A IMPORTÂNCIA DO TESTE PARA A QUALIDADE E CUSTO DO SOFTWARE

Angélica Cristina Lombas de Campos[1], Débora Pelicano Diniz[1]

[1] Faculdade de Tecnologia de FATEC Ribeirão Preto (FATEC)
Ribeirão Preto, SP – Brasil

angelica_lombas@hotmail.com, debora.diniz2@fatec.sp.gov.br

Resumo. *O teste é um grande aliado da qualidade e do custo do software, pois, com suas técnicas que podem ser aplicadas desde o planejamento do software até o seu desenvolvimento completo, o teste tem como seu principal objetivo encontrar defeitos o quanto antes, evitando assim um custo elevado para correção e garantindo uma maior qualidade ao software. O objetivo deste artigo é demonstrar quais são as principais técnicas de teste de software, os modelos de ciclo de vida de um software e suas relações com o teste, a importância do teste para a qualidade do software e o custo de um defeito em cada etapa do ciclo de desenvolvimento.*

Abstract. *The test is a great ally of the quality and cost of the software, with its techniques that can be applied from the planning of the software until its complete development, the test has as its main objective to find defects as soon as possible, thus avoiding a high cost for correction and guaranteeing a higher quality to the software. The purpose of this article is to demonstrate what are the main software testing techniques, the software life cycle models and their relationship to testing, the importance of testing for software quality and the cost of a defect at each stage development cycle.*

1. Introdução

Atualmente para levar a vida moderna, as pessoas utilizam softwares para quase tudo, tanto na vida pessoal quanto para os negócios, sendo difícil encontrar uma pessoa que não tenha um smartphone com diversos aplicativos ou uma empresa que não tenha um software de contabilidade, controle de estoque ou emissor de nota fiscal eletrônica. E por trás de tudo isso, existem empresas especialistas no desenvolvimento de softwares, que, assim como em qualquer outra área, buscam que seus produtos tenham a máxima qualidade com o menor custo de produção.

Porém, na prática, muitas empresas desenvolvedoras de software perdem bilhões de dólares, por ano, com softwares com defeito (PRESSMAN, 2011). Segundo Pressman (2011), na década de 1990, as empresas e o governo começaram a ficar muito preocupados com o fato de que uma falha grave de software poderia inutilizar importantes infraestruturas, aumentando o custo em dezenas de bilhões.

Muitos fatores podem resultar em um defeito no software, porém, a maioria deles tem uma única origem: erro humano. Como o planejamento e a construção de um software dependem principalmente das habilidades, interpretação e execução de um ser humano, erros podem ser cometidos e resultarem em defeitos (DELAMARO; MALDONADO; JINO, 2007).

Para minimizar as chances de um defeito chegar ao usuário final, um dos principais aliados das empresas desenvolvedoras de software, apontado pelos autores pesquisados, é o teste. O teste, por meio de técnicas que serão explanadas a seguir, consegue identificar defeitos existentes no software antes que eles cheguem ao usuário final, evitando assim um custo elevado para correção e garantindo uma maior qualidade ao software.

Neste contexto, o presente artigo tem como objetivo demonstrar quais são as principais técnicas de teste de software, os modelos de ciclo de vida de um software e suas relações com o teste, a importância do teste para a qualidade do software e o custo de um defeito em cada etapa do ciclo de desenvolvimento.

De forma a atingir o objetivo proposto, foi utilizada a base de dados online Google Acadêmico, para a busca de livros e artigos científicos, e o *Syllabus Certified Tester* do ISTQB - *International Software Testing Qualifications Board*, principal órgão mundial na certificação de competências em teste de software.

Foi definido como critério de inclusão de livros e artigos científicos, publicações realizadas após o ano de 2000, pois, após levantamento inicial, foi identificada uma escassez de publicações sobre o assunto deste artigo, antes deste período.

Inicialmente foi realizado o levantamento de publicações com assuntos relacionados com teste de software, na sequência ciclo de vida de software, qualidade de software e custo do software.

Após o levantamento, foram selecionadas as publicações com maior detalhamento de processos e condensados para a criação do artigo aqui apresentado.

