UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ALFREDO MARTINS POSSIDONIO

DESENVOLVIMENTO DE UM MODELO DE DERIVAÇÃO FUNCIONAL PARA VALIDAÇÃO DE SOFTWARE NA FASE DE INTEGRAÇÃO

CRICIÚMA
2019

ALFREDO MARTINS POSSIDONIO

DESENVOLVIMENTO DE UM MODELO DE DERIVAÇÃO FUNCIONAL PARA VALIDAÇÃO DE SOFTWARE NA FASE DE INTEGRAÇÃO

Trabalho de Conclusão de Curso, apresentado para obtenção do grau de Bacharel no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientador: Prof. Me. Gustavo Bisognin.

CRICIÚMA

2019

ALFREDO MARTINS POSSIDONIO

DESENVOLVIMENTO DE UM MODELO DE DERIVAÇÃO FUNCIONAL PARA VALIDAÇÃO DE SOFTWARE NA FASE DE INTEGRAÇÃO

Trabalho de Conclusão de Curso aprovado pela Banca Examinadora para obtenção do Grau de Bacharel no Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC, com Linha de Pesquisa em Engenharia de Software.

Criciúma, 24 de junho de 2019.

BANCA EXAMINADORA

Prof. Gustavo Bisognin - Mestre- (UNESC) - Orientador

Prof. Fabricio Giordani -Esp - (UNESC)

Prof. Ana Cláudia Garcia Barbosa - Mestre - (UNESC)

Dedico este trabalho a meus pais e avós, que de sua maneira simples, sempre procuraram mostrar a luz que o conhecimento emana. Dedico também à minha irmã Taise que com seu exemplo de luta e superação me fez ver de forma profunda que os problemas só têm o tamanho que dermos a eles.

AGRADECIMENTOS

Agradeço a minha família, aos funcionários da UNESC, pela ajuda ao longo do desenvolvimento do trabalho.

“Eu não tenho ídolos. Tenho admiração por trabalho e dedicação.”

Ayrton Senna da Silva.

RESUMO

A procura pela qualidade das invenções de software se concretizou como uma obrigação. Com isso, profissionais da engenharia de software buscam a melhora de procedimentos e produtos. Através desse contexto, o presente trabalho visa apresentar o uso da derivação de testes funcionais como uma maneira para ampliar os ajustes funcionaisem testes de softwares que sofrem várias manutenções. O trabalho começa com estudos sobre conceitos da engenharia de software, como por exemplo, a qualidade de software, seus modelos, métodos e normatizações, bem como teorias voltadas à manutenção e manutenabilidade do software. Na sequência os testes funcionais são avaliados, explorando as técnicas de teste caixa-preta e caixa-branca, com detalhando nas fases, tipos de testes e critérios de derivação do teste caixa-preta. Nesse momento a fase de testes de regressão também recebe estudos que comprovam sua a importância para validar requisitos funcionais após uma manutenção de software. Após isso o trabalho enfoca os conceitos da derivação de testes, suas ferramentas, riscos de implante e fatores de sucesso. No campo da prática, empregando como critério de derivação ainformação e conhecimento do especialista, o trabalho contempla a criação de uma base de testes e uma matriz de planejamento dos testes, esta última contendo uma lista de requisitos funcionais que facilitam a priorização e seleção de casos de testes condizentes com determinada manutenção.Finalmente, com base em conjecturas de manutenções levantadas para fins didáticos, as implicações são avaliadas e as checagens permitem definir se o objetivo geral do trabalho foi abordado.

Palavras-chave: Derivação. Testes Funcionais. Regressão.

ABSTRACT

The demand for the quality of software inventions has become a must. With this, software engineering professionals hunt the improvement of procedures and products. Through this context, the present work aims to present the use of the derivation of functional tests as a way to extend the functional adjustments in software tests that undergo several maintenance. The work begins with studies on software engineering concepts, such as the quality of software, its models, methods and standards, as well as theories for software maintenance and maintenance. Following the functional tests are evaluated, exploring the techniques of black-box test and white box, with details on the phases, types of tests and criteria of derivation of the black-box test. At this point the regression testing phase also receives studies that prove its importance to validate functional requirements after a software maintenance. After that the work focuses on the concepts of derivation of tests, their tools, risks of implantation and factors of success. In the field of practice, using as a criterion of derivation the information and knowledge of the specialist, the work contemplates the creation of a test base and a test planning matrix, the latter containing a list of functional requirements that facilitate the prioritization and selection of test cases consistent with certain maintenance. Finally, based on conjectures of maintenance raised for didactic purposes, the implications are evaluated and the checks allow to define if the general objective of the work was approached.

Keywords:Derivation. TestsFunctional. Regression.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
ANSI	*American National Standards Institute*
CASE	*Computer-Aided Software/System Engineering*
CenPRA	Centro de Pesquisas Renato Archer
CMMI	*Capability Maturity Model Integration*
DLL	*Dinamic-link Library*
GUI	*Graphical User Interface*
HTML	*Hypertext Markup Language*
IDE	*Integrated Development Environment*
IEC	*International Electrotechnical Commission*
IEEE	*Institute of Electrical and Electronics Engineers*
ISO	*InternationalOrganization for Standardization*
MA-MPS	Método de Avaliação-Melhoria do Processo do Software
MN-MPS	Modelo de Negócios-Melhoria do Processo do Software
MPS.br	Melhoria do Processo do Software Brasileiro
MR-MPS	Modelo de Referência-Melhoria do Processo do Software
NBR	Norma Brasileira
PBQP	Programa Brasileiro de Qualidade e Produtividade em Software
PHP	*PersonalHomePage* ou o acrônimo recursivopara*HypertextPreprocessor*
SEI	*Software EngineeringInstitute*
SOFTEX	Associação para Promoção da Excelência do Software Brasileiro

SUMÁRIO

1 INTRODUÇÃO

A busca pela qualidade no desenvolvimento de software tem recebido uma crescente atenção de institutos públicos e privados atuantes no setor (GUERRA; COLOMBO, 2009;TENÓRIOJUNIOR, 2009). Podemos observar que a engenharia de software tem progredido desde 1968, quando os primeiros estudos sobre a construção de aparelhosmostravam limitações que expandiam os custos de produção e manutenção de software (SOMMERVILLE, 2003).

No momento as associações se valem dos avanços tecnológicos para desenvolverem melhores produtos e serviços, de prática rápida e com cotações menores, porém a tecnologia também impõe desafios mais difíceis ao requerer uma conexão dos softwares a sistemas e componentes de terceiros, em alguns casos empregados no processo produtivo (CHRISSIS; KONRAD; SHRUM, 2007, tradução nossa). Outros desafios da engenharia de software são os sistemas legados, a diferença de ambientes ou componentes, bem como o fornecimento do produto em um tempo apto, acatando prazos contratuais (SOMMERVILLE, 2003).

Conforme Pressman (2006), o software possui um padrão crescente e apresentará *bugs* durante a existência, alguns devido aodesenvolvimento, pois quanto maior a complexidade, aliada a manutenções constantes, maiores serão os defeitos e a queda na qualidade do software.

O estudo da qualidade no desenvolvimento de software é intenso a tal ponto que políticas claras já foram adotadas, como a recente criação do Programa Brasileiro de Qualidade e Produtividade em Software (PBQP), feito para abranger a capacidade competitiva e produtiva, pois incentiva a aplicação das melhores práticas para o desenvolvimento de aparelhos, com uso de ferramentas, normatizações, metodologias e técnicas que visam qualidade (BRASIL, 2006; GUERRA; COLOMBO, 2009).

Neste andamento, os métodos de desenvolvimento cresceram buscando prevenir e eliminar *bugs* no produto, porém é conhecido que os

defeitos ainda ocorrem, produzindo queda na qualidade do software, o que torna necessário o emprego de técnicas que colaborem para a detecção das falhas, como por exemplo, a adoção de equipes de testes nointuito de garantir a qualidade desejada, antes que o produto se torne operacional (BURNSTEIN, 2010, tradução nossa; DELAMARO; MALDONADO; JINO, 2007).

Segundo Viccari (2009), o teste de software é uma tarefa bastante cobrada como garantia de requisitos e de qualidade, porém são tarefas manuais e periódicas.Apesar de o homem ter a capacidade cognitiva que lhe dá vantagem na seleção das melhores alterações em testes, diante da repetição de trabalhossimples, como por exemplo, a produção massiva de casos de testes, os humanos demoram na execução e são propensos a gerar defeitos (PEZZÈ; YOUNG, 2008).

Neste caminho, o gerenciamento do processo de testes requer o uso da derivação de testes de software, buscandotornar mínimo o tempo e custo de produção, com ampliação da qualidade final do produto. Todavia, a derivação de testes também tem seus transtornos, tendo em vista que é um investimento de extenso prazo, que engloba reestruturação de equipes, reorganização de processos e em alguns casos aoalcance e fundação de ferramentas. A produção de alguns conflitos também precisa ser avaliada de forma aproximada, como por exemplo, um possível aumento no tempo de manutenção (*scripts*) e no tempo de preparo de testes (RIOS; MOREIRA, 2006).

Nesta direção, esta pesquisa propõe a utilização de testes de derivação das funcionalidades do software, empregando-se critérios formais ou não formais para derivação dos casos de testes, como por exemplo, particionamento de equivalências, teste de matriz ortogonal ou o próprio conhecimento e experiência do conhecedor de testes, entre outros. Com isso, objetiva-se reduzir o número de *bugs* e o tempo total de execução de testes, garantindo maior qualidade e redução dos custos associados ao projeto de desenvolvimento de software.

1.1 OBJETIVO GERAL

Desenvolver um modelo de processo para a derivação funcional de testes de software como base para a garantia da qualidade do desenvolvimento.

1.2 OBJETIVOS ESPECÍFICOS

O presente trabalho abordará temas que envolvem os seguintes objetivos específicos:

a) conhecer as considerações de qualidade na Engenharia de Software;
b) entender o uso do modelo de derivação funcional para validação de software;
c) priorizar os casos de teste;
d) analisar os resultados da aplicação dos casos de testes;
e) criar um modelo para a derivação funcional de testes de software.

1.3 JUSTIFICATIVA

Os códigos computacionais apresentaram grandes avanços nos últimos 50 anos, progredindo da classe de algoritmos especialistas e científicos, para produtos de software comerciais. A engenharia de software seguiu esse desenvolvimento, sendo constatada com a qualidade dos aparelhos desenvolvidos (PRESSMAN, 2006).

O serviço de software se tornou tão simples ao ponto que sua existência só é percebida quando ocorre um *bug*, sendo assim, no emprego de um caixa automático que apresente a frase “Falha geral no sistema, aposte em utilizar outro caixa.”ou quando um relatório cita um software como provável causa de um acidente nas alturas. Os dois exemplos mostram à baixa qualidade em softwares.

Estes modelos comprovam o quão importante é o estudo da engenharia de software, a qual, segundo Sommerville (2003), essencialmente possui modelagens e métodos idênticos à manutenção prática e à obtenção da autoridade do produto de software. Essa visão fácil parece abordar o raio de

ação da engenharia de software, aliás um cálculo criterioso demonstra que a aplicação da abordagem de engenharia para a produção de software aplica princípios científicos, artifícios, exemplares, padrões e teorias que possibilitam gerenciar, delinear, modelar, esboçar, praticar, mensurar, avaliar, alimentar e progredir aparelhos de software, tudo isso com a fabricação econômica de softwares qualificados (PETERS; PEDRYCZ, 2000, tradução nossa).

Em uma falha de software, detectar, aproximar, definir ações para solução da mesma e provocar conservação tentando evitar que novos desacertos surjam, constituem tarefas duras, porém indispensáveis (SOMMERVILLE, 2003). Segundo apontam Pezzè e Young (2008), tanto um software quanto um automóvel são produtos excepcionais, de comportamentos alteráveis, pois poderão ser contidos a casos diversos, decretando que o teste e o cálculo da qualidade seja algo característico para cada novo modelo do produto.

Neste fluxo, analisando o panorama de crescente busca pela qualidade sem perdas aos prazos de projetos, a tarefa atual mostra-se relevante, já que os testes de derivação funcional apresentam-se viáveis para validação de software na fase de integração, levantando a qualidade desejada e reduzindo o tempo final do processo de software.

1.4 ESTRUTURA DO TRABALHO

Os assuntos relatados pelo trabalho foram colocados em 7 capítulos, o primeiro pela introdução, objetivo geral, objetivos específicos, justificativa e a estrutura do trabalho. Os demais capítulos, os assuntos abordados foram:

a) **capítulo 2, A Engenharia de Software:** explica conceitos ligados à engenharia de software, narrando a importância dos softwares e o comprometimento de se especializar os indivíduos influentes no campo. Foram apresentados os lucros adaptados pela engenharia conforme à qualidade do software e a qualidade do próprio processo produtivo. Foram tratados os modelos genéricos de produção de software, também modelos, normatizações e métricas de qualidade ampliadas por institutos nacionais e internacionais. Colocadasidéias iniciais sobre a importância de se

aplicar os testes de software na busca da qualidade dos produtos. Nos subcapítulos foram abordados temas que passaram pelaimportância de qualidade de software, modelos de qualidade, os impactos da manutenção de software juntamente com ovalor da manutenabilidade para a qualidade do software;

a) **capítulo 3, Testes de Software:**permite temas ligados ao uso de testes de software na validação da qualidade, introduzindo conceitos sobre validação ou invalidação de qualidade, confirmandooslimites da implantação do processo de testes, assim como os ganhos da sua adoção. Também são aplicados conceitos de testes de software, entre eles, as técnicas de testes, tipos de testes e suas fases de aplicação. Definições específicas, como por exemplo, critérios de derivação e suas alterações também são abordadas no capítulo. Nos subcapítulos foienfatizada a técnica de teste caixa-preta e seus critérios de derivação. As considerações da técnica caixa-branca também foram expostos, bem como os tipos de testes, para as duas técnicas. Os subcapítulos atingiram ainda o ciclo de vida dos testes de software, além de serem apresentadas as fases de testes da técnica caixa-preta;

c) **capítulo 4, Trabalhos Correlatos:** o capítulo faz uma referência introdutória sobre os objetivos de se estudar os trabalhos correlatos. Os subcapítulos detalham o resultado dos estudos realizados nos 4 trabalhos correlatos levantados na fase da pesquisa bibliográfica, os quais têm os seguintes títulos,

- Desenvolvimento de Uma Metodologia Aplicada ao Gerenciamento e Acompanhamento de Testes de Software via Web (LOPES, 2009),
- Autotec: Uma Ferramenta de Automação de Testes para Interfaces Dinâmicas (MÜNCHEN, 2010),
- Uma Metodologia para Teste de Software no Contexto da Melhoria de Processo (CRESPO et al, 2012),
- Teste de Software Automatizado: Uma Solução para Maximizar a Cobertura do Plano de Teste e Aumentar a Confiabilidade e

Qualidade em Uso do Software (CATELANI et al, 2010, tradução nossa);

d) **capítulo 5, Trabalho Proposto:**o capítulo faz uma referência ao trabalho proposto e a metodologia.

2 A ENGENHARIA DE SOFTWARE

Bauer (2006) define engenharia de software como “o entendimento e o emprego de adequados princípios de engenharia a fim de obter softwares econômicos que sejam livres de defeitos e que trabalhem corretamente em máquinas reais”. Enquanto que, para o *InstituteofElectricalandElectronicsEngineers* (IEEE) a engenharia de software acontece na arte de obedecer para acrescentar, a operação e conservação do software.

Diante disso, Pressman (2006) diferencia a engenharia de software como sendo uma tecnologia em categorias que agrega processo, procedimentos e ferramentas para o desenvolvimento de 15 softwares (programas, dados e documentos) com a finalidade de fornecer uma estrutura para a construção de software com alta qualidade.

Figura 1 – Engenharia de software em camadas

Fonte: Adaptado de Pressman (2006).

A engenharia de software, para Sommerville (2003, p.78), é “uma cadeira da engenharia que se ocupa de todos os aspectos da produção de software, desde os aprendizados iniciais de particularização do sistema até a conservação desse sistema, depois que ele entrou em intervenção”. Analisando este conceito, destaca nele dois momentos: “curso de engenharia”, que diz respeito ao bom emprego de teorias, procedimentos e ferramentas apropriadas em momentos apropriados; e “todos os aspectos da produção de software”, o que explicaa engenharia de software não se confia apenas dos processos de desenvolvimento de software, mas também do gerenciamento de projetos de software e do aumento de estruturas de apoio a produção de software.

