na DAP, algumas regras existem, e devido ao tempo que o trabalho de conclusão foi desenvolvido, ano de 2007, novas regras surgiram, e por isso será apresentada uma nova classe para composição da renda do agricultor utilizando o que foi apresentado no capítulo 5: testes unitários, TDD e o JUnit.

Segundo Santos (et al., 2007 p.23), “para poder ser enquadrado no PRONAF, o produtor não poderá ter renda bruta anual superior a R$ 110.000,00, que é a renda máxima permitida”.

A regra para enquadramento no Pronaf, diz que as rendas do agricultor devem ser rebatidas, ou seja, descontada para determinadas culturas, de acordo com a tabela 3.

Cultura	**Porcentagem de Rebate**
Açafrão, algodão, amendoim, arroz, aveia, cana-de-açúcar, centeio, cevada, feijão, fumo, girassol, grão de bico, mamona, mandioca, milho, soja, sorgo, trigo, triticale, apicultura, aqüicultura, bovinocultura de corte, cafeicultura, fruticultura, ovinocaprinocultura e sericicultura.	50%
Turismo rural e agroindústria familiares	70%
Avicultura e suinocultura integrada	90%

Tabela 3: Culturas com rebate no Pronaf
Fonte: RICK, 2010

Para entendimento da tabela apresentada, suponha-se que um agricultor tenha renda de R$ 20.000,00 anual com a cultura de fumo, predominante na região de Araranguá. De acordo com a tabela, a cultura de fumo tem 50% de rebate, isto quer dizer que sua renda auferida é de R$ 10.000,00. Isto proporciona que agricultores de maior renda, como por exemplo, os que trabalham com arroz, cultura também predominante na região, possam receber a DAP e obter seus benefícios. Para um produtor de arroz, cuja renda normalmente passa de R$ 110.000,00, e desta forma ficaria impossibilitado de obter uma DAP, agora com as novas regras descritas por Rick (2010), já pode ser beneficiário.

Para melhor organização das culturas, elas serão agrupadas em categorias, conforme mostra a tabela 4.

Porcentagem de Rebate	Categoria
50%	1
70%	2
90%	3

Tabela 4: Categoria das culturas
Fonte: Elaborado pelo Autor

Após a explicação sobre o trabalho e seu conteúdo, e também sobre as novas regras que surgiram, pode-se agora tratar da aplicação dos testes unitários.

Os testes unitários foram utilizados aplicando-se a metodologia do TDD, de forma que primeiro a classe de teste fosse escrita, juntamente com seus métodos, e a classe Assert do framework JUnit invocada através de seus métodos para estes serem executados. Seguindo os passos do TDD, primeiramente foi construída a classe de teste e executada, para depois saber realmente o que precisava ser desenvolvido.

Após isso, a classe e o método foram criados, sendo que o método foi desenvolvido somente com o código necessário para compilação. O teste unitário foi executado e o JUnit apresentou novamente uma falha, pois não foi retornado o resultado esperado.

Dessa forma, a classe foi sendo refatorada e novos testes unitários foram sendo realizados, capturando falhas até que a funcionalidade fosse implementada e os testes passassem com sinal positivo, sem falhas.

Apesar da enorme quantidade de testes existentes, descritos no item 4.2, não é possível capturar todos os erros, não há uma garantia para que isso ocorra, mas testar mais frequentemente com a aplicação dos testes unitários reduzirá significativamente a sua quantidade tornando o sistema muito mais confiável, fácil de manter, agilizará as manutenções e juntamente com os benefícios apresentados no item 5.1, deixará o sistema com um bom padrão de qualidade, e foi por estes motivos que estes foram escolhidos para aplicação neste trabalho.

Desenvolvendo dessa maneira, muitos erros podem ser corrigidos no ponto correto e ainda, rapidamente. Fala-se em ponto correto, pensando em uma aplicação em camadas, como foi o caso do WEBDAP, onde muitos problemas foram encontrados na camada de interface, já que este não foi desenvolvido com a

utilização dos testes unitários, e desta forma, problemas ficaram pendentes na camada de regra de negócio, onde os métodos estão implementados e deveriam ser testados, e assim só foram percebidos na camada de interface.

Para o desenvolvimento da camada de interface, o correto é que os testes já tenham sido realizados, garantindo que as funcionalidades foram atendidas, para neste ponto preocupar-se somente em apresentar uma tela amigável e funcional para o cliente.

