INTERFACE TECNOLÓGICA

A IMPORTÂNCIA DOS TESTES DE SOFTWARE

THE IMPORTANCE OF SOFTWARE TESTING

Bruno Henrique Silvestre – brunosilvestre201010@gmail.com
Faculdade de Tecnologia de Taquaritinga (Fatec) – Taquaritinga – SP – Brasil

Giuliano Scombatti Pinto – giuliano.pinto@fatectq.edu.br
Faculdade de Tecnologia de Taquaritinga (Fatec) – Taquaritinga – SP – Brasil

DOI: 10.31510/infa.v20i2.1775
Data de submissão: 06/09/2023
Data do aceite: 16/11/2023
Data da publicação: 20/12/2023

RESUMO

Os Softwares têm auxiliado os processos do cotidiano e trazido muita facilidade e tranquilidade para quem os utiliza. Porém em alguns casos, eles podem ser disponibilizados com problemas para o usuário final, fazendo assim com que haja descontentamento com a utilização dessas ferramentas. O objetivo deste artigo é explicar e contextualizar a origem desses problemas, demonstrando em processo os verdadeiros conceitos de erro, defeito, engano e falha, mostrar como as empresas responsáveis por essas ferramentas e seus funcionários devem lidar com eles da maneira correta, explicar os métodos e tipos diferentes de teste e em que ocasiões devem ser aplicados e por fim demonstrá-los na prática. As metodologias utilizadas para a realização do presente artigo foram a pesquisa bibliográfica e o estudo de caso. Após o estudo realizado foi possível verificar na prática como a ausência da realização dos testes pode trazer malefícios para todas as partes envolvidas e como a realização correta desses testes pode aumentar a lucratividade para as empresas responsáveis pelo desenvolvimento dos Softwares.

Palavras-chave: Software. Tipos de Testes de Software. Prevenção de Erros. Testes Unitários.

ABSTRACT

The Softwares has been helping with daily processes and brought facility and tranquility to those who use them. However, in some cases, it can be made available with problems for the end user, thus causing dissatisfaction with the use of these tools. The purpose of this article is to explain and contextualize where these problems arise from, demonstrating in process the true concepts of error, defect, mistake and failure, to show how the companies responsible for these tools and their employees must deal with them in the right way, explain the methods and different types of tests and when they should be applied and finally demonstrate them in practice. The methodologies used to carry out this article were the bibliographic research, which consists of examining materials such as books and articles in order to better understand the phenomenon studied and support the thesis, and the case study, which consists of

analyzing real cases to demonstrate the application and thus reinforce the thesis. After the study carried out, it was possible to verify in practice how the absence of carrying out the tests can bring harm to all parties involved and how the correct performance of these tests can increase profitability for the companies responsible for the development of the Software.

Keywords: Software. Types of Software Tests. Error Prevention. Unitary Tests.

1 INTRODUÇÃO

Os softwares são ferramentas muito úteis que têm o objetivo de sanar necessidades de várias maneiras diferentes, como por exemplo: facilitar a realização de tarefas do cotidiano, trazer entretenimento para as pessoas, automatizar processos industriais, auxiliar no controle de informações dentro de empresas, auxiliar na comunicação, dentre outros.

Para que possa ser criada uma ferramenta capaz de realizar tarefas complexas, o nível de complexidade em seu desenvolvimento e manutenção tende a ser alto também. Devido a sua tamanha complexidade, irão ocorrer problemas não mapeados durante esse processo e a maioria deles serão causados por variáveis de natureza humana.

Entregar um software com problemas para um usuário final pode gerar consequências negativas, pois o não cumprimento das expectativas quanto ao seu funcionamento irá gerar descontentamento e transtorno para o usuário.

Para que esse tipo de situação não ocorra com os usuários finais dos softwares, foi estabelecido através de estudos, que haja uma ou mais pessoas para realizar os chamados “Testes de Software”.

O Teste de Software segundo Sommerville (2011, p. 144) “é destinado a mostrar que um programa faz o que é proposto a fazer e para descobrir os defeitos do programa antes do uso.”