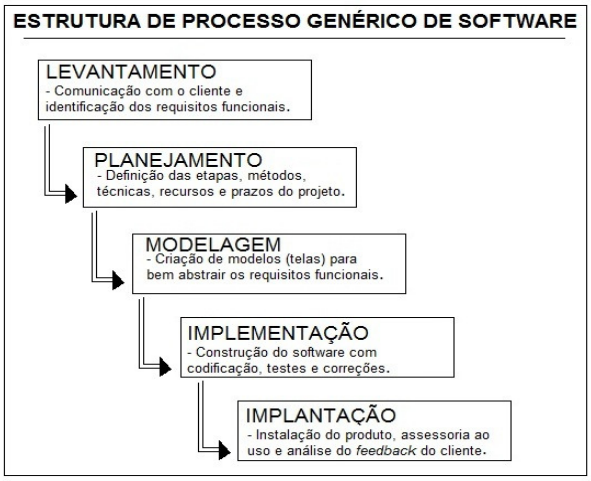
Segundo Souza (2003, p.20), “a engenharia de software é um comando altamente orientado a informação, no qual os fatores de sucesso estão acompanhantes com a experiência das pessoas envolvidas nas seguintes etapas: projeto, construção, teste e implantação”. Uma grande quantidade de conhecimento é produzida ao longo do processo de um aumento de software e este conhecimento precisa ser depositado em locais que promovam a recuperação e acrescentem valor ao processo.

O projeto de um sistema de software tem muitas propriedades em comum com projetos de engenharia, conforme o mapeamento dos requisitos funcionais em soluções tecnológicas. “Projetar sistemas de software constitui determinar como os requisitos funcionais são implementados na forma de estrutura de software”(PETERS, 2001, p.30).

Cada produto de software tem propriedades próprias quanto a busca contra *bugs*, carga, estatura do projeto ou urgência de entrega. Isso determinamodelos de produção que satisfaçam critérios necessários de qualidade e atenção para cada projeto.

Conforme Sommerville (2003) afirma que um processo de software é um conjunto de atividades e resultados acompanhantes que geram um produto de software. Assim, um processo de software se dá pela estruturação de um conjunto de atividades que derivam num produto software. Um processo deve colaborar na diminuição de custos, acrescentamento de qualidade e de produção. Um processo que não aguenta esses objetivos não é considerado um processo apropriado.

Figura 2 – Estrutura do Processo Genérico de Software



Fonte:Pressman (2006).

Para os subcapítulos seguintes, esse trabalho explorará algumas destas normatizações e o uso voltado à qualidade.

2.1 QUALIDADE DE SOFTWARE

A preocupação com o processo de software está relacionada à obrigação de entender, avaliar, controlar, aprender, comunicar, melhorar, predizer e certificar o trabalho dos engenheiros de software. De tal modo, são necessárias estruturas que possibilitem a documentação, medição, definição, análise, avaliação, comparação e a adulteração de processos(ROCHA, 2001).

Conforme Herzum (2000) apropriadas práticas são exigidas em todo o processo de incremento e cobrem todos os aspectos de acréscimo em larga escala, desde alcança de requisitos, verificação, validação e garantia da qualidade e, desde gerenciamento de projeto até engenharia de processos incluindo avaliações e choques de otimização de processo. De um caráter amplo, uma boa prática pode ser definida como sendo todos os aspectos não específicos em termos de tecnologia que são requeridos para preencher a lacuna entre a metodologia detalhada e o desenvolvimento real.

A modelagem por si só não fornece uma vantagem econômica direta, mas os modelos são instrumentos essenciais e a modelagem é uma atividade essencial na melhoria da qualidade de produtos, de processos e reuso funcional. Modelos são funções que fornecem aos desenvolvedores, controle intelectual sobre a máquina de um sistema(BUTLER, 1997).

Conforme Kulpa (2003), um modelo é considerado uma forma das melhores práticas encontradas no estudo de outras organizações que funcionam bem e são muito bem-sucedidas. Um exemplar não contém os passos necessários ou a sequência de passos necessários para implementar um algoritmo de melhoria de processo.

As organizações de software encontram-se por si mesmas complicadas em umas amostras e exemplares de melhoria, um problema que aborrece muitos segmentos da indústria. Muitos desses modelos compartilham ampla igualdade de conteúdo. As instalações de software irão querer mostrar como seus vários processos de incremento se relacionam com processos contidos em vários modelos. Então, os desenvolvedores necessitarão

adaptarseus processos de desenvolvimento com elementos iguais, divididos entre vários modelos de processo, a fim de investigar como os seus processos satisfazem esses padrões(CURTIS, 2000).

O desafio para organizações de sistemas na próxima década será acrescentar atividades de melhoria entre as áreas de engenharia, de modo que possam unir ao invés de somente classificar seus processos de desenvolvimento de sistemas. À medida que as instalações de software começaram a melhorar seus processos, a engenharia de sistemas e outras organizações de incremento começaram a gerar esforços iguais para melhoria.

É comum as empresas de software buscarem a qualidade dos produtos por meio da ampliação de funcionalidades disponíveis a usuários, eliminação de ausências, cumprimento ou antecipação de prazos, entre outras configurações. Porém é necessário ir um pouco além, penetrando a qualidade no próprio processo produtivo. Para essapretensão algumas fases precisam ser comentadas, entre as quais merecem destaque:

a) **garantia e padronizações de qualidade:**estabelecer algoritmos e padrões organizacionais que conduzam a um software de qualidade (SOMMERVILLE, 2003);
b) **planejamento de qualidade:**selecionar procedimentos e padrões específicos para um certo projeto (SOMMERVILLE, 2003);
c) **controle de qualidade:**Assegurar que os procedimentos e padrões são cumpridos pela equipe de desenvolvimento (SOMMERVILLE, 2003);
d) **custo da qualidade:** na qual se define o preço para a aquisição da qualidade desejada (PRESSMAN, 2006).

Na tabela 1 é possível observar o detalhamento das fases no processo de introdução da qualidade, as respectivas subdivisões e o foco da aplicação para cada uma.

A seguir o presente trabalho explorará alguns dos modelos e normas de qualidade disponíveis para a engenharia de software, explanando suas características e estruturas.

Tabela 1 – Etapas para inserção da qualidade no processo de software.

Etapas	Subdivisões	Aplicação das subdivisões
Garantia e padronizações de qualidade	Padronizações do produto	Produto
Garantia e padronizações de qualidade	Padronizações de processo	Processo
Planejamento de qualidade	Referência introdutória ao produto	Produto
Planejamento de qualidade	Planejamento para o produto	Produto
Planejamento de qualidade	Especificações do processo	Processo
Planejamento de qualidade	Metas de qualidade	Produto
Planejamento de qualidade	Riscos e seu gerenciamento	Produto
Controle de qualidade	Revisões de qualidade	Produto e Processo
Controle de qualidade	Avaliação automática de software	Produto
Custo da qualidade	Custos de prevenção	Produto e Processo
Custo da qualidade	Custos de avaliação	Produto e Processo
Custo da qualidade	Custos com as falhas	Produto e Processo

Fonte: Adaptado de Pressman (2006).

2.1.1Modelos de qualidade

Propagar a qualidade é uma obrigação verdadeira e intensa nas empresas de software (SOMMERVILLE, 2003). Nada é mais complicado do que investir pesadamente em tecnologias, estruturas operacionais, equipamentos, equipes altamente treinadas e grandeza de suporte técnico ao uso do sistema, se a qualidade do produto for menor ou incerta a cada nova alteração.

A unificação do processo de software, projetando medidas que dirijam a própria cadeia produtiva é uma escolha importante conforme estes desafios (ROCHA; MALDONADO; WEBER, 2001). Entretanto, logo que o trabalho, institutos como a ISO/IEC (ROCHA; MALDONADO; WEBER, 2001), ANSI/IEEE (ROCHA; MALDONADO; WEBER, 2001), SOFTEX (SOFTEX, 2011) e SEI (SEI, 2010, tradução nossa) não acertam importância na produção e melhoria de documentações agregadas que conduzem a qualidade do processo produtivo e dos produtos de software (GUERRA; COLOMBO, 2009).

As ações governamentais em companhia com organizações públicas e privadas têm provocado a formação de melhores práticas no incremento dos produtos de software por meio do PBQS, que nos últimos anos premiou diversos trabalhos com materiais com o uso de artifícios para cálculo da qualidade (BRASIL, 2006; GUERRA; COLOMBO, 2009). Os artifícios são:

a) CMMI (SEI, 2018, tradução nossa);
b) MPS.br (SOFTEX, 2011);
c) ISO/IEC 9126-1 (ABNT, 2003).

Os detalhes dos artifícios serão relatados em subcapítulos posteriores. Pertence separar que além de aceitar as estruturas fundamentais, também será mostrado o padrão de verificação do produto de cada modelo, com ênfase nos processos de teste de software.

2.1.1.1 Os modelos CMMI e MPS.BR

Diversos modelos de maturidade, padrões, metodologias e diretrizes podem fornecer com os aparelhamentos para que estes abordem suas obrigações de qualidade no processo produtivo, nos produtos de software, sendo que os limites de melhoria agem em uma área específica da cadeia produtiva (SEI, 2018, tradução nossa).

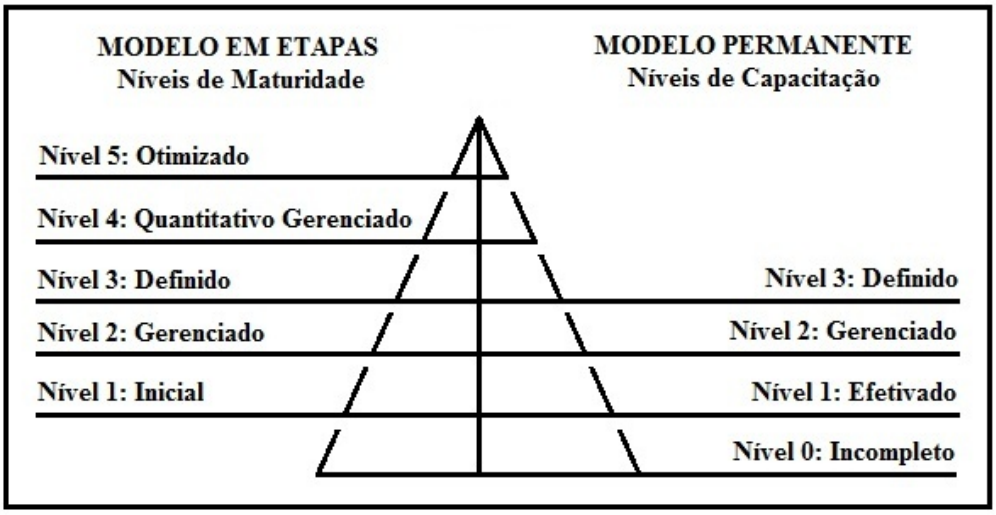
De uma forma ligeira, o Modelo Integrado de Capacitação e Maturidade, do inglês *CapabilityMaturityModelIntegration*(CMMI), inventado pela *Software EngineeringInstitute*(SEI), acerta nos mais perfeitos estágios voltados à ampliação e ao amparo de produtos e dos serviços, invadindo todo o ciclo de vida do produto, desde sua abrangência até a sua oferta e conservação (CHRISSIS, 2003).

Conforme Konrad (2003) é um dos modelos de maturidade mais aceitos no mundo no que diz respeito a medir o amadurecimento dos processos de uma arrumação. Quanto mais maduros forem seus processos, maior será a qualidade obtida no produto final, pois os graus de maturidade ajudam a aperfeiçoar seus processos. Isso ocorre, pois, antecipando o comportamento dos processos, torna-se a organização mais madura e competidora no mercado.

As métricas do CMMI possuem duas formas diferentes de aplicação, o modelo permanente e em fases. O modelo permanente mensura os níveis de capacitação, enquanto que a avaliação dos níveis de maturidade fica a cargo do modelo em fases.

A figura 3 demonstra essa divisão e os níveis de capacitação e maturidade pelos quais o processo de software pode ser avaliado.

Figura 3 – Modos e níveis do CMMI.



Fonte: Adaptado de SEI (2019, tradução nossa).

Para o modelo permanente, os níveis recebem as seguintes definições (SEI, 2018, tradução nossa):

a) **incompleto (nível 0):**quando determinada área do processo sequer é realizada ou se é realizada, ainda não agrada a níveis de capacitação exigidos pelo CMMI para o nível 1, um ou mais objetivos específicos não estão satisfeitos;

b) **efetivado (nível 1):** no qual os níveis de capacitação exigidos para determinada extensão de processo no nível 1 foram atingidos e todas as atividades estão produzindo os resultados relacionados, processos confusos;

c) **gerenciado (nível 2):** em que as áreas do processo estão sob controle, as equipes envolvidas estão atuando de forma plena nas atividades, os resultados da atuação estão sendo gerados conforme o necessário, com os envolvidos tendo acesso aos expedientes e ferramentas que precisam em cada serviço, sendo que a etapa está totalmente supervisionada para garantir a plena aprovação e adesão ao escopo do procedimento, além de todos os critérios do nível 1 terem sido atingidos, possui infraestrutura básica de suporte ao processo;

d) **definido (nível 3):** no qual o processo passou a ocorrer com base em diretivas da própria agência, alimentando assim um esqueleto de processos da própria organização, colaborando com

o resultado funcionalpela melhoria do processo envolvido, sendo que os critérios do nível 2 foram adquiridos, descrito mais rigorosamente.

O modelo em etapas possui os mesmos níveis 1, 2 e 3 citados anteriormente, com aplicação das mesmas leis, contudo no modelo em etapas existem outros dois níveis exclusivos para métrica da maturidade, que são (SEI, 2010, tradução nossa):

a) **quantitativamente gerenciado (nível 4):** em que os critérios do nível 3 foram conseguidos e o processo passa a ser gerenciado de forma quantitativa, por meio de medições da qualidade e performance, controlado por meio de técnicas quantitativas e estatísticas;
b) **otimizado (nível 5):** quando mecanismos estatísticos são utilizados para agenciar uma otimização do processo, gerando um aperfeiçoamento contínuo da área de processo que estiver sob avaliação, sendo que todas as exigências de maturidade até o nível 4 já estão atingidas, entendimento das causas comuns de variação inerentes ao processo.

Outra definição importante do CMMI é que as áreas de processo também estão subdivididas em categorias, as quais estão distribuídas por vários níveis de maturidade. A tabela 2 demonstra as áreas de processo, suas categorias e o respectivo nível de maturidade em que ocorrem (SEI, 2018, tradução nossa).

A tabela 2 exibe as quatro categorias presentes no processo de software do modelo CMMI. Nesse formato cada área do processo produtivo tem suas adequadas metas, também práticas indispensáveis para chegar a determinadas metas. As propriedades de uma área do processo são determinadas pelas suas metas especiais, assimrefinadas pelos exercícios especiais dareferente área do processo (PRESSMAN, 2006).

Deste modo como em diversos modelos de processo de software que miram qualidade, o CMMI utiliza, entre diferentes rotinas, processo de teste para tornar mínimo ou acabar com erros no produto. Os testes estão colocados na área de processo chamada de verificação,que tem como alvo fundamental garantir que produtos de trabalho causados em outras áreas de

processo, estejam ajustados com os requisitos funcionais almejados. Isso invade práticas especiais como a preparação de testes, verificação de desempenho, verificação e assimilação de ações corretivas, verificação do produto final e dos produtos de trabalho intermediários, também de comparações de serviços e aparelhos de serviços, dentre diversos exercícios (SEI, 2018, tradução nossa).

A verificação acontece conforme novos produtos de trabalho aparecem, é definido de processo incremental, começado na demarcação de requisitos e que progride com o produto de software ao longo do ciclo de vida, tornando a verificação aumente substancialmente a esperança de que o produto acatará ao usuário (SEI, 2018, tradução nossa).

Tabela 2 – Áreas de processos, categorias e níveis de maturidade do CMMI.

Área de processo	Categoria	Nível de maturidade
Análise de causas e resoluções	Suporte	5
Gerenciamento de configurações	Suporte	2
Análise de decisões e resoluções	Suporte	3
Projeto de manejo integrado	Gerenciamento de projeto	3
Medição e análise	Suporte	2
Definição de processo organizacional	Gestão de processos	3
Foco no processo organizacional	Gestão de processos	3
Gestão de desempenho organizacional	Gestão de processos	5
Desempenho de processo organizacional	Gestão de processos	4
Treinamento organizacional	Gestão de processos	3
Integração de produto	Engenharia	3
Monitoramento do projeto e controle	Gerenciamento de projeto	2
Planejamento do projeto	Gerenciamento de projeto	2
Processo e garantia da qualidade do produto	Suporte	2
Gerenciamento quantitativo de projeto	Gerenciamento de projeto	4
Desenvolvimento de requisitos	Engenharia	3
Gerenciamento de requisitos	Gerenciamento de projeto	2
Gestão de risco	Gerenciamento de projeto	3
Acordo para gestão de fornecedores	Gerenciamento de projeto	2
Solução técnica	Engenharia	3
Validação	Engenharia	3
Verificação	Engenharia	3

Fonte: Adaptado de SEI (2019, tradução nossa).