Os testes devem ser realizados em sua maioria na camada de regra de negócios, testando-se cada método e suas saídas esperadas, e como isso não foi feito no sistema WEBDAP, gerou muito trabalho no final, pois muitos erros foram encontrados, e dessa forma muito tempo e trabalho foram empregados para solucioná-los, e também na depuração do código. Não há dúvidas de que se o JUnit estivesse presente auxiliando no desenvolvimento, não haveria uma corrida contra o tempo como a que ocorreu mas sim uma corrida em busca de uma melhor qualidade no sistema, sem todo o estresse que gerou e a preocupação de que algo ainda estaria faltando.

6.1 CASO DE TESTE

Segundo Vale (2008 apud Bianchini 2009), o projeto dos casos de testes é importante por possibilitar a execução de várias maneiras as funções do software. Para verificar a aplicabilidade e poder avaliar os testes com o JUnit, torna-se necessário criar um caso de testes.

Para a aplicação dos testes unitários através do JUnit, foram utilizadas as seguintes ferramentas:

- NetBeans IDE versão 5.5 para desenvolvimento do código;
- Framework JUnit versão 3.8 que está acoplada no NetBeans.

Então, agora com os dados para cálculo da renda do agricultor, pode-se começar a escrever a classe para teste. Observa-se que de acordo com o TDD, primeiro se escreve o teste, depois a classe real que implementa o método. A figura

12 apresenta o modelo de classe para as classes Rebate e Renda, as quais serão aplicados os testes, e suas respectivas classes de teste.