Quando se testa um software, o programa é executado usando dados fictícios e os resultados dos testes são verificados em busca de erros, anomalias ou informações sobre atributos não funcionais do programa (SOMMERVILLE, 2011).

O objetivo do presente artigo é demonstrar alguns conceitos essenciais para a realização de testes, demonstrar em casos reais como se deve lidar com problemas em softwares e explicar sobre alguns dos métodos mais conhecidos e empregados de testes.

INTERFACE TECNOLÓGICA

2 CONCEITOS FUNDAMENTAIS PARA OS TESTES DE SOFTWARE

2.1 Erro, defeito, engano e falha

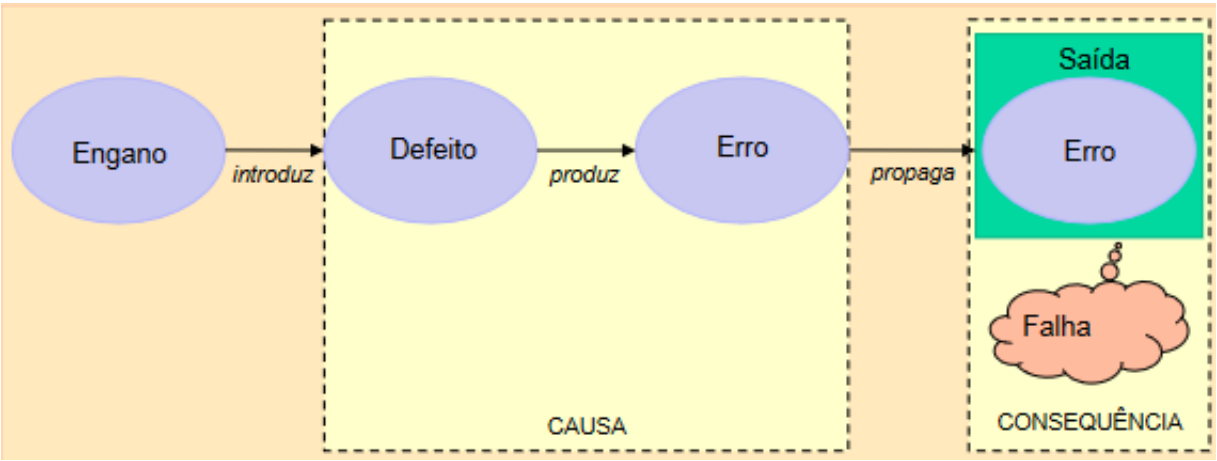
Os conceitos de erro, defeito, engano e falha costumam ser confundidos em meios de comunicação informais, porém para o uso acadêmico é necessário adotar uma visão técnica dos termos e identificar o conceito e a aplicação correta deles para que não haja nenhum desvio dos resultados da pesquisa.

Segundo Maldonado *et al.* (2007, p. 2):

> Definimos defeito (do inglês, *fault*) como sendo um processo, passo ou definição de dados incorreto e engano (*mistake*) como a ação humana que produz um defeito. O estado de um programa ou, mais precisamente, da execução de um programa em determinado instante, é dado pelo valor da memória (ou das variáveis do programa) e do apontador de instruções. A existência de um defeito pode ocasionar a ocorrência de um "erro" (*error*) durante uma execução do programa, que se caracteriza por um estado inconsistente ou inesperado. Tal estado pode levar a uma "falha" (*failure*), ou seja, pode fazer com que o resultado produzido pela execução seja diferente do resultado esperado.

A Figura 1 a seguir demonstra como são empregados os termos citados acima em uma sequência cronológica da forma correta:

Figura 1 – Relação entre engano, defeito, erro e falha



Fonte: Nagakawa (2016)

2.2 Teste de software

Para Neto (2015, p. 54)

> Teste de software é o processo de execução de um produto para determinar se ele atingiu suas especificações e funcionou corretamente no ambiente para qual foi projetado. O seu objetivo é revelar falhas em um produto, para que as causas dessas falhas sejam identificadas e possam ser corrigidas pela equipe de desenvolvimento antes da entrega final. Por conta dessa característica das atividades de teste, dizemos que sua natureza é "destrutiva", e não "construtiva", pois visa o aumento da confiança do produto através da exposição de seus problemas, porém antes de sua entrega ao usuário final.

