ETIC 2014 - Encontro de Iniciação Científica
ISSN 21-76-8498

TESTANDO MAIS PARA DEPURAR MENOS: OS BENEFICIOS DOS TESTES PARA O DESENVOLVIMENTO DE SOFTWARE

Willian Gilson de Melo[1]
Álvaro Ferraz d'Arce[2]

RESUMO: Neste artigo, serão abordadas três metodologias de testes de desenvolvimento de software, visando apresentar os benefícios da utilização de testes antes, durante e após o desenvolvimento.

Palavras Chave: Testes. Caixa Branca. Caixa Preta. Testes Unitários. Qualidade de Software.

1 INTRODUÇÃO

Atualmente a questão de testes é muito abordada na produção de produtos e na prestação serviços visto que as empresas querem cada vez mais ampliar a satisfação do cliente e consequentemente aumentar sua gama de vendas.

No entanto quando referenciamos testes ao desenvolvimento de software, este requisito fica difundido apenas em grandes empresas visto que grande parte das pequenas empresas enxerga esta pratica como desperdício de tempo e dinheiro. Diante de tal pratica questiono vocês como desenvolvedor-empresários. "– Você viajaria em um avião que vai sair do chão pela primeira vez?" Provável resposta: "Não", quais os motivos que o levam a realizar testes falhos ou até menos não os realizar em seu projeto de software? Cumprir prazos? Diminuir custos? Estas e outras questões serão abordadas no decorrer deste artigo.

[1] Discente do 4º ano do curso de Sistemas de Informação das Faculdades Integradas "Antonio Eufrásio de Toledo" de Presidente Prudente. willianmelo@unitoledo.br
[2] MSc. em Ciência da Computação na linha de Engenharia de Software. Docente do curso de Sistemas de Informação das Faculdades Integradas "Antônio Eufrásio de Toledo". alvaro@darce.com.br.

2 PORQUE TESTAR?

Quando aplicamos testes eficazes temos a chance de diminuir inúmeras falhas de lógica e/ou programação, visto que se for utilizado algum método de testes de maneira correta os erros podem ser detectados e corrigidos quase no mesmo momento em que foram implementados, desta maneira otimizando o código e contribuindo para o bom desenvolvimento do software.

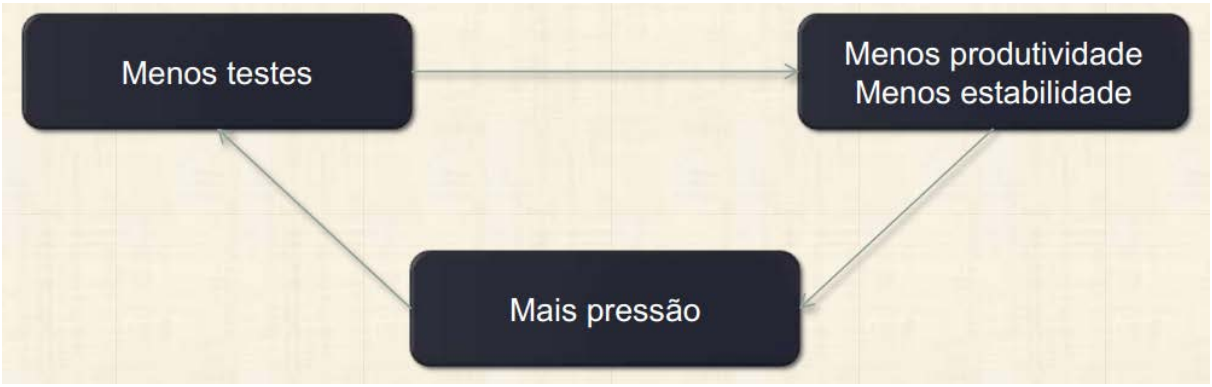
Outro bom motivo para se utilizar dos testes é a satisfação do cliente, um produto final de qualidade vai fazer a diferença no momento de uma indicação, uma renovação de contrato, outra solicitação, etc.