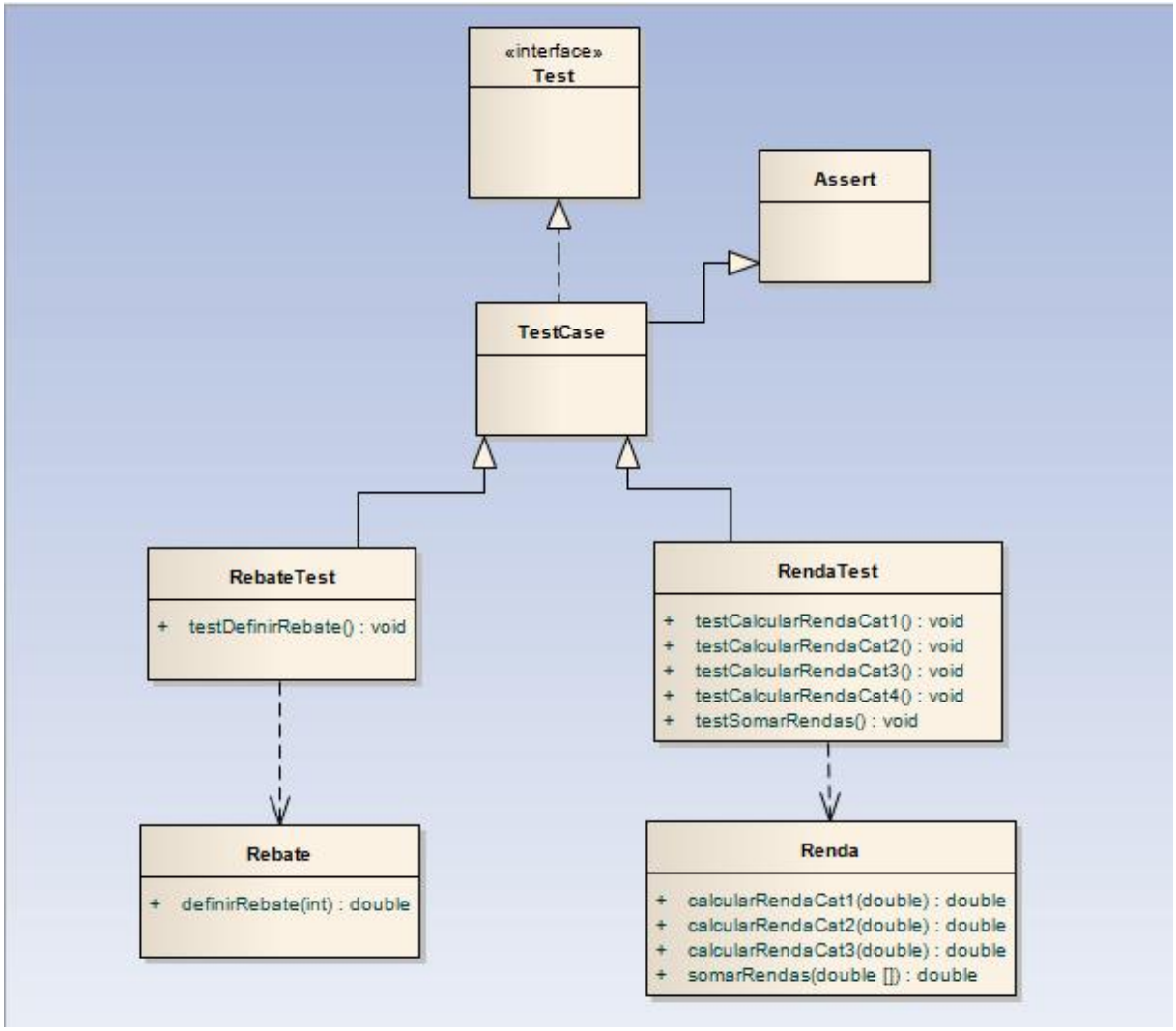

Figura 12: Diagrama de classes da aplicação
Fonte: Elaborado pelo Autor

Tendo os valores estabelecidos para porcentagem de rabate,começa-se a implementar o novo código para geração da renda do agricultor. Como já falado no capítulo 5, os testes em métodos de acesso não precisam ser realizados, então com as culturas devidamente organizadas em suas categorias o teste para composição da renda começa a ser realizado.

Antes de criar a classe Rebate, que terá os métodos para rebater a renda, será criada a classe RebateTest, com o método testDefinirRebate, que verificará se o esquema está correto.

A figura 13 apresenta a classe RebateTest.

```
1  /*
2   * RebateTest.java
3   * JUnit based test
4   *
5   * Created on 18 de Dezembro de 2010, 14:14
6   */
7
8  package renda;
9
10 import junit.framework.*;
11
12 /**
13  *
14  * @author Messias Flor Santos
15  */
16 public class RebateTest extends TestCase {
17
18     public void testDefinirRebate() throws Exception {
19
           Rebate rebate = new Rebate();
21
22         assertEquals(0.5, rebate.definirRebate(1));
23         assertEquals(0.7, rebate.definirRebate(2));
24         assertEquals(0.9, rebate.definirRebate(3));
25     }
26
27 }
28
```

Figura 13: Código da classe RendaTest
Fonte: Elaborado pelo Autor

Ao compilar a classe, 2 problemas foram encontrados: a classe Rebate não existe e o método definirRebate também não existe. Quando se desenvolve desta forma, o compilador irá mostrar aquilo que precisa ser feito, e desta forma, não ocorre aquele problema de codificar além do necessário, pois só será codificado o que for preciso para compilar e passar nos testes.

Após esta compilação, serão criados a classe e o método necessários para execução, figura 14. Após, uma nova tentativa de compilação e agora tudo está funcionando.

Continuando a implementação, o próximo passo é verificar se o código está passando nos testes. Nesta etapa o ideal é que o teste falhe, pois na verdade a funcionalidade do método ainda não foi implementada, e como já foi abordado neste trabalho, no TDD primeiro deve-se criar um teste que capture uma falha.

```
1  package renda;
2
3  /**
4   *
5   * @author Messias Flor Santos
6   */
7  public class Rebate {
8
9      public Double definirRebate(int categ){
10         return 0.0;
11     }
12
13 }
14
```

Figura 14: Código inicial da classe Rebate
Fonte: Elaborado pelo Autor

Executando o teste com JUnit através da IDE NetBeans, a falha esperada aparece, como mostra a figura 15.

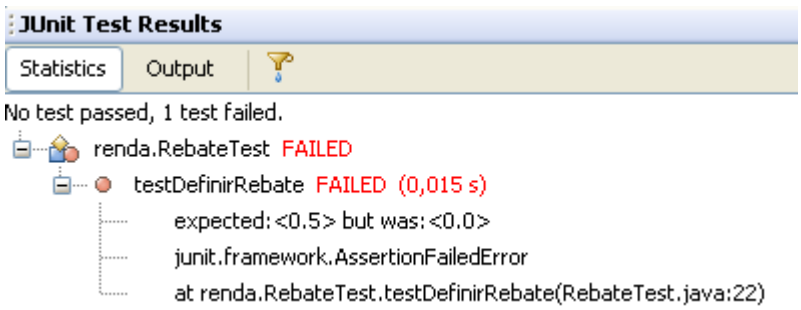

Figura 15: Teste do método definirRebate falhando
Fonte: Elaborado pelo Autor

O resultado mostra a falha obtida através da execução da classe com teste de unidade. O resultado esperado era 0,5, que de acordo com a implementação que dizer 50% de rebate, mas retornou 0. Agora, comprovada a falha, o método será desenvolvido.

```
1  package renda;
2
3  /**
4   *
5   * @author Messias Flor Santos
6   */
7  public class Rebate {
8
9      public Double definirRebate(int categ){
10         if(categ == 1){
11             return 0.5;
12          }
13         else if(categ == 2){
14             return 0.7;
15         }
16         else {
17             return 0.9;
18         }
19     }
20
21 }
```

Figura 16: Código do método definirRebate
Fonte: Elaborado pelo Autor

Com a implementação do código do método definirRebate, novamente será executado os testes. A figura 17 apresenta os resultados.