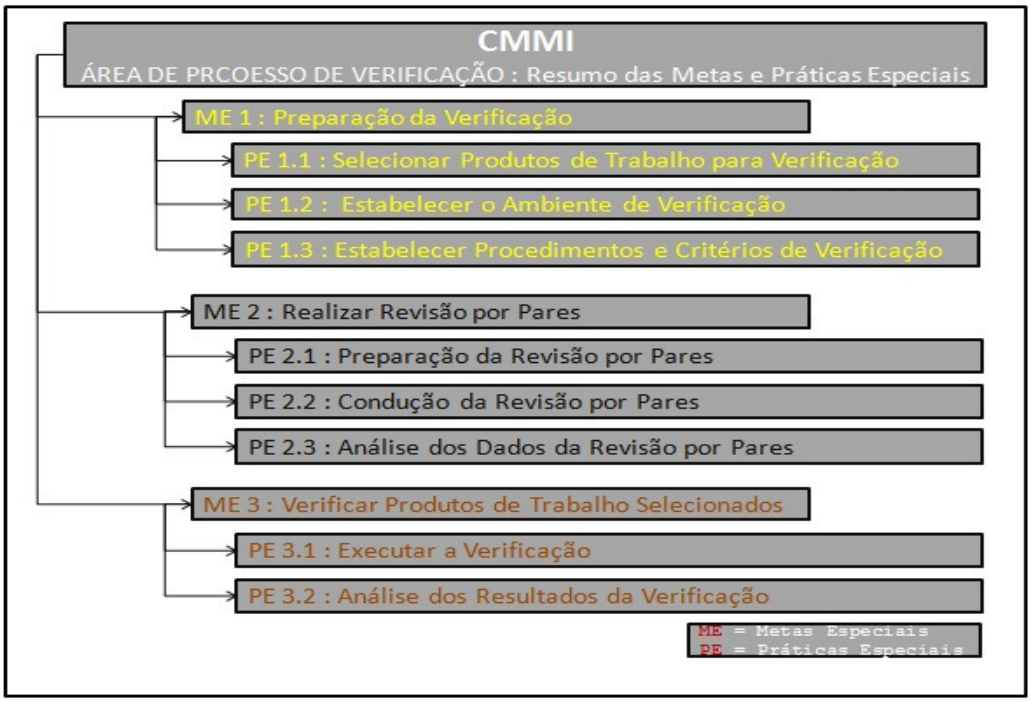
Analisando que o CMMI aceita também a área de processo de validação, ajusta destacar que apesar de idênticos, verificação e validação têm

objetivos diferentes, a verificação averigua se “você construiu direito” e a validação valida se “você construiu a coisa certa” (SOMMERVILLE, 2003).

As áreas de processo do CMMI têm uma composição encadeada de metas e exercícios especiais. Na figura 4 pode-se ver a composição sumarizada da área de processo de verificação, com diversas metas e exercícios especiais determinados pelo modelo CMMI (SEI, 2018, tradução nossa).

Faz jus a destaque um auxilio importante que o modelo CMMI causa para a área de processo de verificação, o cálculo pelos pares, esse aceita que outros membros de grupo acham falhas que possam estar escuras devido a vícios ou hábitos operacionais mal ajustados, permitindo uma chance de enriquecimento no processo produtivo do software (SEI, 2018, tradução nossa).

Figura 4 – Metas e práticas da área de processo de verificação no CMMI.



Fonte: Adaptado de SEI (2019, tradução nossa).

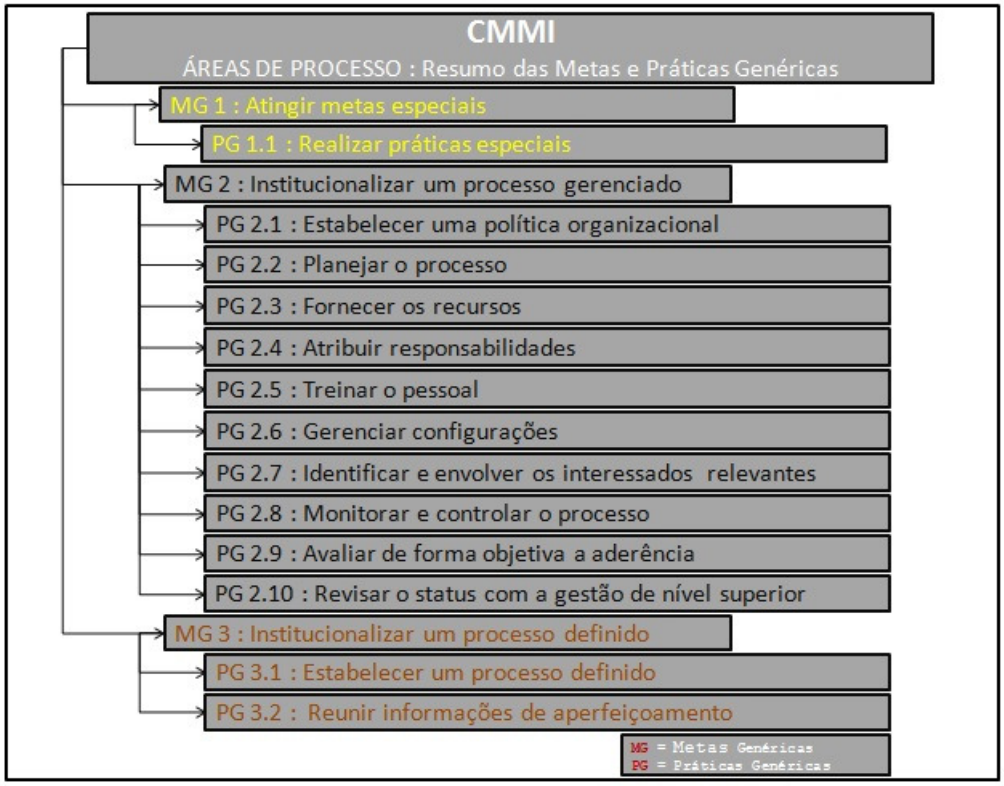
O modelo CMMI acelera também uma composição de metas e exercícios genéricos para serviço nas áreas de processo. São três metas, cada uma com vários exercícios comuns, os quais obedecem aos níveis alcançados, gerenciado e decidido de capacitação. A figura 5 mostra de forma detalhada essa composição genérica de metas e exercícios para as áreas de processo.

Pode-se garantir que o modelo CMMI é combinado por metas e exercícios, sendo aproveitáveis de forma específica no processo de incremento de software. Podemos ressaltar um modelo na figura 3, a qual indica o processo de verificação. Imediatamente na figura 5 podemos considerar alguns exercícios genéricos que são aproveitados a todas as áreas abrangidas.

As metas genéricas apresentam aspectos indispensáveis em que um processo pode ser institucionalizado. A institucionalização menciona ao nível de repetibilidade, unificação e sofisticação do controle (KANUNGO; GOYAL, 2004, tradução nossa). Concluindo conseguir os mais altos graus de maturidade e capacitação transporta o processo de software a uma ascensão da qualidade, durante o processo e no produto final.

Por ser um modelo de processo de software que provoca investimentos abundantes tanto em reestruturação de grupos, quanto na adequada reengenharia de processos, o CMMI se apresenta uma padronização que cai nas barreiras físicas e até financeiras de micro, pequenas e médias empresas brasileiras desenvolvedoras de software (COUTO, 2007; SOFTEX, 2011).

Figura 5 – Metas e práticas genéricas no CMMI.



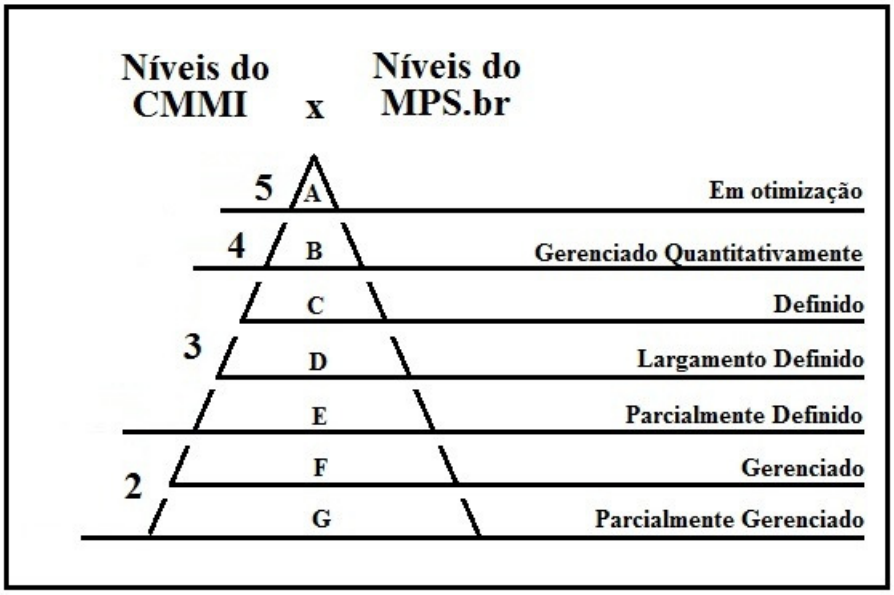
Fonte: Adaptado de SEI (2019, tradução nossa).

Visando driblar esse panorama e permitir que aparelhamentos de todos os tamanhos alcancem disseminar com êxito progressos em seu processo de software, foi inventado o modelo de Melhoria de Processo do Software Brasileiro (MPS.BR) (SOFTEX, 2011).

Desenvolvido e conservado pela SOFTEX, o MPS.BR é combinado pelo Modelo de Referência (MR-MPS), Método de Avaliação (MA-MPS) e Modelo de Negócios (MN-MPS), recursos com os quais apóia as companhias incomodadas em botar um avanço no processo de software (SOFTEX, 2011).

Logo que é aceitável analisar na figura 6, a composição do modelo MPS.BR altera do nível G (mais baixo) até o nível A (mais alto) de maturidade, consisti que estes níveis são aproveitáveis de maneira igual aos níveis do modelo em fases do CMMI. A diferença está nos níveis G e E, que são ao mesmo tempo chamados de parcialmente gerenciado e parcialmente definido. Estes dois novos níveis trabalham como sub-níveis, que promovem a obtenção dos níveis de maturidade e a mudança entre graus, promovendo o acesso das micro, pequenas e médias companhias na melhoria do processo de software (COUTO, 2007; SOFTEX, 2011).

Figura 6 – Níveis do modelo MPS.br X níveis do modelo CMMI.



Fonte: Adaptado de SOFTEX(2011) e SEI(2019, tradução nossa).

Analisando que o enfoco do trabalho presente são as rotinas conectadas ao processo de teste de software, acerta mencionar que para o

modelo MPS.BR, tal solução é colocado no nível D, o qual considera a área do processo de verificação do produto (SOFTEX, 2011), ainda de maneira comparável ao que acontece no nível três do modelo CMMI (SEI, 2018, tradução nossa).

O subcapítulo consequente vai relatar a norma NBR ISO/IEC 9126 (ABNT, 2003), comprovando algumas propriedades, a ajuda de tal modelo para a qualidade de software e como o próprio aplica o processo de testes.

2.1.1.2 O modelo NBR ISO/IEC 9126

A normatização NBR ISO/IEC 9126 (ABNT, 2003) é um modelo de processo de software enfocado na qualidade do produto de software (ABNT, 2003; ROCHA; MALDONADO; WEBER, 2003).

A norma está subdividida entre modelo de qualidade para propriedades internas e externas, bem como em um modelo de qualidade para a qualidade em usabilidade.

O modelo para propriedades internas e externas está desconectado entre seis propriedades, consistindo em que elas se subdividem em características que após são acomodados e analisados conforme regras da norma.

Segundo as normas da ABNT (2003), as seis propriedades que ajeitam o modelo de qualidade interna e externa do produto de software são:

a) **funcionalidade:** determina se o software admite o uso de funções com exigências explícitas ou subentendidas e suas propriedades especiais;
b) **confiabilidade:** define se o software pode se manter garantido, apresentando seus resultados tradicionais por um determinado período de tempo e sob determinadas categorias;
c) **usabilidade:** indica o grau de esforço necessário para o uso do software com base nocálculo de um grupo de clientes;
d) **eficiência:**envia à relação entre o grau de desempenho do software com o número de costumes em uso dentro de um escopo de situações estabelecidas;

e) **manutenabilidade:** refere-se ao grau de complicação ou facilidade em se realizar manutenções no software com baixarepresentação negativa;
f) **portabilidade:** indica o quanto um software pode ser facilmente adaptado de um ambiente operacional ou uma composição de sistemas, para outra.

Também conforme as normas da ABNT (2003), para o modelo de qualidade em usabilidadetêm quatro propriedades que são:

a) **efetividade:** determina o quanto o software possibilita aos clientes que consigam realizar tarefas específicas conforme almejam, mantendo precisão nos resultados obtidos;
b) **produtividade:** define o relacionamento entre o número de costumes utilizados para se obter o máximo de efetividade;
c) **segurança:** indica se os graus de risco oferecido pelo software a terceiros está dentro de níveis aceitáveis, frente a um argumento de uso específico.
d) **Satisfação:** refere-se à forma como o software obtém cálculos de satisfação pelos clientes, também frente a um argumento de uso específico.

Logo queocorre para diferentes normas, a NBR ISO/IEC 9126-1 (ABNT, 2003) necessita ser analisada como uma das escolhas para a avaliação da qualidade em produtos de software, assim sendo, pode seraplicado de maneira acabada, quando indispensável ou parcial, como saída de complementação a outros modelos de processo de software.

Ainda com todas as saídas ajeitadas pelos modelos e normatizações do processo de software, o modo de manutenção de software permanece consistindo em algo capital. O próximo subcapítulo atingirá esse assunto, detalhando suas considerações e fundamentais propriedades.

2.2 MANUTENÇÃO DE SOFTWARE

A manutenção de software é um tanto comum e indispensável quanto a manutenção de qualquer outro benefício, como exemplo, um veículo ou uma escada rolante.

Enquanto os benefícios materiais normalmente suportam consertos ou trocas de equipamentos conforme a corrosão pelo uso consecutivo, o software não absorve essechoque por ser intangível (SOMMERVILLE, 2003). Portanto é complicado decidir tudo que uma manutenção poderá causar de intervenção involuntária nos alvos que trabalhavam antes daoperação, o que deixa a manutenção de softwareum tanto diferenciado.

Segundo Pressman (2006) se desperdiça mais tempo oferecendo manutenção em um software, do que para inventá-lo, chegando a ser gastado cerca de 60% do empenho produtivo para essa atividade. Espontaneamente as despesas atingem á baliza de 60% do valor sendo gastado no serviço de manutenção (GHEZZI; JAZAYERI; MANDRIOLI, 1991, tradução nossa).

Embora conforme Pressman (2006), o hábito de manutenção pode acontecer por múltiplos motivos, entre os quais tem direito a evidência.

a) a busca de atualizá-lo perante a novos artifícios ou adaptá-lo a outros aparelhos;
b) mudança para abranger novas usabilidades requisitadas pelo usuário final, atribuídos por aparelhos governamentais ou outras institutos;
c) ajustes de falhascausadas nos requisitos funcionais, no projeto ou no incremento.

Analisando que as manutenções geralmenteocorrem em momentos pequenos, com acordos ouaparelhos legais exigindo rapidez, maior é a possibilidade de que ocorra a baixa na qualidade do produto. Por isso, os produtos de softwares passam por conservações firmes buscando boas ferramentas, tecnologias ou normas gerando eficácia na qualidade final.

Um dos padrões que são administrados pelas companhias na procura da qualidade é a manutenabilidade, com várias propriedades reveladas no subcapítulo consequente.

2.2.1 Manutenabilidade

Buscar a qualidade em um produto de software que exige ampla manutenção é um alvo que determina um empenho diferenciado dos grupos abrangidos (SOMMERVILLE, 2003).

Segundo Rocha, Maldonado e Weber (2001), no entanto a manutenção precisa acontecer com o desenvolvedor cauteloso aos alvos do projeto que verdadeiramente devem sofrer intervenção, reaplicando as tecnologias que aproveitou para alcançar a qualidade na invenção original. As correções ou alterações devem concordar com os requisitos funcionais e a composição lógica, porém os vários itens em funcionamento antes da intervenção devem continuar operacionais. Por fim deverão acontecer testes funcionaisque compreendam os pontos alterados, também os testes de regressão dos componentes em manutenção, funcionando assim em modo pleno.

Tais propriedades demandam práticas que tornam fácil manutenções posteriores, admitindo que de maneira mais clara que os impactos sejam mais previstos. Busca-se permitir que qualidades, como a confiabilidade, esteja voltada a graus apropriados (GHEZZI; JAZAYERI; MANDRIOLI, 1991, tradução nossa). A essa aptidão aperfeiçoada da manutenção do software é determinada de manutenabilidade.

Todo software necessita de manutenção, sendo para ajustar erros ou acatar novos requisitos. Com o software deve existir a facilidade de permanecer para que estas atualizações gerem sucesso. A segunda propriedade menciona os detalhes da documentação e a adoção de tecnologias, bem como, o exame certo do projeto de origem, antecedendo posteriores manutenções e tentar tornar simples na programação em andamento (GHEZZI; JAZAYERI; MANDRIOLI, 1991, tradução nossa).