“Um desempenho abaixo do esperado frustra os consumidores e custa ás empresas bilhões de dólares. “(Menásce e Almeida, 2001).

É plausível afirmar que quando os desenvolvedores ficam procurando e consertando falhas o projeto não evolui e consequentemente para no tempo por certo período.

Como mostra a figura 1 quando não é aplicado nenhum tipo de teste eficaz no momento do desenvolvimento ou até mesmo antes da entrega se cria um ambiente de desenvolvimento ruim, tanto para empresas quanto para o desenvolvedor.

FIGURA 1 – Ciclo vicioso.



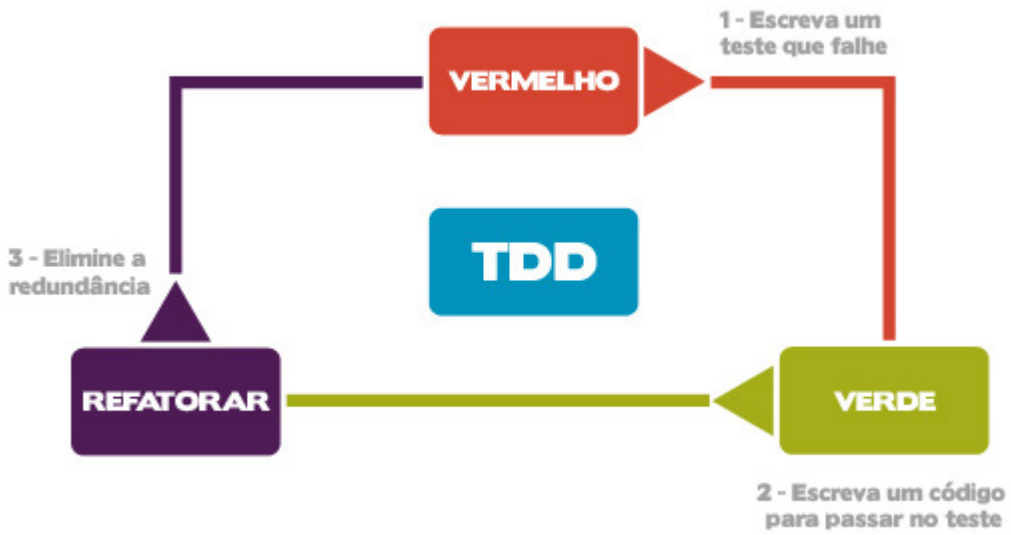
3 METODOLOGIAS DE TESTES

Como vimos anteriormente os testes se fazem necessário para se desenvolver um produto de software com qualidade, a seguir serão explanados três tipos de metodologias de testes: testes unitários, caixa branca e caixa preta.

3.1 TESTES UNITARIOS

TDD é o desenvolvimento de software orientado a testes, ou em inglês,Test Driven Development. Ele se baseia em três passos, vermelho-verde-refatora. O vermelho é a escrita do primeiro teste antes mesmo da lógica existir. O verde é o ponto em que a lógica para que o algoritmo previamente criado passe no teste. Esta lógica deve ser desenvolvida da maneira mais simples possível, eliminando complexidades desnecessárias e fazendo com que a evolução do código ocorra de maneira segura. O refatora é a melhoria do código testado. A figura 2 mostra este processo mais claramente. Ressaltando que os testes devem ser atualizados e devem evoluir junto com o desenvolvimento do software.

FIGURA 2 - Processo para aplicação do TDD.