2. Teste e suas Técnicas

Executar o software ao final do desenvolvimento e verificar se está de acordo com o esperado é sem dúvida a principal atividade do processo de teste, porém, diferente do que muitos acreditam, não é a única. Testar é um processo com várias atividades.

> Testar: processo que consiste em todas as atividades do ciclo de vida, tanto estáticas quanto dinâmicas, voltadas para o planejamento, preparação e avaliação de produtos de software e produtos de trabalho relacionados a fim de determinar se elas satisfazem os requisitos especificados e demonstrar que estão aptas para sua finalidade e para a detecção de defeitos. (ISTQB, 2018b, p.27).

Dentre as atividades do processo de teste pode-se destacar: o planejamento, projeto de casos de teste, execução e análise (DELAMARO; MALDONADO; JINO, 2007).

Para a melhor execução destas atividades, existem algumas técnicas de testes que são comumente utilizadas, essas técnicas podem ser classificadas como técnica de: caixa preta, caixa branca ou baseadas em experiência (ISTQB, 2018a).

2.1. Técnica Teste de Caixa Preta

Teste de caixa-preta, também conhecido como teste funcional, consiste em uma técnica que não analisa como o software foi codificado, quem está testando tem acesso somente ao que o software se propõe a fazer, ou seja, aos requisitos funcionais. Na especificação desta técnica, são identificados os dados de entrada e quais serão as saídas geradas

(SANTOS; OLIVEIRA, 2017).

Segundo Pressman (2011), nesta técnica os testes são feitos para responder as seguintes questões:

a) Como a validade funcional é testada?
b) Como o comportamento e o desempenho do sistema são testados?
c) Que classes de entrada farão bons casos de teste?
d) O sistema é particularmente sensível a certos valores de entrada?
e) Como as fronteiras de uma classe de dados é isolada?
f) Que taxas e volumes de dados o sistema pode tolerar?
g) Que efeito combinações específicas de dados terão sobre a operação do sistema?

Dentre as muitas técnicas de teste derivadas da técnica de caixa-preta, pode-se destacar a técnica de Particionamento de Equivalência, que consiste em dividir o domínio de entrada de um programa em classes de dados a partir das quais podem ser criados casos de teste, casos esses, que conseguiriam encontrar sozinhos uma classe de erro (PRESSMAN, 2011).

2.2. Técnica Teste de Caixa Branca

Teste de caixa-branca, também conhecido como teste estrutural ou teste orientado a lógica, ganhou este nome, pois, é responsável por verificar os comportamentos internos do software, ou seja, é possível ver “dentro do software” (ASSIS, 2013).

A técnica garante que todas as opções independentes de um módulo foram executadas pelo menos uma vez, cobriram todas as decisões lógicas nos seus estados verdadeiro e falso, executaram todos os ciclos em seus limites e dentro de suas fronteiras operacionais, e exercitaram estruturas de dados internas para assegurar a sua validade (PRESSMAN, 2011).

Existem várias técnicas de teste baseadas na técnica de teste caixa-branca, das quais pode-se destacar a técnica de teste Caminho Básico, que consiste em permitir derivar uma medida da complexidade lógica de um projeto procedimental e usar essa medida como guia para definir um conjunto base de caminhos de execução (PRESSMAN, 2011).

2.3. Técnica Teste Baseado em Experiência

Teste baseado em experiência, consiste em uma técnica que se baseia na experiência da pessoa que está realizando os testes, ou seja, os casos de teste são derivados da habilidade e intuição da pessoa que está testando, em relação ao que será testado. Essa técnica normalmente encontra problemas que não seriam encontrados nas outras duas técnicas (ISTQB, 2018a).

Não existem tantas técnicas derivadas desta técnica se comparado com as outras duas apresentadas, porém, pode-se destacar uma técnica extremamente usada por profissionais da área de teste, principalmente quando se tem um período de tempo pequeno para a realização dos testes, que é a técnica Teste Exploratório Baseado em Sessão, que consiste em determinar uma janela de tempo e, dentro dela, ir realizando

testes não pré-definidos (ISTQB, 2018a).

3. Modelo de Ciclo de Vida de Desenvolvimento X Teste

Segundo o ISTQB (2018a), um modelo de ciclo de vida de desenvolvimento de software representa, de forma lógica e cronológica, as atividades realizadas em cada etapa do projeto de desenvolvimento.