A normatização NBR ISO/IEC 9126-1 (ABNT, 2003) oferece a manutenabilidade sendo uma propriedade em tais subcaracterísticas:

a) **analisabilidade:**competência do produto de software de aceitaraanálise de ausências ou causas de falhas no software, ou a identificação de partes a serem modificadas;
b) **modificabilidade:**aptidão do produto de software de permitir que uma modificação especificada seja praticada, implementação inclui modificações no código, projeto e documentação, se o software for modificável pelo usuário final, a modificabilidadepode afetar a operacionalidade;

c) **estabilidade:**capacidade do produto de software de evitar efeitos imprevistos decorrentes de modificações no software;
d) **testabilidade:**disposição do produto de software de permitir que o software, quando alterado, seja validado;
e) **conformidade com a manutenabilidade:**competência do produto de software de estar de ajuste com normas ou convenções relacionadas à manutenibilidade.

Ressaltando a norma NBR ISO/IEC 9126-1 (ABNT, 2003) assim torna precoce a obrigação da concretização do processo de testes visando garantir a qualidade, por meio do tema de manutenabilidade dos produtos de software. Nos capítulos e subcapítulos posteriores serão relatados o processo de testes de software, com detalhes sobre suas considerações, suas tecnologias, ferramentas e respectivas propriedades.

3TESTES DE SOFTWARE

A usabilidade do processo de testes de software tem por alvo aprovar, conforme uns fatos, se os quesitos funcionais incrementados são idênticos ao que foi solicitado. Pode-se relatar que os testes de software buscam caçar as deformidades do produto de software (MYERS, 2004, tradução nossa; ROCHA; MALDONADO; WEBER, 2001; SOMMERVILLE, 2003).

Em uma avaliação colegial o estudante recebe uma nota que equivale ao que ele estudou. As notas mostram se o estudante será desaprovado quanto à qualidade do estudo colocado no teste. De maneira igual um usuário será desaprovado a qualidade de um produto de software se não acatar aos requisitos funcionais conforme a expectativa. Nesse acontecimentovivência de um grupo enfocado nos testes de software, antes que ele se transforme em operacional, atua como um agente contra *bugs*. Conforme a isso o exercício do teste de software é apresentado como um dos fundamentais soluções para manutenção da qualidade em produtos de software(GHEZZI; JAZAYERI; MANDRIOLI, 1991, tradução nossa;MOLINARI, 2010; MYERS, 2004, tradução nossa; RIOS; MOREIRA, 2006; SOMMERVILLE, 2003).

A implantação do processo de testes determina investimento de início em grupos, aparelhamentos e montagens que geram dúvidas de que existiram lucros. No entanto o choque positivo adquirido pela criação do processo de testes não é considerado, diminuindo o andamento de produção, enfraquece cotações e aumenta a qualidade. São geradas maiores produtividades, aumento ganhos lucrativos, com melhorias no contato com o usuário final tornando confiante o produto e o grupo de produção comunicam (BASTOS et. al., 2007).

A diminuição de andamento e cotação produtiva acontece especialmente visto que o teste de software analisa a sua documentação, que admite os manuais ou os determinados requisitos funcionais, ainda o produto do software em si. Isso apura o modelo produtivopara que exista a documentação idêntica ao produto que será oferecido, no término do processo

do programa, o produto como a bibliografia deverão obedecer as requisitos funcionais almejados (RIOS; MOREIRA, 2006).

O resultado será o ajuste de algumas falhas mesmo nas documentações ou na analise do sistema, o que é menos caro, impedindo impactos futuros no andamento e cotação de tarefas, enfraquecendo acontecimentos posteriores para determinados grupos (BASTOS et. al, 2007):

a) **projetistas:** evita que novos pedidos de correções tenham que ser analisados e projetados;
b) **desenvolvedores:** reduz o redesenvolvimento de falhas no projeto em curso ou facilmente o desenvolvimento do projeto de novos pedidos de correções;
c) **equipe de testes:** elimina o reteste de falhas do projeto em curso ou facilmente o teste do projeto de novos petições de correções.

Apesar de ativo no alcance da qualidade, os trabalhos de teste necessitam ser desenvolvidos por grupos especializados, que aceitem táticas e tecnologias de testes (RIOS; MOREIRA, 2006), apresentando conhecimentos sobre regras de negócios transformados em requisitos funcionais do programa. Esses conceitos são importantes pois o grau crucial das rotinas em que o programa é aplicado causará o número de testes a desempenhar (BASTOS et al., 2007).

O teste de software aproveita determinadas tecnologias sendo mencionada dentre várias à técnica de caixa-branca, também chamada de estrutural e a técnica de caixa-preta, conhecida como funcional (PRESSMAN, 2006). Tem assim as técnicas baseadas em erros e baseadas em aparelhos de estados finitos. As técnicas se complementam, competindo um estudo de casos e de mais perfeitas maneiras de aproveitamentos (ROCHA, MALDONADO; WEBER, 2001).

A concretização do processo de testes está repartida em fases, que obedecem ao teste de unidade, teste de integração e teste de sistemas. Cada etapa admite alvos específicos, sendo assim, a de testes de unidade, que cultiva menores pedaços do software ou a de testes de sistemas, em que o produto em execução almejando observação das propriedades do programa

(DELAMARO; MALDONADO; JINO, 2007; PRESSMAN, 2006; ROCHA; MALDONADO; WEBER, 2001).

Dentre os serviços que podem ser feitos no processo de testes de software está à bibliografia dos andamentos e casos de testes, queapontamo implemento do processo de testes acolhendo a avaliação das funcionalidades entre aberturas válidas ou inválidas, gerandonotificações finais de teste (DELAMARO; MALDONADO; JINO, 2007).

A opinião dos roteiros e casos de testes promove o serviço de critériosque determinem melhor compreensão do processo de teste, evitando que aconteça além do necessário. Uma coleção de testes adiante do que precisa, será gerado demora na entrega do produto, de outra maneira, testesem números menores, gera precipitações de baixa qualidade conforme a quantidade de falhas não absorvidas e oferecidas com o produto (RIOS; MOREIRA, 2006; ROCHA, MALDONADO; WEBER, 2001). Pode-se lembrar que os critérios de derivação dos casos de testes funcionais podem ser decompostos em formais e não formais.

Ainda conforme Rios e Moreira (2006) advertem o critério não formal fundamentado na ciência e experiência do especialista é uma alternativa aplicativa para a derivação de casos de testes funcionais andar a lançar frutos admissíveis. O dado se demonstra válido detalhando que o desenvolvimento é simples na indústria de programa da cidade de Criciúma.

Os critérios de derivação formais estão compreendidos entre particionamento de equivalências, análise do valor limite e grafo causa-efeito, também aproveitáveis na técnica de testes funcionais, havendo ainda os critérios com apoio na máquina, com apoio em fluxo de controle e com base no fluxo de dados para a técnica estrutural (PRESSMAN, 2006; ROCHA; MALDONADO; WEBER, 2001). Segundo Pressman (2006), pode-se adicionar na tabela o critério de teste de matriz ortogonal, a qual aproveita na técnica funcional.

O ajuste destes dados deriva em um processo de teste de software estruturado, encontrando falhas, assim restringindo o aproveitamento dos testes para que aconteçam dentro de um alvo admissível de qualidade e tempo determinado. Dando sequências aos temas, no seguinte subcapítulo seráabordada a técnica de teste caixa-preta ou funcional,confirmando suas

fundamentais propriedades, sendo assim, os tipos de testes sofridos e barreiras de técnica, além de ser oferecido em ênfase o tipo de teste funcional de regressão, que é um enfoco fundamental do atual trabalho.

3.1 TESTE CAIXA-PRETA

A técnica de teste caixa-preta não considera a composição interior do produto, assim sendo, funções, algoritmos e métodos do programa, em fundamento com requisitos funcionais e nas regras de negócio na qual o produto se aproveita (ROCHA; MALDONADO; WEBER, 2001).

A composição de casos de testes projetada empreende, por meio de aberturas válidas, as respostas válidas e inválidas, que comprovam a adesão do software aos requisitos funcionais e ao comando indispensável dos dados. O alvo destacar erros conectados à eliminação ou incorreção de funções, mau conduta de interface, defeitos na composição ou ascensão a dados, dificuldade de performance, também desacertos de início e término (PRESSMAN, 2006). Ainda assim sendo, segundo Molinari (2008), uma das barreiras dos testes funcionais ou de caixa-preta é a de não ser possível de se permitir que linhas do algoritmo planejado ou todas as aberturas do fluxo de controle do programa atravessarão por testes.

A técnica funcional admite o serviço vários tipos de testes na caça de falhas características, sendo mencionados os testes de (BASTOS et al., 2007):

a) **requisitos:** buscam isolar falhas nos requisitos do software quando comparados com o produto final obtido para a execução dos testes. Também exploram o tempo que as funcionalidades permanecerão com funcionamento adequado. Ocorrem por meio da criação de *checklists* que dominem recursos funcionais e situações de testes, sendo que as situações de testes são compostas à medida que o ciclo de vida do software progride;
b) **regressão:** visam permitir que partes do programa que já estavam em funcionamento antes de uma alteração ou um novo recurso inventado, permaneçam em pleno funcionamento sem reflexos negativos nas funcionalidades ou na performance. Em

subcapítulo posterior este tipo de teste será largamente abordado diante de sua importância para softwares de manutenção constante, um dos focos do trabalho em andamento;

c) **tratamento de erros:** exploram a capacidade do programa reagir adequadamente a entradas indevidas de dados ou a rotinas incorretamente acionadas pelo cliente. Podem suceder com a aplicação da técnica de *brainstorm*entre a equipe do projeto e de usuários*,* com a qual são elencados possíveis falhas de sistema, para assim serem criados conjuntos de operações de testes que validem o condicionamento do controle e a atitude correta do software frente ao erro;

d) **suporte manual:**averiguam se os procedimentos de suporte manual para os testes têm bibliografias e se estão completos. Também valida se as cargas pelo suporte manual estão claramente definidas no grupo. Devem acontecer já nas fases de inicio do ciclo de vida do software, porém testes extensivos devem ser alcançados em equivalente com as fases de entrega e implantação do programa;

e) **interconexões com outros softwares:**buscam a garantia de que a comunicação e interconexão com outros softwares está em pleno funcionamento, com transferência ou recepção de dados ocorrendo de forma adequada ao indispensável. Também são validados aspectos documentais dessas funcionalidades, questões de execução nos momentos corretos e oarranjo das funções individuais dos programasabrangidos. Ocorrem com a execução simultânea de vários programas que se comunicam, evolvendo análise e validação dos dados transferidos. Podem simular aumento de custo e prazo no processo de software;

f) **controle:** têm a finalidade de explorar se os dados estão alinhados, se as transações são realizadas adequadamente e se dados têm sido recebidos para uma possível auditoria de processos. Também avalia se os processos são realizados de forma competente, econômica e atendendo às necessidades funcionais. São aplicados com acaricatura de controles que

minimizam riscos no software, sendo acatado quase que um sistema dentro de outro. Após isso o responsável pelos testes cria situações de risco para confiar o resultado dos testes de controle. Um dos métodos aproveitáveis é a matriz de riscos, que contém também os controles e o segmento do sistema em que se aproveitam;

g) **versõesparalelas:**apontam a comparação entre uma nova versão de programa com sua versão anterior, validando ou invalidando as funcionalidades antevistas para a nova versão. Podem exigir a atualização dos dados de entrada conforme os modelos esperados pela nova versão em teste. Sucedem com oimplemento simultâneo da versão nova e a versão anterior, com análise comparativa de resultados alcançados nas funcionalidades testadas, tentando isolar possíveis falhas.

Entre todos os tipos assinalados, o teste de regressão se observa como uma alternativa duradoura e aplicável para programas de durável manutenção, cumprindo papel de evidência no abaixamento de vencimentos e cotações, também com aumento da qualidade de determinados produtos de software, sendo exercitado de maneira manual ou com aproveitamento de soluções da automação de testes. Os lucros acontecem por meios da reexecução de casos de testes com conseqüências validas para alvos do programa que não ocorrerem mudanças, mas que necessitam atuar corretamente, como atuavam antes da operação (PRESSMAN, 2006).

A informação sobre os tipos de testes aceita analisar o que será mais apropriado ao argumento de cada software que se almeje conter a um teste funcional ou caixa-preta, bem como frente ao que se almeje designar de andamento e soluções para o alcance de proveito na qualidade da produção. Objetivando obter essa idéia, o subcapítulo consequente atingirá as demarcações dos fundamentais critérios formais e não formais para a derivação de casos de teste caixa-preta.

3.1.1 Critérios para Derivação de Teste Caixa-preta

A preparação de roteiros e casos de testes caixa-preta em relatos sem uso de críticas pode lançar testes que aumentam o andamento do processo de software ou executar determinados testes abaixo do indispensável, ocultando falhas e colidindo na qualidade do produto.

Também é possível analisar que, segundo Myers (2004, tradução nossa), testes cansativos são dificílimos ou inexecutáveis, pois a quantidade de alterações de testes que garanta todos os acordos de aberturas válida ou inválidas pretende a vincular o processo.

Em presença destas barreiras e obrigações a engenharia de software abastece a usabilidade de críticas para ascensão dos roteiros e casos de testes.

Os roteiros de testes estão conectados mais aos requisitos funcionais que podem ser contidos a testes no programa. Os casos de testes colocados à maneira como acontecerão teste de determinados requisitos, com as referentes aberturas válidas ou inválidas, também as saídas anunciadas (MOLINARI, 2008).

Em meio a os critérios mais acentuados para escolha dos roteiros e casos de testes são:

a) **particionamento de equivalência:** as entradas válidas ou inválidas que serão submetidas a testes são ajuntadas em classes de equivalências. Cada classe aguenta a representar um grupo do domínio de entrada, sendo que se um dos elementos do grupo possuir um verificado comportamento válido ou inválido, os demais são considerados equivalentes a ele. Em resumo o número de variações em roteiros ou casos de testes tende a diminuir posto que o teste de um componente de cada grupo corresponderá a submeter todo o grupo ao mesmo teste (DELAMARO; MALDONADO; JINO, 2007; PRESSMAN, 2006);
b) **análise do valor limite:** é um critério complementar ao particionamento de equivalência, porém trabalha com áreas limítrofes de autoridade para as entradas e saídas. A aplicação do critério de análise do valor limite ocorre sobre as partições de

equivalência, lançando roteiros ou casos de testes que cultivarão os limites de domínio por classes (DELAMARO; MALDONADO; JINO, 2007). De maneira abreviada, este critério também reduz a quantidade de roteiros e casos de testes ao diminuir os testes às fronteiras das classes equivalentes (MOLINARI, 2008);

c) **grafo causa-efeito:** os critérios já apontados apresentam barreiras na produção de roteiros ou casos de testes que garantam as reações e comportamentos do software diante de entradas de dados acertadas. Em resumo o grafo causa-efeito elimina referências confusas e imperfeitas nas definições dos requisitos funcionais. O uso desse critério requer que o programa seja dividido em partes para facilitar a criação do grafo. Posteriormente ocorre a assimilação das causas, que seriam entradas válidas ou inválidas, bem como são isolados os efeitos, que correspondem às saídas ou mudanças de estado, recebendo cada um destes um número correspondente. Após isso ocorre àorigem do grafo que define a fluência dos conduta do software e nesse momento são documentadas as causas e efeitos com teste incapaz. Por último deverá ocorrer àorigem de uma tabela de decisão e a partir desta, os casos de testes serão compostos (MYERS, 2004, tradução nossa).

d) **matriz ortogonal:** é um critério que basicamente deve ser aplicado diante de autoridade de entrada que inviabilizam testes cansativos. A fundamental vantagem deste critério é permitir a caça de falhas oriundas de lógica defeituosa no programa (PRESSMAN, 2006). A decisão pela usabilidade desse critério deve considerar que é caro testar tudo, mas incorreto não testar nada. Para usabilidade do critério é preciso inicialmente motivar quantas e quais condições ou combinações serão testadas. Posteriormente define-se o código máximo de valores que cada elemento condicionado ou combinado receberá. Seguindo adiante é preciso motivar quantas e quais serão as variáveis a serem tratadas no teste. Com todas as informações coletadas, monta-se uma tabela na qual o valor aceitável para cada variável

deva estar presente pelo menos uma vez. A partir doarranjo os roteiros e casos de testes serão formados, garantido que cada um dos parâmetros de entrada tenha seus valores possíveis contidos a testes funcionais pelo menos uma vez (MOLINARI, 2008);

e) **experiência e conhecimento do especialista de teste:** além dos critérios formais já apresentados, os quais também têm importância e aplicabilidade, existe também o critério não formal baseado no conhecimento e experimento do especialista de testes (RIOS; MOREIRA, 2006). O critério é empregado pelo conhecedor, que se vale das informações em técnicas de testes, regras de negócios e funcionalidades a testar, além doexperimento de testes anteriores e erros já detectados. Com o uso desses recursos ocorre o levantamento dos casos de testes que tenham coerência com as funcionalidades, que se mostram mais tendentes a identificar erros,que sejam reutilizáveis e que não copiem testes entre si. Por fim a análise de tais fatores deve resultar em um planejamento de testes, documentado em uma matriz ou em um programa de automação do gerenciamento do teste. Abibliografiagerada deve representar um conjunto de casos de testes coerentes, eficientes e poupados frente às funcionalidades que serão estudadas.