Fabio Gomes explicita em seu artigo sobre testes unitários que os benefícios da aplicação destes são:

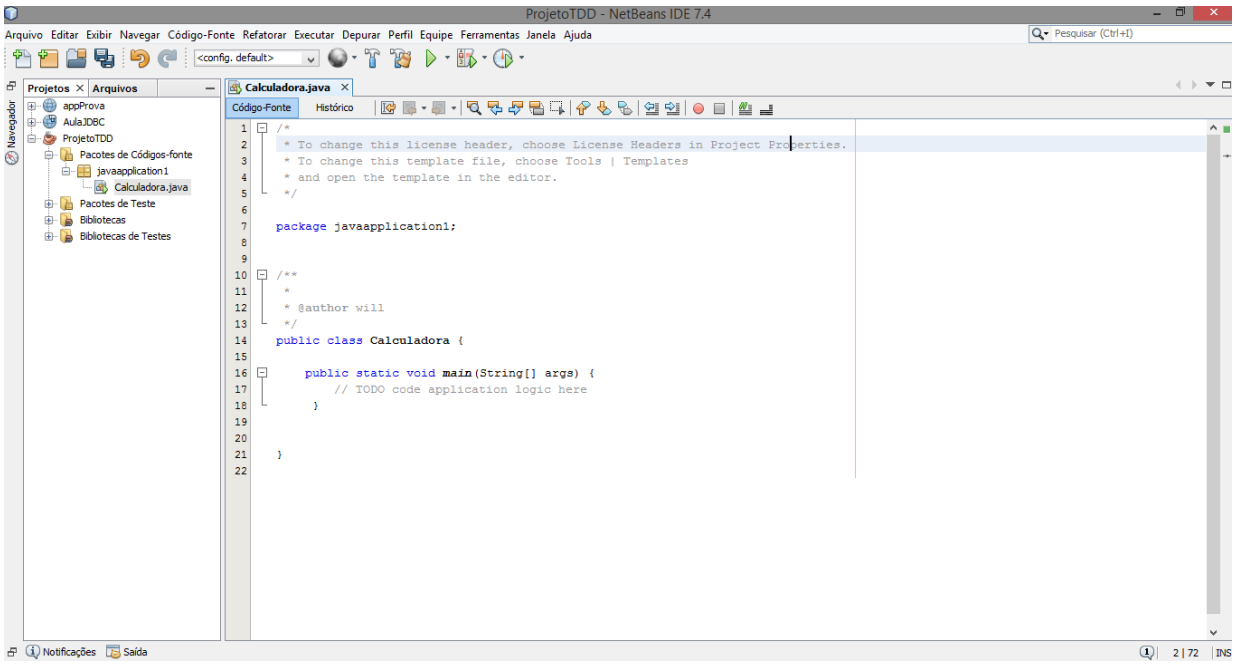
> Em primeira instância, torna o processo mais confiável, reduz custos, pois desenvolvemos quando já sabemos o erro, visto que os testes são criados antes do processo de desenvolvimento, conseguimos testar constantemente. Outro ponto é que se os testes foram criados, isso quer dizer que foram entendidas as regras de negócio durante a fase de desenvolvimento dos testes unitários.

Além dos benefícios citados, quando os testes são bem aplicados e estão atualizados também servem de documentação, para os desenvolvedores isto é ótimo, pois é mais pratico para eles analisarem o código dos testes para entender determinado algoritmo do que foliar paginas e paginas atrás dos métodos documentados.

Existem vários frameworks que auxiliam no processo de criação de testes unitário o JUnit para Java, Jasmine para Javascripit, CUnit para C, entre outros.