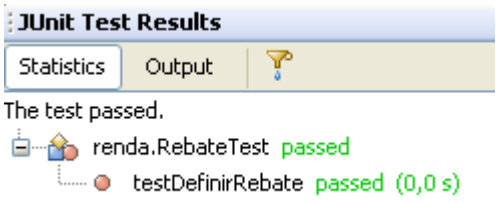

Figura 17: Testes passando no método definirRebate
Fonte: Elaborado pelo Autor

Finalmente os testes passaram, e então será passado para o próximo passo, implementação da classe para composição da renda. Assim como na definição do rebate, primeiro será desenvolvida a classe para teste.

O objetivo da classe renda é receber uma ou mais rendas e calcular a sua soma. As figuras a seguir mostram o passo a passo, como na implementação anterior, primeiramente mostrando o código de teste para a classe Renda, apresentada pela classe RendaTest na figura 18.

```java
package renda;

import junit.framework.*;

/**
 *
 * @author Messias Flor Santos
 */
public class RendaTest extends TestCase {

    Renda renda = new Renda();

    public void testCalcularRendaCat1(){

        assertEquals(5000.00, renda.calcularRendaCat1(10000.00));
        assertEquals(10000.25, renda.calcularRendaCat1(20000.50));
    }
    public void testCalcularRendaCat2(){

        assertEquals(3000.00, renda.calcularRendaCat2(10000.00));
        assertEquals(6000.15, renda.calcularRendaCat2(20000.50));
    }
    public void testCalcularRendaCat3(){

        assertEquals(1000.00, renda.calcularRendaCat3(10000.00));
        assertEquals(2000.09, renda.calcularRendaCat3(20000.90));
    }
    public void testSomarRendas(){

        double [] valores = {10000.25,3000.25,7000.15};

        assertEquals(20000.65, renda.somarRendas(valores));
    }
}
```

Figura 18: Código da classe RendaTest
Fonte: Elaborado pelo Autor

Após compilação e geração de erros, criou-se a classe Renda com seus respectivos métodos, mas apenas retornando o tipo esperado para capturas as falhas, e novo teste foi realizado, como mostra a figura 19.