Uma propriedade admirável e essencial a todos os critérios assinalados e que merece ênfase, é a extraordinária obrigação de que os requisitos funcionais estejam acabados, alinhados e coerentes no momento da composição dos roteiros e casos de testes, pois no caso adverso, as consequências dos testes serão examináveis (DELAMARO; MALDONADO; JINO, 2007).

É aceitável advertir que cada um dos critérios assinalados possui probabilidades deferentes de aproveitamento, admitindo que vários deles sejam acertados para uma mais perfeita produção de roteiros e casos de testes. Aliás para conservar o enfoco didático, o trabalho presente aplicará a usabilidade do critério não formal fundamentado no conhecimento e experimento do especialista, uma vez mais alastrado comercialmente na

cidade de Criciúma, buscando conseqüências mais abrangentes, que possam servir de apoio adequado ecomparativo em posteriores análises invadindo os vários critérios.

3.2 TESTE CAIXA-BRANCA

O teste caixa-branca, ainda chamado de teste estrutural, incide na técnica em que os casos de testes são inventados analisando exclusivamente os recursos lógicos interiores do programa (MYERS, 2004, tradução nossa), assim sendo, estruturas de dados e fluxos de controle (PRESSMAN, 2006). A finalidade fundamental é garantir que o software possua fortaleza e coerência na composição lógica de procedimentos, desempenho, métodos evários recursos planejados(BASTOS et al, 2007).

O programa possui diferentes composições de decisão, assim diferentes acessos nas quais as informações e comandos correrão a passagem de consequências válidas ou inválidas, assim sendo, a passagem da operação perfeita ou das falhas, sendo dificílimo o teste de todos estes acessos aceitáveis (MYERS, 2004, tradução nossa). Por isso a técnica de testes estruturais ainda antecipa o serviço de critérios para a escolha dos casos de testes.

A técnica caixa-branca admite diversos tipos de testes específicos. Em meio a eles podem ser mencionados os testes de:

a) **estresse:** nos quais o programa é contido a execuções que testem os extremos de processamento, memória ou capacidade para leitura de informações;
b) **execução:** em que são comparadas questões de atuação e comportamento frente a padrões pré-estabelecidos no projeto;
c) **recuperação:** cujo objetivo é analisar a capacidade de reversão de uma condição calamitosa ou confusa do programa, como por exemplo, o retorno de um *backup* de banco de dados que tenha sofrido um dano físico ou estrutural;
d) **operação:** que visam analisar se os operadores, com base na bibliografia de operação, conseguem por o software em

funcionamento no ambiente de produção de maneira aceitável eacabada;

e) **conformidade:** os quais analisam se as normas, padrões e manuais de incremento foram adotados durante a produção do programa;
f) **segurança:** nos quais são analisados os recursos que avalizam a integridade e confidencialidade dos documentos e elementos.

É extraordinário também lançar que os testes estruturais podem ser comprometidos por algumas barreiras, como por exemplo, as dificuldades de (DELAMARO; MALDONADO; JINO, 2007):

a) **resultados para caminhos distintos:** não é possível confirmar que dois caminhos diferentes para fluxo de dados ou de controle podem calcular a mesma função;
b) **garantia da execução:** é indefinível que hajam entradas de dados que afirmam a execução de um verificado caminho no fluxo de controle ou de dados do programa;
c) **caminhos ausentes:** é dificílimo garantir que caminhos ausentes, por defeito no projeto ou no incremento, sejam encontrados nos testes;
d) **correção coincidente:** não há seguranças de que uma entrada de informação que gere resultado válido dará o mesmo resultado se outro dado for atribuído como entrada. Tal dificuldade também pode estar presente em outras técnicas e em múltiplos tipos de testes.

Mesmo segundo Delamaro, Maldonado e Jino (2007), apesar destas barreiras, análises têm comprovado que é duradoura a usabilidade da técnica de testes estruturaisna caça do aumento da qualidade nos produtos de software, competindo aos desenvolvedores pelos grupos de testes uma analise formal e esmiuçador em procura na mais perfeita aplicação complementando a usabilidade de outras técnicas, sendo a funcional, detalhada no capítulo antecedente.

O próximo subcapítulo irá relatar questões conectadas a modelos do ciclo de vida do teste de software, suas alterações, as referentes aplicações e proveitos acomodados, assim outros aspectos.

3.3 CICLO DE VIDA DO TESTE DE SOFTWARE

O processo de teste de software pode ser desenvolvido através de um modelo que consiste num arranjo estruturado de técnicas, fases e rotinas, que é determinado de ciclo de vida do teste de software. Este modelo determina tornar dinâmico o processo e manter que cada caminho seja feito de maneira alinhada, oferecendo à rotina posterior o que for possível para o prosseguimento dos testes com ênfase na qualidade (RIOS; MOREIRA, 2006).

Podemos dizer que o aproveitamento de um modelo deriva na concretização de alguma consideração que se pode aplicar (MOLINARI, 2008).

Segundo a SOFTEX (2011), a probabilidade normal é de que o modelo de ciclo de vida acentuado para os testes de software seja colocado de maneira decorada ao modelo seguido para o ciclo de vida do processo de software, com concordância entre os processos. Em outras maneiras, em um ciclo de vida de software incremental, assim sendo, os testes apresentariam seus requisitos acentuados a medida em que os requisitos funcionais serem acertados para o produto e dessa forma consecutivamente inclusive no fim do processo de teste do programa, dessa maneira o produto como o processo com qualidade averiguada fixamente.

Entre os ciclos de vida do teste de software podem ser mencionadososdeterminados modelos clássicos (MOLINARI, 2008):

a) **em cascata ou *waterfall:***modelocomparável ao seu correspondente para o ciclo de vida do programa, ocorrendo de forma retilínea e sequenciada, com a verificação e validação sendo aproveitadas em aprendizadosdiferentes, impedindo por exemplo, que dados técnicos do produto sejam apressados;
b) **modelo V:** é considerado um dos mais comuns e aproveitáveis em projetos alterados, contudo possui como defeito a fraca acoplagem de fasesaceitando por exemplo que erros de comunicação derivem em erros na acepção de requisitos funcionais, o que conforme já exposto em capítulos e subcapítulos antecedentes, será calamitoso para acotação e o

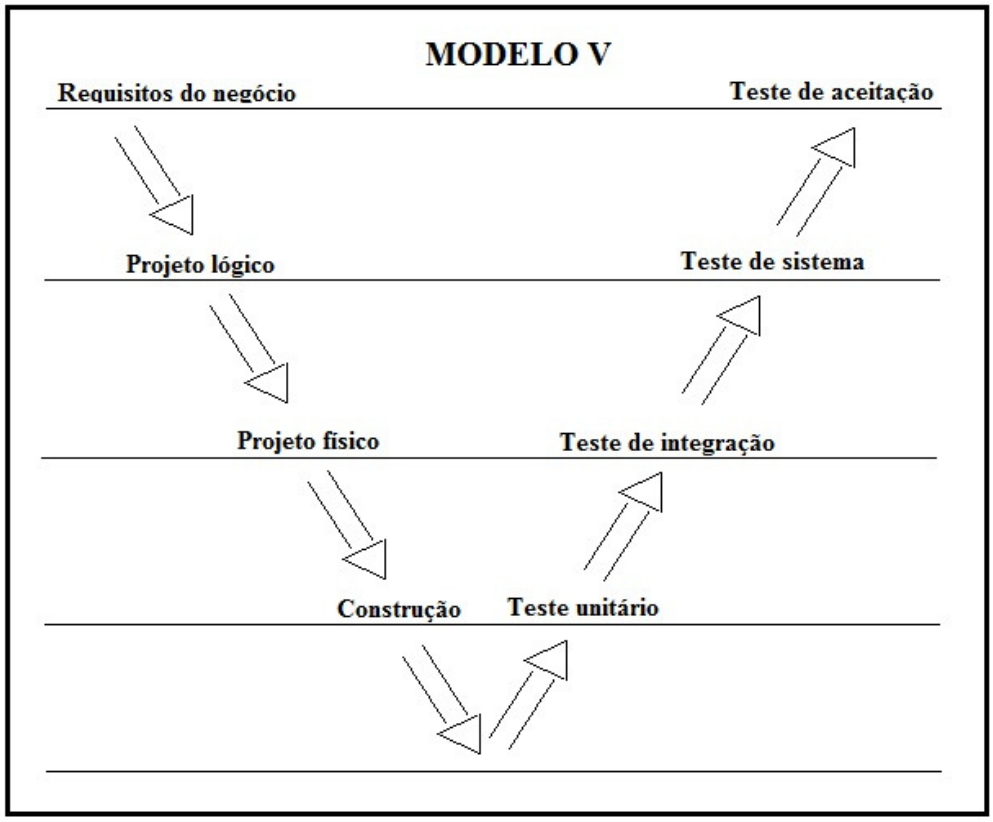
limite do projeto se tais defeitos só forem abrangidos após o ciclo de vida do programa ter andado para fasesposteriores;

c) **modelo espiral:**admite os atos de desenvolver e conter a testes a parte praticada do produto de software. Surgiu com o aparecimento da *UnifiedModelingLanguage* (UML) que propicia iteratividade no incremento de aproveitamentos.

A figura 7 mostra de maneira esquematizada o modelo V, embora sepode advertir o trabalho das rotinas do processo de teste de programas de alcance dos requisitos funcionais ainda os testes de aceitação conforme usuário final (RIOS; MOREIRA, 2006).

Com o alvo de mostrar o ciclo de vida do processo de testes de software de maneira mais explicativa, Rios e Moreira (2006) indicaram o modelo 3P x 3E, o qual trabalha determinadas rotinas do processo de teste consecutivamente e de maneira paralela, ainda que outros trabalhos correm em sequência de forma parecida ao modelo em cascata. A figura 7 mostra essa composição, sendo aceitável a visão de como determinados serviços se agregam.

Figura 7 - Modelo V para ao ciclo de vida do teste de software.



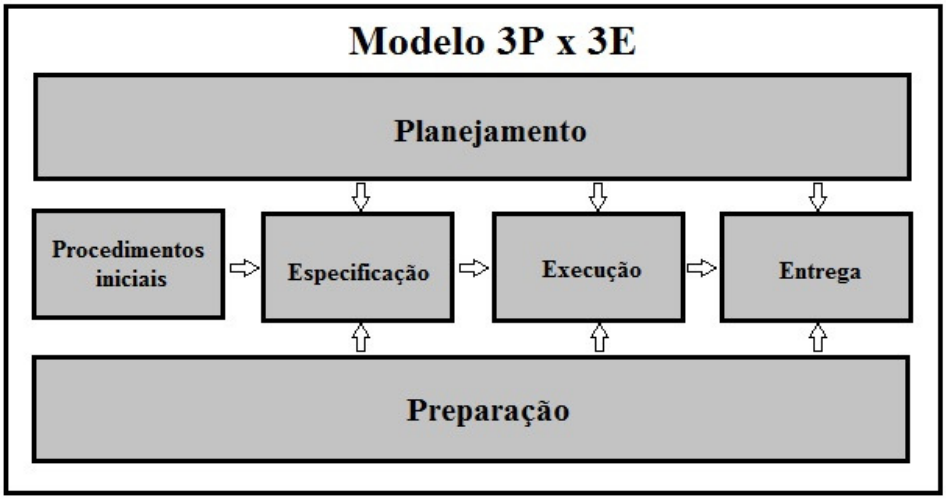
Fonte: Rios e Moreira (2006).

Apesar segundo Rios e Moreira (2006), na figura 7 é aceitável advertir que o modelo 3P x 3E admite abranger como os trabalhos de projeto e de elaboração dos testes acontecem consecutivamente, ainda que os algoritmos iniciais, particularização, implemento e oferecimento ocorre de forma sequencial entre si, contudo paralelas à elaboração e ao projeto. Cada um dosdados acima tem os determinados alvos específicos dentro do modelo 3P x 3E do processo de testes aconselhado (RIOS; MOREIRA, 2006):

a) **procedimentos iniciais:**admite a definição de um acordo de nível de serviço, bem como um alvo da estratégia de testes que será aplicada;
b) **planejamento:** será aposto como recurso de apoio ao processo, portanto tem oalvo de cobrir o perfeito planejamento das tarefas de particularização, cumprimento e oferecimento;
c) **preparação:**temoalvo de apoiar o processo assegurando que bibliografias, ambientes, ferramentas, programas e o pessoal envolvido estejam de modo pleno preparados e disponíveis quando uma etapa de testes for posta em cumprimento;

d) **especificação:** é a fase na qual serão elaborados e revisados os roteiros e casos de testes;
e) **execução:** os roteiros e casos de testes são postos em cumprimento, empregando os recursos já oferecidos pela tarefa de preparação. O uso de ferramentas de automação poderá ser um dos recursos aproveitáveis;
f) **entrega:**permite a finalização dos testes, com bibliografia dos resultados obtidos que possam melhorar o processo produtivo, conservação da documentação do produto e disponibilização do programa para os clientes finais. Um relatório gerencial das melhorias com base em acordos ou não acordos também poderá ser formado.

Figura 8 – Modelo 3P x 3E para o ciclo de vida do processo de testes.



Fonte: Rios e Moreira (2006).

Apesar de haverem outros modelos acabados de ciclo de vida dos testes de software, uma das soluções que podem ser aplicadas é a invenção de um modelo acomodado às propriedades da companhia de incremento ou do produto a ser examinado. Podemos dizer que modelos acabados podem ter suas competências subutilizadas conforme a complicação de um produtoousuperutilizadapor o programa se comum. Os modelos acabados podem sercomprometidos pelos recursos estruturais que a companhia de incremento aprontará para o processo de testes, podendo ser insignificantes ou abertos tanto em grupos quanto em aparelhamentos e programas de base. Para invenção de um modelo adequado é preciso do comando de bons

exercícios em testes de software, a informação daquilo que se pode dar ênfase e abordar com os testes, no final, é indispensável à assimilação do mais prefeito andamento para disseminar o modelo inventado (MOLINARI, 2008).

A invenção de um modelo adequado não é serviço comum, porém, é duradouro perante algumas regras. Para isso Molinari (2008) aconselha a usabilidade do metamodelo, podendo facilitar a invenção de modelos próprios. Essa facilidade acontece analisando que tecnologias, procedimentos e fases são formas diferentes de testes. Estas formas são espalhadas em mediadas no metamodelo. Essas mediadas compostas em:

a) **meta:** para o metamodeloobedece a tipos de testes que têm objetivos diretos como questões funcionais, de desempenho ou de cliente. É considerado a equipe de testes que dita o “o quê” na qualidade do programa;
b) **momento:**admite os tipos de testes que têm um momento certo e adequado para sua ocorrência, como os testes unitário, de integração ou de sistemas. Estabelece da equipe de testes que gera “o quando” para a qualidade do produto;
c) **técnica:** representa a equipe de tipos de testes que diferenciam técnicas diferentes de abordagens do teste e o seu enfoco quanto ao produto, como por exemplo os testes caixa-preta ou caixa-branca. Compreendea equipe de testes que decide “o como” para oalcance da qualidade no programaexaminado;
d) **ambiente:**permite conceitos e meios físicos que geram o local da exercício dos testes, como por exemplo, atmosfera de testes para internet ou usabilidade de servidores mainframes. Incorpora os testes que apontam “o onde” na procura da qualidade bem-sucedida.

O aproveitamento do metamodelo acontecerá com a análise das obrigações de testes, depois estão dispostas dentro do metamodelo sendo feitas derivações que inventam o modelo adequado em uma forma de composição, devendo permitir que os tipos de testes forem conectados aos andamentos acomodados do projeto em que devem ser postos em exercícios (MOLINARI, 2008).

Insensível ao modelo a ser seguido para o ciclo de vida do teste de software é indispensável que os produtos sejam pesquisados, os padrões de testes detalhados, disseminando o modelo de menor precipitação, maior performance produtiva e ampliação na qualidade. É indispensável determinar a ênfase na passagem, é a usabilidade de cálculosconstantes da cadeia produtiva, assim progredir perante as obrigações ou defeitos que podem ser considerados.

Os subcapítulos futuros planejarão as etapas de testes, sendo dados significantes na composição do ciclo de vida do teste de software.

3.3.1 Fase de Unidade

O desenvolvedor aproveita testes no menor componente do programa, comparando as consequências, alcances funcionais e a coerência de cada elemento aplicado no produto, assim sendo, de desempenhos, algoritmos ou metodologia (DELAMARO; MALDONADO; JINO, 2007).

O aproveitamento dos testes acontece por ambiente do incremento de um falso-controlador, ainda apontado de *driver*, que cobre o comando de um falso-controlado, designado de *stub*. Os dois dados têm o alvo de cobrir testes que estudem os alcances ou condutas do alvo do software a que os elementos se aproveitam. Com isso casos de testes são atribuídos pelo falso-controlador ao falso-controlado e as consequências admiráveis são postos em bibliografias. Como ficam colocados dentro do argumento de incremento, os dados do teste de unidade são analisados componentes do programa, simulando ainda custos indiretos do incremento do produto (PRESSMAN, 2006).