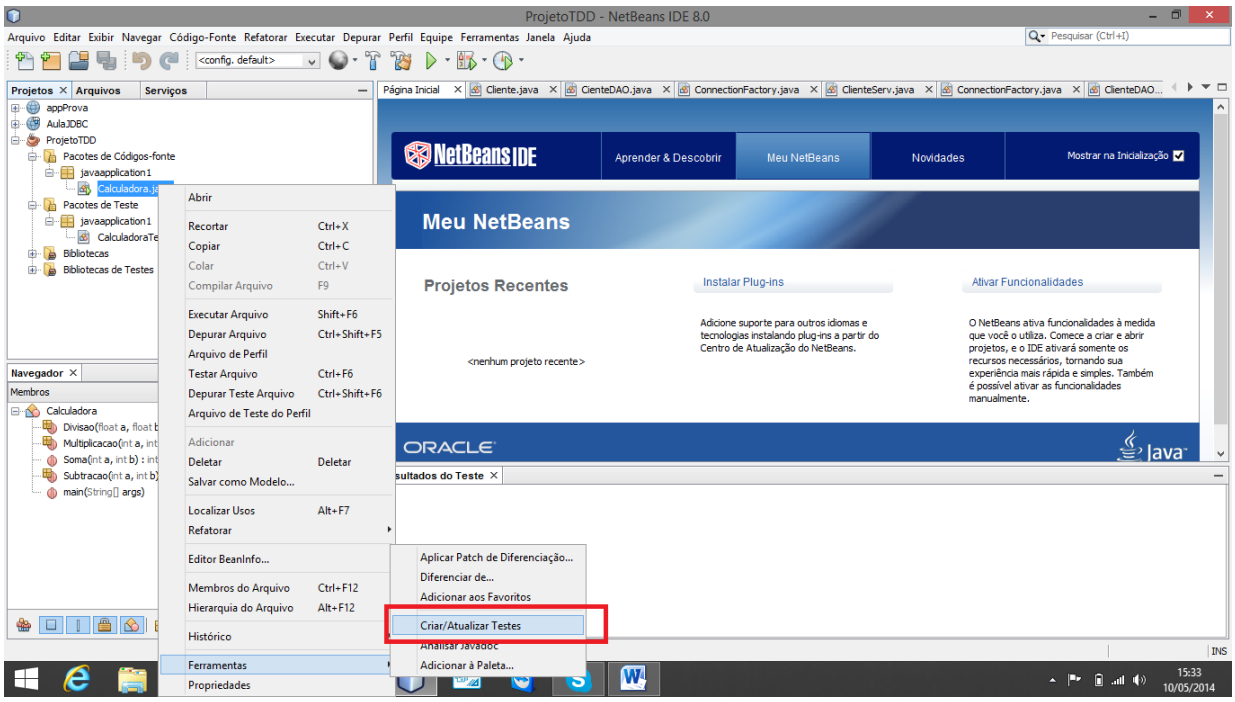
A seguir o exemplo passo-a-passo com a aplicação de testes unitários no desenvolvimento de uma calculadora se utilizando do framework JUnit e a IDE Netbeans.

FIGURA 3 – Criação do projeto e da classe Calculadora.



Na figura 3 é criado um projeto no Netbeans, chamado de ProjedoTDD e nele uma classe chamada “Calculadora” observem que não existe código algum nesta classe, mas já vamos criar os testes como mostra na figura 4.

FIGURA 4 – Criação dos testes.

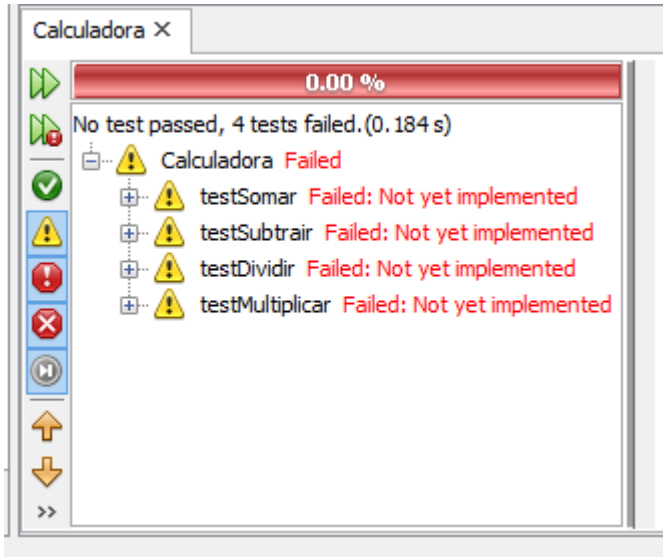


Já na figura 7, podemos começar a observar o processo citado na figura 2, onde os testes são criados para falhar.

FIGURA 5 – Criação dos testes.