Existem vários modelos de ciclo de vida de desenvolvimento de software, mas a maioria realiza o mesmo conjunto de atividades metodológicas genéricas: comunicação, planejamento, modelagem, construção e emprego, como está apresentado na Figura 1 (PRESSMAN, 2011).

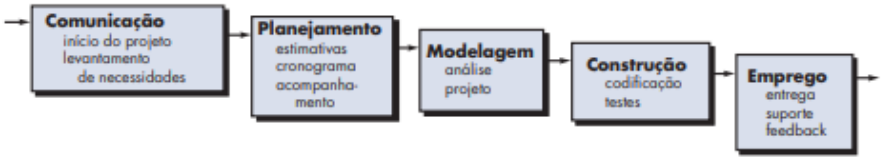

Figura 1. Conjunto de Atividades
Fonte: Pressman (2011)

Saber qual o modelo de ciclo de vida de software será utilizado pela equipe de desenvolvimento é muito importante para quem irá realizar os testes, pois, com essas informações é possível realizar o planejamento e execução dos testes de forma mais efetiva. Os modelos mais comumente utilizados pelas empresas desenvolvedoras de software são os modelos Interativo Incremental, Sequencial e Dirigido a Teste que serão apresentados a seguir, mostrando a relação de cada um com o teste.

3.1. Modelo de Desenvolvimento Iterativo e Incremental

O Modelo de Desenvolvimento Iterativo e Incremental, combina elementos dos fluxos de processos lineares e paralelos, envolvendo o estabelecimento de requisitos, modelagem, construção e teste de um sistema em partes, chamadas de incremento (Figura 2), que podem variar de tamanho. Todas as etapas do incremento são realizadas em uma série de ciclos, chamados de iteração (Figura 2). A cada iteração, portanto, um incremento é liberado até que o software final seja entregue (ISTQB, 2018a).

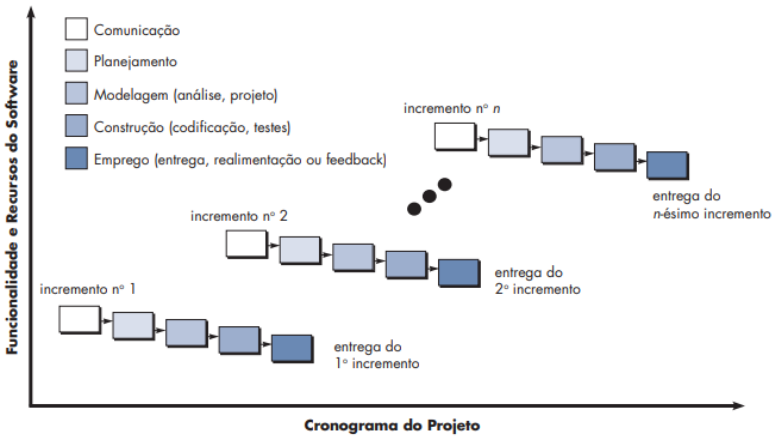

Figura 2. Modelo Incremental
Fonte: Pressman (2011)

O foco do modelo incremental, portanto, é entregar um produto minimamente operacional e testado a cada iteração, para que o usuário já consiga ir usando as partes desenvolvidas do software até que a versão final, gerada após todas as iterações, seja liberada (PRESSMAN, 2011).

3.2. Modelo de Desenvolvimento Sequencial

O Modelo de Desenvolvimento Sequencial, consiste em um modelo de desenvolvimento de software no qual as atividades são realizadas sequencialmente de forma linear, ou seja, cada atividade só é iniciada quando a anterior foi finalizada (ISTQB, 2018a).

Um exemplo de um modelo de desenvolvimento baseado no modelo sequencial é o Modelo V, este modelo integra o teste ao longo do processo de desenvolvimento, portanto, conforme o desenvolvimento avança (lado esquerdo) a etapa de teste correspondente (lado direito) é refinada, podendo já encontrar possíveis problemas, e ao final do desenvolvimento são efetivamente realizados os testes, conforme é mostrado na Figura 3 (PRESSMAN, 2011).