3.3.2 Fase de Integração

Abrange a etapa na qual o programador testa a acoplagem dos elementos do programa, em meio a eles o banco de dados, outros códigos, bibliotecas acessórias ou modulações (PRESSMAN, 2006).

A iteratividade entre os elementos do programa pode ser analisada para permitir que unidos agem de maneira natural, sem oferecer falhas. Sendo

um teste que agencia aberto informação estrutural do programa, assim a obrigação de ser concretizado com preferência pelo adequado grupo de incremento (DELAMARO; MALDONADO; JINO, 2007).

A etapa de integração pode acontecer por múltiplos ambientes, com evidência as determinadas afirmações (PRESSMAN, 2006):

a) ***big-bang*:** na qual o software tem seus elementosconectados com base em uma acordo anterior e todos eles são avaliados de uma só vez, derivando em uma equipe de erros que de maneira inevitável conduzirá a um resultado confuso, gerando a necessidade de várias assentadas de testes de integração até que o produto se torne comum;
b) **incremental:** em que o produto de programa tem sua acoplagem avaliadagradativamente, à medida que os elementos vão sendo ampliados, produzindo umarranjo menor de erros, antecipando as fases de correção e as novas fases de testes de integração. Os testes de integração estão subdivididos entre as táticas descendentes ou ascensionárias, desse modo que a inicialacontece com a integração de maneira descendente na camada de conexãodos modulação ou elementos, enquanto que a secundáriaacontece de forma ascendente em partida dos elementos mais baixos na ala de camada até os mais altos.

Impassível da tática empregada no testes de integração, o efeito adquirido pode ser um produto conectado, comum e acabado para ser contido à fase futura de testes (PRESSMAN, 2006).

3.3.3 Fase de Sistema

A etapa dos testes de sistema admira o produto agora com seus elementos agregados e estes são praticados com a averiguação de que cada um compete seus desempenhos de forma acabada e aceitável (PRESSMAN, 2006).

Esse exame da integralidade das funcionalidades é alcançada com o implemento dinâmico do programa, comparando no ambiente de entradas válidas se os efeitos arranjados nas saídas combinam com o desejado em

cada entrada válida e se continuam trabalhando corretamente perante as entradas inválidas (ROCHA; MALDONADO; WEBER, 2001).

Segundo Pressman (2006), múltiplos testes podem ser aplicados para a concretização dos testes de sistemas, podendo ser mencionado em meio a eles o teste de recuperação, de segurança, de estresse e de performance. Ainda é uma propriedade dos testes de sistemas a probabilidade de que aconteçam dentro de várias tecnologias e determinadas etapas, como por exemplo o teste de performance, sendo efetuado na fase de teste unitário.

3.3.4 Fase de Regressão

É a etapa em que o software agora está em usabilidade pelo usuário final e depois de uma manutenção, sofrerá por testes afirmando que tanto as funções propagadas por outra operação no produto, tudo quanto os demais soluções que ficaram em completo estágio funcional, continuem inteiramente funcionais no final do procedimento de testes (DELAMARO; MALDONADO; JINO, 2007; PEZZÈ; YOUNG, 2008).

Conforme Pressman (2006), o teste de regressão pode ser acatado ainda um dos tipos de testes aproveitáveis à etapa de testes de integração, por garantir que elementos ou desempenhos sejam praticadas particularmente para afirmar que seu estágio funcional esteja coeso com o projeto.

Ainda é aceitável analisar que a regressão é um tipo de teste que se pode aplicar à prática funcional ou caixa-preta (BASTOS et. al., 2007), em que o programa será contido a testes que pratiquem as funcionalidades esquematizadas para o produto em geral, para alterações, também para alvos acentuados na estrutura de origem.

Insensível à visão aproveitada, os testes de regressão são essenciais para a sustentação da qualidade do programa (BASTOS et. al., 2007; DELAMARO; MALDONADO; JINO, 2007; PRESSMAN, 2006), sendo que as conservações constantes firmemente comprometem o software por determinadas falhas que atribuem ao produto (SOMMERVILLE, 2003). Este é um dado que faz da etapa de regressão essencial para o procedimento de testes em produtos que sofrem fortes adequações, assim sendo, programas com requisitos funcionais determinados por legislações fiscais, contribuintes ou

trabalhistas, campos estes que fixamente suportam adulterações, sendo fundamental o serviço dos testes de regressão visando progresso na qualidade dos produtos.

O aproveitamento do teste caixa-preta ou funcional na etapa de regressão, já divulgado, lança lucros abundantes pela qualidade do software, aliás, já foi mostrado em capítulos antecedentes, os testes manuais possuem alcances que pretendem a vincular a aptidão funcional dos grupos de testes perante as dimensões dos programas a quantia de requisitos funcionais para tornar valida as conservações alcançadas.

4 TRABALHOS CORRELATOS

Os trabalhos correlatos, contados a seguir, apresentam métodos, técnicas e padrões que formam uma base de conhecimentos para a concretização do presente trabalho.

4.1 TESTE DE SOFTWARE UMA NECESSIDADE NAS EMPRESAS

Trabalho proposto por Cristiane Carline para conclusão da graduação em Ciência da Computação pela Universidade do Rio Grande do Sul.

Para Sommerville (2003), alcançar um alto nível de qualidade de produto ou serviço deve ser presentemente o maior objetivo das coordenações, pois o mercado não aceita mais que se ofereça produtos de pouca qualidade.

Bartié (2002) define qualidade de software como “um processo ordenadoque focaliza todas as etapas e componentes produzidos com o objetivo de cobrir a conformidade de processos e produtos, evitando e abolindo falhas”.

Para se obter a qualidade, é indispensável definir e estabelecer umaestrutura de algoritmos e de padrões organizacionais que devem ser aproveitados durante o desenvolvimento do programa que transportam ao produto final de altaqualidade.

A norma ISO/IEC 9126 aproxima seis atributos-chave que um software de qualidade deve haver (PRESSMAN, 2006 apud PINTO JÚNIOR, 2009):

Funcionalidade: o programa deve operar adequadamente e acatar as necessidades apontadas;

Confiabilidade: o software deve trabalhar por determinado período de tempo livre de erros e defeitos;

Usabilidade: o programa deve ser abrangido e utilizado facilmentepelo cliente;

Eficiência:o software deve alcançar seu melhor comportamento utilizandoseus recursos da melhor maneira;

Manutenibilidade: o programa deve realizar alterações e reparos sem grandes problemas;

Portabilidade:o software deve ser espaçado facilmente de um ambiente (plataforma) para outro.

Outros atributos que contribuem para se ter um programa de qualidade são “profissionais experientes e bem adestrados, metodologias e ferramentas acomodadas, participação constante dos clientes finais, bom entendimento do problema e modelagem da solução flexível ao longo prazo” (BARTIÉ, 2002).

4.2 O TESTE DE SOFTWARE NO MERCADO DE TRABALHO

Apresentado em 2010 porFilipe Bernardes Barbosa, o trabalho foi um dos requisitos para obtenção do título de Especialista em Computação.

A literatura define Teste de *Software* (TS) como umadetecção de bugs no sistema. O termo *bug* palavra inglesaque significa “inseto” tem sido empregado pelos engenheirospara considerar pequenos defeitos nas máquinas há mais de umséculo. Thomas Edson, autor da lâmpada incandescente,contou que um bug havia ocasionado problemas de leitura em seufonógrafo.

O primeiro bug em computadores provavelmente ocorreu em1947. Engenheiros do Harvard Mark I, primeiro computadordigital automatizado de grande escala, acharam uma traça emseus cabeamentos, o que originou erros nos cálculos da máquina,prenderam-na no livro de registro e rotularam-na como o“primeiro bug” encontrado.

Na década de 70, os testes eram executados pelos própriosdesenvolvedores: “A partir dos anos 1970, as atividades de TS eram efetuadas pelos própriosprogramadores ou analistas de sistemas. Nos anos seguintes,os clientes passaram a produzir quando um sistema poderiaser entregue à produção”.

4.3 O PROCESSO DE TESTE DE SOFTWARE

Artigo elaborado pelo pesquisador Josenilson dos Santos Silva.

O Processo de Testes de Software simula uma estruturação de fases, atividades, artefatos, papéis e cargos que tem o objetivo de padronizar os trabalhos, além de elevar ao máximo a organização e monitoramento dos projetos de testes. Portanto, o objetivo do processo de teste é abastecer

informação para garantir a qualidade do produto, disposições e processos para uma atividade de teste. Além disso, o processo de teste busca garantir que nenhum caminho crítico do processo seja perdido, ou seja, que todas as atividades sejam realizadas na ordem adequada.

Vimos que o teste é muito importante para um software, porém á condições em que o teste precisa ser obstruído. A interrupção deve acontecer assim que o custo dos testes superar acotação do defeito para o negócio.Sempre existirá um ponto de equilíbrio para definir a suspensão dos testes, nem sempre é simples de encontrar esse ponto, mas muitos técnicos informam que o melhor a serfeito, é interromper os testes quando o intervalo de ocorrência de falhas começarem a aumentar muito, por exemplo, de horas para dias. Para isso usa-se os termos under test e over test, que significa “falta de teste” e “excesso de teste”.

Os testes sempre diminuem a probabilidade de defeitos encontrados no software, mesmos que nenhum erro seja encontrado, isso não garante a conformidade. Os testes podem ser bem cansativos, mesmo utilizando as ferramentas de automação, com isso é impossível se afirmar que tudo foi testado. Muitas vezes as empresas usam como testador os analistas de sistema, um erro, pois para testar um software deve ter profissionais diplomados, usando metodologia adequada, em ambiente acertado e ferramentas de automação na maioria dos casos.

Quanto mais rápido se inicia o teste menor será o custo, Barry Boehm, em seu livro Software engineering, afirma que o custo de um erro detectado nos requisitos é X; se encontrado na fase de teste, seu custo passa a ser 10X; e se encontrado na produção, 100X. Com isso, como citado antes, quanto mais rápido encontrarmos falhas, mais barato fica para ajustar. Estudos estimam que de 80% dos defeitos, 20% pertence ao código, ao identificar essa parte de erro, pode se dar prioridadeao teste enquanto outros erros são achados.

É importante advertir que os testes precisam ser revisados com frequência, pois se os mesmos testes sejam aproveitados repetidamente, em determinado momento eles deixaram de ser favoráveis, com isso não se encontrará mais falhas, prejudicando então a eficiência que deverá ter o software. Porém com todos esses princípios é importante sempre que o

programa atenda sempre as necessidades do cliente, abreviando, um software deve ser testado sempre.

4.4 A IMPORTANCIA DA ATIVIDADE DE TESTE NO DESENVOLVIMENTO DE SOFTWARE

O artigo foi proposto em 2013 pela autora Karla Pires de Souza.
A atividade de teste é uma das etapas do ciclo de desenvolvimento de software e tem o objetivo de relatar possíveis falhas existentes no sistema para que estes sejam resolvidos. Nesta fase verifica-se se aconduta do sistema está de acordo com o mencionado nos requisitos levantados junto ao usuário. A partir desta atividade pode-se diagnosticar o grau de qualidade do sistema. O principal objetivo da realização do teste de software é reduzir a esperança de ocorrência de erros quando o sistema estiver em produção (MOLINARI, 2012).

Na década de 70 a execução do teste nos softwares era feita pelos próprios desenvolvedores dos sistemas, a atividade era vista como um trabalho secundário, sem muita gravidade, feito apenas se o prazo de entrega e custo do produto aceitasse. Com o passar do tempo, devido à concorrência existente no mercado e ao aumento da complicação dos sistemas, o nível de cobrança por qualidade aumentou e com isso a necessidade de testes mais ativos.
Para Pressman (2002), "teste de software é um elemento crucial da garantia de qualidade de software e representa a revisão final da especialização, projeto e geração de algoritmo". Por ser a última etapa, antes da entrega ao usuário, à fase de teste tem a carga de encontrar os defeitos inseridos no decorrer do projeto.

Pressman (2002) determina qualidade como a satisfação de requisitos funcionais e de performance explicitamente declaradas, normas de desenvolvimento explicitamente documentadas e características subentendidas que são esperadas em todo software desenvolvido profissionalmente. O conceito de qualidade é alterável, para a equipe de desenvolvimento de software um produto possui qualidade quando se suporta conforme está descrito nos requisitos, já para o usuário um produto terá qualidade quando este atender suas obrigações. Devido ao aumento da cobrança por qualidade,

várias normas, metodologias ou modelos (ágil e RUP, por exemplo) e órgão reguladores como CMMI, MPS.Br e ISO foram criados.

5 DESENVOLVIMENTO DE UM MODELO DE DERIVAÇÃO FUNCIONAL PARA VALIDAÇÃO DE SOFTWARE NA FASE DE INTEGRAÇÃO

O trabalho consiste na aplicação de um modelo de derivação funcional para validação de software em produção.

5.1 METODOLOGIA

A metodologia aplicada para a elaboração do trabalho foi distribuída, contemplando desde o levantamento de bibliografias, passando por estudos realizados sobre as informações levantadas, bem como pela aplicação de técnicas, métodos e ferramentas pesquisadas.

5.2 RESULTADOS OBTIDOS

Considerando as leituras realizadas para a elaboração deste trabalho, podese averiguar que a preocupação com a qualidade de software tem se tornado cada vez maior em função do grande volume de softwares produzidos e das cobranças dos usuários que almejam antes de tudo softwares seguros, eficientes e de boa qualidade.

O teste de software é uma das atividades que busca colaborar para amelhoria da qualidade de um sistema. O objetivo dos testes é desvendar a presença de defeitos no software, para que estes possam ser corrigidos antes de serem entregues ao usuário.

Este trabalho apresentou os diferentes tipos de testes que podem serexecutados para se obter um software de qualidade. Cabe cada coordenação analisar quais são os testes mais adaptados para o sistema que está sendodesenvolvido.

A escolha do conjunto de testes esta sujeito muitas vezes das restrições dequalidade, do preço, do prazo e dos recursos disponíveis para o desenvolvimento de um sistema.

Identificar e remover tantas falhas quanto possível é um objetivo durante o desenvolvimento, mas encontrar todas os defeitos é quase impossível. Os testes não podem existir para sempre. Os softwares devem ser entregues

quando atingem um nível apropriado de funcionalidade e qualidade (PEZZÈ e YOUNG, 2008).

Pode-se dizer que este trabalho é de extrema importância, pois é assistencial no conhecimento da atividade de teste de software e dá uma idéia de como as corporações estão realizando os testes. E o resultado do trabalho foi aceitável, pois foi adquirido o objetivo proposto.

As dificuldades encontradas para a realização deste trabalho de conclusão de curso foi a de encontrar empresas organizadas e o pouco material encontrado sobre testes automatizados e metodologias ágeis.

Como trabalhos futuros sugerem-se fazer visitas as empresas dedesenvolvimento de software para averiguar como são feitos os testes; implantar a atividade de teste em uma empresa de TI para avaliar e comparar os resultados obtidos com a implantação; estudar ferramentas de testes automatizados.

Foram relatados os diversos casos de teste no ambiente virtual da Universidade do Extremo Sul Catarinense com os seguintes casos de testes:

# Funcionalidade	TRM#
1 Efetuar Login	TRM1
2 Selecionar Graduação	TRM2
3 Recuperar senha	TRM3
4 Selecionar disciplina	TRM4
5 Selecionar meus recursos	TRM5
6 Selecionar salas de aula	TRM6
7 Selecionar disco virtual	TRM7
8 Selecionar Monitoria	TRM8
9 Selecionar parla	TRM9

Com os ciclos de desenvolvimento:

Ciclo 1	Ambiente		COD
TRM	SO	Browser	Versão
TRM1	WIN	IE	v1
TRM1	WIN	FX	v2
TRM1	WIN	CR	v3
TRM1	LIN	FX	v4

Ciclo 2	Ambiente		COD
TRM	SO	Browser	Versão
TRM2	WIN	IE	v1
TRM2	WIN	FX	v2
TRM2	WIN	CROME	v3
TRM2	LIN	FX	v4

Ciclo 3	Ambiente		COD
TRM	SO	Browser	Versão
TRM3	WIN	IE	v1
TRM3	WIN	FX	v2
TRM3	WIN	CROME	v3
TRM3	LIN	FX	v4

Ciclo 4	Ambiente		COD
TRM	SO	Browser	Versão
TRM4	WIN	IE	v1
TRM4	WIN	FX	v2
TRM4	WIN	CROME	v3
TRM4	LIN	FX	v4

Ciclo 5	Ambiente		COD
TRM	SO	Browser	Versão
TRM5	WIN	IE	v1
TRM5	WIN	FX	v2
TRM5	WIN	CROME	v3
TRM5	LIN	FX	v4

Ciclo 6	Ambiente		COD
TRM	SO	Browser	Versão
TRM6	WIN	IE	v1
TRM6	WIN	FX	v2
TRM6	WIN	CROME	v3
TRM6	LIN	FX	v4

Ciclo 7	Ambiente		COD
TRM	SO	Browser	Versão
TRM7	WIN	IE	v1
TRM7	WIN	FX	v2
TRM7	WIN	CROME	v3
TRM7	LIN	FX	v4

Ciclo 8	Ambiente		COD
TRM	SO	Browser	Versão
TRM8	WIN	IE	v1
TRM8	WIN	FX	v2
TRM8	WIN	CROME	v3
TRM8	LIN	FX	v4

Ciclo 9	Ambiente		COD
TRM	SO	Browser	Versão
TRM9	WIN	IE	v1
TRM9	WIN	FX	v2
TRM9	WIN	CROME	v3
TRM9	LIN	FX	v4

Diversos dados de qualidade no Windows com o internet Explorer, mozila Firefox, Google chrome e Linux, relatando se foi aprovado cada ambiente de teste, conforme as versões.