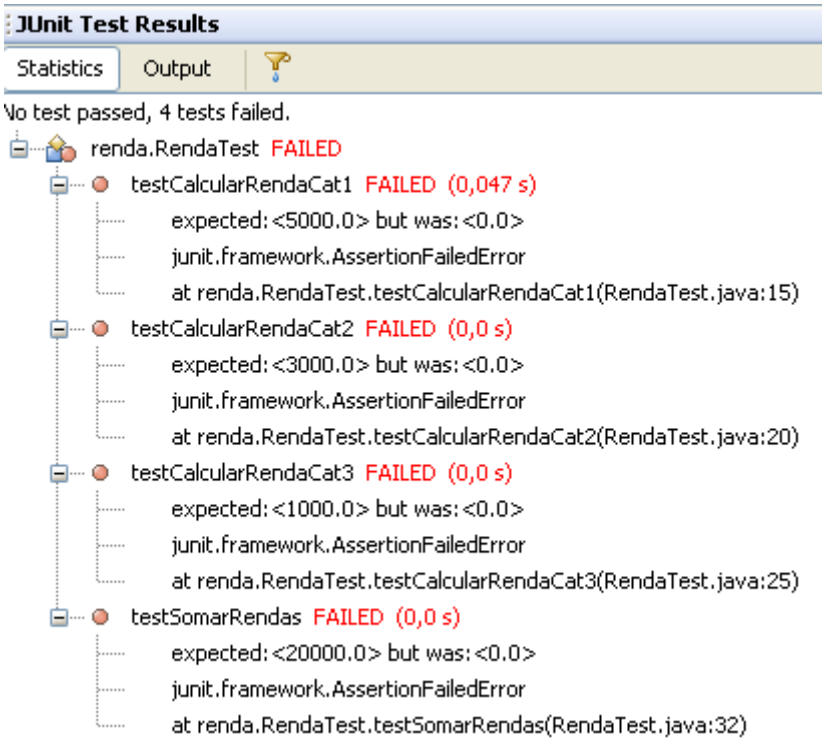

Figura 19: Testes na classe Renda
Fonte: Elaborado pelo Autor

Após a ocorrência destas falhas, os métodos serão agora implementados para atender a classe RendaTest. A figura 20 apresenta a classe Renda implementada.

```java
package renda;

import java.text.NumberFormat;

/**
 *
 * @author Messias Flor Santos
 */
public class Renda {

    public double calcularRendaCat1(double renda){
        renda = renda*0.5;
        return renda;
    }
    public double calcularRendaCat2(double renda){
        renda = renda*0.3;
        return renda;
    }
    public double calcularRendaCat3(double renda){
        renda = renda*0.1;
        return renda;
    }
    public double somarRendas(double [] valores){
        double renda = 0.0;
        for(int i=0;i<valores.length;i++){
            renda = renda + valores[i];
        }
        return renda;
    }
```

Figura 20: Código da classe Renda
Fonte: Elaborado pelo Autor

Com a classe implementada, novos testes são realizados, como pode ser observado através da figura 21.

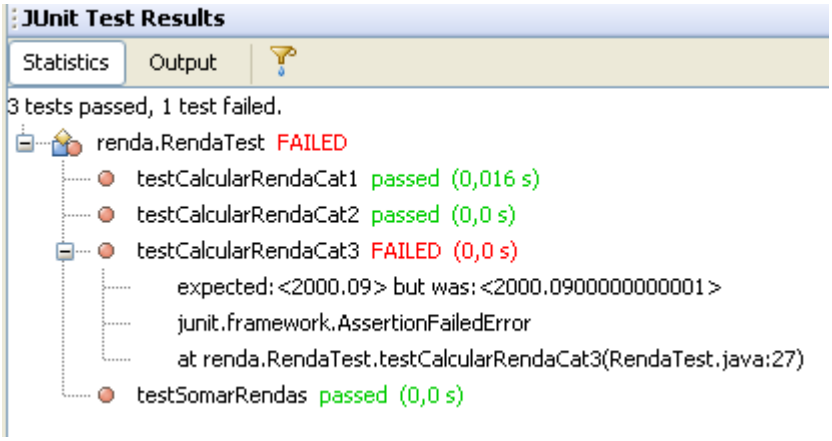

Figura 21: Testes falhando na classe Renda modificada
Fonte: Elaborado pelo Autor

Para surpresa, pois era esperado que este teste passasse, uma nova falha ocorreu e o valor retornado foi diferente do esperado. Um erro que acontece com a manipulação da variável *double*. Novamente, a classe Renda será modificada, alterando o código do método calcularRendaCat3. A figura 22 apresenta o novo método.

```
22    public double calcularRendaCat3(double renda) throws ParseException{
23        renda = renda*0.1;
24        String valor = numero.format(renda);
25        double novoValor = numero.parse(valor).doubleValue();
26        return novoValor;
27    }
```

Figura 22: Código do método calcularRendaCat3 modificado
Fonte: Elaborado pelo Autor

Para execução dos testes unitários, uma pequena modificação foi realizada na classe de teste para lançar a exceção para o método. Após, o teste foi realizado com o JUnit, e a figura mostra o resultado.

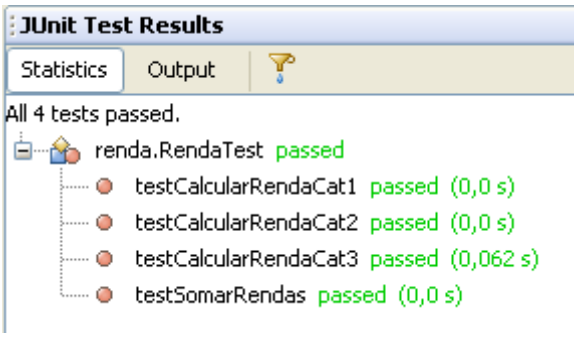

Figura 23: Teste passando na classe Renda
Fonte: Elaborado pelo Autor

Como pode ser observado, os métodos agora estão funcionando, graças aos testes unitários e a ajuda do JUnit. As falhas puderam ser encontradas em tempo de execução, e tudo pôde ser resolvido melhorando a qualidade da implementação.

6.2 ANÁLISES E RESULTADOS

Como visto neste trabalho, no item 4.2, existem diversos tipos de teste, e todos tem sua devida importância em cada processo de desenvolvimento, e devem ser realizados conforme a necessidade da empresa. Mas os testes unitários devem ser realizados pelas empresas que realmente desejam garantir a melhor qualidade do sistema e conseguir a confiança de seus clientes para se tornar uma empresa modelo em qualidade.