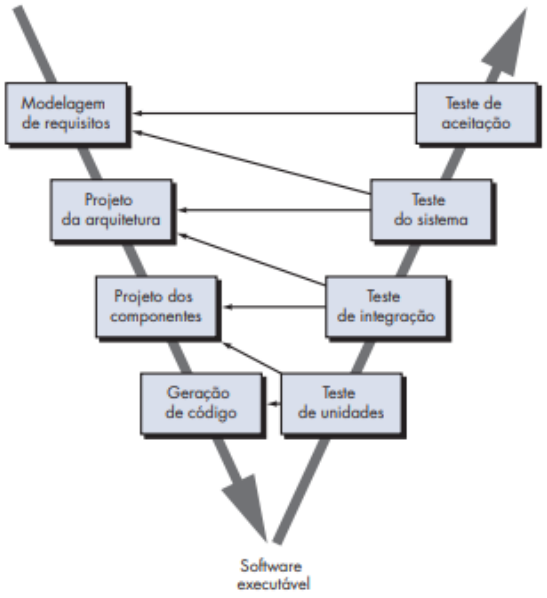

Figura 3. Modelo V
Fonte: Pressman (2011)

Este modelo é bem visto no que diz respeito a testes, por introduzi-lo desde o início do desenvolvimento, porém, para o desenvolvimento não é tão vantajoso, pois, normalmente exige meses ou anos para entregar o software aos usuários finais (ISTQB, 2018a).

3.3. Modelo de Desenvolvimento Dirigido a Testes

No Modelo de Desenvolvimento Dirigido a Testes (TDD, do inglês *Test-Driven Development*), primeiramente são escritos os testes, especificando o comportamento esperado do software, e só após é iniciado o desenvolvimento do código que deverá passar no teste escrito (BARAÚNA; HARDARDT, 2020).

Esse processo pode ser divido em algumas etapas, como mostrado na Figura 4. Essas etapas são (SOMMERVILLE, 2011, p.155):

> 1. Você começa identificando o incremento de funcionalidade necessário. Este, normalmente, deve ser pequeno e implementável em poucas linhas de código.
> 2. Você escreve um teste para essa funcionalidade e o implementa como um teste automatizado. Isso significa que o teste pode ser executado e relatará se passou ou falhou.
> 3. Você, então, executa o teste, junto com todos os outros testes implementados. Inicialmente, você não terá implementado a funcionalidade, logo, o novo teste falhará. Isso é proposital, pois mostra que o teste acrescenta algo ao conjunto de testes.
> 4. Você, então, implementa a funcionalidade e executa novamente o teste. Isso pode envolver a refatoração do código existente para melhorá-lo e adicionar um novo código sobre o que já está lá.
> 5. Depois que todos os testes forem executados com sucesso, você caminha para implementar a próxima parte da funcionalidade.

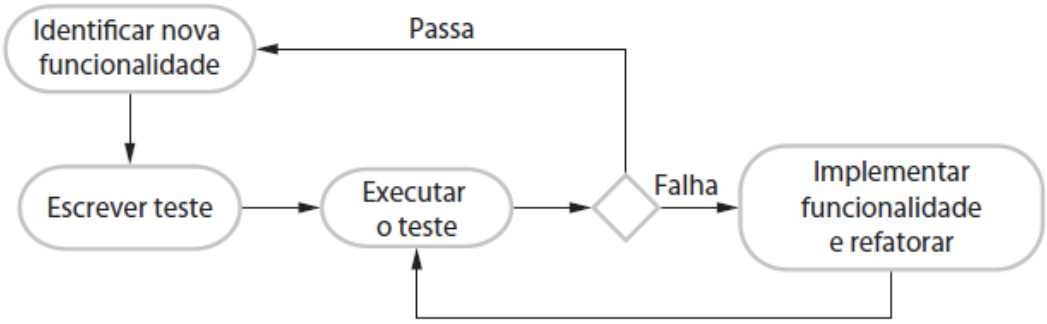

Figura 4. Desenvolvimento dirigido a testes
Fonte: Sommerville (2011)

Esse modelo de desenvolvimento é uma das práticas essenciais da metodologia ágil de desenvolvimento *Extreme Programming* (XP) e hoje é visto como uma das bases para o desenvolvimento de software com qualidade (BARAÚNA; HARDARDT, 2020).