5.3 APENDICE

A busca pela qualidade de software

Alfredo M. Possidonio[1]

[1]Instituto de Informática – Universidade do Extremo Sul Catarinense (UNESC)

Criciúma – SC – Brasil

{possidonio,alfredo}alfredomartinsp01@gmail.com

Abstract. *This article shows that software testing is the process of running programs in order to find fault. A test diagram has to be well detailed, will have as knowledge in the diagram of the test diagram, the project manager, software engineers, or test connoisseurs. Software testing is not the safety of software that is fully fault-free, so it is imperative and very important that you have a test throughout your software planning because it greatly reduces defects and helps with costs.*

Resumo. *Este artigo mostra que o teste de software é o processo de executar programas, com o objetivo de achar falhas. Um diagrama de teste tem que ser bem detalhado, terá como conhecimento no plano do diagrama de teste, o gerente de projeto, engenheiros de software, ou conhecedores em testes. O teste de software não é a segurança de um software plenamente livre de falhas, sendo assim é indispensável e muito importante que se tenha teste em todo o planejamento de um software, pois diminui de forma ampla os defeitos e ajuda em relação aos custos.*

1. Introdução

O teste de software é o processo de implemento de um produto para determinar se ele abordou suas particularizações e trabalhou corretamente no ambiente para o qual foi esquematizado. O seu objetivo é revelar defeitos em um produto, para que as causas desses defeitos sejam identificadas e possam ser ajustadas pela equipe de incremento antes da entrega final. Por conta desse atributo das atividades de teste, dizemos que sua natureza é “destrutiva”, e não “construtiva”, pois visa ao acrescentamento da certeza de um produto através da apresentação de seus problemas, porém antes de sua entrega ao cliente final.

Particularmente, o teste de software pode ser abrangido através de uma visão intuitiva ou mesmo de uma maneira protocolar. Existem hoje várias definições para esse julgamento. De uma forma simples, testar um software significa averiguar através de uma execução controlada se a conduta corre de acordo com o mencionado. O objetivo principal desta atividade é revelar o número máximo de defeitos dispondo de pouco empenho, ou seja, mostrar aos que desenvolvem se os frutos estão ou não de acordo com os modelos estabelecidos.

2. Engenharia de software

A engenharia de software como “o entendimento e o emprego de adaptados princípios de engenharia a fim de obter softwares econômicos que sejam livres de falhas e que trabalhem corretamente em máquinas reais”. Enquanto que, para o Institute of Electrical and Electronics Engineers (IEEE) a engenharia de software advém na arte de obedecer para acrescentar, a operação e amparo do software.

A engenharia de software como sendo uma tecnologia em categorias que agrega processo, procedimentos e ferramentas para o desenvolvimento de 15 softwares (programas, dados e documentos) com o alvo de fornecer uma estrutura para a construção de software com altiva qualidade.

3. Os processos de testes

O Processo de Testes de Software representa uma estruturação de fases, atividades, artefatos, papéis e cargas que tem o objetivo de padronizar os trabalhos, além de maximizar a aparelhamento e monitoramento dos projetos de testes. Portanto, o objetivo do processo de teste é fornecer o conhecimento para garantir a qualidade do produto, decisões e processos para uma atividade de teste. Além disso, o processo de teste busca garantir que nenhum passo crítico do artifício seja esquecido, ou seja, que todas as atividades sejam realizadas na ordem adequada.

Vimos que o teste é muito importante para um software, porém á condições em que o teste precisa ser interrompido. A interrupção deve acontecer assim que o custo dos testes superar o custo da falha para o negócio.Sempre haverá um ponto de equilíbrio para definir a interrupção dos testes, nem sempre é fácil de encontrar esse ponto, mas muitos especialistas informam que o melhor a ser feito, é interromper os testes quando o intervalo de ocorrência de defeitos começarem a aumentar muito, por exemplo, de horas para dias. Para isso usa-se os termos under test e over test, que significa “falta de teste” e “excesso de teste”.

4. Ciclo de vida do processo de teste

Como mencionado antes, quanto antes os testes iniciarem, mais barato será corrigir os defeitos encontrados.Para que isso aconteça todo teste deve ter um ciclo de vida.

4.1 Processos

Etapas do teste:

1- Procedimentos inicias

Nesta etapa deverá ser afundado um estudo dos requisitos do negócio que dará origem ao sistema de informação a ser testado, de modo a garantir que o mesmo esteja completo e sem nenhuma ambiguidade. No final desta etapa é elaborado o GOT – Guia Operacional de Teste.

2- Planejamento

Consiste em elaborar a Estratégia de Teste e o Plano de Teste a ser utilizados de modo a minimizar os principais riscos do negócio e abastecer os caminhos para as próximas etapas.

3- Preparação

Nesta etapa, o objetivo básico é preparar o ambiente de teste (equipamentos, pessoal, ferramentas de automação, hardware e software), para que os testes sejam executados perfeitamente.

4- Especificação

É elaborado e revisado os casos de teste e os de roteiros teste. Os casos de teste e os roteiros de teste devem ser elaborados de modo dinâmico durante o decorrer do projeto de teste.

5- Execução

Os testes deverão ser efetuados de acordo com os casos de teste e os roteiros de teste. Devem ser usados scripts de teste, caso seja empregada alguma ferramenta de automação de testes.

6- Entrega

Esta fase é onde acontece a entrega do software depois de ter passado por todas as fases, com objetivo de tornar mínimo todas as chances possível de erros (bugs). Essa representação de etapas de teste é uma estrutura básica, existem outras, porém a idéia é sempre a mesma mostrada à cima. Para que o teste der certo, desde o início é preciso ter uma equipe reservada para o trabalho, a equipe deve ser dividida em cada fase de teste para que o teste seja bem-sucedido.

5. Ambiente de teste

O ambiente de teste é toda a infraestrutura onde o teste será desenvolvido, abrangendo configurações de hardware, software, ferramentas de automação, equipe envolvida, aspectos organizacionais, suprimentos, rede e documentação. Seu alvo é proporcionar a formatação de testes em categorias conhecidas e controladas. A figura a baixo representa o ambiente e seus elementos:

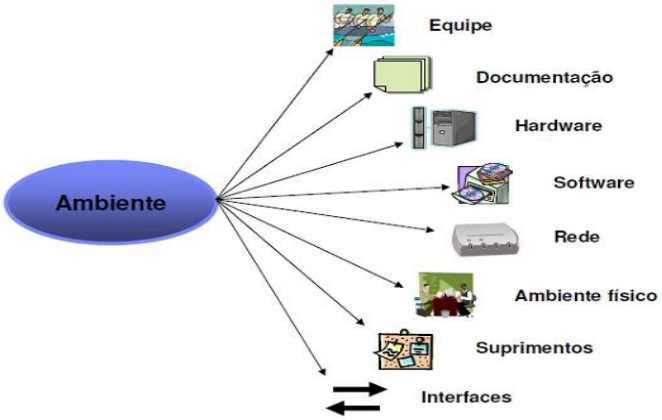


Figure 1. O ambiente de testes.

Figura 2 engenharia de software em camadas

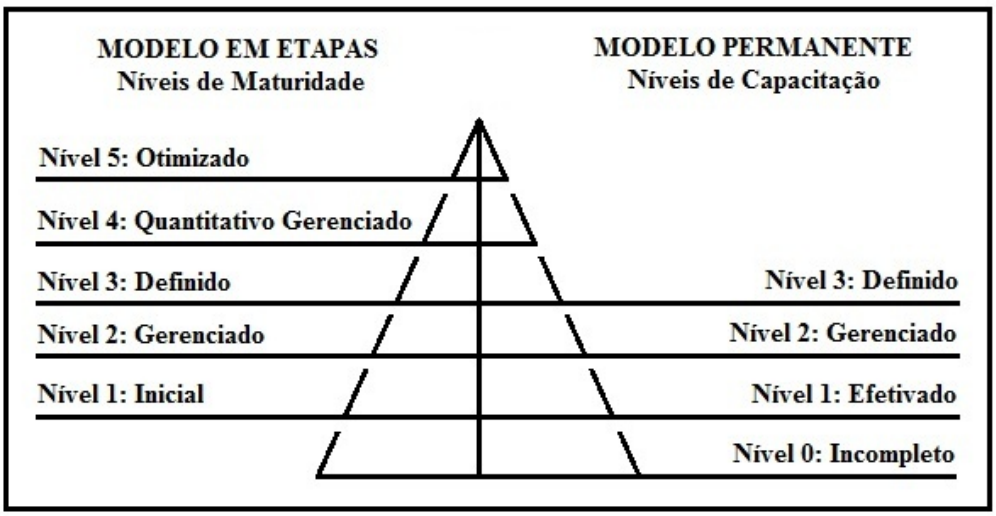


Figura 3 – Modos e níveis do CMMI.

6. Técnicas de testes

Existem várias maneiras de testar um software, todas as maneiras terão um mesmo objetivo. Técnica é o processo que vai assegurar perfeito funcionamento de alguns aspectos de software ou de sua integração. Separado em dois grupos temos as principais técnicas de testes, são elas, Estrutural e Funcional.

- Estrutural O Teste estrutural também conhecido como teste da caixa branca, tem por objetivo testar o código fonte, testar cada linha de código possível, testar os fluxos básicos e os alternativos. Nesse teste podemos classificá-los em alguns itens como: Teste de estresse: Verifica como o sistema é executado com determinados volumes de dados. Teste de execução: Se o sistema atinge o nível desejado de eficiência.
- Teste de recuperação (contingência): Se o sistema é capaz de retornar o nível anterior antes do defeito. Teste de operação: Se o sistema opera conforme sua documentação.
- Teste de conformidade: Se o sistema foi desenvolvido conforme padrões e procedimentos. Teste de segurança: Se o sistema está protegido conforme normas e políticas do aparelhamento.
- Funcional O teste funcional conhecido como teste da caixa preta, é baseado na análise funcional do software ele garante que os requisitos funcionem conforme o especificado, ele não se preocupa na forma como ele foi implementado, é inserido alguns dados.

7. Ferramentas de suporte de teste de software

Ao longo dos anos, à medida que o teste vem ganhando mais espaço no mercado, diversas ferramentas têm surgido visando à melhoria no gerenciamento e na execução das atividades voltadas para o Teste de Software.

A utilização de ferramentas é um importante aliado para avaliar se o software está exposto a vulnerabilidades. Independente da função, o objetivo gira em torno da simplificação das atividades, controle, e, principalmente, garantia da qualidade em cada uma das etapas de um Processo de Teste. Outro fator relevante é que elas permitem sobretudo, tornar mínimo os riscos, inconformidades e uma série de defeitos que podem ser detectadas durante a execução dos testes.

Além de simplificar as atividades, a adoção dessas ferramentas pode aumentar consideravelmente o número de falhas encontradas, e consequentemente agilizar os testes, aprimorando a qualidade das aplicações. Elas também podem levar a melhorias na confiabilidade do software, tornando os testes mais ativos e produtivos.

As ferramentas são classificadas de acordo com as atividades de teste que elas aturam. Seja para o gerenciamento ou para execução de um teste, elas são categorizadas de acordo com os tipos de teste que executam. Os tipos de teste, unitário, segurança, sistema, ambiente, integração e desempenho, possuem diferentes funções e cada um deles visa avaliar uma determinada característica do software. Um fator interessante é que há ferramentas de suporte que contemplam praticamente todos os tipos de teste viventes.

Nos dias atuais, algumas ferramentas suportam várias atividades, como gestão dos testes, automação e controle dos defeitos, enquanto outras suportam apenas uma. Mesmo assim, apresentá-las intitulando uma ou outra como “a melhor” não é possível. Esse título vai depender de uma série de fatores como, por exemplo, investimento disponível, aplicabilidade, tipo de software a ser testado, tamanho da equipe e esperança de retorno. Em função disso, é interessante analisar as opções disponíveis para que seja possível definir qual se enquadra melhor às suas esperanças.

8. Diferença entre ferramentas e técnicas de testes

Ferramentas e técnicas de testes são muito importantes para o desempenho do processo de teste. A pessoa que irá testar precisar conhecer primeiro as técnicas de teste para depois entender quais ferramentas devem ser usadas com cada técnica. É tão importante o testador conhecer as técnicas de teste, o quanto para um médico ter primeiro a informação da doença, para só então, poder fazer medicação correta conforme o problema apresentado ao doente. Assim funcionam com o testador, primeiramente obter os conhecimentos necessários das técnicas, para então usar a ferramenta adequada.

Algumas técnicas são manuais e outras automáticas; algumas fazem testes estáticos, e outras, os dinâmicos. Outras avaliam a estrutura do sistema e sua função, na questão das ferramentas algumas são extremamente técnicas e demandam uma noção afundada de programação e do sistema a ser testado. Outras são genéricas por natureza e podem ser úteis para quase todos os profissionais do ramo de teste. Existem ferramentas para criar informações de teste e para a validação dos testes.

9. Elaboração do teste

Na fase de elaboração do teste, a atividade principal é a elaboração dos cenários de testes, que serão testados na fase de execução. O cenário de teste é um caminho que deve ser seguido ou a situação que deve ser testada, o cenário que serve de base para o teste é descrito numa especificação do sistema, por exemplo, na Unified Modeing Languge (UML), é o caso de uso. O caso de teste é o cenário a ser executado para verificar se o que foi especificado está devidamente disseminado.

10. Documentação do teste

Um analista de teste gasta em média 50% a 60% do seu tempo em documentação do teste. A norma IEEE 829-19982 é um padrão que define a estrutura de vários documentos relativos a testes de software. A escolha dos documentos que darão o suporte suficiente aos testes depende da complicação deste, da equipe de testes disponível e do tempo dedicado aos testes durante o desenvolvimento do programa. Os documentos definidos pela norma cobrem as tarefas de planejamento, especificação ou elaboração e apreciação dos resultados.

11. Plano de teste

O plano de testes é o documento principal dos testes de software. Nele estão contidas informações importantes sobre as partes envolvidas no teste, os objetivos do teste, as partes do programa a serem testados, os critérios de aceitação e os passos necessários para executar os testes.

Cada plano de testes possui descrição de um ou mais casos de teste. Um caso de teste compreende no mínimo um conjunto de ações a serem executadas e um critério de aceitação, que diz se o teste foi bem-sucedido ou não. Na maioria das vezes os casos de teste possuem também critérios de entrada ou requisitos, e vem acompanhado de uma breve descrição do item a ser testado e dos objetivos que ele busca abordar.

À medida que a disciplina de testes de software foi se profissionalizando nos últimos anos, novas seções foram adicionadas aos

planos de teste, como sumário de alto nível e lições aprendidas. Estas seções tem o intuito de situar o plano de testes e a equipe de testes com relação a outros conjuntos e ao planejamento do projeto totalmente.

12. Especificação do projeto de teste

É o detalhamento da abordagem apresentada no plano de teste que identifica as funcionalidades e as características que serão testadas pelo projeto. Estes documentos também apontam os casos e os procedimentos de teste, se haverem, e apresenta os critérios de aprovação.

13.Especificação do caso de teste

Uma Especificação de Casos de Testes deve contemplar os casos de teste válidos e inválidos. Casos de teste inválidos são aqueles que exercitam a entrada de valores inválidos nos campos do aparelho em teste. Segundo Bastos et al. (2007), a utilização dos casos de uso como referência para a geração de casos de testes é uma regra bem comum no mercado. Nem sempre existe tempo/saídas para utilização das técnicas e os testes são focados na validação de códigos de negócio.

14. Especificação do procedimento de teste

Identifica todos os passos necessários para a operação do aparelho e o exercício dos casos de teste especificados, onde cobre o projeto de teste planificado. Os procedimentos de teste formam um documento separado, que se espera que seja seguido cada passo, sem problemas.

15. Execução dos testes

Conforme mencionado ao longo desse artigo que o teste é extremante importante e devem ser testados em todas as etapas do ciclo de vida do processo de desenvolvimento de software, desde os requisitos até o teste de aceitação, na fase de homologação e liberar o software para fabricação. O projeto de teste deve ser desenvolvido em paralelo e está integrado ao projeto de incremento.

15.1 Teste de unidade

Também conhecido como testes unitários. Tem por objetivo explorar a menor unidade do projeto, procurando provocar defeitos ocasionados por falhas de lógica e de prática em cada módulo, separadamente. O universo alvo desse tipo de teste são os métodos dos objetos ou mesmo pequenos trechos de algoritmo.

15.2 Teste de integração

Visa provocar falhas associadas às interfaces entre os módulos quando esses são integrados para arquitetar a estrutura do programa que foi estabelecida na fase de projeto.