Desta forma, os testes unitários foram aplicados neste trabalho aumentando consideravelmente a qualidade da funcionalidade e garantindo uma maior abrangência da cobertura das falhas que poderiam ocorrer se estes não estivessem sido aplicados. Com sua aplicação, as falhas puderam ser corrigidas quase que instantaneamente pelo desenvolvedor, e ainda, já que foi utilizado o TDD, pensava-se já na solução para algumas falhas antes destas serem encontradas.

Pôde-se observar através desta aplicação em duas classes do sistema que se eles tivessem sido utilizados desde o início do desenvolvimento do sistema WEBDAP, um ganho significativo de tempo e trabalho seria conquistado, além de que, ver os testes serem executados e o JUnit retornando que tudo está *OK*, gera uma boa sensação de que seu trabalho está sendo feito de maneira correta e atendendo aquilo que está sendo esperado.

Como o sistema foi desenvolvido aplicando-se os testes somente no final, vários testes que são lembrados no momento do desenvolvimento deixaram de ser executados, pois nem todas as situações são lembradas para serem testadas. Esta forma de desenvolvimento pode até ser mais fácil para execução dos testes, gera menos trabalho, pois com os testes unitários os testes são menos fáceis, mas o trabalho é compensado por causa dos erros que precisam ser corrigidos no final.

A cada dia como desenvolvedor aprende-se novas coisas e no desenvolvimento deste trabalho, não foi diferente. Os testes unitários fazem você garantir que quase tudo está perfeito, pois como foi dito no capítulo 6, não há como afirmar que não existem mais erros ou falhas, mas com esses testes você pode garantir que se obteve uma melhor qualidade, que se você aplicou eles nos pontos críticos e garantiu que todos estão funcionando, dificilmente os erros aparecerão, e se aparecer algum, rode a bateria de testes com este problema encontrado, e ache o ponto exato para realizar a manutenção.Desta forma toma-se como lição que mesmo tendo diversos tipos de testes para realizar, os testes unitários não podem

deixar de entrar nesta lista, e não devem ser considerados uma atividade extra, como já foi dito no item 5.4, mas sim um diferencial para garantir que o sistema está atendendo suas funcionalidade com a melhor qualidade possível.

Os erros quando encontrados no momento do desenvolvimento são fáceis e rápidos de serem solucionados, e isto gera uma boa otimização, pois por experiência, sabe-se que quando não se executa os testes unitários na parte lógica da aplicação, quando há muita pressa e é passado direto para etapa de interface com o usuário, certamente os erros aparecem e o que acontece é que até lembrar e focar o pensamento novamente na lógica e no comportamento real esperado, tempo e trabalho na depuração são perdidos, e sabe-se que tempo é dinheiro. Então eles podem ser considerados otimizadores de tempo, pois o que é utilizado para criá-los e executá-los na verdade gera ganho no final.

Para a empresa que utiliza os testes unitários pode ocorrer certa resistência por parte dos desenvolvedores no início, mas tornando eles parte do processo de desenvolvimento e exigindo como parte de uma entrega, isso acaba se tornando um padrão na empresa e depois disso a equipe não se sente mais segura se eles não são executados, o desenvolvedor fica mais confiante com o seu trabalho e garante que o que está sendo produzido, está sendo bem feito. Na empresa onde o autor deste trabalho desenvolve, alguns projetistas exigem que eles sejam entregues, e quando eles não são realizados, quase sempre o sistema volta para manutenção. Este é um exemplo claro de que eles realmente fazem a diferença.

Com tudo isso, quem mais tem a ganhar é o cliente que recebe um produto que atende suas necessidades e está quase livre de erros, facilitando e agilizando seu trabalho, e não tendo muitos problemas para serem reportados para equipe de desenvolvimento, fazendo também com que seu serviço seja mais produtivo e menos estressante, e o sistema sirva como um diferencial em seu trabalho, não como um empecilho, como ocorre algumas vezes.

7 CONCLUSÕES E TRABALHOS FUTUROS

A qualidade é muito importante em todos os produtos e serviços, e por isso, há algum tempo deixou de ser um diferencial competitivo e se tornou um requisito essencial para a empresa permanecer e ter sucesso no mercado.

Em softwares ela é fundamental porque se aplicada durante todo o processo de desenvolvimento, gera um produto mais robusto, mais estruturado, possibilitando ao usuário mais segurança, mais habilidade para desenvolver suas atividade, e confiança para que seu trabalho possa ser feito também com qualidade.