4. Relação entre Teste e Qualidade de Software

Quando uma empresa de desenvolvimento de sistemas se preocupa em realizar testes de software, significa que ela tem uma preocupação com a qualidade de seu produto, pois, os testes são um dos principais processos para garantir a qualidade de um software. Segundo Rios e Moreira Filho (2013), considera-se um software com qualidade quando:

a) o número e a severidade dos defeitos residuais do processo de testes são aceitáveis pela organização;

b) o software é entregue dentro do prazo e custo, atende aos requisitos e/ou às expectativas;

c) foi construído de tal maneira que possa ser mantido de forma eficiente após sua implantação.

Na mesma linha de pensamento, Bartié (2002) entende que quanto mais cenários de testes simulados, maior o nível de validação que se obtém do produto, caracterizando maior nível de qualidade do software desenvolvido.

4.1. Custo de um Defeito

Segundo Patton (2005), o custo da correção de um defeito cresce de acordo com seu avanço nas etapas do ciclo de vida do software. Como pode ser observado pela imagem apresentada na Figura 5, um defeito pode custar até 100 vezes mais se encontrado na etapa de liberação do software ao usuário em comparação com um defeito encontrado na etapa de design.

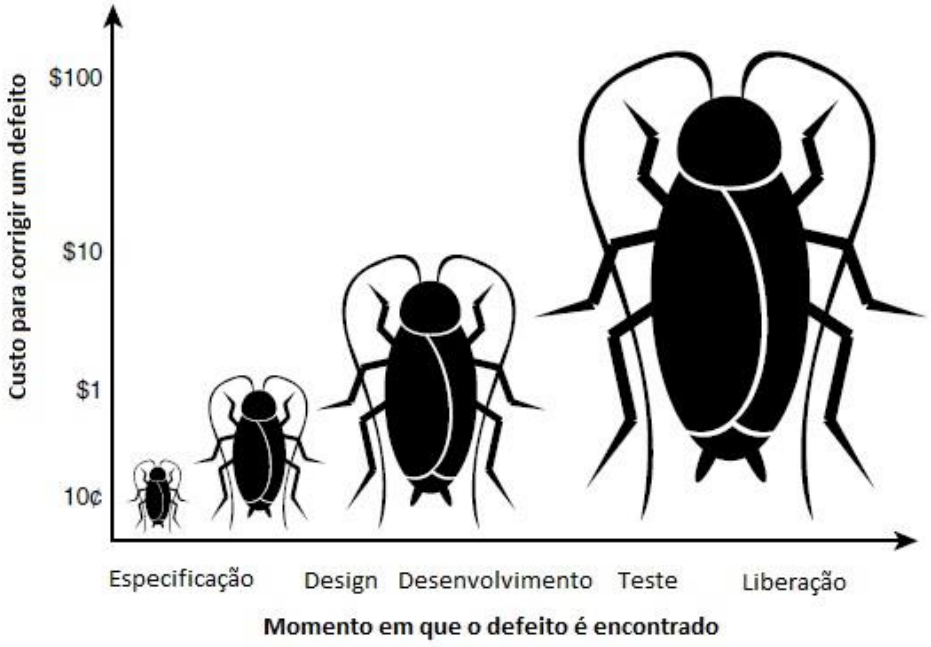

Figura 5. Aumento do custo para corrigir um defeito
Fonte: Adaptado de Patton (2005)

Esse aumento se deve ao retrabalho da equipe de desenvolvimento, pois, quanto mais etapas o defeito avança, mais etapas o defeito terá que retroceder para ser corrigido, fazendo assim, com que mais horas de trabalho da equipe sejam gastas no desenvolvimento do software.

4.2. Custo do Defeito X Custo da Equipe de Teste

Muitas empresas justificam a falta de teste devido ao custo de se ter uma equipe de teste, porém, como está representado na Tabela 1, o custo de se ter uma equipe de teste pode ser menor que o custo da correção do software quando não houve teste desde o começo do desenvolvimento.