15.3 Teste de sistema

Avalia o software em busca de falhas por meio da utilização do mesmo, como se fosse um cliente final. Dessa maneira, os testes são executados nos mesmos espaços, com as mesmas condições e com os mesmos dados de entrada que um usuário utilizaria no seu dia-a-dia de manipulação do programa. Verifica se o produto satisfaz seus parâmetros.

15.4 Teste de aceitação

São realizados geralmente por um restrito grupo de usuários finais do sistema. Esses simulam operações de rotina do aparelho de modo a verificar se seu comportamento está de acordo com o necessário.

O plano de teste deve incluir todos os elementos necessários para que os testes sejam executados corretamente. Em cada uma das etapas do processo de teste, é preciso executar os testes e analisar os resultados anunciados, e todos os registros da execução dos testes devem ser arquivados sob ferramentas de gestão dos testes, o que permite o acompanhamento do avanço dessa execução.

O usuário faz parte deste teste, pois é de sua responsabilidade. Mas tem também os testadores que trabalham no teste de aceitação para ajudar o cliente. Os testadores podem incluir um plano para o teste de aceitação no plano de teste, porém muitos autores acham que esse plano foge do alvo do trabalho dos testadores e que não poderia ser considerado para definição do tipo de teste efetuado pela equipe de teste.

Uma boa idéia seria se os clientes participassem de todo o processo de desenvolvimento e teste de software, o teste de aceitação seria executado muito mais simples e acelerado, levando em consideração que os problemas principais já terão sido analisados.

O teste de aceitação é projetado para determinar se o software que foi desenvolvido ou a parte que está liberada para teste estão aptos a serem utilizados pelo cliente em ambiente de produção.

Apto é uma expressão muito importante para garantir que o software foi desenvolvido de modo a atender aos requisitos de negócio inicialmente definidos. “Estar Apto” envolve quatro componentes: dados, pessoas, estrutura e regras.

Resumindo essas informações mostradas no artigo, não basta o software ser desenvolvido, testado e liberado para a produção. O bom funcionamento e o bom uso dependem também de outros fatores importantes, como foram mostrados aqui. Tem o ambiente onde o software irá operar. Bem, de qualquer forma, o ambiente de teste deve ser adequado e equivalente ao ambiente de produção.

16. Definição dos critérios de aceitação

Para definir os critérios de aceitação o usuário deve considerar alguns pontos importantes, ter pleno conhecimento das regras de negócio, familiarizar-se com a aplicação que está sendo implantada na produção, entender os riscos e benefícios da metodologia de desenvolvimento e de teste, e entender as consequências de adicionar uma nova funcionalidade para melhorar ou completar o sistema.Os requisitos que o software deve atender: requisitos de funcionalidades, requisitos de performance e requisitos de qualidades.

Antes dos critérios de aceitação do usuário, é preciso definir o nível de criticidade do software. Alguns fatores que definem o nível de criticidade só software são: importância do sistema para a organização, consequências das falhas, complexidade do projeto, riscos de tecnologia, complexidade do ambiente do usuário. Temos ainda o desenvolvimento do plano de aceitação e a execução do teste de aceitação, essas informações detalhadas vocês podem encontrar no livro BASTOS, Anderson et al. Base de conhecimento em teste de software. 3. Ed. São Paulo: Martins Fontes, selo Martins, 2012. 208 p.

17. A qualidade de software

A apreensão com o processo de software está relacionada à obrigação de abranger, avaliar, controlar, estudar, comunicar, melhorar, predizer e certificar o trabalho dos engenheiros de software. De tal modo, são necessárias estruturas que possibilitem a documentação, medição, definição, análise, avaliação, comparação e a adulteração de processos(ROCHA, 2001).

Conforme Herzum (2000) apropriadas práticas são exigidas em todo o processo de incremento e cobrem todos os aspectos de acréscimo em larga escala, desde alcança de requisitos, verificação, validação e garantia da qualidade e, desde gerenciamento de projeto até engenharia de processos incluindo avaliações e choques de otimização de processo. De um caráter amplo, uma boa prática pode ser definida como sendo todos os aspectos não

específicos em termos de tecnologia que são requeridos para preencher a lacuna entre a metodologia detalhada e o desenvolvimento legítimo.

A modelagem por si só não fornece uma vantagem econômica direta, mas os modelos são instrumentos essenciais e a modelagem é uma atividade essencial na melhoria da qualidade de produtos, de processos e reuso operacional. Modelos são funções que fornecem aos desenvolvedores, controle intelectual sobre a máquina de um sistema (BUTLER, 1997).

Conforme Kulpa (2003), um modelo é considerado uma forma das melhores práticas encontradas no estudo de outras organizações que funcionam bem e são muito bem-sucedidas. Um exemplar não contém os passos necessários ou a sequência de passos necessários para implementar um código de melhoria de processo.

As organizações de software encontram-se por si mesmas complicadas em umas amostras e exemplares de melhoria, um problema que aborrece muitos segmentos da indústria. Muitos desses modelos compartilham ampla igualdade de conteúdo. As instalações de software irão querer mostrar como seus vários processos de incremento se relacionam com processos contidos em vários exemplares. Então, os desenvolvedores necessitarão adaptar seus processos de desenvolvimento com elementos iguais, divididos entre vários modelos de processo, a fim de investigar como os seus processos satisfazem esses padrões(CURTIS, 2000).

O desafio para aparelhamentos de sistemas na próxima década será acrescentar atividades de melhoria entre as áreas de engenharia, de modo que possam unir ao invés de somente classificar seus processos de desenvolvimento de sistemas. À medida que as instalações de software começaram a melhorar seus processos, a engenharia de sistemas e outras organizações de incremento começaram a gerar esforços iguais para avanço.

> É comum as empresas de software buscarem a qualidade dos produtos por meio da ampliação de funcionalidades disponíveis a usuários, eliminação de ausências, cumprimento ou antecipação de limites, entre outras configurações. Porém é necessário ir um pouco além, adentrando a qualidade no próprio processo produtivo.

18. Conclusão

Esse artigo aprovou de modo rápido e simples o conhecimento em teste de software, foi abordado aqui a área de teste de software, onde abrangemos processos, ambiente de teste, elaboração do teste, execução dos testes, e aceitação dos testes.

O alvo do artigo foi alcançado com sucesso, mostrando aos clientes uma forma fácil de envolver o teste de software. Como vimos nesse artigo o teste de software é muito importante para a preparação de um software, detalhando conhecimentos de engenharia de software.

19. Referências

ABNT, Associação Brasileira de Normas Técnicas. **NBR ISO/IEC 9126-1 - Engenharia de software - Qualidade de produto - Parte 1: Modelo de qualidade.**Rio de Janeiro: ABNT, 2003.

AUTOIT Consulting Ltd. **AUTOIT: *Automation and Scripting Language.***England, 2012. Disponível em: < http://www.autoitscript.com/site/autoit/>. Acesso em: 11/06/2012.

BACH, James; KANER, Cem; PETTICHORD, Bret.*Lessons Learned in Software Testing: a Context-driven Approach.*New York: John Wiley& Sons, Inc., 2002.

BARBOSA, Filipe. **O teste de software no mercado de trabalho.** Paraíba: 2009.

BASTOS, Aderson et al.Base de conhecimento em teste de software.São Paulo: Martins, 2007.

BAUER, Carlos. Teste de software.São Paulo: 2006.

BURNSTEIN, Ilene. ***Practical Software Testing.*** New York: Springer-Verlag New York, Inc., 2010.

BUTLER, Luiz. **Teste de software.** São Paulo: 1997.

BRASIL. Ministério da Ciência e Tecnologia. **Tecnologia da Informação: Programa Brasileiro de Qualidade e Produtividade em Software.**Brasília, DF, 2006.

CARLINE, Cristiane. **Teste de software uma necessidade nas empresas.** Porto Alegre, 2012.

CATELANI, Marcantonio et al. ***Software automated testing: A solution to maximize the test plan coverage and to increase software reliability and quality in use.*** 7 f.Artigo-Universidade de Florença. Florença: Elsevier, 2010. Disponível em <http://www.sciencedirect.com/science/article/pii/S092054891000084X>. Acessoem: 25/06/2012.

CHRISSIS, Mary Beth; KONRAD, Mike; SHRUM, Sandy. CMMI: *Guidelines for process integration and product improvement.*Boston: Pearson Education, Inc., 2007.

COUTO, Ana Brasil. CMMI: Integração dos Modelos de Capacitação e Maturidade de Sistemas. Rio de Janeiro: Ciência Moderna, 2007.

6 CONCLUSÃO

O emprego de programas tem se mostrado cada vez mais essencial, tornando a qualidade de tais produtos em algo importante e exigindo que profissionais da área lutem por melhorias do processo de software.

Os estudos do presente trabalho permitiram observar as contribuições já existentes, como por exemplo, as metodologias, modelos e normatizações que visam o progresso, tanto do processo de casos de testes, quanto dos produtos em si. Com base em tais observações, compreendeu-se que a procura da qualidade passa pelo emprego da tarefa de testes que autentiquem a qualidade dos produtos.

Também foi constatado que entre as técnicas de testes, a funcional ou caixa-preta é o mais acentuado pela capacidade de validar os requisitos funcionais. Dentre as fases de testes, fazer jus a destaque a de regressão, justamente por assegurar a adesão do software aos requisitos funcionais originalmente planejados, impassível ao tempo decorrido desde o desenvolvimento original.

Em outras formas de estudo, o trabalho denotou o fato de que softwares que sofrem constantes manutenções são mais predispostos a apresentar falhas, tornando-os mais desprovidos de testes a cada autentificação. Tal informação constata que a derivação de testes funcionais regressivos é um requisito duradouro e aplicativo para tais softwares.

O trabalho relata cada caso de testes em varias versões de ambientes de internet.

REFERÊNCIAS

ABNT, Associação Brasileira de Normas Técnicas. **NBR ISO/IEC 9126-1 - Engenharia de software - Qualidade de produto - Parte 1: Modelo de qualidade.**Rio de Janeiro: ABNT, 2003.

AUTOIT Consulting Ltd. **AUTOIT: *Automation and Scripting Language.***England, 2012. Disponível em: < http://www.autoitscript.com/site/autoit/>. Acesso em: 11/06/2012.

BACH, James; KANER, Cem; PETTICHORD, Bret.*Lessons Learned in Software Testing: a Context-driven Approach.*New York: John Wiley& Sons, Inc., 2002.

BARBOSA, Filipe. **O teste de software no mercado de trabalho.** Paraíba: 2009.

BASTOS, Aderson et al.Base de conhecimento em teste de software.São Paulo: Martins, 2007.

BAUER, Carlos. Teste de software.São Paulo: 2006.

BURNSTEIN, Ilene. ***Practical Software Testing.*** New York: Springer-Verlag New York, Inc., 2010.

BUTLER, Luiz. **Teste de software.** São Paulo: 1997.

BRASIL. Ministério da Ciência e Tecnologia. **Tecnologia da Informação: Programa Brasileiro de Qualidade e Produtividade em Software.**Brasília, DF, 2006.

CARLINE, Cristiane. **Teste de software uma necessidade nas empresas.** Porto Alegre, 2012.

CATELANI, Marcantonio et al. ***Software automated testing: A solution to maximize the test plan coverage and to increase software reliability and quality in use.*** 7 f.Artigo-Universidade de Florença. Florença: Elsevier, 2010. Disponível em <http://www.sciencedirect.com/science/article/pii/S092054891000084X>. Acessoem: 25/06/2012.

CHRISSIS, Mary Beth; KONRAD, Mike; SHRUM, Sandy. CMMI: *Guidelines for process integration and product improvement.*Boston: Pearson Education, Inc., 2007.

COUTO, Ana Brasil. CMMI: Integração dos Modelos de Capacitação e Maturidade de Sistemas. Rio de Janeiro: Ciência Moderna, 2007.

CRESPO, AdalbertoNobiato et al. Uma Metodologia Para Testes de Software no Contexto de Melhoria do Processo.15 f. Artigo-Centro de

Pesquisas Renato Archer (CENPRA). Campinas, 2012.Disponível em: <bibliotecadigital.sbc.org.br/download.php?paper=254>. Acesso em: 27/06/2012.

CURTIS, Jair. Teste de software. São Paulo: 2000.

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

GHEZZI, Carlo; JAZAYERI, Mehdi; MANDRIOLI, Dino. *Fundamentals Of Software Engineering.* New Gersey: Prentice-Hall Inc. A Simon &Schster Company, 1991.

FEWSTER, Mark; GRAHAM, Dorothy.***Software Test Automation : Effective use of test automation tools.***GreatBritain: ACM Press Books, 1999.

GUERRA, Ana Cervigni; COLOMBO, Regina Maria Thienne. **Qualidade de Produto de Software.** Brasília: Ministério da Ciência e Tecnologia. Secretaria de Política de Informática, 2009.

HERZUM, Claudio. Teste de software. São Paulo: 2000.

IEEE, Institute of Electrical and Electronics Engineers, Inc. ***Draft* IEEE *Standard for software and system test documentation.*** New York: Institute of Electrical and Electronics Engineers, 2008.Disponívelem<http://ebookbrowse.com/gdoc.php?id=139543310&url=973c447b187bebe539e28b977e6a0f5b>Acessoem: 27/06/2012.

ISO/IEC, *International Organization for Standardization/ International Electrotechnical Commission.* **ISO/IEC TR 15504 *Software Process Assessment*.** USA: ISO/IEC,1998.

KANUNGO,Shivraj; GOYAL, Asha. **CMMI *Implementation: Embarking on High Maturity Practices.*** New Delhi: Tata McGraw-Hill Publishing Company Limited, 2004.

KONRAD, Lucas. **Teste de software.**São Paulo: 2003.

KULPA, Marcos. **Teste de software.**São Paulo: 2003.

LEDGARD, Henry F. ***Professional Software : Software Engineering Concepts.***USA: Addison-Wesley PublishingCompany, 1987.

LOPES, Fernando Bettiol.**Desenvolvimentode Uma Metodologia Aplicada ao Gerenciamento e Acompanhamento de Testes de Software Via *Web.*** 69 f. Monografia (Graduação em Ciência da Computação)-Universidade do Extremo Sul Catarinense-UNESC. Criciúma, 2009.

MOLINARI, Leonardo. Testes Funcionais de Software. Florianópolis: Visual Books, 2008.

MOLINARI, Leonardo. Inovação e Automação de Testes de Software.São Paulo: Erica, 2010.

MÜNCHEN, Ismael. **Autotec: Uma Ferramenta de Automação de Testes Para Interfaces Dinâmicas.** 50 f. Dissertação (Mestrado com Especialização em Computação Aplicada)-Universidade do Estado de Santa Catarina-UDESC. Joinville, 2010.

MYERS, Glenford J. ***The Art of Software Testing*.** New Jersey: John Wiley & Sons, Inc., 2004.
PETERS, James F.; PEDRYCZ, Witold. ***Software engineering : an engineering approach*.** USA: John Wiley & Sons, Inc., 2000.

PEZZÈ, Mauro; YOUNG, Michal. **Teste e Análise de Software.**Porto Alegre: Bookman, 2008.

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

RIOS, Emerson; MOREIRA, Trayahú. **Teste de Software. 2. ed. rev. ampl.** Brasil: Alta Books, 2006.

ROCHA, Ana Regina Cavalcanti; MALDONADO, José Carlos; WEBER, Kival Chaves. **Qualidade de Software: Teoria e Prática.** São Paulo: Prentice Hall, 2001.

SILVA, Josenilson. **O processo de teste de software.** Brasil: 2016.

SEI, Software Engineering Institute. **CMMI *for Development, Version* 1.3.**USA: 2018. Disponível em: <http://www.sei.cmu.edu/reports/10tr033.pdf>. Acesso em: 17/10/2018.

SELENIUM. **Selenium*WebApplicationTestingSystem*.**Disponível em: < http://seleniumhq.org/>. Acesso em: 13/09/2012.

SMARTBEAR. **TestComplete*AutomatedTesting***. USA: 2012. Disponível em: < http://smartbear.com/products/qa-tools/automated-testing-tools/automated-software-testing>. Acesso em: 24/10/2012.

SOMMERVILLE, Ian. Engenhariadesoftware**. São Paulo: Addison-Wesley, 2003.**

SOUSA, Henrique. **Teste de software.** São Paulo: 2003.

SOUZA, Karla. **A importância da atividade de teste no desenvolvimento de software.** Salvador: 2013.

TENÓRIO JUNIOR, Nelson Nunes. **Modelo–E10: Um modelo para estimativas de esforço em manutenção de software.** 134f. Tese (Doutorado

em Ciência daComputação)-Pontifícia Universidade Católica do Rio Grande do Sul. Porto Alegre, 2009.

TESTLINK. **TestLink.** Disponível em <http://www.teamst.org/>. Acesso em: 11/06/2012.

VICCARI, Leonardo Davi. **Automação de teste de software através de linhas de produtos e teste baseados em modelos.** 83 f. Dissertação (Mestrado em Ciência da Computação)-Pontifícia Universidade Católica do Rio Grande do Sul. Porto Alegre, 2009.