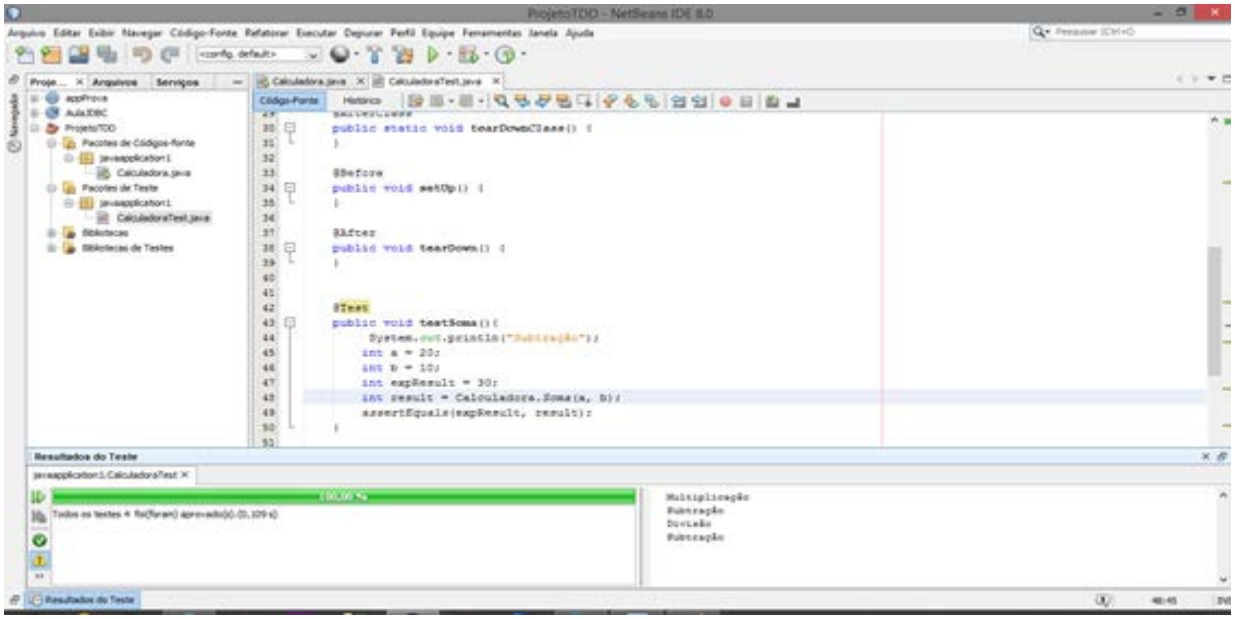
```
19  public class Calculadora {
20
21      public Calculadora() {
22      }
23
24      @Test
25      public void testSomar()
26      {
27          fail("Not yet implemented");
28      }
29
30       @Test
31      public void testSubtrair()
32      {
33          fail("Not yet implemented");
34      }
35
36       @Test
37      public void testMultiplicar()
38      {
39          fail("Not yet implemented");
40      }
41
42       @Test
43      public void testDividir()
44      {
45          fail("Not yet implemented");
46      }
47
48  }
```

FIGURA 6 – Erro de teste.



Após a criação dos testes e dos mesmos retornarem vermelho, é a hora da criação dos métodos da classe Calculadora (somar, subtrair, dividir e multiplicar). Como mostra na figura 6 e na figura 7.

FIGURA 7 – Criação dos métodos da classe principal.



Na imagem acima foi criada a classe principal e já foi executado o JUnit para analise, o mesmo passou (Verde), isto significa que se os testes foram aplicados de maneira eficiente e passaram, o código está “livre” de falhas.

Não será necessário refatorar visto que o código é curto, e não possui redundância.