Erros em softwares são grandes problemas quando descobertos somente pelos usuários, e se agravam ainda mais quando interferem ou interrompem o serviço que está sendo realizado. Quando isso ocorre, a empresa desenvolvedora acaba perdendo a credibilidade com o cliente, consequentemente com o mercado e se torna vulnerável a concorrência perdendo espaço, confiança de seus clientes, e dependendo de sua base, pode ir à falência.

Para o cliente, quando acontece algum problema e este pode ser solucionado com rapidez, não há tanto impacto, mas caso seja um problema mais grave, como o citado anteriormente, isso gerará entre outras coisas perda de lucro ou talvez até prejuízo, e isso é o mais agravante que pode acontecer para o gestor cliente, e dessa forma a reação não será das mais amistosas, com certeza isso que foi perdido será cobrado de alguma forma da empresa desenvolvedora, ou quem sabe em casos mais graves, até um processo contra a empresa, gerando grande incômodo entre as partes e manchando ainda mais a sua imagem.

Para que isso não aconteça, é imprescindível que não haja falhas no sistema e que este forneça suporte para a realização das tarefas e auxilie o usuário ao invés lhe trazer mais problemas.

Como existem diversas formas para a empresa garantir melhor qualidade em seus sistemas, desde melhoria em seus processos organizacionais, padronização destes processos, utilizando padrões nacional ou internacionalmente conhecidos, conforme a abrangência e metas que a empresa deseja alcançar, ela deve optar por um caminho a ser seguido a fim de alcançar um nível de qualidade alto que garanta a satisfação de seus clientes.

Para conseguir maior qualidade do software devem ser realizados os testes, que são vitais para os projetos, e devem ser sempre realizados, conforme a necessidade de cada sistema, sendo bem elaborados e praticados para que o sistema possa atingir a satisfação do usuário e ser reconhecido como um produto de

qualidade. Quanto mais testes puderem ser realizados, melhor será a qualidade, então, quanto mais tempo puder ser investido nesta etapa, melhor será a garantia de qualidade.

Por isso também os testes unitários são ótimos para melhorar a qualidade, pois a cada código desenvolvido está se garantindo mais qualidade, possibilitando que ao final do desenvolvimento não ocorram muitos erros para serem corrigidos, filtrando e testando cada parte crítica, investindo um tempo maior na sua implementação, que ao final do projeto será recompensada, já que se gasta muito tempo na descoberta e reparação dos erros quando estes não são encontrados precocemente.

Os testes unitários podem ser executados um a um, e no final do desenvolvimento, todos de uma vez para garantir que tudo está rodando perfeitamente, e que os requisitos solicitados pelo cliente estão todos implementados e validados através destes testes.

Então os testes unitários possibilitam um código eficaz e de fácil manutenção, pois quando uma alteração precisa ser realizada, os testes estarão prontos para rodar e verificar se a modificação afetou alguma parte que estava funcionando, agilizando e favorecendo entregas mais rápidas para o cliente.

Desta forma, o investimento na qualidade deve ser constante, inclusive com o tempo para realização de testes unitários, para permitir que seja entregue um sistema com o mínimo de erros possíveis, já que um sistema 100% livre destes não existe, para que o usuário possa ficar satisfeito e assim a empresa conseguir cada vez mais expandir seus negócios através da oferta de um produto diferenciado que possa ser reconhecido nacional ou internacionalmente, e permaneça por quanto tempo desejar no mercado.

Como sugestão para trabalhos futuros sugere-se que sejam aplicados os testes unitários através do JUnit em classes e métodos mais complexos e que sejam explorados novos frameworks para aplicação dos testes em outras linguagens de programação.

REFERÊNCIAS

ALATS - Associação Latino Americana de Teste de Software. **MPT. BR – Melhoria de Processo de Teste de Software Brasileiro.** Disponível em: http://www.alats.org.br/Default.aspx?tabid=252. Acesso em: 16 jul. 2010.

AMBLER, Scott W. **Introduction to Test Driven Design (TDD).** Disponível em: HTTP://WWW.AGILEDATA.ORG/ESSAYS/TDD.HTML. Acesso em 03 Set. 2010.

_____. **Modelagem Ágil:** Práticas eficazes para a programação extrema e o processo unificado. Porto Alegre: Bookman, 2004.

BARTIÉ, Alexandre. **Garantia da Qualidade de Software.** Rio de Janeiro: Elsevier, 2002.