Ainda segundo Neto (2015, p. 54)

> O conceito de teste de software pode ser compreendido através de uma visão intuitiva ou mesmo de uma maneira formal. Existem atualmente várias definições para esse conceito. De uma forma simples, testar um software significa verificar através de uma execução controlada se o seu comportamento ocorre de acordo com o especificado. O objetivo principal desta tarefa é revelar o maior número de falhas dispondo do mínimo de esforço, ou seja, mostrar aos que desenvolvem se os resultados estão ou não de acordo com os padrões estabelecidos.

2.3 Fases da atividade de teste

Segundo Maldonado *et al.* (2007, p. 3), "a atividade de teste é complexa. São diversos os fatores podem colaborar para a ocorrência de erros."

Por conta das distintas naturezas dos erros que podem ocorrer, os testes são divididos em fases em fases que seguem um padrão de grandeza de escopo:

- Testes unitários.
- Testes de integração.
- Testes de sistemas.

Os testes unitários ocorrem no mais baixo nível de um sistema em suas menores unidades, podendo ser elas métodos, classes, funções e procedimentos. O código é testado conforme a sua construção e integrado gradualmente. Com essa estratégia, torna-se muito mais fácil isolar e identificar a razão de uma falha no sistema. (PATTON, 2005).

Os testes de integração, que devem ser realizados após os testes unitários, têm como foco verificar se as unidades trabalham em conjunto de forma correta. Quando uma falha é encontrada no nível unitário, ela pode ser atribuída apenas a sua unidade, porém quando é encontrada no nível de integração, é possível dizer que está atrelada à interação entre as unidades. (PATTON, 2005).

Os testes de sistemas ocorrem após a verificação das integrações, quando o sistema está completo e com todas as partes integradas. O objetivo é verificar se as funcionalidades

especificadas durante a fase de planejamento foram corretamente implementadas. (MALDONADO, 2007).

2.4 Dados de teste e casos de teste

Segundo Maldonado *et al.* (2007), o dado de teste é um valor de um determinado conjunto de entrada para um software, e o caso de teste é um par formado por esse valor de entrada e seu respectivo resultado esperado após ser processado pelo programa.

2.5 Testes funcionais

O teste funcional, também conhecido como teste de caixa preta, segundo Jorgensen (1995) é baseado na inserção de valores de um domínio de entrada e na análise dos valores de um domínio de saída, sem que o conteúdo do software seja conhecido.

O fato de não se ter visão e conhecimento do interior deste software é que define a relação com a caixa preta.

2.6 Testes estruturais

O teste estrutural, também conhecido como teste de caixa branca, para Jorgensen (1995), é uma abordagem fundamental que se consiste em visualizar o "interior" de um software, para que seja possível identificar os casos de teste baseados nas funções implementadas.

Esse tipo de teste está associado às fases de testes unitários e testes de integração, pois seguem o princípio de verificar internamente os elementos menores do sistema.

A relação com a caixa branca se dá pelos responsáveis terem conhecimento interno do software, em contraste com os testes de caixa preta, que conforme já mencionado, não tem essa visualização.

2.7 Testes de regressão

Segundo Jorgensen (1995), o objetivo dos testes de regressão é garantir que as funções que estavam corretas antes de uma atualização ao sistema, permaneceram funcionando após a adição de uma nova atualização de código.

O teste de regressão costuma ser associado a manutenção, portanto, em projetos que sejam realizadas atualizações constantes, essa técnica é bastante utilizada. (JORGENSEN, 1995).