3.2 TESTES CAIXA BRANCA

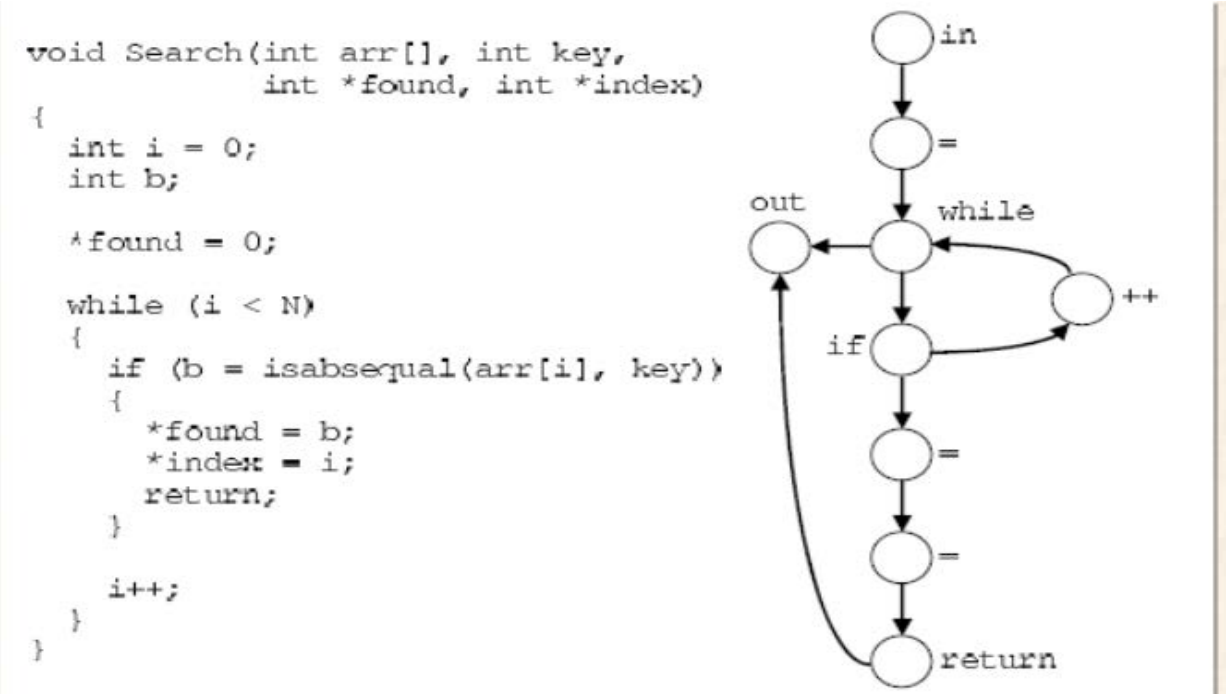
Teste caixa branca ou teste estrutural é uma metodologia é aplicada após o desenvolvimento de um módulo ou classe, e tem como intuito testar os caminhos possíveis, é necessário que o testador tenha amplo acesso e conhecimento em programação, visto que os testes serão aplicados no código fonte. Esta metodologia tem a intenção de testar se os caminhos e não se a lógica está correta.

“A desvantagem da técnica de caixa de caixa branca é que não analisa se a especificação esta certa, concentra apenas no código fonte e não verifica a lógica da especificação .“ (LEWIS e VEERAPILLAI, 2005).

Este tipo de teste aumenta o custo do software visto que testar todos os caminhos será praticamente impossível e também muito árduo para o testador

A quantidade e quais os caminhos mínimos a serem testados são definidos por grafos, que são nada mais que a representação gráfica do código fonte, na figura 8 pode – é possível observar os caminhos possíveis para determinado código fonte.

FIGURA 8 – Utilização de grafos.



A técnica de testes caixa branca funciona melhor quando aplicada junto aos testes caixa preta, como foi dito acima o caixa branca apenas testa os caminhos e os testes do tipo caixa preta verifica a operacionalidade do sistema, ou seja, um faz o inverso do outro testando os dois lados do software (código e funcionalidade).

3.2 TESTES CAIXA PRETA

Esta técnica visa testar a operacionalidade do sistema não se preocupando com o modo que foi implementado o software, basicamente teste as entradas e verifica as saídas de cada tela, módulo ou algoritmo. Por exemplo, verificar se um cpf foi validado, se é possível inserir uma data de nascimento igual a data atual ou até uma data futura, entre outros.