BECK, Kent. **Programação extrema (XP) explicada**: acolha as mudanças. Porto Alegre: Bookman, 2004a.

_____. **JUNIT:** Pocket guide. United States of America: O'Reilly Media, 2004b.

BECK Kent, et al. **Manifesto for Agile Software Development**. Disponível em: http://www.agilemanifesto.org. Acesso em: 14 jun. 2010a.

_____. **Principles behind the Agile Manifesto.** Disponível em: http://www.agilemanifesto.org/principles.html. Acesso em: 14 de jun. 2010b.

BIANCHINI, Ricardo Alexandre. **Avaliação de ferramentas de testes para uso no desenvolvimento de soluções web.** 2009. 47f. Monografia (Especialização em Engenharia de Projetos de Software) – Universidade do Sul de Santa Catarina, Palhoça, 2009.

FOWLER, Martin. **Refatoração:** aperfeiçoando o projeto do código existente. Porto Alegre: Bookman, 2004.

GOLDENSON, Dennis R., GIBSON, Diane L., KOST, Keith. **Performance Results of CMMI-Based Process Improvement.** Disponível em: http://www.sei.cmu.edu/reports/06tr004.pdf. Acesso em: 12 jul. 2010.

HAMILL, Paul. **Unit Test Frameworks**. . United States of America: O'Reilly Media, 2004.

INTHURN, Cândida. **Qualidade & Teste de Software.** Florianópolis: Visual Books, 2001.

KALINOWSKI, Marcos et al. **MPS.BR:** Promovendo a Adoção de Boas Práticas de Engenharia de Software pela Indústria Brasileira. Disponível em: http://www.softex.br/portal/softexweb/uploadDocuments/CIBSE2010_MPSBR_CameraReady.pdf. Acesso em: 13 jul. 2010.

KOSCIANSKI, André; SOARES, Michel dos Santos. **Qualidade de software:** aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. 2ª Ed. São Paulo: Novatec Editora, 2007.

JUNIT Sourceforge. **JUnit A Cook's Tour.** Disponível em: http://junit.sourceforge.net/doc/cookstour/cookstour.htm. Acesso em: 11 abr. 2012.
.

MARINESCU, Floyd. **Padrões de Projeto EJB.** Porto Alegre: Bookman, 2002.

MASSOL, Vincent; HUSTED, Ted. **Junit em Ação.** Rio de Janeiro: Ciência Moderna, 2005.

MCT – Ministério da Ciência e Tecnologia. **Qualificações CMM e CMMI no Brasil.** Disponível em: http://www.mct.gov.br/upd_blob/0009/9238.pdf. Acesso em: 13 jul. 2010.

MOLINARI, Leonardo. **Testes de Software:** Produzindo sistemas melhores e mais confiáveis. São Paulo: Érica, 2003.

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

RAINSBERGER, J.B. **JUnit recipes:** practical methods for programmer testing. United States: Manning Publications, 2004.

RICK, Nilton Ariberto. **Crédito Rural:** Safra 2010/2011. Araranguá, 2010. Apostila elaborada pelo agente técnico do banco do Brasil.

SANTOS, Messias Flor; HESPANHOL, Mariana Machado. **Sistema Web para DAP - WEBDAP**. 2007. 127 f. Monografia (Graduação em Sistemas de Informação) – Universidade do Sul de Santa Catarina, Araranguá, 2007.

SEI - Software Engineering Institute. **Capability Maturity Model Integration.** Disponível em: http://www.sei.cmu.edu/cmmi/. Acesso em: 12 jul. 2010.

SOFTEX. **Guias.** Disponível em: http://www.softex.br/mpsbr/_guias/default.asp. Acesso em: 14 jul. 2010a.

_____. **Guia Geral**. Disponível em: http://www.softex.br/mpsbr/_guias/guias/MPS.BR_Guia_Geral_2009.pdf. Acesso em: 14 jul. 2010b

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

Standish Group. **The Chaos Report 2009.** Disponível em: http://www.standishgroup.com. Acesso em: 14 jul. 2010

TELES, Vinícius Magalhães. **Extreme Programming**: Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade. São Paulo: Novatec Editora, 2006.

TRAVASSOS, Guilherme Horta; KALINOWSKI, Marcos. **iMPS 2009 :** caracterização e variação de desempenho de organizações que adotaram o modelo MPS. Campinas, SP: SOFTEX, 2009. Disponível em: http://www.softex.br/mpsbr/_livros/arquivos/Softex%20iMPS%202009%20Portugues_vFinal_12jan10.pdf. Acesso em: 16 jul. 2010.