2.8 Testes de usuário

Segundo Sommerville (2011), os testes de usuário são realizados com usuários ou clientes de um software para que eles possam realizar entradas e dar sua opinião sobre a utilização dele, e são essenciais pois pode haver divergências de comportamento entre o ambiente artificial utilizado pelos desenvolvedores e o ambiente de produção utilizado pelos usuários.

2.9 Testes de estresse

Para Patton (2005), o teste de estresse consiste em expor um software em um ambiente de funcionamento que tenha condições abaixo do ideal, para que assim se possa descobrir as suas dependências mínimas.

2.10 Testes automatizados

Como citado anteriormente, a estrutura de um teste no geral consiste em realizar uma inserção em um sistema e verificar se o resultado gerado condiz com o resultado esperado. Porém, para cada modificação realizada no sistema, é necessário realizar testes de regressão para verificar o funcionamento dos componentes que já existiam. Esse processo repetitivo pode levar muito tempo e para isso existem as ferramentas de automação de testes.

Segundo Patton (2005, p. 232) os atributos que ferramentas de automação de teste são:

- Velocidade: o teste automatizado, leva um tempo para configuração inicial, porém, à médio-longo prazo são mais rápido do que realizar testes manuais.
- Eficiência: por trazerem velocidade, sobra tempo para realizar outras tarefas.
- Precisão: após a execução de uma grande quantidade de testes, o responsável por estes pode vir a cometer erros, diminuindo a precisão dos testes.
- Redução de recursos: com as ferramentas de automação é possível simular cenários ideias para a realização de testes e com isso reduzir as demandas de recursos.
- Simulação: é possível simular cenários que seriam difíceis de reproduzir manualmente e expor o sistema que se deseja testar a eles.
- Implacabilidade: ferramentas de automação de testes não desistem e não ficam cansadas.

3 METODOLÓGIAS DE ESTUDO

3.1 Pesquisas bibliográficas

A pesquisa bibliográfica para Sousa et al. (2021, p. 66):

> É o levantamento ou revisão de obras publicadas sobre a teoria que irá direcionar o trabalho científico, o que necessita uma dedicação, estudo e análise pelo pesquisador que irá executar o trabalho científico, e tem como objetivo reunir e analisar texto publicados, para apoiar o trabalho científico.

3.2 Estudos de casos

O estudo de caso para Menezes (s.d.) "é uma estratégia de pesquisa científica que analisa um fenômeno atual em seu contexto real e as variáveis que o influenciam. Trata-se de um estudo intensivo e sistemático sobre uma instituição, comunidade ou indivíduo que permite examinar fenômenos complexos."

4 RESULTADOS

4.1 Testes de unidades automatizados com o DUnitX

O DUnitX é um framework de testes voltado para Delphi. Nele é possível criar testes unitários que serão responsáveis por executar uma função com um conjunto de entradas e verificar se as saídas processadas por essa função correspondem ao que era esperado.

Na figura 2 é demonstrada uma classe de venda e uma classe de itens desta venda.

Figura 2 – Exemplo de classes de venda e item da venda

```
unit VendaClass;

interface
  uses System.Generics.Collections;

type
  TItemVenda = class
  private
    FValor  : Real;
    FCodigo : Integer;
  public
    property Codigo : Integer read FCodigo write FCodigo;
    property Valor  : Real    read FValor  write FValor;
  end;

  TVenda = class
  private
    FItems : TObjectList<TItemVenda>;
  public
    constructor Create;
    destructor Destroy;

    procedure Iniciar;
    procedure LancarItem(const ACodigo: Integer; AValor: Real);
    procedure Finalizar;
  end;

implementation

constructor TVenda.Create;
begin
  FItems := TObjectList<TItemVenda>.Create;
end;

destructor TVenda.Destroy;
begin
  FItems.Destroy;
end;

procedure TVenda.Finalizar;
begin

end;

procedure TVenda.Iniciar;
begin

end;

procedure TVenda.LancarItem(const ACodigo: Integer; AValor: Real);
begin

end;

end.
```

Fonte: Paulino (2016)