Na Tabela1 estão apresentados dados que ilustram um projeto com 1.000 defeitos em dois cenários, um em que não foram realizados testes por uma equipe de teste, portanto, não foram encontrados defeitos pela mesma, e outro no qual foram realizados testes por uma equipe de teste, e foram encontrados 350 defeitos. Comparando os dois cenários, no cenário no qual não houve envolvimento da equipe de teste teve o custo de qualidade (custo das correções mais a equipe de teste) 38,45% maior que no cenário no qual houve envolvimento da equipe de teste.

Tabela 1. Processo com teste X Processo sem teste

	PORCESSO SEM ESTRUTURA FORMAL PARA TESTES	PROCESSO COM ESTRUTURA FORMAL PARA TESTES (MANUAL)
HORAS DO PROJETO	10.000	10.000
ERROS ENCONTRADOS	1.000	1.000
INVESTIMENTO EM TESTES		
Pessoal	0,00	90.000,00
Infraestrutura	0,00	16.000,00
TOTAL INVESTIMENTO	0,00	106.000,00
DESENVOLVIMENTO/TESTES		
Defeitos encontrados	250	250
Custo de Correção	2.500,00	2.500,00
TESTE(*)		
Defeitos encontrados	0	350
Custo de correção	0,00	35.000,00
MANUTENÇÃO(**)		
Defeitos encontrados	750	400
Custo de Correção	750.000,00	400.000,00
CUSTO DA QUALIDADE	752.500,00	543.500,00
RETORNO		
(*) Testes realizados pela equipe de testes		
(**) Defeitos encontrados após a implantação afetando os usários		

Fonte: Adaptado de Rios e Moreira Filho (2013)

Conclusão

Como foi comentado, os defeitos são causados pincipalmente por erro humano, portanto, é inevitável que erros ocorram durante o desenvolvimento de um software, porém se forem realizados os devidos testes, o risco desses defeitos chegarem ao usuário final é muito menor, diminuindo assim o custo com a correção desses possíveis defeitos.

Outro ponto importante apresentado é que o custo de manter uma equipe de testes é menor que o custo de corrigir defeitos encontrados após a liberação do software ao usuário final, portanto, financeiramente é mais viável para uma empresa desenvolvedora de software manter uma equipe de testes, que arcar com o custo da correção dos possíveis defeitos encontrados após a liberação

Também foi possível observar que o teste é um dos principais processos para garantir a qualidade de um software, pois quanto mais cenários de testes simulado, maior o nível de qualidade do software.

Sendo assim, foi possível concluir que o teste é muito importante para a qualidade e custo do software.

Referências

ASSIS, Marcio Roberto Miranda. Informática para Concursos Públicos de Informática. São Paulo: Novatec, 2013.

BARAÚNA, Hugo; HARDARDT, Philippe. TDD e BDD na prática: Construa aplicações Ruby usando RSpec e Cucumber. São Paulo: Casa do Código, 2020.

BARTIÉ, Alexandre. Garantia da Qualidade de Software: adquirindo maturidade organizacional, 5. ed. Rio de Janeiro: Elsevier, 2002.

DELAMARO, Márcio Eduardo; MALDONADO, José Carlos; JINO, Mario. Introdução ao Teste de Software. Rio de Janeiro: Elsevier, 2007.

ISTQB. *Certified Tester: foundation level syllabus. Foundation Level Syllabus*. 2018a. Disponível em: <https://www.bstqb.org.br/uploads/syllabus/syllabus_ctfl_2018br.pdf>. Acesso em: 30 ago. 2020.

ISTQB. Glossário de Termos. 2018b. Disponível em: <https://www.bstqb.org.br/uploads/glossario/glossario_ctfl_3.2br.pdf>. Acesso em 23 de ago. 2020.

PATTON, Ron. *Software Testing*, 2. ed. Indianápolis: Sams Publishing, 2005.

PRESSMAN. Roger S.. Engenharia de Software: uma abordagem profissional, 7. ed. Porto Alegre: AMGH, 2011.

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

SANTOS, Luiz Diego Vidal; OLIVEIRA, Catuxe Varjão de Santana. Introdução à Garantia de Qualidade de Software. Timburi: Editora Cia do Ebook, 2017.

SOMMERVILLE, Ian. Engenharia de Software. 9. ed. São Paulo: Pearson, 2011.