3.2.3 PARTIÇÃO DE EQUIVALÊNCIA

Nesta parte do teste caixa preta é verificado as possíveis entradas e suas possíveis saídas de cada tela então é feito uma tabela que contem: Condições de entrada, classes de equivalência validas e classes de equivalência invalidas. A figura 9 mostra um exemplo de partição de equivalência.

FIGURA 9 - Exemplo partição de equivalência.

Condições de Entrada	Classes de Equivalência Válidas	Classes de Equivalência Inválidas
DDD pode estar presente (booleana)	Sim (1) Não (2)	
Valor DDD definido (intervalo)	0011 <= DDD <= 0999 (3)	DDD < 0011 (4) DDD > 0999 (5)
Valor Prefixo especificado (intervalo)	20 <= Prefixo <= 99 (6)	Prefixo < 20 (7) Prefixo > 99 (8)
Tamanho do Sufixo (valor)	Tamanho = 4 (9)	Tamanho < 4 (10) Tamanho > 4 (11)
Senha pode estar presente (booleana)	Sim (12)	Não (13)
Tamanho da Senha (valor)	Tamanho = 6 (14)	Tamanho < 6 (15) Tamanho > 6 (16)
Comando (conjunto)	Comando = "aplicação" ou "depósito" ou "extrato" ou "saldo" (17)	Comando não pertence ao conjunto especificado (18)

Visto que nas condições de entrada são definidas as entradas e seus respectivos tipos de condição de entrada. As classes válidas representam as condições de entrada que satisfazem os requisitos de entrada do módulo sendo testado, e as classes inválidas representam as condições de entrada que violam tais requisitos.

4 CONCLUSÃO

A aplicação de testes no desenvolvimento de software está em ascensão, cada vez mais desenvolvedores e empresários do ramo estão descobrindo os benefícios da utilização dos testes de maneira eficaz, os resultados são vistos em

um produto final de qualidade, para empresa isto é excelente pois tem como feedback de seus esforços no investimento em testes: redução de custos, redução de manutenções de reparo, melhorias no código fonte, desenvolvedores mais preparados além de ter um cliente satisfeito. Ou seja, se os testes forem utilizados de maneira eficaz, o projeto como um todo terá um ganho em eficiência.

5 REFERENCIAS

BROCKA, Bruce; BROCKA, M. Suzanne. **Gerenciamento da qualidade.** São Paulo: Makron Books, 1994.

DELAMARO, Márcio Eduardo; MALDONADO, José Carlos; JINO, Mario. **Introdução ao teste de software.** Rio de Janeiro: Elsevier, Campus, 2007.

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

PRESSMAN, Roger S. **Engenharia de software.** 6. ed. Porto Alegre: McGraw-Hill, Bookman, AMGH, 2010.

Sommerville. **Engenharia de software.** 8. ed. São Paulo: Pearson Addison Wesley, 2007.

TDD: FUNDAMENTO DO DESENVOLVIMENTO ORIENTADO A TESTES. Disponível em:< http://www.devmedia.com.br/tdd-fundamentos-do-desenvolvimento-orientado-a-testes/28151#ixzz32D023YjZ> Acesso em 12 de maio de 2014.

TESTE DE CAIXA-BRANCA. Disponível em:< http://qualidade-de-software.blogspot.com.br/2010/01/teste-de-caixa-branca.html> Acesso em 08 de maio de 2014.

TESTES DE SOFTWARE - TESTE UNITÁRIO. Disponível em:< http://www.devmedia.com.br/testes-de-software-teste-unitario/22284> Acesso em 18 de maio de 2014.

UMA VISÃO DA TÉCNICA DE TESTE DE CAIXA BRANCA. Disponível em: <http://www.devmedia.com.br/uma-visao-da-tecnica-de-teste-de-caixa-branca/15610> Acesso em 10 de maio de 2014.