O objetivo do teste unitário que será demonstrado na Figura 3 é verificar se caso uma venda tente ser finalizada sem nenhum item, irá retornar uma mensagem na tela para o usuário:

Figura 3 – Exemplo de teste unitário na prática

```
type
  [TestFixture]
  TVendaTest = class(TObject)
  private
    eVenda: TVenda;
  public
    [Setup]
    procedure Setup;
    [TearDown]
    procedure TearDown;
    [Test]
    procedure FinalizarVendaSemItensLancados;
  end;

implementation

procedure TVendaTest.Setup;
begin
  eVenda := TVenda.Create;
end;

procedure TVendaTest.TearDown;
begin
  eVenda.Free;
end;

procedure TVendaTest.FinalizarVendaSemItensLancados;
begin
  Assert.WillRaiseWithMessage(procedure
                              begin
                                eVenda.Iniciar;
                                eVenda.Finalizar;
                              end ,nil,'É necessário pelo menos um item lançado para finalizar a venda');
end;
```

Fonte: Paulino (2016)

Como é possível notar na Figura 3, o teste unitário executa as funções de iniciar e finalizar da venda sem inserir nenhum item, para que assim possa verificar se a classe da mesma trata este problema enviando uma mensagem para o usuário.

Estes testes podem ser executados quantas vezes forem necessárias, tornando muito mais rápida a manutenção nas funções de um software e a realização dos testes de regressão.

4.2 Automação de testes de interface com o Selenium

Além de ferramentas de automação para testes a nível de desenvolvimento, também existem ferramentas de automação para testes de usuário, que simulam o comportamento humano afim de encontrar problemas menos óbvios, e uma dessas ferramentas é o Selenium.

Segundo Guedes (2020, n.p.):

> Selenium é um conjunto de ferramentas de código aberto multiplataforma, usado para testar aplicações web pelo browser de forma automatizada. Ele executa testes de funcionalidades da aplicação web e testes de compatibilidade entre browser e plataformas diferentes. O Selenium suporta diversas linguagens de programação, como por exemplo C#, Java e Python, e vários navegadores web como o Chrome e o Firefox.

4.3 Consequências de não realizar testes de software

Como mencionado anteriormente, concluir o processo de desenvolvimento de um software e colocá-lo em ambiente de produção sem realizar os devidos testes antes, pode gerar graves problemas e e m alguns casos, esses problemas podem ser catastróficos.

Segundo Baraniuk (2015), em seu artigo para a BBC News, em 4 de junho de 1996, o foguete Ariane 5 da Agência Espacial Europeia teria seu primeiro voo não tripulado com o objetivo de levar quatro satélites científicos ao espaço. Porém, por conta de um simples erro gerado por um "estouro de inteiros", onde um número não pode ser armazenado corretamente por conta de ser grande demais para suas especificações de armazenamento, o foguete explodiu no ar 39 segundos após sua decolagem, trazendo um prejuízo de 370 milhões de dólares.

Em uma investigação realizada foi descoberto que em um processo utilizado para seu antecessor, o foguete Ariane 4, foi capturado uma medida de velocidade inesperada por conta de o novo foguete ser mais rápido e assim, seu software não conseguiu lidar com esse número tão grande. (BARANIUK, 2015).

4.4 Retorno sobre investimento dos testes de software

Em todas as áreas de negócios, é de grande importância considerar e analisar o retorno potencial dos investimentos em requisitos para a realização dos testes de software.

Segundo Cerqueira (2013, p. 21)

> Assim como os investimentos em ferramentas, tecnologia e pessoal, os investimentos em qualidade também são uma decisão importante para o negócio, visto que os investimentos adequados em qualidade podem maximizar o lucro das organizações desenvolvedoras de softwares. [...] O ideal é que a detecção dos defeitos e a correção deles sejam realizadas antes que o software seja repassado ao usuário final, onde os custos com reparação de defeitos tornam-se muito mais altos.

Esses custos de manutenção pós-lançamento se tornam altos à medida que impossibilitam o usuário de utilizar o software da maneira correta e trazem resultados diferentes do esperado. (CERQUEIRA, 2013).

5 CONCLUSÃO

Do ponto de vista das empresas, a qualidade é um aspecto muito importante quando se fala em entregar um produto para um cliente. Se uma medida tomada traz maior qualidade

para o consumidor deste produto, isso é também muito positivo para quem o produziu, pois isso agrega à imagem da empresa, fideliza clientes e até mesmo atrai mais clientes.

No desenvolvimento de softwares não é diferente. A prevenção e manutenção de erros afeta diretamente a satisfação do usuário com a utilização destas ferramentas, e isso pode trazer uma maior lucratividade e até mesmo evitar situações em que as empresas responsáveis pelo desenvolvimento desses softwares precisem lidar com problemas legais.

Dessa forma, os testes de software aplicados corretamente podem ser vistos como o meio para avaliar se o produto está maduro o suficiente para competir no mercado através de um ponto de vista prático, após essa avaliação eles trazem uma oportunidade de reparar e melhorar o processo atual e como resultado, trazem excelência para o usuário deste produto.

Todavia é importante notar que realizar a aplicação de testes em um software sem ter conhecimento sobre os conceitos e as metodologias de teste, pode não trazer qualidade ao seu produto, tornando assim, o investimento em mão de obra e ferramentas para a realização do processo de teste, um completo desperdício de tempo e dinheiro.

REFERÊNCIAS

BARANIUK, C. **As falhas numéricas que podem causar desastres.** 2015. Disponível em: http://www.bbc.com/portuguese/noticias/2015/05/150513_vert_fut_bug_digital_ml. Acesso em: 11 set. 2022.

CERQUEIRA, D. V. Unb. **Aplicação de metodologia de análise de retorno sobre investimento no contexto do Centro de Qualidade e Testes de Software.** Brasília, jan. 2014. Disponível em: https://bdm.unb.br/handle/10483/7002. Acesso em: 17 set. 2022.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução ao teste de software.** Elsevier, 2007.

GUEDES, M. **O que é Selenium.** 2020. Disponível em: https://www.treinaweb.com.br/blog/o-que-e-selenium Acesso em: 20 nov. 2023.

JORGENSEN, P. **Software Testing: a craftsman's approach.** 1ª Edição, Boca Raton, CRC Press, 1995.

MENEZES, P. Significados. **Estudo de Caso.** Disponível em: https://www.significados.com.br/estudo-de-caso. Acesso em: 15 set. 2022.

NAGAKAWA, E. Y. **Teste de Software.** São Paulo, jan. 2016. Disponível em: https://edisciplinas.usp.br/pluginfile.php/1196441/mod_resource/content/1/Aula09_TesteSoftware_Parte1.pdf . Acesso em: 06 set. 2022.

NETO, A. C. D. Research Gate. **Introdução a Teste de Software.** Manaus: UFAM, 2015. Disponível em: https://www.researchgate.net/profile/Arilo-Neto/publication/266356473_Introducao_a_Teste_de_Software/links/5554ee6408ae6fd2d821ba3a/Introducao-a-Teste-de-Software.pdf. Acesso em: 08 set. 2022.

PATTON, R. **Software Testing.** 2ª Edição, Indianapolis, Sans Pub, 2005.

PAULINO, C. E. **TDD – Desenvolvimento Orientado a Testes com Delphi.** 2016. Disponível em: https://delphicleancode.wordpress.com/2016/04/26/tdd-desenvolvimento-orientado-a-testes-com-delphi/. Acesso em: 29 ago. 2023.

SOMMERVILLE, I. **Engenharia de Software.** 9ª Edição, São Paulo, Pearson Education do Brasil, 2011.

SOUSA, A. S.; OLIVEIRA, G. S.; ALVES, L. H. Fucamp. **A pesquisa bibliográfica: Princípios e Fundamentos.** Monte Carmelo, mar. 2021. Disponível em: https://revistas.fucamp.edu.br/index.php/cadernos/article/view/2336. Acesso em: 15 set. 2022.